The Computational Complexity of (XOR, AND)-Counting Problems

Andrzej Ehrenfeucht
Dept. of Computer Science
University of Colorado,
Boulder, Colorado 8030L
andrzej@boulder.colorado.edu

Marek Karpinski *
Dept. of Computer Science
University of Bonn, 5300 Bonn 1
and
International Computer Science Institute
Berkeley, California
marek@icsi.berkeley.edu

Abstract

We characterize the computational complexity of counting the exact number of satisfying assignments in (XOR, AND)-formulas in their RSE-representation (i.e., equivalently, polynomials in $GF[2][x_1, \ldots, x_n]$). This problem refrained for some time effords to find a polynomial time solution and the efforts to prove the problem to be $\#P$-complete. Both main results can be generalized to the arbitrary finite fields GF[q]. Because counting the number of solutions of polynomials over finite fields is generic for many other algebraic counting problems, the results of this paper settle a border line for the algebraic problems with a polynomial time counting algorithms and for problems which are $\#P$-complete. In [KL 89] the couting problem for arbitrary multivariate polynomials over GF[2] has been proved to have randomized polynomial time approximation algorithms.

*Supported in part by the Leibniz Center for Research in Computer Science, by DFG Grant KA 673/2-1, and by the SERC Grant GR-E 68297.

1 Introduction

Let us denote by kXOR the class of all formulas f of the form $f = \bigoplus a_A \wedge \bigwedge_{i\in A} x_i$, for a 0-1 vector $(a_A)_{A\subseteq\{1,\dots,n\}}$ such that $|A| \leq k$ (or equivalently, kXOR-formulas f are Galois polynomials $f \in GF[2][x_1,\dots,x_n]$of degree at most k). XOR $= \bigcup_k k$XOR. For a formula $f \in$ XOR with n variables, denote $\#f = |\{(x_1,\dots,x_n)|f(x_1,\dots,x_n) = 1\}|$. The counting problem for kXOR is the problem of computing $\#f$ for any given formula $f \in k$XOR.

In this paper we prove that the problem of exact counting the number of satisfying arguments of 3XOR-formulas (polynomials of degree 3 over GF[2]) is $\#P$-complete. We design also an $O(n^3)$-time algorithm for the 2XOR-counting problem.

2 Some Auxiliary Lemmas on Polynomials over GF[2]

Suppose $w_i \in GF[2][x_i,\dots,x_n]$, $\quad i = 1,\dots,m$, define a polynomial $u = \bigoplus_{i=1}^m w_i z_i$ for new variables $z_i \notin \{x_1,\dots,x_n\}$. Define by $\#s(\{w_i\})$ the number of solutions of the system $\{w_i = 0\}_{i=1,\dots,m}$. For a single polynomial u, $\#s(u)$ denotes the number of solutions of u, i.e. $\#s(u) = \#\{\bar{x}|u(\bar{x}) = 0\}$. With this notation we formulate the following

Lemma 1.

$$\#s(u) = \#s(\{w_i\})2^m + (2^n - \#s(\{w_i\}))2^{m-1}$$

Proof:
Suppose $x \in s(\{w_i\})$, then all the vectors $z \in \{0,1\}^m$ are solutions of $u = \bigoplus_{i=1}^m w_i z_i$. There are $\#s(\{w_i\})2^m$ of them. Suppose now that $x \notin s(\{w_i\})$. Denote by $K_x = \{i|w_i(x) \neq 0\}$ the set of all indices of polynomials w_i so that $w_i(x) \neq 0$.

Let us characterize the vectors $y \in \{0,1\}^m$ such that xy is a solution of u. y could be 0 or 1 everywhere besides the coordinates in K_x. On the coordinates of K_x, the number of 1's must add up to 0 (mod 2). There are therefore

$$2^{m-|K_x|}\sum_{r=0}^{|K_x|/2}\left(\begin{array}{c}|K_x|\\ 2r\end{array}\right) = 2^{m-|K_x|}2^{|K_x|-1} = 2^{m-1}$$

vectors y such that xy is a solution of u. We note that this number now is independent of the particular form of K_x. This gives for different $x \notin s(\{w_i\})$ different solutions of u, and results in $(2^n - \#s(\{w_i\})2^{m-1})$ additional solutions of u. $\square$

We derive some corollaries from Lemma 1.

Lemma 2. The system $\{w_i = 0\}_{i=1,\dots m}$ has a solution iff $\#s(u) > 2^{n+m-1}$.

($\#s(u) \geq 2^{n+m-1}$ always holds.)

Lemma 3.

$$\#s(\{w_i\}) = \frac{\#s(u) - 2^{n+m-1}}{2^{m-1}}$$

In the next section we shall make use of the Lemmas above.

3 3XOR–Counting and –Majority Problems are Hard to Compute

We state now our main *hardness* result.

Theorem 1. Given an arbitrary 3XOR formula $f \in GF[2][x_1, \ldots, x_n]$, the problem of computing $\#f$ is $\#P$-complete.

Proof:
Let us take a monotone 2DNF formula $f = c_1 \vee c_2 \vee \ldots \vee c_m$ where $c_i = (a_i \wedge b_i)$ and a_i, b_i are variables. The problem of computing $\#f$ for any given monotone 2DNF formula is $\#P$-complete (cf., e.g. [V 79]). We define the system w_i of polynomials by $w_i = a_i b_i, \quad i = 1, \ldots, m$ and construct the polynomial $u = \bigoplus_{i=1}^{m} w_i z_i$ as in section 2.

By Lemma 3

$$\#f = 2^n - \frac{\#s(u) - 2^{n+m-1}}{2^{m-1}}.$$

Therefore computing $\#f$ for monotone 2DNF formulas f is polynomial time reducible to computing $\#s(u)$ for 3XOR formulas.

We characterize also Majority and Solutions' Equilibrium Problems for 4XOR-formulas. (SAT for polynomials f is equivalent with checking whether $f \equiv 0$, trivially doable for explicitely given f.)

For the corresponding results for the $(\wedge, \vee, \neg)$-basis see [G 77].

Theorem 2. Given any 4XOR formula $f \in GF[2][x_1, \ldots, x_n]$, the problems of deciding whether $\#f > 2^{n-1}$ and $\#f = 2^{n-1}$ are both NP-hard.

Proof:
Let us take 3CNF formula $f = \bigwedge_{i=1}^{m}(a_i \vee b_i \vee c_i)$ over n variables $x_1, \ldots, x_n$ where a_i, b_i, c_i are literals (nonnegated and negated variables). We shall rewrite f into the system of m equations $\{w_i = (a_i \vee b_i \vee c_i) \oplus 1\}_{i=1,\ldots,m}$ in (XOR, AND) basis by writing

$$\neg x = 1 \oplus x$$

and

$$(a_i \vee b_i \vee c_i) = a_i \oplus b_i \oplus c_i \oplus a_i b_i \oplus a_i c_i \oplus b_i c_i \oplus a_i b_i c_i.$$

Let us construct a polynomial $u \in GF[2][x_1, \dots, x_n, x_{n+1}, \dots, x_{n+m}]$ as in Section 2. For $k = n+m$, the problem of deciding 3CNF SAT is polynomial time reducible to the problem of checking whether $\#s(u) > 2^{k-1}$ or $\#s(u) = 2^{k-1}$. $\square$

Remark: Using Valiant's result (cf. [GJ 79], p. 251) on systems of algebraic equations over GF[2], we can analogously prove that the Majority and Equilibrium Problems are NP-hard already for 3XOR-formulas.

4 2XOR–Counting Problem

We are going to design an algorithm to count the number $\#f$ for arbitrary $f \in$ 2XOR $(f \in F[2][x_1, \dots, x_n],\ f$ is polynomial of degree 2).

Theorem 3. Given arbitrary 2XOR-formula f, there exists an algorithm working in $O(n^3)$ time for computing $\#f$.

We shall call $f \in GF[2][x_1, \dots, x_n]$ *read-once* if every variable x_i in f appears in f at most once.

The proof of Theorem 3 will be based on the following sequence of results.

Lemma 4. Given arbitrary 2XOR-formula f, $f \in$GF[2]$[x_1, \dots, x_n]$, there exists a *read-once* 2XOR-formula $g \in$GF[2]$[y_0, \dots, y_m]$, $m \leq n$, a nonsingular $m \times n$ matrix $T = (t_{ij})$ and an m vector $C = (c_i)$ such that

$$g(\bigoplus_{j=1}^{n} t_{0j}x_j + c_0, \bigoplus_{j=1}^{n} t_{1j}x_j + c_1, \dots, \bigoplus_{j=1}^{n} t_{m-1,j}x_j + c_{m-1}) = f(x_1, \dots, x_n).$$

There exists an algorithm for computing matrix $T = (t_{ij})$ and vector $C = (c_1)$ for arbitrary 2XOR-formulas f working in $O(n^3)$ time. The form of g can be chosen to be

$$g = y_0 \oplus y_1y_2 \oplus y_3y_4 \oplus \dots \oplus y_{m-2}y_{m-1} \oplus z \qquad \text{or}$$

$$g = y_0y_1 \oplus y_2y_3 \oplus \dots \oplus y_{m-2}y_{m-1}q \oplus z$$

where $z \in \{0, 1\}$.

Proof:
We shall describe an algorithm for computing matrix $T = (t_{ij})$, vector $C = (c_i)$ and constant z. The algorithm will be by recursion on the set of variables $\mathrm{Var}(f) = \{x_1, \dots, x_n\}$.

Recursion Stage x_i:

Let $x := x_i$
Rewrite f as $f = x\alpha \oplus \beta$ where α is a linear form, and β is the rest of f.

Represent (recursively)

$$\beta = y_0 \oplus y_1 y_2 \oplus y_3 y_4 \oplus \ldots \oplus y_{k-2} y_{k-1} \oplus z \qquad \text{type I}$$

or

$$\beta = y_0 y_1 \oplus y_2 y_3 \oplus \ldots \oplus y_{m-2} y_{m-1} \oplus z \qquad \text{type II}$$

where $z \in \mathrm{GF}[2]$ and corresponding nonsingular $k \times (n-i)$ matrix T_β and vector C_β. Note that $k \leq n-i$.

Consider the following cases:

Case 1. $\alpha = 1$.

- β is of type I.
  Construct new variables
  $$\begin{aligned} y'_0 &:= y_0 \oplus x \\ y'_i &:= y_i \qquad i = 1, \ldots, k-1 \end{aligned}$$
- β is of type II.
  Construct new variables
  $$\begin{aligned} y'_0 &:= x \\ y'_{i+1} &:= y_i \qquad i = 0, \ldots, k-1 \end{aligned}$$

Case 2. α is linear *independent* of the variables of β (α cannot be expressed as a linear combination of the rows of matrix T_β). Note that in this case, $k < n-i$.

Construct new variables
$$\begin{aligned} y'_i &:= y_i \qquad i = 0, \ldots, k-1 \\ y'_k &:= x \\ y'_{k+1} &:= \alpha \end{aligned}$$

Case 3. α is linear dependent on the variables of β.

Let $\alpha = y_{i_1} \oplus \ldots \oplus y_{i_s} \oplus \begin{cases} 0 \\ 1 \end{cases}$

3.a. y_s and y_t in α form a term of β.

$$\ldots \oplus x y_s \oplus x y_t \oplus y_s y_t = \ldots (x \oplus y_s)(x \oplus y_t) \oplus \underbrace{x}_{\text{an 'extra' } x}$$

Construct new variables
$$\begin{aligned} y'_s &:= y_s \oplus x \\ y'_t &:= y_t \oplus x \end{aligned}$$

3.b. y_s is in α but its 'partner' y_t in a term of β is not in α.

$$\ldots \oplus x y_s \oplus y_s y_t = \ldots y_s (x \oplus y_t)$$

Construct new variables
$$\begin{aligned} y'_s &:= y_s \\ y'_t &:= y_t \oplus x \end{aligned}$$

3.c. β is of type I.

- α is independent of y_0 and the number of 'free' x is odd.
  Construct new variable
  $$y_0' \; := \; y_0 \oplus x$$
- α is dependent of y_0 and the number of 'free' x is odd.
  $$\ldots \oplus xy_0 \oplus y_0 \oplus x = \ldots (x \oplus 1)(y_0 \oplus 1) \oplus 1$$
  Construct new variables
  $$\begin{array}{rcll} z' & := & z \oplus 1 & \\ y_i' & := & y_{i+1} & i = 0, \ldots, k-2 \\ c_i' & := & c_{i+1} & i = 0, \ldots, k-2 \\ y_{k-1}' & := & y_0 & \\ c_{k-1}' & := & c_0 + 1 & \\ y_k' & := & x & \\ c_k' & := & 1 & \end{array}$$
  g is of type II.
- α is dependent of y_0 and the number of 'free' x is even.
  $$\ldots \oplus xy_0 \oplus y_0 = \ldots (x \oplus 1)y_0$$
  Construct new variables
  $$\begin{array}{rcll} z' & := & z & \\ y_i' & := & y_{i+1} & i = 0, \ldots, k-2 \\ c_i' & := & c_{i+1} & i = 0, \ldots, k-2 \\ y_{k-1}' & := & y_0 & \\ c_{k-1}' & := & c_0 & \\ y_k' & := & x & \\ c_k' & := & 1 & \end{array}$$
  g is of type II.

3.d. β is of type II and the number of 'free' x is odd.
Construct new variables
$$\begin{array}{rcll} y_0' & := & x & \\ y_{i+1}' & := & y_i & i = 0, \ldots, k-1 \end{array}$$
g is of type I.

It is not difficult to check that the algorithm produces the *substitution* matrix $T = [t_{ij}]$ as defined in Lemma 4.

The algorithm works in n recursive steps and each step runs in $O(n^2)$ time. □

We complete the proof of Theorem 3.

Lemma 5.
$$\#f = \#g 2^{n-m}$$

Proof: Obvious from linear algebra.

Finally, the direct counting arguments give us the following.

Lemma 6.

1. Given a 2XOR-formula $g \in GF[2][x_1, \ldots, x_n]$, $g = x_1x_2 \oplus x_3x_4 \oplus \ldots \oplus x_{n-2}x_{n-1} \oplus x_n$,
$$\#g = 2^{n-1}.$$
2. Given a 2XOR-formula $g \in GF[2][x_1, \ldots, x_n]$, $g = x_1x_2 \oplus x_3x_4 \oplus \ldots \oplus x_{n-1}x_n$
$$\#g = 2^{n-1} - 2^{\frac{n-2}{2}}.$$

$\square$ $\square$

5 Acknowledgements

We are thankful to Avi Wigderson, Dick Karp, Mike Luby and Thorsten Werther for the number of interesting conversations.

References

[AW 85] Ajtai, M. and Wigderson, A., *Deterministic Simulation of Probabilistic Constant Depth Circuits*, Proc. 26^{th} IEEE FOCS (1985), pp. 11 - 19

[G 77] Gill, J., *Computational Complexity of Probabilistic Turing Machines*, SIAM J. Comput. **6**, pp. 675 - 694

[GJ 79] Garey, M.R. and Johnson, D.S., *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman and Company, New York (1979)

[KL 83] Karp, R.M. and Luby, M., *Monte-Carlo Algorithms for Enumeration and Reliability Problems*, Proc. 24^{th} IEEE FOCS (1983), pp. 56-64

[KL 85] Karp, R.M. and Luby, M., *Monte-Carlo Algorithms for the Planar Multiterminal Network Reliability Problem*, J. of Complexity **1** (1985), pp. 45 - 64

[KL 89] Karpinski, M. and Luby, M., *Approximating the Number of Solutions of a GF[2]-Polynomial*, manuscript, 1989

[KLM 89] Karp, R.M., Luby, M. and Madras, N., *Monte-Carlo Approximation Algorithms for Enumeration Problems*, J. of Algorithms **10** (1989), pp. 429 - 448

[V 79] Valiant, L.G., *The Complexity of Enumeration and Reliability Problems*, SIAM J. Comput. **8**, pp. 410 - 421

[W 87] Wegener, I., *The Complexity of Boolean Functions*, John Wiley, New York, 1987