

# SAT Distributions with Planted Assignments and Phase Transitions between Decision and Optimization Problems

Tassos Dimitriou

*Athens Information Technology,  
Markopoulo Ave., 190 02, Athens, Greece.*

---

## Abstract

We present a generator for *weighted* instances of MAX  $k$ -SAT in which every clause has a weight associated with it and the goal is to maximize the total weight of satisfied clauses. Our generator produces formulas whose hardness can be finely tuned by two parameters  $p$  and  $\delta$  that control the weights of the clauses. Under the right choice of these parameters an easy-hard-easy pattern in the search complexity emerges which is similar to the patterns observed for traditional SAT distributions.

What is remarkable, however, is that the generated distributions seem to lie in the middle ground between decision and optimization problems. Increasing the value of  $p$  from 0 to 1 has the effect of changing the shape of the computational cost from an easy-hard-easy pattern which is typical of *decision* problems to an easy-hard pattern which is typical of *optimization* problems. Thus our distributions seem to bridge the gap between decision and optimization versions of SAT.

Furthermore, we demonstrate that these phase transitions are related to sudden changes to a quantity similar to the backbone of a SAT formula. In our model not only we know how the optimal solution looks like (because we plant it in advance) but we also give evidence that it is *unique*. Thus our generator comes with an indication of optimality of the planted assignment which is basically the structural property that is related to the phase transition phenomena observed.

---

## 1 Introduction

Phase transition phenomena in combinatorial search problems have proved a fertile source of research activity for over a decade. An informal description of

---

*Email address:* [tdim@ait.edu.gr](mailto:tdim@ait.edu.gr) (Tassos Dimitriou).

a “phase transition” is the behavior whereby “small” changes in certain parameters of a system cause dramatic shifts in some globally observed quantity. A typical example of such a behavior is the satisfiability (SAT) of Boolean formulas. The computational cost of solving random 3-SAT instances (formulas in Conjunctive Normal Form with 3 literals per clause) exhibits transitions from easy to hard and back to easy [11,7] as the ratio of number of clauses to variables increases. In combinatorial graph theory, similar phenomena have been observed with respect to random  $n$ -vertex graphs in which edges are added with some probability  $p(n)$ ; when one considers a certain property  $\Pi$  (connectivity, 3-colorability, etc.) then there is a value for the edge probability  $p(n)$  where the property  $\Pi$  appears abruptly [6,2].

The interest in phase transition phenomena stems from experimental studies of search heuristics for NP-complete problems [4,9,11,7,8], where the probability of a random instance having a solution is mirrored in the run-time behavior of the methods used to find the solution. Phase transitions usually depend on some control or order parameter that can be adjusted to control the hardness of the problem. For example, the probability that a random graph is connected or has a hamilton cycle depends on the edge density[6,2]; the satisfiability and the hardness of 3-SAT formulas depends on the ratio of clauses to variables [4,11,5,7], and so on.

Furthermore, it has been observed that instances “outside” the threshold region are typically solved easily as opposed to instances close to the threshold point which are much harder to solve. In addition, phase transitions for NP-complete decision problems typically have easy-hard-easy patterns while phase transitions for the corresponding optimization problems follow easy-hard patterns [8,16,15].

Our motivation for this research is threefold; First we want to introduce a new distribution of SAT instances that will bridge the gap and possibly help us understand the relationship between the phase transitions of decision problems and those of their optimization counterparts. Second, we want to identify and locate difficult instances that can be used in the development of new solving methods. Finally, we want to understand the characteristics of optimal solutions and the behavior of algorithms for finding them with respect to certain structural properties of the instances at hand.

The instances we generate are  $k$ -SAT formulas where every clause has a *weight* associated with it. The goal is to find an assignment that maximizes the *sum of weights* of the satisfied clauses. The generated instances are parameterized by two quantities  $p$  and  $\delta$  which control the weights associated with the clauses. By carefully setting the values of these two parameters one can generate difficult to solve formulas, thus making it possible to test algorithms on *hard* generated instances only.

In addition, our generator has two more important characteristics that are related to the values of  $p$  and  $\delta$ . We generate our formulas by first splitting the variables in two predefined sets  $G$  and  $B$  of equal size and then assigning the weights to the clauses according to the set clause variables come from. We demonstrate experimentally that the assignment that has set the variables from  $G$  to true and the ones from  $B$  to false (or vice versa) is not only optimal when  $\delta$  is large enough but is also *unique*. Since any satisfiability heuristic when fed with an instance from our generator will try to maximize the 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 the optimal assignment. Although the problem we consider here is a *maximization* one and phase transitions should exhibit “easy-hard” patterns [8,16,15], by increasing the value of  $p$  from 0 to 1 one starts with “easy-hard-easy” patterns which are typical of *decision* problems to end up with “easy-hard” patterns which are typical of *optimization* problems. Thus our distributions seem to bridge the gap between decision versions and optimization versions of SAT.

Furthermore, 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 planted assignment is unique. The point to be made is that this transition coincides with the hardest to solve problem instances.

## 2 Generator for MAX $k$ -WSAT

Our generator produces *weighted* instances of the MAX  $k$ -SAT problem, which we call 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 with it. 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  $k$ -WSAT is NP-hard as MAX  $k$ -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  $k$ -WSAT, in which *all* weights to

**The model  $F_{n,p,\delta}$  (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  $\beta$ .
  - 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.

the clauses are either  $\beta$  or  $\beta + 1$ , where  $\beta$  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. 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 3, this leads to faster implementations of heuristics treating WSAT formulas.

We call our model  $F_{n,p,\delta}$ , where  $n$  indicates the number of variables of each color and  $p, \delta$  are the parameters used to control the maximum total weight achieved by the hidden assignment (Figure 1). The user can choose any values for  $\delta$  and  $p$  provided  $p + \delta \leq 1$ . The reason for this restriction will become clear in Lemma 3. We do not include the weight  $\beta$  in the definition of the model as this will be set to a specific value later on (Lemma 4). (While we only show the generator for 2-WSAT formulas, the extension to  $k$ -WSAT formulas should be straightforward.)

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  $c_{x,y}^1 = (x + y)$  and  $c_{x,y}^2 = (\bar{x} + \bar{y})$ .

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 give an indication later on (Lemma 7) that the best assignment (the *planted assignment* as we call it) 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 results.

**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  $F_{n,p,\delta}$ .

**Lemma 3 (Monochromatic clauses are lighter on average)**

*If  $x, y$  have the same color then*

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

*If  $x, y$  have different colors then*

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

**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  $\beta$  and with probability  $\delta(1 - p)^{-1}$  increased its weight. The probability of these two events is  $p + (1 - p)\delta(1 - p)^{-1} = p + \delta$ .  $\square$

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 for.

**Lemma 4 (Look for split assignments)** *When the weight  $\beta$  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  $0 < v < n$  variables set to true and  $2n - v$  variables set to false. We will show that by choosing  $\beta$  accordingly, there exists a better assignment that achieves greater weight and has its variables split.

Consider the bipartite graph  $(L, R)$  formed by putting the true variables on side  $L$  and the false on side  $R$ . Furthermore, for every pair  $(x, y)$  where  $x \in L$  and  $y \in R$  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  $v(2n - v)$  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  $\beta$ . Then the total weight will be equal to  $W = m(\beta + 1) + [v(2n - v) - m]\beta = v(2n - v)\beta + m$ , where the  $m$  term comes from the edges with weight  $\beta + 1$ . In any case,  $m \leq v(2n - v) < n^2$ . Thus,

$$W < v(2n - v)\beta + n^2 = n^2\beta - [(n - v)^2\beta - n^2]. \quad (1)$$

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  $0 < v < n$ , by choosing  $\beta = n^2$  we see that the term  $[(n - v)^2\beta - n^2]$  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.  $\square$

Although the previous proof focuses on the 2-SAT case, the proof generalizes to  $k$ -SAT instances as well. For example, in the 3-SAT case we have to modify the bipartite graph by considering “hyperedges” formed by triples of variables  $x, y, z$ , where at least two of these variables belong to *different* sides. In this case, the weight we assign to each such hyperedge is simply the clause weight  $w(x, y, z)$ . As in the 2-SAT case, only these clauses contribute their weights to the total sum and there can be at most  $v(2n - v)(n - 1)$  satisfied super-clauses. Working exactly the same way as before, we see that for  $\beta = n^2$  the best assignments split their variables.

Furthermore observe that the 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  $c_{x,y}^1$  and  $c_{x,y}^2$  of the super-clause. Call this new model  $F'_{n,p,\delta}$ . Now, we have to take into account the weight incurred by these clauses even if both literals have the same truth value.

**Lemma 5 (Equivalence of the two models)** *An assignment  $A$  achieves total weight  $W$  for a formula  $f$  generated according to  $F_{n,p,\delta}$  if and only if it achieves total weight  $W + c_f$  when the formula is generated according to  $F'_{n,p,\delta}$ , where  $c_f$  is a constant that is easily computable and depends only on the particular formula  $f$ .*

The proof is very similar to the proof of Lemma 4 and is omitted. 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  $F'_{n,p,\delta}$  split their variables. 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 4 we are allowed to do so.

**Definition 6** *We say an assignment has distance  $k$  from the planted 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  $k = 0$ . Our goal now is to provide evidence that for a suitable choice of the parameter  $\delta$ , the optimal assignment is one that has the green variables set to true and the blue variables set to false (or vice versa). We do this by comparing the expected weight achieved by the planted assignment with that of an assignment that is at distance  $k$  from the hidden one.

**Lemma 7** *The expected total weight achieved by the hidden assignment outweighs that achieved by an assignment at distance  $k$ .*

**PROOF.** Consider any assignment  $A$  and let  $(L, R)$  be a bipartite graph formed by putting the true variables of  $A$  on the left side and the false ones on the right side of the graph. Again for any clause  $c(x, y)$  where  $x \in L$  and  $y \in R$  add the edge from  $x$  to  $y$  and assign to it a weight equal to  $w(x, y) - n^2$ . Notice that here we have modified the edge weights a little bit. The reason is that we want to focus only on the clauses that have weight  $n^2 + 1$ . Since all clauses have weights  $n^2$  or  $n^2 + 1$ , by subtracting  $\beta = n^2$  from the weights, we are left with edges that have weight one, corresponding exactly to the heavier clauses.

What happens now, is that edges corresponding to monochromatic clauses appear with probability  $p$  while edges corresponding to non-monochromatic clauses appear with probability  $q = p + \delta$ .

Let's compute now the average number of edges in the bipartite graph when  $A$  is the planted assignment. From our construction, there are exactly  $n^2$  potential edges, each appearing with probability  $q$ , thus the average number of edges is simply  $E_0 = n^2q$ . Consider now the case where the assignment  $A$  is one at distance  $k$  from the planted. This means that there are  $k$  blue and  $n - k$  green variables that are assigned the value true. Thus in the bipartite graph there are exactly  $2k(n - k)$  potential monochromatic edges and  $[k^2 + (n - k)^2]$  non-monochromatic ones. The expected number of edges is therefore

$$E_k = 2k(n - k)p + [k^2 + (n - k)^2]q.$$

The values  $E_0$  and  $E_k$  correspond to the total weight achieved by these assignments in excess of the weight  $n^2\beta$  achieved by all assignments. If  $E_0$  is bigger than  $E_k$ , this is an indication that the planted assignment achieves better overall weight. This is verified easily:  $E_0$  is greater than  $E_k$  provided  $\delta \geq 0$ .  $\square$

The above lemma doesn't say that it is only the planted assignment that maximizes some WSAT formula. Other assignments may perform equally well, as is also demonstrated in the experimental section. However, as the value of  $\delta$  increases, we expect the planted assignment not only to be optimal but to be the *unique* optimal one (see Section 4).

### 3 Hardness Results for WSAT

Our motivation in this section is to show that easy and hard  $k$ -WSAT instances can be predictable in advance. This will enable designers of local search SAT heuristics to test their algorithms on hard  $k$ -SAT instances only in which the optimal solution is known beforehand. 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 [3]. In all the figures that follow each sample point was computed after generating 1000 random instances of MAX 2-WSAT.

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

Table 1  
Changes to the basic WalkSat algorithm.

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

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  $\beta$  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.

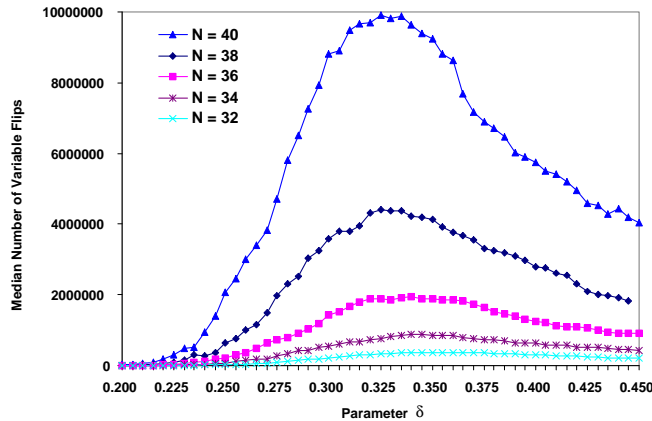


Fig. 2. Median number of total variable flips for random 2-WSAT formulas as a function of the parameter  $\delta$ , when  $p = 1/2$ .

Figure 2 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 2-WSAT formulas with  $p$  equal to  $\frac{1}{2}$  and  $n = 32, 34, 36, 38$  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 [11,7].

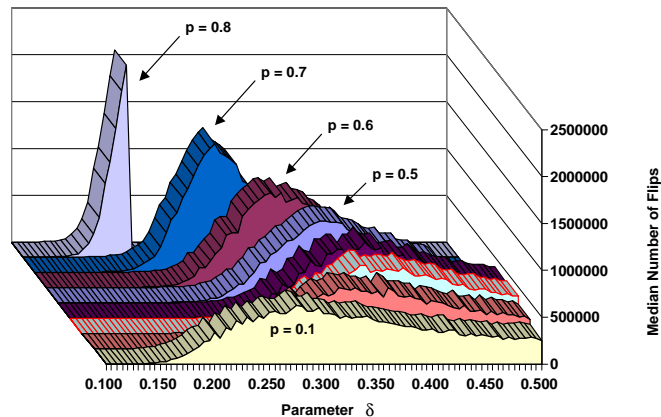


Fig. 3. Computational cost for random 2-WSAT formulas with  $n = 34$  and various values of  $p$  and  $\delta$ .

It is perhaps instructive at this point to comment a little on the shape of the curves in Figure 2. Although the computational cost follows an easy-hard-easy pattern, the second “easy” region where  $\delta$  is large is no longer very easy compared to the first region where  $\delta$  is small. This is reminiscent of the behavior of  $3\text{-SAT}(B)$ , 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 [8,16,14,15] 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? As we will see in Figure 3, WSAT formulas exhibit the behavior of optimization problems but only when  $p$  grows larger than  $1/2$ . Thus indeed 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.

Figure 3 shows the computational cost required to find a good assignment for 2-WSAT formulas with  $n = 34$  variables and  $p$  ranging from 0.1 to 0.8. Starting with  $p = 0.1$  (curve in the front) we see that the point of maximum cost is moving slowly to the right with a parallel increase in its maximum value, as  $p$  becomes 0.4. However, when  $p$  becomes 0.5 or larger the points of maximum cost are moving slowly to the left to acquire the maximum value when  $p = 0.8$ . All the curves exhibit easy-hard-“less easy” patterns with the exception of the curve for  $p = 0.8$  which has an easy-hard pattern as  $\delta$  increases from 0 to its maximum value 0.200. Thus in this particular curve the computational cost remains maximum for values of  $\delta > 0.200$ .

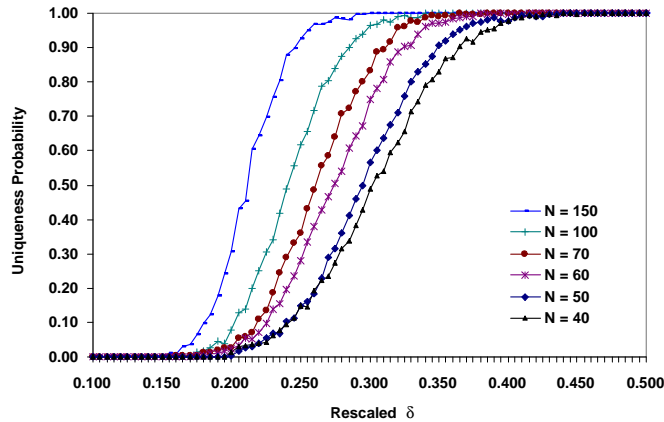


Fig. 4. Phase transition for  $p = 1/2$  and various values of  $n$ .

## 4 Phase Transitions

An important characteristic of Figure 2 is that the transition region becomes narrower (occurs for a smaller range of  $\delta$ ) 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 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 that of a *backbone* which in some sense is the set of all *frozen decisions* [12,14], i.e. those with fixed outcomes for all possible solutions. For example, in SAT the backbone of a formula is the set of all literals that are true in all satisfying assignments [12]. A phase transition in such a case has the backbone ratio raise from nearly 0 to nearly 1, with the hardest instances lying around the 50% point, not only in their decision version but in their optimization as well [12,14–16]. In the case of WSAT formulas, however, we chose not to work with backbones as there is essentially only one solution and most of the variables have a fixed value. We were able, however, to relate the WSAT behavior with the probability of uniqueness of the hidden assignment, which is the crucial structural property of WSAT formulas.

Figure 4 shows the uniqueness probability of the optimal solution for  $p = 1/2$  and a large range of values for  $n$ . Observe how the threshold function sharpens up for larger values of  $n$ , like the satisfiability threshold function for random  $k$ -SAT formulas[11]. So, now, the question becomes: given an arbitrary value of  $n$  how can we determine the value of  $\delta$  that results in the most difficult to solve instances? The answer is given by *finite-size scaling* [10], in which the horizontal axis is rescaled by a quantity that is a function of  $n$ .

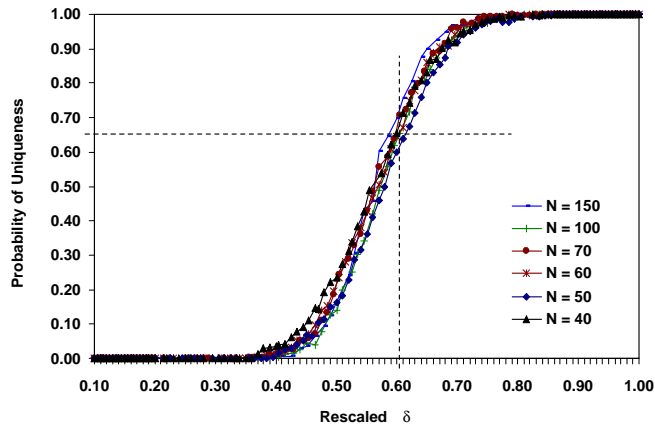


Fig. 5. Phase transition for  $p = 1/2$  and various values of  $n$ , after rescaling.

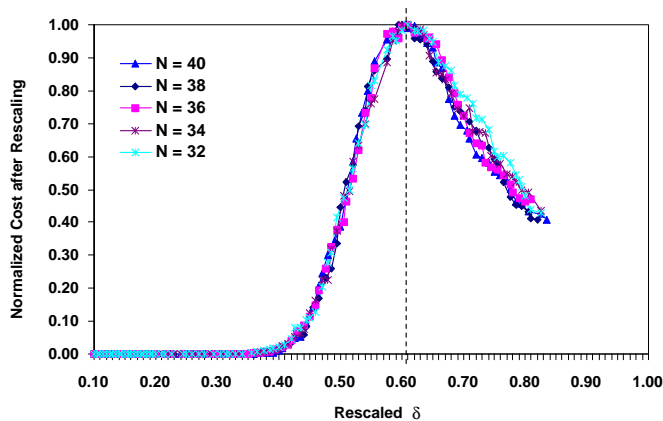


Fig. 6. Computational cost for  $p = 1/2$  and various values of  $n$  after rescaling.

Figure 5 shows the result of rescaling<sup>1</sup> the curves of Figure 4. The uniqueness probability is plotted against  $\delta'$ , a rescaled version of  $\delta$  equal to  $\delta' = \delta n^{\epsilon/2} \sqrt{1 - \epsilon}$ , where  $\epsilon = 0.56$ . Finally, Figure 6 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 2 and then applied the rescaling described previously. We see clearly that the critical point is when the *rescaled*  $\delta$  is equal to 0.60 which corresponds to the 65% uniqueness probability in Figure 5. Thus the main empirical observation we can draw from these pictures is that when  $p = 1/2$ , *the hardest 2-WSAT formulas lie at the point where about 65% of them have the hidden assignment as the optimal one.*

<sup>1</sup> Although we were able to come up with a rescaling formula that works for a wide range of values, the results should be considered “approximate”. However, even if we don’t know how to obtain a proof of the formula, the results are validated experimentally.

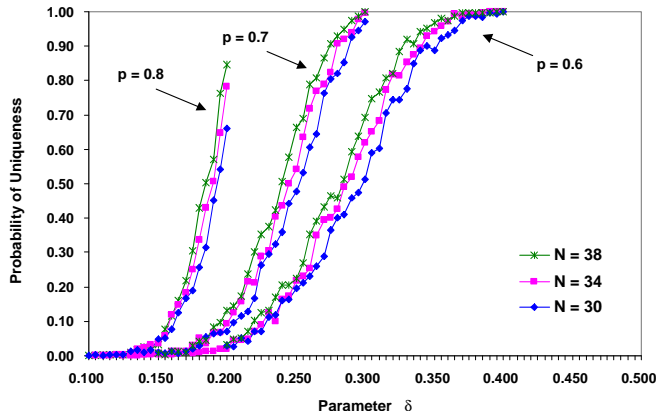


Fig. 7. Phase transition for  $p = 0.6, 0.7, 0.8$  and various values of  $n$ .

To summarize our findings so far, we have seen that WSAT formulas exhibit phase transitions that are related to the uniqueness probability of the optimal assignment. Furthermore, we saw that by increasing the value of  $p$  (Figure 3) one obtains the hardest to solve instances, with easy-hard patterns in their cost. An interesting question is *why* does this happen. We believe that when  $p$  is large, most clauses (irrespective of their color) have large weights which makes it extremely difficult to locate the variables that achieve the largest overall weight. When on the other hand  $p$  is small, it is the value of  $\delta$  that defines the hardest instances. Thus in this case we get easy-hard-easy patterns with medium values for  $\delta$  defining the most difficult to solve problems.

Another interesting question is whether the characterization we obtained using finite size scaling for the case  $p = 1/2$  also applies to other values of  $p$ . In particular, it is important to know the threshold point for distributions with large values of  $p$  as these generate the most interesting formulas from the algorithm designer's point of view.

Figure 7 shows how the probability of uniqueness changes as a function of  $\delta$ , when  $p = 0.6, 0.7, 0.8$  and  $n = 30, 34, 38$ . We were able to plot all these curves in the same figure as the separation introduced by increasing the value of  $p$  was a lot more than the separation introduced by increasing the value of  $n$ . Furthermore, it is worthwhile to observe how the threshold function sharpens up for larger values for  $p$  indicating an abrupt change in the uniqueness probability, similar to that observed for the backbone of SAT distributions [12,14–16].

It turns out that the rescaling formula

$$\delta' = \delta n^{\epsilon/2} \sqrt{1 - \epsilon} \quad \text{with } \epsilon = 0.56$$

works equally well for other values of  $p$  ( $\neq \frac{1}{2}$ ) provided  $p$  is kept fixed and only

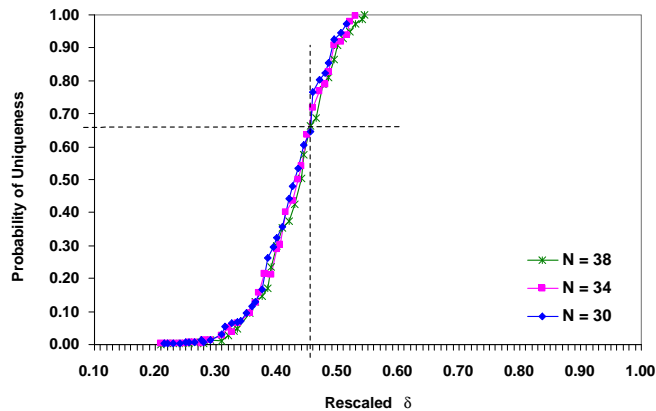


Fig. 8. Phase transition for  $p = 0.7$  and various values of  $n$ , after rescaling.

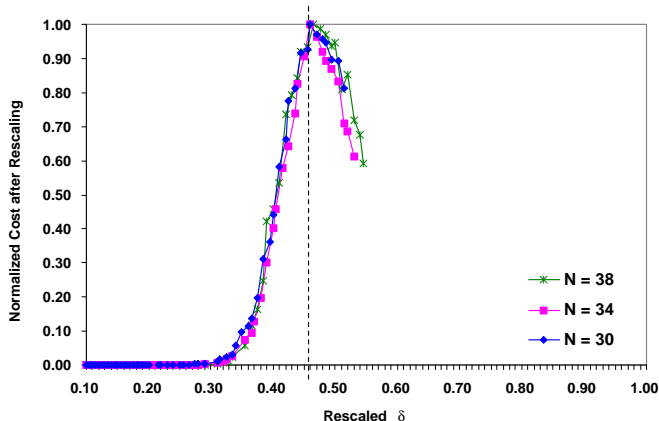


Fig. 9. Computational cost for  $p = 0.7$  and various values of  $n$  after rescaling.

$n$  varies. Figure 8 shows the result of rescaling the curves of Figure 7, when  $p = 0.7$ . When the same rescaling is applied to the *normalized* computational costs of finding the best assignment for values of  $n = 30, 34, 38$ , we obtain the universal match shown in Figure 9. The only difference is that the peak value happens for a different value of the rescaled  $\delta$  (in this case  $\delta' = 0.46$ ). What is remarkable however, is that again the hardest to solve instances seem to live at the 65% probability of uniqueness point as shown in Figure 8. Thus using this approach one can concentrate on large values of  $p$  (where the really hard WSAT distributions are) and use the rescaling formula to generate the hardest to solve instances.

## 5 Conclusions and Future Research

In this work we presented a generator for instances of MAX  $k$ -WSAT in which every clause has a weight associated with it and the goal is to maximize

the total weight of satisfied clauses. We showed that our generator produces formulas whose hardness can be finely tuned by two parameters  $p$  and  $\delta$  that control the weights of the clauses. Under the right choice of these parameters an easy-hard-easy pattern in the search complexity emerges which is similar to the patterns observed for traditional SAT distributions.

Furthermore, the distributions examined here seem to lie in the middle ground between decision and optimization problems. Increasing the value of  $p$  from 0 to 1 has the effect of changing the shape of the computational cost from an easy-hard-easy pattern typical of *decision* problems to an easy-hard pattern typical of *optimization* problems. Thus our distributions seem to bridge the gap between decision and optimization versions of SAT. Furthermore, the hardest instances overall seem to be the ones with the largest value of  $p$ . 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  $\delta$ , 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. Clearly, one research 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. In a similar setting, Achlioptas *et al.* [1] demonstrated how one can generate satisfiable boolean formulas starting from a partially filled Latin square with guaranteed solution and transforming these instances to  $k$ -SAT formulas by standard reduction techniques. However, they left open the question whether a similar generator can be developed directly for  $k$ -SAT.

Another important question is whether 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 a linear one. 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. Coming up with such an algorithm may pinpoint the important characteristics of the WSAT formulas and may help in the simplification of them as well as in the evaluation of other SAT search methods. Furthermore, coming up with a fast algorithm, even for a smaller range of the parameters, may lead to an efficient approximation algorithm for the general problem.

## Acknowledgements

The author wishes to thank Christos Papadimitriou and the anonymous reviewers 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] Noga Alon, Joel H. Spencer. *The probabilistic method*, Wiley, 2000.
- [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] P. Cheeseman, B. Kanefsky and W.M. Taylor. Where the really hard problems are. In *Proc. IJCAI-91*, pp. 331–337, Australia, 1991.
- [5] J. Crawford and L.D. Auton. Experimental results on the cross over point in satisfiability problems. In *Proc. AAAI-93*, pp. 21–27, 1993
- [6] P. Erdős and A. Rényi. On the evolution of random graphs. In *Mat Kutato Int. Kozl*, 5, 17–60, 1960.
- [7] I. Gent and T. Walsh. The SAT Phase Transition. In *Proc. ECAI-94*, 105-109.
- [8] I. Gent and T. Walsh. The TSP Phase Transition. *Artif. Intel.*, 88:349–358, 1996.
- [9] T. Hogg, B.A. huberman and C. Williams. Phase transitions and the search problem. *Artif. Intell.*, 81:1-15, 1996.

- [10] Kirkpatrick, S. and Selman, B. Critical behavior in the satisfiability of random Boolean expressions. *Science*, 264, 1297-1301, 1994.
- [11] 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.
- [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.