MATHEMATICS OF COMPUTATION
Volume 72, Number 243, Pages 1443–1452
S 0025-5718(02)01478-3
Article electronically published on December 18, 2002

A FAST ALGORITHM FOR TESTING REDUCIBILITY OF TRINOMIALS MOD 2 AND SOME NEW PRIMITIVE TRINOMIALS OF DEGREE 3021377

RICHARD P. BRENT, SAMULI LARVALA, AND PAUL ZIMMERMANN

Abstract. The standard algorithm for testing reducibility of a trinomial of prime degree r over GF(2) requires $2r + O(1)$ bits of memory. We describe a new algorithm which requires only $3r/2+O(1)$ bits of memory and significantly fewer memory references and bit-operations than the standard algorithm.

If $2^r - 1$ is a Mersenne prime, then an irreducible trinomial of degree r is necessarily primitive. We give primitive trinomials for the Mersenne exponents $r = 756839$, 859433, and 3021377. The results for $r = 859433$ extend and correct some computations of Kumada *et al.* The two results for $r = 3021377$ are primitive trinomials of the highest known degree.

1. Introduction

Throughout this paper all polynomials are assumed to be in $Z_2[x]$. A polynomial $P(x)$ is *reducible* if it has nontrivial factors; otherwise it is *irreducible*. A polynomial $P(x)$ of degree $r > 1$ is *primitive* if $P(x)$ is irreducible and $x^j \neq 1 \bmod P(x)$ for $0 < j < 2^r - 1$. If $P(x)$ is primitive, then x is a generator for the multiplicative group of the field $Z_2[x]/(P(x))$, giving a useful representation of GF(2^r). See Lidl and Niederreiter [16] or Menezes *et al.* [18] for background information.

There is an interest in discovering primitive polynomials of high degree r because of their connection with fast, high-quality pseudorandom number generators of period at least $2^r - 1$: see [4] and the references given there. In such applications it is desirable to use primitive polynomials with a small number of nonzero terms. In particular, we are interested in *trinomials* of the form $x^r + x^s + 1$, where $r > s > 0$.

If $P(x)$ is irreducible and $\deg(P) = r$, then the order of x in $Z_2[x]/(P(x))$ is a divisor of $2^r - 1$. To test if $P(x)$ is primitive, we must test if the order of x is exactly $2^r - 1$. To do this efficiently it appears that we need to know the complete prime factorization of $2^r - 1$. At the time of writing these factorizations are known for $r < 673$ and certain larger r, see [6].

We say that r is a *Mersenne exponent* if $2^r - 1$ is prime. In this case the factorization of $2^r - 1$ is trivial and an irreducible polynomial of degree r is necessarily

Received by the editor July 9, 2001.

2000 *Mathematics Subject Classification.* Primary 11B83, 11Y16; Secondary 11-04, 11K35, 11N35, 11R09, 11T06, 11Y55, 12-04, 65Y10, 68Q25.

Key words and phrases. Irreducible polynomials, irreducible trinomials, primitive polynomials, primitive trinomials, Mersenne exponents, Mersenne numbers, random number generators.

primitive. Large Mersenne exponents are known [8], so there is a possibility of finding primitive trinomials of high degree if we have an efficient algorithm for testing reducibility.

Zierler [26] gave all primitive trinomials of Mersenne exponent $r \leq 11213$. Kurita and Matsumoto [14] extended the search to $r \leq 86243$, and Heringa *et al.* [10] to $r \leq 216091$. Kumada *et al.* [12] conducted an exhaustive search for $r = 859433$ and found one primitive trinomial.

In this paper we describe a new algorithm for testing reducibility of trinomials of odd degree, and give some results obtained by applying the algorithm to the Mersenne exponents $r \leq 3021377$. In particular, we have verified the published results for $r \leq 216091$, found three new primitive trinomials for $r = 756839$, found a primitive trinomial for $r = 859433$ which was missed by Kumada *et al.* [12], and found two new primitive trinomials for $r = 3021377$. The search for Mersenne exponents $r \leq 3021377$ is now complete.

Sieving is discussed in §2. In §3 we describe the standard algorithm for testing reducibility, and in §4 we describe our new algorithm. Some performance figures are given in §5. The computational results are summarised in §6 and Tables 3–4.

The reader may wish to refer to the preliminary report [5] for details which are omitted here.

2. Sieving

The following theorem characterises the irreducible polynomials of given degree. The proof is well-known, see for example [16].

Theorem 1. *Let $\Phi_{d,1}, \Phi_{d,2}, \ldots$ be the irreducible polynomials of degree d in $Z_2[x]$. Then, for $n \geq 1$,*

$$\prod_{d|n} \prod_{j} \Phi_{d,j}(x) = x^{2^n} + x \, .$$

Testing a polynomial $P(x)$ for irreducibility is analogous to testing a number N for primality. We can save time by first checking if $P(x)$ is divisible by an irreducible polynomial of low degree, in the same way that we can save time by checking if N is divisible by a small prime.

From Theorem 1, we can check if $P(x)$ is divisible by some irreducible polynomial $\Phi_{d,j}(x)$ of degree $d|n$ by computing $\mathrm{GCD}(P(x), x^{2^n} + x)$. By analogy with the process of sieving out small integer factors, this process will be called *sieving*, although the sieve is performed by a GCD computation. We are interested in the case that $P(x) = x^r + x^s + 1$ is a trinomial, $r > s > 0$.

Consider the computation of $G = \mathrm{GCD}(x^r + x^s + 1, x^{2^n} + x)$. If $k = 2^n - 1$, then $G = \mathrm{GCD}(x^r + x^s + 1, x^k + 1)$. In practice, we distinguish two cases:

(1) If $r \geq k$, we can use the fact that $G = \mathrm{GCD}(x^{r'} + x^{s'} + 1, x^k + 1)$, where $r' = r \bmod k$, $s' = s \bmod k$. Thus, for small k the computation of G is trivial.
(2) If $k > r$, the standard Euclidean algorithm would take time $O(k^2)$ and space $\Omega(k)$. We save time and space by first computing $x^{2^n} \bmod P(x)$ by squaring and reducing n times. Then we apply the Euclidean algorithm to compute $G = \mathrm{GCD}((x^{2^n} \bmod P(x)) + x, P(x))$. The overall time is $O(r(n + r))$ and space $\Omega(r)$. In practice the GCD computation runs much

faster than the worst-case bound, because $x^{2^n} \bmod P(x)$ is sparse if $2^n/r$ is not too large.

Sieving is performed with $n = 2, 3, 4, \ldots$ until one of the following holds:

(1) We find a nontrivial GCD, in which case $P(x)$ is reducible.
(2) The estimated time $T_s(n)$ which would be required to perform another sieving step satisfies $nT_s(n) \geq T_f(r)$, where $T_f(r)$ is the (estimated) time required for the full reducibility test of §4.
(3) $n > r/2$, in which case $P(x)$ is irreducible (in practice, it is unlikely that we sieve this far).

For $r = 3021377$, in most cases we sieve for $n \leq 24$, and sieving takes about 8% of the overall time while eliminating about 93% of trinomials from consideration. The remaining 7% of trinomials require the full reducibility test, which takes about 92% of the total computing time.

In §§3–4 we assume that a trinomial being tested for reducibility has already survived the sieving phase.

3. The standard algorithm

The standard algorithm for testing reducibility of a polynomial $P(x)$ uses the following theorem, which is an easy consequence of Theorem 1.

Theorem 2. *Let $P(x) \in Z_2[x]$, $\deg(P) = r > 1$. Then $P(x)$ is irreducible iff*

$$x^{2^r} = x \bmod P(x)$$

and, for all d, $1 \leq d < r$, if $d|r$ then

$$\mathrm{GCD}(x^{2^d} + x, P(x)) = 1 \ .$$

The second condition in Theorem 2 is trivial if the degree r is prime, which is the case in our applications (see §6). Hence, from now on we avoid complications by assuming that r is prime, although the algorithms described here and in §4 can easily be extended to handle the more general case. When r is prime and $P(x)$ is a trinomial, Theorem 2 reduces to

Corollary 1. *If $r > s > 0$, where r is prime, then $P(x) = x^r + x^s + 1 \in Z_2[x]$ is irreducible iff*

$$x^{2^r} = x \bmod P(x) \ .$$

The obvious implementation of Corollary 1 has a loop, executed r times, consisting of a *squaring* step $A(x) \leftarrow A(x)^2$ and a *reduction* step $A(x) \leftarrow A(x) \bmod P(x)$. Each of these steps can be implemented in $\Theta(r)$ bit-operations and $\Theta(r)$ space.

To consider an implementation in more detail, we assume that the polynomial $A(x) = a_0 + a_1x + \cdots + a_dx^d$ is represented as a bit-string $a_0a_1 \ldots a_d$. Because we are working in $Z_2[x]$, the cross terms in $A(x)^2$ vanish and we have simply

$$A(x)^2 = a_0 + a_1x^2 + \cdots + a_dx^{2d} \ ,$$

which is represented as a bit-string (with optional zero padding on the right):

$$a_0 0 a_1 0 a_2 0 \ldots a_{d-1} 0 a_d \ .$$

In the following, for clarity we describe algorithms in terms of bit-operations, but an efficient implementation will use word-operations so that several bit-operations can be performed with one machine instruction.

3.1. **Squaring.** On a byte-addressable machine, the squaring operation can be performed eight bits at a time using a table lookup. We precompute a table of 256 16-bit integers representing the "squares" of the 256 possible 8-bit sequences (where the integers encode the coefficients of polynomials in $Z_2[x]$). Thus, squaring a polynomial of degree $r-1$ can be done with $\lceil r/8 \rceil$ table lookups.

We have found that, on some machines, the table lookup method is not the fastest, even if the table size is optimised to give the best results. An alternative on a machine with wordlength $w = 2^k$ bits is to perform squaring by a sequence of $\lceil 2r/w \rceil \log_2(w/2)$ iterations of the logical operations "shift right", "$\vee$" and "$\wedge$". For details see [5].

3.2. **Reduction mod** $P(x)$**.** Reduction of $A(x) = a_0 + a_1x + \cdots + a_dx^d \bmod P(x)$, where $P(x) = x^r + x^s + 1$, is performed by a sequence of "exclusive or" operations (written "$\oplus$"). While $d = \deg(A) \geq r$, we set $A(x) \leftarrow A(x) - (x^d + x^{d+s-r} + x^{d-r})$, since $x^d + x^{d+s-r} + x^{d-r} = 0 \bmod P(x)$. In terms of bit-operations:

```
for $d \leftarrow \deg(A)$ downto $r$ do
    begin
    $a_{d-r} \leftarrow a_{d-r} \oplus a_d$;
    $a_{d+s-r} \leftarrow a_{d+s-r} \oplus a_d$;
    $a_d \leftarrow 0$;  {This can be implicit}
    end.
```

Although complete descriptions are not always given in the original papers, previous computations [9, 10, 12, 14, 20, 21, 24, 26] involving irreducible or primitive trinomials seem to have used some variant of sieving combined with squaring and reduction, much as described above. This is why we call it the *standard algorithm.* In §4 we describe an improvement which, although mathematically trivial, gives a significant reduction in computing time.

4. The new algorithm

The standard algorithm is inefficient because many of the bit-operations are performed on bits which are necessarily zero. Our new algorithm avoids this. We assume that both r and s are odd. This is not a serious restriction. We already assumed that r is prime. If s is even, we simply replace s by $r - s$, because the reciprocal trinomial $x^r + x^{r-s} + 1$ is reducible iff $x^r + x^s + 1$ is reducible.

Before giving the algorithm in the general case, we illustrate with the example $r = 7$, $s = 3$.

We initialise $A(x) \leftarrow x$, i.e., $a_0 \ldots a_6 \leftarrow 0100000$. The "squaring" operation is implicit: we keep the bit-vector 0100000 and regard this as representing

$$a_0a_2a_4a_6a_8a_{10}a_{12}$$

(the odd-numbered coefficients $a_1, a_3, \ldots$ in the square are necessarily zero, so need not be computed). We now reduce mod $P(x) = 1 + x^3 + x^7$. Observe that $x^{12} = x^5 + x^8 \bmod P(x)$, so we replace a_8 by $a_8 \oplus a_{12}$. We should also replace a_5 by $a_5 \oplus a_{12}$, but a_5 is currently zero, so we can simply regard the rightmost bit as representing a_5 rather than a_{12}. Thus, after the first step of the reduction we have a bit-vector representing

$$a_0a_2a_4a_6\underline{a_8}a_{10}a_5 \ ,$$

where the only bits which could have changed, because they depend on the results of "$\oplus$" operations, are underlined.

Proceeding in a similar fashion, we observe that $x^{10} = x^3 + x^6 \bmod P(x)$, but $a_3 = 0$, so we replace a_6 by $a_6 \oplus a_{10}$ and implicitly regard the second bit from the right as representing a_3 rather than a_{10}. Thus, after the reduction we have a bit-vector representing

$$a_0a_2a_4\underline{a_6}a_8a_3a_5 \ .$$

One more step of reduction gives a bit-vector representing

$$a_0a_2\underline{a_4}a_6a_1a_3a_5 \ .$$

Observe that this bit-vector contains the coefficients of $A(x)^2 \bmod P(x)$, but in a shuffled order. We need to apply an *interleave* permutation to get back to the natural order

$$a_0a_1a_2a_3a_4a_5a_6 \ .$$

Interleaving is closely related to the "squaring" operation described in §3.1. In fact, if we square $a_0a_2a_4a_6$ giving $a_00a_20a_40a_6$, square and rightshift $a_1a_3a_50$ giving $0a_10a_30a_50$, and apply a bitwise "$\vee$" operation, we obtain $a_0a_1a_2a_3a_4a_5a_6$. Thus, interleaving can be implemented by squaring and a few additional operations. Two squarings are necessary, but the bit-vectors are only half as long as in §3.1, so the work involved is almost the same.

We now describe the new algorithm in terms of bit-operations. To avoid confusion, we denote the working bit-array by $b_0b_1\cdots b_{r-1}$. This bit-array is used to represent the coefficients $a_0a_1\cdots a_{r-1}$ of the polynomial $A(x)$, but not necessarily in the natural order. In a program, only the b-array is required.

Let $\alpha = (r-1)/2$ and $\delta = (r-s)/2$. Since r and s are odd, α and δ are integers. Initially we set $b_1 \leftarrow 1$ and the other $b_j \leftarrow 0$ to represent $A(x) = x$. The algorithm involves a sequence of r steps of (implicit) squaring and reduction followed by interleaving.

4.1. **Implicit squaring and reduction.**

In terms of bit-operations, each squaring and reduction step is implemented by:

```
for $j \leftarrow r-1$ downto $\alpha+1$ do
    $b_{j-\delta} \leftarrow b_{j-\delta} \oplus b_j$.
```

Note that there are only $r/2 + O(1)$ "$\oplus$" bit-operations in the loop, which is a 75% reduction over the $2r + O(1)$ for the reduction step of the standard algorithm.

4.2. **Interleaving.**

A straightforward implementation of the interleaving step requires another bit-array (say $c_0c_1\cdots c_{r-1}$) for the result. For example:

```
$c_0 \leftarrow b_0$;
for $j \leftarrow 1$ to $\alpha$ do
    begin
    $c_{2j-1} \leftarrow b_{j+\alpha}$;
    $c_{2j} \leftarrow b_j$;
    end.
```

We call this a "forward interleave" because the loop index j increases (there is an analogous "backward interleave" where the loop index j decreases). We avoid

TABLE 1. Time to test reducibility, $c = \text{time(nsec)}/r^2$, $r = 3021377$.

processor	algorithm	compiler/assembler	cache size	c
300 Mhz Pentium II	standard	C code (gcc)	512KB	7.86
”	”	assembler	”	6.31
”	new	C code (gcc)	”	3.54
”	”	assembler	”	1.64
500 Mhz Pentium III	”	”	”	0.77
833 Mhz Pentium III	”	”	256KB	1.66
300 Mhz Ultrasparc 10	”	C code (gcc)	large	2.90
300 Mhz SGI R12000	”	C code (cc)	”	1.16
667 Mhz Alpha	”	C code (gcc)	”	0.60

copying the array c back to b by alternately using the array b and the array c (or by interchanging pointers appropriately).

The arrays b and c can overlap, in fact b_j can occupy the same memory as $c_{j+\alpha}$ $(j = 0, 1, \dots)$, if we alternate forward and backward interleaves. This reduces the storage requirement from $2r + O(1)$ bits to $3r/2 + O(1)$ bits. For details see [5, §4.3].

Partially overlapping the arrays b and c in this manner can improve performance dramatically on machines with memory hierarchies and cache sizes of less than $2r$ bits, because the working set is reduced in size by 25%. It has little effect on machines with much larger caches.

It is possible to perform interleaving using only $r + O(1)$ bits, by splitting the required permutation into a product of cycles. However, it seems difficult to implement such an algorithm efficiently using word-operations.

To summarise, the new algorithm has 75% fewer "$\oplus$" operations than the standard algorithm. Perhaps more significant is the number of memory references, which is reduced by 56%, from $8r/w + O(1)$ to $3.5r/w + O(1)$ loads/stores on a machine with w-bit words. Also, the working set size is reduced by 25%, so memory references are more likely to be in the cache. In practice the improvement provided by the new algorithm is at least a factor of two (see Table 1).

5. Performance

We expect the running time of our program (excluding sieving) to be $T = 10^{-9}cr^2$ seconds for a full reducibility test, where c is machine-dependent and approximately constant. In practice, because of cache effects, c is not independent of r. In Table 1 we give c for $r = 3021377$ on various machines. For IBM PCs (Intel Pentium II and Pentium III) we give the size of the L2 cache (here and below 1KB = 1024 bytes). If the cache size is given as "large" this means that it is significantly larger than the working set size ($3r/2$ bits). Since $3r/2$ bits is 553KB, the program performs much better on PCs with a 512KB cache than a 256KB cache. The programs run on PCs had inner loops written in assembler, unless otherwise noted. The NASM [23] assembler routines use MMX instructions, which operate on 64-bit registers [11]. The speedup over pure C code is approximately a factor of two. For other machines the possibilities for improving performance by using assembler were minimal, so the program was written purely in C (the times quoted are for the best compiler options,

Table 2. Time to test reducibility on 300 Mhz Pentium II.

r	time T (sec)	$c = 10^9T/r^2$
19937	0.42	1.06
110503	14.4	1.18
859433	1027	1.39
3021377	15010	1.64
6972593	199300	4.10

discovered by experiment, using 64-bit integers for SGI and Alpha processors). In Table 1:

(1) We compare the standard and new algorithms (both C and assembler implementations) on the same processor.
(2) We compare the new algorithm on different Pentium processors. Note the dramatic influence of the cache size on performance – an 833 Mhz Pentium III with 256KB cache runs slower than a 300 Mhz Pentium II with a 512KB cache.
(3) Finally, we compare the C implementation on some other processors.

In Table 2 we show the time for a full reducibility test with our new algorithm and various r on a machine (300 Mhz Pentium II) with 512KB L2 cache. The times given in the table do not include sieving, but this is relatively insignificant (e.g. sieving to $n = 24$ for $r = 3021377$ takes about 1200 seconds, or 7.4% of the total time).

6. Computational results

In Table 3 we give a table of primitive trinomials x^r+x^s+1, where r is a Mersenne exponent. We assume that $0 < 2s \leq r$ (the reciprocal trinomial $x^r + x^{r-s} + 1$ is not listed). To save space we have omitted the entries for $r \leq 11213$, which are given by Zierler [26], (see also [5, Table 5] and [18, Table 4.9]). Entries with $r = \pm 3 \bmod 8$ are unlikely, since $s = 2$ (or $r - 2$) is then the only possibility, by Swan's theorem.[1] In particular, there are no primitive trinomials[2] whose degree is the Mersenne exponent r = 21701, 86243, 216091, 1257787, 1398629, 2976221 or 13466917.

The computations for $r < 3021377$ have been checked by running at least two different programs on different machines. During this checking process, which confirmed the published results for $r < 756839$, the entry with $r = 859433$, $s = 170340$ was found. This was a surprise, because Kumada *et al.* [12] claimed to have searched the whole range for $r = 859433$ without finding this entry. In fact Kumada *et al.* missed the entry because of a bug in their sieving routine [13, 17].

[1]Swan's Theorem 1 is a rediscovery of a result of Stickelberger (1897); see Swan [22, p. 1099]. An elementary proof is given by Berlekamp [1, p. 159]. For a generalisation, see Blake [2]. We only need Swan's Corollary 5.

[2]Following a suggestion of Blake *et al.* [3], in all the cases of Mersenne exponent $r = \pm 3 \bmod 8$ with $5 < r \leq 1257787$, we have found trinomials (of degree slightly greater than r) which have a primitive polynomial factor of exact degree r. The cases $r < 500$ were considered in [3, Table 4]. In many applications such trinomials are as useful as primitive trinomials of degree r. Details will appear elsewhere.

TABLE 3. Primitive trinomials with degree a Mersenne exponent.

r	s	Notes
19937	881, 7083, 9842	Kurita and Matsumoto [14]
23209	1530, 6619, 9739	Kurita and Matsumoto [14]
44497	8575, 21034	Kurita and Matsumoto [14]
110503	25230, 53719	Heringa *et al.* [10]
132049	7000, 33912, 41469, 52549, 54454	Heringa *et al.* [10]
756839	215747, 267428, 279695	Brent *et al.* [5]
859433	170340, 288477	See text, §6
3021377	361604, 1010202	Brent *et al.* [5]

The three entries for $r = 756839$ are new (Kumada *et al.* did not search for this r), as are the two entries for $r = 3021377$. The search for $r = 3021377$ is complete, but has not been verified by an independent computation, so there is a possibility that a primitive trinomial has been missed due to a machine or programming error.[3]

Since the average time for a single reducibility test is $O(r^2)$, the time required to test all trinomials of degree r is $O(r^3)$. Thus, ignoring the variability of c and the effect of different sieving cutoffs,[4] we expect a search for $r = 3021377$ to take about 43 times as long as that for 859433, and a search for $r = 6972593$ to take about 12 times as long as that for $r = 3021377$.

By a theorem of Ore [19], if $P(x) = x^r + x^s + 1$ is primitive, then the trinomial $P^{[2]}(x) = x^{2^r-1} + x^{2^s-1} + 1$ of degree $2^r - 1$ is irreducible.[5] Thus, from the entries in Table 3 we can obtain irreducible trinomials of Mersenne prime degree as high as $2^{3021377} - 1$.

There is a large gap between some of the Mersenne exponents r for which primitive trinomials exist, *e.g.*, none exist in the interval $132049 < r < 756839$. In Table 4 we give some irreducible trinomials to fill this gap. As usual, we only list $s \le r/2$. The exponents r were chosen to be close to the arithmetic progression $10^5, 2 \times 10^5, 3 \times 10^5, \ldots$ with the constraints that r is prime, $2^r - 1$ is composite, and no prime factors of $2^r - 1$ are known. Such factors are certainly larger than 2^{56}: see [8]. The trinomials listed in Table 4 are extremely likely to be primitive,[6] but we are unable to prove primitivity without knowing the factorizations of $2^r - 1$.

There is an interest in finding irreducible trinomials $x^r + x^s + 1$ with $s \le 2$. Heuristic arguments suggest that the number with $r \le n$ is $\Theta(\log n)$, and that the number with prime $r \le n$ is $\Theta(\log \log n)$. Zierler [27] searched for irreducible trinomials $x^r + x + 1$ with $r \le 30000$, and Fredricksen and Wisniewski [7] searched for

[3]The search for $r = 3021377$ was completed in April 2001. In February 2001 we commenced a search with the next known Mersenne exponent $r = 6972593$; by February 2002 the search was 26% complete. For information on current status and results, see `http://www.comlab.ox.ac.uk/oucl/work/richard.brent/trinom.html` .

[4]If the sieving cutoff is $\Theta(\log r)$, then it is plausible that the overall time is $\Theta(r^3/\log r)$.

[5]Ore's theorem was rediscovered by Gleason and Marsh, and generalised by Zierler – see Golomb [9, §5.4], Lidl [16, Theorem 3.63], and Zierler [25].

[6]There are $(2^r - 2)/r$ irreducible *polynomials* of degree r, and of these $\phi(2^r - 1)/r$ are primitive, so the probability ρ that a randomly chosen irreducible polynomial of degree r is also primitive is $\phi(2^r - 1)/(2^r - 2)$; this is close to 1 as $2^r - 1$ has no small prime factors. In Table 4, $r \le 1000121$ and the prime factors of $2^r - 1$ are at least 2^{56}, so $\rho \ge (1 - 2^{-56})^{r/56} \ge 1 - 10^{-12}$.

Table 4. Some irreducible trinomials with prime degree r.

r	s	Search status
100151	4764, 15503	complete
200033	10175, 55224, 95397, 96236, 97575, 98763	"
300073	—	"
300151	49950, 87430	"
400033	17865, 103623	"
500231	4862, 10101, 203207, 205310	"
600071	111503	"
700057	24829, 121384	"
800057	92487, 140565, 161777, 192416, 249828	"
900217	82555, 437251	65% complete
1000121	39528, 144815, 154157	45% complete

irreducible trinomials $x^r + x^2 + 1$ with $r \leq 60000$. We have searched for $s \leq 2$, odd prime $r < 2.0 \times 10^6$. By Swan's theorem, we have $s = 1$ if $r = \pm 1 \bmod 8$, and $s = 2$ if $r = \pm 3 \bmod 8$. In addition to the known odd prime values $r = 3, 5, 7, 11, 29, 127$ (see for example [18, Table 4.6]), we found just one new example: $r = 80141$, $s = 2$. One further (large) example, $r = 2^{127} - 1$, is a consequence of Ore's theorem [19] and is discussed by Lenstra and Schoof [15, Lemma 4.1].

Acknowledgments. We thank the users of several workstations at INRIA Lorraine and Oxford University Computing Laboratory for making their idle time available. The Oxford Supercomputing Centre provided time to run the first author's programs on *Oscar*, an SGI Origin 2000, and *Tosca*, a PC cluster. Barry Mead and Mark Rodenkirch contributed some CPU cycles. Robert Hedges supplied a copy of [2], and Shuhong Gao kindly provided a preprint of his paper [3]. Mike Yoder drew our attention to [7], and verified our new results with his independently-written Ada program. Finally, we thank an anonymous referee for his comments on an earlier version of this paper.

References

[1] E. R. Berlekamp, *Algebraic Coding Theory*, McGraw-Hill, New York, 1968. MR **38:**6873

[2] I. F. Blake, S. Gao and R. J. Lambert, Constructive problems for irreducible polynomials over finite fields, in *Information Theory and Applications*, *Lecture Notes in Computer Science* **793**, Springer-Verlag, Berlin, 1994, 1–23. MR **95c:**94007

[3] I. F. Blake, S. Gao and R. J. Lambert, Construction and distribution problems for irreducible trinomials over finite fields, *Applications of Finite Fields*, Oxford Univ. Press, New York, 1996, pp. 19–32. MR **98a:**11170

[4] R. P. Brent, Random number generation and simulation on vector and parallel computers (extended abstract), *Proc. Fourth Euro-Par Conference*, *Lecture Notes in Computer Science* **1470**, Springer-Verlag, Berlin, 1998, 1–20. `http://www.comlab.ox.ac.uk/oucl/work/richard.brent/pub/pub185.html`

[5] R. P. Brent, S. Larvala and P. Zimmermann, *A Fast Algorithm for Testing Irreducibility of Trinomials* mod 2 (preliminary report), Report PRG-TR-13-00, Oxford University Computing Laboratory, 30 December 2000. See `http://www.comlab.ox.ac.uk/oucl/work/richard.brent/pub/pub199.html`

[6] J. Brillhart, D. H. Lehmer, J. L. Selfridge, B. Tuckerman, and S. S. Wagstaff, Jr., *Factorizations of* $b^n \pm 1$, $b = 2, 3, 5, 6, 7, 10, 11, 12$ *up to high powers*, 2nd ed., Amer. Math. Soc., Providence, RI, 1988. Updates available from `http://www.cerias.purdue.edu/homes/ssw/cun/`. MR **90d:**11009

[7] H. Fredricksen and R. Wisniewski, "On trinomials $x^n + x^2 + 1$ and $x^{8l\pm 3} + x^k + 1$ irreducible over GF(2)", *Inform. and Control* **50** (1981), 58–63. MR **84i:**12013

[8] GIMPS, The Great Internet Mersenne Prime Search, `http://www.mersenne.org/`

[9] S. W. Golomb, *Shift register sequences,* Holden-Day, San Francisco, 1967. MR **39:**3906 Revised edition, Aegean Park Press, 1982, ISBN 0-89412-048-4

[10] J. R. Heringa, H. W. J. Blöte and A. Compagner. New primitive trinomials of Mersenne-exponent degrees for random-number generation, *International J. of Modern Physics C* **3** (1992), 561–564. MR **94a:**11118

[11] Intel Corporation, *MMX Technology Programmer's Reference Manual.* Available from `http://developer.intel.com`.

[12] 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. MR **2000i:**11183

[13] T. Kumada, Y. Kurita and M. Matsumoto, Corrigenda to "New primitive t-nomials ($t = 3$, 5) over GF(2) whose degree is a Mersenne exponent, and some new primitive pentanomials", Math. Comp. **71** (2002), 1337–1338.

[14] Y. Kurita and M. Matsumoto, Primitive t-nomials ($t = 3, 5$) over GF(2) whose degree is a Mersenne exponent ≤ 44497, *Math. Comp.* **56** (1991), 817–821. MR **91h:**11138

[15] H. W. Lenstra and R. J. Schoof, Primitive normal bases for finite fields, *Math. Comp.* **48** (1987), 217–231. MR **88c:**11076

[16] R. Lidl and H. Niederreiter, *Introduction to Finite Fields and their Applications*, Cambridge Univ. Press, Cambridge, second edition, 1994. MR **95f:**11098

[17] M. Matsumoto, Private communication by email, 17 July 2000.

[18] A. J. Menezes, P. C. van Oorschot and S. A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, New York, 1997. `http://cacr.math.uwaterloo.ca/hac/` . MR **99g:**94015

[19] O. Ore, "Contributions to the theory of finite fields", *Trans. Amer. Math. Soc.* **36** (1934), 243–274.

[20] E. R. Rodemich and H. Rumsey, Jr., Primitive trinomials of high degree, *Math. Comp.* **22** (1968), 863–865. MR **39:**177

[21] W. Stahnke, Primitive binary polynomials, *Math. Comp.* **27** (1973), 977–980. MR **48:**6064

[22] R. G. Swan, Factorization of polynomials over finite fields, *Pacific J. Math.* **12** (1962), 1099–1106. MR **26:**2432

[23] S. Tatham and J. Hall, *NASM v0.98, the Netwide Assembler*, available from `http://www.web-sites.co.uk/nasm/docs/` .

[24] E. J. Watson, Primitive polynomials (mod 2), *Math. Comp.* **16** (1962), 368–369. MR **26:**5764

[25] N. Zierler, "On the theorem of Gleason and Marsh", *Proc. Amer. Math. Soc.* **9** (1958), 236–237. MR **20:**851

[26] N. Zierler, Primitive trinomials whose degree is a Mersenne exponent, *Inform. and Control* **15** (1969), 67–69. MR **39:**5522

[27] N. Zierler, On $x^n + x + 1$ over GF(2), *Inform. and Control* **16** (1970), 502–505. MR **42:**5955

Oxford University Computing Laboratory, Wolfson Building, Parks Road, Oxford, OX1 3QD, England
E-mail address: `Richard.Brent@comlab.ox.ac.uk`

Helsinki University of Technology, Espoo, Finland
E-mail address: `slarvala@cc.hut.fi`

LORIA/INRIA Lorraine, 615 rue du jardin botanique, BP 101, F-54602 Villers-lès-Nancy, France
E-mail address: `Paul.Zimmermann@loria.fr`