# Factoring Multilinear Boolean Polynomials

Michael Monagan
Simon Fraser University

Finite Fields and Applications
ACA 2025

# Multilinear Boolean Polynomials

Let $f \in F[x_1, x_2, \ldots, x_n]$ where $F$ is a field.
$f$ is multilinear if $\deg(f, x_i) = 1$ for $1 \leq i \leq n$.
$f$ is boolean if $F = GF(2)$.

**Problem:** Factor a multilinear boolean $f$ in $n$ variables with $t$ terms.

**Example**

$$f = x_1x_3x_4 + x_1 + x_2x_3x_4 + x_2 = (x_1 + x_2)(x_3x_4 + 1).$$

If we encode $f = 1011\ 1000\ 0111\ 0100$ then size$(f)$ is $nt$ bits.
Can we factor $f$ in $O(nt \log t)$ bit complexity?
Existing factorization algorithms for $F = GF(2)$ work over $GF(2^k)$ and are not linear.

# Properties of multilinear boolean polynomials

$$f = x_1x_3x_4 + x_1 + x_2x_3x_4 + x_2 = (x_1 + x_2)(x_3x_4 + 1).$$

Let $f, g$ be a multilinear boolean polynomial. Let $\#f$ denote the number of terms of $f$.

1. If $f = f_1f_2$ then $vars(f_1) \cap vars(f_2) = \phi$.
2. If $f = f_1f_2$ then $\#f = \#f_1\#f_2$. $(x^2 - 1) = (x - 1)(x + 1)$
3. The irreducible factors of $f$ over every field $F$ are the same!
4. If the terms of $f$ are sorted in lex order with $x_1 > x_2 > \cdots > x_n$ then the terms of $coeff(f, x_i, 1)$ and $coeff(f, x_i, 0)$ remain sorted. E.g. $coeff(f, x_3, 1) = x_1x_4 + x_2x_4$.
5. If $g|f$ then $\dfrac{f}{g} = Proj(f, vars(f) - vars(g)) = f(y = 1 | y \in vars(g))/\#g$.

   E.g. if $g = x_1 + x_2$ then $\dfrac{f}{g} = f(x_1 = 1, x_2 = 1)/2 = (\underbrace{x_3x_4 + 1 + x_3x_4 + 1}_{\text{not sorted}})/2 = x_3x_4 + 1.$

# Emelyanov and Ponomaryov's GCD algorithm from [1].

Algorithm GcdFactor
Input: multilinear boolean $f$ in $x_1, \ldots, x_n$.
Output: irreducible factors of $f$.

1. If $vars(f) = \phi$ **return**() else pick $x$ from $vars(f)$.
2. Set $a = coeff(f, x, 1)$ and $b = coeff(f, x, 0)$ so that $f = ax + b$.
3. Compute $g = \gcd(a, b)$ over $\mathbb{F}_2$ or $\mathbb{Q}$.
4. Call Algorithm GcdFactor to factor $g$ recursively; Let $g_1, \ldots, g_k$ be the factors of $g$.
5. Since $f/g$ is irreducible **return**$(f/g, g_1, \ldots, g_k)$.

How do we compute $f/g$? Use Proj$(f, vars(g))$.

How do we compute $\gcd(a, b)$?
Emelyanov and Ponomaryov use our Zippel GCD from [2] which does $O(nt^3)$ field operations.
Zippel's GCD needs a large field. $GF(p)$ is better than $GF(2^k)$.

# Emelyanov and Ponomaryov's FDE algorithm

Assume $f$ has no trivial factors, i.e. $x_i$ is not a factor.

Let $x \in vars(f)$ and $f = Ax + B$.

Let $c = \gcd(A, B)$ and $p = f/c$ so that $f = cp$.

Suppose we pick $y \in vars(f)$ with $y \neq x$. Is $y$ in $vars(c)$ or $vars(p)$ ?

**Theorem (Emelyanov and Ponomaryov)**

*Let $S = AB$ and $S_y = \partial S/\partial y$. If $S_y = 0$ then $y \in vars(c)$ else $y \in vars(p)$.*

This yields the following algorithm.

Algorithm FDexp (Emelyanov and Ponomaryov [1])
Input: $f$ be non-zero multilinear with no trivial factor of $x_i$.
Output: irreducible factors of $f$.

1 If $\deg f \leq 1$ return $f$.
2 Pick $x \in vars(f)$.
3 Let $f = Ax + B$ and set $S = AB$.
4 Initialize $\Sigma_c = \phi$ and $\Sigma_p = \{x\}$.
5 for $y \in vars(f) - \{x\}$ do
  5a Compute $S_y = \partial S / \partial y$.
  5b If $S_y = 0$ then $\Sigma_c = \Sigma_c \cup \{y\}$ else $\Sigma_p = \Sigma_p \cup \{y\}$.
  end for.
6 Set $p = Proj(f, \Sigma_p)$ and $c = Proj(f, \Sigma_c)$.
7 Let $c_1, \ldots, c_r$ be the factors of $c$ (computed recursively).
8 Output $p, c_1, \ldots, c_r$.

Problem: $S = AB$ has bit size $O(nt^2)$. This leads to and $O(n^2t^2)$ algorithm for factoring $f$.

Let $S = AB = (Cy + E)(Dy + F)$.
Then $S_y = \partial S/\partial y = AD + BC$.
Do not compute $AD$ and $BC$! Instead apply Schwartz-Zippel to test if $AD + BC = 0$.

**Theorem (The Schwartz-Zippel Lemma [3, 4])**

*Let $D$ be an integral domain and $S$ be a finite subset of $D$.*
*Let $f$ be non-zero in $D[x_1, x_2, \ldots, x_n]$.*
*If $\beta$ is chosen at random from $S^n$ then* $Prob[f(\beta) = 0] \leq \frac{\deg f}{|S|}$.

6. Compute $C = coeff(A, y, 1)$ and $D = coeff(B, y, 1)$.
7. Pick $\beta \in GF(2^k)$ at random for large $k$.
   If $A(\alpha)D(\alpha) + B(\alpha)C(\alpha) = 0$ then $\Sigma_c = \Sigma_c \cup \{y\}$ else $\Sigma_p = \Sigma_p \cup \{y\}$.

The new algorithm does $O(n^2 t)$ ring operations in $GF(2^k)$.
It's Monte Carlo: the error probability $< 2n \deg(f) 2^{-k}$.

# Implementation

C code is available at `www.cecm.sfu.ca/~mmonagan/code/facpoly`
If the input file foo has this in it

```
x1*x3*x4*x5+x2*x3*x4*x5+x1*x3+x2*x3
```

the Unix command

```
./facpoly 5 foo
```

produces this output

```
f1 := x2+x1;
f2 := x3;
f3 := x4*x5+1;
```

# Implementation

I use $k = 63$ so I can implement a multiplication in $GF(2^k)$ using bit operations on a 64 bit computer.

```
#define ULONG unsigned long
ULONG mul( ULONG a, ULONG b, ULONG B, ULONG m, ULONG M ) {
  // Compute a b mod m in F2[x]
  // M = 2^deg(m), B = 2^deg(b)
  if( a==0 || b==0 ) return 0;
  ULONG c = 0;
  while( B ) {
    c = c << 1; // c = 2 c
    if( c & M ) c = c ^ m; // c = c xor m
    if( b & B ) c = c ^ a; // c = c xor a
    B = B >> 1; // B = B/2
  }
  return c;
}
```

This does $\leq 378 = 63 \cdot 6$ bit operations per multiplication.
To reduce the probability of error for testing $AD + BC = 0$ I use two points $\alpha, \beta \in GF(2^{63})$.

# Benchmark

| $s$ | $t$ | GCD | FDexp($AD - BC$) | FDSZ($AD - BC$) | GCD4 |
|---|---|---|---|---|---|
| 10 | 100 | 2.027 | 4.068(99.9%) | 0.473(99.9%) | 0.008 |
| 25 | 40 | 1.716 | 3.903(99.9%) | 0.471(99.9%) | 0.007 |
| 50 | 20 | 2.328 | 3.561(99.9%) | 0.490(99.9%) | 0.006 |
| 100 | 10 | 2.344 | 3.781(99.8%) | 0.525(99.9%) | 0.006 |
| 10 | 1000 | 207.8 | 641.4(99.9%) | 4.658(99.9%) | 0.080 |
| 25 | 400 | 76.21 | 509.7(99.9%) | 4.855(99.9%) | 0.080 |
| 50 | 200 | 102.5 | 579.4(99.9%) | 4.917(99.9%) | 0.074 |
| 100 | 100 | 159.7 | 606.4(99.9%) | 4.942(99.9%) | 0.067 |
| 50 | 2000 | 6106.8 | NA | 47.19(99.9%) | 0.785 |
| 100 | 1000 | 8369.8 | NA | 48.16(99.9%) | 0.750 |
| 200 | 500 | 12581.6 | NA | 47.97(99.9%) | 0.745 |
| 316 | 316 | 13479.6 | NA | 47.80(99.9%) | 0.789 |
| 100 | 10000 | NA | NA | 476.4(99.9%) | 7.978 |
| 200 | 5000 | NA | NA | 469.7(99.9%) | 7.595 |
| 500 | 2000 | NA | NA | 473.9(99.9%) | 7.585 |
| 1000 | 1000 | NA | NA | 475.8(99.9%) | 7.888 |

Table: CPU time in seconds $n = 100$ variables with two factors with $s$ and $t$ terms.

Define $moncont(a) = \prod_{i=1}^{n} x_i$ if $x_i | a$
Example $moncont(x_1x_2 + x_2x_3) = x_2$.

Algorithm GCD4
Input: $a, b \in \mathbb{F}_2[x_1, \ldots, x_n]$ such that $a$ and $b$ are non-zero and multilinear.
Ouput: $g = \gcd(a, b)$.

1. $ma = moncont(a)$; $a = a/ma$;
   $mb = moncont(b)$; $b = b/mb$;
   $m_g = \gcd(m_a, m_b)$.
2. **if** $a = b$ **return** $m_g \times a$ **end if**
3. **if** $vars(a) \cap vars(b) = \phi$ **return** $m_g$ **end if**
4. Pick a variable $x$ from $vars(a) \cap vars(b)$
   Let $a = a_1x + a_0$ and $b = b_1x + b_0$.
   Here $moncont(a) = moncont(b) = 1 \implies a_0a_1b_0b_1 \neq 0$.
5. $c_a = GCD4(a_0, a_1)$; $p_a = a/c_a$. // $\#a_0 + \#a_1 = \#a$
   $c_b = GCD4(b_0, b_1)$; $p_b = b/c_b$. // $\#b_1 + \#b_1 = \#b$
   $c_g = GCD4(c_a, c_b)$.
6. **if** $p_a = p_b$ **return** $m_g c_g p_a$ **else return** $m_g c_g$ **end if**

# References


Pavel Emelyanov and Denis Ponomaryov. On a Polytime Factorization Algorithm for Multilinear Polynomials over $\mathbb{F}_2$. Proceedings of CASC 2018, LLNCS **11077**:164–176, Springer, 2018.

Jennifer de Kleine, Michael Monagan, and Allan Wittkopf. Algorithms for the Non-monic case of the Sparse Modular GCD Algorithm. *Proceedings of ISSAC '2005*, pp. 124–131, ACM, 2005.

Jacob Schwartz. Probabilistic algorithms for verification of polynomial identities. Proceedings of EUROSAM '79, LNCS **72**:200–215, Springer, 1979.

Richard Zippel. Probabilistic algorithms for sparse polynomials. Proceedings of EUROSAM '79, LNCS **72**:216–226, Springer, 1979.