Department of Computer Science
George Mason University Technical Reports

4400 University Drive MS#4A5
Fairfax, VA 22030-4444 USA
http://cs.gmu.edu/ 703-993-1530

Parallel Model Listing and #3SAT Through Ordering Variables

Michael Connor mconnor@gmu.edu
Fei Li lifei@cs.gmu.edu

Technical Report GMU-CS-TR-2013-1

Abstract

We consider listing and counting all the solutions M to a given Boolean 3SAT formula with n variables. This problem is known to be NP-hard (Garey and Johnson, 1979). A straight-forward enumeration scheme to find M takes worst-case running time $O(2^n)$. In this paper, we minimize the total running time and memory requirement by ordering the assignments of parallel values to the literals. Our objective is to establish theoretical bounds on an exponential running time $O(a^b)$ such that $a < 2$ and $b < n$ for certain Boolean formulas. We also conduct experiments to support our theoretical analysis.

1 Introduction

We consider Boolean operations over Boolean variables. A Boolean variable can have its value either **T** (TRUE) or **F** (FALSE). Let V denote a set of Boolean variables, $|V| = n$. A Boolean variable or its negative is called a *literal*. A *clause* is a disjunction of literals. A clause is TRUE (or is called *satisfied*) if at least one of its literals are TRUE. Let C denote a set of Boolean clauses of V. Let F denote a Boolean formula as a conjunction of $|C|$ clauses. A Boolean formula is TRUE (or is called *satisfied*) if all of its clauses are TRUE. Let a set of #SAT models that be denoted M. If every clause has exactly 3 literals from 3 variables, then this problem is called the 3*SAT problem*. We are going to calculate all the combinations M such that #3SAT$(F) = M$. That is, we assign **T/F** values to V such that all clauses C in F are satisfied.

In this paper, we consider listing and counting all the solutions M to a given Boolean 3SAT formula in a conjunctive normal form (CNF) with n variables. This problem is a well-known NP-hard problem [3] such that it is likely that no polynomial-time algorithm exists. A straight-forward enumeration scheme to find M takes worst-case running time $O(2^n)$. We take a new approach. We minimize the total running time and memory requirement by ordering the assignments of values in parallel to the literals. We establish theoretical bounds on an exponential running time $O(a^b)$, such that $a < 2$ and $b < n$ for certain $|V|$, $|C|$, and F. We also conduct experiments to support our theoretical analysis.

This work is organized as follows. Section 2 summarizes previous work. Our algorithms and their analysis are introduced in Section 3. Section 4 shows some experimental results of our algorithm.

2 Previous Work

The DPLL (initials of the authors Davis, Putnam, Logemann, and Loveland) algorithm [1, 2] is a complete, backtracking-based search algorithm for deciding the satisfiability of SAT problems. DPLL depends on the choice of *branching literal*, which is the literal considered in the backtracking step. As we can see, DPLL is not exactly an algorithm, but rather a family of algorithms, one for each possible way of choosing the branching literal. DPLL's efficiency is strongly affected by the choice of the branching literal. The worst-case running time complexity is $O(2^n)$ and worst-case space requirement is $O(n)$. Our proposed algorithm in Section 3 adopts a core heuristic of the DPLL-based #SAT solvers and it wisely selects branching literals with reduced running time and space requirement.

Sang, Beame, and Kautz [5] used a few widely known branching heuristics from DPLL-based SAT solvers which we summarize briefly here. *Literal count heuristics* [6], which count the appearance of the positive or negative literals or the sum or the difference of both, and select the highest scoring literal at each successive step. Heuristics for #SAT and SAT differ in their experimental performance [5]. Randomization is a hallmark of DPLL SAT solvers, but has no utility to DPLL-based #SAT solvers as all sections of the search space must be searched anyway [5]. Sang, Beame, and Kautz [5] also tested randomization and discovered that it hurt performance, and so we avoid it entirely in our algorithm design and analysis. All DPLL-based SAT and #SAT

solvers must employ backtracking of some variety. but we have no use for this as both literal choices are made in parallel.

The size of M can actually be loosely described, depending on some parameters. In Krivelevich, Sudakov, and Vilenchik's work [4], they first discussed dense formulas with a clause to variable ratio r_F such that $r_F \geq C_0$, for a large constant C_0. They showed that such formulas cluster around solutions forming connected components. For even denser formulas with a ratio $r_F \geq C_0 \log n$, they show that such formulas have only one solution with a high probability.

3 Algorithm and Analysis

Let m_t denote the set of models that we have at time t. Let v_t denote the branching variable selected to be assigned a value at time t and c_t the number of clauses selected at time t. In each time step, we will assign both **T/F** values to a Boolean variable $v_t \in V$. Initially, set $m_0 = 0$, $v_0 = \varnothing$, and $c_0 = 0$. When our algorithm terminates, we will have $m_n = M$, $\{v_1, \ldots, v_n\} = V$, and $\sum_{t=1}^{n} c_t = |C|$.

3.1 Algorithm

Let us start from the enumeration method to develop our ideas to deal with the #3SAT problem. At each step t, we (*a*.) duplicate m_t, (*b*.) assign TRUE to a branching variable v_t for one set m_t and assign FALSE to the same branching variable v_t for the other set m_t, and (*c*.) use c_t existing clauses to eliminate unsatisfiable models from the two halves. Concurrently assigning TRUE and FALSE to v_t is to make our algorithm running in parallel. The traditional approach is to assign values **T** or **F** and then to proceed down one branch, keeping a configuration with a maximum length $|V|$ in memory and a maximum running time of 2^n steps. Instead we assign **T** and **F** in parallel and proceed down both branches, keeping at maximum 2^n configurations in memory and taking $|V|$ steps, which is a transpose of the traditional approach. After we assign values to a variable, double the models and then eliminate some unsatisfiable ones, we have a new set of models m_{t+1} and the running time for this step is bounded by $2 \cdot |m_t| = O(|m_t|)$. As we have $n = |V|$ variables, we conduct at most n steps during the course of assigning Boolean values to variables. The total running time is then

$$Z := \sum_{t=1}^{n} |m_t|.$$

As we can see from the above enumeration method, we should minimize the cumulative size $|m_t|$ of the models over time. How to order variables in the assignment procedure determines the number of clauses and of models and thus, determines the total running time Z. Define

$$\begin{aligned} r_t &:= \frac{m_{t+1}}{m_t} \\ r_{\max} &:= \max_t r_t. \end{aligned}$$

Obviously, $0 \leq r_t \leq 2$. We have the total running time Z bounded by $n \cdot (r_{\max})^{n-1}$. Our algorithm is based on the ideas of wisely selecting v_{t+1} to assign a value such that $r_{\max}$ is minimized.

One insight is that if there are sufficiently many clauses per variable to effectively constrain each additional variable, then *there exists a value-assignment path to M where every additional variable includes at least one clause satisfiable*. The *variable density* or the *variable connectivity ordering* determines the priority of selecting variables to assign values.

We adapt a greedy approach to select variables to assign values. The favored variables are those that occur in the formula more often, also called *literal count heuristic*. If we select the variables with the highest literal counts first, then *we can include more clauses with fewer variables*. Our algorithm is described in Algorithm 1.

Algorithm 1 Parallel Ordering

1: Duplicate m_t from v_0 to v_t as m_{t+1}.
2: Identify v_{t+1} based on the decreasing order of the literal counts of V.
3: Set v_{t+1} to **T** in one copy of m_t in m_{t+1} and **F** in the other copy.
4: Use c_{t+1} clauses to remove unsatisfactory models from m_{t+1}.
5: **return** m_{t+1}.

3.2 Analysis

In this subsection, we analyze Algorithm 1's performance in terms of expected running time and memory requirement. Recall that in our algorithm and analysis, we consider 3CNFs. We start with the preliminary results of random expectation and then study the effects of our algorithm by means of dense regions and literal expectation.

3.2.1 Preliminary results: random expectation

The expectation of a newly added variable adding clauses is explained below using an example of a formula with 10 clauses and 10 variables — each variable appears as 3 literals in 3 clauses, a uniform distribution.

There are bounds for this expectation as the first two variables will make 0 clauses satisfied or unsatisfied, and the last variable added will make 3 clauses satisfied or unsatisfied immediately. For our purposes *the variable*

order within a clause does not matter, nor does the clause order in the formula, not does the literal value.

The expectation of adding the third variable depends on the possible filled clauses divided by the possible configurations. For $t = 3$, there is a maximum of 1 clause that can be filled in but in 10 locations, with the other variables in some combination of configurations; and these are only a few of all of the configurations of the two variables. For $t = 4$, there is a maximum of 3 clauses that can be filled but also possibilities for 2 or 1 or 0 clauses to be filled. So the probable filled clauses are an average of the configurations with 3, 2, 1, and 0 clauses. The probability of these happening is predicated on the probability of the placed variables being in their combinations in clauses. For example, for 3 clauses to be filled at $t = 4$, there is only 1 configuration. There is 1 configuration for 3 clauses to be filled, but there are 3 configurations for 2 clauses to be filled and 3 for 1 clause as well and 1 for 0 clauses.

For each literal, it can be the variable itself or this variable's negative. Given V variables (that is, $2|V|$ literals), there are $8 \cdot \binom{V}{3}$ possible 3CNF clauses. For simplicity, we use V to represent $|V|$ and C for $|C|$ in equations, and we will drop the literal coefficients: so there are $\binom{V}{3}$ clauses (positive clauses) and $|V|$ literals (sum of both values).

Another view of the expectation of a newly added clauses that come with an additional variable is as follows. We can assume that the addition of variables into $|C|$ will occur at the same rate as in the maximum available clauses, $\binom{V}{3}$. Start with V variables, say 10, and instead of 10 clauses we use $\binom{10}{3}$, or 120 clauses. Now the clauses, c_t, added with each variable is $c_t = \binom{t}{2}$ with $c_0 = 0$ and $\{c_1, c_2, \ldots, c_9\} = \{0, 1, 3, 6, 10, 15, 21, 28, 36\}$. And for each v_t there is an accompanying percentage of clauses from the total that accompany it. If the clauses are distributed randomly from the set of total possible clauses then their incremental addition should follow the same percentages. The random expectation of clauses from all possible clauses is $\binom{t}{2}/\binom{V}{3}$ from 1 to $|V|$ summing up to 1. So at a scaling of $|C|$, the expectation sums to $|C|$.

$$E(c_t) = \frac{|C| \cdot \binom{t}{2}}{\binom{V}{3}}.$$

At $v_t = \frac{|V|}{2}$, $c_t = 0.75 \cdot c_{avg}$ clauses are added. At $v_t = \frac{|V|}{\sqrt{3}}$, $c_t = c_{avg}$ clauses are added. For $\max(m_t) = \prod_t r_t$, $c_t > 1$; we have $\max(m_t) = 2^{5n/9}$.

Figure 1 illustrates this random expectation of a uniform distribution of literals to variables, of clauses to variables, and of r_t to variables. The area under the $L(v)$ $[l(v)]$ curve sums up to $3 \cdot |V| \cdot c_{avg}$. The area under the c_t $[c(v)]$ curve sums up to $|C|$. The area under the r_t $[r(v)]$ curve sums to the log of $\prod_t r_t$ from 1 to $|V|$.

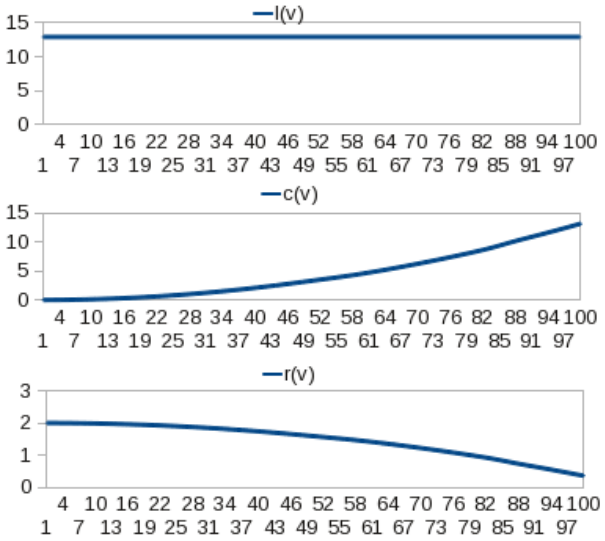

Figure 1: Uniform Distribution of $l(v)$, $c(v)$, and $r(v)$

3.2.2 Density expectation

There exist subsets of clauses that include fewer variables to form a dense region. The extent of the dense region is directly proportional to the density of the formula. This provides a path to M where every variable includes at least one clause and results in a constraint on a in $O(a^b)$ beyond the random expectation.

There are $\binom{V}{2}$ pairs of variables and by combining them in triples of pairs with 3 variables, there are no more than $\frac{\binom{V}{2}}{\binom{3}{2}}$ unique clauses that may be formed without sharing more than 1 variable, R_1. Thus $\binom{V}{3} - \frac{\binom{V}{2}}{3}$ clauses have at least 2 variables in common with other clauses, R_2. We have

$$\begin{aligned} R_1 &:= \frac{\binom{V}{3}}{\frac{\binom{V}{2}}{3}} \\ &= \frac{1}{V-2} \\ R_2 &:= 1 - R_1 \\ &= \frac{V-3}{V-2} \end{aligned} \tag{1}$$

where R_2 is a measure of the density of possible clauses given V variables, which seems very dense.

Let us discuss Equation (1). It seems R is dense. However, only an extremely small subset of possible clauses are present in a CNF formula. The Birthday Paradox illustrates that this subset is not dense. For C clauses and for $3 \cdot C$ pairs of variables, the probability of 1 repeated pair is $\frac{\binom{V}{2}!}{\binom{V}{2}^{3C} \cdot (\binom{V}{2} - 3C)!}$ or $\prod_{k=1}^{3C} \left(1 - \frac{k}{\binom{V}{2}}\right)$.

An example is that for 100 variables, 84 pairs or 28 clauses are required for an even chance of 2 clauses sharing more than 1 variable. So how many repeats are

within the clauses of a typical 3CNF formula? After 1 repetition has reached its 0.5 chance, given the current variables how many additional variables provide the next 0.5 chance? The $\log_{0.5}$ of the probability shows the number of potential repetitions that produce pairs. Since they all have 0.5 chance, we can expect half of them.

$$E(Pairs) \approx \frac{1}{2}\log_{0.5}\left(\prod_{k=1}^{3C}\left(1-\frac{k}{\binom{V}{2}}\right)\right)$$

A more precise estimate would take into account that variables/pairs within a clause cannot repeat each other so that pattern of

$$1\cdot\frac{n-1}{n}\cdot\frac{n-2}{n}\cdot\frac{n-3}{n}\cdot\frac{n-4}{n}\cdots$$

adjusts to

$$1\cdot 1\cdot 1\cdot\frac{n-3}{n}\cdot\frac{n-3}{n}\cdot\frac{n-3}{n}\cdot\frac{n-6}{n}\cdot\frac{n-6}{n}\cdots$$

Also, repetitions do not generate additional pairs to match, so probable repeating should be accounted to subtract from the available matches. This yields the following.

$$E(Pairs_i) = \left[\frac{3}{2}\log_{0.5}\left(\prod_{k=1}^{C}\left(1-\frac{k+E(Pairs_{i-1})}{\frac{\binom{V}{2}}{3}}\right)\right)\right]$$

In order to account for the fact that variables can not repeat each other in the same clause, the 3 in $3C$ has moved outside of the product and under $\binom{V}{2}$. Also, the previous $E(Pairs_i)$ has been added to k to account for repeated pairs, turning this into a recursive algorithm.

3.2.3 Dense regions

For example, 100 variables in 430 clauses should expect 123 pair repetitions. *This number of repetitions stabilizes as $|V|$ increases*: for 500 variables and 2150 clauses the expectation is 120 and for 1000 variables and 4300 clauses as well. The expectation of pairs depends on the ratio of $|C| : |V|$ or c_{avg}.

As stated before, there are around 120 pairs given that $c_{avg} = 4.3$. These pairs could be shared by a very few variables or stretched to a chain of 120 variables. By starting the addition of variables from variables in this dense region, there will always be at least 1 clause per variable. The chain's form depends on F so we scale by an unknown constant α, to produce the effect on r_t:

$$r_t = 2 - \frac{c_t}{8} - \frac{\alpha\cdot 120}{|V|}$$

For formulas with a low $|V|$, $\frac{\alpha\cdot 120}{|V|} \geq 0.25$ and $r_t \leq 1.75$. When $t = \frac{|V|}{6}$, $c_t \simeq 1$, so the path of variables with at least one clause only has to reach $1/6$ of the way. For all t, $c_t > 1$ with high probability while $|V| < \alpha\cdot 720$. For such dense formulas this r_t shifts the c_t curve to the left. The degree of the shift of c_t is proportional to the square of the increase in r_t, and so a formula with $x\cdot|V|$ variables and $\sqrt{x}\cdot|C|$ clauses has as many pairs per variable as a formula with $|V|$ and $|C|$. The reduction of $r_{\max}$ to 1.75 is a constraint of $a \leq 1.75$ on $O(a^b)$. The effect on M is that $\max(m_t) = 2^{\frac{2n}{5}}$ or $\max(m_t) = 1.75^{\frac{n}{2}}$.

3.2.4 Literal expectation

The majority of the literals reside in the minority of the variables. By selecting them first, we include more clauses early and further restrict the growth of M. In this way, we result in a constraint on b in $O(a^b)$, which beats a random expectation.

The expectation here is cast as the addition of literals. In a uniform distribution, each variable adds $\frac{3\cdot C}{V}$ literals. For the expectation of a variable with a random number of literals, $L(v_t)$, and a clause to variable ration of c_{avg}, we scale the expectation by its number of literals over the random expectation, or $\frac{L(v_t)}{3\cdot c_{avg}}$.

In order to map the accumulation of literals to that average variables we sum the literals and then divide by the formulas average literal to variable ratio, $\frac{1}{3\cdot c_{avg}}\sum_{i=1}^{t} L(v_i)$, to represent the closest v_t, or the apparent variable to substitute for t in $\binom{t}{2}$. The expectation of the literals, $L(v_t)$, depends on a normal distribution of literals among variables. It can be modeled as the reverse of the inverse of the cumulative distribution function, i.e., for 100 variables adding the 4-th variable uses that inverse at 0.96 and the 5-th variable uses 0.95.

$$E(c_t) = \frac{\frac{L(v_t)}{3\cdot c_{avg}}\cdot|C|\cdot\binom{\frac{1}{3\cdot c_{avg}}\sum_{i=1}^{t} L(v_i)}{2}}{\binom{V}{3}}$$

We may also consider how many variables are required until half of the literals are present or how many literals are present when the larger half of variables are present and what those mean for clauses. Scaling the literals shifts the c_t curve to the left. Replacing variables with a normal distribution of literals shifts the c_t curve to the left as well. The amount of left shift depends on the standard deviation of the $L(v)$. With c_{avg} set to 4.3, the mean literal to variable ratio is 12.9 and we use a standard deviation of 3.2 from a test formula.

At $v_t = \frac{|V|}{2}$, $c_t > c_{avg}$ clauses are added. If we redistribute the addition of variables as a reflection of the addition of literals, then we get a new $\max(m_t) = 1.75^{\frac{2\cdot n}{5}}$.

In Figure 2 we see the left shift of $L(v)$ and its effect on c_t and r_t The sum of $L(v)$ is still $3\cdot|V|\cdot c_{avg}$ and the sum of c_t is still $|C|$, but here r_t takes a sharper dip and then increases at the end. The area under the r_t curve from 1 until r_t increases sums up to the log of $\prod_t r_t$ at

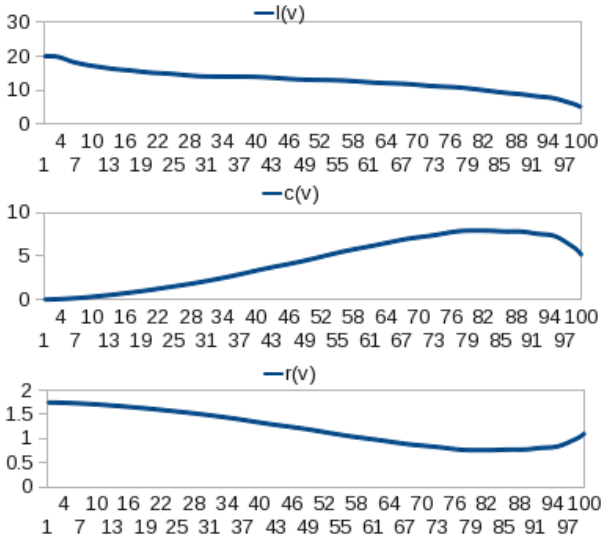

Figure 2: Inversion Cumulative Distribution of $l(v)$, $c(v)$, and $r(v)$

$\max_t m_t$. The increase of r_t at the end occurs because $L(v)$ decreases which also causes c_t to decrease.

4 Experiments

For our tests we use an algorithm that uses the above properties and a few more properties which may be presented in later papers. The reason for this is that $O(a^b)$ can still produce intractably large m_t and in order to expand the scope of tests, we need to reduce Z as much as possible.

The maximum model count does not reach 2^n, on the contrary it only goes so far as $2^{n/5}$. We ran test sets for $n = 100$ and $n = 125$. For the tests 100-109 on $n = 100$, m_t reached a maximum of 932152 or $2^{19.8}$ on average with a range of maximums from 148836 to 1596104. For the tests 100-91 on $n = 125$, m_t reached a maximum of 92674069 or $2^{26.5}$ on average with a range of maximums from 17811821 to 164137776. Notice that m_t grows at a decreasing exponential rate that tapers off. The graph in Figure 3 is of $n = 100$, the y-axis is m_t.

The decrease in growth in Figure 4 also proceeds at a steady pace. The start at $\widetilde{1.75}$ (≈ 1.75) represents the addition of 1 clause with the addition of each variable. The drop down to below 1.4 occurs at the partitioning and creates a shelf of > 2 clauses per variable that persists until it rejoins the regular slope. This slope crosses 1 at around $n/2$.

We used this more efficient algorithm to run the following demonstration in a reasonable time for Figure 5. To demonstrate the effect of dense regions we ran formulas with various values for $|C|$ while holding $|V|$ constant, which produces different values for c_{avg} (y-axis), and the x-axis is the maximum of m_t on a logarithmic

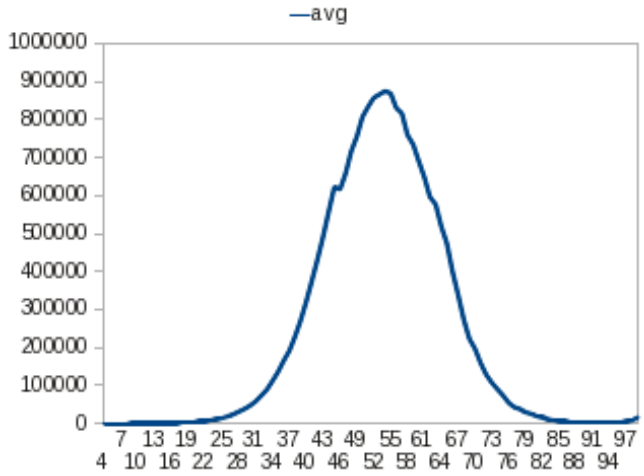

Figure 3: An sample average m_t over $|V|$

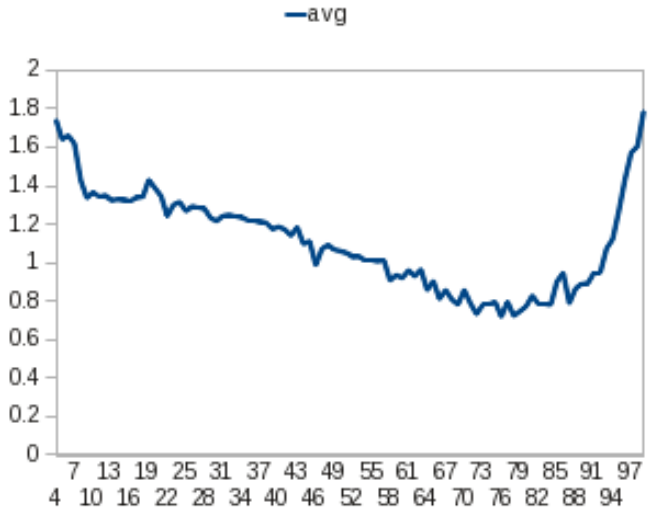

Figure 4: An sample average r_t over $|V|$

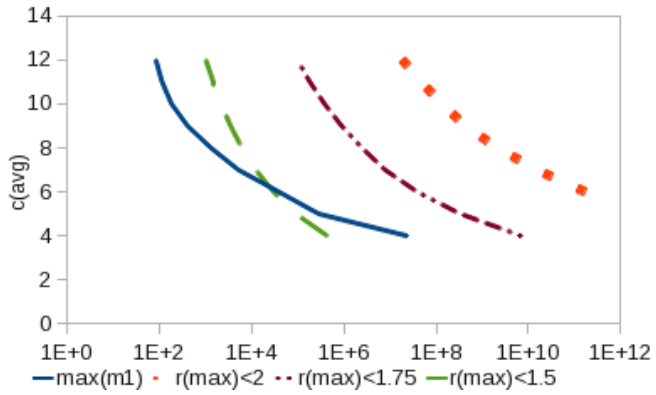

Figure 5: The effect of dense regions

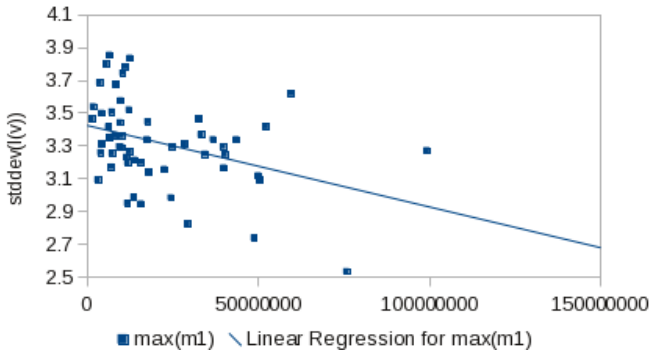

Figure 6: Literal distributions

scale. The line for $\max(m1)$ represents the average of 10 tests for each value of c_{avg} from 4 to 12. When $c_{avg} < 4$, m_t becomes intractably large. The other three contour lines represent the expectation of $\max(m_t)$ where $r_{\max}$ is set to simulate an effect of the dense regions. Shortly below 4, $\max(m_t)$ crosses the middle contour line. As c_{avg} increases, $\max(m_t)$ moves left and crosses the 1.5 contour line. The contour lines can be drawn at $(1, 2]$. Near 1, $|C|$ represents all possible clauses of $|V|$ variables, and at 2 $|C|$ is .

For a in $O(a^b)$, the density of the variables in the clauses reduces it by .25 for every clause per variable provided by the path to M along repeated pairs. *So with the function of the path* $p(F) = clause/variable \cdot 0.25$, *there is a big-O of* $O((a - p(F))^b$.

The next series of tests measure the effect of literals in the formula; Figure 6. The noise of the points is due to the differences in each test formula. As the literals differentiate themselves farther with a larger standard deviation, the maximum m_t has a trend to decrease. As the standard derivation increases, the number of variables that contain the majority of literals decreases.

For b in $O(a^b)$, there is an effect without such a clear mathematical relationship. So for a function of the exponent b, all we can say is that $b = l(n, \sigma)$, for a big-O of $O((a - p(F))^{l(n,\sigma)}$.

References

[1] M. Davis, G. Logemann, and D. Loveland. A Machine Program for Theorem-Proving. *CACM*, 5(7):394-397, 1962

[2] M. Davis and H. Putnam. A Computing Procedure for Quantification Theory. *Journal of the ACM*, 7:201-215, 1960.

[3] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman, 1979.

[4] M. Krivelevich, B. Sudakov, and D. Vilenchik. On the random satisfiable 3CNF process.

[5] T. Sang, P. Beame, and H. Kautz. Heuristics for Fast Exact Model Counting. In *Proceedings of the 8th International Conference on Theory and Applications of Satisfiability*. 226-240. 2005

[6] J. Silva. The impact of branching heuristics in propositional satisfiability algorithms. In *Proceedings of the 9th Portuguese Conference on Artificial Intelligence: Progress in Artificial Intelligence*, pages 62-74, 1999.