Three Ways to Test Irreducibility

Richard P. Brent
Australian National University

joint work with

Paul Zimmermann
INRIA, Nancy
France

7 April 2009

Outline

- Polynomials over finite fields
- Irreducibility criteria
- First algorithm — via repeated squaring
- Modular composition
- Second algorithm — via modular composition using matrix multiplication
- Third algorithm — via "fast" modular composition
- Comparison of the algorithms
- The "best" algorithm
- New primitive trinomials

Polynomials over finite fields

We consider univariate polynomials $T(x)$ over a finite field F. The algorithms apply, with minor changes, for any small positive characteristic, but since time is limited we assume that the characteristic is two, and $F = \mathbb{Z}/2\mathbb{Z} = \mathrm{GF}(2)$.

$T(x)$ is *irreducible* if it has no nontrivial factors. If $T(x)$ is irreducible of degree d, then [Gauss]

$$x^{2^d} = x \bmod T(x).$$

Thus $T(x)$ divides the polynomial $\mathcal{P}_d(x) = x^{2^d} - x$. In fact, $\mathcal{P}_d(x)$ is the product of all irreducible polynomials of degree m, where m runs over the divisors of d. Thus, the number of irreducible polynomials of degree d is

$$\frac{2^d}{d} + O\left(\frac{2^{d/2}}{d}\right).$$

Since there are 2^d polynomials of degree d, the probability that a randomly selected polynomial is irreducible is $\sim 1/d \to 0$ as $d \to +\infty$. In this sense, *almost all* polynomials are reducible.

Irreducibility criteria

Since irreducible polynomials are "rare" but useful in many applications, we are interested in algorithms for testing irreducibility.

From the previous slide, $T(x)$ of degree d is irreducible iff

$$x^{2^d} = x \bmod T(x)$$

and, for all prime divisors m of d, we have

$$\mathrm{GCD}\left(x^{2^{d/m}} - x, T(x)\right) = 1\ .$$

The second condition is required to rule out the possibility that $T(x)$ is a product of irreducible factors of some degree(s) $k = d/m$, $m|d$.

Since the second condition does not significantly change anything, let us assume that d is prime (as it is in all our examples). Then $T(x)$ is irreducible iff

$$x^{2^d} = x \bmod T(x).$$

One more assumption

All the algorithms involve computations mod $T(x)$, that is, in the ring GF(2)$[x]/T(x)$.

In the complexity analysis we assume that $T(x)$ is *sparse*, that is, the number of nonzero coefficients is small. Thus, reduction of a polynomial mod $T(x)$ can be done in linear time. (The comparison of the algorithms might be different without this assumption.)

In applications $T(x)$ is often a *trinomial*

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

First algorithm — repeated squaring

Our first and simplest algorithm for testing irreducibility is just *repeated squaring*:

$P(x) \leftarrow x$;
for $j \leftarrow 1$ to d do
  $P(x) \leftarrow P(x)^2 \bmod T(x)$;
if $P(x) = x$ then
  return **irreducible**
else
  return **reducible**.

The operation $P(x) \leftarrow P(x)^2 \bmod T(x)$ can be performed in time $O(d)$. The constant factor is small. We recommend the fast squaring algorithm of Brent, Larvala and Zimmermann (2003). This saves both operations and memory references, and is about 2.2 times faster than the obvious squaring algorithm (as implemented in most otherwise-good software packages).

Since the test involves d squarings, the overall time is $O(d^2)$.

Polynomial multiplication

Before describing other algorithms for irreducibility testing, we digress to discuss polynomial multiplication, matrix multiplication, and modular composition.

To multiply two polynomials $A(x)$ and $B(x)$ of degree (at most) d, the "classical" algorithm takes time $O(d^2)$. There are faster algorithms, e.g. Karatsuba, Toom-Cook, and FFT-based algorithms.

For polynomials over GF(2), the asymptotically fastest known algorithm is due to Schönhage. (The Schönhage-Strassen algorithm does not work in characteristic 2, and it is not clear whether Fürer's ideas are useful here.)

Schönhage's algorithm runs in time

$$M(d) = O(d \log d \log\log d) \, .$$

In practice, for $d \approx 32\,000\,000$, a multiplication takes about 480 times as long as a squaring.

Matrix multiplication

Let ω be the exponent of matrix multiplication, so we can multiply $n \times n$ matrices in time $O(n^{\omega+\varepsilon})$ for any $\varepsilon > 0$. The best result is Coppersmith and Winograd's $\omega < 2.376$, though in practice we would use the classical ($\omega = 3$) or Strassen ($\omega = \log_2 7 \approx 2.807$) algorithm.

Since we are working over GF(2), our matrices have single-bit entries. This means that the classical algorithm can be implemented very efficiently using full-word operations (32 or 64 bits at a time). Nevertheless, Strassen's algorithm is faster if n is larger than about 1000.

Good in practice is the "Four Russians" algorithm [Arlazarov, Dinic, Kronod & Faradzev, 1970]. It computes $n \times n$ Boolean matrix multiplication in time $O(n^3/\log n)$.

We can use the Four Russians' algorithm up to some threshold, say $n = 1024$, and Strassen's recursion for larger n, combining the advantages of both.

Modular composition

The *modular composition* problem is: given polynomials $A(x)$, $B(x)$, $T(x)$, compute

$$C(x) = A(B(x)) \bmod T(x).$$

If $\max(\deg(A), \deg(B)) < d = \deg(T)$, then we could compute $A(B(x))$, a polynomial of degree at most $(d-1)^2$, and reduce it modulo $T(x)$. However, this wastes both time and space.

Better is to compute

$$C(x) = \sum_{j \le \deg(A)} a_j (B(x))^j \bmod T(x)$$

by Horner's rule, reducing mod $T(x)$ as we go, in time $O(dM(d))$ and space $O(d)$. Using Schönhage's algorithm for polynomial multiplication, we can compute $C(x)$ in time $O(d^2 \log d \log\log d)$.

Faster modular composition

Using an algorithm of Brent & Kung (1978), based on an idea of Paterson and Stockmeyer, we can reduce the modular composition problem to a problem of matrix multiplication. If the degrees of the polynomials are at most d, and $m = \lceil d^{1/2} \rceil$, then we have to perform m multiplications of $m \times m$ matrices. The matrices are over the same field as the polynomials (that is, GF(2) here).

The Brent-Kung modular composition algorithm takes time

$$O(d^{(\omega+1)/2}) + O(d^{1/2} M(d)),$$

where the first term is for the matrix multiplications and the second term is the time for computing the relevant matrices.

Assuming Strassen's matrix multiplication, the first term is $O(d^{1.904})$ and the second term is $O(d^{1.5} \log d \log\log d)$. Thus, the second term is asymptotically negligible (but maybe not in practice).

Using modular composition

Recall that our problem is to compute $x^{2^d} \bmod T(x)$. Repeated squaring is not the only way to do this.

Let $A_k(x) = x^{2^k} \bmod T(x)$. Then a modular composition algorithm can be used to compute $A_k(A_m(x)) \bmod T(x)$. Since

$$A_k(A_m(x)) = \left(x^{2^m}\right)^{2^k} \bmod T(x) = A_{m+k}(x),$$

we can compute $x^{2^d} \bmod T(x)$ with about $\log_2(d)$ modular compositions (instead of d squarings).

For example, if $d = 17$, we have (with all computations in $\mathrm{GF}(2)[x]/T(x)$):

$A_1(x) = x^2$, (trivial)
$A_2(x) = A_1(A_1(x)) = x^4$, ($\equiv$ 1 squaring)
$A_4(x) = A_2(A_2(x)) = x^{16}$, ($\equiv$ 2 squarings)
$A_8(x) = A_4(A_4(x)) = x^{256}$, ($\equiv$ 4 squarings)
$A_{16}(x) = A_8(A_8(x)) = x^{2^{16}}$, ($\equiv$ 8 squarings)
$A_{17}(x) = A_{16}(x)^2 = x^{2^{17}}$, (1 squaring)

using only 4 modular composition steps.

Second algorithm

To summarise, we can compute $A_d(x) = x^{2^d} \bmod T(x)$ by the following recursive algorithm that uses the binary representation of d (*not* that of 2^d):

```
if d = 0 then
  return x
else if d even then
  {U(x) ← A_{d/2}(x);
  return U(U(x)) mod T(x)}
else
  return A_{d-1}(x)^2 mod T(x).
```

The algorithm takes about $\log_2(d)$ modular compositions. Hence, if Strassen's algorithm is used in the Brent-Kung modular composition algorithm, we can test irreducibility in time $O(d^{1.904} \log d)$.

Third algorithm

Recently, Kedlaya and Umans (2008) proposed an asymptotically fast modular composition algorithm that runs in time $O_\varepsilon(d^{1+\varepsilon})$ for any $\varepsilon > 0$.

The algorithm is complicated, involving iterated reductions to multipoint multivariate polynomial evaluation, multidimensional FFTs, and the Chinese remainder theorem.
See the papers on Umans's web site
`www.cs.caltech.edu/~umans/research.htm`

Using the Kedlaya-Umans fast modular composition instead of the Brent-Kung reduction to matrix multiplication, we can test irreducibility in time $O_\varepsilon(d^{1+\varepsilon})$.

Warning: the "$O_\varepsilon(\cdots)$" notation indicates that the implicit constant depends on ε. In this case, it is a rather large and rapidly increasing function of $1/\varepsilon$.

Comparison of the algorithms

> So the last shall be first,
> and the first last
>
> Matthew 20:16

The theoretical time bounds predict that the third algorithm should be the fastest, and the first algorithm the slowest. However, this is only for *sufficiently large* degrees d.

In practice, for d up to at least 4.3×10^7, the situation is reversed! The first algorithm is the fastest, and the third algorithm is the slowest.

A drawback of the first (squaring) algorithm is that it is hard to speed up on a parallel machine. The other algorithms are much easier to parallelise. However, this is not a major consideration if we are working on many trinomials, as we can let different processors work on different trinomials in parallel.

Example, $d = 32\,582\,657$

Following are actual or estimated times on a 2.2 Ghz AMD Opteron 275 for $d = 32\,582\,657$ (a Mersenne exponent).

1. Squaring (actual): 64 hours
2. Brent-Kung (estimates):
   - classical: 265 hours (19% mm)
   - Strassen: 254 hours (15% mm)
   - Four Russians: 239 hours (10% mm) (plus Strassen for $n > 1024$)
3. Kedlaya-Umans (estimate): $> 10^{10}$ years

The Brent-Kung algorithm would be the fastest if the matrix multiplication was dominant; unfortunately the $O(d^{1/2}M(d))$ overhead term is dominant.

Since the overhead scales roughly as $d^{1.5}$, we estimate that the Brent-Kung algorithm would be faster than the squaring algorithm for $d > 7 \times 10^8$ (approximately).

Note on Kedlaya-Umans

Éric Schost writes:

> The Kedlaya-Umans algorithm reduces modular composition to the multipoint evaluation of a multivariate polynomial, assuming the base field is large enough.
>
> The input of the evaluation is over F_p; the algorithm works over $\mathbb{Z}$ and reduces mod p in the end. The evaluation over $\mathbb{Z}$ is done by CRT modulo a bunch of smaller primes, and so on. At the end-point of the recursion, we do a naive evaluation on all of F_{p^m}, where p is the current prime and m the number of variables. So the cost here is $\geq p^m$.
>
> [Now he considers choices of m; all give $p^m \geq 1.36 \times 10^{27}$.]

Our estimate of $> 10^{10}$ years is based on a time of 1 nsec per evaluation (very optimistic).

The "best" algorithm

Comparing the second algorithm with the first, observe that the modular compositions do not all save equal numbers of squarings. In fact the *last* modular composition saves $\lfloor d/2 \rfloor$ squarings, the *second-last* saves $\lfloor d/4 \rfloor$ squarings, etc.

Each modular composition has the same cost. Thus, if we can use only one modular composition, *it should be the one that saves the most squarings*.

If we use $\lfloor d/2 \rfloor$ squarings to compute $x^{2^{\lfloor d/2 \rfloor}} \bmod T(x)$, then use one modular composition (and one further squaring, if d is odd), we can compute $x^{2^d} \bmod T(x)$ faster than with any of the algorithms considered so far, provided d exceeds a certain threshold.

In the example, the time would be reduced from 64 hours to 44 hours, a saving of 31%.

Doing two modular compositions would reduce the time to 40 hours, a saving of 37%.

Computational results

In 2007-8 Paul Zimmermann and I conducted a search for irreducible trinomials $x^d + x^s + 1$ whose degree d is a (known) Mersenne exponent. Since $2^d - 1$ is prime, *irreducible* implies *primitive*. The previous record degree of a primitive trinomial was $d = 6972593$.

d	s
24036583	8412642, 8785528
25964951	880890, 4627670, 4830131, 6383880
30402457	2162059
32582657	5110722, 5552421, 7545455

Table 1: Ten new primitive trinomials $x^d + x^s + 1$ of degree a Mersenne exponent, for $s \le d/2$.

We used the first algorithm to test irreducibility of the most difficult cases. Most of the time was spent discarding the vast majority of trinomials that have a small factor, using a new factoring algorithm with good average-case behaviour (the topic of another talk).

Recent results

Since Sept 2008 we have been searching for primitive trinomials of degree 43112609 (the largest known Mersenne exponent).

Dan Bernstein and Tanja Lange have joined in the search and contributed CPU cycles.

So far we completed about 98% of the search and found four new primitive trinomials $x^{43112609} + x^s + 1$:

$$s = 3569337,\ 4463337,\ 17212521,\ 21078848$$

Testing irreducibility took about 119 hours per trinomial on a 2.2 Ghz AMD Opteron, using our first algorithm. The "best" algorithm would take about 69 hours (saving 42%).

Most of the time (about 22 processor-years) was spent sieving out reducible trinomials at an average rate of about 32 sec per trinomial ($\times 43112609/2$ trinomials).

Acknowledgement

Thanks to Éric Schost for his comments on the work of Kedlaya and Umans; to Dan Bernstein and Tanja Lange for contributing computer time to the search for $d = 43112609$; to Alan Steel for independently verifying our new primitive trinomials using Magma; and to Victor Shoup for his package NTL, which was used to debug our software.

References

[1] V. L. Arlazarov, E. A. Dinic, M. A. Kronod & I. A. Faradzev, On economical construction of the transitive closure of an oriented graph, *Soviet Math. Dokl.* **11** (1975), 1209–1210.

[2] W. Bosma & 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, P. Gaudry, E. Thomé & P. Zimmermann, Faster multiplication in $GF(2)[x]$, *Proc. ANTS VIII 2008*, *Lecture Notes in Computer Science* **5011**, 153–166. `http://wwwmaths.anu.edu.au/~brent/pub/pub232.html`

[4] R. P. Brent & H. T. Kung, Fast algorithms for manipulating formal power series, *J. ACM* **25** (1978), 581–595. `.../pub045.html`

[5] R. P. Brent, S. Larvala & 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. `.../pub199.html`

[6] R. P. Brent & P. Zimmermann, A multi-level blocking distinct-degree factorization algorithm, *Finite Fields and Applications: Contemporary Mathematics* **461** (2008), 47–58. `.../pub230.html`

[7] R. P. Brent & P. Zimmermann, Ten new primitive binary trinomials, *Math. Comp.* **78** (2009), 1197–1199. `.../pub233.html`

[8] P. Bürgisser, M. Clausen & M. A. Shokrollahi, *Algebraic Complexity Theory*, volume 315 of *Grundlehren der mathematischen Wissenschaften*, Springer-Verlag, 1997.

[9] D. Coppersmith & W. Winograd, Matrix multiplication via arithmetic progressions, *J. Symb. Comput.* **9** (1980), 251–280.

[10] M. Fürer, Faster integer multiplication, *Proc. 48th STOC Conference*, 2007, 57–66.

[11] J. von zur Gathen & J. Gerhard, *Modern Computer Algebra*, Cambridge Univ. Press, 1999.

[12] K. Kedlaya & C. Umans, Fast modular composition in any characteristic, *Proc. FOCS 2008*, 146–155. `http://www.cs.caltech.edu/~umans/research.htm`

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

[14] É. Schost, *Fast irreducibility test*, personal communication, 4 June 2008.

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

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