Abstract

k-SAT is the very well studied restriction of the SAT problem which has attracted much attention due to its NP-complete complexity and close relation to many practical problems in Artificial Intelligence. Inherited from its parent, k-SAT is also NP-complete and many practical problems can be reduced to it efficiently (i.e. in polynomial time). In this text we introduce a class of problems, namely {2+p},{3}-SAT, obtained by restricting 3-SAT such that no variable occurs more than three times in an instance. We show that there is an efficient transformation which takes every 3-SAT instance to a {2+p},{3}-SAT instance.

The motivation behind this work is strongly enhanced by our intuition that the more we restrict the problem, the easier it will get to reveal the underlying structure of SAT and in particular, of 3-SAT. Moreover, we investigate whether there is a gain obtained by transforming instances of 3-SAT to {2+p},{3}-SAT, in terms of computational cost for satisfiability checking. We later show that although random {2+p},3-SAT is distinctly easier than random 3-SAT, transformed instances are particularly harder than their originating 3-SAT instances.

Contents

List of Figures

Acknowledgements

Above everything, Breanndán Ó Nualláin has been more than a supervisor to me throughout this journey. I'm grateful for his warm attention and contributions to me and my work.

The loved ones are Csilla Csukonyi & Taylan Tokmak for always encouraging me and giving a shoulder in my hard days, whenever I needed, they were there. Your support is priceless and unique. Thanks for everything.

I can not forget Kenan Ağtaş for his support and comments in solving my problems. I admire our friendship.

All the Weesperstraat-9 people were so warm and friendly. I find myself lucky that I got to meet you all.

To my parents
Hüseyin and Ümran
and also
my best friend and my sweet sister
Rahşan

You are my most valuable belongings in life, thanks for always being there.

From 3-SAT to $\{2+p\}$,$\{3\}$-SAT

Mehmet Giritli
ILLC, Amsterdam
The Netherlands

November 2001

1 Introduction

Revolutions cost many human lives combined with huge destruction, boiling water requires high energy, learning mathematics requires lots of effort, when recovering from a disease our bodies burn in high fever, the list goes on and on. It seems like that it is a permanent feature of nature which entails a high cost, in whatever form of it, for changing global features of its elements from one state to another. The question whether mankind will ever find a way through nature to avoid this feature remains unanswered.

In the case of logic, the first reminder of this feature lies in Satisfiability Problem (SAT). SAT is the NP-hard problem, given a propositional formula with n variables, to determine whether there is an assignment to its variables such that the formula is true. Many practical real-life problems can be efficiently reduced to SAT, hence the importance of the field is outstanding. Among many others, some popular problems[11] that can be solved in SAT are related to automated-reasoning, computer-aided design, computer-aided manufacturing, machine vision, database, robotics, integrated circuit design, computer architecture design and computer network design.

An instance of the SAT is a Boolean formula that has three components [5]:

1. A set of variables: $p_1, p_2, ..., p_n$.
2. A set of literals. A literal is a variable ($L = x$) or a negation of a variable ($L = \neg x$).
3. A set of m distinct clauses: $C_1, C_2, ..., C_m$. Each clause consists of only literals combined by just logical *or* ($\vee$) connective.

So, the goal of the SAT is to determine whether there exists an assignment of truth variables that makes the following Conjunctive Normal Form (CNF) formula satisfiable:

$$C_1 \wedge C_2 \wedge ... \wedge C_m, \tag{1}$$

where $\wedge$ is the logical *and* connective.

A special form of SAT is the class of problems which have a uniform clause length, that is to say, the SAT instances in which every clause contains a fixed number of literals, say k. We refer to this class of SAT problems as k-SAT, k being the uniform clause length.

Usually, when studying SAT, we restrict ourselves to k-SAT without the loss of complexity since for $k > 2$, it is known that k-SAT is NP-hard and moreover, it is NP-complete. For $k = 2$, k-SAT can be solved in polynomial time [6]. However, it shares some of its features with the other NP-hard problem classes in which $k > 2$, as we mention later.

One of the branches of the research in the field is to develop and improve algorithms which, search for satisfying assignments of a given formula (search algorithms) or answer whether a given formula is satisfiable (decision methods). In this context, the most crucial issue is to find challenging k-SAT instances so that we can measure how good these algorithms can actually perform. When the hardness of the real-life problems is considered, it is very obvious that finding challenging instances is a very important issue of this field. However, the method in analyzing the search algorithms is in contrast with the usual approach taken in computer science theory, which is based on the worst-case analysis [13]. In case of search algorithms, it is favorable to focus on the typical behavior rather than on a given single problem. Considering our limited knowledge about the nature of SAT, the best way to do this is to evaluate the algorithms using a randomized set of problem instances. To supply the necessary, usually random instances are generated according to a probability distribution, so that we can have an idea about the on average behavior of the algorithms. We call a randomly generated set of instances as an *ensemble* and instead of k-SAT we say *random* k-SAT to express the random generation process.

The issue of generating challenging formulas is highly related to the notion of *phase transition*[7]. Given an instance t, let n denote the number of distinct variables occurring in t and let m be the number of clauses in t. Now define α be the ratio of clauses to variables, $\alpha = \frac{m}{n}$. We will use these notations all the way in this text unless otherwise stated. In almost all random generation models (we explain this in detail in section 3), the instances are generated as the value of α is varied. Thus, we observe the properties of random k-SAT as the value of α is varied. One of the most important property we will be interested in is the *probability of satisfiability*. At the low values of α, the probability is almost always equal to 1. When the α is increased, at some point, it enters to a region in which the probability starts dropping from 1 towards 0. When α is increased enough to leave this transition region, the probability of satisfiability is almost always equal to 0 (fig.5). This change in the probability of satisfiability is known as the *phase transition phenomenon*, whereas the phases between which the transition occurs are the satisfiable and the unsatisfiable phases. Phase transitions exist in k-SAT, at different locations for different values of k. For $k = 2$, its location is theoretically shown to be at $\alpha = 1$ [6]. Unfortunately so far nobody could ever succeeded to determine the exact location of the phase transition for $k > 2$. Instead, there are several upper and lower bounds trying to approximate its location. The best bounds we know for random 3-SAT are $3.003 < \alpha < 4.758$ [8]. On the other hand, the experiments strongly suggest that the phase transition in 3-SAT occurs at $\alpha \approx 4.25$.

Now, let us enrich this picture with the computational cost issues. In almost all of the experiments done by many researchers, concerning the computational cost issues of k-SAT, the location where almost half of the formulas are satisfiable (i.e. when the probability of satisfiability is roughly 0.5) contributes the hardest instances of the the ensembles (fig.4). In the other two regions, where the probability of satisfiability is almost 1 and almost 0, the computational cost is distinctly lower. Thus, phase transitions characterize the high cost required when we are changing from one phase of state to another, just like the examples we gave in the very beginning. Nevertheless, for $k = 2$, although there is a phase transition occurring at exactly $\alpha = 1$, there is no region which its formulas require exponential cost. Actually, 2-SAT can be solved in polynomial time. So, the connection between the phase transition and the problem hardness holds only when $k > 2$ (i.e. when the problem is NP-complete).

Given the membership of 3-SAT in the NP-complete problems family, it is very interesting to consider equivalent problems which 3-SAT can efficiently be reduced to. The theory of NP-completeness tells us that any efficient solution to such a problem can be applied to not only 3-SAT but also to all NP-hard problems.

We define the notion for a problem class {x},{y}-SAT, whereas every member instance has a clause length of x and no variable occurs more than y times. For instance, {2},{3}-SAT is the problem class which every member instance has a clause length of two and no variable occurs more than three times.

Let us use m_3, m_2 for identifying the number of 3-length and the number of 2-length clauses in an instance, respectively. Being a NP-complete problem for $0 < p < 1$, {2+p},{3}-SAT is the restriction of 3-SAT whose instances has $m_3 = mp$, $m_2 = m - mp$ and no variable occurs more than three times. Note that as p is varied in its range from 0 ({2},{3}-SAT) to 1 ({3},{3}-SAT), it traces all the problem classes between and including {2},{3}-SAT and {3},{3}-SAT. Interestingly, it is theoretically proven that every instance of {3},{3}-SAT is satisfiable and can be solved immediately [4].

In section 4, we show that any 3-SAT instance can be efficiently (ie. in polytime) reduced to a {2+p},{3}-SAT instance by a satisfiability preserving transformation. We say that a transformation preserves satisfiability if whenever the originating instance is satisfiable, the transformed instance is satisfiable as well and vice versa. In this way we can safely concentrate on the transformed instances to find a satisfying assignment. This induces the main motivation behind this work. We have at least two perspectives looking at the desired gains from this transformation. One of them is to see whether the transformation makes satisfiability testing any easier than for 3-SAT. And the second perspective is to see whether we can get any insight into the 3-SAT instances, via their images under this transformation.

As mentioned before, the difference between 3-SAT and {2+p},{3}-SAT is that in a 3-SAT instance, there is no restriction on the number of occurrences of variables whereas in a {2+p},{3}-SAT instance, no variable may occur more than three times. In 3-SAT, it is very likely that one will find variables occurring more (sometimes much more) than three times. This is especially the case whenever $\alpha > 1$ since the average number of occurrences per variable is equal to 3α. Moreover, as the value of α is increased, the repetitions of the variables increase. At the high values of α, the average number of occurrences per variable is much higher than three. In section 4 we introduce the techniques which reduce the number of occurrences of variables occurring more than three times.

In section 2 we explain the algorithms and their general properties that we use in this work. Section 3 tells about the probability distributions and the models we use to generate random sets of formulas. Section 5 is about the exact-{2+p},{3}-SAT and {3},{3}-SAT classes of problems which can be used in order to make estimations about the global properties of {2+p},{3}-SAT problem class. Section 6 discusses the location of the transformed instances from

3-SAT in the {2+p},{3}-SAT space. Section 7 is dedicated to the computational cost issues of the {2+p},{3}-SAT and the transformation from 3-SAT. In section 8 we discuss the effects of our transformation on some very well known practical AI problems. Finally, we summarize in section 9.

2 Algorithms

Satisfiability checking algorithms are divided into two main streams depending on their approach in solving instances of SAT. Some algorithms always give a result, which can sometimes be very expensive to achieve, especially when the complexity of SAT is considered. This disadvantage of the algorithms caused some researchers to come up with the idea of semi-decision methods, where the determination of satisfiability is uncertain. In this section, we give a detailed description of the methods we use for our work in this text.

2.1 Completeness

We say that an algorithm is *sound* if every "yes, it is satisfiable" output is actually for a satisfiable input. If a sound algorithm can return "yes, it is satisfiable" for every satisfiable input then we say that the algorithm is *complete*.

Besides the popular attention complete algorithms attract, the application of incomplete algorithms for SAT problems still requires quite a lot of work. Most of the incomplete applications we have do very poorly in general, when compared to the complete methods. However there are a couple of exceptions which keep our hopes alive about this concept. In contrast, these exceptions perform much better in general than the best-performing complete algorithms we know. One way to see this difference in performance is via the experiments done with some well studied benchmark problems. When solving these problems, these incomplete algorithms perform much better than the all complete algorithms we know, on almost all of the problems included in the benchmark set [6, 22].

2.2 Complete Algorithms

Complete Algorithms, as mentioned before, have the advantage of being capable of determining satisfiability for any given instance in finite time, although it may require extremely large amount of time. The algorithm of Davis-Putnam in 1960 [17], which we will refer to as DP60, is actually the main source of inspiration for most of the algorithms available today. In DP60, a slight variation of the *resolution* technique was used. Roughly speaking, DP60 generates all of the possible resolvents based on a variable and then deletes all the clauses mentioning that variable. In return, this causes a sub-problem with one less variable but possibly exponentially more clauses. Only in 2-SAT, at the worst case scenario, this factor is quadratic.

Resolution is an important technique widely used in the satisfiability checking algorithms and so we explain it briefly. In a formula, when a variable occurs negatively in a clause and positively in another clause then we can represent these two clauses with one single clause built by adding up the rest of the clauses after the mentioned variable. This newly generated clause is called the *resolvent* and it preserves the satisfiability of the first two clauses whereas it has a total number of variables one less than the total number of variables in both of the initial clauses. A clause can be used more than once to generate a resolvent and thus if a variable occurs p times positively and n times negatively, clearly we can generate pn resolvents. Obviously, an empty resolvent implies that the formula is unsatisfiable since it can only be generated as a result of a contradiction in the formula. Figure 1 illustrates a situation where it is possible to apply resolution. Here, since the variable frequency is small, the negative effect of resolution is not felt. But, if y were to occur 5 times negatively and 5 times positively, this would entail 25 resolvents in the first step.

$$\left.\begin{array}{l}(x \vee y \vee z)\\(\neg y \vee q \vee z)\\(y \vee \neg x \vee p)\end{array}\right\} \Rightarrow \left.\begin{array}{l}(x \vee z \vee q)\\(q \vee z \vee \neg x \vee p)\end{array}\right\} \Rightarrow (z \vee q \vee p)$$

Figure 1: A possible case of resolution.

It was noticed that DP60 does very badly on satisfiable instances when compared to other available methods' performance. The reason behind this is that DP60 keeps generating lots of clauses in a situation where it is possible to find a satisfying assignment easily with other available methods. This caused some extra research on the procedure and in 1962 the improved version of DP60 showed up [16]. We will refer to this version as DP62. The main change in DP62 from DP60 was the introduction of the splitting rule (Implicitly mentioned before. Eliminating a variable at each step via resolution) instead of the elimination rule. This change in the algorithm causes the generation of two sub-problems at each step instead of one, but this time each of a smaller size. With this improvement, the algorithm may find a solution and quit earlier instead of continually generating clauses as resolution suggests.

Unless otherwise stated, we run DP62 for our evaluation needs in experiments presented in this text. Roughly, DP62 performs a depth-first backtracking search over the space of all possible assignments by at each step assigning a truth-value for a variable and simplifying the formula. The basic algorithm of DP62 is given in figure 2.

Given a set of clauses S defined over a set of variables V:

- If S is empty, return "satisfiable".
- If S contains an empty clause, return "unsatisfiable".
- (Unit-Clause Rule) if S contains a unit clause p, assign p the true value, and return the result of calling DP62 on the simplified formula.
- (Splitting Rule) Select from V a variable v, which has not been assigned a truth-value. Assign it a value, and call DP62 on the simplified formula. If this call returns "satisfiable", then return "satisfiable". Otherwise, set v to the opposite value, and return the result of calling DP62 on the re-simplified formula.

Figure 2: DP62: Based on the Davis-Putnam's algorithm still inspiring many others.

2.3 Incomplete Algorithms

2.3.1 Randomized Local Search

Given the time-consuming procedure structure as an obstacle for complete methods, alternative search algorithms keep appearing in the literature. One recent improvement to the history of satisfiability checking algorithms is the discovery of randomized local search approach. As a fundamental component of this approach, a cost function is defined over the set of truth assignments such that the global minima correspond to the satisfying assignments. The procedure starts with a initial choice of truth assignment for the formula in consideration (usually this is a random assignment). Then, the idea is to improve the current choice of truth assignment with another truth assignment in the neighborhood, such that it has a lower cost with respect to the cost function defined. Generally, the cost function is defined such that it takes a truth assignment to the number of clauses unsatisfied by that assignment.

In the classical notion of randomized local search, a termination occurs in the algorithm whenever there is no more improving step left. Actually this behavior is up to the implementation and for instance, GSAT, one of the well-known algorithms which applies the randomized local search method, keeps searching even if there is no available move left which can improve the cost (So, all of the available moves increases the cost).

The main problem with the above described method derives from the possibility that an algorithm may get stuck at a local minimum of cost function in the neighborhood of the initial guess. Eventually, such a situation will proceed with a restart of the algorithm from a different point by making a new guess for the truth assignment, possibly many times.

2.3.2 Iterative Repair

The aim in this method is to 'cure' the unsatisfiable clauses in a formula based on the approach that, if a formula is unsatisfiable then there must be a wrong assignment for at least one of the variables mentioned in the clause(s) which are unsatisfied. The main difference between the iterative repair and the randomized local search is that the search in iterative repair is based on some known 'ill' part of the formula and if it exists, a satisfying assignment will eventually always be found (although this may require a very expensive cost). Whereas in local search, this is not the case and an algorithm, even it exists, may fail to find a satisfying assignment within the parameterized bound of the maximum retries (guesses).

As we mentioned before, some rare implementations of randomized model finding algorithms perform much better than DP62 and other known complete methods but one should also keep in mind that the incompleteness of these algorithms is not always an acceptable trade-off.

3 Explored Distributions

3.1 Random 3-SAT

Probably this is the most studied problem class among the whole family of SAT problems, it has been under the spot lights for some time now. First of all, 3-SAT is the first NP-hard problem of the k-SAT problem family with smallest value of k. However, it is the most intractable problem when compared to other members of the k-SAT family with larger k. Moreover, it is NP-complete and this implies that any efficient solution to it can be applied to all of the NP-hard problems we know.

Usually, we generate random 3-SAT instances through the *fixed clause-length model*. Some researchers use the *constant-density model* in order to generate random 3-SAT formulas but, it is known that this model suffers from the lack of producing challenging formulas. We will not be using the constant-density model, hence our concentration on the fixed clause-length model. Given k, fixed clause-length model needs to be supplied with two parameters, as one of them characterizes the random k-SAT class which the generated instance will belong to, the other two are responsible to determine its 'location' in that class. With the most common notation these parameters are, n, the number of *available* variables, m, the number of clauses and finally k, the uniform clause length. As the later implies, every clause in a formula generated by this model has a fixed length, hence the model is called the fixed clause-length model. In constant-density model, for instance, instead of a parameter for a uniform clause length, a probabilistic parameter p is considered which stands for the probability of including a variable into a clause. Now, let us give a description on the instance generation process with fixed clause-length model.

In the fixed clause-length model, a clause is generated as a disjunction of k literals where each literal is obtained by first choosing a variable at random from the set of n available variables and then negating it with 50% probability. Repeating this procedure m times gives a random k-SAT formula with m clauses where each clause has k literals and at most n variables occur in the entire formula.

Now that we know how to generate random formulas of 3-SAT, we move on by showing how to pick a suitable ensemble for our experiments. An ensemble is a pair consisting of a problem together with a probability distribution. Fixing n and varying m (actually this can be done other way around, by fixing m and varying n, but we stick to our definition since it is more convenient) and by at each step (which we sometimes refer to as a data-point) generating a number of formulas results with a sample. Here, one should pay reasonable attention to the interaction of parameters and also consider the known features of the ensemble when defining the borders of a sample. In [8] it is shown that the literature contains remarkably many studies with inaccurate results caused by using unsuitable samples for their aims. The validity of an experiment is directly connected to the ensemble chosen in that experiment. For instance, the larger the size of each data-point, the better results we get. The same discussion holds for the size of the steps and the smaller the steps, the better results we get.

Earlier, we mentioned the ratio of clauses to variables, $\alpha = \frac{m}{n}$. The semantics underlying this ratio is extremely important for us. The basic meaning of α is the *average number of variables per clause*. So, as the value of α is varied, the variables become more or less dense in the formula. As they become denser, the constraints in the formula get stronger. But this has a direct effect on the satisfiability of the formula. Given this information about it, we observe the changes in the behaviors (ie. computational cost, probability of satisfiability, etc.) of the random k-SAT ensemble as a function of α.

First, we present the experiments where we duplicate some situations which are common to many studies in the literature [7, 8]. Figures 3, 4 and 5 show the results to these experiments. We used three ensembles, generated by fixing $n = 50, 100$, and 150 and then varying m from 100 to 450, 200 to 880 and 300 to 1300 for each n, respectively. In figure 3, each data-point is the mean behavior of 20 instances. Mean behavior is generally pulled up or down by some instances which exhibit extreme behavior. To prevent the influence of such instances on the ensemble, an alternative way of plotting data is to consider the median behavior instead of the mean as presented in figure 4. This generally gives a correction to the unexpected abrupt changes in the global properties of the observed.

As figures 3,4 and 5 show, when the computational cost is observed, at the low values of α, the instances are pretty easy to decide for satisfiability. Moreover, at these values almost all of the instances are satisfiable. As the ratio is increased, we enter to a region in which the computational cost to determine satisfiability abruptly increases to dramatically high values. In contrast to the easier region above, here the satisfiable and the unsatisfiable instances are mixed. This implies that a prediction concerning the satisfiability of instances from the hard region seems to be very difficult. After a sufficient amount of increase in the value of α, the computational cost drops back to manageable values (although slightly harder than the first region) and moreover, almost all of the instances are unsatisfiable. The observations we made above and many other studies suggest an easy-hard-easy pattern as a permanent symptom in random k-SAT distributions [21, 13, 11, 7]. As the problem size is increased, the computational cost in the hard region gets larger with an increasing rate of growth and this makes the instances from that region intractable with the techniques we know on proving formulas' satisfiability. Moreover, when the problem size is increased, the hard region gets narrower and steeper. On the other hand when we are looking at k-SAT with larger k, the hard region seems to shift to higher values of α with increased cost. Since this is not the main focus of our work, we do not present any material about the latter.

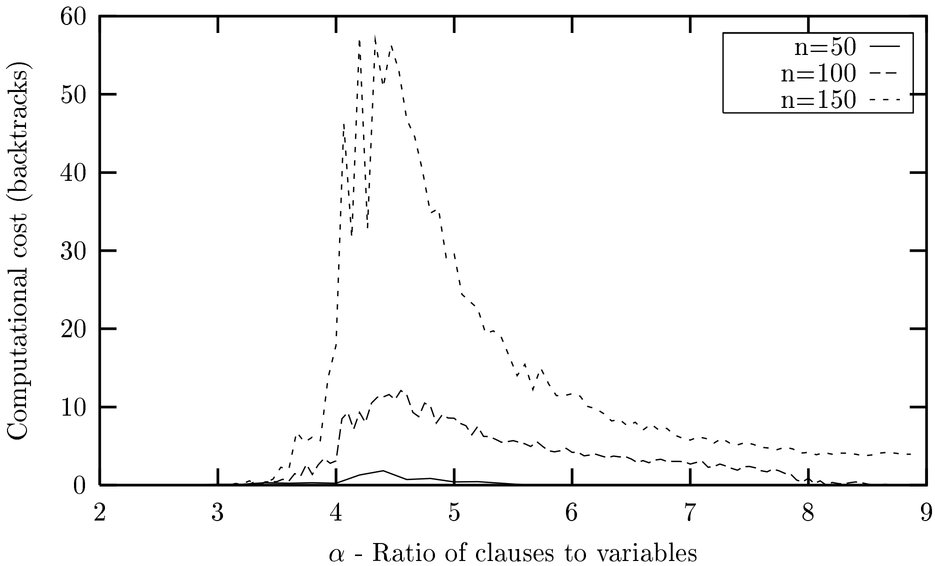

Figure 3: Mean computational cost for 3-SAT as a function of ratio of clauses-to-variables.

As we explained above, there is a change in the satisfiability of instances as α increases in which, instances become almost all unsatisfiable after a state where almost all instances are satisfiable. This well-experienced behaviour is known as the *phase transition phenomenon* where the two phases between which the transition occurs are the satisfiable and the unsatisfiable phases. A phase transition approaches the shape of a step function as the problem size gets sufficiently large. It is experimentally observed that the hardest instances in an ensemble occur at the location where almost the half of the formulas are satisfiable [20]. Thus, there is an underlying connection between the hard region of the easy-hard-easy pattern occurring in computational cost and the phase transition. This also explains that the narrowing of the hard region and the converging of the phase transition to a step function as the problem size increases

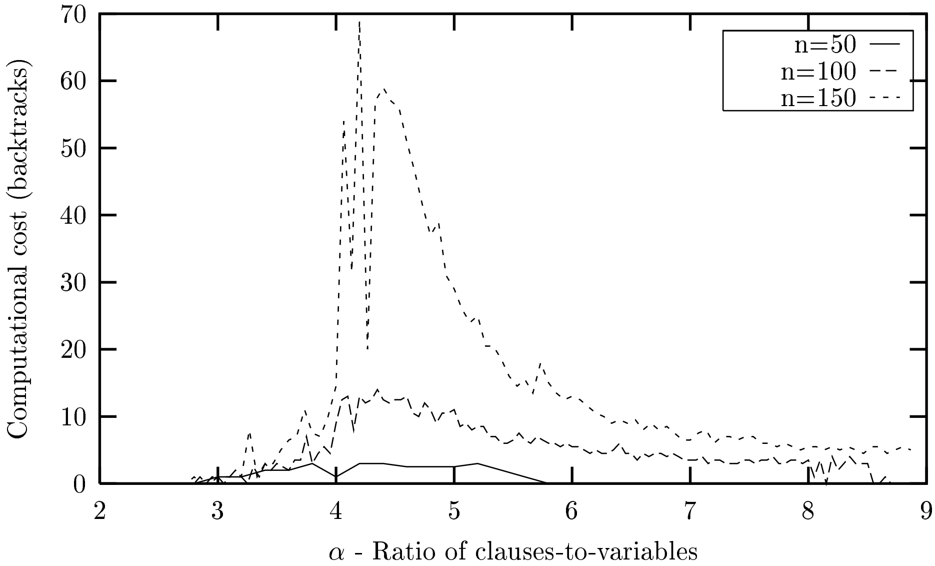

Figure 4: Median computational cost for 3-SAT as a function of ratio of clauses-to-variables.

is no coincidence. In general, the region where the phase transition occurs is the most suitable to find challenging formulas. Unfortunately there is no theoretical proof to the exact location of the phase transition or the hard region when $k > 2$. Instead, there are several approaches to determine the higher and the lower bounds of it. The best known bounds for the phase transition in random 3-SAT are $3.003 < \alpha < 4.758$ [7, 8]. A formal definition of the location of phase transition can be given as follows [21]:

Given k, the phase transition in random k-SAT occurs at α_k such that,

$$\lim_{n\to\infty} P(R_k(n, \alpha n)) = \begin{cases} 0 & ; \alpha > \alpha_k, \\ 1 & ; \alpha < \alpha_k. \end{cases} \tag{2}$$

whereas $R_k(n, m)$ the set of random k-SAT instances with n variables and m clauses and, $P(x)$ is the probability of satisfiability of a set of random instances x.

For 2-SAT, it is theoretically proven that the phase transition occurs at $\alpha = 1$ [6]. The phase transition seems to be a symptom for k-SAT in general however, the associated high cost for determining satisfiability only occurs when $k \geq 3$ which then the problem is NP-hard (moreover, it is NP-complete) whereas for $k < 3$ k-SAT can be solved in polynomial time and thus it is in class P.

The easy-hard-easy pattern in the computational cost figures can be explained by reasoning about the density of n variables in m clauses or, simply by reading the $\alpha = \frac{m}{n}$. When the ratio is at low values, the formulas are under-constrained and they enjoy the heaven of satisfying assignments since there are

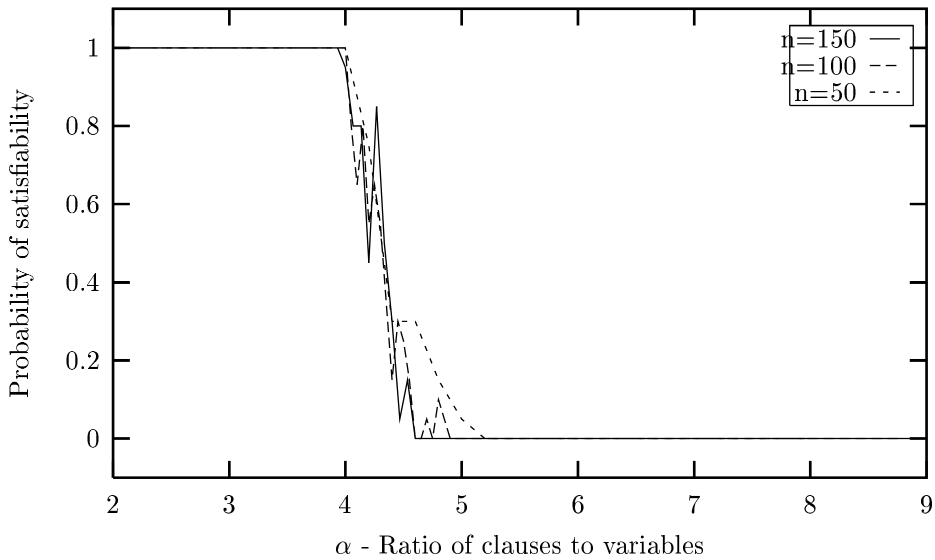

Figure 5: Phase transition phenomenon in 3-SAT.

plenty of assignments which can satisfy the formula. Thus a moderate satisfiability checking algorithm can find one of them very easily, most of the time just by guessing the assignment without any need of backtracking. We will refer to this location as the easy-SAT region.

As the value of α increases, the number of clauses increases but this on the other hand, increases the repetitions of the variables in the formula since the same amount of variables have to build a larger number of clauses this time. Hence, at the high values of α, the contradictions within the formulas are quite easy to find. In high values of α, this effect shows itself very precisely as almost all of the formulas are unsatisfiable. Hence, a moderate satisfiability checking algorithm can detect (for instance, by generating an empty resolvent) one contradiction in a relatively short time and return "unsatisfiable". We abbreviate this location as the easy-UNSAT region.

Finally we shed some light over the mysterious hard region and its counterpart phase transition effect. Since the value of α in this region is relatively large when compared to the the easy-SAT region, there are not that many satisfying assignments which can speed up the search procedure. On the other hand, α is small when compared to the easy-UNSAT region. This means that it is harder to find contradictions than in formulas in the easy-UNSAT region, which can help for an early exit from the procedure. As a result of the combination of these two factors explained, the procedure enters to an exhaustive search procedure causing a huge search tree and hence an expensive search cost. Thus, the formation of both the phase transition and the hard region is a result of the fine balancing between the number of variables n and the number of clauses m.

3.2 Random {2+p},{3}-SAT

Generating in the same way as 3-SAT but allowing 2-length clauses (clauses which are build up from disjunction of 2 literals) and restricting the maximum number of occurrences of variables to 3, one obtains {2+p},{3}-SAT distribution. As mentioned before, the parameter p, ratio of number of 3-length clauses to all clauses, determines the location of a {2+p},{3}-SAT distribution between {2},{3}-SAT and {3},{3}-SAT distributions. It is worthy to mention that one should be cautious with understanding the big distinction between {2},{3}-SAT, {3},{3}-SAT distributions and 2-SAT, 3-SAT distributions respectively, which are common in clause lengths but they are quite distinct distributions due to the limitation on the number of occurrences of variables in {2+p},{3}-SAT.

Eventually, the restriction on variables in {2+p},{3}-SAT instances to a limited number of occurrences brings a new shape to the ensembles from what we were used to with random k-SAT. Note that, allowing identical clauses, the α dimension of a random k-SAT ensemble can be arbitrarily large whereas in {2+p},{3}-SAT, the maximum value α can get is 1.5 at $p = 0$, as we see below. The factor which leads to this difference is the restriction on the occurrences of variables in {2+p},{3}-SAT. Since there is no such restriction in random k-SAT, bounds only appear when the identical clauses are disallowed. Nevertheless, even in this case, the bounds on the value of α are negligibly large as we show below in (3). In both random 3-SAT and {2+p},{3}-SAT distributions, the minimum value α can have is $\frac{1}{3}$ since for m clauses the maximum value n can have is $3m$. But in random k-SAT, in general, without any repetition of variables in clauses and disallowing identical clauses, given n, m and k, the maximum value that α can have is:

$$\alpha \leq \frac{(n-1)!}{(n-k)!k!} \tag{3}$$

In {2+p},{3}-SAT, the restriction of the number of occurrences of a variable to three means that with n variables one can obtain $3n$ literals at maximum. By definition, given m and p, an instance has mp many 3-length clauses and thus $m(1-p)$ many 2-length clauses. Hence we have the following:

$$3n \geq 3mp + 2m(1-p) \tag{4}$$

Some arrangements and simplifying gives:

$$\frac{3}{p+2} \geq \frac{m}{n} = \alpha \tag{5}$$

This equation says that α is bounded by a function of parameter p, and hence gives a perspective on the shape of the αp-space. Since p varies from 0 to 1, substituting them into the above equation gives that α reaches its maximum value when $p = 0$, which in that case $\alpha \leq 1.5$. As p increases, this upper bound on α decreases and we get the smallest bound when $p = 1$, which then $\alpha \leq 1$. This shows that the maximum value of α in random 3-SAT is much larger than

```
procedure generate_cnf (n m p) returns cnf;

  • let stack = generate_variables (n);

  • return generate_cnf_with_p (mp 3 stack) ⋃ generate_cnf_with_p
    ((m(1-p)) 2 stack);

procedure generate_cnf_with_p (m k stack) returns cnf;

  • let cnf = null;

  • for i = 1 to m do {

      – let clause = null;
      – for j = 1 to k do {
          * let literal = choose_random_literal (stack, clause);
          * let clause = clause ⋃ literal;
          * let stack = stack - (absolute literal);}
      – let cnf = cnf ⋃ clause;}

  • return cnf;

procedure generate_variables (n) returns list;

  • let l = null;

  • let l = build a list which contains three copies of each
    variable;

  • return l;

procedure choose_random_literal (list l, list c) returns literal;

  • let sign be minus with 50% probability, otherwise let it be
    plus;

  • let index = random (between 0 and length(l));

  • let var = element(l, index);

  • if var occurs in c then return choose_random_literal (l, c);

  • return (sign var);
```

Figure 6: Algorithm we implemented to generate random {2+p},{3}-SAT instances

the one in {2+p},{3}-SAT as we claimed before. Figure 23 at section 4 gives a good picture of the {2+p},{3}-SAT space.

The first series of experiments we did with this distribution family was a complete disappointment for our basic expectations, it seem to us that the results we had were far away from the point we wanted to reach. First we give a specification of the experiments and then explain what went wrong for us. The formulas we generated spread over α from 0.5 to about almost 1.5 by first fixing m[1] to $200, 500$ and then after setting $0 \leq p \leq 1$, varying $\frac{600p+400(1-p)}{3} \leq n \leq 400$ for the earlier and $\frac{1500p+1000(1-p)}{3} \leq n \leq 1000$ for the later value of m. At each data-point we generated 100 instances.

The main unacceptable negative effect exhibited was the the lack of unsatisfiable instances in all regions in both experiments, so almost all instances were satisfiable. Hence, contrary to our expectations for a phase transition effect, there was no phase of unsatisfiable formulas!. On the other hand, since we know that random 3-SAT is transformable to {2+p},{3}-SAT with a satisfiability preserving transformation, it follows that we should be able to find a region where the unsatisfiability reigns or more generally speaking, as many occurrences of unsatisfiable instances as satisfiable instances.

The results of this preliminary experiment gave us insight into designing a more focused experiment. From the first experiment where we set problem size $m = 200$, there were only two instances which were unsatisfiable, and from the second experiment, only one. With a very small difference between them, the value of α for two unsatisfiable instances, one from each experiment, were 1.351 and 1.333. The second unsatisfiable instance from the first experiment had an α of 1.428. As we will see later, the unsatisfiable instances in {2+p},{3}-SAT are densely packed in a relatively tiny region and thus in order to explore them, one needs an experiment with very finely adjusted parameters (with respect to k-SAT) but moreover, to achieve this, introduction of new parameters will also be obligatory.

3.3 The role of bipolar variables in {2+p},{3}-SAT

Having missed the UNSAT region in our first series of experiments, we focus on the definition of the distribution we used. The fact that 3-SAT is transformable to {2+p},{3}-SAT with a satisfiability preserving map implies the existence of unsatisfiable instances in {2+p},{3}-SAT. In order to find them, we concentrate on a factor which we believe to be effective on the discovery of unsatisfiable instances. The factor we will discuss about in this section is the percentage of variables which occur both negatively and positively. We call a variable *bipolar* if it occurs both negatively and positively in a formula.

When a variable occurs once or more but not bipolar, then we say that the variable occurs *trivially*. Obviously, trivial variables cause the formulas

[1] We mentioned before that in generating random k-SAT, it is convenient to fix n first and then vary m. However, in {2+p},{3}-SAT, because of the parameter p, it is easier to fix m first and then vary p. Nevertheless, this is completely safe.

in which they occur to be easier. Assigning a truth value to such a variable which evaluates the corresponding literal(s) to true will eventually make all of the clauses true in which they occur. So, after trivial variables, the search for a satisfying assignment will be done without the clauses which contained trivial variables. Actually, this method is implemented in most of the moderate satisfiability checking algorithms. The rule which cuts the clauses containing trivial variables off is called the *pure literal rule*.

Note that in a {2+p},{3}-SAT formula a variable must occur bipolarly in one of the configurations: (- +), (- + +) or (- - +). Obviously, a single occurrence is a trivial one.

The figure 7 shows the percentage of bipolar variables over all variables. The results were taken from our first series of experiments, discussed in previous section which we complained about the lack of unsatisfiable instances. Remember that the random generator we implemented negates any variable with 50% probability. The figure shows that with this method, one can get at most 75% of the variables to occur bipolarly (and the rest to occur trivially). Considering our experience with this method, we suspect that this factor plays an important role in generating samples with more unsatisfiable instances. With the current situation in the samples, we feel that the instances are easily satisfiable due to the vast number of trivial variables contained in them. If we can somehow force more variables to occur bipolarly then this will cause more constraints over the instances and hence, we might find more unsatisfiable instances in our samples in this way.

In figure 7 we see that at every value of α, when the value of p is increased, the percentage of bipolar variables over all variables gets larger. The obvious reason behind this is that, in the instances having same amount of variables and clauses but a different amount of 3-length clauses (hence, different p), because of the larger number of literals in the instances with larger p, the repetition of variables is more frequent. Hence, the chance that a variable occurs bipolarly is higher. Thus, instances generated with distributions of larger p, contains more bipolarly occurring variables than the ones with a smaller value of p. In the other dimension of the changes in figure 7, for all p, when the value of α is decreased, the percentage decreases as well. Actually, this is just the opposite case of the situation we explained above. When α, ratio of clauses-to-variables decreases, since m is fixed, this means that n increases. This increase in n causes the variables to occur less repetitively and hence the probability that a variable occurs bipolarly is also less.

The main hypothesis we form from this picture is that, it is a good possibility that the unsatisfiable instances are hidden under the role of the bipolarity factor. With this result, we change our method of generating random {2+p},{3}-SAT instances. Our procedure this time is as follows: First, we generate maximum number of literals that could be generated out of n variables, that is $3n$ literals, without assigning their polarity. Secondly, we negate one of the each three occurrences of every variable. At the end, we get a set of literals in which all of the variables occur bipolarly. Finally, we generate the clauses by choosing literals from this set at random. In this case, if a formula has a configuration

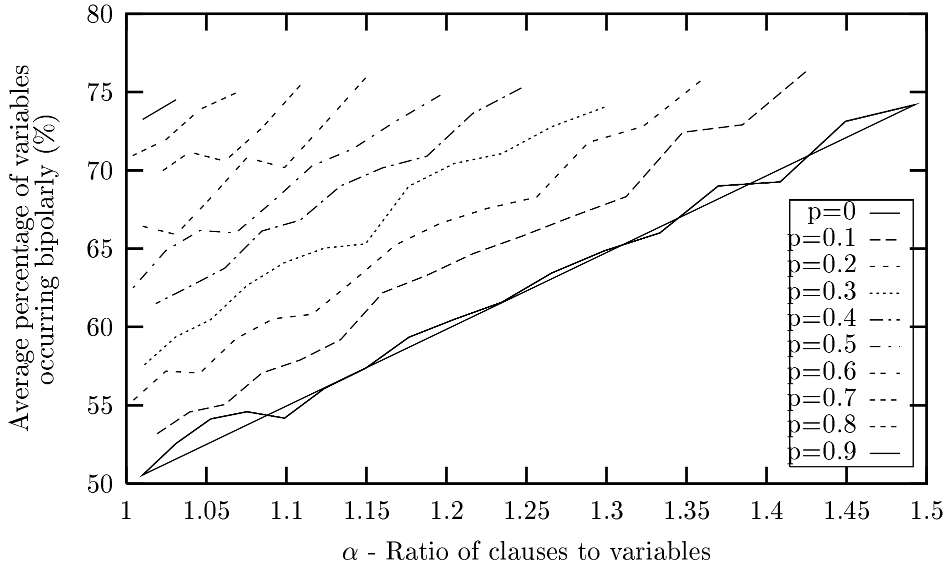

Figure 7: The figure shows that assigning the polarity of variables by 50% probability causes at most 75 percent of the variables to occur bipolarly, hence the rest to occur trivial.

in which all of the variables occurs three times, then all of them must occur bipolarly.

Figure 8 shows the effects of using the new distribution which we described just above to generate {2+p},{3}-SAT instances. This time we succeeded to discover the unsatisfiable instances of {2+p},{3}-SAT and indeed, the phase transition effect. Phase transition occurs at high values of α, around 1.2-1.4, very close to the maximum value which is 1.5. The story behind the formation of phase transition is similar to that of random 3-SAT. As α is increased, the repetitions of variables in the instances increases and this causes the contradictions to become more frequent. Whereas when α is decreased, the repetitions are less frequent and thus, the contradictions are rare, so most of the formulas are satisfiable. It seems that, in order to generate unsatisfiable instances of {2+p},{3}-SAT, forcing all variables to occur bipolarly is obligatory. Otherwise, given the limit on the number of occurrences of each variable, even the smallest value of n which causes maximum repetitiveness of variables (which is when all variables occur three times), can not induce constraints which are strong enough to make instances unsatisfiable.

Now let us concentrate on the behavior related to the change in parameter p. It is precisely seen that, as the value of p increases or, in other words, as the number of 3-length clauses increases, the phase transition effect shifts to the right, to higher values of α. Since the α dimension in a {2+p},{3}-SAT space is bounded by 1.5 from above, this means that the unsatisfiable region

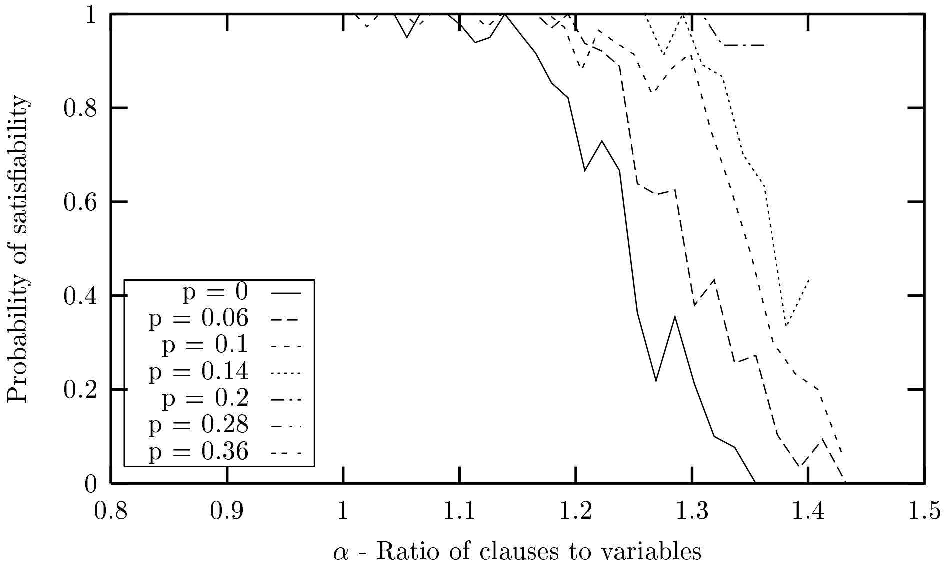

Figure 8: The phase transition in {2+p},{3}-SAT obtained by using a stricter distribution.

gets narrower as p increases. But this is not all. Consider equation (5), in which we saw that the maximum value of α is determined by a function of p, which decreases with increasing p. This implies that an increase in the value of parameter p causes a decrease in the highest value α can get. Hence, this makes the narrowing of unsatisfiable region even faster when p is increased. On the other hand, this shows that after some threshold value of p, we should expect the unsatisfiable region and the phase transition to disappear in {2+p},{3}-SAT, which is exactly what we see in the graph.

Now let us look at the shift of phase transition with varying p from another perspective. Exact-{2+p},{3}-SAT is the distribution which has the same properties with {2+p},{3}-SAT except that, with this distribution, every variable occurs *exactly three times*. We will get back to exact-{2+p},{3}-SAT later in detail but since it is not the primary issue here, we do not give any more details yet. Consider the {2+p},{3}-SAT instances which have a high value of α, where the number of variables is relatively small and repetitions are many and thus, most of the variables occur three times. But, as we just mentioned, exact-{2+p},{3}-SAT instances differ from {2+p},{3}-SAT instances only by the fact that *all* variables in exact-{2+p},{3}-SAT instances occur exactly three times. Thus, at high values of α, there is a close similarity between the instances generated with these two distributions, and we expect them to share almost the same global behaviors.

Now, consider exact-{3},{3}-SAT distributions(i.e. exact-{2+p},{3}-SAT with $p = 1$). Here, all variables occur exactly three times and in addition to

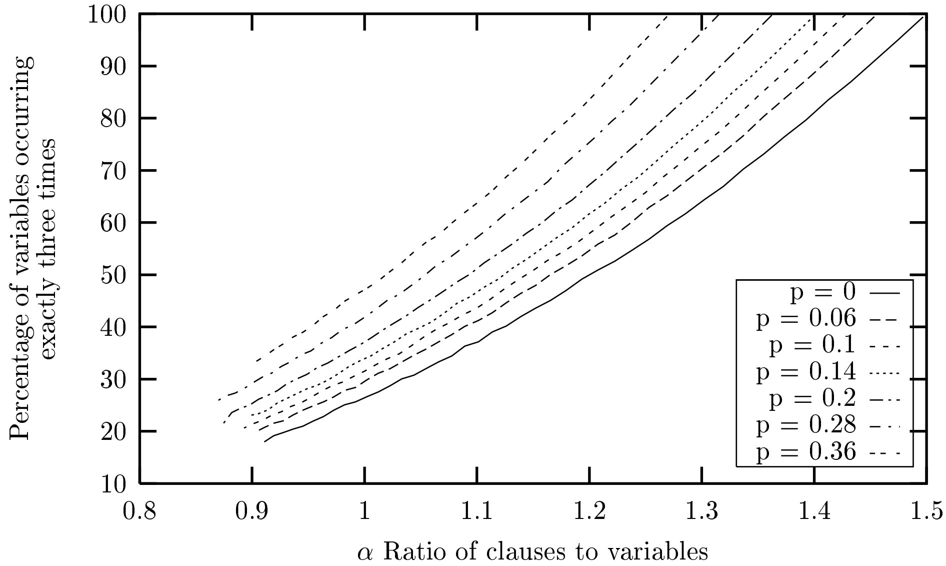

Figure 9: The percentage of variables occurring exactly three times against the ratio of clauses-to-variables, α.

this, all clauses have a clause length of three. The interesting thing about this distribution is that, it is theoretically proved that, *every* instance generated with this distribution is satisfiable [4]. So, in {2+p},{3}-SAT, as we increase $p \to 1$, the instances generated with a high value of α should be very similar to exact-{3},{3}-SAT instances. Thus, we should expect them to be almost all satisfiable. This gives another confirmation to the behavior observed in phase transition effect with increasing p.

But even with this shape, retrieving random {2+p},{3}-SAT distributions is problematic for us. As we will see later, when we transform instances from random 3-SAT to {2+p},{3}-SAT, we know that all instances are mapped at and around $p = 0.25$. So, we are very much interested in the ensembles of random {2+p},{3}-SAT generated at around $p = 0.25$. But figure 8 implies that the threshold value for p after which the unsatisfiable phase disappears, is smaller than 0.25, the last time we see an unsatisfiable instance is at $p = 0.2$. But this contradicts the observation that our transformation maps random 3-SAT instances to around 0.25. So, the threshold can not be as proposed by figure 8. In the next section we introduce a new distribution which causes the threshold on p to be corrected and with which, we can retrieve ensembles similar to the ones resulting from transforming random 3-SAT, in terms of satisfiability.

3.4 Random polarized-{2+p},{3}-SAT

In this section, we try to alter the distribution method we used so that we can generate ensembles more similar to the random 3-SAT images obtained by our transformation method. This suggests that, since we need a longer lasting phase transition effect as p increases, here we need a distribution which can generate more constrained instances of {2+p},{3}-SAT. On the other hand, the last option that can cause more constraints for {2+p},{3}-SAT is the possible manipulations we can make with the polarity of the variables.

Now consider the following: If a variable is bipolar, then the two occurrences with opposite polarity may occur both in 3-length clauses, both in 2-length clauses or they may occur separately, each in a different size clause. Intuitively, we suspect that the second case, where the occurrences of opposite polarity are contained in 2-length clauses, causes more constraints on the instances rather than the rest of the options. The reason behind this is simple. In other two situations, since there is a three length clause which contains one (or both) occurrence, it is easier to escape a conflict. To justify this hypothesis, we investigate how the bipolar variables are distributed over different-sized clauses with our current random {2+p},{3}-SAT distribution method.

Figure 10 shows the percentage of the bipolar variables that has at least two of its occurrences in 2-length clauses to all bipolar variables. Let us call the bipolar variables having at least c occurrences in k-length clauses as (k,c)-bipolar. At $p = 0$, obviously all of the bipolar variables are in 2-length clauses since there is no 3-length clauses for this value of p. As p increases, the number of 2-length clauses decreases and because of this, the percentage of (2,2)-bipolars decreases. This causes the distinct lines for each value of p as we see in the figure.

In general, when the value of α is increased, the number of bipolar variables increases overall. From the figure it seems that when the value of α is increased, the number of (2,2)-bipolars increase faster than the other bipolar variables. This is reflected in the figure by the slope in the lines for every value of p. Note that the latter behavior is only the case for when $p < 0.5$, when most of the clauses are 2-length.

As mentioned earlier, we are mainly interested in the situation around $p = 0.25$ since we know that our transformation method maps random 3-SAT formulas at and around this location. In figure 10 we see that at this location, only about 50% of the bipolar variables are (2,2)-bipolar. This implies that, if our intuition about the distribution of bipolar variables over clauses is correct, then we should expect quite a different behavior in {2+p},{3}-SAT ensemble when the current distribution of bipolar variables is altered such that more of them occur in 2-length clauses (i.e. more of them are (2,2)-bipolar). Moreover, in the instances transformed from random 3-SAT, we know that exactly one of the variables in the 2-length clauses occurs negatively. This means that another similarity between the transformed instances and randomly generated {2+p},{3}-SAT instances can be induced in this way.

To explore the validity of the above, we modify our distribution method and

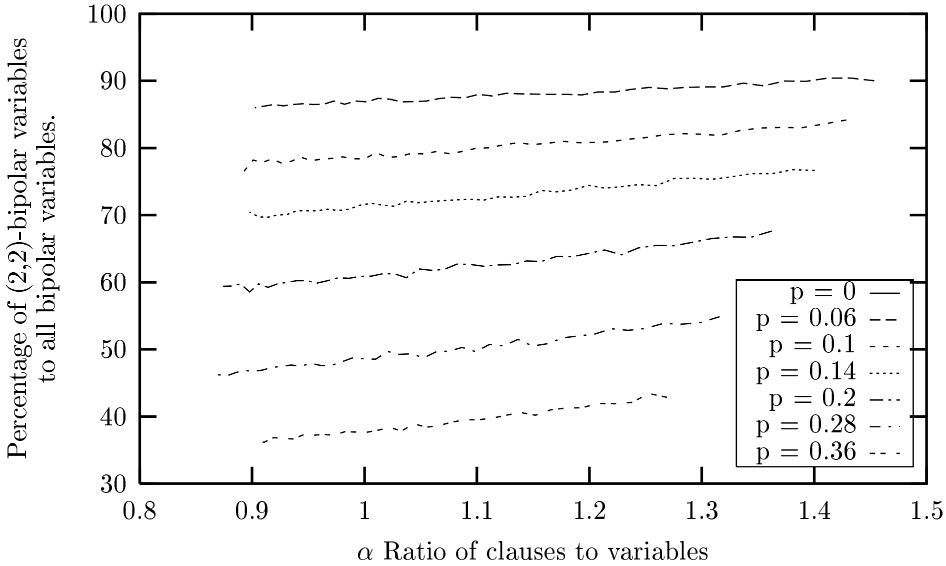

Figure 10: The percentage of (2,2)-bipolar variables over all bipolar variables as a function of α.

force more (2,2)-bipolar variables. The procedure can be described as follows: We first generate 3-length clauses and then the 2-length clauses at a certain number such that the ratio of 3-length clauses to all clauses is equal to p. First, we change our random variable selecting method. We generate three copies of the set of variables in which every variable occurs exactly one time, without assigning any polarity. Then, we select our variables starting from the first set and switching to the next one, as all of the variables in the current set have used once. In this sense, the variables chosen from the first set are the first occurrences of the variables in the generated formula, ones chosen from the second set are the second occurrences of variables and so on.

We negate the first and the second occurrence of a variable with 50% probability. As to the third occurrence, we check the polarity of the second occurrence. If the second occurrence is negative (positive) then we do not negate (we do negate) the third occurrence. This causes the second and third occurrences of a variable to always occur in opposite polarity. Since we first generate the 3-length clauses and then the 2-length clauses and moreover, if we consider the order we choose variables, we should expect that most of the variables occur (2,2)-bipolar rather than (3,2)-bipolar. We call the resulting distribution as random polarized-{2+p},{3}-SAT. Figure 11 shows the algorithm we use in the generation process.

Note that if the third occurrences are not needed in generation process[1] then

[1]The case in which this might happen is simply when the parameter n has a large value with respect to m.

procedure generate_cnf (integer n, integer m, float p) returns cnf;

- define array stack[3];
- for i = 1 to 3 do
  - let stack[i] = generate_variables (n);
- return generate_cnf_with_p (m*p 3 stack) $\bigcup$ generate_cnf_with_p ((m*(1-p)) 2 stack);

procedure generate_cnf_with_p (integer m, integer k, array stack) returns cnf;

- let cnf = null;
- for i = 1 to m do {
  - let clause = null;
  - for j = 1 to k do {
    - let index = select the smallest value from 1 to 3 which stack[index] is nonempty;
    - let literal = choose_random_literal (index, stack[index], clause);
    - let clause = clause $\bigcup$ literal;
    - let stack = stack[index] - (absolute literal);}
  - let cnf = cnf $\bigcup$ clause;}
- return cnf;

procedure generate_variables (integer n) returns list;

- let l = null;
- let l = build a list which contains one copy of each variable;
- return l;

procedure choose_random_literal (integer index, list l, list c) returns literal;

- let i = random (between 0 and length(l));
- let var = element(l, i);
- if var occurs in c then return choose_random_literal (l, c);
- select index;
  - case 1 or 2: let sign = minus with 50% probability, otherwise let it be plus;
  - case 3: let sign = minus if var has a positive negative occurrence otherwise plus;
- return (sign var);

Figure 11: Algorithm we implemented to generate random polarized-{2+p},{3}-SAT instances.

this method would not work so well. Nevertheless, we are not interested in the small values of α where n is large w.r.t. m but in the opposite situation where the n is small w.r.t. m and most of the variables occur three times. Figure 12 confirms what we have just said about the random polarized-{2+p},{3}-SAT distribution.

As α is increased, at low values, there is a decrease in the value of the percentage of (2,2)-bipolars among all bipolars. Then, for each value of p at a different location, there is a minimum point in which the value of the percentage starts to increase afterwards. The minimum point actually corresponds to the value of α which after that value, the third occurrences of the variables is used. When α is smaller than this value, all of the variables occurring in the instances must occur two times at most and moreover, each occurrence is negated with 50% probability. Thus, when the value of α is increased or in other words, when the value of n is decreased, the repetitions of variables increases but since there is no forcing on the polarity of variables at those values of α, bipolar occurrences mostly occurs in 3-length clauses. This explains the decrease in the percentage of (2,2)-bipolar variables until the minimum point. As we said above, after the minimum point, the third occurrences of variables begins to be used. This means that our method described in this section becomes effective. The forcing on the polarity of the third occurrences of variables causes the percentage of (2,2)-bipolar variables to increase. We see that at high values of α the method is really effective.

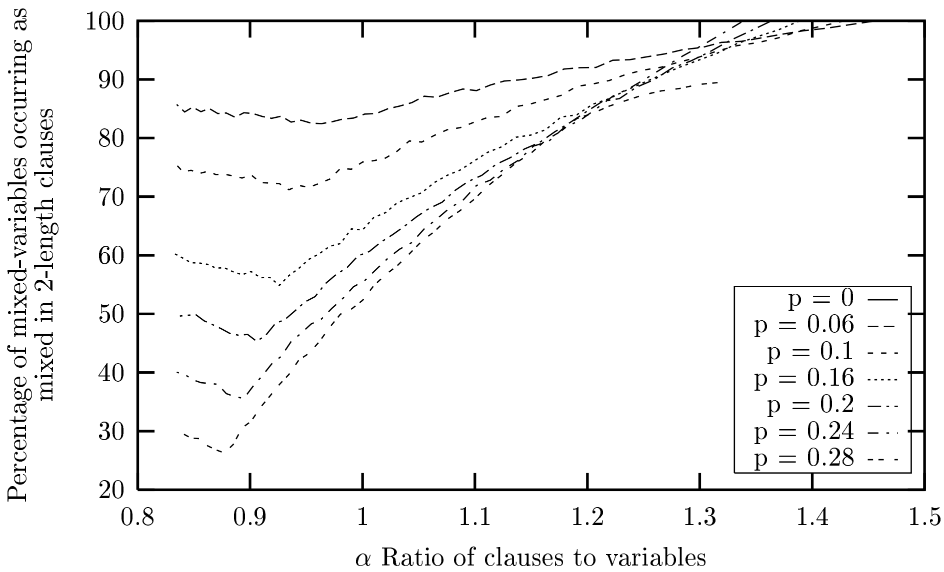

Figure 12: The new distribution of bipolar variables over different-size clauses. Percentage of (2,2)-bipolar variables as a function of α.

Figure 13 exhibits the change in the satisfiability of random polarized-

{2+p},{3}-SAT and it confirms our intuitions. With random polarized-{2+p},{3}-SAT, we see that the phase transition effect, again, shifts to the right with increasing p but this time, with much smaller steps. Thus, with random polarized-{2+p},{3}-SAT, now we have a longer lasting phase transition effect as p increases. Moreover, we see the existence of a phase transition effect in random polarized-{2+p},{3}-SAT for $p = 0.24$. Remember that our transformation maps random 3-SAT instances to values of p which are very close to 0.25.

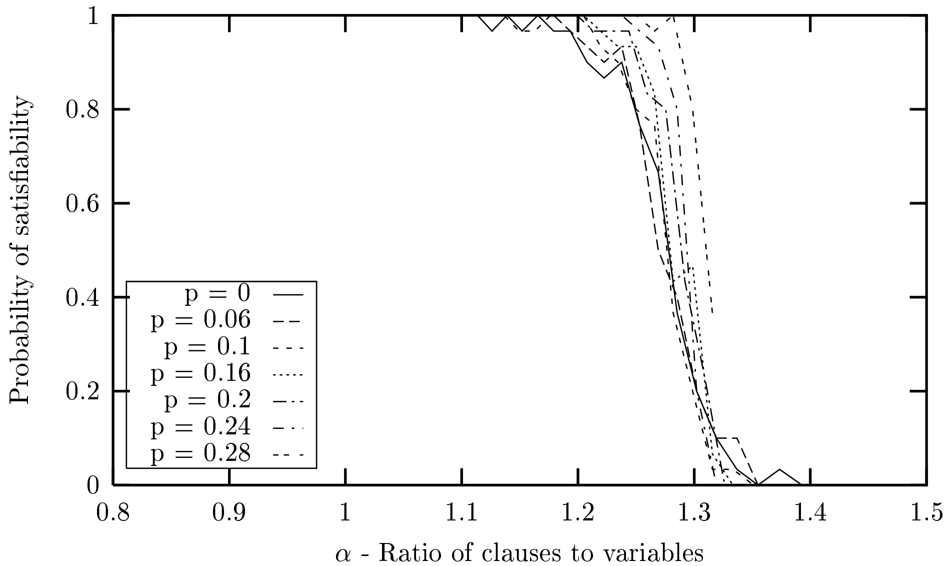

Figure 13: The phase transition in random polarized-{2+p},{3}-SAT.

4 Transforming 3-SAT to {2+p},{3}-SAT

A random 3-SAT instance t may contain at most $3m$ distinct variables and each variable may occur at most m times, whereas m is the number of clauses in t. It is possible to define a satisfiability preserving transformation which takes every 3-SAT instance to a {2+p},{3}-SAT instance. As we saw in section 2, no variable in a {2+p},{3}-SAT instance occurs more than three times. Thus, such a transformation must regulate the number of occurrences of each variable appearing in the formulas of its domain such that no variable occurs more than three times and the satisfiability is preserved. Such a mapping can be established easily by adding fresh variables to the formula in order to decrease the number of occurrences of variables as we explain below.

We say that two formulas t_1 and t_2 are *SAT-equivalent* iff:

t_1 is satisfiable if and only if t_2 is satisfiable.

Note that the SAT-equivalence is a distinctly different notion than the plain formula equivalence we know from propositional logic. When two formulas are SAT-equivalent, it does not necessarily follows that they are equivalent. In SAT, we are mainly interested in answering whether a formula is satisfiable or not. Hence, for us, SAT-equivalence is a more suitable notion than the plain equivalence. Now, we are ready to define our transformation.

Let t be a random 3-SAT instance containing m clauses and n variables. Call the set of n variables occurring in t Φ_t and let Φ'_t denote a set of fresh variables (i.e. not in Φ_t). For any variable $p \in t$, let $\Delta_{p,t}$ denote the number of occurrences of p in t. Partition the set Φ_t into two such that $\Phi_{t,\leq 3} = \{p | \Delta_{p,t} \leq 3\}$ and $\Phi_{t,>3} = \{p | \Delta_{p,t} > 3\}$. Hence, $\Phi_{t,\leq 3} \cup \Phi_{t,>3} = \Phi_t$ and $\Phi_{t,\leq 3} \cap \Phi_{t,>3} = \emptyset$. Now, let φ_t be a mapping from the set Φ_t to Φ'_t. Given $p \in \Phi_t$, define φ_t as follows:

$$\varphi_t(p) := \begin{cases} p & \text{if } p \in \Phi_{t,\leq 3}, \\ \Psi_{t,p} & \text{if } p \in \Phi_{t,>3}. \end{cases} \tag{6}$$

where $\Psi_{t,p} \subseteq \Phi'_t$ and $|\Psi_{t,p}| = \Delta_{p,t}$.

The main idea of the transformation is to substitute each occurrence of $p \in \Phi_{t,\leq 3}$ in t by an element of $\Psi_{t,p}$. If we do that and ensure the fresh variables added from $\Psi_{t,p}$ for each p are equivalent (i.e. all have the same truth value) then the resulting formula must be SAT-equivalent with the originating formula. First we describe the substitution process.

Let p_i be an enumeration of the members of the set $\Psi_{t,p}$ such that $0 < i \leq \Delta_{p,t}$. For each $p \in \Phi_{t,>3}$ and for each occurrence of p in t, if p occurs positively (negatively) then replace this occurrence of p by a positive (negative) occurrence of p_i for some arbitrary i and do not use index i for replacing any more occurrences. Repeat this process until p disappears from t and call the resulting formula t^T. This completes the substitution phase of the transformation. After this process is completed, next, we ensure that the variables in $\Psi_{t,p}$ for each p are equivalent. If a variable p had n occurrences in t, $n > 3$, our aim is to add the following in the transformed formula:

$$(p_1 \rightarrow p_2 \rightarrow \dots \rightarrow p_n \rightarrow p_1) \tag{7}$$

which is,

$$(p_1 \rightarrow p_2) \wedge (p_2 \rightarrow p_3) \dots \wedge (p_{n-1} \rightarrow p_n) \wedge (p_n \rightarrow p_1) \tag{8}$$

Using basic propositional logic it easily follows from here that the above is equivalent to:

$$(\neg p_1 \vee p_2) \wedge (\neg p_2 \vee p_3) \dots \wedge (\neg p_{n-1} \vee p_n) \wedge (\neg p_n \vee p_1) \tag{9}$$

Moreover, the later is in CNF and thus can be easily appended to t^T without any extra effort. Hence, appending this component to t^T for each p results in a formula which is SAT-equivalent to t and no variable occurs more than three times. This completes the definition of transformation from random 3-SAT to {2+p},{3}-SAT.

From above, we see where the 2-length clauses come from and the reason behind that. It is easily seen that for a variable p, occurring i times, transformation map introduces i fresh variables and adds i new 2-length clauses. This shows that a knowledge of the average number of occurrences of variables in random 3-SAT is crucial.

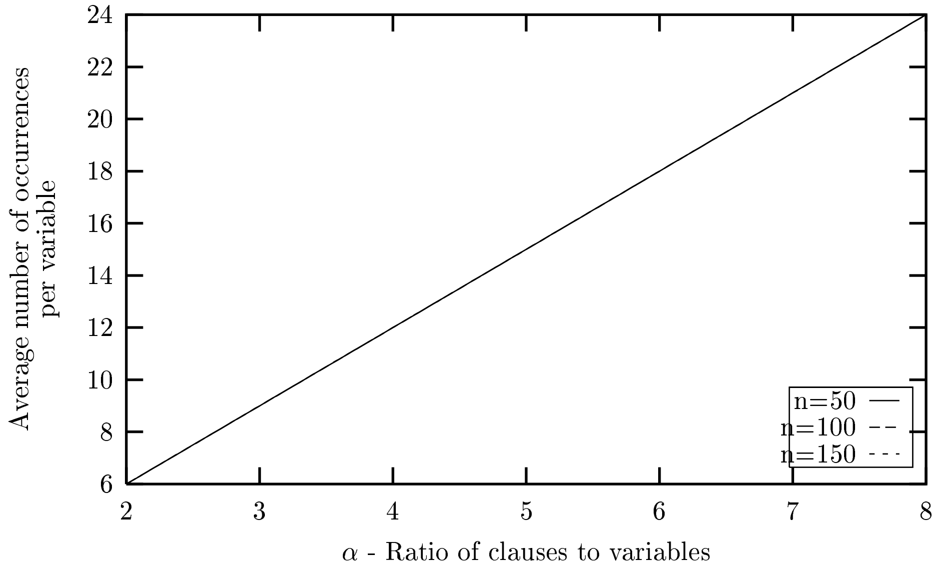

Figure 14: Average number of variable occurrences against α

From figure 14, it is easily seen that the variable occurrences increase linearly with the α. An increase in α by one causes the average number of variable occurrences to increase by three. Figure 14 is plotted for three different problem sizes, for each value of n in 50, 100, and 150. Thus, the figure also confirms that the algorithm we used to generate random 3-SAT has a uniform strategy in selecting variables. For a random 3-SAT instance t, which has a α ratio about 8, transformer introduces roughly 24 fresh variables and same amount of new 2-length clauses for almost every variable in t. If we were conducting an experiment using $n = 150$ variables, transforming an instance which roughly has $\alpha = 8$ requires 3600 new variables and 3600 2-length clauses. Using the average number of occurrences of variables, we can actually make more precise predictions about the resulting {2+p},{3}-SAT formulas.

One immediate question which rises at this point is whether we can predict the location of a 3-SAT instance in the αp-space after transforming it to {2+p},{3}-SAT. Note that equation 6 implies that, when a variable occurs three or less times then no change is done concerning that variable. Now let t be a random 3-SAT instance with n variables and assume that each variable in t occurs exactly r times. Then if m is the number of clauses in t:

$$m = \frac{nr}{3} \tag{10}$$

Now let t^T be the image of t under the transformation map and m_2^T, m_3^T be the number of 2-length and 3-length clauses respectively. Call the total number of clauses in t^T m^T. Given $r > 3$, we have that $m_2^T = nr$, $m_3^T = m = \frac{nr}{3}$. Since $m^T = m_2^T + m_3^T$, from here it easily follows that:

$$p = \frac{m_3^T}{m^T} = \frac{\frac{nr}{3}}{\frac{nr}{3} + nr} = \frac{1}{4} \tag{11}$$

This gives a good perspective on the expectations for p. Note that we got to this value by assuming that every variable occurs r times. So, in reality, the value of p should be somewhat different than 0.25. This is because every variable does not occur exactly r times. To see how it is, we conduct an experiment by transforming a random 3-SAT ensemble to {2+p},{3}-SAT. The random 3-SAT ensemble was generated using $n = 150$ and α was varied from 2 to 8. The figure 15 shows the change in p against the change in α. Now, we explain the reason behind the distribution of mapped instances in the αp-space as seen in figure 15.

Obviously, the smallest value n can take in a random 3-SAT instance is 3. Assume that each of those three variables occurs r times. If $r > 3$ then we add $3r$ new 2-length clauses. This gives a value of 0.25 to p. Now consider the following: The largest value n can take in a random 3-SAT instance is $3m$ which is the case when every variable occurs only once in the entire instance. In this case, such a formula is already a {2+p},{3}-SAT instance and obviously the value of p is 1 since $m_3^T = m^T$. This shows that as the variables repeat less, the value of p increases. But in practice, the case in which $p = 1$ hardly occurs since the instances in which all the variables occur once are rare.

It is seen that the value of parameter p is hardly greater than 0.27, which is the case at the low values of α. Moreover, as the α increases, p strongly converges to 0.25. The reason behind this is explained later in this section.

Let us refer to the ratio of clauses-to-variables of the transformed instances as α^T. Now we try to make a prediction for the value of α^T. Let t be a random 3-SAT instance with m clauses and n variables. Assume that every variable in t occurs exactly r times and that $r > 3$. So, all the variables in t will be replaced by rn many fresh variables and rn many new 2-length clauses will be added by the transformation process. Since we assumed that the n variables occur uniformly, we must have $m = \frac{nr}{3}$. Thus, we must also have $m^T = \frac{rn}{3} + rn$. From here we have:

$$\alpha^T = \frac{\frac{rn}{3} + rn}{rn} = \frac{4}{3} = 1.333 \tag{12}$$

Figure 16 shows the results concerning the change in ratio of clauses-to-variables in the experiment we mentioned above. Roughly speaking, the broad scale of α in 3-SAT is mapped to a relatively small region in {2+p},{3}-SAT, having a width of 0.018. Our prediction for α^T above seems to be quite accurate

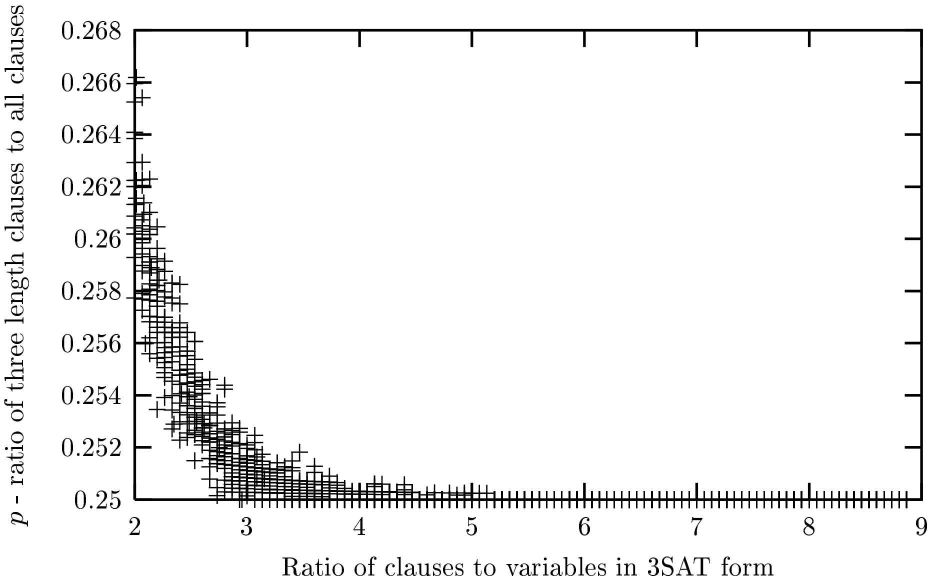

Figure 15: The distribution of parameter p as the α changes.

since we see that most of the instances are gathered around 1.334. This is especially the case when the value of α is above 4. For $\alpha > 4$, almost all of the instances are mapped to almost the same α^T in {2+p},{3}-SAT.

Above, we promised to explain more about the shape of figure 15. Actually the argument about this figure and the last one, figure 16, is the same. What we have in common for both situations is that, as the value of α gets larger, the corresponding results get very close or the same with our theoretic predictions. In figure 15, when $\alpha > 4$, the value of p is almost always 0.25. As to the figure 16, when $\alpha > 4$, α^T is almost always 1.333. What happens here is that, when $\alpha > 4$, there is almost no variables occurring less than three times. As a result, 3-SAT instances are mapped to {2+p},{3}-SAT instances which has almost all of their variables occur exactly three times, so they occur very uniformly in number. This explains the reason behind the similarity between the predicted values and the observed values, when the assumption we made for uniform variable occurrence, referred by r, is considered.

5 Two helpful distributions: exact-{2+p},{3}-SAT and {3},{3}-SAT

We have already mentioned in this text several times that exact-{2+p},{3}-SAT and {3},{3}-SAT might prove useful in a couple of occasions. One of those occasions was about the shifting of phase transition effect with increasing p and then disappearing after a threshold value on p. We said that this behavior can be

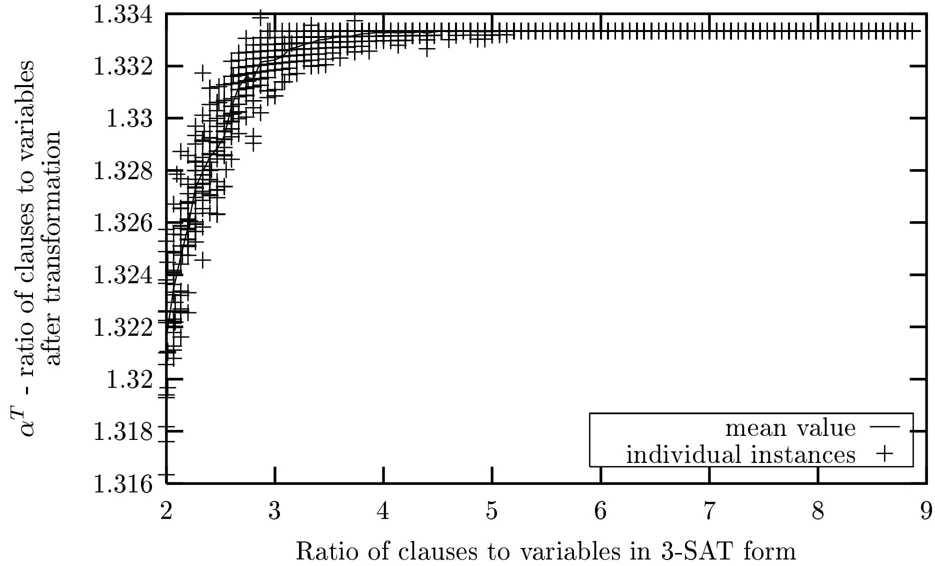

Figure 16: The change in the α after transformation.

confirmed by using {3},{3}-SAT distributions as well. In the other occasion, we claimed that the image of a random 3-SAT ensemble under the transformation will always be very similar to a exact-{2+p},{3}-SAT ensemble. In this section we give some insight into these claims and search for proof of their validity.

Letting $p = 1$ for a {2+p},{3}-SAT distribution results with {3},{3}-SAT distribution. By definition, all of the clauses in an instance generated using this distribution has a length of three and no variable occurs more than three times in the entire formula. Although it is obtained easily from {2+p},{3}-SAT by just letting $p = 1$, this distribution has some other features not offered by {2+p},{3}-SAT when $p < 1$. The first nice thing about {3},{3}-SAT is that it can be solved immediately. This rises from the following claim [4]:
Let $SAT_{\leq k}$ be any SAT problem in CNF which has the following properties:

1. every clause consists of exactly three literals (ie. clause length is three),
2. no variable occurs more than once in a clause,
3. no variable occurs more than k times in the entire instance.

Then every instance of $SAT_{\leq k}$ is satisfiable.

Hence it follows that every instance of {3},{3}-SAT is satisfiable.

Remember that both in random {2+p},{3}-SAT and random polarized-{2+p},{3}-SAT we experienced a decrease in the size of unsatisfiable region and moreover, after some threshold value on p the whole unsatisfiable region disappeared. As p is increased, naturally, both {2+p},{3}-SAT and polarized-{2+p},{3}-SAT approach {3},{3}-SAT. Hence, as $p \to 1$ the probability of satisfiability goes to 1. But this is exactly what we see with {2+p},{3}-SAT and polarized-{2+p},{3}-SAT.

Now let us concentrate on exact-{2+p},{3}-SAT. This is the restriction of {2+p},{3}-SAT in which every variable occurs exactly three times. Remember that when we transform instances of random 3-SAT, every variable in the 3-SAT instance which occurs more than three times gives birth to several new variables in the image instance which all occur exactly three times. As to the variables occurring less than or equal to three times, they remain untouched. So, the number of variables occurring less than three times in the transformed formula is determined by the number of variables occurring less than three times in the 3-SAT instance.

To get a perspective on this, use the Possion distributions. The Possion formula which we state just below (equation 13) gives the probability of an event to occur, given the average frequency of occurrence of that event.

$$P(n) = \frac{\lambda^n}{n!e^\lambda} \tag{13}$$

where λ is the average frequency of occurrence. From here the probability that a variable occurs less than three times is,

$$\begin{aligned} P(\text{a variable occurs} < 3 \text{ times}) &= P(1) + P(2) \\ &= \tfrac{\lambda}{e^\lambda} + \tfrac{\lambda^2}{2e^\lambda} \\ &= \tfrac{2\lambda+\lambda^2}{2e^\lambda} \end{aligned} \tag{14}$$

From figure 14 we know that in random 3-SAT, the average number of occurrences per variable is about three times the value of the α. Plugging in the information that $\lambda = 3\alpha$, we get the probability of a variable to occur less than three times as a function of α. Figure 17 shows the probability of a variable to occur less than three times as a function of α.

Figure 17 shows that in random 3-SAT, except extremely low values of α, almost no variable occurs less than three times. This shows that the image of random 3-SAT is actually very close to exact-{2+p},{3}-SAT. Note that for extremely low values of α, we already know that almost all instances are satisfiable. So, this also shows that in terms of satisfiability, the observed behavior in figure 17 does not cause any distinctions between these problems.

Now consider generating random exact-{2+p},{3}-SAT ensembles. The procedure for this distribution is a bit different with the other distributions we have been discussing so far. To see that consider the following:

Given p, m and n, since we know that every variable occurs exactly three times, we must have $3n$ literals such that:

$$3mp + 2(m - mp) = 3n \tag{15}$$

with some simplification and substituting $\alpha = \frac{m}{n}$, we get

$$\alpha = \frac{m}{n} = \frac{3}{p+2} \tag{16}$$

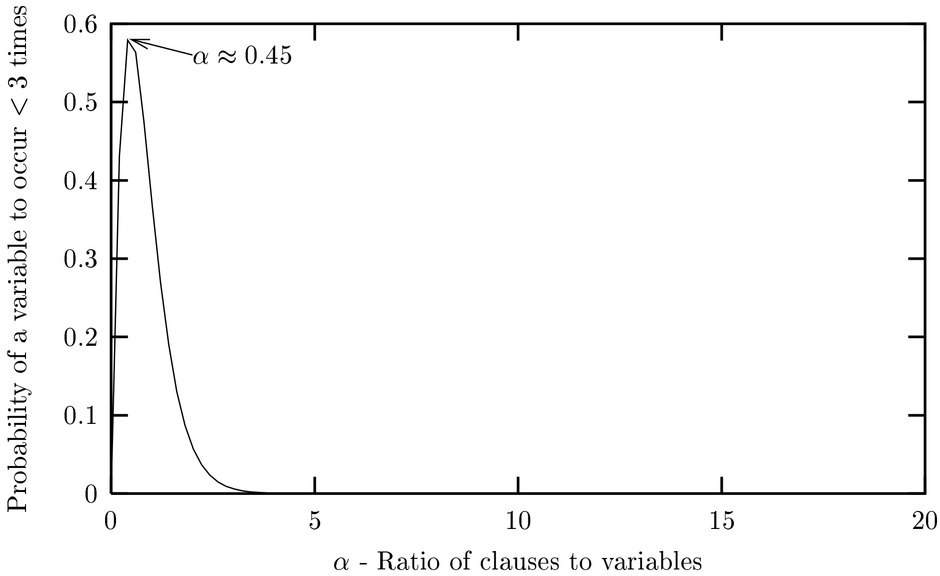

Figure 17: The probability of a variable to occur less than three times using the Possion distribution.

Equation 16 shows that, in exact-{2+p},{3}-SAT the parameter α is actually a function of parameter p. Thus, when generating random instances, it is better to fix m first and then generate instances by varying p. The value of n is simply pulled out of the equation above. Note that unlike in other distributions we discussed, the value of α can not be varied when p is fixed. This was our strategy with those other distributions. We give an update to the algorithm 11 in order to generate exact-{2+p},{3}-SAT instances (figure 18).

Figure 19 shows the probability of satisfiability in an ensemble, generated by exact-{2+p},{3}-SAT distribution, as a function of p. At low values of p, almost all of the instances are unsatisfiable. Instances keep being all unsatisfiable until $p = 0.26$. After this value, the probability of satisfiability begins to increase rapidly towards 1. The transition region is quite steep and it ends when $p = 0.3$. After $p = 0.3$, almost all of the instances are satisfiable. It is quite easy to say why the probability of satisfiability starts with an unsatisfiable region and ends with a satisfiable region, just opposite of what we used to in other distributions. As equation (16) shows, α is a function of parameter p and moreover they are in inverse relation. With this equation we know that a scale for p from 0 to 1 corresponds to a scale for α from 1.5 to 1. So this time we are looking at the α scale from other way around. Hence the given behavior in figure 19. At $p = 0.26$ the corresponding value of α is 1.327 whereas for $p = 0.30$, α is 1.304. These are the apparent boundaries of the transition region in figure 19. Considering the location of the transition regions in {2+p},{3}-SAT and in polarized-{2+p},{3}-SAT ensembles, exact-{2+p},{3}-SAT is very suitable

```
procedure find_best_approximation (integer m, float p) returns
integer;

  • if (isInteger ((m * (p + 2)) / 3)) then
      – return m;
  • else
      – find_best_approximation ((m + 1) p);

procedure generate_cnf (integer m, float p) returns cnf;

  • let m = find_best_approximation (m, p);
  • let n = ((m * (p + 2)) / 3);
  • define array stack[3];
  • for i = 1 to 3 do
      – let stack[i] = generate_variables (n);
  • return generate_cnf_with_p (m*p 3 stack) ⋃ generate_cnf_with_p
    ((m*(1-p)) 2 stack);
```

Figure 18: Algorithm we implemented to generate random exact-{2+p},{3}-SAT instances.

to represent these ensembles. Moreover, using exact-{2+p},{3}-SAT, a good approximation on the value of p which after the unsatisfiable region disappears, can be made. Figure 19 suggests that after $p = 0.3$, almost all of the instances are satisfiable.

6 A guide to {2+p},{3}-SAT

In section 2, we gave a deep analysis of {2+p},{3}-SAT distributions and found very similar symptoms to random 3-SAT. One of those similarities was that, as the value of α is increased, the formulas become more constrained and at some finely balanced ratio of clauses-to-variables, the phase transition phenomenon occurs in both problems. At low values of α, the formulas are under-constrained and thus almost all of them are satisfiable. Whereas in the higher values of α, they are over-constrained and this makes almost all of the formulas unsatisfiable. Between these two phases of satisfiability, there is an interval of ratios of clauses-to-variables, α, in which the probability of an instance to be satisfiable is around 0.5. Naturally, we expect this behavior occurring in random 3-SAT to occur in {2+p},{3}-SAT as well. Moreover, this time we have another player in the

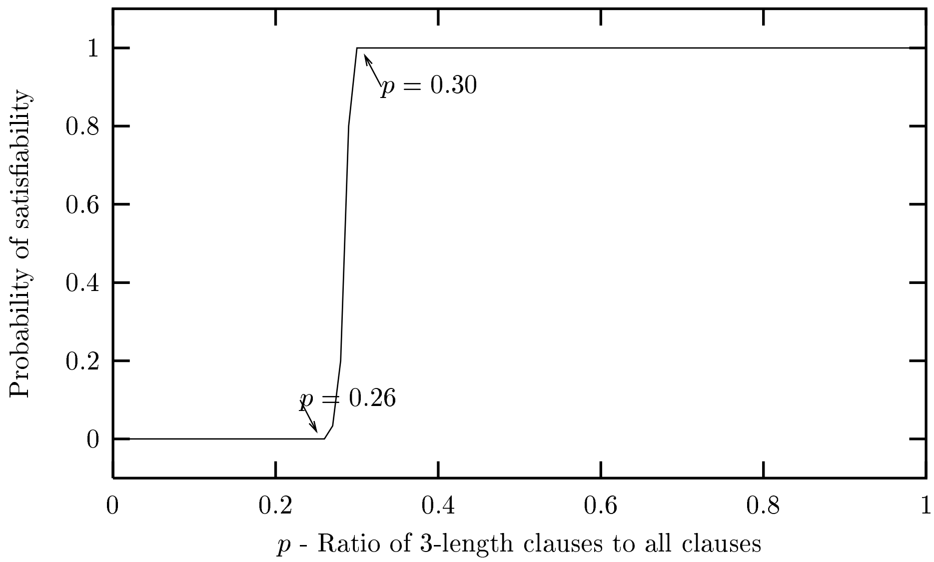

Figure 19: Phase transition in exact-{2+p},{3}-SAT.

game, namely the ratio of 3-length clauses to all clauses, p, which is very likely to affect this behavior.

We noted before that the broad scale of α in 3-SAT is mapped to a relatively small region in {2+p},{3}-SAT by the transformation. But we never talked about how the transformed instances are located throughout {2+p},{3}-SAT space. This is the motivation behind this section.

The figure 20 shows the results of an experiment in which we transform random 3-SAT to {2+p},{3}-SAT. Random 3-SAT ensemble was generated using $n = 150$ and α was varied from 2 to 8. We transformed instances of random 3-SAT to {2+p},{3}-SAT, recording the values of α, α^T and p for every instance. The figure shows that the relative order of values of α is preserved under transformation. In other words, we have $\alpha_1^T < \alpha_2^T$ iff $\alpha_1 < \alpha_2$. The lowest value we get for α^T is roughly 1.316, occurring for some rare satisfiable instances. As to the parameter p, when α is increased, the value of p decreases. The highest value p gets is 0.266, again, for some rare satisfiable instances. Both α^T and p approach to fixed values when $\alpha > 4$. The reason behind this is discussed in detail in section 4.

To get a better insight on the location of mapped formulas, we plot the same parameters in the above experiment for satisfiable and unsatisfiable formulas separately (figures 21 and 22). The satisfiable instances are definitely more divergent than the unsatisfiable instances. This can be easily explained since we know that unsatisfiable instances occur at high values of α and so, the variables in those instances are much more uniformly occurring than the lower valued instances, which are mostly satisfiable. Consistent with this argument, the only

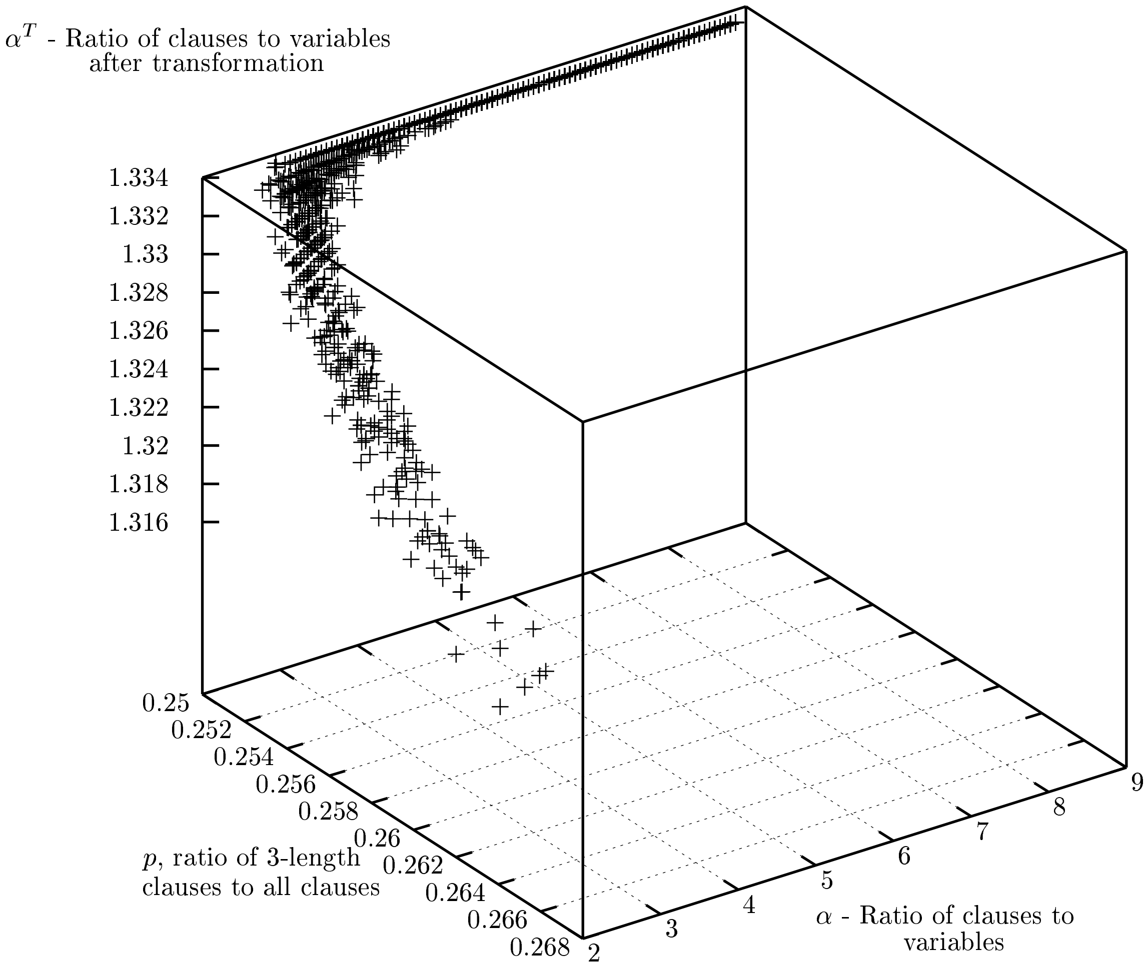

Figure 20: The location of the transformed formulas in αp-space is shown.

divergent instances which are unsatisfiable are from the transition region. But we know that these are the unsatisfiable instances having the lowest values of α. A very similar discussion to this one is in section 4.

Now let us step back and look at the big picture of {2+p},{3}-SAT. So far we gave very little information about the shape of the αp-space of {2+p},{3}-SAT ensembles. However, there is more to say about it. Figure 23 gives a good idea as to what it looks like. The αp-space in {2+p},{3}-SAT is bounded from both above and below. These boundaries can be identified with straightforward calculations on the parameters. Now consider the following: The maximum number of variables a {2+p},{3}-SAT instance may has is $2m_2 + 3m_3$, which is the number of literals in the entire formula. From here we get,

$$\alpha = \frac{m}{n} \geq \frac{m}{2m_2 + 3m_3} \tag{17}$$

Moreover we know that,

$$p = \frac{m_3}{m} \Rightarrow m_3 = mp \text{ and } m_2 = m(1-p) \tag{18}$$

substituting this in equation (17) we get,

$$\alpha \geq \frac{m}{2m_2 + 3m_3} = \frac{1}{2+p} \tag{19}$$

This equation gives the curve on the leftmost side of the figure 23, titled "lower boundary". This curve characterizes the lowest values α can get as a function of p. The construction behind the upper boundary is already done in section 3.2, equation (5). This gives the upper boundary appearing on the rightmost side of the figure 23. Parallel to the boundaries, there are two more curves. The one which occurs right next to the upper boundary is the curve which traces the exact-{2+p},{3}-SAT instances. To see that, consider the following: Since every variable in a exact-{2+p},{3}-SAT instance occurs exactly three times, the number of literals must be equal to three times the number of variables. So,

$$m_2 + 3m_3 = 3n \tag{20}$$

using (18), we get

$$m(2+p) = 3n \Rightarrow \alpha = \frac{3}{2+p} \tag{21}$$

This yields the curve marked "exact-{2+p},{3}-SAT". On the other hand, we wonder where the instances with less repetitive variables are located. To see this, consider exact-{2+p},{2}-SAT distribution in which every variable occurs at most two times. Repeating the equations 20 and 21 for this distribution yields,

$$m(2+p) = 2n \Rightarrow \alpha = \frac{2}{2+p} \tag{22}$$

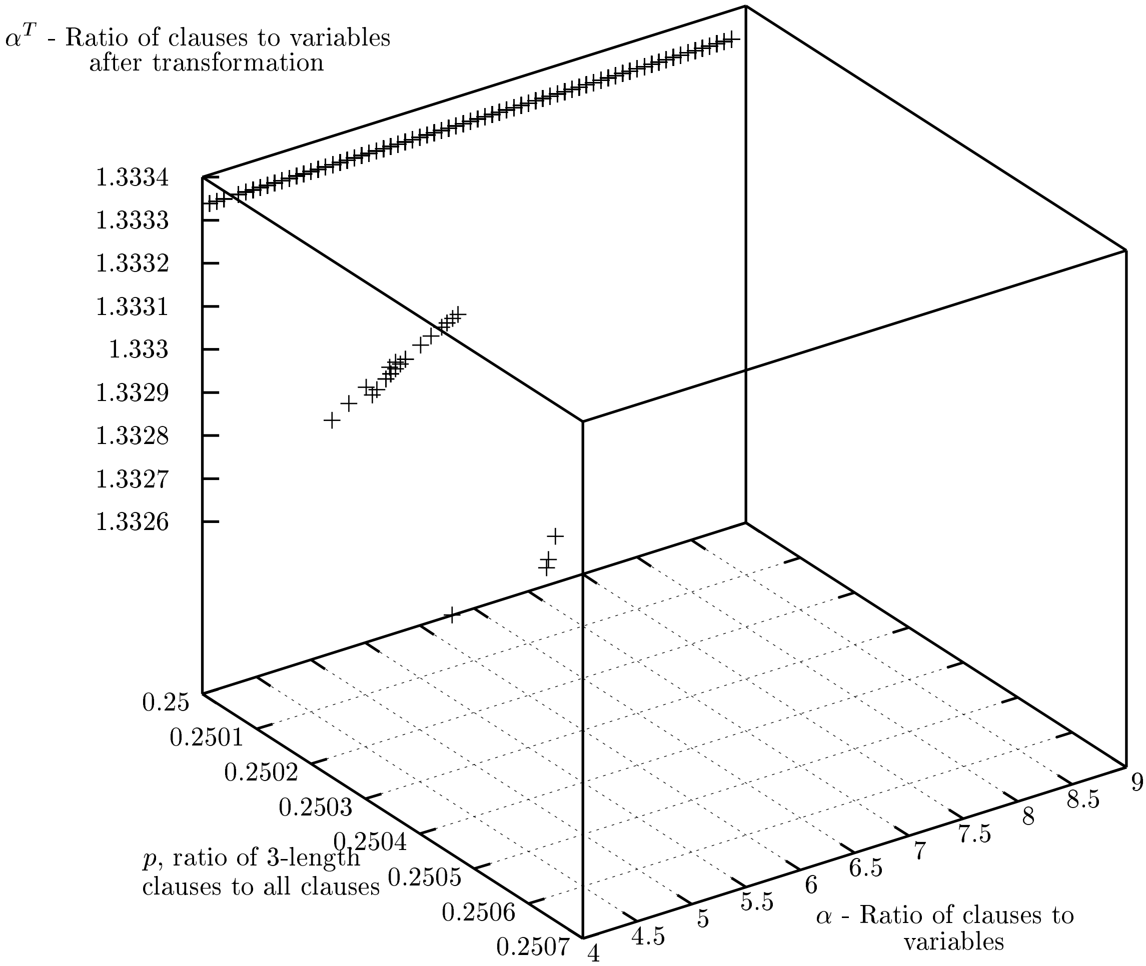

Figure 21: The location of the transformed formulas in αp-space is shown (only unsatisfiable).

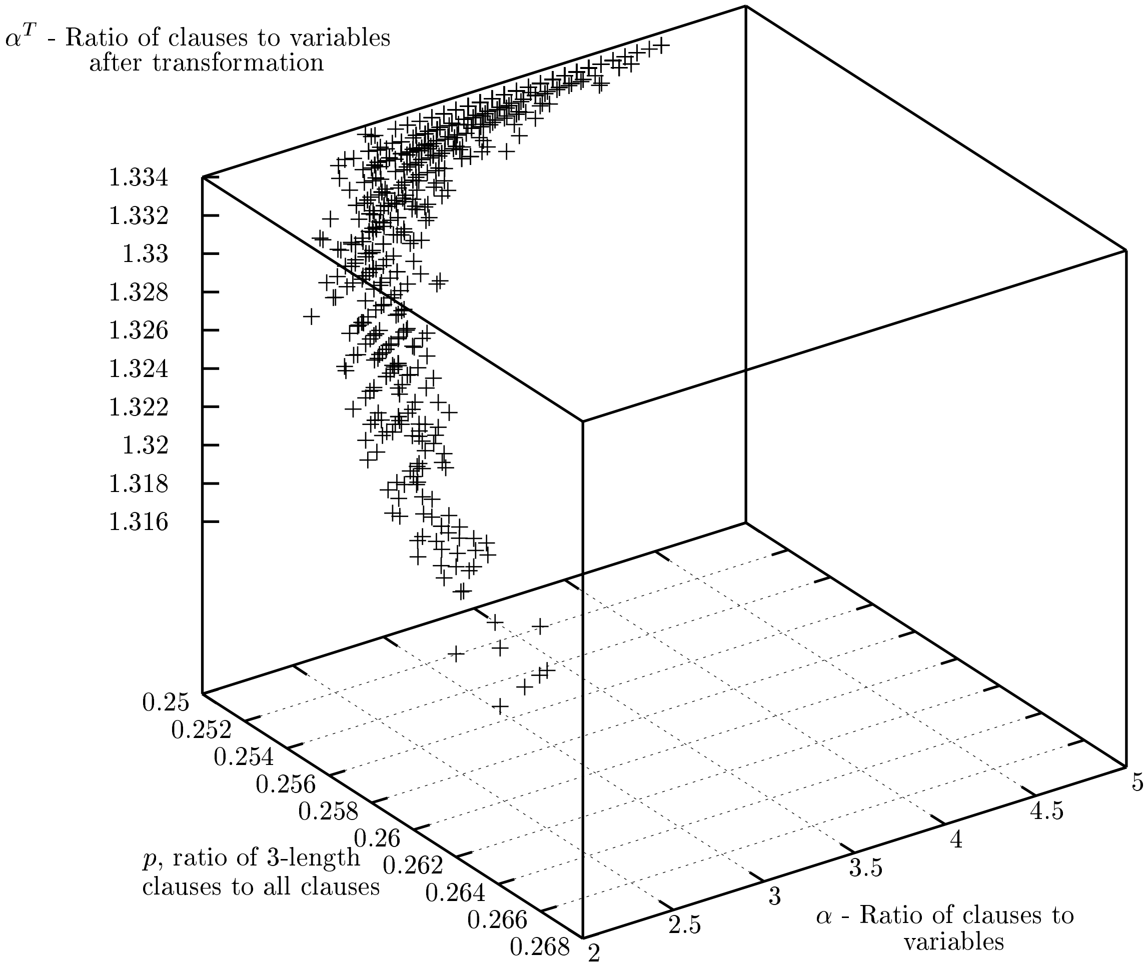

Figure 22: The location of the transformed formulas in αp-space is shown (only satisfiable).

This curve is drawn right in the middle of the two boundaries and is labeled "exact-{2+p},{2}-SAT". Note that it is a very natural result that the value of alpha decreases as the repetitions of the variables become less.

Finally we talk about the region where the images of random 3-SAT formulas are mapped. In figure 23, the curve labeled "Mapped 3-SAT" represents the location of these formulas. Note that the shape of the curve resembles the shape we encountered in figure 20. However, since we know the range of values of p and α over which the mapped instances are spread, the region of the curve which we are interested is marked with arrows. The equation of the curve is obtained as follows:

Let t be a 3-SAT instance with m clauses and n variables. Assume that every variable occur exactly r times. Assume that $r > 3$. Transforming the instance t into {2+p},{3}-SAT will produce a formula which has $m_3^T = m$, $m_2^T = nr$, $m^T = m_3^T + m_2^T = m + nr$ and $n^T = nr$. From here t has the following parameter values:

$$\alpha^T = \frac{m + nr}{nr} = \frac{\alpha + r}{r} \tag{23}$$

$$p = \frac{m}{m + nr} = \frac{\alpha}{\alpha + r} \tag{24}$$

From 23 and 24 it easily follows that,

$$p = \frac{\alpha^T - 1^T}{\alpha} \tag{25}$$

which is the equation we are looking for. Note that we do not say that all of the transformed instances are strictly mapped onto this curve. Since we assumed that every variable in 3-SAT instance t occurs exactly r times, the situation in reality is somewhat different. Nevertheless, it is a very good approximation to the real case and it gives a good idea as to where it occurs.

7 Computational cost issues

7.1 Computational cost of {2+p},{3}-SAT and polarized-{2+p},{3}-SAT

In this section we handle the computational cost issues of the {2+p},{3}-SAT and polarized-{2+p},{3}-SAT distributions. Remember that, in {2+p},{3}-SAT distribution, we choose the polarity of variables at 50% probability without making any distinctions as to how and where the opposite polarized occurrences of variables occur. We observed that with this way of generating random formulas, it is unlikely to get an unsatisfiable region in the random {2+p},{3}-SAT ensemble for values of $p \geq 0.2$. With the polarized-{2+p},{3}-SAT distribution, we forced variables to occur bipolar mostly in the 2-length clauses in order to get a longer lasting unsatisfiable region in the ensembles. We saw that with this distribution, it is possible to see an unsatisfiable region even for $p = 0.25$.

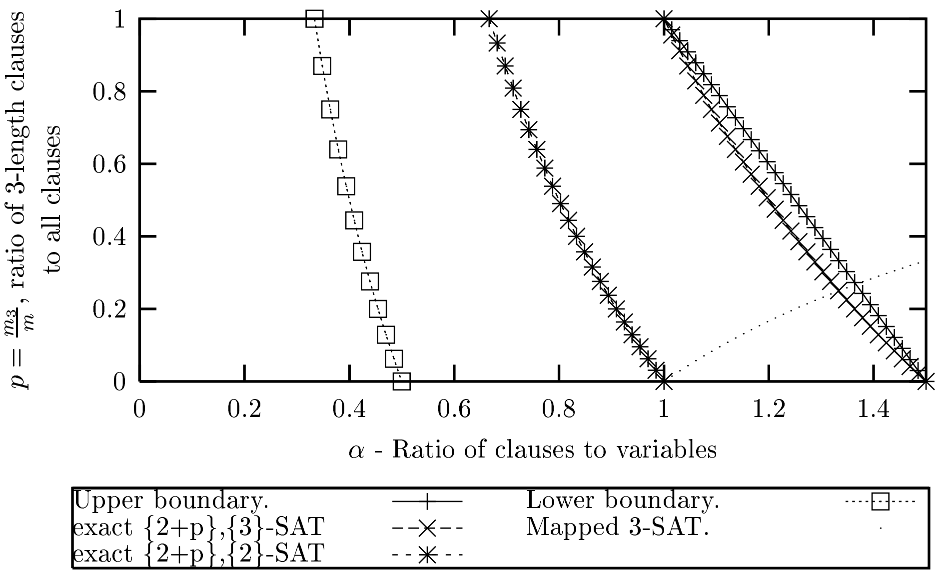

Figure 23: A map to the αp-space.

In this section, besides observing the cost issues for each distribution individually, we will check to see if there is a difference in terms of cost between these distributions.

In random k-SAT there is a very characteristic behavior occurring in the computational cost figures for values of $k > 2$. Remember that the random k-SAT ensembles are roughly partitioned into three regions. The first of them is the easy satisfiable region where the computational cost is the lowest of the entire ensemble. The reason behind this is the existence of numerous satisfying assignments for formulas in this region. Most of the time a moderate satisfiability checking algorithm will find one satisfying assignment for such a formula without any need of backtracks, merely by guessing the assignments for the variables. At the other easy region, where almost all of the formulas are unsatisfiable, the computational cost is relatively low, although it is slightly higher than the cost is the easy satisfiable region. This behavior can be explained by the frequent existence of contradictions in the formulas of this region. One of those contradictions can be detected by the algorithms in a relatively short time, lowering the cost. Finally, in the transition region of random k-SAT ensembles, we know that the computational cost for determining satisfiability increases to dramatically high values very quickly. This high difficulty in the middle of two easy regions is a result of the lack of both factors we mentioned above, namely the lack of contradictions and satisfying assignments.

Given this background information about the random k-SAT ensembles, we wonder about the situation in random {2+p},{3}-SAT. As we have seen earlier, there is a phase transition effect in {2+p},{3}-SAT as well, when p is roughly

≤ 0.2. In this case, shall we expect the same features of random 3-SAT in the computational cost for random {2+p},{3}-SAT? To answer these questions we present two figures (24 and 25) regarding the computational cost to determine satisfiability in random {2+p},{3}-SAT and random polarized-{2+p},{3}-SAT, respectively. Each figure represents one of the methods we used to generate {2+p},{3}-SAT instances. Interestingly, there was no instance generated by either method which required more than 1 backtrack. Hence, our concentration on the number of branches as a measure of computational cost.

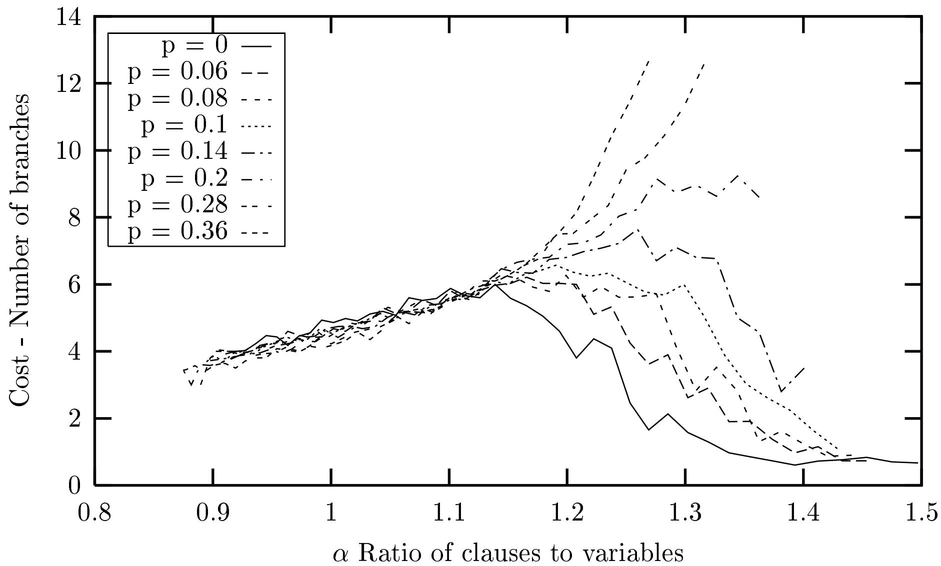

Figure 24: Computational cost in {2+p},{3}-SAT.

Figure 24 is obtained by generating the ensemble with our first distribution where in its ensembles, the unsatisfiable region disappears rapidly as p increases. Here, as p is increased, the cost increases accordingly. Moreover, with increasing p, the rate of growth in the computational cost gets larger. This is one of the features one can easily observe in random 3-SAT as well.

Now consider the shifting of the phase transition with increasing p as we see in figure 8, it is precisely the effect of this shifting which we see in figure 24 regarding the computational cost. As the p increases, the computationally hardest part is shifting to the right together with the transition region. We explained before that an increase in the value of p causes not only a shift in the transition region but it also pulls the maximum value of the α^T down from 1.5, which is the maximum value of α^T occurring at $p = 0$. When p reaches the maximum value, the maximum value for α^T is 1. This feature is seen in the cost figures as we see less of the easy-hard-easy pattern with increasing p. The last value of p in which we can observe the whole easy-hard-easy pattern is roughly at 0.14. After this value, at $p = 0.2$, we see the cost until the hard region and after that there is no more easy region. This is completely in accordance with the disappearing of unsatisfiable region when we were investigating the phase transition. Over 0.2, we do not see even the complete hard region but the only cost associated with the easy satisfiable region.

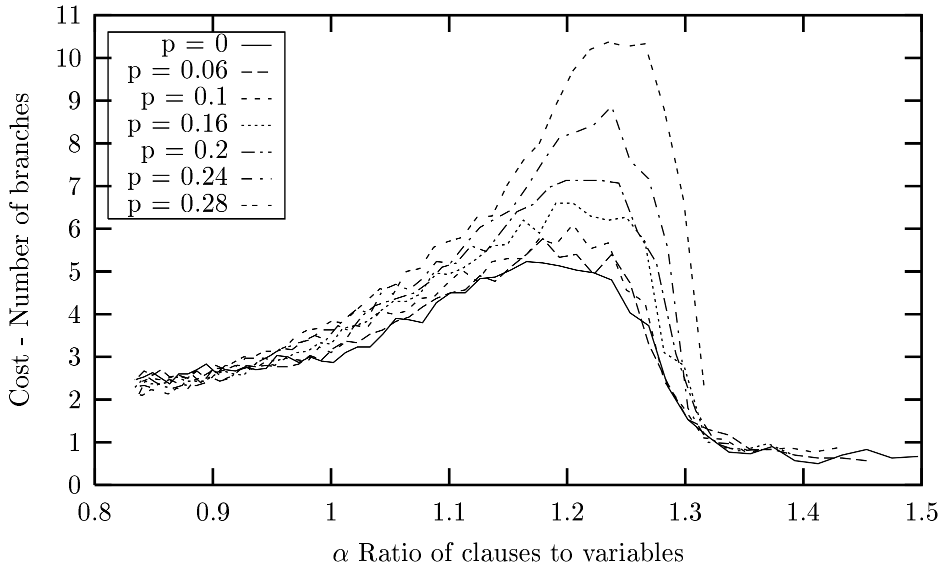

Figure 25: Computational cost in polarized-{2+p},{3}-SAT.

The figure 25 concerns the computational cost figures obtained from the ensembles generated using the polarized-{2+p},{3}-SAT distribution. Again, the results are in accordance with the general properties of random k-SAT. Remember that with polarized-{2+p},{3}-SAT, we found that the fast shifting of phase transition with the first distribution has slowed down and we got to see more of the unsatisfiable phase as p is increased. Thus, we naturally expect the reflection of this behavior in the computational cost figures. The figure 25 fully confirms our expectations. Contrary to the first figure about computational cost distribution, we got to see the whole easy-hard-easy pattern for values of p as high as 0.25. However, the figure shows that polarized-{2+p},{3}-SAT is actually not harder than {2+p},{3}-SAT but the contrary. At every value of p we tested, the polarized-{2+p},{3}-SAT instances in the transition region are slightly easier than {2+p},{3}-SAT instances. But this is just a difference by several branches.

In this section, we covered the computational cost requirements of {2+p},{3}-SAT distributions. We encountered differences in the shapes of the cost distributions but not a significant difference in the amount of the cost they require. The two figures we discussed about shows that with both distributions for generating random {2+p},{3}-SAT formulas, the hardest formulas required almost the same cost for determining satisfiability. But in general, the instances of random

$\{2+p\},\{3\}$-SAT are much easier than the instances of random 3-SAT. As we said before none of the instances required more than one backtrack, showing that cost required by the random $\{2+p\},\{3\}$-SAT instances is distinctly easier than random 3-SAT. As we have these results for random $\{2+p\},\{3\}$-SAT ensembles, we wonder about the hardness of the instances transformed from random 3-SAT. Note that the transformed instances are not 'natural' in $\{2+p\},\{3\}$-SAT as they are not randomly generated as random $\{2+p\},\{3\}$-SAT instances but they are 'engineered' to represent the instances of random 3-SAT. Well, is this going to affect the computational cost requirements of the instances belonging to the very same problem class? We discuss this in the next section.

7.2 Computational cost of the random 3-SAT image in {2+p},{3}-SAT

First, let's talk about what the cost figures can be or can not be like. Given the membership of $\{2+p\},\{3\}$-SAT in the NP-complete class of problems, this implies that, if one comes up with a efficient solution to it then this solution can be applied to all of the NP-hard problems entailing $P = NP$, which ends the hopes to find a polytime solution to such images of 3-SAT.

In the previous section, we gave computational cost analysis of the $\{2+p\},\{3\}$-SAT and showed that the random instances of $\{2+p\},\{3\}$-SAT are distinctly easier than the random instances of 3-SAT. But, does this give a correct insight into the cost requirements of the formulas obtained by transforming from random 3-SAT instances? There is one thing we have to think about before trying to answer this question. The instances transformed from random 3-SAT are not natural instances of random $\{2+p\},\{3\}$-SAT. They are instances 'engineered' to represent the instances of random 3-SAT. Thus, it is very likely that those instances will be at least as hard as their 3-SAT ancestors.

In order to have an insight on the computational cost distribution of transformed formulas, we conduct an experiment where we measure the cost required to determine satisfiability for instances of random 3-SAT and the corresponding images of these instances in $\{2+p\},\{3\}$-SAT. The experiment process can be roughly described as follows: First, we generate an instance in random 3-SAT and check for satisfiability, then we record the cost required. Second, we transform the instance to $\{2+p\},\{3\}$-SAT and check the satisfiability of the image instance. We again record the cost required to determine satisfiability.

The figures 26, 27 and 28 show the results of the experiments we conducted. Each figure is dedicated to a problem size of n for 50, 100 and 150. It is easily seen that the transformed instances require a higher cost for determining satisfiability. The hardness of transformed instances follow the pattern of the cost distribution required by the 3-SAT instances but they occur at much higher values. In general, easy-hard-easy pattern is preserved and there is not too much of a difference in the two easy regions. Especially the formulas belonging to the easy satisfiable region remain the same in terms of cost. As we start to pick instances from the transition region, naturally, the cost increases rapidly. But for the transformed instances, the rate of growth is much higher. As the value of

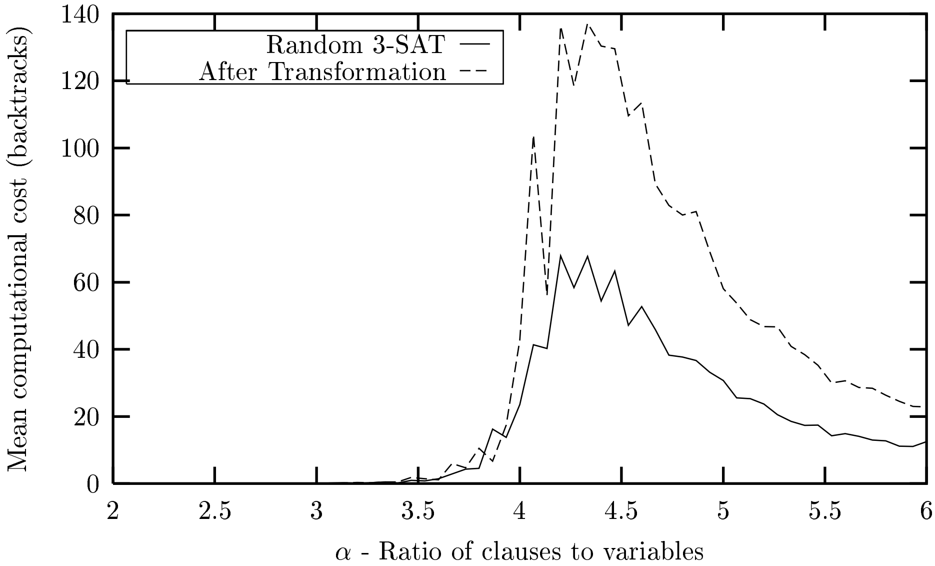

Figure 26: Comparison between the random 3-SAT with $n = 150$ and its image in {2+p},{3}-SAT.

α is increased in the transition region, the difference between the required cost increases dramatically. In figure 26, the difference in the cost at the beginning of the transition region by about 20 backtracks, jumps to a difference of about 65 backtracks at the peak point, where the hardest instances occur. In general, the difference in the required cost is around by a factor of 2, in all problem sizes.

In the easy unsatisfiable region, unlike the easy satisfiable region, the transformed instances require a cost which is slightly greater than the 3-SAT instances. As we know in random 3-SAT, the instances from the easy unsatisfiable region are more expensive for satisfiability checking than the instances from easy satisfiable region. The explanation behind this is as follows: When a procedure searches for a satisfying assignment of a given instance, when it succeeds to find a satisfying assignment then there is no need to continue on searching[1] and thus, the procedure quits. But, when the instance is actually unsatisfiable, then all of the search paths have to be checked so that no possible assignments remain unchecked. This results with a huge search tree and hence a more expensive cost. This explains the reason behind the relatively large cost in the unsatisfiable region of random 3-SAT. Because of this fact, the difference between the 3-SAT instances and their images turns out to be bigger, again, almost by a factor of 2.

To see with more detail, we separate the satisfiable instances from the un-

[1]Unless the procedure is forced to find all satisfying assignments. Here, since we are interested only in the determination of satisfiability, we look for at least one assignment satisfying the instance.

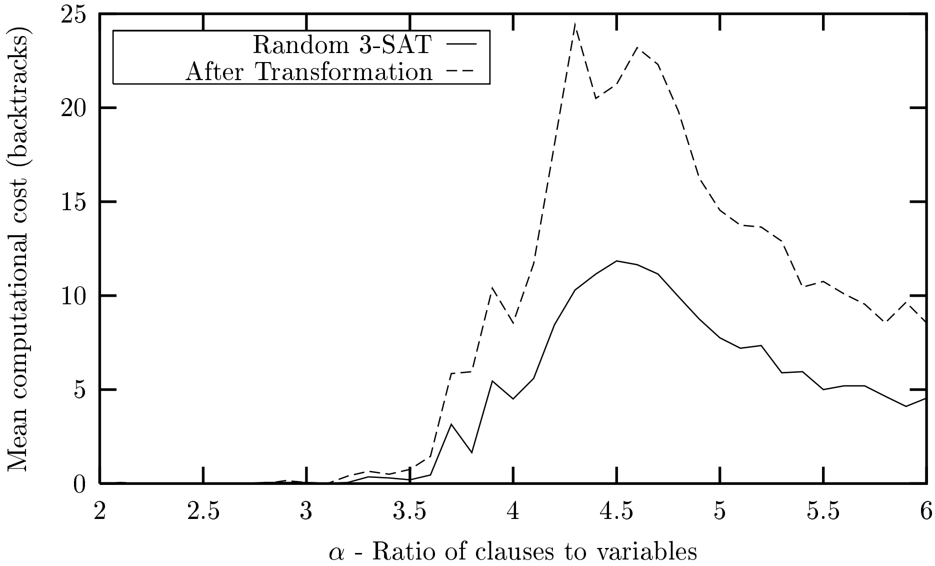

Figure 27: Comparison between the random 3-SAT with $n = 100$ and its image in {2+p},{3}-SAT.

satisfiable instances in the ensemble of problem size $n = 150$. The figures 29 and 30 show the cost distribution of random 3-SAT instances and their images for satisfiable and unsatisfiable instances, respectively. From figure 30, we see that the highest cost is contributed by the unsatisfiable formulas rather than the satisfiable ones. In both figures, the difficulty factor between the 3-SAT instances and their images remain the same.

The picture we drew above is not very positive as to whether such a transformation from random 3-SAT to {2+p},{3}-SAT is favorable in terms of problem hardness. Did we actually make everything worse by transforming instances from random 3-SAT? And, are the transformed instances actually much harder than the instances of random 3-SAT or is it our satisfiability checking algorithm that makes everything look worse? We try to answer these questions in the next section.

7.3 Is it really like that?

There are some studies in the literature concentrated on the question as to whether the extreme difficulty of some rare formulas, especially when they are from the easy satisfiable region of 3-SAT, is an inherent property or not [23]. If these formulas are not inherently difficult then this would mean that there is a way to solve them with almost the same cost as the other formulas coming from the same region. The results achieved in these studies show that the extreme difficulty is not an inherent property but just a result of the DP-like

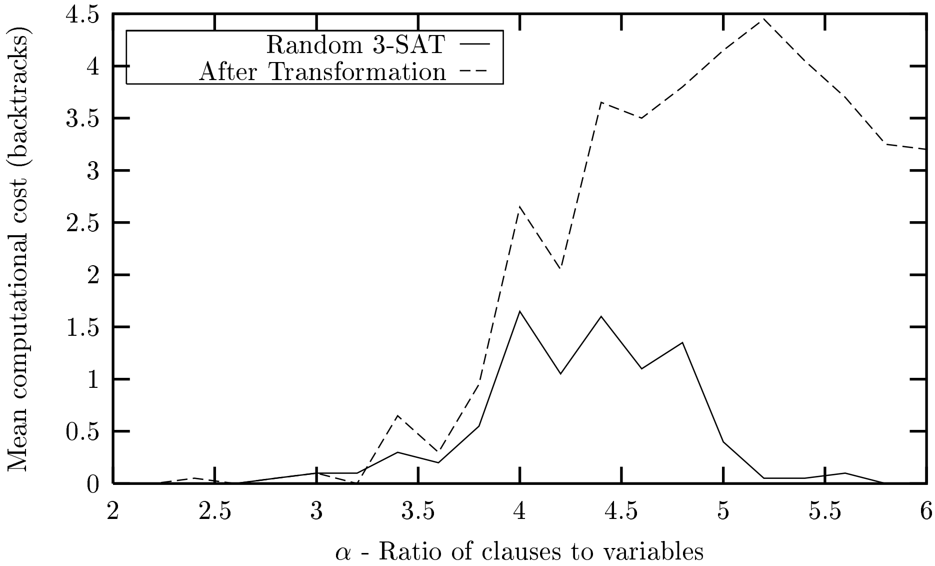

Figure 28: Comparison between the random 3-SAT with $n = 50$ and its image in {2+p},{3}-SAT.

systematic problem solving algorithms. In these algorithms, when a decision is to be taken in order to branch on a variable, a random choice is made among the available variables unless heuristics can help with the decision. However, in such a situation, the chosen variable may lead to a bad search path and in return, requiring more backtracks and hence a higher cost. The work around of this situation is to run the algorithm on the same instance many times or, alternatively, restart it after a certain number of backtracks that is believed to be high for that formula, so that different branching choices will be tested. As a result, we will have an idea about the real difficulty of the formula.

Figure 31 shows the results of the experiment in which we run the satisfiability checking algorithm 100 times per instance. We do this both for instances of 3-SAT and their images in {2+p},{3}-SAT under the transformation. Finally, we compare the average computational cost required to find a satisfying assignment in 3-SAT and {2+p},{3}-SAT. The cost required by an instance for deciding satisfiability is calculated by taking the mean of the number of backtracks appearing at each of the 100 runs.

7.4 What about the incomplete methods?

In this text, we mainly used complete algorithms as our satisfiability checking method. However it is a very meaningful question to ask how the incomplete algorithms cope with the transformed instances of random 3-SAT. In this section, we give results from an experiment in which the decision for satisfiability

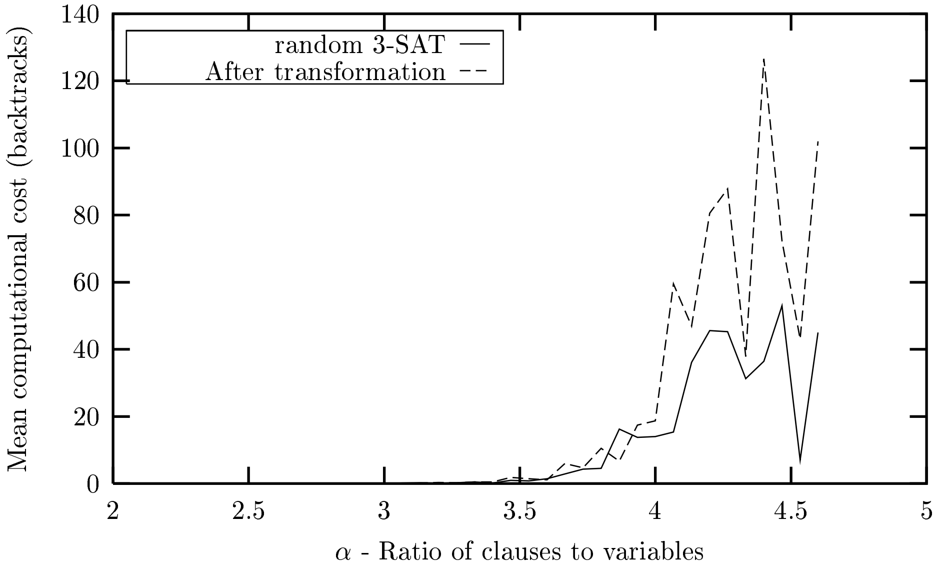

Figure 29: Comparison between the random 3-SAT with $n = 150$ and its image in {2+p},{3}-SAT for only satisfiable instances.

is given by an incomplete algorithm. For an overview about the incomplete methods refer to section 2.

One of the most popular incomplete algorithms in the literature is WalkSAT and it is also our choice here, as a representative of the incomplete methods. In the experiments included in this section, we will be interested in two kinds of data concerning the performance of WalkSAT. The first kind of data is the average number of flips which concerns the computational cost issues. WalkSAT reports the average number of flips until the first satisfying assignment is found. The search process in WalkSAT is done by flipping the sign of the assignment to a variable at each step. Hence, these are the data we are interested in to measure the computational cost required by the algorithm to find a satisfying assignment. Note that WalkSAT does not report the number of flips when an assignment could not be found.

Another parameter in WalkSAT is the success rate of the algorithm over the multiple tries. WalkSAT needs to be supplied with a parameter which tells the algorithm how many times it shall try to search for a satisfying assignment using a random initial choice. This is an important parameter for incomplete methods since they can get stuck within a local minimum occurring in the cost function of the formula, in which case they need to be restarted using another initial random assignment. We record the success rate for 10 tries of WalkSAT on every instance.

Figure 32 is about the computational cost requirements of WalkSAT both over random 3-SAT and its image in {2+p},{3}-SAT. The very high values of the

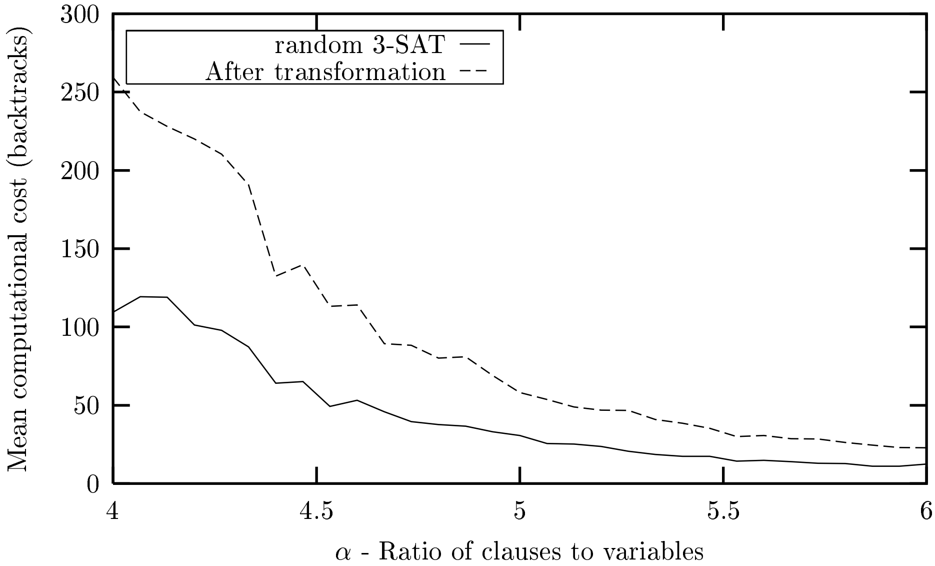

Figure 30: Comparison between the random 3-SAT with $n = 150$ and its image in {2+p},{3}-SAT for only unsatisfiable instances.

average number of flips made it necessary to plot the data on a log-scale. There is a huge difference between the cost required by the instances of random 3-SAT and their images. The image instances seem to follow the same pattern with their originating instances but appearing at very high levels of computational cost. Unlike the complete methods, the increase in the cost does not start to show itself with the transition region but from the earlier values of α. One common thing that we notice with the complete methods is the existence of a constant factor between the values of required cost. Here, a factor by 100 seems to be a good approximation to it. The half-cut appearance of the cost distribution as α increases is caused by the fact that WalkSAT does not report any measures on number of flips when a satisfying assignment could not be found.

The figure 33 gives information about the success rate of the WalkSAT on random 3-SAT and the comparison to its image in {2+p},{3}-SAT. Actually, the results in figure 32 gave some clues as to what shall we expect from the success rate comparison. The fall of the success rate in {2+p},{3}-SAT is much earlier than the fall of success rate in 3-SAT. When the rate for 3-SAT begins to drop, we see that {2+p},{3}-SAT success rate is already below 10%. This shows that WalkSAT can not find assignments for satisfying formulas as easily as in 3-SAT. Looking at the performance of WalkSAT on 3-SAT and comparing to figure 5, we see that WalkSAT succeeded to find at least one satisfying assignment for almost all of the satisfiable 3-SAT formulas including the ones in the transition region. However, it is also seen from the figure 33 that WalkSAT couldn't find

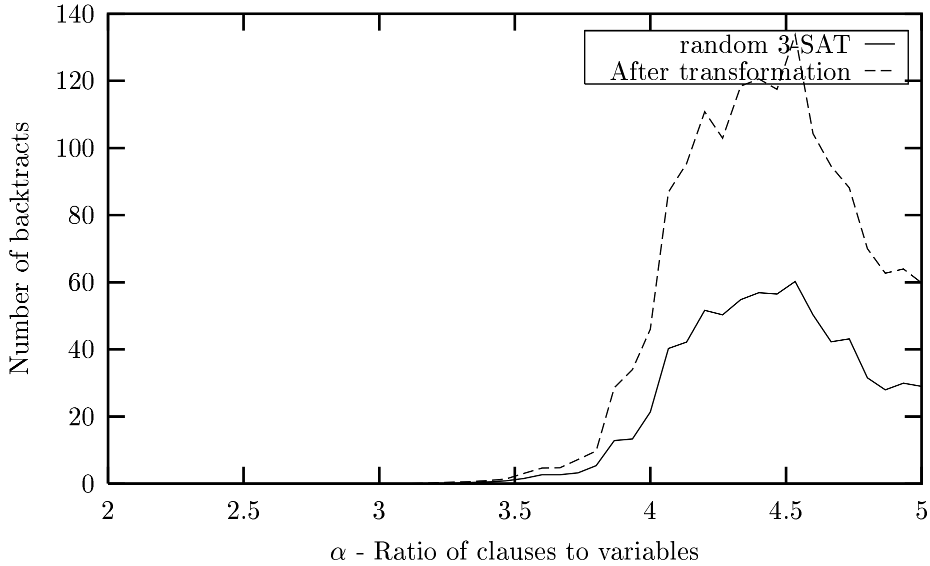

Figure 31: Comparison between the random 3-SAT with $n = 150$ and its image in {2+p},{3}-SAT by running the algorithm 100 times per instance.

any assignments for the rare satisfiable {2+p},{3}-SAT formulas occurring in the right of the transition region.

The discussion above concerning the performance of the WalkSAT shows that, if we regard WalkSAT as a moderate representative to the incomplete methods, the level of difficulty of the formulas obtained by transforming 3-SAT instances to {2+p},{3}-SAT is much higher with incomplete methods when compared to the complete methods.

8 Some well known applications

Given the fact that many practical real-life problems can be encoded into SAT, we ask the question "What happens to these problems under our transformation?". There are numerous problems that can be encoded in SAT but here we will be exploring only three of them. For more information about SAT-encoded problems please check [22].

The examples we include here are challenging for most of the known satisfiability checkers. Our interest is to see the performance of the very same algorithms over the images of these instances under our transformation introduced in section 4 and then make a comparison with the performance exhibited for the originating instances.

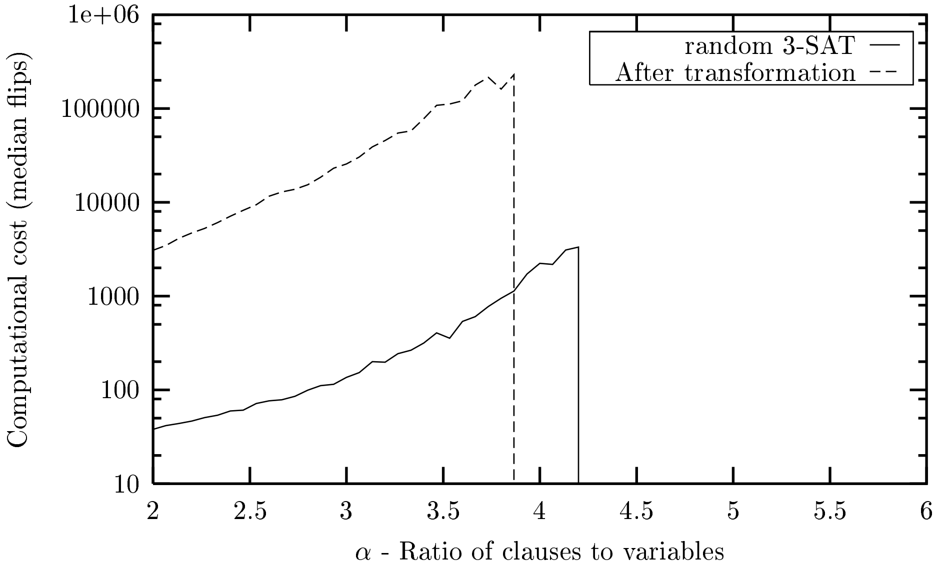

Figure 32: The computational cost comparison between the random 3-SAT with $n = 150$ and its image in {2+p},{3}-SAT with WalkSAT.

8.1 SAT-encoded "Flat" Graph Coloring Problems

The Graph Coloring Problem (GCP) [22] is a well-known combinatorial problem from graph theory : Given a graph $G = (V, E)$, where $V = v_1, v_2, ..., v_n$ is the set of vertices and E the set of edges connecting the vertices, find a coloring $C : V \mapsto N$, such that connected vertices always have different colors. There are two variants of this problem: In the optimisation variant, the goal is to find a coloring with a minimal number of colours, whereas in the decision variant, the question is to decide whether for a particular number of colours, a coloring of the given graph exists. The two variants are tightly related, as optimal colorings can be found by solving a series of decision problems, using, for instance, binary search to determine the minimal number of colours. In the context of SAT-encoded Graph Coloring problems, we focus on the decision variant.

8.1.1 SAT encoding

Following earlier approaches in literature, we use a straightforward strategy for encoding GCP instances into SAT: Each assignment of a color to a single vertex is represented by a propositional variable; each coloring constraint (edge of the graph) is represented by a set of clauses ensuring that the corresponding vertices have different colours, and two additional sets of clauses ensure that valid SAT assignments assign exactly one colour to each vertex.

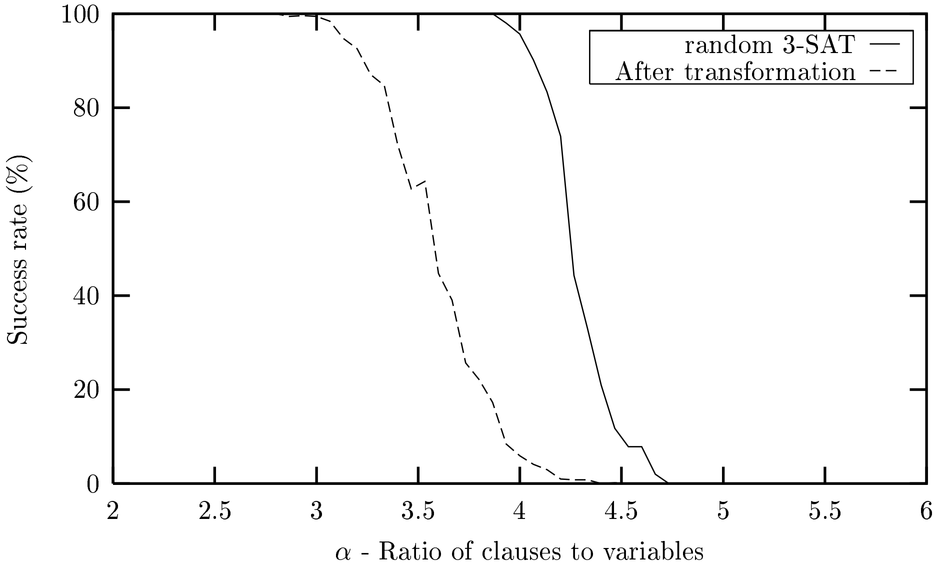

Figure 33: The success comparison between the random 3-SAT with $n = 150$ and its image in {2+p},{3}-SAT with WalkSAT.

8.1.2 GCP test-sets

The test-sets provided here (table 1) are generated using J.Culberson's flat graph generator [14]. They consist of 3-colorable flat graphs the connectivity of which is adjusted in such a way that the instances are very hard to solve (on average) for graph coloring algorithms like the Brelaz heuristic [12]. All instances are satisfiable.

test-set	**instances**	**vertices**	**edges**	**colors**	**vars**	**clauses**
flat30-60	100	30	60	3	90	300
flat50-115	1000	50	115	3	150	545
flat75-180	100	75	180	3	225	840
flat100-239	100	100	239	3	300	1117
flat125-301	100	125	301	3	375	1403
flat150-360	100	150	360	3	450	1680
flat175-417	100	175	417	3	525	1951
flat200-479	100	200	479	3	600	2237

Table 1: SAT-encoded Graph Coloring test-sets (flat random graphs). All instances are satisfiable.

8.2 Backbone-minimal sub-instances

8.2.1 The Backbone and Backbone-minimality

A literal l is *entailed* by a satisfiable SAT instance C iff $C \wedge \neg l$ is unsatisfiable. The backbone of a satisfiable SAT instance is the set of entailed literals and so, the backbone size is the number of entailed literals. An instance C is backbone-minimal if for each clause c in C there exists a literal l in the backbone of C such that l is not in the backbone of $C - \{c\}$ [22].

8.2.2 Instance Generation

Two types of instance are provided (table 2). Firstly, some standard satisfiable threshold random k-SAT instances. These are generated according to 3 parameters: k: the number of literals per clause, n: the number of variables, m: the number of clauses. Each random k-SAT instance is generated as follows: Generate m random k-clauses. For each clause: Select at random k distinct variables from the set of n. Negate each with probability 0.5. These form the literals of the clause. If the instance is unsatisfiable, restart the generation process.

The second type of instances were generated from the above random k-SAT instances. For each random 3-SAT instance C, one backbone-minimal sub-instance of C was found. This was done by removing clauses from C such that the backbone was unaffected. This was iterated until the resulting instance was backbone-minimal. At each step the removed clause was selected at random from those whose removal did not affect the backbone.

[25] used these instances as part of a study which aimed to explain why cost for local search algorithms was high in the threshold region of random 3-SAT. For the local search algorithm WSAT/SKC, the backbone-minimal sub-instances had a far higher search cost than the random 3-SAT instances from which they were derived, even though the sub-instances have fewer clauses, the same backbone size and at least as many solutions.

test-set	**instances**	**literals per clause**	**vars**	**clauses**
RTI_k3_n100_m429	500	3	100	429
BMS_k3_n100_m429	500	3	100	varies

Table 2: Random 3-SAT (RTI) and backbone minimal sub-instance (BMS) test-sets. Each instance in the Random-3-SAT collection has a corresponding backbone minimal sub-instance in the other collection. All instances are satisfiable.

8.3 Results

Tables 4 and 3 show the results of the comparison tests. The instances in all three applications are first proved using our standard satisfiability checker and the data concerning computational cost is recorded. Then we applied our transformation to all of the instances and get instances in which no variables occur more than three times. Note that the output instances are not necessarily members of the problem class {2+p},{3}-SAT. The obvious reason behind this is that the originating instances may include clauses which has length more than three and most of the time they have. So, the transformer we are using here is only concerned with the number of occurrences of the variables. Table 4 follows the pattern in this entire work: The computational cost increases precisely with the transformation. The difference between the originating instances and the transformed instances is roughly by a factor of 2.

test-set	**Number of backtracks**			
	Mean	**Median**	**Maximum**	**Minimum**
	original instance **transformed instance**			
flat30-60	0.05 0.0	0.0 0.0	2 0	0 0
flat50-115	0.26 0.03	0.0 0	4 2	0 0
flat75-180	0.89 0.23	0.0 0.0	7 3	0 0
flat100-239	2.65 1.0	2.0 1.0	16 11	0 0
flat125-301	7.83 2.14	4.0 1.0	80 13	0 0
flat150-360	17.79 6.04	10.0 4.0	181 57	0 0
flat175-417	37.17 11.91	21.5 8.0	184 53	0 0
flat200-479	85.4 28.04	56.5 16.0	944 157	0 0

Table 3: Comparison table for flat graph coloring instances

The surprise comes with the flat graph coloring problem. In contrary to the pattern we mentioned above, this time the transformation causes an impressive decrease in the computational cost. The maximum number of backtracks required for the test-set flat200-479 is 944 where as after transformation this number is 157. This is an improvement by a factor of 6. This improvement can be clearly seen in all test-sets of the flat graph coloring problem. Besides flat graph coloring, the same effect exists in the transformation of morphed graph coloring instances. However, we do not include any work related to morphed graph coloring problem.

test-set	**Number of backtracks**			
	Mean	**Median**	**Maximum**	**Minimum**
	original instance **transformed instance**			
RTI_k3_n100_m429	5.84 11.86	5 10	29 52	0 0
BMS_k3_n100_m429	24.76 52.26	19 42	125 209	0 0

Table 4: Comparison table for backbone-minimal sub-instances

9 Conclusions

It has been shown that instances of 3-SAT can be transformed efficiently to instances of {2+p},{3}-SAT. We have given a detailed description of the {2+p},{3}-SAT problem in terms of both computational cost and the structure. Later we saw that the instances obtained by transforming 3-SAT instances are distinctly harder than the hardness of their ancestors in 3-SAT. However, it does not seem to be possible to generate random {2+p},{3}-SAT instances which are as hard as the images of 3-SAT instances. Actually, we could not generate any instances of {2+p},{3}-SAT which required more than 1 backtrack. The reason behind this difference between the instances of this very same problem is the fact that the instances which are 3-SAT images are special constructions to represent their originating instances in 3-SAT.

When the structural issues are reviewed, we see that with the transformation from 3-SAT to {2+p},{3}-SAT, there is a big difference in the dimensions of the space of parameters of {2+p},{3}-SAT. We gave a map of the αp space and show that both parameters are strictly bounded and moreover, both the upper and the lower bounds of *alpha* is determined by the value of p. On the other hand the instances transformed from the 3-SAT are encapsulated in a small piece of this bounded space.

We saw that there are relations between the phase transition effect in {2+p},{3}-SAT and the parameter p, ratio of 3-length clauses to all clauses. As the value of p is increased, the phase transition effect shifts to higher values of α in the corresponding ensemble. Moreover, considering that the upper bound on the value of α decreases with increasing p, after some value of p the phase transition effect disappears. Using the exact-{2+p},{3}-SAT distribution, we estimated that the highest value which we can still see some change in the probability of satisfiability is 0.3. Given the fact that every instance of exact-{3},{3}-SAT is satisfiable, this behavior in {2+p},{3}-SAT with increasing p is quite natural.

9.1 Future research

As mentioned before, the image of 3-SAT in {2+p},{3}-SAT is quite dense. For instance, most of the unsatisfiable instances and especially the ones from the unsatisfiable region have the same value of ratio of clauses-to-variables. This fact may turn out to be very useful in the investigation of the properties of these instances. It might prove useful to study the transformed instances to understand the 3-SAT instances themselves. One open question which lies here is whether the image of 3-SAT can give us any clues about the satisfiability of the instances in the transition region. One other is whether we can improve the our current theoretical knowledge about the location of the phase transition in 3-SAT.

We have used the common satisfiability checking algorithms available. But the fact that the instances resulting from transforming 3-SAT instances are unnatural makes us to think that more optimistic performance results can be achieved by implementing a special search algorithm for 3-SAT images in {2+p},{3}-SAT. There are examples of this in the literature where special algorithms are coded to achieve success in individual problem forms [15]. Also there are some other algorithms which we wonder about their performances on the transformed instances [19].

One other source of curiosity concerns the disappearing of the phase transition effect in {2+p},{3}-SAT with increasing p. We are suspicious whether the result which says that every instance of {3},{3}-SAT is satisfiable can be extended [4]. Our results suggest that every instance of {2+p},{3}-SAT with $p > 0.3$ is satisfiable. The validity of the later should be investigated and a theoretic proof should be given if it is applicable.

The results we achieved with graph coloring problems point out that extra research on some other applications is essential. The transformation method seems to have different effect on different problem structures. The studies for the transformation's effect on different applications should reveal the reasons behind its success on particular problems.

References

[1] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without bdds. *In the Proceedings of the Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS99)*, 1999.

[2] B.Selman, H. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. *Proceedings of the 10th National Conference on Artificial Intelligence, AAAI*, pages 440–446, 1992.

[3] B.Selman and S.Kirkpatrick. Critical behaviour in the satisfiability of random boolean formulae. *Racah Institute of Physics and Center for Neural Computation, Hebrew University, Israel*, 1993.

[4] C.H.Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.

[5] S.A. Cook. The complexity of theorem-proving procedures. *Proceedings of the third ACM symposium on Theory of Computing*, pages 151–158, 1971.

[6] S.A. Cook and D.G. Mitchell. Finding hard instances of the satisfiability problem: A survey. *Dimacs Series in Discrete Mathematics and Theoretical Computer Science*, 35, 1997.

[7] D.Mitchell, B.Selman, and H.Levesque. Hard and easy distributions of sat problems. *Proceedings on the 10th National Conference on Artificial Intelligence, AAAI*, pages 459–465, 1992.

[8] D.Mitchell and H.Levesque. Some pitfalls for experimenters with random sat. *Proceedings on the 10th National Conference on Artificial Intelligence, AAAI*, 1992.

[9] Ian P. Gent and Toby Walsh. Easy problems are sometimes hard. *Artificial Intelligence*, pages 335–345, 1994.

[10] I.P. Gent and T. Walsh. The search for satisfaction. *Department of Computer Science University of Strathclyde, Scotland*, 1999.

[11] Jun Gu, Paul W. Purdom, John Franco, and Benjamin W. Wah. Algorithms for the satisfiability (sat) problem: A survey. *Dimacs Series in Discrete Mathematics and Theoretical Computer Science*, 1997.

[12] Tad Hogg. Refining the phase transition in combinatorial search. *Artificial Intelligence*, 81:127–154, 1996.

[13] Tad Hogg, Bernardo A. Huberman, and Colin P. Williams. Phase transitions and the search problem, editorial. *Artificial Intelligence*, 81:1–15, 1996.

[14] J.Culberson. *Joe Culbersons's Graph Coloring Page.* http://web.cs.ualberta.ca/ joe/Coloring/Generators/flat.html.

[15] Chu Min Li. Integrating equivalency reasoning into davis-putnam procedure. *Proceedings of AAAI*, 2000.

[16] M.Davis, G.Logemann, and D.Loveland. A machine program for theorem proving. *Comms. ACM*, 5:394–397, 1962.

[17] M.Davis and H.Putnam. A computing procedure for quantification theory. *Journal of the Association for Computing Machinery*, pages 201–215, 1960.

[18] D. Mitchell, B.Selman, and H.Levesque. Hard and easy distributions of sat problems. *Proceedings on the 10th National Conference on AI*, pages 459–465, 1992.

[19] Matthew W. Moskewicz, Conor F. Madigan, and Ying Zhao et al. Chaff: Engineering an efficient sat solver. 2001.

[20] P.Cheeseman, B.Kanefsky, and W.M.Taylor. Where the really hard problems are. *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, pages 331–337, 1991.

[21] Ian P.Gent and Toby Walsh. The sat phase transition. *Proceedings of 11th ECAI*, pages 105–109, 1994.

[22] Satlib. *Benchmark instances and descriptions on the web.* http://www.satlib.org/benchm.html, 2000.

[23] B. Selman and S. Kirkpatrick. Critical behaviour in the computational cost of satisfiability testing. *Artificial Intelligence*, 81:273–295, 1996.

[24] Ofer Shtrichman. Tuning sat checkers for bounded model checking. *In E.A. Emerson and A.P. Sistla Eds, Proceedings of Computer Aided Verification 2000 (CAV2000)*, 2000.

[25] Josh Singer, Ian Gent, and Alan Smaill. Backbone fragility and the local search cost peak. *Journal of Artificial Intelligence Research*, 12:235–270, 2000.