On SAT Distributions with Planted Assignments (Extended Abstract)

Tassos Dimitriou
Athens Information Technology, Greece.
tassos@ait.gr

Abstract

While it is known how to generate satisfiable instances by reducing certain computational problems to SAT, it is not known how a similar generator can be developed *directly* for k-SAT. In this work we almost answer this question affirmatively by improving upon previous results in many ways. First, we give a generator for instances of MAX k-SAT, the version of k-SAT where one wants to *maximize* the number of satisfied clauses. Second, we provide a useful characterization of the optimal solution. In our model not only we know how the optimal solution looks like but we also prove it is *unique*. Finally, we show that our generator has certain useful computational properties among which is the ability to control the hardness of the generated instances, the appearance of an easy-hard-easy pattern in the search complexity for good assignments and a new type of phase transition which is related to the uniqueness of the optimal solution.

1 Introduction

Testing the satisfiability of a particular formula is a problem (SAT) that lies at the heart of many AI formalisms such as reasoning, planning, etc. The use of distributions for generating random SAT instances is therefore an important set of benchmark problems for evaluating local search SAT heuristics. However, as was shown by Mitchell and Levesque[1996], the value of any study whose goal is to evaluate the performance of any SAT algorithm depends upon the proper selection of formula distribution and parameter values. By being more careful in generating problem instances one can obtain really hard search and reasoning problems. And it is exactly this controlled hardness that ultimately led to the development of powerful stochastic search methods for constraint satisfaction and satisfiability problems (see [Selman *et al.*, 1993; Morris, 1993; Li and Anbulagan, 1997], to name a few).

The key property of such "useful" distributions is that they generate instances that appear to be *critically constrained*; at a certain ratio of variables to constraints instances become extremely hard to solve and the average computational cost of finding a solution scales exponentially with the size of the input formula. Around this critical region however, instances are much easier to handle mainly because in the "underconstrained" region solutions are numerous while in the "overconstrained" region solutions are nonexistent. At the phase transition, where approximately half the instances are satisfiable, one finds the most difficult to solve problem instances.

What limits the applicability of these distributions is the fact that they generate both satisfiable and unsatisfiable hard instances. Since the unsatisfiable instances must be filtered out with the use of *complete* methods before they can be used in the evaluation of any incomplete SAT heuristic, the size of the problem instances considered this way unfortunately becomes limited. This drawback was remedied in part by the work of Achlioptas *et al.*[2000] where a generator for *satisfiable instances only* was developed.

This generator was based on transforming an instance of the quasigroup completion problem (QCP) to a formula that is guaranteed to be satisfiable. The generator starts with a complete Latin square of order N, that is an $N \times N$ table where each entry has one of the N possible colors and where there are no repeated colors in any row or column. Then the colors from a fraction p of the entries get deleted leaving a partial table that is guaranteed to be satisfiable.

As was demonstrated by Achlioptas *et al.*, this generator has a number of important characteristics. The first one is the ability to finely control the hardness of the generated instances by tuning the value of p. The second one is the appearance of a new kind of phase transition in the space of problem instances; under the right parameterization this transition coincides with the hardest region of the *satisfiable* instances and corresponds to a threshold phenomenon in the size of the *backbone*, a quantity which measures the number of the variables which take the same value in all solutions[Monasson *et al.*, 1999; Slaney and Walsh, 2001]. It is interesting to note however that while the QCP problem leads nicely to a satisfiable instance generator with good computational properties, Achlioptas *et al.* ask whether a similar generator can be developed directly for k-SAT.

In this work we almost answer this question affirmatively by introducing a generator for MAX k-WSAT formulas, a weighted version of k-SAT. In MAX k-WSAT each clause has a number associated to it, called *weight* or *multiplicity*, which denotes how many copies of the clause appear in the formula. While in MAX k-SAT one is looking for an assignment that maximizes the number of satisfied clauses, in the weighted version of MAX k-SAT the goal is to find an assignment that essentially maximizes the *sum of weights* of

the satisfied clauses, since such clauses contribute their multiplicities to the overall number of satisfied clauses. Clearly MAX k-SAT reduces to this problem by making all weights equal to one.

Our generator has a number of important characteristics. The first one is a theoretical result that provides a unique characterization of the optimal assignment. Since any satisfiability heuristic when fed with an instance from our generator will try to maximize the number/weight of satisfied clauses, this characterization provides algorithm designers with an *a priori* knowledge of the optimal assignment. We call this solution the *hidden* or *planted* assignment. Thus by knowing what to expect, algorithm designers will be able to evaluate the effectiveness of their algorithms.

The second characteristic is the appearance of an easy-hard-easy pattern in the search complexity for good assignments. Traditional phase phenomena usually involve a transition from satisfiable to unsatisfiable instances in the search space. This is not the case here since our generator outputs only instances that can be satisfied in the MAX k-SAT sense. Under the right choice of parameters however, an easy-hard-easy pattern emerges that makes it possible to test algorithms on hard generated instances only.

Finally, we were able to link this behavior with a new threshold phenomenon which is related to the uniqueness of the hidden assignment. Below the threshold, there are other solutions that achieve equal total weight and differ from the hidden one in a few variables. Above the threshold however, the hidden assignment becomes the unique optimal solution. Thus there exists a transition from a phase where there are more than one good assignments to a phase where the optimal assignment is unique. The point to be made is that this transition coincides with the hardest to solve problem instances.

2 The Model

We start our exposition by showing how to generate instances of the MAX 2-WSAT problem. Later in Section 4, we will extend our results to instances of MAX k-WSAT. In general, MAX k-WSAT consists of Boolean expressions in Conjuctive Normal Form, i.e. collection of clauses in which every clause consists of exactly k literals and has a positive integer weight associated to it denoting the multiplicity of each clause in the formula. Given an instance of this problem, one is looking for an assignment to the variables that satisfies a set of clauses with maximum total weight.

It is clear that MAX 2-WSAT is NP-hard as MAX 2-SAT reduces to it by setting all weights equal to one. In this work we will present a generator for instances for a degenerate version of MAX 2-WSAT, in which *all* weights to the clauses are either β or $\beta+1$, where β is a fixed integer greater than 0. While this simplification may seem very restrictive at first look, it is all we need to create a generator of k-SAT instances with useful computational properties.

To generate a formula with the above properties we first start with $2n$ variables, n green and n blue, create the clauses and finally assign weights to them. Here we adopt the view of working with weights directly and not actually creating multiple instances of the same clause as proofs become simpler. Furthermore, as explained in Section 5, this leads to faster implementations of heuristics treating WSAT formulas.

We call our model $\mathcal{F}_{n,\delta}$, where n indicates the number of variables of each color and δ is a parameter used to control the maximum total weight achieved by the hidden assignment. We do not include the weight β in the definition of the model since this will be set to a specific value later on.

The model $\mathcal{F}_{n,\delta}$ (with super-clauses)

1. Start with $2n$ variables, n green and n blue.
2. **(Create the formula)** For every pair of variables x, y, irrespective of their color and without repetitions, add to the formula the "super-clause" $c(x,y) = (x\bar{y} + \bar{x}y)$.
3. **(Assign the weights)**
   - For all clauses $c(x,y)$, with probability $\frac{1}{2}$ set the weight $w(x,y)$ of the clause equal to $\beta+1$, otherwise set it equal to β.
   - For all clauses $c(x,y)$, such that x, y have *different* colors *and* $w(x,y) = \beta$, with probability 2δ increase the weight of the clause to $\beta+1$.

The careful reader should have observed by now that the "clauses" $c(x,y)$ are not really clauses in the ordinary 2-SAT sense. In fact, $c(x,y) = (x+y)\cdot(\bar{x}+\bar{y})$. We chose, however, to work with super-clauses as the results are much easier to describe and the passing to ordinary 2-SAT expressions is again easy. We will denote the two simple clauses of $c(x,y)$ by $c^1_{x,y} = (x+y)$ and $c^2_{x,y} = (\bar{x}+\bar{y})$.

It is also clear from the model that the generated formulas are "dense" in that they consist of all possible combinations of the $2n$ variables. Thus it makes no sense to try to satisfy all super-clauses but it makes sense to try to satisfy a suitable subset of those that incures the maximum possible total weight. We will be able to show later on that the best assignment is the one that has the green variables set to true and the blue set to false (or vice versa). However, before we proceed with our main result we need a few definitions and preliminary lemmas.

Definition 1 *An assignment is said to split the variables if exactly n variables are set to true and n are set to false (irrespective of their color).*

Lemma 1 (Monochromatic clauses are lighter) *If x, y have the same color then*

$$w(x,y) = \begin{cases} \beta+1, & \textit{with probability } \frac{1}{2} \\ \beta, & \textit{otherwise} \end{cases}$$

If x, y have different colors then

$$w(x,y) = \begin{cases} \beta+1, & \textit{with probability } \frac{1}{2}+\delta \\ \beta, & \textit{otherwise} \end{cases}$$

Proof: The first statement is obvious since by definition monochromatic clauses (clauses with variables of the same color) have weight β or $\beta+1$ with probability a half. To prove the second statement observe that a non-monochromatic clause will have weight $\beta+1$ if it was initially assigned this weight, or if it had weight β and with probability 2δ increased its weight. The probability of these two events is $\frac{1}{2} + \frac{1}{2}2\delta = \frac{1}{2} + \delta$. ◇

This lemma provides an alternative definition for our model and it will come handy when we prove our result about the optimality of the hidden assignment. The next lemma is used to reduce the space of good assignments. Since our goal is to be able to generate formulas where assignments are planted, this lemma allows algorithm designers to test their algorithms by knowing what to expect for.

Lemma 2 (Look for split assignments) *When the weight β is equal to n^2, the best assignments split their variables.*

Proof: (Sketch) Given any assignment we construct a bipartite graph that has on one side the true variables and on the other the false ones. Furthermore, for each pair of nodes on different sides, we create an edge and assign to it the weight of the corresponding super-clause.

Then we show that the particular choice of weights assigned to clauses makes the overall weight achieved by the *best* unevenly split assignment *less* than the weight achieved by *any* assignment with split variables. Thus it is always best to look for split assignments. ◇

Observe that the previous discussion is valid only if the super-clauses are satisfied as a whole or at least in the NAE-SAT sense (NAESAT for Not All Equal SAT, is the variant of SAT where we don't allow all literals in a clause to have the same truth value). To pass to ordinary 2-SAT models, since most algorithms are not restricted in their search for assignments, we modify the model by assigning the weight $w(x, y)$ to each of the clauses $c^1_{x,y}$ and $c^2_{x,y}$ of the super-clause. Call this new model $\mathcal{F}'_{n,\delta}$. Now, we have to take into account the weight incurred by these clauses even if both literals have the same truth value.

Lemma 3 (Equivalence of the two models) *An assignment A achieves total weight W for a formula f generated according to $\mathcal{F}_{n,\delta}$ if and only if it achieves total weight $W + c_f$ when the formula is generated according to $\mathcal{F}'_{n,\delta}$, where c_f in a constant that is easily computable and depends only on the particular formula f.*

Proof: The proof is very similar to the proof of Lemma 2 and is omitted from this extended abstract. ◇

An immediate corollary of this lemma is that again we only have to look for split assignments in the new model.

Corollary 1 *By choosing $\beta = n^2$, the best assignments for formulas generated according to $\mathcal{F}'_{n,\delta}$ again split their variables.*

3 Characterizing the Optimal Assignment

In the previous section we showed that the two models are equivalent. Thus from now on we will work only with formulas that consist of super-clauses. To simplify things further we will work only with split assignments since by Lemma 2 we are allowed to do so.

Our goal in this section is to show that for a suitable choice of the parameter δ, the optimal assignment is one that has the green variables set to True and the blue variables set to False (or vice versa).

Definition 2 *We say an assignment has distance k from the optimal one, where $0 \leq k \leq \frac{n}{2}$, if it has split the variables and furthermore it has k blue and $n - k$ green variables set to True.*

Thus in some sense the value of k counts the distance from the planted assignment which has $k = 0$ and as we will show in a while it is the optimal one with high probability.

Theorem 1 (Optimality of hidden assignment) *There is a constant such that for values of $\delta \geq \Omega(\sqrt{\ln n/n})$, the assignment which has only the green variables set to true is optimal with high probability.*

Proof: We only give a sketch of the proof here since this result is provided only for completeness. The first lemma we need is one which shows that assignments of distance k from the hidden one achieve total weight close to their expected values. This is easy to prove since by using Chernoff bounds we can estimate with great accuracy the total weight achieved by the given assignment. An immediate corollary of this result is that no assignment of distance greater than some predefined k_0 achieves better weight than the hidden one.

The second lemma we need is one which proves that any assignment of distance smaller than a predefined value k_1, has a neighboring assignment that achieves even better weight except of course the hidden assignment. This last result suggests that these assignments cannot be optimal. Combining the two lemmas, we get that the hidden assignment is optimal with high probability for the range of δ described in the theorem. ◇

4 Extension to MAX k-WSAT

To generate instances of MAX 3-WSAT or MAX k-WSAT, $k \geq 3$, we follow the same approach as for the 2-SAT case. We start again with $2n$ variables, n green and n blue. The only difference now is that clauses consist of exactly k variables. We call our model $\mathcal{F}^k_{n,\delta}$, where k indicates that we working with k-WSAT formulas.

The model $\mathcal{F}^k_{n,\delta}$ (with super-clauses)

1. Start with $2n$ variables, n green and n blue.
2. **(Create the formula)** For every k-tuple of variables $x_1, x_2, \ldots, x_k$, irrespective of their color and without repetitions, add to the formula the "super-clause"
$$c(x_1, x_2, \ldots, x_k) = \neg(x_1 x_2 \cdots x_k + \bar{x}_1 \bar{x}_2 \cdots \bar{x}_k).$$
3. **(Assign the weights)**
   - For all clauses $c(x_1, x_2, \ldots, x_k)$, with probability $\frac{1}{2}$ set the weight $w(x_1, x_2, \ldots, x_k)$ of the clause equal to $\beta + 1$, otherwise set it equal to β.
   - For all *non-monochromatic* clauses $c(x_1, x_2, \ldots, x_k)$, such that $w(x_1, x_2, \ldots, x_k) = \beta$, with probability 2δ increase the weight of the clause to $\beta + 1$.

As in the 2-WSAT case, the super-clauses are satisfied only when not all variables have the same truth value. To pass to ordinary k-SAT formulas observe that $c(x_1, x_2, \ldots, x_k) = (x_1 + x_2 + \cdots + x_k)(\bar{x}_1 + \bar{x}_2 + \cdots + \bar{x}_k)$.

We then have to modify the model by assigning the weight $w(x_1, x_2, \ldots, x_k)$ to each of the sub-clauses of the super-clause.

A lemma similar in spirit to Lemma 2 shows that again we have to concentrate our search for split assignments by setting $\beta = n^2$.

5 Locating the Hard Instances

In this section we present experimental results showing that random instances can be generated by our model in such a way that easy and hard instances can be predictable in advance. Our motivation is to provide developers of local search SAT heuristics with a challenging set of k-SAT instances in which the optimal solution is known beforehand.

The local search procedure we used for our tests is a modified version of WalkSat[Selman *et al.*, 1993] which we describe below. The main reason for choosing WalkSat is because it is one of the best performing SAT procedures and because we believe that these results on hard instances will be applicable to other SAT heuristics as well.

To apply WalkSat to formulas with weights on clauses (even if the weights degenerate to the two values β and $\beta+1$) we need the intuitive modification of the algorithm shown on Table 1. Basically what this table says is replace "number of satisfied clauses" with "weight of satisfied clauses". The rest of the algorithm remains the same. Also observe how the weighted version reduces to the classic WalkSat when all weights are set to one. The reason for this modification is to avoid the extra overhead in running time caused by having multiple copies of the same clause. Since each clause would have to appear at least $\beta = n^2$ times, this would greatly slow down the execution time of any SAT heuristic.

	WalkSat	**Weighted version of WalkSat**
Goal	Maximize the *number* of satisfied clauses	Maximize the *weight* of satisfied clauses
Strategy	Pick a random unsatisfied clause and flip the variable that results in the smallest decrease in the *number* of satisfied clauses	Pick a random unsatisfied clause and flip the variable that results in the smallest decrease in the *weight* of satisfied clauses

Table 1: Changes to the basic WalkSat algorithm.

In the experiments that follow we chose to work with MAX 2-WSAT formulas to illustrate the fact that these formulas become extremely difficult to optimize in direct contrast to ordinary 2-SAT formulas, which are solvable in linear time[Aspvall *et al.*, 1979].

In all the figures each sample point was computed after generating 1000 random instances of MAX 2-WSAT. Data represent the *median* of the *total number of variable flips* required to locate an assignment that achieves the maximum total weight. We used medians and not means in order to eliminate the fluctuations caused by a small number of instances incurring extremely high values.

In Figure 1 we considered the performance of WalkSat on such random formulas. This figure shows the total number

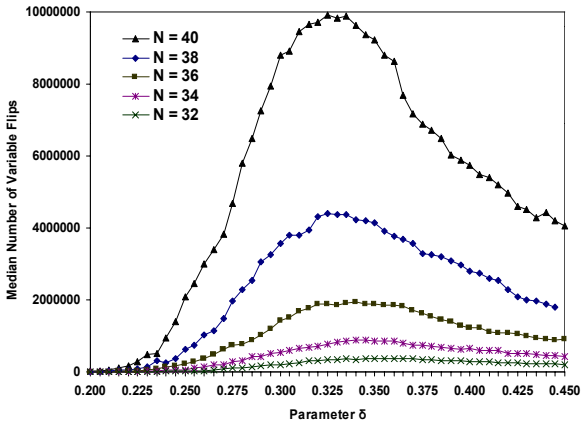

Figure 1: Median number of total variable flips for random 2-WSAT formulas as a function of the parameter δ.

of flips required by WalkSat to find an assignment that beats the weight of the hidden assignment for formulas of size $n = 32, 34, 36, 38$ and 40. As can be seen, an easy-hard-easy pattern emerges which results in an exponential increase in computational cost in the hardest region. This behavior is similar to the behavior of 3-SAT formulas [Mitchell *et al.*, 1996; Gent and Walsh, 1994]. This figure also suggests that the notions of *under-constrained* and *over-constrained* formulas apply here as well although we cannot link this behavior with the length of the formulas as all of them have the same number of clauses. But we will return to this issue in the next section where we try to relate this behavior with a phase transition in structural properties of the WSAT instances.

6 Phase Transition

An important characteristic of Figure 1 is that the transition region becomes narrower (occurs for a smaller range of δ) for larger values of n when at the same time the peak shifts to the left as n is increased. Our goal in this section is to demonstrate a relationship between the hard region and a phase transition in the structural properties of the generated formulas.

It is clear that we cannot have a SAT/UNSAT transition as all instances are unsatisfiable. Neither we can relate it to the so-called *backbone* variables[Monasson *et al.*, 1999; Slaney and Walsh, 2001]. The backbone ratio of a SAT problem is the ratio of its variables that take the same values in *all* solutions, i.e. they are *fully constrained*. A phase transition in such a case has the the backbone ratio drop from nearly 1 to nearly 0, with the hardest instances lying around the 50% point. In the case of WSAT formulas however, their solution is planted and provided that δ is sufficiently large, the solution is also unique. So we cannot relate the hardness peak with the backbone as there is essentially only one solution. However, we were able to relate this behavior with the probability of uniqueness of the hidden assignment.

In Figure 2 we show how the probability that there exist good assignments other than the planted one changes as a function of the parameter δ for a large range of values for n. Observe how the threshold function sharpens up for larger

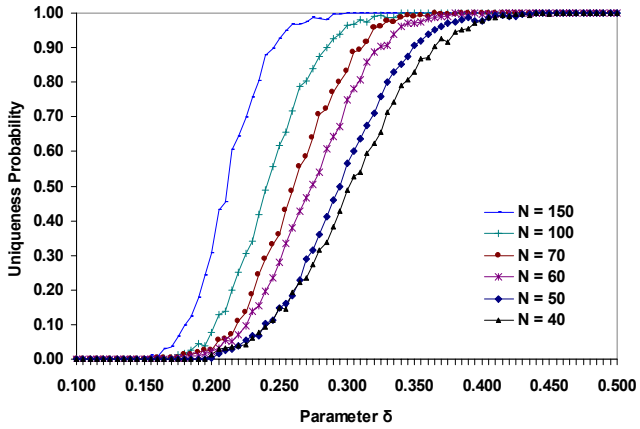

Figure 2: Phase transition for various values of n.

values of n, like the satisfiability threshold function for random k-SAT formulas[Mitchell *et al.*, 1996]. One difference however is that curves do not cross. Instead the curves are moving to the left, something that is to be expected since the hidden solution is with high probability unique for values of δ larger than $c\sqrt{\ln n/n}$, for some constant c.

All this discussion leads naturally to the question of how one can generate the hardest 2-WSAT formulas. Given some arbitrary value of n how can we determine the value of δ that results in the most difficult to solve instances? The answer is given by *finite-size scaling*[Kirkpatrick and Selman, 1994], in which the horizontal axis is rescaled by a quantity that is a function of n. This has the effect of slowing down the transition for larger values of n and mapping the different curves into a single "universal" curve from which one can derive by working backwards the point where the hardest instances lie.

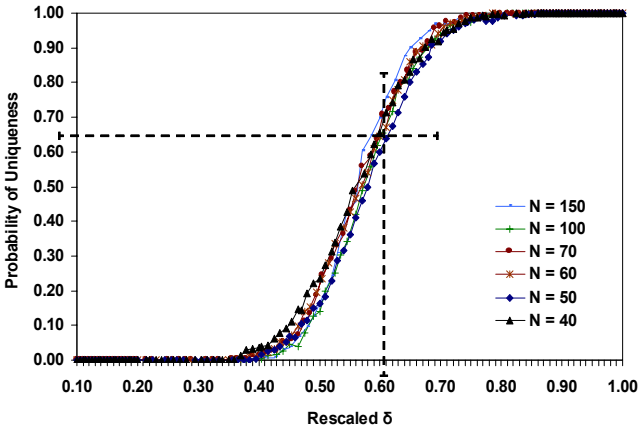

Figure 3: Phase transition for various values of n after rescaling.

Figure 3 shows the result of rescaling the curves of Figure 2. The uniqueness probability is plotted against δ', a rescaled

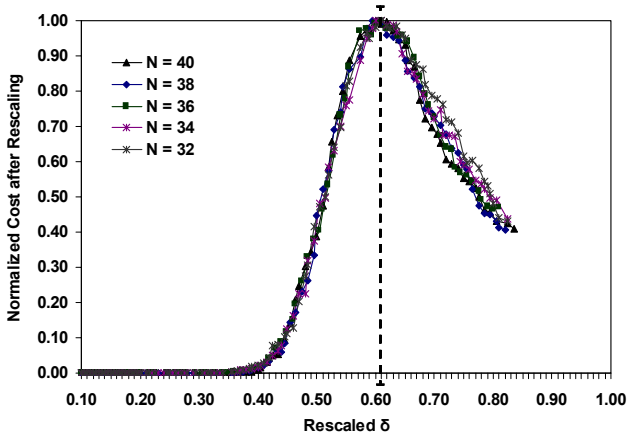

Figure 4: Computational cost for various values of n after rescaling.

version of δ equal to $\delta' = \delta n^{\epsilon/2}\sqrt{1-\epsilon}$, where $\epsilon = 0.56$. It is perhaps instructive at this point to describe how we derived the rescaling factor $n^{\epsilon/2}\sqrt{1-\epsilon}$.

Theorem 1 tells us that the planted solution is unique when $\delta = \Omega(\sqrt{\ln n/n})$. This led us to believe that the threshold point will also be a function of this quantity, something like $\delta_0 = c\sqrt{\ln n/n}$, for some (unknown) constant c. If n is to be rescaled and become $n^{1-\epsilon}$, then the translated point must become $\delta_0' = c\sqrt{\frac{\ln n^{1-\epsilon}}{n^{1-\epsilon}}}$. By some algebraic manipulation we see that δ_0 and δ_0' are related by the equation

$$\delta_0' = \delta_0 n^{\epsilon/2}\sqrt{1-\epsilon},$$

which, when applied to all values of δ, gives us the universal match shown in Figure 3.

Finally, Figure 4 demonstrates how the *computational cost* for various values of n collapses into a universal curve. To obtain these data we first normalized the curves shown in Figure 1 and then applied the rescaling described previously. We see clearly that the critical point is when the *rescaled* δ is equal to 0.60 which corresponds to the 65% uniqueness probability in Figure 3. Thus the main empirical observation we can draw from these pictures is that when $p = 1/2$, *the hardest 2-WSAT formulas for WALKSAT lie at the point where about 65% of them have the hidden assignment as the optimal one.*

7 Conclusions

In this work we presented a generator for instances of k-SAT in which every clause has a weight or multiplicity associated with it and the goal is to maximize the total number of satisfied clauses. We showed that our generator produces formulas whose hardness can be finely tuned by a parameter δ that controls the weights of the clauses. Under the right choice of this parameter an easy-hard-easy pattern in the search complexity emerges which is similar to the patterns observed for traditional SAT formulas and complete methods.

We were also able to relate this behavior with a new type of phase transition in the structural properties of the gener-

ated formulas. In particular, we showed how the hardness peak corresponds to a point where there is a transition from formulas which have many optimal assignments to formulas where the optimal assignment is unique. And this is perhaps the most important characteristic of our generator; under the right choice of the parameter δ, not only we know that the optimal solution is unique but we also know that it must assign (a predefined) half of the variables to TRUE and half to FALSE. In conclusion, we believe that our generator will be useful in the analysis and development of future SAT heuristics since by knowing what to expect algorithm designers will better test the effectiveness of their search procedures.

References

Achlioptas D., Gomes, C., Kautz H. and Selman B. Generating satisfiable problem instances. In *Proc. AAAI-00*, 2000.

Bengt Aspvall, Micahel F. Plass and Robert E. Tarjan. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Information Processing Letters*, 8(3):121-123, March 1979.

M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of ACM*, 7:201-215, 1960.

I. Gent and T. Walsh. The SAT Phase Transition. In *Proc. ECAI-94*, 105-109, 1994.

Kirkpatrick, S. and Selman, B. Critical behavior in the satisfiability of random Boolean expressions. *Science*, 264, 1297-1301, 1994.

Li, Chu Min and Anbulagan. Heuristics based on unit propagation and satisfiability problems. In *Proc. IJCAI-97*, 1997, 366-371.

Mitchell, D. and Levesque, H.J. Some pitfalls for experimenters with random SAT. *Artificial Intelligence*, Vol. 81(1-2), 1996, 111-125.

Mitchell, D., Selman, B., and Levesque, H.J. Generating hard satisfiability problems. *Artificial Intelligence*, Vol. 81(1-2), 1996. A previous version appeared in *Proc. AAAI-92*, pp. 459--465, San Jose, CA, 1992.

Monasson, R., Zecchina, R., Kirkpatrick, S., Selman, B., and Troyansky, L. Determining computational complexity from characteristic `phase transitions'. In *Nature*, Vol. 400(8), 1999.

Morris, P. The breakout method for escaping from local minima. In *Proc. AAAI-94*, 1993, 40-45.

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

J. Slaney and T. Walsh. Backbones in Optimization and Approximation. In *Proc. IJCAI-01*, 2001.