Greedy algorithms, ordering of variables, and d-degenerate instances

Abstract

We consider the MAX-2-SAT problem on d-degenerate formulas. This class of 2-CNFs is a generalization of 2-CNFs of bounded tree width. We first show that the class of d-degenerate formulas is very broad, since random formulas can be shown to be d-degenerate for an appropriate d. Then we test several heuristic algorithms on random d-degenerate formulas and compare results against similar tests on regular random 2-CNFs. Finally, we model the performance of one of these algorithms, the greedy one, on both random and random d-degenerate formulas by systems of differential equations.

Keywords: Satisfiability, algorithm, analysis, random satisfiability, d-degenerate

Introduction

Solving instances having tree-like structure and finding an efficient ordering of variables are among the most popular approaches to the Satisfiability problem (SAT). In practical SAT solvers, especially backtrack ones, finding a good variable ordering is extremely important, and this has been an area of intensive research, see (Biere et al. 2009, Chapter 7) for a comprehensive survey.

There are many ways to define 'tree-likeness', but probably the most widely accepted one is through tree width (Robertson and Seymour 1986). As is well known from the results of (Courcelle 1990), many combinatorial problems become polynomial time solvable when restricted on instances of bounded tree width. In the case of SAT it works as follows. For every instance there is a graph associated with it: variables become vertices, and two vertices are connected with an edge if they appear together in the same clause. If this graph has tree width k then the instance is said to be of tree width k. If we restrict the input with instances of tree width at most k then such a restricted SAT, as well as the Maximum Satisfiability problem (MAX-SAT), is solvable in polynomial time (Dechter and Pearl 1989; Freuder 1982; 1990; Bjesse et al. 2003); at least theoretically. The class of instances of small tree width is quite restricted, however, even if an instance does not have small tree width, often it can be converted into one with these desired properties by, for example, removing a small number of variables, see, e.g. (Williams, Gomes, and Selman 2003). Thus, solving restricted classes of instances can be helpful in the general setting.

Bounded tree width is also related to certain orderings of variables. One of the equivalent characterizations of graphs of tree width k is the existence of an *elimination order* (Bodlaender 2006). To decide if an ordering of variables π is an elimination order, first, for each variable v we connect all its neighbors that appear in π after v; then we check if in the resulting graph every variable v has at most k neighbors that appear in π after v. Unfortunately, finding an elimination order is usually difficult. Although if tree width is fixed, an elimination order can be found in linear time (Bodlaender and Kloks 1991; Bodlaender et al. 1991), the associated constant factors are so large that these algorithms are impractical beyond very small widths.

In this paper we make an attempt to find a relaxation of bounded tree width that would serve two purposes. Firstly it should be more practical to use, and secondly it should inherit to some extent the good properties of bounded tree width and allow more efficient algorithms. One natural way to relax bounded tree width is to require the existence of a weaker elimination order. More precisely, in the definition of elimination order we omit the step of adding edges. What we obtain is the class of *d-degenerate graphs*, see, e.g. (Thomassen 1995). More precisely, a graph $G = (V, E)$ is said to be *d-degenerate* if its vertices can be ordered $V = \{v_1, \ldots, v_n\}$ so that for each v_i the number of its neighbors v_j with $j > i$ does not exceed d.

We will focus on the Maximum 2-Satisfiability problem (MAX-2-SAT), in which every clause contains at most two variables (literals). While the corresponding decision problem 2-SAT is easily solvable, it is hard to find an optimal solution of MAX-2-SAT (Garey, Johnson, and Stockmeyer 1974) and even to approximate it (Håstad 2001; Trevisan et al. 2000). We address the following question: can MAX-2-SAT algorithms benefit from restricting on d-degenerate instances?

The class of d-degenerate graphs (and, therefore, 2-CNFs) is a proper generalisation of the class of graphs (2-CNFs) of tree width d. Indeed, any elimination order for the latter witnesses that the graph is d-degenerate. On the other hand, a $n \times m$-grid, the graph whose vertices are located in

the nodes of a rectangular grid and connected to the closest vertical and horizontal neighbors, is 4-degenerate, but its tree width equals $\min(n, m)$. The class of d-degenerate graphs (2-CNFs) satisfies the first desirable property: as we shall see, it is very easy to find an elimination order for a d-degenerate graph.

The second condition is not so clear. Observe that graphs of small degree, say, 3-regular graphs, are d-degenerate, and it is an easy excersise to show that finding an optimal solution of a 3-regular Max-2-SAT problem remains NP-hard. We will also show that a random 2-CNF from $\Phi^{2-\mathsf{SAT}}(n, pn)$, that is, the uniform distribution of 2-CNFs with n variables and pn clauses, with high probability is d-degenerate provided d is large enough (but depends only on p). This result virtually rules out the hope to obtain worst-or average-case polynomial time algorithms. However, this does not mean that restricting Max-2-SAT to d-degenerate problems is useless. Firstly, the approximation properties of such problems are unknown, and there is a possibility (although unlike) that they admit better approximation than the general MAX-2-SAT. It is also possible that the existing heuristics find a better approximation on d-degenerate instances, or certain algorithms perform well on such instances by exploiting their properties such as an elimination order, provided d is small enough to deviate significantly from ordinary random instances.

We consider one of the versions of the greedy algorithm that makes use of variable ordering, and two local search type algorithms: GSAT and the basic local search. We compare the performance of these methods on random 2-CNFs from $\Phi^{2-\mathsf{SAT}}(n, pn)$ and the random distribution $\Phi^{\mathsf{d-deg}}(n, pn, d)$ of d-degenerate formulas for n variables and pn clauses, with different values of p and d. Notice that generating random d-degenerate formulas is a difficult problem; in particular we had to use various tricks just to generate such formulas with sufficiently many variables. The results of the tests show that there is no significant difference in the performance of all 3 algorithms on random and random d-degenerate instances. Since we could only test relatively small instances, we also model the behavior of the greedy algorithm on $\Phi^{2-\mathsf{SAT}}(n, pn)$ and $\Phi^{\mathsf{d-geg}}(n, pn, d)$ using Wormald's theorem (Wormald 1995) and differential equations. Again, the model does not show any significant difference.

Thus, the class of d-degenerate instances does not seem to be restrictive enough to make MAX-2-SAT more tractable.

d-degenerate graphs and formulas

d-degenerate formulas and elimination orders

A graph $G = (V, E)$ is said to be *d-degenerate* if the vertices from V can be oredred $v_1, \dots v_n$ in such a way that for every $i \in \{1, \dots, n\}$ the set $\{v_j \mid j > i, \text{ and } v_i v_j \in E\}$ contains at most d elements. For a 2-CNF ϕ with set of variables V by G_ϕ we denote the graph whose vertices are variables from V and a pair vw is an edge if and only if v and w appear in the same clause and $v \neq w$. We call 2-CNF ϕ *d-degenerate* if G_ϕ is d-degenerate.

The class of d-degenerate graphs (2-CNFs) is quite large. For example it contains all graphs of maximum degree at most d.

Checking whether a graph is d-degenerate is very simple. The procedure given in Algorithm 1 is based on the following straightforward statement.

Lemma 1 *(1) If $G = (V, E)$ is d-degenerate and $v \in V$ has degree at most d, then the graph induced by $V - \{v\}$ is d-degenerate.*

(2) If every vertex of $G = (V, E)$ has degree at least $d + 1$ then G is not d-degenerate.

Proof: (1) Let $G' = (V', E')$ be the subgraph of G induced by $V' = V - \{v\}$, and $v_1, \dots, v_n$ an elimination order for G. Assuming $v = v_j$ set $v'_1, \dots, v'_{n-1}$ to be the sequence of vertices of V' given by $v'_i = v_i$ if $i < j$ and $v'_i = v_{i+1}$ if $i > j$. As is easily seen, $v'_1, \dots, v'_{n-1}$ is an elimination order for G'.

(2) The first vertex in an elimination order has degree at most d, hence, G has no elimination order. □

Algorithm 1 constructs an elimination order of a given graph G by iteratively putting in the order a vertex of small degree and excluding it from the graph. It fails to find an elimination order if it arrives to an induced subgraph G' of G such that every vertex of G' has degree at least $d+1$. Such an induced subgraph is called a *d-core* of G. In the former case the algorithm produces an elimination order, while in the latter case by Lemma 1 it proves that G is not d-degenerate.

Algorithm 1 Finding an elimination order

Require: a 2-CNF ϕ
Ensure: an elimination order of variables of ϕ if one exists, NO if ϕ is not d-degenerate

```
1: G = G_φ, π = ∅
2: while G is nonempty do
3:    if there is a vertex v of G of degree at most d then
4:       append v to π
5:       remove v and all edges incident to v from G
6:    else
7:       return NO
8:    end if
9: end while
10: return π
```

Random vs. d-degenerate formulas

The Random SAT problem, where instances are chosen uniformly at random from the set of CNFs with given number of variables and fixed clause-to-variables ratio, is a widely used benchmark problem. More precisely, let p be a positive (rational) number. Then $\Phi^{2-\mathsf{sat}}(n, pn)$ denotes the uniform distribution of 2-CNFs with exactly n variables and pn clauses (Selman, Mitchell, and Levesque 1996). To generate a random formula we pn times repeat the following: Choose randomly 2 (distinct) variables out of n, and then negate each of them with probability 1/2. Add the resulting clause to the formula.

The proposition below amounts to say that provided d sufficiently high (but depends only on p) almost all 2-CNFs in $\Phi^{2-\mathsf{sat}}(n, pn)$ are d-degenerate.

Proposition 2 *Let $\phi \in \Phi^{2-\mathsf{sat}}(n, pn)$ be a random formula with n variables and pn clauses, $p > 1$ a constant. For any $d > 6p$ the probability that ϕ is not d-degenerate is $o(n^{-4})$.*

Remark 3 Proposition 2 can be restated in terms of graphs as follows: If $d > 6p$ a random graph with n vertices and pn edges with high probability does not contain a d-core. Similar results were obtained in (Pittel, Spencer, and Wormald 1996), where the conditions of appearance of a d-core in a random graph were studied. However, the results of (Pittel, Spencer, and Wormald 1996) concerns *giant* d-cores, that is, d-cores that contain at least αn vertices for a certain (although arbitrarily small) constant α. On the contrary, Proposition 2 claims that even small d-cores are unlikely in random graphs, although this requires larger values of p.

Proof: We will consider a 2-CNF as a (possibly) multigraph in which vertices are connected with edges of 4 different types that correspond to the 4 types of 2-clauses. Any two vertices can be connected with more that one edge, but with at most one edge of each type. In a random multigraph of this kind pn edges are selected uniformly at random from the $2n(n-1)$ possible edges.

So, let $G = (V, E)$ be a random multigraph with n vertices and pn edges. It suffices to show that with high probability every subset of V contains a vertex of degree less than d. We prove a stronger statement: With probability $1 - o(n^{-4})$ for any subset $W \subseteq V$ the average degree of vertices of W does not exceed d.

Fix a set $W \subseteq V$, and let $|W| = \ell$. If $\ell > \frac{n}{3}$ then the claim is trivially true, as G has only $pn < \frac{d\ell}{2}$ edges; so assume $\ell \leq \frac{n}{3}$. The probability that a fixed edge, one of the $2\ell(\ell-1)$ possible edges between vertices in W, is an edge in G equals (up to an infitely small amount) $\frac{p}{2n}$. Fixed set of ℓ vertices. The probability that there is a set of $\frac{d\ell}{2}$ pairs of these vertices that are all edges does not exceed (we use the inequality $\binom{s}{t} \leq \left(\frac{se}{t}\right)^t$)

$$\binom{2\ell(\ell-1)}{\frac{d\ell}{2}} \left(\frac{p}{2n}\right)^{\frac{d\ell}{2}} \leq \binom{2\ell^2}{\frac{d\ell}{2}} \left(\frac{p}{2n}\right)^{\frac{d\ell}{2}} \leq \left(\frac{2\ell ep}{dn}\right)^{\frac{d\ell}{2}}.$$

Now using the union bound, the probability that a set of size ℓ exists such that the degree of all vertices in the subgraph induced by this set are at least d does not exceed

$$\begin{aligned}
&\binom{n}{\ell}\left(\frac{2\ell ep}{dn}\right)^{\frac{d\ell}{2}} \\
\leq\ &\left(\frac{ne}{\ell}\right)^{\ell}\left(\frac{2\ell ep}{dn}\right)^{\frac{d\ell}{2}} \\
=\ &\left(\frac{\ell}{n}\right)^{\left(1-\frac{2}{d}\right)\frac{d\ell}{2}} \cdot \left(\frac{2pe^{(1+2/d)}}{d}\right)^{\frac{d\ell}{2}} \\
\leq\ &\left(\frac{\ell}{n}\right)^{\left(1-\frac{2}{d}\right)\frac{d\ell}{2}} \\
\leq\ &\left(\frac{\ell}{n}\right)^{\ell}.
\end{aligned}$$

We consider two cases. If $\ell \leq n^{1/6}$ then, since $\ell > 6$,

$$\left(\frac{\ell}{n}\right)^{\ell} \leq \left(\frac{1}{n}\right)^{\frac{5\ell}{6}} \leq \left(\frac{1}{n}\right)^{5}.$$

If $\ell > n^{1/6}$ then

$$\left(\frac{\ell}{n}\right)^{\ell} \leq \left(\frac{n/3}{n}\right)^{n^{1/6}} \leq \left(\frac{1}{3}\right)^{n^{1/6}} \leq \left(\frac{1}{n}\right)^{5}.$$

As $\ell < \frac{n}{3}$, the probability that there is such a set of size $\ell < \frac{n}{3}$ is bounded from above by

$$\sum_{\ell=6}^{n/3}\left(\frac{1}{n}\right)^{5} \leq \left(\frac{1}{n}\right)^{4},$$

that completes the proof. □

Generating random d-degenerate formulas

We test the algorithms on random d-degenerate formulas. It turns out that sampling random d-degenerate formulas is much more difficult than sampling just random formulas, and even defining a proper distribution is trickier. We use the dynamic programming algorithm for generating d-degenerate graphs from (Bauer, Krug, and Wagner 2010), and amend it so that it outputs 2-CNFs. Distribution $\Phi^{\mathsf{d-gen}}(n, pn, d)$, where n is the number of variables, p is the clause-to-variables ratio, and d is the degeneracy parameter, is defined as follows: We fix an order of variables $x_1, \ldots, x_n$; then include every 2-CNF over $x_1, \ldots, x_n$ containing pn clauses, and for which $x_1, \ldots, x_n$ is an elimination order. Let $D(n, pn, d)$ denote the number of formulas in $\Phi^{\mathsf{d-gen}}(n, pn, d)$.

For a given number of variables n, clause-to-variables ratio p, and d, the algorithm works as follows:

Algorithms and experiments

Algorithms

We test 3 algorithms: GSAT (Selman, Levesque, and Mitchell 1992), basic local search (LS) (Gu 1992), in which a variable to flip is chosen randomly rather than greedily,

Algorithm 2 Generating a random d-degenerate 2-CNF

Require: n, p, d
Ensure: a d-degenerate 2-CNF ϕ with n variables and pn clauses

Let $\{x_1, \ldots, x_n\}$ be variables. $N = 0$.
for $i = 0$ to n **do**
  for $k = 0$ to d **do**
    Compute $D_k = D(n - i, pn - N - k, d)$. This way we can evaluate the probability that x_i has k neighbors among $\{x_{i+1}, \ldots, x_n\}$.
  end for
  $D = \sum_{k=0}^{d} D_k$
  Select $k \in \{0, \ldots, d\}$ at random accordingly to the distribution $D_0/D, \ldots, D_d/D$.
  Choose random $x_{i_1}, \ldots, x_{i_k}$ such that $i_1, \ldots, i_k \in \{i+1, \ldots, n\}$.
  For each $1 \leq \ell \leq k$ include into ϕ one of the 4 possible clauses containing x_i, x_{i_ℓ} with equal probability.
  $N = N - k$
end for
return ϕ

and a version of the greedy (GR) algorithm described below. The algorithm GR tries to take advantage from using an elimination order of a d-degenerate instance. It takes as input a 2-CNF ϕ whose variables are in a certain order $\{x_1, \ldots, x_n\}$, and sets $\phi_0 = \phi$. Then in step i it assigns x_i the value satisfying the maximum number of clauses of ϕ_{i-1}. Then it constructs ϕ_i by removing all clauses of ϕ_{i-1} that do not contain variables from $\{x_{i+1}, \ldots, x_n\}$, removing all clauses satisfied by x_i, and replacing every binary clause containing x_i, x_j, $j > i$, and not satisfied by x_i into a unary clause on x_j. A formal description of the algorithm is as follows:

Algorithm 3 GR(ϕ, $V = \{x_1, \ldots, x_n\}$)

$\phi_0 = \phi$, $\pi = \emptyset$
for $i = 1$ to n **do**
  if $x_i = 0$ satisfies more clauses of ϕ_{i-1} than $x_i = 1$ **then**
    $\pi = \pi \cup \{x_i = 0\}$
    $\phi_i = \phi_{i-1} \setminus (\{(x_i), (\bar{x}_i)\} \cup \{(\bar{x}_i, x_j), (\bar{x}_i, \bar{x}_j) \mid j > i\}$
    replace $(x_i, x_j), (x_i, \bar{x}_j)$ with $(x_j), (\bar{x}_j)$, respectively
  else
    $\pi = \pi \cup \{x_i = 1\}$
    $\phi_i = \phi_{i-1} \setminus (\{(x_i), (\bar{x}_i)\} \cup \{(x_i, x_j), (x_i, \bar{x}_j) \mid j > i\}$
    replace $(\bar{x}_i, x_j), (\bar{x}_i, \bar{x}_j)$ with $(x_j), (\bar{x}_j)$, respectively
  end if
end for
return π;

Experiments

We run the 3 algorithms on random and random d-degenerate instances with 1000 variables, and parameters p and d as shown in Tables 1 and 2. The parameters of GSAT are set to be 50 restarts and 100000 variable flips per restart. Tables 1 and 2 show number of clauses remaining unsatisfied. These numbers are averaged over 10 runs.

Since for $d > 6p$ almost every random instance is d-degenerate by Proposition 2, we can have improvements in the performance of the algorithms only when d is small and close to p, so that the structure of a random d-degenerate instance deviates significantly from that of a random instance. This explains the choice of values of d.

Table 1: Performance of GR, LS, and GSAT on random d-degenerate instances

Input parameter			Number of unsatisfied clauses		
n	pn	d	GS	LS	GSAT
1000	10000	11	1532	1401	1310
		20	1533	1416	1304
		30	1535	1410	1307
	15000	17	2571	2374	2283
		25	2556	2375	2281
		35	2556	2395	2288
	20000	23	3588	3368	3292
		30	3600	3399	3286
		40	3604	3403	3280
	25000	30	4683	4435	4332
		35	4695	4458	4335
		45	4677	4439	4342
	30000	37	5763	5481	5400
		40	5780	5510	5382
		50	5772	5495	5396

Table 2: Performance of GR, LS, and GSAT on random instances

Input parameters		Number of unsatisfied clauses		
n	pn	GS	LS	GSAT
1000	10000	1543	1405	1315
	15000	2562	2385	2280
	20000	3611	3409	3289
	25000	4680	4451	4344
	30000	5773	5488	5390

We see that the performance of all three algorithms virtually does not depend on the value of d; and, although the algorithms perform marginally better on d-degenerate instances with small value of d, the difference is too small to draw any conclusion. To verify our findings we simulate the greedy algorithm by differential equations, as shown in the next section.

Modeling the greedy algorithm

In this section, we construct a model of the algorithm GR using a system of differential equations. Such analy-

sis has been used before to study SAT and MAXSAT, see e.g. (Achlioptas 2001), (Bulatov and Skvortsov 2006), and (Amiri and Skvortsov 2007). We use Wormald's theorem (Wormald 1995). Informally, this theorem allows one to simulate random processes using a deterministic system of differential equations, provided the process satisfies certain conditions. Suppose the random process such as the work of a SAT algorithm is described by parameters $e_1, \dots, e_k$ that are considered to be random variables, and depend on a point of time. So, $e_i(t)$ is the value of the parameter e_i (say, the number of currently satisfied clauses) on t-th iteration of the algorithm. Then we try to evaluate the expectation of the difference $e_i(t+1) - e_i(t)$ for each parameter. The main requirement then is that the process must be Markov, that is, $\mathbb{E}[e_i(t+1) - e_i(t)]$ depends only on the current state of the process, the values $e_1(t), \dots, e_k(t)$, and does not depend on its history. Thus we express the expectations as functions of $e_1(t), \dots, e_k(t)$:

$$\begin{aligned} \mathbb{E}[e_1(t+1) - e_1(t)] &= f_1(e_1(t), \dots, e_k(t)), \\ &\dots \\ \mathbb{E}[e_k(t+1) - e_k(t)] &= f_k(e_1(t), \dots, e_k(t)). \end{aligned}$$

Two more conditions of the theorem are the following. First, the process must not behave 'wildly', that is the parameters describing the process must not experience sharp changes. More precisely, $e_i(t+1) - e_i(t)$ should be bounded, or, at least the probability that it is not bounded must be very small. Second, the functions $f_1, \dots, f_k$ must be differentiable functions whose derivatives are bounded within certain area (Lipschitz condition). Wormald's theorem then claims that if all the conditions are met, the process is well described by the system of differential equations

$$\begin{aligned} \frac{de_1}{dt} &= f_1(e_1, \dots, e_k), \\ &\dots \\ \frac{de_k}{dt} &= f_k(e_1, \dots, e_k), \end{aligned}$$

in the sense that the actual parameters of the process, for instance, the values of these parameters in a run of the algorithm, with high probability do not deviate far from the solution (with certain initial parameters) of the system of differential equations. Thus solving the system (analytically or numerically) we can prove that the results of the process with high probability are very close to certain values.

Suppose the algorithm GR is applied to a 2-CNF ϕ with n variables and pn clauses. Whether the formula is random d-degenerate, or just random only matters on the last stage of the analysis, so we do not specify it here. On each step i of the algorithm we introduce 3 values:

- $e_b(i)$ is the number of binary clauses in ϕ_{i-1};
- $e_u(i)$ is the number of unary clauses in ϕ_{i-1};
- $e_h(i)$ is the number of clauses in ϕ that contain only variables from $\{x_1, \dots, x_{i-1}\}$ and are not satisfied by the current partial assignment.

We find expectations of $e_b(i+1) - e_b(i)$, $e_u(i+1) - e_u(i)$, and $e_h(i+1) - e_h(i)$.

At step i let ℓ, c, k, g denote the number of unary clauses in ϕ_{i-1} containing x_i, the number of positive ones among them, the number of binary clauses in ϕ_{i-1} containing x_i, and the number of those that contain x_i without negation. Then

$$\mathbb{E}[e_b(i+1) - e_b(i)] = \mathbb{E}[k].$$

Let also u equals the probability $\mathsf{Pr}[x_i = 1]$ that the algorithm GR assigns 1 to x_i (probability here is meant over the choice of a random 2-CNF, as GR itself is deterministic). It is not hard to see that

$$u = \sum_{\substack{p,q,r,s \\ q-(p-q)+s-(r-s)\geq 0}} \mathsf{Pr}[\ell = p, c = q, k = r, g = s],$$

where the sum is taken over the combinations of values of ℓ, c, k, g that force the algorithm to assign 1 to x_i. Depending on whether x_i assigned 1 or 0, e_h increases either by $\ell - c$ or by c. Similarly, e_u increases either by $k - g - \ell$ or by $g - \ell$. Therefore

$$\begin{aligned} &\mathbb{E}[e_u(i+1) - e_u(i)] \\ &\quad = \mathbb{E}[k - g - \ell \mid x_i = 1] \cdot u + \mathbb{E}[g - \ell \mid x_i = 0] \cdot (1 - u), \\ &\mathbb{E}[e_h(i+1) - e_h(i)] \\ &\quad = \mathbb{E}[\ell - c \mid x_i = 1] \cdot u + \mathbb{E}[c \mid x_i = 0] \cdot (1 - u). \end{aligned}$$

Expectations of the form $\mathbb{E}[z \mid x_i = 1]$, where z is a function of ℓ, c, k, g can be found as

$$\begin{aligned} &\mathbb{E}[z \mid x_i = 1] \\ &\quad = \frac{1}{u} \sum_{\substack{p,q,r,s \\ q+(p-q)+s-(r-s)\geq 0}} z \cdot \mathsf{Pr}[\ell = p, c = q, k = r, g = s]. \end{aligned}$$

Thus, the problem reduces to computing probabilities of this kind. This can be done by fairly straightforward counting.

In the case of random instances $\mathbb{E}[k] = \frac{e_b(i)}{n-i+1}$, and finding $\mathsf{Pr}[\ell = p, c = q, k = r, g = s]$ is a simple exercise. In the case of random d-degenerate formulas let $D(n, m, d)$ denote the number of d-degenerate formulas with n variables and m clauses. These numbers are byproducts of the random formula generating algorithm from (Bauer, Krug, and Wagner 2010). Then

$$\mathbb{E}[k] = \frac{1}{D(n-i+1, e_b(i), d)} \sum_{s=1}^{d} s \cdot D(n-i, e_b(i) - s, d).$$

Probabilities of the form $\mathsf{Pr}[\ell = p, c = q, k = r, g = s]$ can be found in a similar way.

Now applying Wormald's theorem we can produce two systems of differential equations of the form

$$\begin{aligned} \frac{de_b}{dt} &= f_1(e_b, e_u, e_h), \\ \frac{de_u}{dt} &= f_2(e_b, e_u, e_h), \\ \frac{de_h}{dt} &= f_3(e_b, e_u, e_h), \end{aligned}$$

where f_1, f_2, f_3 are functions computed as shown above.

The remaining conditions of Wormald's theorem can be shown as follows. The differences $e_b(i+1) - e_b(i)$, $e_u(i+1) - e_u(i)$, $e_h(i+1) - e_h(i)$ are bounded by the number of clauses in which x_i occurs (the degree of x_i). It is well known that the probability that a random formula contains a variable of high degree is exponentially small. Lipschitz condition requires some attention, but it follows from the fact that all functions f_1, f_2, f_3 are elementary. Thus, we obtain the following

Theorem 4 *For any positive p and d, $d \geq p$, there are constants c_1, c_2 such that for a random 2-CNF sampled accordingly to distributions $\Phi^{2-\mathsf{SAT}}(n, pn)$ and $\Phi^{\mathsf{d-deg}}(n, pn, d)$ with high probability the GR algorithm finds an assignment which satisfies $c_1 n + o(n)$ and $c_2 n + o(n)$ clauses, respectively.*

The following tables show values c_1, c_2 obtained by solving numerically the systems of differential equations constructed above for some combinations of p and d. We compare these values with the results of tests described in the previous section. Thus the values in the row 'Experiment' are taken from Tables 1,2.

Table 3: Result on random formulas

p	10	15	20	25	30
c_1 (experiment)	1.54	2.56	3.61	4.68	5.77
c_2 (model)	1.57	2.60	3.66	4.74	5.84

Table 4: Result on random d-degenerate formulas

(p, d)	(10,11)	(10,30)	(15,17)	(15,35)
c (experiment)	1.53	1.54	2.57	2.60
c (model)	1.50	1.55	2.51	2.57
(p, d)	(20,23)	(20,40)	(25,30)	(25,45)
c (experiment)	3.58	3.60	4.68	4.67
c (model)	3.56	3.62	4.63	4.69
(p, d)	(30,37)	(30,50)		
c (experiment)	5.76	5.77		
c (model)	5.73	5.77		

References

Achlioptas, D. 2001. Lower bounds for random 3-SAT via differential equations. *Theor. Comput. Sci.* 265(1-2):159–185.

Amiri, E., and Skvortsov, E. 2007. Pushing random walk beyond golden ratio. In *CSR*, 44–55.

Bauer, R.; Krug, M.; and Wagner, D. 2010. Enumerating and generating labeled k-degenerate graphs. In *ANALCO*, 90–98.

Biere, A.; Heule, M.; van Maaren, H.; and Walsh, T., eds. 2009. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press.

Bjesse, P.; Kukula, J.; Damiano, R.; Stanion, T.; and Zhu, Y. 2003. Guiding SAT diagnosis with tree decompositions. In *SAT*, 315–329.

Bodlaender, H. L., and Kloks, T. 1991. Better algorithms for the pathwidth and treewidth of graphs. In *ICALP*, 544–555.

Bodlaender, H. L.; Gilbert, J. R.; Kloks, T.; and Hafsteinsson, H. 1991. Approximating treewidth, pathwidth, and minimum elimination tree height. In *WG*, 1–12.

Bodlaender, H. 2006. Treewidth: Characterizations, applications, and computations. In *WG*, 1–14.

Bulatov, A., and Skvortsov, E. 2006. Efficiency of local search. In *SAT*, 297–310.

Courcelle, B. 1990. Graph rewriting: An algebraic and logic approach. In van Leeuwen, J., ed., *Handbook of Theoretical Computer Science*. Amsterdam: Elsevier Science Publishers. chapter 5, 194–242.

Dechter, R., and Pearl, J. 1989. Tree clustering for constraint networks. *Artificial Intelligence* 38:353–366.

Freuder, E. 1982. A sufficient condition for backtrack-free search. *Journal of the ACM* 29(1):24–32.

Freuder, E. 1990. Complexity of k-tree structured constraint satisfaction problems. In *Proceedings of 8th National Conference on Artificial Intelligence AAAI-90*, 4–9.

Garey, M. R.; Johnson, D. S.; and Stockmeyer, L. J. 1974. Some simplified np-complete problems. In *STOC*, 47–63.

Gu, J. 1992. Efficient local search for very large-scale satisfiability problem. *ACM SIGART Bulletin* 3(1):8–12.

Håstad, J. 2001. Some optimal inapproximability results. *J. ACM* 48(4):798–859.

Pittel, B.; Spencer, J.; and Wormald, N. C. 1996. Sudden emergence of a giant-core in a random graph. *J. Comb. Theory, Ser. B* 67(1):111–151.

Robertson, N., and Seymour, P. D. 1986. Graph minors. ii. algorithmic aspects of tree-width. *J. Algorithms* 7(3):309–322.

Selman, B.; Levesque, H.; and Mitchell, D. 1992. A new method for solving hard satisfiability problems. In *AAAI*, 440–446.

Selman, B.; Mitchell, D. G.; and Levesque, H. J. 1996. Generating hard satisfiability problems. *Artif. Intell.* 81(1-2):17–29.

Thomassen, C. 1995. Decomposing a planar graph into degenerate graphs. *J. Comb. Theory, Ser. B* 65(2):305–314.

Trevisan, L.; Sorkin, G. B.; Sudan, M.; and Williamson, D. P. 2000. Gadgets, approximation, and linear programming. *SIAM J. Comput.* 29(6):2074–2097.

Williams, R.; Gomes, C. P.; and Selman, B. 2003. Backdoors to typical case complexity. In *IJCAI*, 1173–1178.

Wormald, N. 1995. Differential equations for random processes and random graphs. *The Annals of Applied Probability* 5(4):1217–1235.