Evidence for a Satisfiability Threshold for Random 3CNF Formulas

Tracy Larrabee*
Yumi Tsuji†

UCSC-CRL-92-42
November 6, 1992

Baskin Center for
Computer Engineering & Information Sciences
University of California, Santa Cruz
Santa Cruz, CA 95064 USA

ABSTRACT

This paper presents empirical evidence of a satisfiability threshold in random 3CNF formulas. The paper also expands on and supports the conjecture of Mitchell, Selman, and Levesque [13] that *hard* randomly generated CNF formulas will be hard for any reasonable satisfiability algorithm. We report statistics for a much larger set of variables than have been previously reported; in particular, we show that for each clause to variable ratio less than 4.2, the percentage of satisfiable formulas increases, and for each clause to variable ratio greater than 4.2, the percentage of satisfiable formulas decreases as a function of number of variables. We found that several algorithms behaved qualitatively in the same fashion. We report on the relative performance of each algorithm.

Keywords: backtracking, logic, reasoning, satisfiability, threshold behavior

---

*Computer Engineering Board of Studies, University of California, Santa Cruz, CA 95064. Supported by NSF grant MIP-9158490.

†Computer and Information Sciences Board of Studies, University of California, Santa Cruz, CA 95064. Supported by NSF grant CCR-8958590.

1 Introduction

Boolean formula satisfiability is a fundamental problem in computer science. In addition to its role in computational complexity as the mother of all NP-complete problems, it is of tremendous practical interest. Applications of satisfiability include program and machine verification, and many aspects of VLSI design and test.

In comparing satisfiability algorithms, it is important to agree on what family of instance distributions to consider and how to parameterize the varying degree of difficulty across the distributions. Mitchell, Selman, and Levesque [13], among others, have pointed out that the ratio of the number of clauses to the number of variables could be used as a parameter to characterize difficulty levels for the random k-CNF problems. In particular, their empirical study on the random 3CNF formulas revealed an "easy-hard-easy" pattern for their Davis-Putnam style algorithm, where the hard peak occurs when the ratio is at about 4.3. They further conjectured that all reasonable satisfiability algorithms would achieve the same qualitative pattern. Their graphs showed that these same difficult ratios were those where the probability of satisfiability of randomly generated formulas is switching from 100% to 0%.

After visiting a few crucial definitions and mentioning previous important results, we will present our results, which support their conjecture using very different satisfiability algorithms. We will report on the satisfiability percentage of randomly generated formulas for varying ratios for formulas of up to 170 variables. We will then describe our algorithms. Finally, we will question the ability of metrics aimed at randomly generated formulas to accurately model the difficulty of satisfiability problems arising from practical problems.

2 Definitions

A *CNF formula* is a Boolean expression in a conjunctive normal form, i.e., each conjunct in the formula is a *clause*, which is itself the disjunction of literals. Each *literal* is either a *variable*, or the complement of a variable. For each variable x, define the complement $\overline{x}$ of x by $\overline{x} = 1 - x$.

A *truth assignment* is a mapping that assigns 0 or 1 to each variable in its domain. A truth assignment satisfies a specific clause if and only if at least one literal in the clause is mapped to 1; the assignment satisfies a formula if and only if it satisfies every clause in that formula. A formula is *satisfiable* if there exists at least one assignment that will satisfy the formula; if no such assignments exist, the formula is *unsatisfiable*. A clause is *trivial* if it contains a variable and the variable's complement, and a clause is *empty* if it contains no literals.

Given positive integers n, m, and k, a random k-CNF formula of n variables and m clauses consists of m clauses each containing k literals. Each clause is chosen independently at random according to the uniform distribution over all $\binom{n}{k}2^k$ non-trivial clauses of size k. For most of this paper we will be concerned with random formulas containing clauses of size 3: random 3CNF formulas.

It has long been conjectured that random formulas exhibit a threshold phenomenon, much like the threshold behavior of certain well-studied properties of random graphs (cf. Bollobas [2]). The precise formulation of this conjecture is that for every integer $k > 1$, there is a constant c_k such that a random k-CNF formula with the clause-to-variable ratio

less than c_k is satisfiable with probability approaching 1, and one with the ratio greater than c_k is unsatisfiable with probability approaching 1, as n increases.

Threshold properties for random structures are inherently interesting. For the case of random formulas, there is additional motivation for wanting to know the value of the threshold parameter c_k: it is commonly believed that the difficult instances of satisfiability should occur when formulas in n variables have roughly $c_k n$ clauses.

Goerdt as well as Chvatal and Reed have proved this conjecture for the case $k = 2$ by establishing that $c_2 = 1$ [11, 5]. Franco with Chao, Ho and Paull have published several papers over the last decade that have shown that c_3 must be at least 1 and c_3 must be no more than 5.2 [10, 4, 9, 5]. Broder, Frieze, and Upfal have recently tightened the lower bound on c_3 to 1.63 [3]. Several researchers have conjectured that c_3 is somewhere around 4.

We use mean CPU time rather than median recursive calls as measure of performance because use of means allows us to draw on the considerable body of knowledge about expected behavior, and use of CPU time on the same machine allows us to compare disparate algorithms that do not necessarily have a metric such as "number of recursive calls" in common.

3 Empirical Evidence

Using two separate programs implementing two separate satisfiability algorithms, we have obtained empirical results about c_3 that are more conclusive than previous empirical results implying the existence of a threshold. Other incomplete algorithms have been presented that can find an assignment for satisfiable formulas of larger size than reported here, but our algorithms are complete (that is, they detect unsatisfiability as well as satisfiability) and thus we can report on the percentage of satisfiable formulas for given n, m, and k.

There are two interesting outcomes of our experiments:

- The value of c_3 is very close to 4.2. As the number of variables is increased (up to $n = 170$), the threshold behavior at formulas with $4.2n$ clauses becomes more and more sharply pronounced. Mitchell, Selman, and Levesque have recently reported similar empirical results for much smaller values of n ($n = 50$) [13].
- All of the curves (showing the percentage of random formulas satisfiable as the number of clauses is increased from $3.6n$ to $5.2n$) go through the point $(4.2, 68\% \pm 3\%)$.

In Figure 3.1, each data point represents the percentage of 500 randomly generated formulas that were shown satisfiable by our second algorithm, which will be described in Section 5. The five curves give the results for 50, 80, 110, 140, and 170 variables. As the number of variables in the randomly generated formulas increase, the transition from 100% satisfiability to 0% satisfiability becomes more abrupt. The curves almost give the appearance of rotating around m/n equal to 4.2.

Figure 3.2 presents the same data, but includes the percent satisfiability for 500 randomly generated formulas for those m/n values surrounding 4.2. Results are given for n equal to 60, 70, 80, etc. up through 170. The error-bars represent the 95% confidence interval. This graph makes it easy to see that while the percentage of satisfiable formulas remains relatively constant at m/n equal to 4.2, the percentage falls for m/n greater than 4.2, and it climbs for m/n less than 4.2. We see this as experimental evidence that the value of c_3 is very close to 4.2.

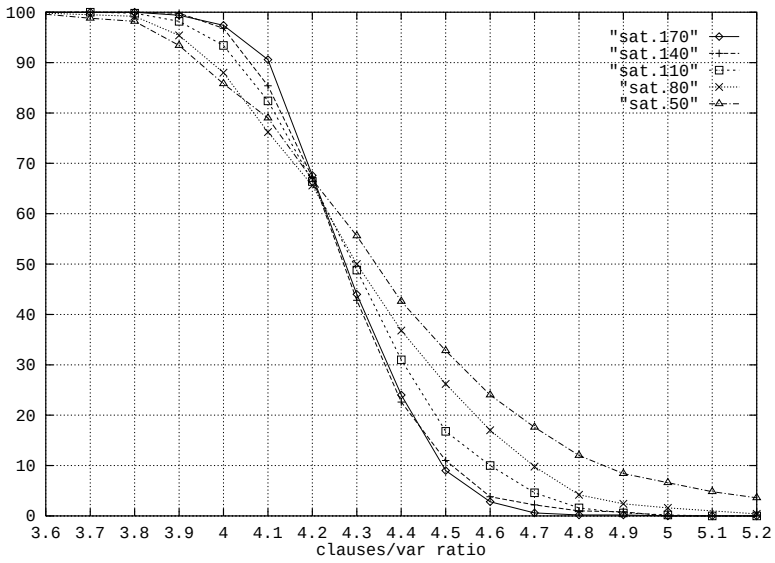

Figure 3.1: Percent satisfiable of 500 formulas for each ratio

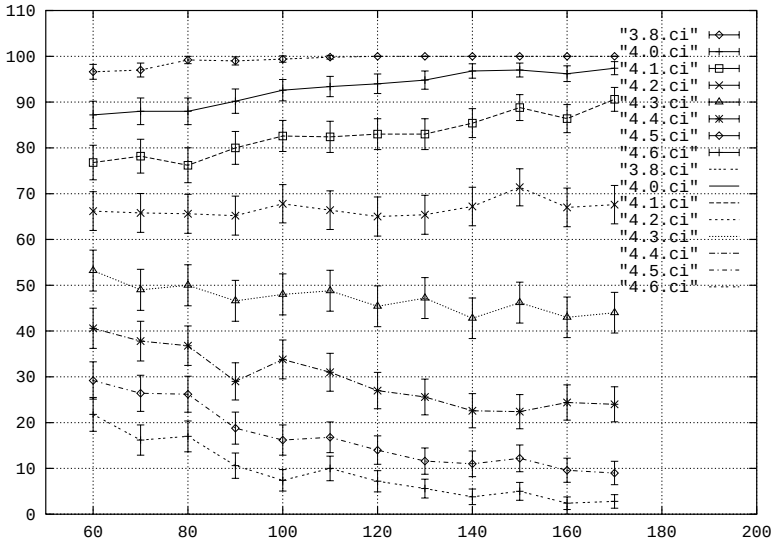

Figure 3.2: Percent satisfiable at each ratio as a function of number of variables

4 The First Algorithm

Our first algorithm will be referred to herein as *VG* as it was developed by A. Van Gelder [14]; it will be presented in three different forms, which will be called *VG-normal*, *VG-dp*, and *VG-mis*. They all include the basic Davis-Putnam rules (unit clause and pure literal) as originally described [7] and the implementation of the backtracking techniques described by Davis, Logemann, and Loveland [6]. The major difference among the three forms is in the strategy for selecting branch variables when the splitting rule is applied resulting in

effectively three different algorithms.

- **VG-normal**: The full algorithm described by Van Gelder [14]. In summary, VG-normal as applied to clausal formulas employs unit-clause and pure-literal rules. It also selects branch variables dynamically according to its own scoring scheme, which weights the variables.
- **VG-dp**: VG-normal degraded to simulate the Davis-Putnam like procedure used by Mitchell, Selman, and Levesque [13].
- **VG-mis**: VG-normal with a new branch-variable selection criterion; it transforms the input formula guided by a scheme using an algorithm which finds maximal independent set among the variables with certain properties.

As the only variation that has not be fully detailed before, we describe VG-mis below.

This strategy is essentially a divide-and-conquer approach based on the following fact: If a formula is a conjunction of subformulas that are independent of other subformulas, in the sense that no two subformulas share a common variable, then this formula is satisfiable if and only if all of its subformulas are independently satisfiable.

Let us view a formula F in a predicate calculus style, $(\exists \bar{x})F$ where $\bar{x}$ represents the set of free variables appearing in F. We wish to rewrite it as $(\exists \bar{x_0})((\exists \bar{x_1} F_1) \wedge (\exists \bar{x_2} F_2) \wedge \ldots (\exists \bar{x_k} F_k))$ where all the variables in $\bar{x}$ are partitioned into the subsets $\bar{x_0}$, $\bar{x_1}$,..., $\bar{x_k}$. This process, to be repeatedly applied to the resulting subformulas maybe visualized as that of pushing the existential quantifiers down as deep as possible.

We perform bottom-up processing to establish our variable partition. Let G be a graph where the nodes are the free variables in F and the edges represent the relation that two variables appear together in some clause. Let $M = \{x_1, x_2, \ldots x_k\}$ be a maximal independent set for G. Then for each i, $1 \leq i \leq k$, we combine the clauses in F containing x_i together to form a subformula F_i and will express it as $\exists \bar{x_i} F_i$ where $\bar{x_i}$ represents the set of all free variables in F_i that do not appear outside F_i. This process is repeated on the resultant formulas. The larger the set of variables, the finer our partition, but finding an independent set of maximum size is another NP-hard problem, so we aim only to find some maximal independent set in a polynomial time. Below is the VG-mis formula transformation algorithm, in which *MaxIndSet* represents a partition algorithm having the properties just discussed.

Input: Boolean formula
Output: Equivalent Boolean Formula
Method:

```
set F to the input formula.
simplify F via unit clause and pure literal rules.
set G to be the set of all the variables in the input formula.
while (G is not empty)
        set M to MaxIndSet(a graph of F restricted to the free variables).
        regroup subformulas in F according to M.
        push down the existential quantifiers for any local variables.
        set G to the set of free variables for the subformulas of F.
```

For example, if the above transformation is applied to the formula $F = f(a,b) \wedge g(b,c,x) \wedge h(a,b,c,y) \wedge p(d) \wedge q(a,d,z_1,z_2,z_3)$, we find $\{x, y, z_1\}$ to be a maximal independent set. After regrouping, the second application of the VG-mis algorithm will yield $\{b, d\}$, and finally

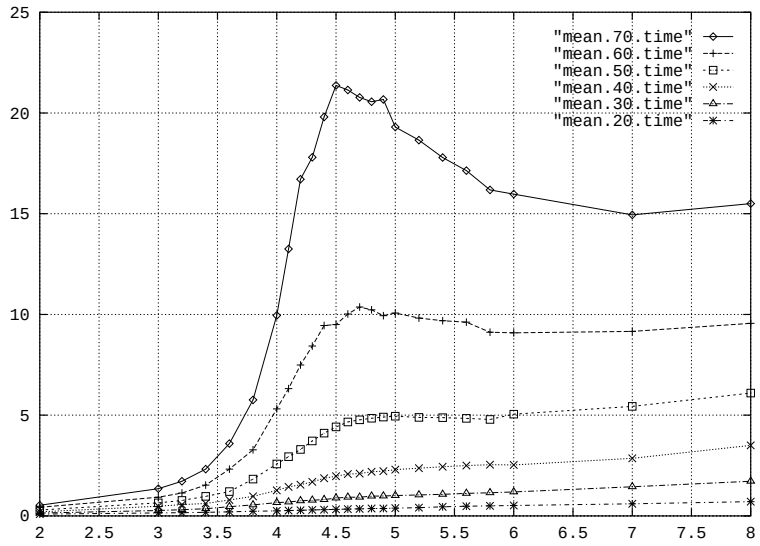

Figure 4.1: Mean CPU time for each ratio for the first algorithm

$\{a\}$. In this way, $(\exists a,b,c,d,x,y,z_1,z_2,z_3)F$ becomes $(\exists a)\{(\exists b,c)\{f(a,b) \wedge (\exists x)g(b,c,x) \wedge (\exists y)h(a,b,c,y)\} \wedge (\exists d)\{p(d) \wedge (\exists z_1,z_2,z_3)q(a,d,z_1,z_2,z_3)\}\}$.

There are two aspects of the VG-normal algorithm that need to be modified in order to take advantage of the the transformed formula and its associated information. First we must modify the choice of branch variable, and second we must modify our handling of the conjunction of independent subformulas.

At branch time, the free variables with the highest estimated likelihood of producing a conjunction of independent subformulas will be chosen. When there are no free variables, we are already in the position to independently process each subformula of the current formula.

Figure 4.1 shows the running time on a Sparcstation 1+ for VG-normal, a version of our first satisfiability algorithm. Each data point shows the average running time for 500 randomly generated formulas of 20, 30, 40, 50, 60 and 70 variables. The y-axis shows running time in seconds and the x-axis shows m/n. We have also made parallel runs using the other two versions, VG-mis and VG-dp, with the similar qualitative result; VG-normal performed, on the average, two times faster than VG-mis, while VG-mis performed, on the average, ten times faster than VG-dp.

5 The Second Algorithm

Our second satisfiability algorithm is part of Nemesis, a successful Automatic Test Pattern Generation system [12]. Nemesis generates a Boolean formula that includes all the constraints that must be satisfied in order to generate a test pattern for a particular fault in a circuit, and then it satisfies the resultant formula. The design of the satisfier was originally motivated by the preponderance of width-2 clauses in the formulas generated in order to detect circuit faults: at least 2/3 of the clauses in every formula had only two literals, and more commonly 80–90% of the clauses were width-2 clauses.

Accordingly, we decided to use the 2SAT satisfier proposed by Aspvall, Tarjan, and Plass [1] in order to generate a satisfying assignment for a 3CNF expression. To summarize the method: if necessary we assign values to a few variables until we have a portion of the formula that is purely 2CNF. We then iterate through the 2SAT satisfying assignments until we either find one that is consistent with the non-2CNF portion of the formula or until we prove that there is no such assignment. When we completed the satisfier we discovered that it worked well on 3CNF formulas that were not primarily composed of width-2 clauses.

The main loop of our satisfiability algorithm follows:

```
Satisfy (formula)
    loop
        if not backtracking
            select one of the narrowest clauses to satisfy
            if there were no clauses, you are done: formula satisfied
            else if you selected a binary clause
                iterate through 2SAT bindings
                until no falsified clauses or no more 2sat bindings
                if exhausted all 2sat bindings, start backtracking
            otherwise
                select a clause to satisfy
                satisfy the first unbound literal and falsify all others
                if there are any unsatisfied clauses, start backtracking
        otherwise, if you are backtracking
            if your stack is empty, you are done: formula unsatisfiable
            otherwise
                look at the most recently satisfied clause
                unbind the variable bound in order to satisfy that clause
                if there are any other mutable literals in the clause
                    unbind the mutable literal from its old value
                    bind the mutable literal in order to satisfy the clause
                    if there are no unsatisfied clauses, stop backtracking
                otherwise
                    unbind all the variables bound because of this clause
                    consider the next most recently bound clause
    endloop
end Satisfy.
```

Figure 5.1 shows the running time on a Sparcstation 1+ for the Nemesis satisfiability algorithm using the formulas of 170, 160, 150 and 140 variables. The graph shows the similarity to the result obtained using the VG-algorithm for smaller number of variables in the location of the peak ratio point. There is a pronounced difference between the two algorithms in the slope of the lines to the right of the peak. This is explained by the nature of the VG algorithm, which works more efficiently for the satisfiable formulas than for the unsatisfiable formulas, whereas the Nemesis satisfiability algorithm doesn't have such skew.

The time varies from less than 10 seconds at either end of the graph to almost 45 minutes at the "hard-spot", which is centered around 4.3. This result is consistent with that reported by Mitchell, *et al.* [13], which reported number of recursive calls instead of CPU time. The graphs for each size n that we have investigated has the same basic shape, though the extreme in cpu time for formulas with a given number of variables has occurred

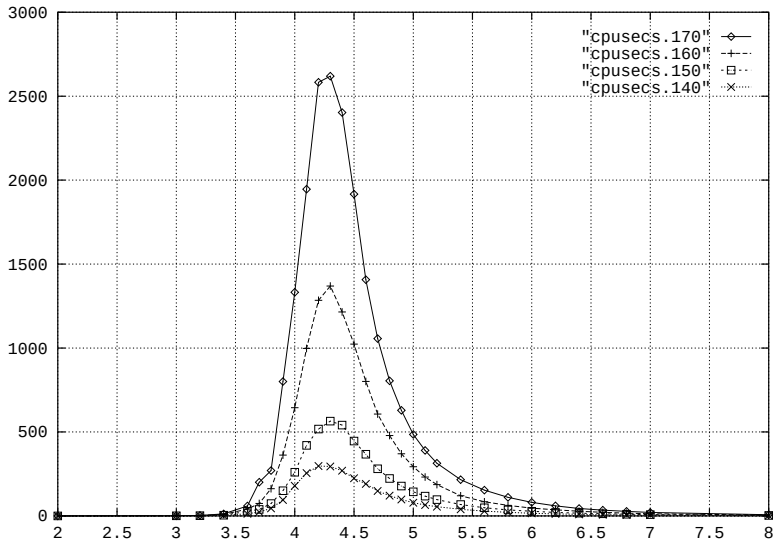

Figure 5.1: Mean CPU time for each ratio for the second algorithm

from 4.2 to 4.5. In general, we also observed that the Nemesis algorithm, the largest average time for a given n is twice the time of $n - 10$.

For further details, see the description of the Nemesis ATPG system [12].

6 Random Versus Non-random Formulas

We have also applied these algorithms to a certain set of structured non-clausal formulas derived from cycle-covering problems [1] provided us by Dan Pehoushek of Stanford University. VG-mis has outperformed VG-normal in the range of 1.5 times for a formula of 40 variables to 40 times for a formula of 84 variables. The Nemesis satisfiability algorithm's performance on a CNF version of the some problem was very similar to the VG-mis algorithm on the non-clausal problem.

It is striking to us that, although the maximal independent sets strategy is more efficient on this non-random formula, the VG-normal algorithm outperformed VG-mis on the average for the randomly generated 3CNF formulas (the Nemesis satisfiability algorithm is faster on average than either of the VG algorithms when processing random formulas).

Having observed this phenomenon, we tried all of our algorithms on four very difficult, very large (2500 variables and 8000 clauses) CNF formulas representing the legal test sets for detecting bridge faults in CMOS ICs [8]. The four formulas each caused the test pattern generation system to "time out" when attempting to generate a test (or prove that no test existed). When the four formulas were submitted to the Nemesis satisfier with no time-out, two of them finished in about a minute of CPU time, but the other two were not successfully completed after a weekend's worth of processing time on a Decstation 5000/240. After about a day of Decstation CPU time, the Davis-Putnam and VG implementations were not able to terminate when working on these four formulas.

[1] A cycle covering of undirected graph G is a subgraph that contains all nodes of G such that each node has degree 2. The question is NP-complete in general.

These observations raises question as to what distribution space is really meaningful in terms of practical use. Many formulas of interest do not come in 3CNF form. The four formulas mentioned were about evenly divided between 2-clauses and 3-clauses, with a few unit clauses and about 5% longer clauses. It is not clear to which distribution space such formulas would belong. As future work, we would like to investigate different metrics of Boolean formula difficulty.

7 Conclusions

Our empirical results strongly suggest that there is a satisfiability threshold for 3CNF formulas and that the threshold is quite near to 4.2. We found further evidence to support the conjecture of Mitchell, Selman and Levesque that *hard* randomly generated CNF formulas will be hard for any reasonable satisfiability algorithm, and we report statistics for a larger set of variables than have been previously reported. We have reported on two new algorithms and compared their performance via CPU time to each other and to the Davis-Putnam algorithm. We have raised the question of the suitability of random 3CNF formulas in modeling algorithm efficiency on real-world satisfiability problems.

8 Acknowledgements

We thank Bart Selman for providing the challenge to test his conjecture that the "hard" distribution is indeed "hard" for any reasonable satisfiability algorithm. We also appreciate the personal interaction with and programs received from Dr. Selman; we used the same random formula generator as was used by Mitchell, Selman, and Levesque [13], and we were also able to do a consistency check between our results and theirs by running their algorithm directly on our machines.

We are indebted to Allen Van Gelder for initially suggesting the "maximal independent set" strategy as a new branching strategy to his original algorithm. We are indebted to Allen Van Gelder, Phokion Kolaitis, and Anna Karlin for their interest, encouragement, and critical analysis of our paper, and to all the participants of the satisfiability seminar held at UCSC in the Spring of 1992 for their contribution of ideas and seminal discussions. We thank J. Alicia Grice for her assistance in running our experiments.

Digital Equipment Corporation's gift of a Decstation 5000/240 as part of its External Research program has made our most recent investigations more efficient, and we wish to thank them for their support.

References

[1] B. Aspvall, M. Plass, and R. Tarjan. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Information Processing Letters*, 8:121–123, 1979.

[2] B. Bollobas. *Random Graphs*. Academic Press, 1985.

[3] A. Broder, A. Frieze, and E. Upfal. On the satisfiability and maximum satisfiability of random 3-CNF formulae. In *Proceedings of Symposium on Discrete Algorithms*, 1993.

[4] M. Chao and J Franco. Probabilistic analysis of two heuristics for the 3-satisfiability problem. *SIAM Journal of Computation*, 15:1106–1118, 1986.

[5] V. Chvatal and B. Reeds. Nick gets some (the odds are on his side). In *Proceedings of the IEEE Symposium on the Foundations of Computer Science*, volume 33, 1992.

[6] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Communications of the Association for Computing Machinery*, 5:394–397, 1962.

[7] M. Davis and H. Putman. A computing procedure for quantification theory. *Journal of the Association for Computing Machinery*, 7:201–215, 1960.

[8] F. J. Ferguson and T. Larrabee. Test pattern generation for realistic bridge faults in CMOS ICs. In *Proceedings of International Test Conference*, pages 492–499. IEEE, 1991.

[9] J Franco and Y. Ho. Probabilistic performance of a heuristic for the satisfiability problem. *Discrete Applied Mathematics*, 22:35–51, 1989.

[10] J. Franco and M. Paull. Probabilistic analysis of the Davis–Putnam procedure for solving the satisfiability problem. *Discrete Applied Mathematics*, 5:77–87, 1983.

[11] A. Goerdt. A threshold for unsatisfiability. In *Proceedings of the International Symposium on Mathematical Foundations of Computer Science*, August 1992.

[12] T. Larrabee. Test pattern generation using Boolean satisfiability. *IEEE Transactions on Computer-Aided Design*, pages 6–22, January 1992.

[13] D. Mitchell, B. Selman, and H. Levesque. Hard and easy distributions of SAT problems. In *Proceedings of AAAI*, 1992.

[14] A. Van Gelder. A satisfiability tester for non-clausal propositional calculus. *Information and Computation*, 79, October 1988.