Hiding Satisfying Assignments: Two are Better than One

Dimitris Achlioptas
Microsoft Research
Redmond, Washington
optas@microsoft.com

Haixia Jia and **Cristopher Moore***
Computer Science Department
University of New Mexico
{hjia,moore}@cs.unm.edu

Abstract

The evaluation of incomplete satisfiability solvers depends critically on the availability of hard satisfiable instances. A plausible source of such instances are k-CNF formulas whose clauses are chosen uniformly at random among all clauses satisfying some randomly chosen truth assignment A. Unfortunately, instances generated in this manner are relatively easy and can be solved efficiently by practical heuristics. Roughly speaking, as the number of clauses is increased, A acts as a stronger and stronger attractor. Motivated by recent results on the geometry of the space of solutions for random k-SAT and NAE-k-SAT instances, we propose a very simple twist on this model that greatly increases the hardness of the resulting formulas. Namely, in addition to forbidding the clauses violated by the hidden assignment A, we also forbid the clauses violated by its complement, so that both A and $\overline{A}$ are satisfying. It appears that under this "symmetrization" the effects of the two attractors largely cancel out, making it much harder for an algorithm to "feel" (and find) either one. We give theoretical and experimental evidence supporting this assertion.

Introduction

Recent years have witnessed the rapid development and application of search methods for constraint satisfaction and Boolean satisfiability. An important factor in the success of these algorithms is the availability of good sets of benchmark problems to evaluate and fine-tune them. There are two main sources of such problems: the real world, and random instance generators. Real-world problems are arguably the best benchmark, but unfortunately are in short supply. Moreover, using real-world problems carries the risk of tuning algorithms toward the specific application domains for which good benchmarks are available. In that sense, random instance generators are a good additional source, with the advantage of controllable characteristics, such as size and expected hardness. Hard random instances have led to the development of new stochastic search methods such as `WalkSAT` (Selman, Kautz, & Cohen 1996) and the breakout procedure (Morris 1993), and have been used in detailed comparisons of local search methods for graph coloring and related graph problems (Johnson *et al.* 1989).

*Supported by NSF grant PHY-0200909.

A key limitation of current problem generators concerns their use in evaluating incomplete local search methods. When a local search algorithm does not find a solution, it can be difficult to determine whether this is because the instance is unsatisfiable, or simply because the algorithm failed in finding a satisfying assignment. The standard way of dealing with this problem is to use a complete search method to filter out the unsatisfiable cases. However, this limits the size and difficulty of problem instances that can be considered. Ideally, one would use problem generators that generate satisfiable instances only. One recent source of such problems is the quasigroup completion problem (Shaw, Stergiou, & Walsh 1998; Kautz *et al.* 2001; Achlioptas *et al.* 2000). However, a generator for random hard satisfiability instances has remained elusive.

A very natural candidate for generating random hard satisfiable 3-SAT formulas is the following. Generate a random truth assignment A, and then generate a formula with n variables and rn random clauses, rejecting any clause that is violated by A. In particular, if we work close to the transition region $r \approx 4.25$ where the hardest 3-SAT problems seem to be (Cheeseman, Kanefsky, & Taylor 1991; Hogg, Huberman, & Williams 1996; Mitchell, Selman, & Levesque 1992), we might hope that this would generate hard satisfiable instances. Unfortunately, though, this generator is highly biased towards formulas with many assignments clustered around A. When given to local search methods such as `WalkSAT`, these formulas turn out to be much easier than formulas of comparable size obtained by filtering satisfiable instances from a 3-SAT generator. More sophisticated versions of this "hidden assignment" scheme (Asahiro, Iwama, & Miyano 1996; Van Gelder 1993) improve matters somewhat but still lead to biased samples.

In this paper we introduce a new generator of random satisfiable problems. The idea is simple: we pick a random 3-SAT formula that has a "hidden" **complementary pair** of satisfying assignments, A and $\overline{A}$, by rejecting clauses that are violated by either A or $\overline{A}$. Our motivation comes from recent work (Achlioptas & Moore 2002b) which showed that moving from random k-SAT to random NAE-k-SAT (in which every clause in the k-CNF must have a true *and* a false literal) tremendously reduces the correlation between solutions. That is, whereas in random k-SAT, satisfying assignments tend to form clumps, in random NAE-k-SAT

the solutions appear to be scattered throughout the hypercube in a rather uniform "mist", even for densities extremely close to the threshold. An intuitive explanation of this phenomenon is that since the complement of every NAE-assignment is also an NAE-assignment, the pull of pairs of solutions largely cancel out. In this paper we impose a similar symmetry on the hidden assignments A and $\overline{A}$, so that *their* attractions cancel out, making it hard for a wide variety of algorithms to "feel" either one.

A key feature of our generator is that it corresponds to an extremely simple probabilistic process, in sharp contrast with, say, 3-SAT generators based on cryptographic ideas (Massacci 1999). As a result, it is *readily amenable to mathematical analysis.* Here we take some significant steps in this direction, indeed using tools similar to those employed in the study of random k-SAT formulas.

To ascertain the hardness of our formulas we performed extensive computational experiments comparing them to "1-hidden" and "0-hidden" formulas. That is, we compared our formulas to random 3-SAT formulas with one hidden assignment and to standard random 3-SAT formulas (with no hidden assignment). We examined four leading algorithms: two complete solvers, `zChaff` and `Satz`, and two incomplete ones, `WalkSAT` and the recently introduced `SP`.

For all these algorithms, we find that our formulas are much harder than 1-hidden formulas and, more importantly, *about as hard 0-hidden ones.*

The space of solutions

In this section we compare 1-hidden and 2-hidden formulas with respect to the expected number of solutions at a given distance from the hidden assignment(s).

Let X be the number of satisfying truth assignments in a random formula with n variables and $m = rn$ clauses chosen uniformly and independently among all k-clauses with *at least one positive literal*, i.e., 1-hidden formulas where we hide the all–1s truth assignment. Then, using linearity of expectation, clause independence, the selection of the literals in each clause with replacement, and Stirling's approximation for the factorial, we get (1) below (where $\sim$ suppresses terms polynomial in n):

$$\begin{aligned}
\mathbf{E}[X] &= \sum_{\sigma\in\{0,1\}^n} \Pr[\sigma \text{ is satisfying}] \\
&= \sum_{z=0}^{n} \binom{n}{z} \Pr\left[\begin{array}{c}\text{a truth assignment with } z \text{ ones}\\ \text{satisfies a random clause}\end{array}\right]^m \\
&= \sum_{z=0}^{n} \binom{n}{z} \left(1 - \sum_{j=1}^{k} \binom{k}{j} \frac{(1-z/n)^j (z/n)^{k-j}}{2^k-1}\right)^m \\
&= \sum_{z=0}^{n} \binom{n}{z} \left(1 - \frac{1-(z/n)^k}{2^k-1}\right)^m \\
&\sim \max_{\alpha\in[0,1]} \left[\frac{1}{\alpha^\alpha(1-\alpha)^{1-\alpha}} \left(1 - \frac{1-\alpha^k}{2^k-1}\right)^r\right]^n \\
&\equiv \max_{\alpha\in[0,1]} \left[f_{k,r}(\alpha)\right]^n \qquad . \qquad (1)
\end{aligned}$$

From (1) we see that $\mathbf{E}[X]$ is dominated by the contribution of truth assignments with γn ones, where γ is the maximizer of $f_{k,r}$ in $[0,1]$. Observe now that $f_{k,r}$ is a strictly increasing function of α, maximized at some $\gamma > 1/2$ for *every* $r > 0$. Thus, always, most solutions have a majority of 1s, i.e., the hidden assignment is "felt". Moreover, as r is increased $\gamma \to 1$ (see Figure 1 below).

To study 2-hidden formulas now, let X be the number of satisfying truth assignments in a random formula with n variables and $m = rn$ clauses chosen uniformly among all k-clauses with at least one positive *and at least one negative literal*; that is, 2-hidden formulas where we hide the all–1s truth assignment *and* its complement. Proceeding as above, but with the second line replaced by

$$\sum_{z=0}^{n} \binom{n}{z} \left(1 - \sum_{j=1}^{k-1} \binom{k}{j} \frac{(1-z/n)^j (z/n)^{k-j}}{2^k-2}\right)^m$$

and carrying through the ensuing changes we find that now $\mathbf{E}[X] \sim \max_{\alpha\in[0,1]}[g_{k,r}(\alpha)]^n$ where

$$g_{k,r}(\alpha) = \frac{1}{\alpha^\alpha(1-\alpha)^{1-\alpha}} \left(1 - \frac{1-\alpha^k-(1-\alpha)^k}{2^k-2}\right)^r \quad .$$

Observe that the function $g_{k,r}$ is *symmetric* in $[0,1]$, unlike $f_{k,r}$. Indeed, its global maximum occurs at $\alpha = 1/2$ for all r up to extremely close to the random k-SAT threshold. In other words, for such r the hidden assignments do not cause satisfying assignments to cluster around them. More precisely, there exists a sequence $\epsilon_k \to 0$ such that $g_{k,r}(1/2) > g_{k,r}(\alpha)$ for all $\alpha \neq 1/2$, as long as

$$r \leq 2^k \ln 2 - \frac{\ln 2}{2} - 1 - \epsilon_k \quad . \qquad (2)$$

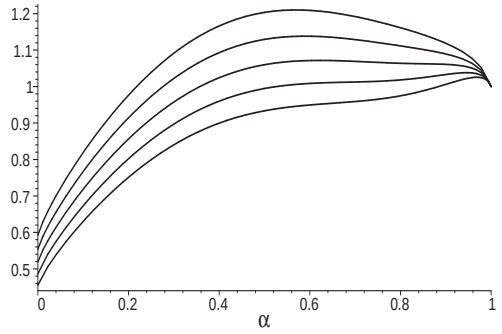
Contrast this with the fact (implicit in (Kirousis *et al.* 1998)) that there exists a sequence $\delta_k \to 0$ such that for

$$r \geq 2^k \ln 2 - \frac{\ln 2}{2} - \frac{1}{2} - \delta_k \quad , \qquad (3)$$

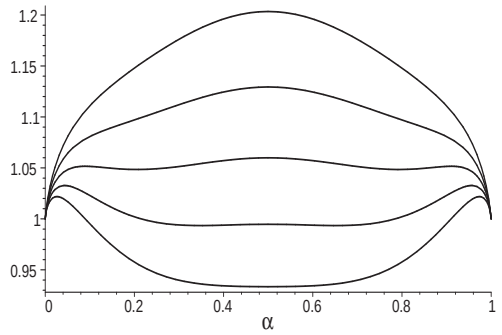
a random k-SAT formula with n variables and $m = rn$ clauses is unsatisfiable with probability $1 - o(1)$. Equations (2) and (3) differ by just $1/2$ as $k \to \infty$; for instance, for $k = 10$ they give 708.40 and 708.94 respectively.

Below we plot[1] $f_{k,r}$ and $g_{k,r}$ for $k = 5$ and $r = 16, 18, 20, 22, 24$ (from top to bottom). We see that in the case of 1-hidden formulas, i.e., $f_{k,r}$, the maximum always occurs to the right of $\alpha = 1/2$. Moreover, observe that for $r = 22, 24$, i.e., after we cross the 5-SAT threshold (which occurs at $r \approx 21$) we have a dramatic shift in the location of the maximum and, thus, in the extent of the bias: as one would expect, the only remaining satisfying assignments above the threshold are those extremely close to the hidden assignment.

[1]The reason we plot $k = 5$ is that this is the smallest value of k for which the evolution of $g_{k,r}$ exhibits its full complexity (the evolution is qualitatively the same for all $k \geq 5$). For $k = 3, 4$ there are never more than *two* local extrema, as the emergence of maxima away from $\alpha = 1/2$ coincides with $\alpha = 1/2$ becoming a local minimum.



1-hidden formulas



2-hidden formulas

Figure 1: The nth root of the expected number of solutions f and g as functions of the fraction $\alpha = z/n$ of 1s. Here $k = 5$ and $r = 16, 18, 20, 22, 24$ from top to bottom.

In the case of 2-hidden formulas, on the other hand, we see that for $r = 16, 18, 20$ the global maximum occurs at $\alpha = 1/2$. For $r = 20$, we also have two local maxima, near $\alpha = 0, 1$, but since $g_{k,r}$ is raised to the nth power, these are exponentially suppressed. Naturally, for r above the threshold, i.e., $r = 22, 24$, these local maxima become global, signifying that indeed the only remaining truth assignments are those extremely close to one of the two hidden ones.

Unit Clause Resolution and DPLL algorithms

Consider the following linear-time heuristic, called Unit Clause (UC): if there are any unit clauses, satisfy them; else (a "free step") pick a random literal and satisfy it. One can think of UC as the first branch of the simplest possible DPLL algorithm S: set variables in a random order, each time choosing randomly which branch to take first. Clearly, UC succeeding on a formula F is equivalent to S succeeding on F on its very first try, i.e., without any backtracking. In (Chao & Franco 1986), it was shown that UC succeeds with constant probability on random 3-SAT formulas for $r < 8/3$, and fails w.h.p. for $r > 8/3$.

To analyze UC on random 1-hidden and 2-hidden formulas we actually analyze UC on arbitrary initial distributions of 3-clauses, i.e., where for each $0 \le j \le 3$ we specify the initial number of 3-clauses with j positive literals and $3-j$ negative ones. We use the method of differential equations; see (Achlioptas 2001) for a review. To simplify notation, we assume that our 1-hidden formulas forbid clauses where all literals are negative, while our 2-hidden formulas forbid all-negative and all-positive clauses. A *round* of UC is a free step and the ensuing chain of unit-clause propagations. For $0 \le i \le 3$ and $0 \le j \le i$, let $s_{i,j}$ be the density of clauses of length i with j positive literals and $i-j$ negative ones, and let x be the fraction of variables set so far. Since $s_{1,0} = s_{1,1} = 0$ at the beginning of each round, our "state space" will consist of the variables $s_{i,j}$ for $i \ge 2$.

Let m_T and m_F be the expected number of variables set True and False in a round; we will calculate these below. Then rescaling the expected effect of a round gives the following system of differential equations,

$$\begin{aligned} \frac{\mathrm{d}s_{3,j}}{\mathrm{d}x} &= -\frac{3s_{3,j}}{1-x} \\ \frac{\mathrm{d}s_{2,j}}{\mathrm{d}x} &= -\frac{2s_{2,j}}{1-x} + \frac{m_F(j+1)s_{3,j+1} + m_T(3-j)s_{3,j}}{(m_T+m_F)(1-x)} \end{aligned} \tag{4}$$

To see this, note that a variable appears positively in a clause of type i, j with probability $j/(n-X)$, and negatively with probability $(i-j)/(n-X)$. The terms above correspond to clauses being "hit" by the variables set, and the "flow" of 3-clauses to 2-clauses. Then Wormald's theorem (Wormald 1995) ensures that w.h.p. for all x, the $s_{i,j}$ are within $o(1)$ of the solutions to this system.

To calculate m_T and m_F, we model the chain of unit clauses created in a round as a two-type branching process, which we analyze as in (Achlioptas & Moore 2002a). Since the free step gives the chosen variable a random value, the initial expected population of unit clauses can be represented by a vector $p_0 = (1/2, 1/2)^T$. Moreover, at time x, a unit clause procreates according to the matrix

$$M = \frac{1}{1-x} \begin{pmatrix} s_{2,1} & 2s_{2,0} \\ 2s_{2,2} & s_{2,1} \end{pmatrix} .$$

For instance, satisfying a negative unit clause creates, in expectation, $M_{1,1} = s_{2,1}/(1-x)$ negative unit clauses and $M_{2,1} = 2s_{2,2}/(1-x)$ positive unit clauses.

Let λ_1 be the largest eigenvalue of M. If $\lambda_1 < 1$ for all x, then the branching process is subcritical, UC succeeds with constant probability, and $(m_F, m_T)^T = (I-M)^{-1} \cdot p_0$ where I is the identity matrix. On the other hand, if λ_1 ever exceeds 1, then the branching process becomes supercritical, with high probability the unit clauses proliferate and the algorithm fails. Note that

$$\lambda_1 = \frac{s_{2,1} + 2\sqrt{s_{2,0}\, s_{2,2}}}{1-x} . \tag{5}$$

Now, suppose our initial distribution of 3-clauses is symmetric, i.e., $s_{3,0}(0) = s_{3,3}(0)$ and $s_{3,1}(0) = s_{3,2}(0)$. It is easy to see from (4) that in that case, both the 3-clauses and the 2-clauses are symmetric at all times, i.e., $s_{i,j} = s_{i,i-j}$ and $m_F = m_T$. In that case $s_{2,1} + 2\sqrt{s_{2,0}s_{2,2}} = s_2$, so the criterion for subcriticality is $\lambda_1 = \frac{s_2}{1-x} < 1$. Moreover, since the system (4) is now symmetric with respect to j, summing over j gives precisely the differential equations $\frac{\mathrm{d}s_3}{\mathrm{d}x} = -\frac{3s_3}{1-x}$ and $\frac{\mathrm{d}s_2}{\mathrm{d}x} = -\frac{2s_2}{1-x} + \frac{3s_3}{2(1-x)}$ for UC on random (0-hidden) instances of 3-SAT.

Since 2-hidden formulas correspond to symmetric initial conditions, we have thus shown that UC succeeds on them with constant probability if and only if $r < 8/3$, i.e., that UC *fails on 2-hidden formulas at exactly the same density for which it fails on 0-hidden formulas*. (In contrast, integrating (4) with the initial conditions corresponding to 1-hidden formulas shows that UC succeeds for them at a slightly higher density, up to $r < 2.679$.)

Of course, UC can easily be improved by making the free step choice more intelligent. We believe that a lot of the progress that has been made in analyzing the performance of algorithms on instances of random 3-SAT, i.e., 0-hidden formulas, can be "pushed through" to 2-hidden formulas. Specifically, nearly all DPLL-type algorithms analyzed so far have the property that given as input a symmetric initial distribution of 3-clauses, e.g. random 3-SAT, their residual formulas consist of symmetric mixes of 2- and 3-clauses. As a result, methods as above can be used to show that such algorithms act on 2-hidden formulas exactly as they do on 0-hidden ones, failing w.h.p. at the same density.

This last point acquires additional significance in the light of recent results on the onset of backtracking in DPLL algorithms for random 3-SAT formulas. Specifically, for the algorithm S defined above (the backtracking extension of UC), it was proved (Achlioptas, Beame, & Molloy 2001) that exponential behavior already occurs for $r > 3.81$, well below the conjectured satisfiability threshold $r \approx 4.2$. In fact, non-rigorous but mathematically sophisticated calculations of (Cocco & Monasson 2001a; 2001b) suggest that exponential behavior begins right at the density where UC begins to fail with high probability, i.e., $r = 8/3$. (Indeed, that follows rigorously from the conjecture that the "tricritical" point in random (2+p)-SAT is $p_c = 2/5$.) As a single branch of S acts on 2-hidden formulas in exactly the same way as on 0-hidden formulas, a very exciting possibility is to *prove* that S takes exponential time on 2-hidden formulas above a certain density by extending the existing proof for random 3-SAT. Indeed, one can hope to do this for all DPLL algorithms with the symmetry discussed above.

Experimental results

In this section we report experimental results on our 2-hidden formulas, and compare them to 1-hidden and 0-hidden ones. We use two leading complete solvers, zChaff and Satz, and two leading incomplete solvers, WalkSAT and the new Survey Propagation algorithm SP. In an attempt to avoid spurious features often present in "too-small" random instances, we restricted our attention to experiments where $n \geq 1000$. This meant that zChaff and Satz could only be examined at densities significantly above the satisfiability threshold, i.e., 4.2.., as neither algorithm could practically solve either 0-hidden *or* 2-hidden formulas with $n \sim 1000$ variables and such density. For WalkSAT and SP on the other hand, we can easily run experiments in the hardest range (around the satisfiability threshold) for $n \sim 10^4$.

zChaff and Satz

In order to do experiments with $n \geq 1000$ with zChaff and Satz, we focused on the regime where r is relatively large, $20 < r < 60$. This is because for r near the satisfiability threshold, 0-hidden and 2-hidden random formulas with $n \sim 1000$ variables seem completely out of the reach of either algorithm. At the same time, in the overconstrained regime we considered, the formulas are still challenging, but the presence of many forced steps allows both solvers to completely explore the space fairly quickly.

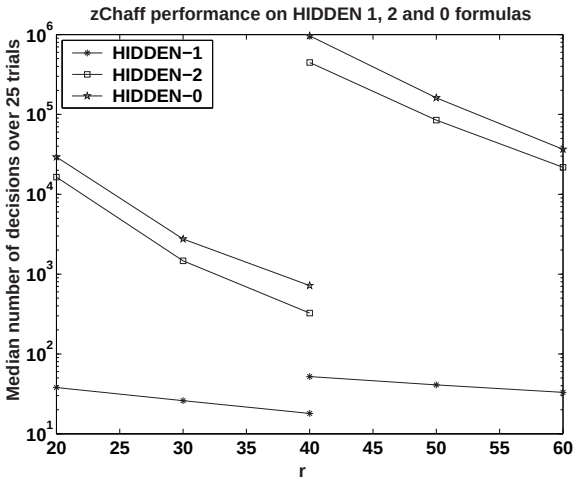


Figure 2: The number of branches of zChaffon 0-, 1-, and 2-hidden 3-SAT formulas, with $n = 1000$ for $20 \leq r \leq 40$ and $n = 3000$ for $40 \leq r \leq 60$. Each point is the median of 25 trials. The 2-hidden formulas are almost as hard as 0-hidden ones; the 1-hidden formulas are much easier.

We obtained zChaff from Princeton (Zhang). Figure 2 shows its performance on random 0-hidden, 1-hidden and 2-hidden formulas. We see that for all three types of problems, the number of branches decreases rapidly as r increases, consistent with earlier findings for complete solvers on random 3-SAT formulas.

Figure 2 shows that zChaff finds 2-hidden formulas almost as difficult as 0-hidden ones, which for this range of r are unsatisfiable with overwhelming probability. On the other hand, the 1-hidden formulas are much easier, with a number of branchings between 2 and 5 orders of magnitude smaller. It appears that while zChaff's smarts allow it to quickly "zero in" on a single hidden assignment, when there are two hidden assignments it only "stumbles" upon one of them after a search that is nearly as exhaustive as for unsatisfiable random 3-SAT formulas of the same density.

We performed similar experiments on Satz, and found that all three types of formulas had roughly the same running time. The reason for this is that while Satz makes intelligent decisions about which variable to branch on, it tries these branches in a fixed order, attempting first to set each variable false (Li 1997). Therefore, a given hidden assignment will appear at a uniformly random leaf in Satz's search tree. We believe that if Satz were modified to use, say, the majority heuristic to choose a variable's first value, it would find 1-hidden formulas much easier than 2-hidden or 0-hidden ones. We plan to confirm this experimentally.

SP

SP is a recently introduced incomplete solver (Mézard & Zecchina 2002) based on a generalization of belief propagation the authors call *survey propagation*. We compared SP's performance on the three types of problems near the satisfiability threshold. For $n = 10^4$ SP solves 2-hidden formulas at densities somewhat above the threshold, up to $r \approx 4.8$, while it solves 1-hidden formulas up to $r \approx 5.6$. These results stayed essentially the same when we increased n to 2×10^4, and when we used 5000 iterations, instead of the default 1000, for SP's convergence procedure.

Presumably the 1-hidden formulas are easier for SP since the "messages" from clauses to variables tend to push the algorithm towards a hidden assignment. Having two hidden assignments appears to cancel these messages out to some extent, causing SP to fail at a lower density. However, this argument doesn't explain why the threshold for 2-hidden formulas should be higher than the satisfiability threshold; nor does it explain why SP doesn't solve 1-hidden formulas for arbitrarily large r. Indeed, we find this latter result surprising, since as r increases the messages should point more and more consistently towards the hidden assignment in the 1-hidden case. These observations seem to us like an interesting direction for further investigation of SP.

WalkSAT

We conclude with a local search algorithm, WalkSAT. Unlike complete solvers, WalkSAT can solve problems with $n = 10^4$ fairly close to the threshold, although not quite up to the densities reachable with SP. We performed experiments on WalkSAT both with a random initial state, and with a biased initial state where the algorithm is lucky enough to start with 75% agreement with one of the hidden assignments (note that this is exponentially unlikely). In both cases, we performed trials of 10^8 flips for each formula, without random restarts or novelty heuristics, performing random or greedy flips with equal probability. Since random initial states have w.h.p. roughly 50% overlap with both hidden assignments, we expect their attractions to cancel out so that WalkSAT will have difficulty finding either of them. On the other hand, if we begin with a biased initial state, then the attraction from the nearby assignment will be much stronger than the other one; this situation is similar to a 1-hidden formula, and we expect WalkSAT to find it easily.

In Figure 3 we measure WalkSAT's performance on the three types of problems with $n = 10^4$ and r ranging from 3.7 to 5.5, and compare them with 0-hidden formulas for r ranging from 3.7 up to 4.1, just below the satisfiability threshold. We see that, below the threshold, the 2-hidden formulas are just as hard as the 0-hidden ones when WalkSAT sets its initial state randomly, and become the hardest when $r \approx 4.2$ (where 10^8 flips no longer suffice to solve them). For all r the 2-hidden formulas are much harder than the 1-hidden ones, unless we are lucky enough to start with a biased initial state. In Figure 4, we compare 0-hidden, 1-hidden and 2-hidden problems with both types of initial state as a function of n at $r = 4.25$ (we filter the 0-hidden problems and only count the satisfiable ones). Consistent with experiments of (Barthel *et al.* 2002), the median running time is

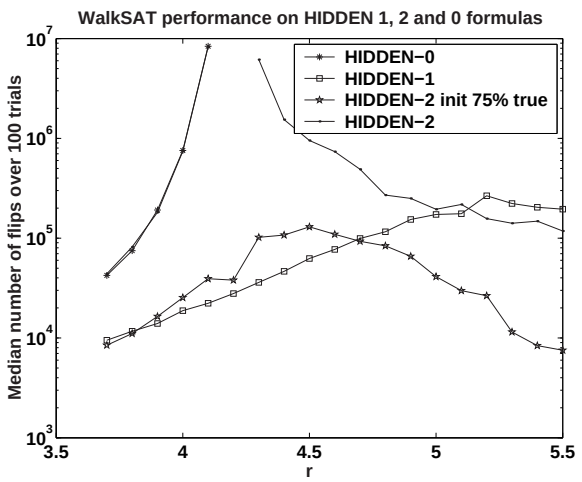


Figure 3: The median number of flips needed by WalkSAT for 3-SAT formulas of all three types as a function of r with $n = 10^4$. Below the threshold, 2-hidden formulas are just as hard as 0-hidden ones, and their running time increases steeply as we approach the threshold. For all r, 2-hidden formulas are much harder than 1-hidden ones, unless the algorithm starts with a (very lucky) biased initial state.

polynomial in n. However, the power of n (the slope in the the log-log plot) is the same for the 2-hidden and 0-hidden formulas, while for 1-hidden formulas (and 2-hidden with biased initial state) it is much lower.

Conclusions

We introduce an extremely simple new generator of random satisfiable 3-SAT instances which is amenable to all the mathematical tools developed for the study of random 3-SAT. Our generator appears to produce instances that are as hard as random 3-SAT instances, in sharp contrast to those with a single hidden assignment. Our experiments show that this hardness is quite robust, both above and below the satisfiability threshold, and for very different algorithms, i.e., DPLL solvers (zChaff and Satz), local search algorithms (WalkSAT), and survey propagation (SP).

We believe that random 2-hidden instances could make excellent satisfiable benchmarks, especially just around the satisfiability threshold, say at $r = 4.25$ where they appear to be the hardest for WalkSAT (although beating SP requires somewhat higher densities). We propose several exciting directions for further work:

1. Proving that the expected running time of natural DPLL algorithms on 2-hidden formulas is exponential in n.
2. Explaining the different threshold behaviors of SP on 1-hidden and 2-hidden formulas.
3. Understanding how long WalkSAT takes at the midpoint between the two hidden assignments, before it becomes sufficiently unbalanced to converge to one of them.
4. Studying random 2-hidden formulas in the dense case where there are $\omega(n)$ clauses.

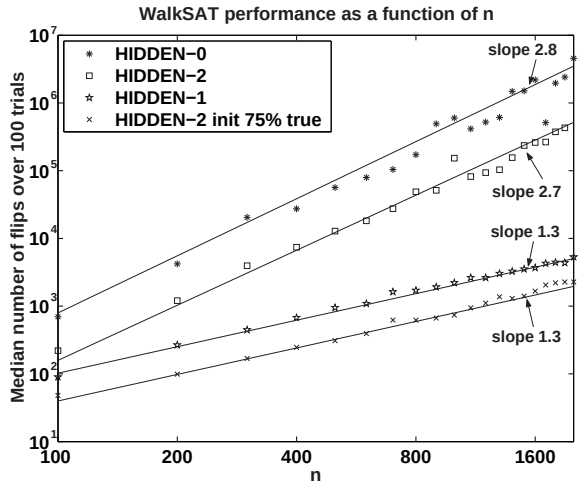


Figure 4: The median number of flips needed by `WalkSAT` to solve 0-, 1-, and 2-hidden 3-SAT formulas at $r = 4.25$, with n ranging from 100 to 2000. While the median running time is polynomial, for 2-hidden problems it grows just as fast as for 0-hidden ones, while 1-hidden problems are similar to 2-hidden problems with a biased initial state.

References

Achlioptas, D., and Moore, C. 2002a. Almost all graphs with average degree 4 are 3-colorable. *STOC* 199–208.

Achlioptas, D., and Moore, C. 2002b. The asymptotic order of the random k-sat threshold. *FOCS* 779–788.

Achlioptas, D.; Beame, P.; and Molloy, M. 2001. A sharp threshold in proof complexity. *STOC* 337–346.

Achlioptas, D.; Gomes, C.; Kautz, H.; and Selman, B. 2000. Generating satisfiable problem instances. *AAAI* 256–261.

Achlioptas, D. 2001. Lower bounds for random 3-sat via differential equations. *Theor. Comp. Sci.* 265:159–185.

Asahiro, Y.; Iwama, K.; and Miyano, E. 1996. Random generation of test instances with controlled attributes. *DIMACS Series in Disc. Math. and Theor. Comp. Sci.* 26.

Barthel, W.; Hartmann, A.; Leone, M.; Ricci-Tersenghi, F.; Weigt, M.; and Zecchina, R. 2002. Hiding solutions in random satisfiability problems: A statistical mechanics approach. *Phys. Rev. Lett* 88(188701).

Chao, M., and Franco, J. 1986. Probabilistic analysis of two heuristics for the 3-satisfiability problem. *SIAM J. Comput.* 15(4):1106–1118.

Cheeseman, P.; Kanefsky, R.; and Taylor, W. 1991. Where the really hard problems are. *IJCAI* 163–169.

Cocco, S., and Monasson, R. 2001a. Analysis of the computational complexity of solving random satisfiability problems using branch and bound search algorithms. *Eur. Phys. J. B* 22:505–.

Cocco, S., and Monasson, R. 2001b. Trajectories in phase diagrams, growth processes and computational complexity: how search algorithms solve the 3-satisfiability problem. *Physical Review Letters* 86:1654–.

Hogg, T.; Huberman, B.; and Williams, C. 1996. Phase transitions and complexity. *Artificial Intelligence* 81. special issue.

Johnson, D.; Aragon, C.; McGeoch, L.; and C., S. 1989. Optimization by simulated annealing: An experimental evaluation. *Operations Research* 37(6):865–892.

Kautz, H.; Ruan, Y.; Achlioptas, D.; Gomes, C.; Selman, B.; and Stickel, M. 2001. Balance and filtering in structured satisfiable problems. *IJCAI* 351–358.

Kirousis, L.; Kranakis, E.; Krizanc, D.; and Stamatiou, Y. 1998. Approximating the unsatisfiability threshold of random formulas. *Random Structures Algorithms* 12(3):253–269.

Li, C. 1997. Anbulagan, heuristics based on unit propagation for satisfiability problems. *IJCAI* 366–371.

Massacci, F. 1999. Using walk-sat and rel-sat for cyptographic key search. *IJCAI '99* 290–295.

Mézard, M., and Zecchina, R. 2002. Random k-satisfiability: from an analytic solution to a new efficient algorithm. *Phys. Rev. E* 66. Available at: http://www.ictp.trieste.it/~zecchina/SP/.

Mitchell, D.; Selman, B.; and Levesque, H. 1992. Hard and easy distributions of sat problems. *AAAI* 459–465.

Morris, P. 1993. The breakout method for escaping from local minima. *AAAI* 40–45.

Selman, B.; Kautz, H.; and Cohen, B. 1996. Local search strategies for satisfiability testing. *2nd DIMACS Challange on Cliques, Coloring, and Satisfiability*.

Shaw, P.; Stergiou, K.; and Walsh, T. 1998. Arc consistency and quasigroup completion. *ECAI, workshop on binary constraints*.

Van Gelder, A. 1993. Problem generator `mkcnf.c`. *DIMACS*. Challenge archive.

Wormald, N. 1995. Differential equations for random processes and random graphs. *Ann. Appl. Probab.* 5(4):1217–1235.

Zhang, L. zchaff source site. Available at: http://ee.princeton.edu/~chaff/zchaff.php.