Enumerating Satisfiable Propositional Formulæ

Nachum Dershowitz* Mitchell A. Harris†

May 15, 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, University of Illinois at Urbana-Champaign, 1304 W. Springfield Avenue, Urbana, Illinois 61801, USA. Email: maharri@cs.uiuc.edu

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.

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. It is expected that further analysis of the closed form enumeration found here will result in establishing an analytic solution to the satisfiability threshold problem ([JSV00]).

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 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. 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.

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 below. The binary function $\wedge$ has the obvious dual behavior.

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

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 the operator table above:

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

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.

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 1 *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.

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.

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.