Enumerating Satisfiable Propositional Formulæ

Nachum Dershowitz[*] Mitchell A. Harris[†]

October 30, 2003

Abstract

It is known experimentally that there is a threshold for satisfiability in 3-CNF formulæ around the value 4.25 for the ratio of variables to clauses. It is also known that the threshold is sharp [Fri99], but that proof does not give a value for the threshold.

We use purely combinatorial techniques to count the number of satisfiable boolean formulæ given in conjunctive normal form. The intention is to provide information about the relative frequency of boolean functions with respect to statements of a given size, and to give a closed-form formula for any number of variables, literals and clauses. We describe a correspondence between the syntax of propositions to the semantics of functions using a system of equations and show how to solve such a system.

1 Introduction

The purpose of this paper is to apply combinatorial techniques to count the number of satisfiable boolean formulæ for a given syntax and provide information about the relative frequency of boolean functions with respect to statements of a given size. This in turn may help one understand the performance of algorithms that decide problems such as satisfiability and validity,

[*]School of Computer Science, Tel Aviv University, Ramat Aviv, Tel Aviv 69978, Israel. Supported in part by the Israel Science Foundation. Email: `nachumd@tau.ac.il`

[†]Department of Computer Science, Technical University of Dresden, 01062 Dresden, Germany. Email: `maharri@tcs.inf.tu-dresden.de`

and may aid in finding bounds on the threshold between satisfiability and unsatisfiability. The method we use is an explicit counting of those formulæ that are satisfiable. We restrict ourselves to k-CNF form, and describe a correspondence between this syntax of propositions and the semantics of the boolean functions they represent, using a system of equations. Then we show how to solve such a system, giving a general closed-form solution. For the traditional counting model for literals within a clause (unordered without replacement, no contradictory literals), we simplify it to a more specific solution.

There is great experimental evidence for the phenomenon of a 'phase transition' of satisfiability. That is, there is an experimentally confirmed threshold between satisfiable and unsatisfiable formulæ as the ratio of clauses to variables increases, where below the threshold almost all formulæ are satisfiable and beyond almost all are unsatisfiable (see Kilpatrick and Selman [KS94]). Much work has been done to establish that this phase transition is indeed a sharp threshold (see Friedgut [Fri99], with qualifications as to "sharpness") and to place upper and lower bounds on that threshold (see Janson et al. [JSV00] for a summary of the progress on those bounds). The closed form provided here could perhaps be used to help determine the exact threshold analytically.

The importance of the threshold is that, again experimentally, it seems that the running times of the Davis-Putnam algorithm for deciding satisfiability peak around the threshold. The intent of the present paper is to determine exact probabilities.

Dershowitz and Lindenstrauss [DL89] use generating function techniques for counting with boolean formulæ; that method is extended here to solving systems of equations counting boolean functions generated from a given syntax. Chauvin, Flajolet et al. [CFGG02] form a similar set of equations for the case of unrestricted syntax.

2 Syntax and Semantics

The counting problem we address corresponds to the logical problem "k-CNF-SAT". We have a set V of v independent propositional variables, and a set $\overline{V}$ of their v negations. Variables and their negations are called *literals*. A *clause* is a disjunction of a sequence of k literals, and a *boolean formula* is a conjunction of a sequence of c clauses. Most of the literature on k-CNF

clause size, k	k	1	1	1	k	k
clauses, c	1	c	1	c	1	c
vars, v	1	1	v	v	v	1
total # formulæ	2^k	2^c	$2v$	$(2v)^c$	$(2v)^k$	2^{kc}
# satisfiable	2^k	2	$2v$	$\sum_{j=1}^{v} \binom{v}{j} 2^j \left\{ {c \atop j} \right\} j!$	$(2v)^k$	$2(2^k-1)^c - (2^k-2)^c$

Table 1: Counting satisfiable formulæ for small values of v, k, and c.

views a clause as a set of literals (unordered without replacement), but a formula as a sequence of clauses. We use sequences instead of sets implying that a literal or clause may repeat within a clause or formula, respectively, and that the order of the literals or clauses matter. For example,

$$(p \vee q \vee p) \wedge (\overline{p} \vee q \vee q) \wedge (p \vee \overline{p} \vee \overline{q}) \wedge (q \vee p \vee p)$$

is in 3-CNF form with 4 clauses and 2 variables.

Because of the restricted syntax, the set of k-CNF formulæ is straightforward to specify as a regular language: $((V + \overline{V})^k)^c$. So the total number of formulæ over v variables, k literals, and c clauses is $(2v)^{kc}$. From this one can quickly derive the number of satisfiable formulæ for degenerate and small values of k, c, and v.

There are two nontrivial entries. For $k = 1$, to be satisfiable, a literal and its negation cannot appear. So, for the number of variables appearing, $t \geq 1$, we choose first the variables, $\binom{v}{t}$, then their sign, 2^t, then their locations, $\left\{ {c \atop t} \right\} t!$, the number of set partitions of c of size t (Stirling numbers of the second kind), where the order of the partition matters, that is, the number of surjective functions from c to t.

For $v = 1$, we consider the set of four boolean functions and how they are produced syntactically. First, there are 2^k possible clauses and none of them can be $\mathbf{F}$.[1] There is exactly one way to produce $\mathbf{P}$ (likewise $\overline{\mathbf{P}}$) in a single clause (by having all the literals identical), and so there are $2^k - 2$ ways to produce $\mathbf{T}$. In a sequence of c of these clauses, to get $\mathbf{T}$, all the clauses must be $\mathbf{T}$, so there are $(2^k-2)^c$ ways for $\mathbf{T}$. One can only get $\mathbf{P}$ from a sequence

[1] We use **BOLD CAPITAL** notation for both the boolean function itself and the number of formulæ representing that function.

f_0	$= f_{00}$	$= \mathbf{F}$
f_1	$= f_{01}$	$= \overline{\mathbf{P}}$
f_2	$= f_{10}$	$= \mathbf{P}$
f_3	$= f_{11}$	$= \mathbf{T}$

(a)

$\vee$	$\mathbf{F}$	$\overline{\mathbf{P}}$	$\mathbf{P}$	$\mathbf{T}$
$\mathbf{F}$	$\mathbf{F}$	$\overline{\mathbf{P}}$	$\mathbf{P}$	$\mathbf{T}$
$\overline{\mathbf{P}}$	$\overline{\mathbf{P}}$	$\overline{\mathbf{P}}$	$\mathbf{T}$	$\mathbf{T}$
$\mathbf{P}$	$\mathbf{P}$	$\mathbf{T}$	$\mathbf{P}$	$\mathbf{T}$
$\mathbf{T}$	$\mathbf{T}$	$\mathbf{T}$	$\mathbf{T}$	$\mathbf{T}$

(b)

$\vee$	$\mathbf{F}_1$	$\overline{\mathbf{P}}_1$	$\mathbf{P}_1$	$\mathbf{T}_1$
$\mathbf{F}_{k-1}$	$\mathbf{F}_k$	$\overline{\mathbf{P}}_k$	$\mathbf{P}_k$	$\mathbf{T}_k$
$\overline{\mathbf{P}}_{k-1}$	$\overline{\mathbf{P}}_k$	$\overline{\mathbf{P}}_k$	$\mathbf{T}_k$	$\mathbf{T}_k$
$\mathbf{P}_{k-1}$	$\mathbf{P}_k$	$\mathbf{T}_k$	$\mathbf{P}_k$	$\mathbf{T}_k$
$\mathbf{T}_{k-1}$	$\mathbf{T}_k$	$\mathbf{T}_k$	$\mathbf{T}_k$	$\mathbf{T}_k$

(c)

Table 2: (a) The four binary functions of one variable; (b) the operator $\vee$ on them; (c) the recurrence for binary operator $\vee$ for one variable.

of $\mathbf{P}$ and $\mathbf{T}$ clauses with at least one $\mathbf{P}$ clause. So we subtract the completely true sequences from $(2^k - 1)^c$, for a total $(2^k - 1)^c - (2^k - 2)^c$. The number for $\overline{\mathbf{P}}$ is the same, so the total of satisfiable formulæ is:

$$2((2^k - 1)^c - (2^k - 2)^c) + (2^k - 2)^c = 2(2^k - 1)^c - (2^k - 2)^c$$

For arbitrary (positive integral) values of all three parameters and for every one of the 2^{2^v} boolean functions, we will count the number of formulæ (by number of literals and clauses) that evaluate to each function. For $v = 1$, there are 4 functions, $\mathbf{F}$, $\overline{\mathbf{P}}$, $\mathbf{P}$, and $\mathbf{T}$. For an arbitrary number of variables, we index a function by the binary representation of the integer corresponding to its truth table (see Table 2(a)).

For any given boolean operator (here we restrict ourselves to $\vee$ and $\wedge$, but the method can be applied to any operator), the function produced depends only on the functions represented by the two operands. For example, $\vee$ has the behavior shown in Table 2(b). The binary function $\wedge$ has the obvious dual behavior.

3 The enumeration

From the table of a logical operator acting on boolean functions, one can construct a system of equations whose solution is the number of formulæ for each function.

3.1 A system of equations

For the moment, let us consider the functions that can be represented with a single clause over one variable.

Let $\mathbf{F}_k$ be the number of formulæ for the boolean function f using k literals, $\vee$, and a single variable. Given a clause of length $k-1$, we can compute $\mathbf{F}_k$ by constructing a recurrence from Table 2(c):

$$\begin{aligned}
\mathbf{F}_k &= \mathbf{F}_1\mathbf{F}_{k-1} \\
\overline{\mathbf{P}}_k &= \mathbf{F}_1\overline{\mathbf{P}}_{k-1} + \overline{\mathbf{P}}_1(\mathbf{F}_{k-1} + \overline{\mathbf{P}}_{k-1}) \\
\mathbf{P}_k &= \mathbf{F}_1\mathbf{P}_{k-1} + \mathbf{P}_1(\mathbf{F}_{k-1} + \mathbf{P}_{k-1}) \\
\mathbf{T}_k &= \mathbf{F}_1\mathbf{T}_{k-1} + \overline{\mathbf{P}}_1(\overline{\mathbf{P}}_{k-1} + \mathbf{T}_{k-1}) + \\
&\qquad \mathbf{P}_1(\mathbf{P}_{k-1} + \mathbf{T}_{k-1}) + \mathbf{T}_1(\mathbf{F}_{k-1} + \overline{\mathbf{P}}_{k-1} + \mathbf{P}_{k-1} + \mathbf{T}_{k-1})
\end{aligned}$$

with base cases

$$\begin{aligned}
\mathbf{F}_1 &= 0 \\
\overline{\mathbf{P}}_1 &= 1 \\
\mathbf{P}_1 &= 1 \\
\mathbf{T}_1 &= 0
\end{aligned}$$

the ones produced by the literals, zeroes elsewhere. Note that a solution for this system is also a solution for a system based on $\wedge$, but with functions ordered in reverse (the complement of the bitwise representation).

The linear system can be represented as a matrix of coefficients for the recurrence:

$$\begin{bmatrix} \mathbf{F}_k \\ \overline{\mathbf{P}}_k \\ \mathbf{P}_k \\ \mathbf{T}_k \end{bmatrix} = \begin{bmatrix} f_{00} & f_{01} & f_{10} & f_{11} \\ 0 & f_{00}+f_{01} & 0 & f_{10}+f_{11} \\ 0 & 0 & f_{00}+f_{10} & f_{01}+f_{11} \\ 0 & 0 & 0 & f_{00}+f_{01}+f_{10}+f_{11} \end{bmatrix}^k \cdot \begin{bmatrix} \mathbf{F}_1 \\ \overline{\mathbf{P}}_1 \\ \mathbf{P}_1 \\ \mathbf{T}_1 \end{bmatrix}$$

Letting $\bar{f}_k$ be the vector for the set of all boolean functions formed by a clause of length k, and $\mathbf{OR}(1)$ be the above matrix, the equation

$$\bar{f}_k = \mathbf{OR}(1)^k \cdot \bar{f}_1$$

can be solved by simply multiplying out the matrix for fixed k, or, symbolically, by Gaussian elimination. But we would like to solve the system symbolically for an arbitrary number of variables. That the system is linear is a direct consequence of the grammar for k-CNF being regular.

The linear system for an arbitrary number of variables can be described recursively.

Definition 1

$$\begin{aligned} or(n) &= \begin{bmatrix} 0:or(n-1) & 1:or(n-1) \\ 0 & 0:or(n-1)+1:or(n-1) \end{bmatrix} \\ i:or(0) &= f_i \end{aligned}$$

where $x : or(n)$ appends the digits x to the front of all digits referred to in the matrix $or(n)$ and $+$ is matrix addition.

Since we are concerned with $n = 2^v$, we end up with a system of 2^{2^v} equations.

Definition 2

$$\mathbf{OR}(v) = or(2^v)$$

For instance,

$$\mathbf{OR}(0) = or(1) = \begin{bmatrix} 0:or(0) & 1:or(0) \\ 0 & 0:or(0)+1:or(0) \end{bmatrix} = \begin{bmatrix} f_0 & f_1 \\ 0 & f_0+f_1 \end{bmatrix}$$

$$\mathbf{OR}(1) = or(2) = \begin{bmatrix} 0:or(1) & 1:or(1) \\ 0 & 0:or(1)+1:or(1) \end{bmatrix} =$$

$$\begin{bmatrix} \begin{bmatrix} f_{00} & f_{01} \\ 0 & f_{00}+f_{01} \end{bmatrix} & \begin{bmatrix} f_{10} & f_{11} \\ 0 & f_{10}+f_{11} \end{bmatrix} \\ 0 & \begin{bmatrix} f_{00} & f_{01} \\ 0 & f_{00}+f_{01} \end{bmatrix} + \begin{bmatrix} f_{10} & f_{11} \\ 0 & f_{10}+f_{11} \end{bmatrix} \end{bmatrix}$$

as seen above.

All we need now is to justify that the recursively constructed system counts the functions as expected:

Theorem 1 *If f is the vector that counts the boolean functions on v variables, and f_i^n is a conjunction of n such functions, then*

$$f^n = \mathbf{OR}(v)^{n-1} \cdot f$$

Proof: We'll work with $or(n)$ first; the result follows immediately for $\mathbf{OR}(2^n)$. The matrix $or(n)$, and so also $or(n)^k$, must be upper triangular, since for $i \preceq j$, there is no x such that f_i will contribute to $f_x \vee f_j$. The $[i,j]$ entry of $or(v)$ is the sum of all the f_x such that $f_x \vee f_i = f_j$. This leads to 3 entries in $or(n+1)$: in the upper left quadrant, a zero bit has been added to the truth table representation, so nothing has changed; in the upper right, the additional bit is one and so only those functions with 1 in the most significant place are added; and in the lower right, the extra bit can be either 0 or 1 and so both sets of functions are added. □

The system for $\mathbf{AND}(v)$ is dual, so we can use the same matrix, reversing the indices for the entries.

Notice how we are being more general than just using the traditional literals by allowing an atomic symbol for any function.

3.2 Solving the system

Let $\mathcal{B}_n$ be the Boolean lattice with n generators, with partial order $\preceq$, where $a \preceq b$ if $a \vee b = b$. The symbol $\vee$ is conveniently overloaded for both the lattice's least upper bound and the bitwise-or operation on the binary representation of integers from 0 to $2^{2^n}-1$. For v variables, we are concerned with $\mathcal{B}_{2^v}$.

Theorem 2 *The number of formulæ of length n equivalent to f_i (generated as a disjunction of atomic function symbols $f_0, f_1, \ldots$) is given by:*

$$\mathbf{OR}(v)_i^n = \sum_{s \preceq i} \left[(-1)^{2^v - d(s)} \left(\sum_{t \preceq s} f_t^1 \right)^n \right] \tag{1}$$

For example, $\mathbf{OR}(1)_3^n = (f_0+f_1+f_2+f_3)^n$ - $(f_0+f_1)^n-$ $(f_0+f_2)^n+$ f_0^n. If the base cases are as above, then $\mathbf{OR}(1)_3^n = 2^n - 2$. Note that to reduce the symbolic complexity of Equation 1, the base case f_m^1 is implied.

Proof: By induction on n. If $n=1$, then by inclusion-exclusion, all that is left is f_i^1. From the linear system in (1), $\mathbf{OR}(v)^n = \mathbf{OR}(v) \cdot \mathbf{OR}(v)^{n-1}$. Any entry in $\mathbf{OR}(v)^n$ is the dot product of constants, the coefficients from $\mathbf{OR}(v)$ being the complement of those in $\mathbf{OR}(v)^{n-1}$. Since these are of opposite sign, what is left over for a term is

$$\left(\sum_{t \preceq s} f_t^1\right)\left(\sum_{t \preceq s} f_t^1\right)^{n-1} = \left(\sum_{t \preceq s} f_t^1\right)^{n}$$

$\square$

This is a closed-form formula for the number of propositional formulæ, as a sequence of clauses in disjunction, regardless of the makeup of the clauses. Were some other method used to count clauses, or even some other syntax used to construct items in place of clauses, the above theorem would still allow us to enumerate the disjunctions representing any particular boolean function.

There is a difficulty in purely computational terms: the summations range over all the items in the boolean algebra, namely all 2^{2^v} functions. So we seek simplifications.

3.3 Simplifying the answer

The number of satisfiable c-clause k-CNF formulæ over v variables is the total number of formulæ less the unsatisfiable ones, namely $(2v)^{kc} - \mathbf{AND}_0^k$, where the base case $\mathbf{AND}_i^1 = \mathbf{OR}(v)_{2^{2^v}-i}$. As a corollary to Theorem 2, we first determine $\mathbf{OR}(v)_i^1$ for all i, and then $\mathbf{OR}(v)_i^k$, and then use that to compute $\mathbf{AND}(v)_0^c$.

Under different models of random selection, there are different ways of specifying $\mathbf{OR}(v)_i^1$. Theorem 2 works for a disjunction (or dually a conjunction) as a sequence; it does not matter how the clauses f are computed.

We can use the model of selection for how a clause is generated to simplify. The traditional method for generating random k-CNF formulæ is first by the literals in the clause, as a set of variables without replacement, then by sign for each variable. This gives $2^k\binom{v}{k}$ possible clauses, and these clauses all produce distinct functions.

Since there is only one of each of these functions, we can apply this to Theorem 2, and simplify. Under this model, in Equation 2, there are at most $2^k\binom{v}{k}$ possible distinct terms. In computing the number of unsatisfiable formulæ, $\left(2^k\binom{v}{k}\right)^c$ is the largest term, and is removed by subtracting to get the satisfiable formulæ. The next largest term is $\left((2^k-1)\binom{v}{k}\right)^c$. This is an upper bound on the number of satisfiable formulæ. The ratio of satisfiable to total is then

$$\begin{aligned}
\frac{\left(2^v(2^k-1)\binom{v}{k}\right)^c}{\left(2^k\binom{v}{k}\right)^c} &= \left(2^v\frac{2^k-1}{2^k}\right)^c \\
&= \left(2\left(\frac{2^k-1}{2^k}\right)^r\right)^v
\end{aligned}$$

when $c = vr$, a linear ratio of v. The limit of this, as v goes to infinity, is a step function where

$$2\left(\frac{2^k-1}{2^k}\right)^r = 1$$

or, for $k = 3$, at $r = \frac{\log 2}{\log 8/7} \approx 5.191$, a well-known upper bound for the 3-SAT threshold.

4 Conclusion

Using elementary arguments, we were able to count the number of k-CNF formulæ for every boolean function, and specifically the number of satisfiable formulæ for a given number of variables, clauses, and literals per clause. The method of creating a system of equations to count the functions can be applied to any formula syntax using any set of operators. In this case, CNF syntax and the dual boolean operators simplify the analysis considerably.

How does the general result compare with other counting methods? It is terribly impractical for $v \geq 4$; might other methods be better? The generate-and-test method would involve generating all formulæ and then running a satisfiability decider on each. Though the number of formulæ, $(2v)^{kc}$, is far below doubly exponential, the test time is (conjectured to be—of course) worst case exponential in v. Though $(2v)^{kc}2^v$ is still much better than 2^{2^v} in terms of calculation, the general formula quantifies the situation much better and is amenable to symbolic manipulation.

Simply counting combinatorial objects is interesting enough in its own right, but the method and the result can be used for other things. The general method can be applied to other families of formulæ, especially those with a simple syntax and those with other logical operators. For non-linear grammars, that is non-regular grammars, the results will involve the Catalan numbers and its analogues.

Once the result is refined to give the exact function for the number of satisfiable k-CNF formulæ, asymptotic analysis will give better bounds on the SAT/UNSAT threshold phenomenon.

References

[CFGG02] Brigitte Chauvin, Philippe Flajolet, Daniéle Gardy, and Bernhard Gittenberger. And/or trees revisited. *Submitted to Combinatorics, Probability, and Computing*, page 27, December 2002.

[DL89] Nachum Dershowitz and Naomi Lindenstrauss. Average time analyses related to logic programming. In *Logic programming (Lisbon, 1989)*, pages 369–381. MIT Press, Cambridge, MA, 1989.

[Fri99] Ehud Friedgut. Sharp thresholds of graph properties, and the k-sat problem. *J. Amer. Math. Soc.*, 12(4):1017–1054, 1999. With an appendix by Jean Bourgain.

[JSV00] Svante Janson, Yannis C. Stamatiou, and Malvina Vamvakari. Bounding the unsatisfiability threshold of random 3-SAT. *Random Structures Algorithms*, 17(2):103–116, 2000.

[KS94] Scott Kirkpatrick and Bart Selman. Critical behavior in the satisfiability of random Boolean expressions. *Science*, 264(5163):1297–1301, 1994.