A Multi-level Blocking Distinct Degree Factorization Algorithm [*]

Richard P. Brent
Australian National University
Canberra, Australia
Fq8@rpbrent.com

Paul Zimmermann
INRIA Lorraine/LORIA
Villers-lès-Nancy, France
Paul.Zimmermann@loria.fr

12 July 2007

[*]Copyright ©2007, R. P. Brent and P. Zimmermann.
Fq8t

Introduction

The problem of factoring a univariate polynomial $P(x)$ over a finite field F often arises in computational algebra. An important case is when F has small characteristic and $P(x)$ has high degree but is *sparse* (has only a small number of nonzero terms).

Since I only have 20 minutes, I will restrict attention to the case that $F = \mathrm{GF}(2)$ and $P(x)$ is a *trinomial*

$$P(x) = x^r + x^s + 1, \quad r > s > 0,$$

although the ideas apply more generally.

The aim is to give an algorithm with good *amortized complexity*, that is, one that works well *on average* where, since we are restricting attention to trinomials, we average over all trinomials of fixed degree r.

Distinct degree factorization

I will only consider *distinct degree factorization.* That is, if $P(x)$ has several factors of the same degree d, the algorithm will produce the product of these factors. The Cantor-Zassenhaus algorithm can be used to split this product into distinct factors. This is usually cheap because the product usually has either small degree or consists of just one factor.

Factor of smallest degree

To simplify the complexity analysis and speed up the algorithm in the common application of searching for irreducible polynomials, I only consider the time required to find *one* nontrivial factor (it will be a factor of smallest degree) or output "irreducible".

Certificates of reducibility

A nontrivial factor (preferably of smallest degree) gives a "reducibility certificate" that can quickly be checked.

Factorization over GF(2)

It is well-known that $x^{2^d} + x$ is the product of all irreducible polynomials of degree dividing d. For example,

$$x^{2^3} + x = x(x+1)(x^3+x+1)(x^3+x^2+1)\ .$$

Thus, a simple algorithm to find a factor of smallest degree of $P(x)$ is to compute $\mathrm{GCD}(x^{2^d} + x, P(x))$ for $d = 1, 2, \ldots$. The first time that the GCD is nontrivial, it contains a factor of minimal degree d. If the GCD has degree $> d$, it must be a product of factors of degree d.

If no factor has been found for $d \le r/2$, where $r = \deg(P(x))$, then $P(x)$ must be irreducible.

Application to trinomials

Some simplifications are possible when $P(x) = x^r + x^s + 1$ is a trinomial.

- We can skip the case $d = 1$ because a trinomial can not have a factor of degree 1.
- Since $x^r P(1/x) = x^r + x^{r-s} + 1$, we only need consider $s \le r/2$.
- By applying Swan's theorem, we can often show that the trinomial under consideration has an odd number of factors; in this case we only need check $d \le r/3$.

Note that x^{2^d} should not be computed explicitly; instead compute $x^{2^d} \bmod P(x)$ by repeated squaring. The complexity of squaring modulo a trinomial of degree r is only $O(r)$ bit-operations. Thus, most of the time is spent performing GCD computations.

Complexity of squaring

As well as performing GCD computations we need to perform multiplications of polynomials in GF(2)/$P(x)$, and the special case of squaring a polynomial, so let's first consider the bit-complexity of these operations.

Squaring means squaring of polynomials of degree $< r$ and reduction mod $P(x)$. Squaring over GF(2) can be performed in time $S(r) = \Theta(r) \ll M(r)$.

Where possible we use the memory-efficient squaring algorithm of Brent, Larvala and Zimmermann (2003), which (at least in our implementation) is about 2.2 times faster than the obvious squaring algorithm.

Complexity of multiplication

Multiplication of polynomials of degree r over GF(2) can be performed in time $M(r) = O(r \log r \log\log r)$. We have implemented an algorithm of Schönhage (1977) that achieves this bound. (Note: the algorithm uses a base-3 FFT and is *not* the better-known Schönhage-Strassen algorithm.)

We have also implemented classical, Karatsuba and Toom-Cook algorithms that have $M(r) = O(r^\alpha)$, $1 < \alpha \le 2$, since these algorithms are easier to implement and are faster for small r. The Toom-Cook algorithms TC3 and TC4 are based on recent ideas of Bodrato (2007).

For brevity we assume that r is large and Schönhage's algorithm is used. On a 64-bit machine the crossover versus TC4 occurs near $r = 108000$. (The crossover versus the $O(r^2)$ algorithm is much smaller.)

In the complexity estimates we assume that $M(r)$ is a sufficiently smooth and well-behaved function.

Complexity of GCD

For GCDs we use a sub-quadratic algorithm that runs in time $G(r) = O(M(r) \log r)$.

More precisely,

$$G(2r) = 2G(r) + O(M(r)) \ ,$$

so

$$M(r) = O(r^\alpha) \Rightarrow G(r) = O(M(r)) \ ,$$

but

$$M(r) = O(r \log r \log\log r) \Rightarrow G(r) = \Theta(M(r) \log r) \ .$$

In practice, for $r \approx 2.4 \times 10^7$ and our implementation on a 2.2 Ghz Opteron,

$$S(r) \approx 0.005 \ \text{seconds},$$

$$M(r) \approx 2 \ \text{seconds},$$

$$G(r) \approx 80 \ \text{seconds},$$

$$M(r)/S(r) \approx 400 \ ,$$

$$G(r)/M(r) \approx 40 \ .$$

Avoiding GCD computations

In the context of integer factorization, Pollard (1975) suggested a blocking strategy to avoid most GCD computations and thus reduce the amortized cost; von zur Gathen and Shoup (1992) applied the same idea to polynomial factorization.

The idea of blocking is to choose a parameter $\ell > 0$ and, instead of computing

$$\mathrm{GCD}(x^{2^d} + x, P(x)) \text{ for } d \in [d', d' + \ell) ,$$

compute

$$\mathrm{GCD}(p_\ell(x^{2^{d'}}, x), P(x)) ,$$

where the *interval polynomial* $p_\ell(X, x)$ is defined by

$$p_\ell(X, x) = \prod_{j=0}^{\ell-1} \left(X^{2^j} + x\right) .$$

In this way we replace ℓ GCDs by one GCD and $\ell - 1$ multiplications mod $P(x)$.

Backtracking

The drawback of blocking is that we may have to backtrack if $P(x)$ has more than one factor with degrees in $[d', d' + \ell)$, so ℓ should not be too large. The optimal strategy depends on the expected size distribution of factors and the ratio of times for GCDs and multiplications.

New idea - multi-level blocking

Our new idea is to use a finer level of blocking to replace most multiplications by squarings, which speeds up the computation in $\mathrm{GF}(2)[x]/P(x)$ of the interval polynomials $p_m(x^{2^d}, x)$, where

$$p_m(X, x) = \prod_{j=0}^{m-1} \left(X^{2^j} + x\right) = \sum_{j=0}^{m} x^{m-j} s_{j,m}(X) ,$$

$$s_{j,m}(X) = \sum_{0 \le k < 2^m,\, w(k) = j} X^k ,$$

and $w(k)$ denotes the Hamming weight of k.

Note that $s_{j,m}(X^2) = s_{j,m}(X)^2$ in $\mathrm{GF}(2)[x]/P(x)$. Thus, $p_m(x^{2^d}, x)$ can be computed with cost $m^2 S(r)$ if we already know $s_{j,m}(x^{2^{d-m}})$ for $0 < j \le m$.

In this way we replace m multiplications and m squarings by one multiplication and m^2 squarings. Choosing $m \approx \sqrt{M(r)/S(r)}$ (about 20 if $M(r)/S(r) \approx 400$), the speedup over single-level blocking is about $m/2 \approx 10$.

Fast initialization

The polynomials

$$s_{j,m}(x) = \sum_{0 \le k < 2^m,\, w(k) = j} x^k$$

satisfy a "Pascal triangle" recurrence relation

$$s_{j,m}(x) = s_{j,m-1}(x^2) + x s_{j-1,m-1}(x^2)$$

with boundary conditions

$$s_{j,m}(x) = 0 \text{ if } j > m ,$$

$$s_{0,m}(x) = 1 .$$

Thus, it is easy to compute $s_{j,m}(x) \bmod P(x)$ in time $O(m^2 r)$ even though the definition involves $O(2^m)$ terms.

Recapitulation

To summarize, we use two levels of blocking:

- The outer level replaces most GCDs by multiplications.
- The inner level replaces most multiplications by squarings.
- The blocking parameter $m \approx \sqrt{M(r)/S(r)}$ is used for the inner level of blocking.
- A different parameter $\ell = km$ is used for the outer level of blocking.

Example

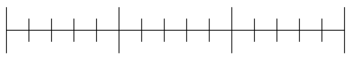

Figure 1: $\ell = 15$, $m = 5$

In the example, $S = 1/25$, $M = 1$, $G = 10$

No blocking: cost $15G + 15S = 150.6$

1-level blocking: $G + 14M + 15S = 24.6$

2-level blocking: $G + 2M + 75S = 15.0$

More realistically, suppose $\ell = 80$, $m = 20$, $S = 1/400$, $M = 1$, $G = 40$

No blocking: cost $80G + 80S = 3200.2$

1-level blocking: $G + 79M + 80S = 119.2$

2-level blocking: $G + 3M + 1600S = 47.0$

Sieving

A *small* factor is one with degree $d < \frac{1}{2}\log_2 r$, so $2^d < \sqrt{r}$. It would be inefficient to find small factors in the same way as large factors. Instead, let $d' = 2^d - 1$, $r' = r \bmod d'$, $s' = s \bmod d'$. Then

$$P(x) = x^r + x^s + 1 = x^{r'} + x^{s'} + 1 \bmod (x^{d'} - 1) ,$$

so we only need compute

$$\mathrm{GCD}(x^{r'} + x^{s'} + 1, x^{d'} - 1) .$$

The cost of finding small factors is negligible (both theoretically and in practice), so will be ignored.

In the definition, the fraction $\frac{1}{2}$ can be replaced by $1 - \varepsilon$.

Distribution of degrees of factors

In order to predict the expected behaviour of our algorithm, we need to know the expected distribution of degrees of irreducible factors. Our complexity estimates here are based on the assumption that trinomials of degree r behave like the set of all polynomials of the same degree, up to a constant factor:

Assumption 1 *Over all trinomials $x^r + x^s + 1$ of degree r over* GF(2)*, the probability π_d that a trinomial has no nontrivial factor of degree $\leq d$ is at most c/d, where c is a constant and $1 < d \leq r$.*

This assumption is plausible and in agreement with experiments, though not proven. Under the assumption, we use an amortized model to obtain the total complexity over all trinomials of degree r.

From Assumption 1, the probability that a trinomial does not have a small factor is $O(1/\log r)$.

Simpler approximation

Although not strictly correct, it is a good approximation to say that the probability $p_d = \pi_{d-1} - \pi_d$ that the smallest nontrivial factor of a randomly chosen trinomial has degree $d \geq 2$ is of order $1/d^2$, provided d is not too large.

I will use this approximation because it simplifies the amortized complexity analysis, but the same results can be obtained from Assumption 1 using summation by parts.

Table 1: Statistics for $r = 6972593$

d	$d\pi_d$	$d^2 p_d$
2	1.33	1.33
3	1.43	1.71
4	1.52	1.52
5	1.54	1.84
6	1.60	1.47
7	1.60	1.85
8	1.67	1.29
9	1.64	2.10
10	1.65	1.73
100	1.77	
1000	1.76	
10000	1.88	
100000	1.62	
226887	2.08	
$r-1$	2.00	

Outer level blocking strategy

The blocksize in the outer level of blocking is $\ell = km$. We take an increasing sequence

$$k = k_0 j \text{ for } j = 1, 2, 3, \ldots ,$$

where $k_0 m$ is of order $\log r$ (since small factors will have been found by sieving). This leads to a quadratic polynomial for the interval bounds.

There is nothing magic about a quadratic polynomial, but it is simple to implement and experiments show that it is reasonably close to optimal.

Using the data that we have obtained on the distribution of degrees of smallest factors of trinomials, and assuming that this distribution is insensitive to the degree r, we could obtain a strategy that is close to optimal. However, the choice $k_0 j$ with suitable k_0 is simple and not too far from optimal. The number of GCD and sqr/mul operations is usually within a factor of 1.5 of the minimum possible.

Expected cost of sqr/mul

Recall that the inner level of blocking replaces m multiplications by m^2 squarings and one multiplication, where $m \approx \sqrt{M(r)/S(r)}$ makes the cost of squarings about equal to the cost of multiplications.

For a smallest factor of degree d, the expected number of squarings is $m(d + O(\sqrt{d}))$. Averaging over all trinomials of degree r, the expected number is

$$O\left(m \sum_{d \leq r/2} \frac{d + O(\sqrt{d})}{d^2}\right) = O\left(m \log r\right) .$$

Thus, the expected cost of sqr/mul operations per trinomial is

$$\begin{aligned} & O\left(S(r) \log r \sqrt{M(r)/S(r)}\right) \\ = \; & O\left(\log r \sqrt{M(r)S(r)}\right) \\ = \; & O\left(r (\log r)^{3/2} (\log\log r)^{1/2}\right) . \end{aligned}$$

Expected cost of GCDs

Suppose that $P(x)$ has smallest factor of degree d. The number of GCDs required to find the factor, using our (quadratic polynomial) blocking strategy, is $O(\sqrt{d})$. By Assumption 1, the expected number of GCDs for a trinomial with no small factor is

$$1 + O\left(\sum_{(\lg r)/2 < d \le r/2} \frac{\sqrt{d}}{d^2}\right) = 1 + O\left(\frac{1}{\sqrt{\log r}}\right) .$$

Thus the expected cost of GCDs per trinomial is

$$O(G(r)/\log r) = O(M(r)) = O(r \log r \log\log r) .$$

This is asymptotically $\ll$ expected cost of sqr/mul operations

On the other hand, if $M(r) = O(r^\alpha)$ with $\alpha > 1$, the expected cost of GCDs is $O(r^\alpha/\log r)$ $\gg$ expected cost of sqr/mul $O(r^{(1+\alpha)/2}\log r)$.

In practice, for $r \approx 2.4 \times 10^7$, $\alpha \approx 1.2$ and GCDs take about 65% of the time versus 35% for sqr/mul.

Comparison with classical algorithms

For simplicity I will use the $\widetilde{O}$ notation which ignores log factors.

The "classical" algorithm, as implemented by BLZ (2003) and others, takes an expected time $\widetilde{O}(r^2)$ per trinomial, or $\widetilde{O}(r^3)$ to cover all trinomials of degree r.

The new algorithm takes expected time $\widetilde{O}(r)$ per trinomial, or $\widetilde{O}(r^2)$ to cover all trinomials of degree r.

In practice, the new algorithm is faster by a factor of about 160 for $r = 6972593$, and by a factor of about 560 for $r = 24036583$.

Primitive trinomials

We are particularly interested in trinomials $x^r + x^s + 1$ where r is a *Mersenne exponent*, i.e. $2^r - 1$ is prime, because any irreducible trinomial of degree r must be primitive.

The largest published primitive trinomial is

$$x^{6972593} + x^{3037958} + 1 \quad (\textit{Bibury})$$

found by Brent, Larvala and Zimmermann in 2002 using a classical algorithm.

In March–April 2007, we tested our new program by verifying the published results on primitive trinomials for Mersenne exponents $r \le 6972593$, and in the process produced certificates of reducibility (lists of smallest factors for each reducible trinomial). These are available from my website `http://wwwmaths.anu.edu.au/~brent/trinom.html` .

New world record

Since 25 April 2007 we have been running our new algorithm to search for primitive trinomials of degree $r = 24036583$. This is the next Mersenne exponent, apart from two that are trivial to exclude by Swan's theorem. It would take about 41 times as long as for $r = 6972593$ by the classical algorithm, but our new program is 560 times faster than the classical algorithm. Each trinomial takes on average about 16 seconds on a 2.2 Ghz Opteron.

We have now checked more than 75 percent of the trinomials of degree 24036583, using about 24 Opteron and Core 2 processors located at ANU and INRIA (France).

We have found two new primitive trinomials of (equal) record degree:

$$x^{24036583} + x^{8412642} + 1 \quad (\textit{Eugénie})$$

$$x^{24036583} + x^{8785528} + 1 \quad (\textit{Judy-anne})$$

Independent verification

Allan Steel kindly verified irreducibility of *Judy-anne* (which we found on 27 June) using Magma.

The verification took 247348 secs (2.86 days) on an 2.4GHz Intel Core 2 processor.

Postscript (9 July)

Allan Steel has also verified irreducibility of *Eugénie* (which we found on 4 July).

Conclusion

The new double-blocking strategy works and, combined with fast multiplication and GCD algorithms, has allowed us to find new primitive trinomials of record degree.

The same ideas work over finite fields GF(p) for small prime $p > 2$, and for factoring sparse polynomials $P(x)$ that are not necessarily trinomials: all we need is that the time for p-th powers (mod $P(x)$) is much less than the time for multiplication (mod $P(x)$).

References

[1] M. Bodrato, Towards Optimal Toom-Cook Multiplication for Univariate and Multivariate Polynomials in Characteristic 2 and 0, *Lecture Notes in Computer Science* **4547**, 119–136. Springer, 2007. `http://bodrato.it/papers/#WAIFI2007`

[2] W. Bosma, and J. Cannon, *Handbook of Magma Functions*, School of Mathematics and Statistics, University of Sydney, 1995. `http://magma.maths.usyd.edu.au/`

[3] R. P. Brent, S. Larvala and P. Zimmermann, A fast algorithm for testing reducibility of trinomials mod 2 and some new primitive trinomials of degree 3021377, *Math. Comp.* **72** (2003), 1443–1452. `http://wwwmaths.anu.edu.au/~brent/pub/pub199.html`

[4] R. P. Brent, S. Larvala and P. Zimmermann, A primitive trinomial of degree 6972593, *Math. Comp.* **74** (2005), 1001–1002, `http://wwwmaths.anu.edu.au/~brent/pub/pub224.html`

[5] D. G. Cantor and H. Zassenhaus, A new algorithm for factoring polynomials over finite fields, *Math. Comp.* **36** (1981), 587–592.

[6] J. von zur Gathen and J. Gerhard, Polynomial factorization over F_2, *Math. Comp.* **71** (2002), 1677–1698.

[7] J. von zur Gathen and V. Shoup, Computing Frobenius maps and factoring polynomials, *Computational Complexity* **2** (1992), 187–224. `http://www.shoup.net/papers/`

[8] T. Kumada, H. Leeb, Y. Kurita and M. Matsumoto, New primitive t-nomials ($t = 3, 5$) over GF(2) whose degree is a Mersenne exponent, *Math. Comp.* **69** (2000), 811–814. Corrigenda: *ibid* **71** (2002), 1337–1338.

[9] A.-E. Pellet, Sur la décomposition d'une fonction entière en facteurs irréductibles suivant un module premier p, *Comptes Rendus de l'Académie des Sciences Paris* **86** (1878), 1071–1072.

[10] J. M. Pollard. A Monte Carlo method for factorization, *BIT* **15** (1975), 331–334,

[11] A. Schönhage, Schnelle Multiplikation von Polynomen über Körpern der Charakteristik 2, *Acta Inf.* **7** (1977), 395–398.

[12] V. Shoup, NTL: A library for doing number theory, `http:www.shoup.net/ntl/`

[13] L. Stickelberger, Über eine neue Eigenschaft der Diskriminanten algebraischer Zahlkörper, *Verhandlungen des ersten Internationalen Mathematiker-Kongresses*, Zürich, 1897, 182–193.

[14] R. G. Swan, Factorization of polynomials over finite fields, *Pacific J. Math.* **12** (1962), 1099–1106.

[15] G. Woltman *et al*, GIMPS, The Great Internet Mersenne Prime Search, `http://www.mersenne.org/`