A wealth of SAT Distributions with Planted Assignments

Tassos Dimitriou

Athens Information Technology, tassos@ait.gr

Abstract. Evaluation of local search heuristics for constraint satisfaction and satisfiability problems is based on the generation of instances that are guaranteed to be satisfiable. One popular method for creating hard satisfiable instances is the use of *complete* search procedures to filter out unsatisfiable instances. This approach however has two problems; first, the size of instances produced is limited considerably and second, the generated instances are far from being random.
Although one can 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 provide a generator for an *optimization* version of k-SAT that has certain useful properties. First, we show how to produce weighted instances of MAX k-SAT where one seeks to maximize the *weight* of satisfied clauses. Second, we provide a nice 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 *tunable complexity*; by appropriately choosing parameters one can control the hardness of the generated instances leading to an easy-hard-easy pattern in the search complexity for good assignments and a new type of phase transition.

1 Introduction

The use of distributions for generating random SAT instances is an important set of benchmark problems for evaluating local search SAT heuristics. Mitchell and Levesque[10], however, have shown that 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. 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[11, 5]. One important limitation, however, in the applicability of these distributions is the fact that they generate both satisfiable and unsatisfiable 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 unfortunately becomes limited.

One way to create a satisfiable distribution of SAT formulas is to start from any assignment (say the all ones assignment) and then create a random formula at the critical region (for the case of 3-SAT this would be when the ratio of clauses to variables is about 4.26). Of course the generated formula doesn't necessarily satisfy the chosen assignment so one would have to delete all unsatisfied clauses first. This method would guarantee that the formula generated is satisfiable. The problem with this approach is that the formula is usually very easy to solve. Local search heuristics easily come up with assignments for such formulas mainly because of the abundance of satisfying assignments around the chosen one. Thus the generated instances are biased and not really on the critical region.

In a similar setting Asahiro, Iwama and Miyano[2] were able to provide hard instances with a single satisfying assignment using a variant of the previous method. Although the AIM approach was an improvement over the previous method, again the generated instances were biased and far from being random. Furthermore, the generator lacked any control parameter that could be used to produce a reasonable phase transition. Other efforts in this area include the translation of cryptographic problems into SAT instances[4, 9]. Although the translated instances are guaranteed to be satisfiable, in practice they are very hard to solve because their solution reduces to a simple exhaustive search of the SAT space (as is usually the case for the cryptographic counterparts). This suggests that these problems are outside the realm of current SAT solvers and cannot be used for their evaluation. Furthermore, the translated instances are neither scalable nor tunable in their hardness.

These drawbacks were remedied in part by Achlioptas *et al.*[1] 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. This instance is then translated to an equivalent Boolean formula which is made k-SAT with the usual introduction of new variables. As demonstrated by Achlioptas *et al.*, this generator has a number of important characteristics. The first is the ability to finely control the hardness of the generated instances by tuning the value of p. The second is the appearance of a new kind of phase transition in the space of problem

instances. It is interesting to note however that while the QCP generator can finely control the hardness of the generated instances, Achlioptas *et al.* ask whether a similar generator can be developed directly for k-SAT.

In this work we introduce a generator for MAX k-WSAT formulas, a *weighted* and *optimization* 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 proving that the optimal assignment is *unique*. Since any satisfiability heuristic when feed 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 better 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 in-

stances of MAX k-WSAT. In general, MAX k-WSAT consists of Boolean expressions in Conjunctive 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 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 sight, it is all we need to create a generator of k-SAT instances with useful computational properties. Furthermore, even when $k = 2$ the problem still remains NP-hard.

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 5.1, this leads to faster implementations of heuristics treating WSAT formulas. We call our model $\mathcal{F}_{n,p,\delta}$, where n indicates the number of variables of each color and p, δ are the parameters used to control the maximum total weight achieved by the hidden assignment (Figure 1). The user can choose any values for δ and p provided $p + \delta \leq 1$. The reason for this restriction will become clear in Lemma 1. We do not include the weight β in the definition of the model as this will be set to a specific value later on (Lemma 2).

The model $\mathcal{F}_{n,p,\delta}$ (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 p 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 $\delta(1 - p)^{-1} \leq 1$ increase the weight of the clause to $\beta + 1$.

Fig. 1. Description of the generator.

By looking at Figure 1 one should observe 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 incurs the maximum possible total weight. We will be able to show later on (Theorem 1) 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. *A super-clause is called monochromatic if it consists of variables of the same color.*

Definition 2. *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).*

We are now ready to prove the first fact that is a simple consequence of the model $\mathcal{F}_{n,p,\delta}$.

Lemma 1. (Monochromatic clauses are lighter on average) *If x, y have the same color then $w(x, y) = \beta + 1$ with probability p and β otherwise. If x, y have different colors then $w(x, y) = \beta + 1$ with probability $p + \delta$ and β otherwise.*

Proof. The first statement is obvious since by definition monochromatic clauses have weight $\beta+1$ with probability p. 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 $\delta(1 - p)^{-1}$ increased its weight. The probability of these two events is $p + (1 - p)\delta(1 - p)^{-1} = p + \delta$. □

This lemma provides an alternative definition for our model and is used in the proof of 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.

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

Proof. Suppose there is an assignment A that achieves total weight W and has $0 < k < n$ variables set to True and $2n - k$ variables set to False. We will show that by choosing β accordingly, there exists a better assignment that achieves greater weight and has its variables split.

Consider the bipartite graph (A, B) formed by putting the true variables on side A and the false on side B. Furthermore, for every pair (x, y) where $x \in A$ and $y \in B$ add the edge from x to y and assign to it the weight of the super-clause $c(x, y)$.

Consider now an arbitrary super-clause $c(x, y) = (x\bar{y} + \bar{x}y)$. This super-clause simply spells the fact that x and y must have *different* truth values in order for $c(x, y)$ to be satisfied and contribute its weight $w(x, y)$ to the total sum. Thus, given the particular assignment A, there can be at most $k(2n - k)$ satisfied super-clauses and the total weight W incurred by A will be equal to the sum of the edges' weights in the bipartite graph. Let there be m edges of weight $\beta+1$ and the rest with weight β. Then the total weight will be equal to $W = m(\beta+1)+[k(2n-k)-m]\beta = k(2n-k)\beta+m$, where the m term comes from the edges with weight $\beta + 1$. In any case, $m \leq k(2n - k) < n^2$. Thus,

$$W < k(2n - k)\beta + n^2 = n^2\beta - [(n - k)^2\beta - n^2]. \tag{1}$$

Consider now any assignment A' that have its variables split and let m' be the number of edges of weight $\beta + 1$. By the same argument as before the total weight W' achieved by A' will be at least $W' = n^2\beta + m' \geq n^2\beta$. Since $0 < k < n$, by choosing $\beta = n^2$ we see that the term $[(n - k)^2\beta - n^2]$ in (1) is always positive, thus making the weight W smaller than the weight W' of any assignment with split variables. We conclude that 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 NAESAT 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 $\mathcal{F}'_{n,p,\delta}$. 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 $\mathcal{F}_{n,p,\delta}$ if and*

only if it achieves total weight $W + c_f$ when the formula is generated according to $\mathcal{F}'_{n,p,\delta}$, where c_f in a constant that is easily computable and depends only on the particular formula f.

The proof is very similar to the proof of Lemma 2 and is omitted from this extended abstract. Again we only have to look for split assignments in the new model since by choosing $\beta = n^2$, the best assignments for formulas generated according to $\mathcal{F}'_{n,p,\delta}$ 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 3. *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 $\delta \geq \Omega(\sqrt{(1-p)\ln n/n})$, 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. We leave a full proof for the final version of the paper. 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. □

The optimality theorem characterizes implicitly the values of p for which it is safe to assume that the hidden assignment is optimal with high probability. Since by the definition of the model we know that δ must be less than $1-p$, it is clear that the theorem will be true for values of p satisfying $1-p \geq \Omega(\sqrt{(1-p)\ln n/n})$ or equivalently

$$p \leq 1 - c\frac{\ln n}{n} \tag{2}$$

for some constant c. Thus our approach cannot be used for all formulas, but only for formulas where the monochromatic clauses are not too heavy, as indicated by Equation 2 and Lemma 1.

4 Extension to MAX k-WSAT

To generate MAX k-WSAT instances , $k \geq 3$, we follow the same approach as for the 2-WSAT 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 $\mathcal{F}^k_{n,p,\delta}$, where k indicates that we working with k-WSAT formulas (Figure 2).

The model $\mathcal{F}^k_{n,p,\delta}$ (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 p set the clause weight $w(x_1, x_2, \ldots, x_k)$ 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 $\delta(p-1)^{-1}$ increase the weight of the clause to $\beta + 1$.

Fig. 2. Generator for k-WSAT formulas.

As in the 2-WSAT case, the super-clauses are satisfied only when clause variables have different truth values. 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 Experimental Results; the case for $p = \frac{1}{2}$

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. For ease of exposition we decided to work with formulas satisfying $p = \frac{1}{2}$. Thus monochromatic clauses get weight $\beta + 1$ with probability a half while non-monochromatic clauses get that weight with probability $\frac{1}{2} + \delta$. Although we leave a more detailed experimentation for a final version of this paper we will see that even in this case the formulas generated exhibit some very important properties.

5.1 Locating the Hard Instances

The local search procedure we used for our tests is a modified version of WalkSat[13] 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. In subsequent work we plan to perform a more thorough analysis using a more represenative collection of search methods. 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.

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

	WalkSat	Weighted version of WalkSat
Goal	Maximize *number* of satisfied clauses	Maximize *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.

solvable in linear time[3]. Although we leave a more detailed analysis of k-WSAT formulas, $k \geq 3$, for the final version of this paper, preliminary work shows that they exhibit similar properties to the 2-WSAT case. In all the figures that follow each sample point was computed after generating 1000 random instances of MAX 2-WSAT.

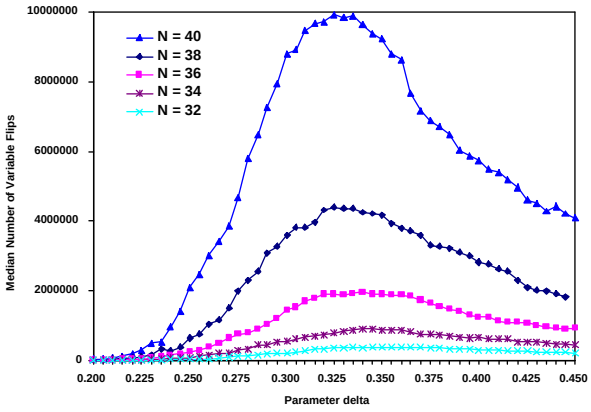


Fig. 3. Median number of total variable flips for random 2-WSAT formulas as a function of the parameter δ.

Figure 3 shows the *median* of the *total number of variable flips* required by WalkSat to locate an assignment that achieves the maximum total weight (as is implied by 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 similar to the behavior of ordinary 3-SAT formulas [11, 5]. This figure also suggests the existence of a *critical region* although we cannot link this behavior with the length of the formulas as all of them have the same number of clauses. We were able, however, to relate this

behavior with a phase transition in structural properties of the WSAT instances. It is perhaps worthwhile to comment a little on the shape of the curves in Figure 3. Although the computational cost follows an easy-hard-easy pattern, the second "easy" region where δ is large is no longer very easy compared to the first region where δ is small. This is reminiscent of the behavior of 3-SAT(B), the *bounded decision* versions of 3-SAT defined by Zhang[15], where one is looking for an assignment that violates no more than B constraints. When $B = 0$, one has 3-SAT; when B is the optimal solution cost, one has MAX 3-SAT. Thus, such distributions lie in some sense between the decision problem and its optimization counterpart and like the WSAT instances exhibit easy-hard-"less easy" patterns.

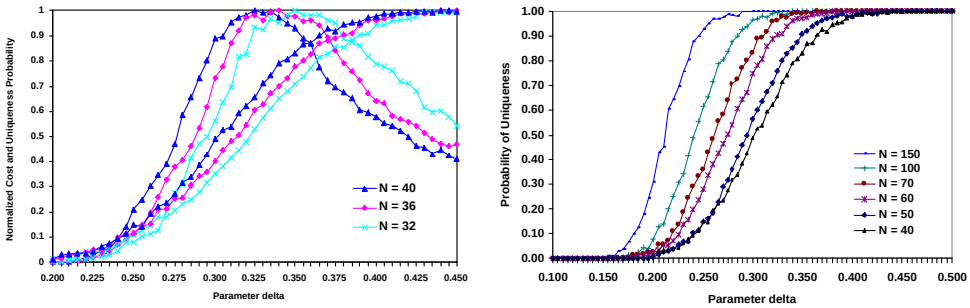
In general, as was shown in [6, 16, 14, 15] and other works, the phase transitions of some NP-complete *decision* problems follow easy-hard-easy patterns and the phase transitions of some NP-hard *optimization* problems follow easy-hard patterns. Thus one may ask, where is the easy-hard behavior of the WSAT formulas? We performed some initial experiments and found that actually WSAT formulas exhibit the behavior of optimization problems, but only when p grows larger than 1/2. Thus the value of $p = 1/2$ is middle ground and by increasing the value of p one gets a wealth of distributions with higher computational costs and easy-hard patterns.

5.2 Phase Transition

An important characteristic of Figure 3 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 now is to demonstrate a relationship between the hard region and a phase transition in the structural properties of the WSAT formulas.

It is clear that we cannot have a SAT/UNSAT transition as all instances are unsatisfiable. A more profound concept related to phase transitions is the *backbone* ratio of a problem which 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 raise from nearly 0 to nearly 1, with the hardest instances lying around the 50% point. In the case of WSAT formulas, however, we chose not to work with backbones because now solutions are planted and, provided that δ is sufficiently large, solutions are also unique. Thus there is no point in trying to relate the hardness peak with the backbone as there is essentially only one solution and most of the variables have a fixed value. (Furthermore, such results were examined by Zhang[15] on MAX-3SAT formulas.) We were able, however, to

relate WSAT's behavior with the probability of uniqueness of the hidden assignment, which is the crucial structural property of WSAT formulas.



(a) (b)

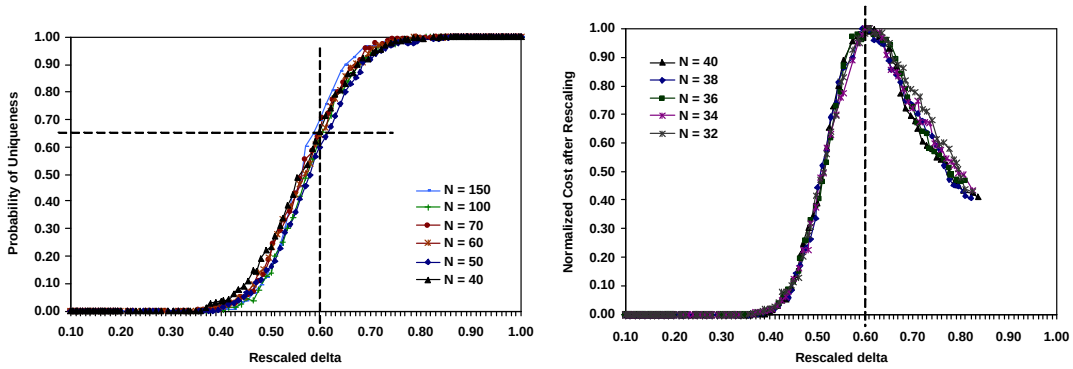
Fig. 4. (a) Normalized cost and phase transition. The bell shaped curves correspond to median number of flips normalized to 1 while the sigma shaped curves show the uniqueness probability of the hidden assignment. (b) Phase transition for various values of n.

In Figure 4(a) we show how the probability that there exist good assignments other than the planted one changes as a function of the parameter δ for $n = 32, 36, 40$. In the same figure we also included the *normalized* cost of WalkSat for locating the hidden assignment (bell shaped curves). Observe now the clear movement to the left and the remarkable correspondence between the hardness peak and the point where the uniqueness probability of the optimal solution is about 65%. The main empirical observation we can draw from this picture is that *the hardest 2-WSAT formulas for WalkSat lie at the point where about 65% of the formulas have the planted solution as the optimal one*. We named the WalkSat algorithm in this conclusion since we expect the location of the peak to depend on the particular heuristic used. Other heuristics may peak at different values but in any case we expect their behavior to be similar to WalkSat's.

In Figure 4(a) there is a clear shift in computational cost as the value of n increases. In Figure 4(b) we observe a similar behavior for the uniqueness probability of the optimal solution for a larger range of values. Observe how the threshold function sharpens up for larger values of n, like the satisfiability threshold function for random k-SAT formulas[11]. 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 (set $p = 1/2$ in Theorem 1).

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*[8], 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.



(a) (b)

Fig. 5. (a) Phase transition for various values of n after rescaling. (b) Computational cost for various values of n after rescaling.

Figure 5(a) shows the result of rescaling the curves of Figure 4(b). The uniqueness probability is plotted against δ', a rescaled version of δ equal to $\delta' = \delta n^{\epsilon/2}\sqrt{1-\epsilon}$, where $\epsilon = 0.56$. It is perhaps instructive to describe how we derived the rescaling factor $n^{\epsilon/2}\sqrt{1-\epsilon}$. Theorem 1 (with $p = 1/2$) 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 $n^{1-\epsilon}$, for some ϵ, then the translated point must become $\delta_0' = c\sqrt{\frac{\ln n^{1-\epsilon}}{n^{1-\epsilon}}}$. By some algebraic manipulation, 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 5(a). Finally, Figure 5(b) demonstrates how the computational cost for various values of n collapses into a universal curve. To obtain this, we first normalized the curves Figure 3 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 5(a).

6 Conclusions and Future Research

In this work we presented a generator for instances of MAX k-WSAT 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. Furthermore, the distributions examined here seem to lie in the middle ground between decision and optimization problems. When p, the other parameter of our generator, is set to $1/2$, the computational cost of finding the optimal solution exhibits an easy-hard-"less easy" pattern which is not typical of optimization problems as should be the case for MAX-WSAT formulas. However, as we hinted in Section 5.1, when the value of p increases, instances should behave more like traditional optimization problems and easy-hard patterns should emerge.

We were able to relate this behavior of WSAT formulas with a new type of phase transition in the structural properties of the generated instances. 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.

Our work leaves open some ground for further improvements and research. One direction would be to eliminate the weights from the clauses and produce a generator for [MAX] k-SAT instances directly. It seems that the weights are only used to limit the search for split assignments so one may ask if there is a way to do this using no weights. Unfortunately, at this point we don't know how this can be done without losing the structure of the hidden assignment and the a priori knowledge of optimality. Another important question is if the quadratic number of clauses in the case for 2-WSAT (and the $O(n^k)$ number for the general case) can be reduced to linear. Is it possible to generate formulas, even with weights, in which the number of clauses is linear and the hidden assignment is preserved? This would speedup the execution time of algorithms and would further strengthen the hardness results of the generated instances.

Finally, our model is reminiscent of graph theoretic models in which a solution is planted in advance (such as in the clique or coloring problem). The purpose of planting solutions to such problems is to come up with

algorithms that are able to recover the planted structure, hoping that these algorithms will behave equally well in real life instances. Our findings for WalkSat do *not* imply that such an algorithm is unlikely to exist for the WSAT model we propose here. In fact Theorem 1 suggests that such a specialized algorithm may exist. Coming up with such an algorithm may pinpoint the important characteristics of the WSAT formulas and may further help in the simplification of them as well as in the evaluation of other SAT search methods.

Acknowledgements

The author wishes to thank Christos Papadimitriou and the anonymous referees for some very useful comments.

References

1. Achlioptas D., Gomes, C., Kautz H. and Selman B. Generating satisfiable problem instances. In *Proc. AAAI-00*, 2000.
2. Asahiro, Y., Iwama, K. and Miyano, E. Random generation of test instances with controlled attributes. In *Second DIMACS Challenge on Cliques, Coloring and Satisfiability*, October 1993.
3. 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.
4. J. Crawford and M. Kearns. Instances for learning the parity function. In [7].
5. I. Gent and T. Walsh. The SAT Phase Transition. In *Proc. ECAI-94*, 105-109.
6. I. Gent and T. Walsh. The TSP Phase Transition. *Artif. Intel.*, 88:349–358, 1996.
7. H. Hoos. SATLIB. A collection of SAT tools and data, 1999. `www.informatic.tu-darmstadt.de/AI/SATLIB`.
8. Kirkpatrick, S. and Selman, B. Critical behavior in the satisfiability of random Boolean expressions. *Science*, 264, 1297-1301, 1994.
9. Massacci, F. Using WalkSAt and RelSat for cryptographic key search. In *Proc. IJCAI-99*, 199, pp. 290–295.
10. Mitchell, D. and Levesque, H.J. Some pitfalls for experimenters with random SAT. *Artificial Intelligence*, Vol. 81(1-2), 1996, 111-125.
11. Mitchell, D., Selman, B., and Levesque, H.J. Generating hard satisfiability problems. *Artificial Intelligence*, Vol. 81(1-2), 1996.
12. Monasson, R., Zecchina, R., Kirkpatrick, S., Selman, B., and Troyansky, L. Determining computational complexity from characteristic 'phase transitions'. In *Nature*, Vol. 400(8), 1999.
13. B. Selman, H. A. Kautz and B. Cohen. Local search strategies for satisfiability testing. In *Second DIMACS Challenge on Cliques, Coloring and Satisfiability*, 1993.
14. J. Slaney and T. Walsh. Backbones in Optimization and Approximation. In *Proc. IJCAI-01*, 2001.
15. W. Zhang. Phase transitions and backbones of 3-SAT and MAX 3-SAT. In *Proc. CP-2001*.
16. W. Zhang and R. E. Korf. A study of complexity transitions on the asymmetric Travelling Salesman Problem. *Artificial Intelligence*, 81:223–239, 1996.