Scientific Journal of Faculty of Education, Misurata University-Libya, Vol. 1, No. 7, Mar. 2017

Published on Web 01/3/2017

المجلة العلمية لكلية التربية, جامعة مصراتة, ليبيا, المجلد الاول ـ العدد السابع, يوليو 2017م

Why does Look-ahead work?

Harold Connamacher[*] and Abubaker Abdallah Alguttar[**]

Abstract:

This paper examines the behavior of the look-ahead heuristic on random 3-SAT. A well-known theorem is that DPLL with unit clause propagation requires exponential time to decide a uniformly random instance of 3-SAT that has a clause density near the satisfiability threshold. However, DPLL-based SAT-solvers are using look-ahead to find solutions on instances with thousands of variables, even at the satisfiability threshold. Slightly away from the threshold, these look-ahead SAT solvers are finding solutions to problems with millions of variables. This paper experimentally demonstrates that look-ahead is exploiting a structure that appears in (relatively) small instances of random 3-SAT.

Introduction

A well-known property of random 3-SAT, and random *k*-SAT in general, is that there exists a phase transition in the clause to variable density. If the clause density is below what is known as the satisfiability threshold, the random formula has a satisfiability assignment, with high probability (w.h.p), but if the density is above the satisfiability threshold the formula w.h.p unsatisfiability. Let Ω_n be a sample space of all problem instances on n variables, let ε_n be a property, and consider the probability of ε_n in the limit as n approaches infinity. We say ε_n occurs *with high probability* (w.h.p.) if $\lim_{n\to\infty} \Pr[\varepsilon_n] = 1$. An equally well-known and rigorously proven property of random 3-SAT is that simple variation of DPLL will require exponential

---

[*] Case Western Reserve University, Cleveland, Ohio, United States.

[**] Higher Institute for Science and Technology, Misurata, Libya.

time, with uniform positive probability (w.u.p.)[*], to decide an uniformly random formula with a clause density close to the threshold. On the other hand, state-of-the-art DPLL-based SAT solvers are using technique called *look-ahead*, and at densities close to the satisfiability threshold, these look-ahead algorithms are able to solve uniformly random formulas with thousands of variables. If we consider formulas with densities slightly away from the satisfiability threshold, DPLL solves using look-ahead are solving random 3-SAT problems with millions of variables [1].

This paper explores the success of look-ahead on random SAT. Look-ahead is sufficiently different from the DPLL variation about which we are have theoretical results, and so the theorems are not valid for look-ahead variants. However, we argue that the intuition behind the DPLL theorems should still apply to look-ahead, and in particular DPLL with look-ahead should require exponential time to find a solution for a SAT formula near the threshold. Our examination of the theory will also reveal a structure that exists in small instances of random SAT, and we present evidence that look-ahead is exploiting this structure to run in polynomial time on these small instances.

We note that the success of look-ahead for DPLL based SAT solvers is currently limited to satisfiable formula. Verifying the unsatisfiability of uniformly random formula with just a few hundred variables is not of reach for today's state of the art solvers [2].

DPLL [3] is an algorithm framework that forms the basis for most complete SAT solvers. DPLL iteratively builds an assignment to the SAT formula by, at each step, choosing an unassigned variable and choosing a value for the variable. After each variable assignment, DPLL modifies the SAT instance. If the variable assignment makes literal x true, then any clause containing x is now satisfied, and that clause is removed from the instance. Any clause containing x must be satisfied by a different literal in the clause,

---

[*] We say $\mathcal{E}_n$ occurs *with uniform positive probability* (w.u.p.p.) if there exists a positive constant c, independent of n, such that $\liminf_{n\to\infty} \Pr[\mathcal{E}_n] = c$.

and that clause is reduced by removing $\bar{x}$ from the clause. The result of this modification is a residual formula. If at any iteration of DPLL, the residual formula contains an empty clause, then all of the literals that were in the original, unreduced clauses are set to false by DPLL. In this case, we have a conflict in the partial assignment, and DPLL must backtrack. With DPLL, we supplement the algorithm framework with heuristics for choosing the next variable to assign, the value to assign a variable, and how to backtrack once a conflict is discovered. One common heuristics used with DPLL is unit clause propagation. If a clause containing a single literal is ever formed, that literal must be true for the clause to be satisfied, and we gain no benefit if we delay satisfying this clause propagation, we repeatedly assign variables from unit clauses remaining in the residual formula.

Look-ahead is a heuristic used to determine the next variable and value assignment for DPLL. Like DPLL, look-ahead is an algorithm framework in which there exist many variations. The basic form of the look-ahead strategy [4] is to take each literal in the current residual formula and see what would happen if we made that literal true. In one round of look-ahead, we compute, for each literal in the formula. The residual formula the results from setting the literal true and applying unit clause propagation, and we give that residual formula a score based on some metric. The literal assignment that produces the residual formula with the best score is chosen as the next variable-value pair assigned by DPLL. Common enhancements to look-ahead include identifying forced literals (also called *frozen variables*) and learning clauses. A forced literal is a literal that must be true in all satisfying assignments. Look-ahead identifies these literals by identifying when look-ahead on literal x produces a conflict. If such an event occurs $\bar{x}$ must be true, and we say that the variable of literal x is frozen. In clause learning, If look-ahead on literal x forces literal y to be true then we can add the clause $(\bar{x},y)$ to be formula.

Theoretical Background on DPLL

Much of the theoretical work on random SAT was inspired by a number of experimental studies on the behavior of DPLL; see e.g. [5,6,7,8,9]. These studies ran DPLL on random 3-SAT formulas, and the results suggested both an exact, sharp threshold in the satisfiability of random formulas and that random formulas with clause density close to the threshold require the most time to decide. Later Friedgut [10] proved a sharp threshold exists in *k*-SAT satisfiability, but neither the location of the threshold nor whether the threshold is constant for all problem sizes is known. More recently, Ding, Sly, and Sun [11] gave the exact threshold value for large *k*, but the location for the 3-SAT threshold is still open.

For the rest of the paper, we will let *n* be the number of variables in the SAT formula and *cn* the number of clauses. To bound the running time of DPLL, we separately consider satisfiable and unsatisfiable instances. A well known observation, first made in Galil [12], is that the running time of DPLL on an unsatisfiable instance of CNF-SAT is lower bounded by the length of the shortest resolution proof of unsatisfiability (the resolution complexity). Chvátal and Szemerédi [13] proved that a random instance of *k*-SAT, $k \geq 3$, with a linear number of clauses has exponential resolution complexity (w.h.p.).

To analyze the behavior on satisfiable instances, we note that until the first contradiction is discovered, the behavior of DPLL is completely determined by the heuristic DPLL uses to choose the next variable and value to assign. Two well-studied heuristics for DPLL on random SAT are denoted Unit Clause (UC) and Generalized Unit Clause (GUC). UC is a variation of unit clause propagation where, if no unit clause exists, the next variable and its value is chosen uniformly randomly. GUC denote a variation of the generalized unit clause or shortest clause heuristics where a clause and a variable in the clause are chosen uniformly randomly from the shortest clauses in the formula and the variable is assigned the value that satisfies the clause.

We will use DPLL+UC and DPLL+GUC to denote the DPLL variations that use these heuristics. In two early studies, Chao and Franco [14] found that if $c < \frac{8}{3}$, then UC finds a satisfying assignment (w.u.p.p.), and Frieze and Suen [15] found that if $c < 3.003\ldots.$, GUC finds a satisfying assignment (w.u.p.p.). UC and GUC are from a class of algorithms called *myopic* algorithms in [16]. The key property of myopic algorithms that permit precise mathematical analysis is that these heuristics only examine local information and the residual formula produced after each application of the heuristic is still uniformly random, conditional on the number of clauses of each size.

The techniques of [14] and [15], simplified in Achlioptas [17], let us trace the behavior of DPLL+UC and DPLL+GUC, until the first backtrack occurs, through the space of random SAT instances with a mixture of clause sizes (w.h.p.). If the trajectory stays in the region of the random formula space where the 2-clause density is below 1, then the unit clauses can be handled as soon as they appear (w.u.p.p.). If the trajectory ever leaves this region, the unit clauses will accumulate, increasing the probability of both (x) and ($\bar{x}$) appearing together as unit clauses. We use (2+p)-SAT [18] to denote a SAT instance with a mixture of 2- and 3-clauses. This model allows us to ignore the unit clauses in the trajectory analysis.

Using this trajectory technique, Achlioptas, Beame and Molloy [19] provided the proof that DPLL requires exponential time (w.u.p.p.) to solve satisfiable instances that have a clause density below the conjectured satisfiability threshold. [19] Proves that a uniformly random instance with (1-ϵ)n 2-clauses and a rn 3-clauses has exponential resolution complexity, for any positive constant r. If we let r be the least value such that a uniformly random (2+p)-SAT instance with (1-ϵ) n 2-clauses and rn 3-clauses does not have a solution (w.h.p.),

[19] Traces the behavior of DPLL+UC and DPLL+GUC backward to find the original random 3-SAT clause density that will produce this residual formula. The current best bound for r is (1+ϵ) from Achlioptas and Menchaca-Mendez [20]. Therefore, DPLL+UC takes exponential time to find a satisfying assignment for a 3-SAT formula with n variables, cn clauses with

$c \geq 2.71$ (w.u.p.p.). DPLL+GUC takes exponential time if $c \geq 3.1$. In both cases, the DPLL execution will produce a uniformly random residual formula with $(1-\epsilon)n'$ 2-clause and $(1+\epsilon)n'$ 3-clauses where $n' = \Theta(n)$ (w.u.p.p.), this residual formula is unsatisfiable (w.h.p.), and DPLL+UC will require exponential time to recognize this and backtrack out (w.h.p.).

The conjecture [21,22] is that $r=\frac{2}{3}$ is the correct threshold separating satisfiable random $(2+p)$-SAT instances from unsatisfiable instances. If true, this will make the bounds of [14] and [15] the threshold between polynomial and exponential running time for DPLL+UC and DPLL+GUC.

Comparing Look-Ahead with the Theory

The techniques we use in the results above require that the residual formula produced in each round of DPLL is uniformly random. Because of this restriction, our proofs are currently limited to DPLL variants in which, at each step of DPLL, we do not expose much of the formula. We do not have this limited exposure with look-ahead because a single round of the look-ahead procedure will expose all of the clauses of the formula.

Despite this limitation, we argue that above theory results still suggest that look-ahead variants of DPLL will need exponential time to solve random SAT instances. The theory states that DPLL+UC and DPLL+GUC makes poor initial choices in the assignments that leads DPL to an unsatisfiable $(2+p)$-SAT residual formula. These myopic heuristics are making decisions based on local information. For look-ahead, even though look-ahead is testing the result of assigning each variable in the formula, it is still only examining the local changes those assignments make. We argue below that for large n the unit clause propagation will only extend a constant distance from the initial literal. Long range correlations are not being found, and the current hypothesis is that long range correlations must be examined in order to avoid the unsatisfiable residual formula that the theory states will require an exponentially large resolution refutation (w.h.p.). This hypothesis about long range correlations is not verified for 3-SAT, but it is proven for k-SAT, $k \geq 8$ [23].

Our hypothesis is that an unsatisfiable (2+p)-SAT instance will have a short proof of unsatisfiability if n is small, and even n=1,000,000 is small. While look-ahead comes in many variants, we hypothesize that the primary reason for its success is that it is doing a branching process that is able to identify this short proof of unsatisfiability and prevent DPLL from exploring the unsatisfiable residual formula. We hypothesize that look-ahead will not demonstrate speedup over vanilla DPLL when n is large enough that unsatisfiable random (2+p)-SAT instances require exponential size refutations (w.h.p.).

There is a theoretical result that may explain the results we are seeing with look-ahead solvers. The seminal paper Bollobás, Borgs, Chayes, Kim, and Wilson [24] characterizes the satisfiability threshold for finite instances random 2-SAT by exploring how assignments to variables propagate through the clause structure of the formula. A well-known property of 2-SAT is that we can create a directed graph that has for each clause (x, y) of the 2-SAT formula, the directed edges $x \to y$ and $\bar{y} \to x$. We state that $v_1 \sim v_k$ if there exists a directed path $v_1 \to v_2$, $v_2 \to v_3, \ldots, v_{k-1} \to v_k$. If $x \sim \bar{x}$ then literal x must be *false* in any satisfying assignment to the formula, and the variable of x is frozen. A famous theorem is that a 2-SAT instance is unsatisfiable if and only there exists a conflict, a literal x with $x \sim \bar{x} \sim x$. Bollobás, et al. [24] track the growth in frozen variables, in the 2-SAT formula as the density of the formula crosses the satisfiability threshold. Actually examines the spine, The spine is the number of literals where if we add the literal as a unit clause to some satisfiable sub formula, the result is unsatisfiable. If the 2-SAT formula is satisfiable, the size of the spine is exactly the number of frozen variables. To do so, they use a branching process to calculate the number of literals forced by a single literal. The *forced component* of literal x is the set $\{y | x \sim y\}$. [24] divides the satisfiability threshold into three regions. Given a 2-SAT instance with n variables, if we have $(1-\epsilon)n$ 2-clauses, for any $\epsilon>0$, we are below the satisfiability threshold, there are $\Theta(1)$$ frozen variables, and the forced components for each literal have size $\Theta(1)$ (w.h.p.). If we increase the number of 2-clauses to $(1 - \epsilon_0 n^{-1/3})n$, we will be inside the satisfiability threshold, and here the probability that the formula is satisfiable is bounded

away from both 0 and 1. Initially, if we have just enough clauses to enter the satisfiability threshold region, there will be $\Omega(1)$ literals that have forced components with size $\Theta(n^{2/3})$ (w.u.p.p.). As we add additional 2-clauses, adding $\Theta(n^{2/3})$ clauses at a time, the probability of satisfiability will transition from 1 to 0. At first, the additional $\Theta(n^{2/3})$ 2-clauses will connect different forced components into a few *giant components*. If we then add $\Theta(n^{2/3})$ more 2-clauses, w.u.p.p. we will add some clause $(\bar{u}, \bar{v})$ where u and v are in the same

giant component. Adding this clause causes all the common parents of u and v in the component to be frozen, and w.u.p.p. this will introduce $\Theta(n^{2/3})$ frozen variables. When we have $(1+\epsilon)n$ 2-clauses, we are above the satisfiability threshold, there are be $\Theta(n)$ frozen variables and there will be a conflict (w.h.p.).

In its basic form, look-ahead sets a literal true and propagates that assignment. If we consider look-ahead on 2-SAT, this behavior is exactly modeled by the branching process technique used by [24]. The branching process of [24] to compute the forced component of a literal in random 2-SAT is as follows. Initially, the forced component consists of only the literal, and the literal is in the *frontier* of the component. At each round, we choose a literal x from the frontier, and for each pair consisting of x and any literal y not in the forced component, we flip a coin and with probability p_2, $(\bar{x}, y)$ is a clause of the formula and y is added to the forced component. Once all such pairs xy have been explored, x is removed from the frontier though it stays in the forced component. The process stops when the frontier becomes empty. If the 2-SAT formula has c_2n clauses, then

$$p_2 = \frac{c_2 n}{4\binom{n}{2}} \approx \frac{c_2}{2n}.$$

The number of literals added to the frontier at round t of the process, also known as the *birthrate* of the process, is $(2_n - s(t))p_2$, where $s(t)$ is the size of the forced component at round t, and until s grows sufficiently large, the birthrate is approximately c_2.

We can expand this process for formulas that also contain c_3n 3-clauses. Besides testing each pair xy, we test each triple axy where a is a

non-frontier member of the forced component. With probability p_3, $(\bar{a}, \bar{x}, y)$ is a 3-clause of the formula, and y is added to the frontier. If the formula has c_3n 3-clauses, $p_3 = \frac{c_3 n}{8\binom{n}{3}} \approx \frac{3c_3}{4n^2}$. On round t of the branching process, the non-frontier part of the forced component will have size t-1, and the probability that a literal y is added to the frontier is $p_2 + (t\text{-}1)p_3 - O(tn^{-3})$. The birthrate of the process is $(2n\text{-}s(t))(p_2 + (t\text{-}1)p_3 - O(tn^{-3}))$, and until s grows sufficiently large this is approximately

$$c_2 + \frac{3(t-1)}{2n} c_3.$$

To model look-ahead on a literal as a branching process with birthrate (1), we require that the residual formula is uniformly random. Consider a situation where we run DPLL using a myopic heuristic, stopping when we the 2-clause density is 1-ϵ for any positive ϵ>0, and then apply look-ahead. If we stopped DPLL after r iterations, the residual formula will have n-r variables, (1-ϵ)(n-r) 2-clauses and $c_3(r)$(n-r) 3-clauses for some constant $c_3(r)$. If run the branching process at this point, (1) becomes $1\text{-}\epsilon + \frac{3(t-1)}{2(n-r)} c_3(r)$. In the limit as n tends to infinity, the birthrate is 1-ϵ. From [24], the branching process will terminate after a constant number of steps, will not find a conflict, and will find only a constant number of frozen variables (w.h.p.). However, if n is small enough and if the constant sized branching process lasts for a large enough constant t such that $\frac{3}{2} c_3 (r) > \frac{\epsilon(n-r)}{t-1}$, then the process will be effectively replaced by a new process with birthrate $1+\delta$. In [24], such a process will produce a conflict (w.h.p.).

If instead, $\frac{3}{2} c_3 (r) > \frac{\epsilon(n-r) - \epsilon_0 (n-r)^{-1/3}}{t-1}$ then,

the birthrate will be similar to the branching process of [24] where the process does not explode, but forced components grow large enough to discover $\Theta(n^{2/3})$ frozen variables, and assigning these frozen variables will allow us to learn $\Theta(n^{2/3})$ new 2-clauses. As detailed in [24], adding $\Theta(n^{2/3})$ random 2-clauses will be enough to generate a conflict.

The Experimental Evidence

To test our hypothesis, we generated random (2+p)-SAT formulas around the (2+p)-SAT threshold, and we looked for small proofs of unsatisfiability (i.e. conflicts) that could be discovered using a branching process.

We use a∼b to denote the property that literal a forces literal b. Consider the following four rules.

- (1) if (x, y) is a clause and $a \sim \bar{x}$ then $a \sim y$.
- (2a) if (x, y, z) is a clause and $a \sim \bar{x}$ and $a \sim \bar{y}$ then $a \sim z$.
- (2b) if (x, y, z) is a clause and $y \sim b$ and z$\sim b$ then $\bar{x} \sim b$.
- (2c) if (x, y, z) is a clause and $x \sim \bar{x}$ and $\bar{y} \sim z$ then $\bar{z} \sim y$.

To determine all the structural information that can be learned from a branching process, for each instance we computed a transitive closure using various combinations of these rules. The transitive closure was calculated using rule (1) alone (to identify instances where the 2-clauses alone were unsatisfiable), using rules (1) and (2a) (to simulate running standard look-ahead with unit clause propagation from each literal in the formula), using rules (1), (2a), and (2c) (to simulate look-ahead plus clause learning), and using all four rules (to simulate the maximum a branching process could discover). We include rule (2b) because (2b) is the contrapositive of (2a), and it is possible to construct conflict examples that require rule (2b) to discover. If a conflict was found in the transitive closure, that conflict was the proof of unsatisfiability, but for instances where applying all four rules could not find a conflict, we started with a partial assignment made up of the frozen literals discovered in the transitive closure search and we used DPLL with generalized unit clause and restarts to search for a satisfying assignment.

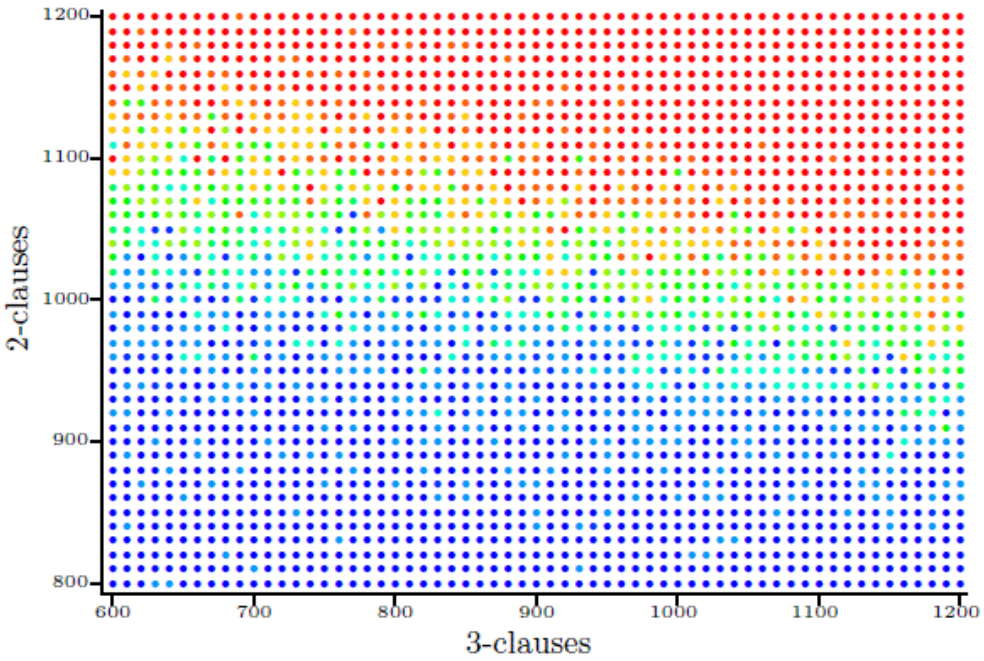
For the experiment, we generated random (2+p)-SAT instances on 1000 variables with clause densities around the conjectured (2+p) satisfiability threshold. These samples will roughly correspond to the residual formula produced by running DPLL on an 3-SAT instance with between 1100 and 1500 variables. While these instances are several orders of

magnitude smaller than the satisfiable instances SAT solvers handle in practice, we chose 1000 variables because we were testing a large number of instances on standard Apple and Linux desktop machines, and we wanted the experiment to complete in a few weeks. It was important for our study that we decide each of the random instances because the most interesting cases would be those that challenged our hypothesis: formulas that were unsatisfiable but for which there was no conflict in the transitive closure. Significantly larger instances were not practical given these computational limitations. However, 1000 variables is not a trivial study. We note that at the recent International SAT Competition, no SAT solver was able to solve all of the satisfiable random 3-SAT instances with fewer than 10,000 variables [1], and we note that 1000 variables is an order of magnitude larger than the largest unsatisfiable instances modern solvers can prove unsatisfiable [2]. The program was written in C and the random formulas were generated using the *random()* function of *stdlib.h*.

We generated instances that had between 800 and 1200 2-clauses and between 600 and 1200 3-clauses. We used a large range of clause densities in order to test instances on both sides of the conjectured (2+*p*)-SAT threshold, since we are dealing with finite instances and 2-SAT has a fairly wide scaling window [24], we do not expect the point where there is 50% satisfiability to fall right at the conjectured threshold. Figure below plots the satisfiable and unsatisfiable instances and demonstrates that our tests did capture both sides of the threshold.

If our hypothesis is correct, with 1000 variables, we should be able to find conflicts using the transitive closure in most of the unsatisfiable instances. In addition, we should find unsatisfiable instances where rules (1) and (2a) would be sufficient to identify the conflict, and unsatisfiable instances where we must use rules (1), (2a), and (2c). This is exactly what we did find. The results of the experiment are as follows. Of the 25,010 instances in the experiment, 11,525 had conflicts found by the computing the transitive closure using all four rules. Of those, 4,407 had a conflict in the 2-clauses alone while the other 7,118 unsatisfiable instances did not. Most importantly,

in each of the 13,485 instances that did not have a conflict in the transitive closure, the instance was satisfiable.



Using rules (1), (2a), and (2c), we found a conflict in all but two unsatisfiable instances. Using only rules (1) and (2a) failed to find conflicts in 1,308 unsatisfiable instances. The next questions for the research are: for what size *n* can we no longer find conflicts using a branching process, how does look-ahead perform on these sizes, and can we further exploit this structure to improve our algorithms on ”small” random formula?

The Figure Conflicts in (2+*p*)-SAT. Each dot corresponds to 10 instances of (2+*p*)-SAT on 1000 variables. The horizontal axis is the number of 3-clauses and the vertical axis is the number of 2-clauses. The hue of the dot indicates the percentage of instances that have conflicts. blue = 0%, green = 50%, red = 100%.

References

[1] SAT Competition 2014 -Sequential, Random SAT track Experiment, May 2014. url: satcompetition.org/edacc/sc14/experiment/24/.

[2] A. Balint, A. Belov, M. J. H. Heule, and M. Jarvisalo. Generating the uniform random benchmarks for SAT competition 2013. In *Proceedings of SAT Competition 2013: Solver and Benchmark Descriptions*, pages 97-98, 2013.

[3] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Communications of the ACM*, 5(7):394-397, 1962.

[4] M. J. H. Heule and H. van Maaren. Look-ahead based SAT solvers. In A. Biere, M. Heule, H. van Maaren, and T.Walsh, editors, *Handbook of Satisfiability, volume 185 of Frontiers in Artificial Intelligence and Applications*, pages 155-184. IOS Press, 2009.

[5] P. Cheeseman, B. Kanefsky, and W. Taylor. Where the really hard problems are. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence -Volume 1*, IJCAI'91, pages 331-337, 1991.

[6] S. Kirkpatrick and B. Selman. Critical behavior in the satisfiability of random boolean expressions. *Science*, 264(5163):1297-1232, 1994.

[7] B. Selman, D. G. Mitchell, and H. J. Levesque. Generating hard satisfiability problems. *Artificial Intelligence*, 81(1-2):17-29, 1996.

[8] J. M. Crawford and L. D. Auton. Experimental results on the crossover point in random 3-SAT. *Artificial Intelligence*, 81(1-2):31-57, 1996.

[9] S. Cocco and R. Monasson. Trajectories in phase diagrams, growth processes, and computational complexity: How search algorithms solve the 3-satisfiability problem. *Physical Review Letters*, 86(8):1654-1657, 2001.

[10] E. Friedgut. Sharp thresholds of graph properties, and the *k*-SAT problem. *Journal of the American Mathematical Society*, 12(4):1017-1054, 1999.

[11] J. Ding, A. Sly, and N. Sun. Proof of the satisfiability conjecture for large k. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing*, STOC '15, pages 59-68, 2015.

[12] Z. Galil. On the complexity of regular resolution and the Davis-Putnam procedure. *Theoretical Computer Science*, 4(1):23-46, 1977.

[13] V. Chvatal and E. Szemeredi. Many hard examples for resolution. *Journal of the ACM*, 35(4):759-768, 1988.

[14] M.-T. Chao and J. Franco. Probabilistic analysis of a generalization of the unit clause literal selection heuristic for the *k*-satisfiability problem. *Information Science*, 51(3):289-314, 1990.

[15] A. Frieze and S. Suen. Analysis of two simple heuristics on a random instance of *k*-SAT. *Journal of Algorithms*, 20(2):312-355, 1996.

[16] D. Achlioptas and G. B. Sorkin. Optimal myopic algorithms for random 3-sat. In *Proceedings of the 41th Annual IEEE Symposium on Foundations of Computer Science*, pages 590-600, 2000.

[17] D. Achlioptas. A survey of lower bounds for random 3-SAT via differential equations. *Theoretical Computer Science*, 256(1-2):159-185, 2001.

[18] R. Monasson, R. Zecchina, S. Kirkpatrick, B. Selman, and L. Troyansky. Phase transitions and search cost in the 2+*p*-SAT problem. In *4th Workshop on Physics and Computation*, 1996.

[19] D. Achlioptas, P. Beame, and M. Molloy. Exponential bounds for DPLL below the satisfiability threshold. In *Proceedings of the Fifteenth Annual* ACM-SIAM *Symposium on Discrete Algorithms*, pages 132-133, 2004.

[20] D. Achlioptas and R. Menchaca-Mendez. Exponential lower bounds for DPLL algorithms on satisfiable random 3-SAT formulas. In *Theory and Applications of Satisfiability Testing - SAT 2012*, Lecture Notes in Computer Science, pages 327-340. Springer, 2012.

[21] D. Achlioptas, L. M. Kirousis, E. Kranakis, and D. Krizanc. Rigorous results for random (2+p)-SAT. *Theoretical Computer Science*, 265(1-2):109-129, 2001.

[22] G. Birioli, R. Monasson, and M. Weight. A variational description of the ground state structure in random satisfiability problems. *European Physical Journal* B, 14:551-568, 2000.

[23] D. Achlioptas and F. Ricci-Tersenghi. On the solution-space geometry of random constraint satisfaction problems. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing*, pages 130-139, 2006.

[24] B. Bollobas, C. Borgs, J. T. Chayes, J. H. Kim, and D. B. Wilson. The scaling window of the 2-SAT transition. *Random Structures and Algorithms*, 18(3):201-256, 2001.