# Faster Random $k$-CNF Satisfiability

**Andrea Lincoln**
MIT, Cambridge, MA, USA
andreali@mit.edu

**Adam Yedidia**
MIT, Cambridge, MA, USA
adamy@mit.edu

## Abstract

We describe an algorithm to solve the problem of Boolean CNF-Satisfiability when the input formula is chosen randomly.

We build upon the algorithms of Schöning 1999 and Dantsin et al. in 2002. The Schöning algorithm works by trying many possible random assignments, and for each one searching systematically in the neighborhood of that assignment for a satisfying solution. Previous algorithms for this problem run in time $O(2^{n(1-\Omega(1)/k)})$.

Our improvement is simple: we count how many clauses are satisfied by each randomly sampled assignment, and only search in the neighborhoods of assignments with abnormally many satisfied clauses. We show that assignments like these are significantly more likely to be near a satisfying assignment. This improvement saves a factor of $2^{n\Omega(\lg^2 k)/k}$, resulting in an overall runtime of $O(2^{n(1-\Omega(\lg^2 k)/k)})$ for random $k$-SAT.

**2012 ACM Subject Classification** Theory of computation → Random search heuristics

**Keywords and phrases** Random k-SAT, Average-Case, Algorithms

**Digital Object Identifier** 10.4230/LIPIcs.ICALP.2020.78

**Category** Track A: Algorithms, Complexity and Games

**Related Version** https://arxiv.org/abs/1903.10618

**Funding** *Andrea Lincoln*: Some of this work was completed while this student was an intern at VMware. This work supported in part by NSF Grants CCF-1909429, CCF-1417238, CCF-1528078 and CCF-1514339, and BSF Grant BSF:2012338.

**Acknowledgements** We are very grateful to Greg Valiant, Virginia Williams, and Nikhil Vyas for their helpful conversations and kind support. Additionally, we are very appreciative of the email correspondence we had with Amin Coja-Oghlan, Allan Sly, and Nike Sun, who answered our many questions. We would also like to thank reviewers for their comments and suggestions. We would especially like to thank an exceptionally detailed and helpful review that we received on our paper last year.

## 1 Introduction

The Boolean Satisfiability problem, known as SAT for short, is one of the best-known and most well-studied problems in computer science (e.g. [28, 1, 19, 3, 14]). In its general form, it describes the following problem: given an input formula $\phi$ composed of conjunctions, disjunctions, and negations of a list of Boolean-valued variables $(x_1, x_2, \ldots, x_n)$, determine whether or not there exists an assignment of variables to Boolean values such that $\phi$ evaluates to TRUE. SAT was the first problem shown to be NP-complete [12, 24].

Every Boolean formula $\phi$ can be written in *conjunctive normal form*, meaning that it is written as the logical conjunction of a series of disjunctive clauses. Each disjunctive clause takes as its value the logical disjunction of a series of *literals*, which takes on either the same value of one of the variables $x_i$ or the negation of that value.

© Andrea Lincoln and Adam Yedidia;
licensed under Creative Commons License CC-BY

If we constrain the input formula $\phi$ to contain only disjunctive clauses that are of size $k$ or smaller, then that more constrained problem is known as *$k$-Satisfiability*, or *$k$-SAT* for short. When $k > 2$ it is known to be NP-complete [22]. As $k$ grows, the best known runtime of the worst-case $k$-SAT problem, $O(2^{1-1/\Theta(k)})$, grows [21, 28].

It is well-known that in real-world Boolean Satisfiability problems, SAT solvers often vastly outperform the best known theoretical runtimes [15, 31]. One possible explanation for this gap in performance is that most input formulas are easily solved without much computation being necessary, but that there exists a "hard core" of difficult-to-solve formulas that are responsible for the apparent difficulty of worst-case SAT.

Another possible explanation for this gap in performance is that, in practice, people usually care about highly structured formulas that are much easier to solve than typical formulas – according to this explanation, there would be an "easy core" of tractable formulas that are responsible for the apparent simplicity of most practical SAT problems.

To try to distinguish between these two explanations, one can study *random Satisfiability*: Boolean Satisfiability for which the input formula $\phi$ is chosen according to some known uniform probability distribution $D_\Phi$, and where we expect to be able to return the correct answer (satisfiable or unsatisfiable) with probability that is exponentially close to 1 in the size of the input. Random k-SAT is a very well studied problem (e.g. [3, 16, 27, 10, 7, 26, 33]).

Typically, attention is restricted to $k$-CNF formulas whose ratio of clauses to variables is *at the threshold*, meaning that the number of clauses $m$ is drawn from a Poisson distribution centered at $d_k n$, where $n$ is the number of variables and $d_k$ is a function of $k$ close to $2^k \ln(2) - \frac{1}{2}(1 + \ln(2))$ [16]. Such formulas are conjectured to be the hardest instances for a given $n$ [13, 31]. It was shown by Ding, Sly, and Sun [16] that away from this threshold, formulas are either overwhelmingly satisfied or overwhelmingly unsatisfied, making the problem less interesting. Notably, away from this threshold one can simply return True or False based on the number of clauses and give the correct answer with high probability. For our purposes, this is a very useful guarantee to have; this is why we use their definition throughout the paper. We go into much greater detail about $D_\Phi$ and the threshold in Section 2.

Away from the threshold, polynomial-time algorithms for SAT have been found and analyzed, first by Chao and Franco [26], and later by Coja-Oghlan et al. [10, 7]. Additionally, a recent result by Vyas and Williams [33] re-analyzes the algorithm of Paturi et al. [29] in the case when the input is drawn from a random distribution, and finds the algorithm to run faster on average in this case by a factor of $2^{n\Omega(\lg k)/k}$, giving a running time of $O(2^{n(1-\Omega(\lg k)/k)})$.

We build upon the work of Schöning [30] and Dantsin et al [14] to solve random $k$-SAT in time $O(2^{n(1-\Omega(\lg^2 k)/k)})$. This represents an algorithmic improvement of $2^{n\Omega(\lg^2 k)/k}$ over the runtime of the algorithm of Paturi et al. as analyzed by Vyas and Williams in [33].

## 1.1 A New Algorithm

In this paper, we restrict our attention to the problem of random $k$-CNF Satisfiability in the limit of large $k$, which approaches general Boolean CNF-Satisfiability. Our algorithm improves upon the previous best known algorithm for solving random $k$-SAT in the limit of large $k$, assuming that the input formulas are chosen according to a known uniform distribution.

Our algorithm improves the running time of $k$-CNF Satisfiability at the threshold by modifying the algorithms of Schöning and Dantsin et al. to only explore in the neighborhood of those sampled assignments that pass an additional test. By adding this test, we get a $2^{n\Omega(\lg^2 k)/k}$ improvement in the runtime of the algorithm. The test is simple: we count how many clauses are satisfied, and if that number is large, only then do we search in the neighborhood of the assignment. In Appendix G.2 of our full version [25], we provide additional motivation for why our improved running time is remarkable.

▶ **Theorem 1** (Main Theorem Informal). *Let $\phi$ be drawn uniformly at random from formulas at the threshold (defined formally in Section 2). There exists an algorithm, $\alpha$-SampleAndTest (described in Section 3), such that:*

- *If $\phi$ is satisfiable, then with probability at least $1-3\cdot2^{-n/(3\ln(2)2^k)}$, $\alpha$-SampleAndTest returns an assignment $\vec{a}$ that satisfies $\phi$.*
- *If $\phi$ is not satisfiable, then $\alpha$-SampleAndTest will return False with certainty.*
- *$\alpha$-SampleAndTest($\phi$) will run in time $O(2^{n(1-\Omega(\lg^2 k)/k)})$.*

A key technique in the proof of our result is connecting a different distribution over inputs (the *planted $k$-SAT distribution*) to the uniformly random $k$-SAT distribution. Reductions between planted $k$-SAT and random $k$-SAT have been shown in previous work as well [6, 4, 33]. In the planted $k$-SAT distribution, an assignment, $\vec{a}$, is picked first. The formula $\phi$ is selected uniformly over $k$-SAT clauses *conditioned on* $\vec{a}$ satisfying those clauses. As a result, the planted distribution has a bias towards picking formulas that have many satisfying assignments, relative to the uniform distribution over all satisfiable formulas. For this reason, the planted distribution tends to generate easier-to-solve formulas $\phi$ than the uniform distribution [18]. We also find that the planted distribution is more easily analyzed.

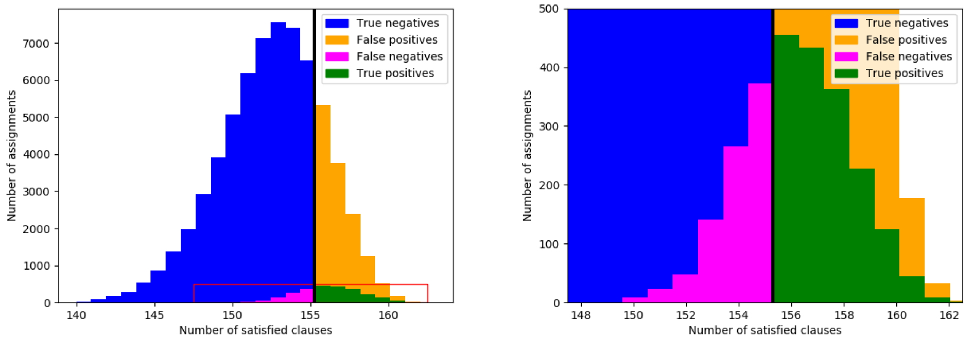
It would be possible, and simpler, to analyze our algorithm only in the planted distribution over formulas. This would *not*, however, correspond to a complete analysis of the algorithm in the random case. In this work, we begin by analyzing the performance of our algorithm when run on inputs drawn from the planted distribution. We show that algorithms with a sufficiently low probability of failure in the planted distribution over input formulas continue to have a low probability of failure in the uniform distribution over input formulas; see Lemma 37 of our full version [25]. Similar reductions have been proven in previous work [6, 4, 33].

The bulk of the analysis of our algorithm presented in this paper will focus on four quantities. Informally:

1. The *true positive rate* $p_{TP}$ describes the fraction of all assignments that are both close to a satisfying assignment in Hamming distance and satisfy a large number of clauses.
2. The *false negative rate* $p_{FN}$ describes the fraction of all assignments that are close to a satisfying assignment in Hamming distance, but do not satisfy a large number of clauses.
3. The *false positive rate* $p_{FP}$ describes the fraction of all assignments that are not close to any satisfying assignment in Hamming distance, but satisfy a large number of clauses.
4. The *true negative rate* $p_{TN}$ describes the fraction of all assignments that are neither close to any assignment in Hamming distance, nor satisfy a large number of clauses.

By showing that the true positive rate is large enough relative to the false positive rate, we show that we do not too often perform a "useless search," i.e. one that will not find a satisfying assignment. And by showing that the true positive rate is large enough relative to the total number of possible assignments, we show that we eventually do find a satisfying assignment without needing to take too many samples. See Fig. 1 for an illustration of these concepts.

To show that our algorithm achieves the desired runtime, we must demonstrate two things. First, we must show that false positives are sufficiently rare; in other words, conditioned on an assignment passing our test, it is sufficiently likely to be a small-Hamming-distance assignment. We prove this in Appendix A of our full version [25]. Second, we must show




(a) The full histogram.

(b) A magnification of the red region shown on the left.

**Figure 1** A histogram of how many clauses are satisfied by every possible assignment. In this example, there are $n = 16$ variables, $m = 163$ clauses, and $k = 4$ literals per clause. For the example, we take $T = 155.5$ to be the clause-satisfaction threshold above which we explore further, and $\alpha n = 4$ to be the small-Hamming-distance threshold at which the exhaustive search algorithm finishes. (In actual runs of the algorithm, both of these parameters are selected more conservatively; we chose these parameters for clarity.)

that true positives are sufficiently common; in other words, conditioned on an assignment being close in Hamming distance to a satisfying assignment, it is sufficiently likely to pass our test. We prove this in Appendix B of our full version [25].

We also note that our algorithm can potentially be used as the seed for a worst-case algorithm. Informally, the correctness of the analysis in this paper depends only on the false positive and false negative rates being sufficiently low. As long as the inputs are guaranteed to come from a family of formulas for which this is the case, our algorithm will work even in the worst case. Or, to put it another way, to build a working worst-case algorithm using our algorithm as a template, one may now restrict one's attention to solving input formulas for which assignments in the neighborhood of the solution do *not* have an abnormally-high number of satisfied clauses; our algorithm can solve the others.

## 1.2 Previous work

Satisfiability and $k$-SAT have been thoroughly studied. We will cover some of the previous work in the area, focusing on the Random $k$-SAT problem.

### Structural Results About Random $k$-SAT

To make the study of the Satisfiability of random formulas interesting, it is important to choose the probability distribution over formulas $D_\phi$ judiciously. In particular, $D_\phi$ must contain formulas where the ratio of Boolean variables to disjunctive clauses is such that the resulting formulas are neither overwhelmingly satisfiable, nor overwhelmingly unsatisfiable. Let $n$ be the number of variables, and $m$ be the number of clauses. If $n \gg m$, then nearly all formulas chosen uniformly from $D_\phi$ will be satisfiable; if $m \gg n$, then nearly all formulas will be unsatisfiable. In order for the problem of correctly identifying formulas as satisfiable or unsatisfiable to be nontrivial, we must choose $m$ and $n$ to be at the right ratio. Throughout this paper we will refer to the ratio of $m$ to $n$ as the *density* of a formula.

In work by Ding, Sly and Sun [16], it was shown that a sharp threshold exists between formulas which are satisfied with high probability and those that are unsatisfied with high probability. More precisely, they describe what happens when the number of clauses $m$ is drawn from a Poisson distribution with mean $d_k n$. When the number of clauses drawn is below $(d_k - \epsilon)n$, only an exponentially small fraction of formulas will be unsatisfiable; when the number of clauses drawn is greater than $(d_k + \epsilon)n$, only an exponentially small fraction of formulas will be satisfiable. This holds true for any $\epsilon > 0$ constant in $n$.

### Previous Average-Case $k$-SAT Algorithms

Feldman et al. studied planted random $k$-SAT and found that given $m = \Omega(n \lg n)$ clauses, the planted solution can be determined using statistical queries [18]. Feldman et al. also conjecture that planted $k$-SAT is easier than random $k$-SAT more generally. Previous work has shown a connection between algorithms that work in the planted distribution and algorithms that work in the random distribution [4, 6, 33]. An algorithm was found by Valiant which runs in time $O(2^{n(1-\Omega(\log(k))/k)})$ at the threshold [32], improving upon PPSZ [28]. Additionally, Vyas and Williams [33] obtained the same runtime by re-analyzing the algorithm of Paturi et al. [29] in the random case.

Some studies of random $k$-SAT have focused on refutation [9, 8, 20, 5]. Refutation aims to return a short certificate of unsatisfiability. For example Coja-Oghlan, Goerdt, and Lanka give an algorithm that provides refutations with high probability when $k = 3$ and $m > \ln(n)^6 n^{3/2}$. Refutation is quite difficult; note that the $m$ is much larger than the threshold which sits at $\Theta(2^k n)$. We, however, focus on returning a satisfying assignment if the formula is satisfied with high probability.

Some studies of random $k$-SAT focus on the case where $k$ is small (e.g. [9, 8, 2, 17]). We, however, focus on the asymptotic behavior when $k$ is large.

### Worst-Case $k$-SAT Algorithms

The previously best-known worst-case $k$-SAT algorithms for large $k$ are due to Paturi et al. who get a running time of $O(2^{n(1-\Omega(1)/k)})$ [28]. Previous work by Schöning gave an algorithm to solve $k$-SAT in the worst case with an expected running time of $O(2^{n(1-\Omega(1)/k)})$ [30]. Dantsin et al. make the algorithm deterministic [14]. Our algorithm is a modification of the algorithms of both Schöning's and Dantsin et al. Their algorithm runs by choosing an assignment at random, and searching in the immediate neighborhood of that assignment by repeatedly choosing an unsatisfied clause and flipping a variable in that clause to satisfy it. They perform the search near the randomly chosen assignment via an exhaustive search. Their algorithm is an improvement over a naive brute-force algorithm because of the savings that result from only considering variable-flips that could possibly cause the formula to become satisfied (rather than also exploring variable-flips that can't possibly be helpful).

## 2 Preliminaries

In this section we will give the definition of random $k$-CNF Satisfiability (random $k$-CNF SAT) at the threshold. We additionally present definitions of several important distributions and functions that are used later in the paper.

## Notation for This Paper

We use $x \sim D$ to indicate that $x$ is a random value drawn from the distribution $D$.

We use the standard notation that $\lg(n) = \log_2(n)$.

We use $f(x) = O^*(g(x))$ to denote that there exists some constant $c$ such that $f(x) = O(g(x)x^c)$. So, to say it another way, $f(x)$ grows at most as quickly as $g(x)$, ignoring polynomial and constant factors.

We often use "small-Hamming-distance assignment" to mean an assignment that is a small Hamming distance from a satisfying assignment.

## Definitions for This Paper

▶ **Definition 2.** *Let $D_{replace}(n,k)$ be the uniform distribution over clauses on $k$ variables where those $k$ variables are chosen with replacement (e.g. $(\bar{x} \vee x \vee y)$ would be a valid clause). Under this definition, there are $n^k 2^k$ possible clauses.*

▶ **Definition 3.** *The density of a formula $\phi$ with $m$ clauses and $n$ variables is $m/n$.*

We will now define the *satisfiability threshold.* Informally, this is a density of clauses such that formulas drawn from below this threshold are with high probability (whp) satisfied, and those formulas drawn from above the threshold are whp unsatisfied.

▶ **Definition 4.** *The satisfiability threshold, $d_k$, is a ratio of clauses to variables such that for all $\epsilon > 0$:*

- *If $m$ is drawn from $Pois[(d_k - \epsilon)n]$, the Poisson distribution with mean $(d_k - \epsilon)n$, and $\phi$ is formed by picking $m$ clauses independently at random from $D_{replace}(n,k)$, then $\phi$ is whp **satisfied**.*
- *If $m$ is drawn from $Pois[(d_k + \epsilon)n]$, the Poisson distribution with mean $(d_k + \epsilon)n$, and $\phi$ is formed by picking $m$ clauses independently at random from $D_{replace}(n,k)$, then $\phi$ is whp **unsatisfied**.*

▶ **Definition 5.** *Let $d_k$ be the density of clauses such that SAT is at its threshold.*

Note that it is not immediate that a satisfiability threshold exists for any given $k$. However, Jian Ding, Allan Sly, and Nike Sun showed that this threshold exists for sufficiently large $k$ [16]. It has been proven that

$$2^k \ln(2) - \frac{1}{2}(1 + \ln(2)) - \epsilon_k \leq d_k \leq 2^k \ln(2) - \frac{1}{2}(1 + \ln(2)) + \epsilon_k,$$

where $\epsilon_k$ is a term that tends to 0 as $k$ grows [23, 11]. It follows that there exists a large enough $k$ such that $2^k \ln(2) - 1 \leq d_k$. Also, there exists a large enough $k$ such that $2^k \ln(2) \geq d_k$.

This density determines the distribution over the number of clauses put in the formula. Specifically, $m$ is drawn from $Pois[d_k n]$. However, we can say that with high probability the number of clauses is nearly $d_k n$ (see Lem. 9).

For many proofs it is convenient to assume $k$ is large (e.g. when $k$ is large, $2^k > 10k$ not just asymptotically but also numerically). We will now define $k^\star$. It will be a value such that $k = k^\star$ is large enough that both $d_k$ is known to be close to $2^k \ln(2) - \frac{1}{2}(1 + \ln(2))$ and large enough for our proofs that depend on $k$ being large.

▶ **Definition 6.** *Let $\epsilon_k = |d_k - 2^k \ln(2) + \frac{1}{2}(1 + \ln(2))|$.*

▶ **Definition 7.** *Let $k_\epsilon$ be the minimum value such that for all $k \geq k_\epsilon$ we have that $\epsilon_k < \frac{1+\ln(2)}{2}$.*

▶ **Definition 8.** *Let* $k^\star = \max(60, k_\epsilon)$*.*

Our choice of 60 in the above is somewhat arbitrary. When $k \geq 60$ the proofs in Appendix C of our full version [25] are simpler, so we analyze our core algorithm in that regime.

▶ **Lemma 9.** *If* $\epsilon_k < \frac{1+\ln(2)}{2}$ *then* $Pr[m > (d_k+1)n] + Pr[m < (d_k-1)n] \leq 2 \cdot 2^{-n/(3\ln(2)2^k)}$*.*

**Proof.** We apply the multiplicative form of the Chernoff bound. We have that $(d_k+1)n/(d_k n) = 1 + 1/d_k$. We also have that $(d_k-1)n/(d_k n) = 1 - 1/d_k$. This gives us

$$Pr[m > (d_k+1)n] + Pr[m < (d_k-1)n] \leq 2^{-(d_k)^{-2}d_k n/3} + 2^{-(d_k)^{-2}d_k n/2}.$$

Which means

$$Pr[m > (d_k+1)n] + Pr[m < (d_k-1)n] \leq 2^{-n/(3d_k)} + 2^{-n/(2d_k)}.$$

$$Pr[m > (d_k+1)n] + Pr[m < (d_k-1)n] \leq 2 \cdot 2^{-n/\left(3(\ln(2)2^k - \frac{1}{2}(1+\ln(2))\right)}.$$

$$Pr[m > (d_k+1)n] + Pr[m < (d_k-1)n] \leq 2 \cdot 2^{-n/(3\ln(2)2^k)}.$$ ◀

It follows that if our algorithm works efficiently for all values of $m \in [(d_k-1)n, (d_k+1)n]$, then it works with high probability at the threshold.

Below are some definitions used in later sections.

▶ **Definition 10.** *Let* $D_\Phi(n,k)$ *be the distribution over formulas* $\phi$ *where all clauses are chosen independently from* $D_{replace}$ *and the number of clauses is chosen from a Poisson distribution with mean* $d_k n$*.*

▶ **Definition 11.** *Let* $D_R(m,n,k)$ *be the distribution over formulas* $\phi$ *where all* $m$ *clauses are chosen independently from* $D_{replace}$*.*

▶ **Definition 12.** *Let* $D_S(m,n,k)$ *be the uniform distribution over* satisfied *formulas* $\phi$ *where all* $m$ *clauses are chosen from* $D_{replace}$*.*

▶ **Definition 13.** *Let* $D_{pc}(n,k,\vec{a})$ *(which we refer to as "the planted-clause distribution") be the uniform distribution over the* $(2^k-1)n^k$ *clauses* $c$ *which are satisfied by* $\vec{a}$*.*

▶ **Definition 14.** *Let* $D_{pa}(m,n,k,\vec{a})$ *(which we refer to as "the planted distribution") be the distribution over formulas* $\phi$ *where every clause is picked IID from* $D_{pc}(n,k,\vec{a})$*. Note that this is equivalent to the uniform distribution over formulas* $\phi$ *which are satisfied by* $\vec{a}$ *and where all* $m$ *clauses are in the support of* $D_{replace}$*.*

▶ **Definition 15.** *Let* $U_{\vec{a}}(n)$ *be the uniform distribution over assignments of length* $n$*,* $\{0,1\}^n$*.*

▶ **Definition 16.** *Let* $\text{NUMCLAUSESSAT}(\phi, \vec{v})$ *be the number of clauses in* $\phi$ *satisfied by the assignment* $\vec{v}$*.*

▶ **Definition 17.** *Let* $\text{NUMCLAUSESUNSAT}(\phi, \vec{v})$ *be the number of clauses in* $\phi$ *left unsatisfied by the assignment* $\vec{v}$*.*

## 3 Algorithm

We will describe our algorithm for random $k$-SAT in this section.

Informally, our algorithm works as follows. Given an input formula, we will sample many randomly-chosen assignments. On those that have a high number of satisfied clauses, we will run the deterministic algorithm for finding a satisfying assignment given an assignment that is within a Hamming distance of at most $\alpha n$ of that satisfying assignment (i.e. a small-Hamming-distance assignment).

Unsurprisingly, in the average case, small-Hamming-distance assignments satisfy more clauses than random assignments[1]. In fact, for many choices of criterion there will be a discrepancy between the values achieved by small-Hamming-distance assignments and random assignments. Lemma 30 of our full version [25], which characterizes this discrepancy, is general enough to be applied immediately to analyzing algorithms that make use of any clause-specific criterion.

We note the following from previous work:

▶ **Lemma 18** (Small Hamming Distance Search [14])**.** *There is a deterministic algorithm* SAT-FROM-$\alpha$-SMALL-HD($\phi, \vec{v}$) *which given*

- *a $k$-CNF formula $\phi$ on $m$ clauses and $n$ variables, and*
- *an assignment $\vec{v}$ which has Hamming distance $\alpha n$ from a true satisfying assignment $\vec{a}^*$,*

*will return a satisfying assignment within Hamming distance $\alpha n$ of $\vec{v}$ if one exists in $k^{\alpha n}$ time.*

This algorithm simply takes the assignment $\vec{a}$ and branches on the first unsatisfied clause, trying all possible variable flips. For each assignment resulting from these possible variable flips, the algorithm repeats the process in what is now the first unsatisfied clause, until it either finds a satisfying assignment or has searched $\alpha n$ flips from the original assignment. This will deterministically yield a satisfying assignment, should one exist, within a Hamming distance of $\alpha n$ of the original assignment.

So, if we find a small-Hamming-distance assignment and run SAT-FROM-$\alpha$-SMALL-HD($\phi, \vec{a}$) on this assignment, we are guaranteed to find the satisfying assignment. Therefore, we could randomly sample points until we expect to find an assignment at Hamming distance $\alpha n$ from the satisfying assignment (call this an $\alpha$-small-Hamming-distance assignment). This is indeed what Schöning's algorithm does for $\alpha = \Theta(1/k)$ [30].

A general class of improvements to this algorithm work by running SAT-FROM-$\alpha$-SMALL-HD($\phi, \vec{a}$) on only a cleverly-chosen subset of these sampled assignments. In our case, we choose this set to be assignments that satisfy an unusually large number of clauses, but in principle one could use any membership criterion for this set.

Let $M$ be the runtime of the membership test for the set of assignments, and let $p_{TP}$, $p_{FP}$, $p_{FN}$, and $p_{TN}$ represent the fraction of assignments that are true positives, false positives, false negatives, and true negatives respectively. Here, just as in Section 1.1, we use "positive" or "negative" to mean an assignment that passes or doesn't pass the test for membership, respectively. The truth or falsehood of that positive or negative represents whether or not that assignment actually has a satisfying assignment within small Hamming distance.

[1] Consider changing one variable's assignment at random; in this case, almost all clauses will remain satisfied. This phenomenon persists even when we flip several variables at once.

We will have to draw samples until we would have found a satisfying assignment with high probability were one to exist. Next, we will have to run SAT-FROM-$\alpha$-SMALL-HD($\phi, \vec{a}$) at least once to find the satisfying assignment itself. Finally, we will have to run it once more for every false positive we find. Hence, the the generalized running time of this class of algorithms is

$$O^*\left(\frac{M}{p_{TP}} + k^{\alpha n} + \left(\frac{p_{FP}}{p_{TP}}\right) k^{\alpha n}\right). \tag{1}$$

This general formula is a powerful tool for analyzing the runtimes of algorithms from this class. For example, if we apply it to analyzing the algorithm of [14], i.e. the special case where the test we use always returns a positive, we see that the third term in Equation (1) dominates, and that $p_{FP} \approx 1$ and $p_{TP} \approx \frac{\binom{n}{\alpha n}}{2^n}$, giving an overall expected runtime of $O^*(2^{n(1-\Theta(\frac{1}{k}))})$ when we choose $\alpha = \Theta(\frac{1}{k+1})$ (using tail bounds to convert $\binom{n}{\alpha n}$ to an exponential). In Appendix G.3 of our full version [25] we discuss a different deterministic search algorithm with a slightly improved runtime (yielding no relevant improvement on the runtime of the overall algorithm for our analysis).

Our algorithm presents improvements for large $k$, but for small $k$ we will simply use the previous algorithm of Dantsin et al [14].

▶ **Lemma 19** (Algorithm for Small $k$ [14]). *For $k \leq k^\star$ there exists a deterministic algorithm, DANTSINLS, that solves $k$-SAT in the worst case in time $2^{n(1-\gamma)}$ for some constant $\gamma > 0$.*

We will now give pseudocode for the $\alpha$-SAMPLEANDTEST algorithm in Algorithm 1. Let NUMCLAUSESSAT($\phi$,$\vec{a}$) return the number of clauses in $\phi$ satisfied by the assignment $\vec{a}$. In Appendix G.1 of our full version [25] we describe a different set of concepts with which the algorithm can be understood.

Note that our algorithm as stated is non-constructive due to our use of the constant $k^\star$. Other than this constant, our algorithm is explicit. While $k^\star$ is known to be constant [11], its exact value is currently unknown. We note in Appendix E of our full version [25] that finding the value of $k^\star$ is an open problem which, if solved, would make our algorithm constructive.

## 3.1 Correctness and Running Time

We will include the theorem statement of correctness and running time here. Its proof depends on bounds on the false positive rate and the true positive rate, which we prove in later sections. In particular, we show in Appendix A of our full version [25] that conditioned on an assignment passing the test, it is sufficiently likely to be an $\alpha$-small-Hamming-distance assignment. We additionally show in Appendix B of our full version [25] that conditioned on an assignment being an $\alpha$-small-Hamming-distance assignment, it is sufficiently likely to pass the test.

Note that much of our probability of returning the wrong value comes from our bounds on the probability that we are drawing a formula with length $m < (d_k - 1)n$. If we knew $m$ to be fixed and greater than $(d_k - 1)n$, we would have a lower error probability.

We will show that $\alpha$-SAMPLEANDTEST($\phi$) has one-sided error and returns the correct answer with high probability. Note that it returns the correct answer with high probability even conditioned on the input being unsatisfied or satisfied. We use Theorem 26 of our full version [25] to bound the false positive rate and use Lemma 43 of our full version [25] to bound the true positive rate, which gives us the desired result.

**Algorithm 1** $\alpha$-SampleAndTest($\phi$).

$\alpha$-SampleAndTest($\phi$):
**if** $k < k^\star$ **then**
  **return DantsinLS($\phi$)**
**end**
Initialize $S$ to the empty set.
▷ *For $k \geq k^\star$ we run our variant of local search:*
**for** $i \in [0, n^2 \cdot 2^n / \binom{n}{\alpha n}]$ **do**
  Sample an assignment $\vec{a}$ uniformly at random from $\{0,1\}^n$.
  ▷ *Only keep assignments which satisfy abnormally many clauses.*
  **if** *NumClausesSAT*$(\phi,\vec{a}) \geq (1 - \frac{1-(1-\alpha)^{2k}}{2^k-1})m$ **then**
    Add $\vec{a}$ to $S$.
    **if** $|S| > 4n^3 2^n / \left(\binom{n}{\alpha n} k^{\alpha n}\right) + 1$ **then**
      **return** False
    **end**
    Run SAT-from-$\alpha$-Small-HD($\phi, \vec{a}$).
    If an assignment was found, return it.
  **end**
**end**
**return** False

In the theorem that follows, we choose $\alpha$ such that $\alpha n$ is an integer. Specifically, we choose:

$$\alpha = \frac{\lfloor \frac{\lg(k)}{16k} n \rfloor}{n}.$$

Note that when we choose $\alpha$ to take on this value, it will always lie in the range $\frac{\lg(k)}{20k} \leq \alpha \leq \frac{\lg(k)}{16k}$ for large $n$.

▶ **Theorem 20.** *Assume $\phi$ is drawn from $D_\Phi(n,k)$. Let $\alpha = \frac{\lfloor n \lg(k)/(16k) \rfloor}{n}$.*

*Conditioned on there being at least one satisfying assignment to $\phi$, $\alpha$-SampleAndTest($\phi$) will return some satisfying assignment with probability at least $1 - 3 \cdot 2^{-n/(3\ln(2)2^k)}$.*

*Conditioned on there being no satisfying assignment to $\phi$, $\alpha$-SampleAndTest($\phi$) will return False with probability $1$.*

*$\alpha$-SampleAndTest($\phi$) will run in time*

$$O\left(2^{n(1-\Omega(\lg^2(k)/k)}\right).$$

**Proof.** Proof given in Appendix D of our full version [25]. ◀

## References


1 Scott Aaronson. P=?NP. *Electronic Colloquium on Computational Complexity (ECCC)*, 24:4, 2017. URL: `https://eccc.weizmann.ac.il/report/2017/004`.

2 D. Achioptas and G. B. Sorkin. Optimal myopic algorithms for random 3-sat. In *Proceedings 41st Annual Symposium on Foundations of Computer Science*, pages 590–600, November 2000. `doi:10.1109/SFCS.2000.892327`.

3 Dimitris Achlioptas. Random satisfiability. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 245–270. IOS Press, 2009. doi:10.3233/978-1-58603-929-5-245.

4 Dimitris Achlioptas and Amin Coja-Oghlan. Algorithmic barriers from phase transitions. In *49th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2008, October 25-28, 2008, Philadelphia, PA, USA*, pages 793–802, 2008. doi:10.1109/FOCS.2008.11.

5 Sarah R. Allen, Ryan O'Donnell, and David Witmer. How to refute a random CSP. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 689–708, 2015. doi:10.1109/FOCS.2015.48.

6 Eli Ben-Sasson, Yonatan Bilu, and Danny Gutfreund. Finding a randomly planted assignment in a random 3CNF. Technical report, In preparation, 2002.

7 Amin Coja-Oghlan. A better algorithm for random $k$-SAT. *SIAM Journal on Computing*, 39(7):2823–2864, 2010.

8 Amin Coja-Oghlan, Colin Cooper, and Alan M. Frieze. An efficient sparse regularity concept. *SIAM J. Discrete Math.*, 23(4):2000–2034, 2010. doi:10.1137/080730160.

9 Amin Coja-Oghlan, Andreas Goerdt, and André Lanka. Strong refutation heuristics for random k-sat. *Combinatorics, Probability & Computing*, 16(1):5–28, 2007. doi:10.1017/S096354830600784X.

10 Amin Coja-Oghlan, Michael Krivelevich, and Dan Vilenchik. Why almost all $k$-CNF formulas are easy. In *Proceedings of the 13th International Conference on Analysis of Algorithms, to appear*, 2007.

11 Amin Coja-Oghlan and Konstantinos Panagiotou. The asymptotic $k$-SAT threshold. *Advances in Mathematics*, 288:985–1068, 2016. doi:10.1016/j.aim.2015.11.007.

12 Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, May 3-5, 1971, Shaker Heights, Ohio, USA*, pages 151–158, 1971. doi:10.1145/800157.805047.

13 Stephen A Cook and David G Mitchell. Finding hard instances of the satisfiability problem. In *Satisfiability Problem: Theory and Applications: DIMACS Workshop*, volume 35, pages 1–17, 1997.

14 Evgeny Dantsin, Andreas Goerdt, Edward A. Hirsch, Ravi Kannan, Jon M. Kleinberg, Christos H. Papadimitriou, Prabhakar Raghavan, and Uwe Schöning. A deterministic $(2-2/(k+1))^n$ algorithm for $k$-SAT based on local search. *Theor. Comput. Sci.*, 289(1):69–83, 2002. doi:10.1016/S0304-3975(01)00174-8.

15 Leonardo de Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In C. R. Ramakrishnan and Jakob Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

16 Jian Ding, Allan Sly, and Nike Sun. Proof of the Satisfiability Conjecture for large $k$. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 59–68, 2015. doi:10.1145/2746539.2746619.

17 Olivier Dubois, Yacine Boufkhad, and Jacques Mandler. Typical random 3-sat formulae and the satisfiability threshold. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '00, pages 126–127, Philadelphia, PA, USA, 2000. Society for Industrial and Applied Mathematics. URL: http://dl.acm.org/citation.cfm?id=338219.338243.

18 Vitaly Feldman, Will Perkins, and Santosh Vempala. On the complexity of random satisfiability problems with planted solutions. *Electronic Colloquium on Computational Complexity (ECCC)*, 21:148, 2014. URL: http://eccc.hpi-web.de/report/2014/148.

19 Jun Gu, Paul W. Purdom, John Franco, and Benjamin W. Wah. Algorithms for the satisfiability (SAT) problem: A survey. In *Satisfiability Problem: Theory and Applications, Proceedings of a DIMACS Workshop, Piscataway, New Jersey, USA, March 11–13, 1996*, pages 19–152, 1996. doi:10.1090/dimacs/035/02.

20 Hiệp Hàn, Yury Person, and Mathias Schacht. Note on strong refutation algorithms for random k-sat formulas. *Electronic Notes in Discrete Mathematics*, 35:157–162, 2009. doi:10.1016/j.endm.2009.11.027.

21 Russell Impagliazzo and Ramamohan Paturi. On the complexity of $k$-SAT. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001. doi:10.1006/jcss.2000.1727.

22 Richard M. Karp. Reducibility among combinatorial problems. In *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA*, pages 85–103, 1972. URL: http://www.cs.berkeley.edu/%7Eluca/cs172/karp.pdf, doi:10.1007/978-1-4684-2001-2_9.

23 Lefteris M. Kirousis, Evangelos Kranakis, Danny Krizanc, and Yannis C. Stamatiou. Approximating the unsatisfiability threshold of random formulas. *Random Struct. Algorithms*, 12(3):253–269, 1998. doi:10.1002/(SICI)1098-2418(199805)12:3<253::AID-RSA3>3.0.CO;2-U.

24 Leonid A. Levin. Universal search problems. *Problems of Information Transmission*, 9(3), 1973.

25 Andrea Lincoln and Adam Yedidia. Faster random $k$-cnf satisfiability. *arXiv preprint*, 2019. arXiv:1903.10618.

26 Chao Ming-Te and John Franco. Probabilistic analysis of a generalization of the unit-clause literal selection heuristics for the $k$-satisfiability problem. *Information Sciences*, 51(3):289–314, 1990.

27 Eugene Nudelman, Kevin Leyton-Brown, Holger H. Hoos, Alex Devkar, and Yoav Shoham. Understanding random SAT: beyond the clauses-to-variables ratio. In *Principles and Practice of Constraint Programming - CP 2004, 10th International Conference, CP 2004, Toronto, Canada, September 27 - October 1, 2004, Proceedings*, pages 438–452, 2004. doi:10.1007/978-3-540-30201-8_33.

28 Ramamohan Paturi, Pavel Pudlák, Michael E. Saks, and Francis Zane. An improved exponential-time algorithm for $k$-SAT. *J. ACM*, 52(3):337–364, 2005. doi:10.1145/1066100.1066101.

29 Ramamohan Paturi, Pavel Pudlák, and Francis Zane. Satisfiability coding lemma. *Chicago J. Theor. Comput. Sci.*, 1999, 1999. URL: http://cjtcs.cs.uchicago.edu/articles/1999/11/contents.html.

30 Uwe Schöning. A probabilistic algorithm for k-sat and constraint satisfaction problems. In *40th Annual Symposium on Foundations of Computer Science, FOCS '99, 17-18 October, 1999, New York, NY, USA*, pages 410–414, 1999. doi:10.1109/SFFCS.1999.814612.

31 Bart Selman, David G. Mitchell, and Hector J. Levesque. Generating hard satisfiability problems. *Artificial Intelligence*, 81(1):17–29, 1996. Frontiers in Problem Solving: Phase Transitions and Complexity. doi:10.1016/0004-3702(95)00045-3.

32 Greg Valiant. Faster random SAT. Personal communication.

33 Nikhil Vyas and Ryan Williams. On super strong ETH. In Mikolás Janota and Inês Lynce, editors, *Theory and Applications of Satisfiability Testing - SAT 2019 - 22nd International Conference, SAT 2019, Lisbon, Portugal, July 9-12, 2019, Proceedings*, volume 11628 of *Lecture Notes in Computer Science*, pages 406–423. Springer, 2019. doi:10.1007/978-3-030-24258-9_28.