New Algorithms for Generating Conway Polynomials over Finite Fields

Lenwood S. Heath[*] Nicholas A. Loehr[†]

July 7, 1998

Abstract

Arithmetic in a finite field of prime characteristic p normally employs an irreducible polynomial in $\mathbb{Z}_p[X]$. A particular class of irreducible polynomials, generally known as Conway polynomials, provides a means for representing several finite fields of characteristic p in a compatible manner. Conway polynomials are used in computational algebra systems such as GAP and Magma to represent finite fields. The generation of the Conway polynomial for a particular finite field has previously been done by an often expensive brute force search. As a consequence, only a few Conway polynomials have been generated. We present two new algorithms for generating Conway polynomials that avoid the brute force search. We have implemented one of these algorithms in Magma and present numerous new Conway polynomials that implementation generated.

1 Introduction

Every finite field $\mathbb{F}$ is characterized by two parameters — its prime characteristic p and its dimension n over $\mathbb{Z}_p$, the integers modulo p. The field $\mathbb{F}$ has p^n elements and is isomorphic to any other field having p^n elements. The field $\mathbb{F}$ is often denoted $\mathrm{GF}(p, n)$ or just $\mathrm{GF}(p^n)$,

[*]Department of Computer Science, Virginia Tech, Blacksburg, VA 24061-0106, `heath@cs.vt.edu`.

[†]Department of Computer Science, Virginia Tech, Blacksburg, VA 24061-0106, `nloehr@vt.edu`.

where GF is for *Galois field.* The multiplicative group of $\mathbb{F}$ is denoted $\mathbb{F}^*$ and is always cyclic. A *primitive element* of $\mathbb{F}^*$ is any element that generates the multiplicative group. In particular, if $\alpha \in \mathbb{F}^*$ is a primitive element, then

$$1 = \alpha^0, \alpha, \alpha^2, \ldots, \alpha^{p^n-2}$$

are the elements of $\mathbb{F}^*$. As we will frequently need the cardinality $p^n - 1$ of the multiplicative group, let $\mathrm{M}_{p,n}$ denote $p^n - 1$.

Let $\mathbb{Z}_p[x]$ be the polynomial ring in one unknown over $\mathbb{Z}_p$. A polynomial $f \in \mathbb{Z}_p[x]$ is *irreducible* if $f = gh$ implies that either g or h is a constant. An irreducible polynomial f of degree n is primitive if some root (and hence every root) of f is a primitive element of $\mathrm{GF}(p^n)^*$. Typically, the finite field $\mathrm{GF}(p^n)$ is represented as the quotient ring $\mathbb{Z}_p[x]/(f)$, where f is an irreducible polynomial of degree n. Moreover, if f is primitive, then a representation of the elements of $\mathrm{GF}(p^n)^*$ can be based on a root α of f. For an element $\gamma \in \mathrm{GF}(p^n)^*$, define the *index* of γ to be the smallest integer $i \geq 0$ such that $\alpha^i = \gamma$. Alternatively, we can uniquely represent each element $\beta \in \mathrm{GF}(p^n)$ as a polynomial in α of degree at most $n - 1$:

$$\beta = \sum_{i=0}^{n-1} b_i \alpha^i.$$

The index representation turns multiplication in $\mathrm{GF}(p^n)^*$ into addition modulo $\mathrm{M}_{p,n}$, while the polynomial representation makes for straightforward addition in $\mathrm{GF}(p^n)$. As described in Example 2.52 of Lidl and Niederreiter [5], an index table that provides the mapping from the index representation to the polynomial representation is the key data structure that completes the support for general arithmetic in $\mathrm{GF}(p^n)$.

Challenges arise when representing more than just the two fields $\mathbb{Z}_p$ and $\mathrm{GF}(p^n)$. Consider the case of a chain of fields $\mathbb{Z}_p \subset \mathrm{GF}(p^{n_1}) \subset \mathrm{GF}(p^{n_2})$, where $1 < n_1 < n_2$. In this case, n_1 divides n_2. Suppose that α_1 and α_2 are primitive elements of $\mathrm{GF}(p^{n_1})$ and $\mathrm{GF}(p^{n_2})$, respectively. The cyclic group $\mathrm{GF}(p^{n_1})^*$ is a subgroup of the cyclic group $\mathrm{GF}(p^{n_2})^*$, and the smallest power of α_2 that gives a generator γ of $\mathrm{GF}(p^{n_1})$ is

$$\gamma \;=\; \alpha_2^{\mathrm{M}_{p,n_2}/\mathrm{M}_{p,n_1}}.$$

Arithmetic in this chain of fields, especially multiplication, will be most convenient if $\gamma = \alpha_1$. If f_1 and f_2 are the minimal polynomials of α_1 and α_2, respectively, then it is easy to see that $\alpha_1 = \alpha_2^{\mathrm{M}_{p,n_2}/\mathrm{M}_{p,n_1}}$ implies

$$f_1(x) \;\mid\; f_2\left(x^{\mathrm{M}_{p,n_2}/\mathrm{M}_{p,n_1}}\right).$$

We generalize these observations as follows. Suppose that for each of the subfields $\mathrm{GF}(p^{n'})$ of a finite field $\mathrm{GF}(p^n)$ we have chosen a primitive, irreducible polynomial $f_{p,n'} \in \mathbb{Z}_p[x]$ of degree n'. If whenever $n_1 \mid n_2$ and $n_2 \mid n$, we have

$$f_{p,n_1}(x) \;\mid\; f_{p,n_2}\left(x^{\mathrm{M}_{p,n_2}/\mathrm{M}_{p,n_1}}\right),$$

then we say that the polynomials chosen are *compatible.* Parker [7] inductively defines the *Conway polynomial* $\mathrm{C}_{p,n}$ for each finite field $\mathrm{GF}(p^n)$, giving a particular set of compatible polynomials.[1] First define a *lexicographic order* $<_{\mathrm{lex}}$ on polynomials of degree d in $\mathbb{Z}_p[x]$:

$$a_d x^d + a_{d-1}x^{d-1} + \cdots + a_1 x + a_0 \quad <_{\mathrm{lex}} \quad b_d x^d + b_{d-1}x^{d-1} + \cdots + b_1 x + b_0$$

if, for some i with $d \geq i > 0$, we have

$$a_d = b_d,\; a_{d-1} = b_{d-1}, \ldots, a_i = b_i$$

and

$$(-1)^{d-i} a_i \;<\; (-1)^{d-i} b_i,$$

where the element order $<$ in $\mathbb{Z}_p$ is given by

$$0 < 1 < \cdots < p-1.$$

[1]These Conway polynomials for finite fields are quite different from the Conway polynomials studied by topologists in knot theory. The only similarity between these two families of polynomials is that both were named in honor of the mathematician John H. Conway.

$C_{3,1}$	$x - 2$
$C_{3,2}$	$x^2 - x + 2$
$C_{3,3}$	$x^3 + 2x - 1$
$C_{3,4}$	$x^4 - x^3 + 2$
$C_{3,5}$	$x^5 + 2x - 2$
$C_{3,6}$	$x^6 + 2x^4 + x^2 - x + 2$

Table 1: An illustration of the definition of Conway polynomial.

The base case of the definition of Conway polynomials is $C_{p,1}(x) = x - \gamma$, where γ is the smallest primitive element of $\mathbb{Z}_p$ with respect to the element order. For the general case, choose $C_{p,n}$ to be the lexicographically smallest monic, irreducible, primitive polynomial of degree n such that, for every $n' < n$ satisfying $n' \mid n$, we have

$$C_{p,n}(x) \quad | \quad C_{p,n'}\left(x^{M_{p,n}/M_{p,n'}}\right).$$

This definition yields the Conway polynomials in Table 1 for $p = 3$.

It is quite natural to require that the Conway polynomial $C_{p,n}$ be primitive and compatible with the Conway polynomials $C_{p,d}$ for the proper divisors d of n, since this compatibility allows easy conversions between the representations of elements in $GF(p^d)$ and the representations of those elements in $GF(p^n)$. However, there is no compelling *algebraic* reason for the requirement that the Conway polynomial be minimal with respect to the $<_{\text{lex}}$ ordering. This requirement only serves to make the Conway polynomial unique for each p and n and to simplify the existing brute force algorithms used to generate the Conway polynomials. These algorithms rely on an exhaustive search through the set of monic polynomials in $\mathbb{Z}_p[x]$ of degree n considered in lexicographic order. For each such polynomial, it must be checked whether it is irreducible, primitive, and compatible with "smaller" Conway polynomials. The first polynomial found to pass these checks is $C_{p,n}$. Clearly, the brute force search is an inefficient algorithm, and, as a consequence, very few Conway polynomials are actually known. Conway polynomials are used in the GAP [2] and Magma [4] computational algebra systems to represent finite fields, and tables of known Conway polynomials can be constructed from their built-in functions for Conway polynomials. Scheerhorn [8] also discusses compatible polynomials (which he calls norm-compatible polynomials) and has implemented some algorithms in the AXIOM computational algebra system [3].

In this paper, we demonstrate that Conway polynomials for larger finite fields can often be found much more efficiently than by the brute force search. We first develop some fundamental results about compatible elements in a group in Section 2. Based on those results, in Section 3, we propose two new algorithms for generating Conway polynomials or, more generally, compatible sets of polynomials. One algorithm is based on computing with elements of $\mathrm{GF}(p^n)$, while the second algorithm is based on computing more directly with the definition of compatible polynomials. After implementing the second of these algorithms in Magma, we are able to generate a number of new Conway polynomials; these are presented in Section 4. Finally, we suggest some additional directions for research in Section 5.

2 Fundamental Results

From the definition of Conway polynomials, it is not even clear that $\mathrm{C}_{p,n}$ always exists. Nickel [6] provides a proof of existence, and Scheerhorn [8] also proves the existence of a parallel class of polynomials that he calls trace-compatible polynomials. In this section, we develop some fundamental results that allow us to prove the existence of Conway polynomials and of compatible polynomials, in general. More importantly, these results form the basis and give the insights for the new algorithms we present in the next section. It turns out that the existence of compatible polynomials only depends on the fact that the multiplicative group of a finite field is cyclic. Consequently, we are able to develop results in this section in the context of cyclic groups and their subgroups.

Fix a positive integer k. Let C_k be the cyclic group of order k, written multiplicatively.

For any element $\alpha \in C_k$, denote the order of α by $o(\alpha)$.

A proof of the following lemma can be found, for example, in Lidl and Niederreiter [5], Theorem 1.15.

Lemma 1. *For $\alpha \in C_k$ and i an integer, the order of α^i is*

$$\frac{o(\alpha)}{\gcd(i, o(\alpha))}.$$

The following lemma is easy to prove using group theoretic techniques.

Lemma 2. *Let j and k be integers satisfying $j \mid k$. Let the function $f : C_k \to C_{k/j}$ be given by $f(x) = x^j$. Then f is a surjective group homomorphism. Moreover, for every $y \in C_{k/j}$, there are precisely j elements $x \in C_k$ satisfying $f(x) = x^j = y$.*

We now precisely develop the notion of compatible elements within a cyclic group. Let $\operatorname{div}(k)$ be the set of divisors of k. A *system of compatible generators* for C_k is a partial function

$$\alpha \quad : \quad \operatorname{div}(k) \to C_k,$$

defined on $\operatorname{def}(\alpha) \subset \operatorname{div}(k)$, satisfying these properties:

1. The function is defined on 1, that is, $1 \in \operatorname{def}(\alpha)$;

2. If $i \in \operatorname{def}(\alpha)$, then $o(\alpha(i)) = i$; and

3. If $i \in \operatorname{def}(\alpha)$ and $j \mid i$, then $j \in \operatorname{def}(\alpha)$ and $\alpha(i)^{i/j} = \alpha(j)$.

A system of compatible generators α' is an *extension* of α if $\operatorname{def}(\alpha) \subset \operatorname{def}(\alpha')$ and if $\alpha'(i) = \alpha(i)$ whenever $i \in \operatorname{def}(\alpha)$. If $\operatorname{div}(k) = \operatorname{def}(\alpha)$ then α is a *complete system of compatible generators.*

The key result on systems of compatible generators asserts that a partial system can always be extended to a complete one.

Theorem 3. *Assume that α is a system of compatible generators for C_k. Then there exists a complete system α' of compatible generators for C_k that extends α.*

Proof: If $k \in \operatorname{def}(\alpha)$, then the theorem immediately follows. Hence, we may assume that $k \notin \operatorname{def}(\alpha)$. We first show how to extend $\operatorname{def}(\alpha)$ by one element. That is, we show that there exists a system of compatible generators α' satisfying $\alpha'(i) = \alpha(i)$ whenever $i \in \operatorname{def}(\alpha)$ and $|\operatorname{def}(\alpha') - \operatorname{def}(\alpha)| = 1$. Let $s = \min T$ be the smallest integer in $\operatorname{div}(k) - \operatorname{def}(\alpha)$. Observe that every proper divisor of s is in $\operatorname{def}(\alpha)$, for otherwise there would be a smaller integer in $\operatorname{div}(k) - \operatorname{def}(\alpha)$.

Let $s = p_1^{e_1} p_2^{e_2} \cdots p_m^{e_m}$ be the unique prime factorization of s. For $1 \leq i \leq m$, define $q_i = s/p_i$. By the observation above, each $q_i \in \operatorname{def}(\alpha)$. Also, each of the $\alpha(q_i)$ is in C_s, the unique cyclic subgroup of C_k of order s.

First suppose that $m = 1$ and $e_1 = 1$. Define α' to be a system of compatible generators that extends α by the one element s, where $\alpha'(s)$ is chosen to be any one of the $s - 1$ generators of C_s.

Now suppose that $m = 1$ and $e_1 > 1$. Then $\alpha(p_1^{e_1 - 1})$ has p_1 distinct p_1th roots in C_s; by Lemma 1, each of these roots has order s and thus generates C_s. Define α' to be a system of compatible generators that extends α by the one element s, where $\alpha'(s)$ is chosen to be any p_1th root of $\alpha(p_1^{e_1 - 1})$.

Finally suppose that $m > 1$. Let $\gamma \in C_k$ be an element of order s, that is, a generator of C_s. There exists r_i satisfying

$$\gamma^{r_i} = \alpha(q_i),$$

for all i with $1 \leq i \leq m$. Since $o(\gamma) = s$ and $o(\alpha(q_i)) = q_i$, we know by Lemma 1 that

$$q_i = \frac{s}{\gcd(r_i, s)}$$

and hence that

$$p_i = \gcd(r_i, s).$$

Applying the generalized Chinese remainder theorem (see Bach and Shallit [1], Section 5.5), we obtain an integer x satisfying the system of congruences

$$x \equiv \frac{r_i}{p_i} \pmod{q_i}, \tag{1}$$

provided that

$$\frac{r_i}{p_i} \equiv \frac{r_j}{p_j} \pmod{\gcd(q_i, q_j)}, \tag{2}$$

for every pair i, j, where $1 \le i < j \le m$. Furthermore, if x exists, it is unique modulo

$$\operatorname{lcm}(q_1, q_2, \ldots, q_m) = s,$$

since $m > 1$.

To establish the congruences (2), fix i and j satisfying $1 \le i < j \le m$. Eliminating q_i and q_j from the congruences (2), we obtain

$$\frac{r_i}{p_i} \equiv \frac{r_j}{p_j} \pmod{s/(p_i p_j)}.$$

Now the element $s/(p_i p_j) \in \operatorname{def}(\alpha)$, by the definition of s. Furthermore,

$$\alpha(s/(p_i p_j)) = \gamma^{r_i p_j} = \gamma^{r_j p_i}.$$

It follows that

$$r_i p_j \equiv r_j p_i \pmod{s}$$

and that

$$\frac{r_i}{p_i} \equiv \frac{r_j}{p_j} \pmod{s/(p_i p_j)},$$

as required. We obtain x satisfying the system of congruences (1). Equivalently, x satisfies this system of congruences:

$$x p_i \equiv r_i \pmod{s}. \tag{3}$$

We now define α' to be a system of compatible generators that extends α by the one element s, where $\alpha'(n) = \gamma^x$. Since x is unique modulo s, γ^x is uniquely defined. We must verify that α' is also a system of compatible generators.

First note that

$$\begin{aligned}(\alpha'(s))^{s/q_i} &= (\gamma^x)^{p_i} \\ &= \gamma^{xp_i} \\ &= \gamma^{r_i} \\ &= \alpha(q_i),\end{aligned}$$

by the system of congruences (3) and the fact that $o(\gamma) = s$.

Second we must show that $o(\alpha'(s)) = s$. Observe that, for each i, $\alpha'(s)^{p_i}$ generates the cyclic group of order q_i. Since $m > 1$, the only subgroup of C_s that contains all those cyclic groups is C_s itself. Thus, $\alpha'(s)$ must generate C_s. We conclude that $o(\alpha'(s)) = s$.

By iteratively extending α for $|\text{div}\,(k) - \text{def}\,(\alpha)\,|$ steps, we reach an extension α' that is a complete system of compatible generators. The theorem follows. $\square$

We also wish to count the number of extensions of α to a complete system of compatible generators. For a prime p and an integer n, define $\nu_p(n)$ to be the highest power of p that divides n; that is, $\nu_p(n) = p^e$, where $p^e \mid n$ and $p^{e+1} \not| \, n$. For a prime p and a pair of integers m and n such that $m \mid n$, define the *p-contribution* of m to n to be

$$\tau_p(m, n) = \begin{cases} \phi(\nu_p(n)) & \text{if } p \not| \, m; \\ \nu_p(n)/\nu_p(m) & \text{if } p \mid m. \end{cases}$$

If M is a set of divisors of n, define the *p-contribution* of M to n to be

$$\tau_p(M, n) = \min_{m \in M} \tau_p(m, n),$$

and define the *contribution* of M to n to be

$$\tau(M, n) = \prod_{p|n} \tau_p(M, n),$$

where p ranges over the prime divisors of n.

Theorem 4. *If α is a system of compatible generators for C_k, then the number of extensions of α to a complete system of compatible generators is $\tau(\text{def}\,(\alpha)\,, k)$.*

Proof: Considering again the proof of Theorem 3, we see that there are three cases to consider in extending α to a complete system of compatible generators, corresponding to the three cases for defining α' for elements of $\mathrm{div}\,(k) - \mathrm{def}\,(\alpha)$.

1. The first case is $p \in \mathrm{div}\,(k) - \mathrm{def}\,(\alpha)$ for some prime p. Then in the extension step that defines a value for $\alpha'(p)$, there are $p-1$ possible values from which to choose.

2. The second case is $p^e \in \mathrm{div}\,(k) - \mathrm{def}\,(\alpha)$ for some prime p and some $e \geq 2$. Then in the extension step that defines a value for $\alpha'(p^e)$, there are p possible values from which to choose.

3. The third case is $s \in \mathrm{div}\,(k) - \mathrm{def}\,(\alpha)$ where s has two or more distinct prime factors. In this extension step there is a unique choice for $\alpha'(s)$.

The total number of extensions of α to a complete system of compatible generators is the product of the number of choices at each step. By examining the cases above, we see that there is more than one choice only when defining the extension on a prime power. Hence, to count the number of extensions, we may take the product of the contributions of the prime factors of k. It is clear that each prime p contributes $\tau_p(\mathrm{def}\,(\alpha)\,, k)$. Therefore, the total number of extensions is $\tau(\mathrm{def}\,(\alpha)\,, k)$, as claimed. □

We can now apply this result to finite fields. Fix a prime p and a positive integer n. A *system of (primitive) roots* for $\mathrm{GF}(p^n)$ is a partial function

$$\beta \ : \ \mathrm{div}\,(n) \to \mathrm{GF}(p^n),$$

defined on $\mathrm{def}\,(\beta) \subset \mathrm{div}\,(n)$, satisfying this property: If $i \in \mathrm{def}\,(\beta)$, then $o(\beta(i)) = \mathrm{M}_{p,i}$, that is, $\beta(i)$ is a primitive element of $\mathrm{GF}(p^i)$. A system of roots β' is an *extension* of β if $\mathrm{def}\,(\beta) \subset \mathrm{def}\,(\beta')$ and if $\beta'(i) = \beta(i)$ whenever $i \in \mathrm{def}\,(\beta)$.

If β is a system of roots, then two roots $\beta(i)$ and $\beta(j)$ are *compatible* if one of these holds:

1. Neither of i and j divides the other;
2. If i divides j, then $\beta(j)^{\mathrm{M}_{p,j}/\mathrm{M}_{p,i}} = \beta(i)$; or
3. If j divides i, then $\beta(i)^{\mathrm{M}_{p,i}/\mathrm{M}_{p,j}} = \beta(j)$.

If $\beta(i)$ and $\beta(j)$ are compatible for every pair i and j, then β is a *compatible system of roots.* If div $(n) =$ def (β) then β is a *complete system of compatible roots.*

Theorem 5. *Let p be a prime and let n be a positive integer. Then there exists a complete system of compatible roots β for* $\mathrm{GF}(p^n)$*.*

Proof: We show how to define each $\beta(i)$ by induction on i. Define $\beta(1)$ to be any one of the $\phi(p-1)$ primitive elements of the multiplicative group $\mathrm{GF}(p)^*$.

Now assume that $i > 1$ and that $\beta(j)$ is defined for all $j \in$ div (n) with $j < i$. We need to define $\beta(i)$ in $G = \mathrm{GF}(p^i)^*$. Now G is a cyclic group of order $k = \mathrm{M}_{p,i}$. We can define a system of compatible generators α for G as follows. For any t that divides k, choose the smallest $j < i$ such that $j \mid i$ and $t \mid \mathrm{M}_{p,j}$. If such a j exists, then by the inductive hypothesis we know that $\beta(j)$ is defined. Define

$$\alpha(t) \;=\; \beta(j)^{\mathrm{M}_{p,j}/t}.$$

To show that α is a system of compatible generators, we need to show that if $t_1, t_2 \in$ def (α) and $t_1 \mid t_2$, then $\alpha(t_2)^{t_2/t_1} = \alpha(t_1)$. Let j_1 and j_2 be such that $\alpha(t_1)$ is defined to be $\beta(j_1)^{\mathrm{M}_{p,j_1}/t_1}$ and $\alpha(t_2)$ is defined to be $\beta(j_2)^{\mathrm{M}_{p,j_2}/t_2}$. Then the smallest subfield of $\mathrm{GF}(p^i)$ containing $\alpha(t_1)$ is $\mathrm{GF}(p^{j_1})$, while the smallest subfield of $\mathrm{GF}(p^i)$ containing $\alpha(t_2)$ is $\mathrm{GF}(p^{j_2})$. Since $t_1 \mid t_2$, we have that $\alpha(t_1)$ is also in $\mathrm{GF}(p^{j_2})$. This implies that

$\mathrm{GF}(p^{j_1}) \subset \mathrm{GF}(p^{j_2})$ and $j_1 \mid j_2$. Hence

$$\begin{aligned}
\alpha(t_2)^{t_2/t_1} &= \left(\beta(j_2)^{\mathrm{M}_{p,j_2}/t_2}\right)^{t_2/t_1} \\
&= \beta(j_2)^{\mathrm{M}_{p,j_2}/t_1} \\
&= \left(\beta(j_2)^{\mathrm{M}_{p,j_2}/\mathrm{M}_{p,j_1}}\right)^{\mathrm{M}_{p,j_1}/t_1} \\
&= \beta(j_1)^{\mathrm{M}_{p,j_1}/t_1} \\
&= \alpha(t_1),
\end{aligned}$$

as required.

Hence α is a system of compatible generators. By Theorem 3, α can be extended to a complete system of compatible generators α' for G. Define $\beta(i) = \alpha'(k)$. It is easily seen that $\beta(i)$ is compatible with all $\beta(j)$ with $j < i$.

The existence of β now follows by induction. □

Theorem 6. *Let p be a prime, and let n be at least 2. Suppose β is a system of compatible roots for all the subfields of* $\mathrm{GF}(p^n)$. *Let $M = \{\mathrm{M}_{p,n} : 1 \leq i < n$ and $i \mid n\}$. Then the number of choices for a primitive element of* $\mathrm{GF}(p^n)$ *that is compatible with β is $\tau(M, \mathrm{M}_{p,n})$.*

Proof: As in Theorem 5, we construct a system of compatible generators α for β and extend it to a complete system of compatible generators for $\mathrm{GF}(p^n)^*$. Theorem 4 tells us that the number of such complete systems of compatible generators is $\tau(M, \mathrm{M}_{p,n})$. Each such system corresponds to a primitive element of $\mathrm{GF}(p^n)$ that is compatible with β. The theorem follows. □

Theorem 7. *The Conway polynomial $C_{p,n}$ exists for all primes p and all positive integers n.*

Proof: We fix a prime p and show that $C_{p,n}$ exists by induction on n. For $n = 1$, let $C_{p,1}(x) = x - \gamma$, where γ is the smallest primitive element in $\mathrm{GF}(p)$. Note that γ exists, since $\mathrm{GF}(p)^*$ is cyclic. Clearly $C_{p,1}$ is the lexicographically smallest, monic, irreducible, primitive polynomial of degree 1.

Now suppose that $n > 1$ and that $C_{p,i}$ is defined for all $i < n$. By induction on proper divisors of n in increasing order, we may define a system of compatible roots β in $\mathrm{GF}(p^n)$ such that

1. For each $i < n$ dividing n, we have $\beta(i)$ is a root of $C_{p,i}$;
2. Whenever $i \mid j$, $j \mid n$, and $j < n$, we have that $\beta(i) = \beta(j)^{\mathrm{M}_{p,j}/\mathrm{M}_{p,i}}$.

By Theorems 5 and 6, we can extend β to a complete system of compatible roots by defining $\beta(n)$ to be any of a number of primitive elements of $\mathrm{GF}(p^n)$. Of those values, we can choose a primitive element with the lexicographically smallest monic minimal polynomial. (The number of such primitive elements is $\tau(M, \mathrm{M}_{p,n}) > 0$, where M is as defined in Theorem 6.) Define $C_{p,n}$ to be that polynomial.

The theorem follows by induction. $\square$

3 New Algorithms

Building on the ideas in Section 2, we present two new algorithms for generating Conway polynomials. To provide a point of comparison, we first review the brute force algorithm.

The following notation will be used throughout this section. Fix a prime p, and suppose n is a positive integer. Let n have prime factorization $n = q_1^{e_1} \cdots q_s^{e_s}$. For $1 \leq i \leq s$, set $d_i = n/q_i$ and $m_i = \mathrm{M}_{p,n}/\mathrm{M}_{p,d_i}$. Finally, set $g = \gcd_{1 \leq i \leq s}\{m_i\}$ and $n_i = m_i/g$.

3.1 An Algorithm Based on Exhaustive Search

We first describe the brute force algorithm currently used by GAP and Magma to compute Conway polynomials. We present this algorithm here to contrast its performance with our two new algorithms given later.

The simplest version of the brute force algorithm to compute $\mathrm{C}_{p,n}$ starts by looking up or calculating $\mathrm{C}_{p,d}$ for all proper divisors d of n. Next, the algorithm enumerates the monic polynomials of degree n over $\mathbb{Z}_p$ in lexicographic order. Each polynomial is checked for primitivity and for compatibility with the polynomials $\mathrm{C}_{p,d}$. The first polynomial passing both checks is $\mathrm{C}_{p,n}$.

Note that the search space for this algorithm has size p^n, since there are p possible coefficients for each power of x from 0 to $n-1$ inclusive. It will be shown in Theorem 9 that the number of monic polynomials of degree n that are compatible with the lower order Conway polynomials is g; moreover, by Theorem 6, the number of primitive candidates among the g compatible candidates is $\tau(M, \mathrm{M}_{p,n})$. In any case, there are no more than g primitive, compatible polynomials in the search space. Hence, assuming that these polynomials are distributed randomly (uniformly) in the lexicographic listing of all degree n polynomials, we expect the brute force algorithm to test *roughly* p^n/g polynomials before finding the first acceptable one. If n is composite and even moderately large (say, $n \geq 40$), then $g << p^n$ in general, and the brute force algorithm is impractical.

One improvement to the naive algorithm is obtained by noting that the constant term of $\mathrm{C}_{p,n}$ must be $(-1)^n\gamma$, where γ is the smallest primitive element of $\mathrm{GF}(p)$. This observation easily follows from the requirement that $\mathrm{C}_{p,n}$ be compatible with $\mathrm{C}_{p,1}$. Knowing the constant term reduces the size of the search space by a factor of p. Unfortunately, there is no analogous quick method for obtaining the higher order coefficients of $\mathrm{C}_{p,n}$.

Another improvement to the algorithm involves the compatibility checks with lower order polynomials. Recall that $d_1, \ldots, d_s$ are the maximal proper divisors of n. Suppose $r(x)$ is a particular candidate polynomial that is compatible with all the polynomials $\mathrm{C}_{p,d_i}(x)$ in the sense that

$$r(x) \mid \mathrm{C}_{p,d_i}\left(x^{\mathrm{M}_{p,n}/\mathrm{M}_{p,d_i}}\right). \tag{4}$$

Let d be any proper divisor of n. Then d divides some d_i. By definition,

$$\mathrm{C}_{p,d_i}(x) \mid \mathrm{C}_{p,d}\left(x^{\mathrm{M}_{p,d_i}/\mathrm{M}_{p,d}}\right)$$

so that

$$\mathrm{C}_{p,d_i}\left(x^{\mathrm{M}_{p,n}/\mathrm{M}_{p,d_i}}\right) \mid \mathrm{C}_{p,d}\left(\left[x^{\mathrm{M}_{p,n}/\mathrm{M}_{p,d_i}}\right]^{\frac{\mathrm{M}_{p,d_i}}{\mathrm{M}_{p,d}}}\right) = \mathrm{C}_{p,d}\left(x^{\mathrm{M}_{p,n}/\mathrm{M}_{p,d}}\right).$$

Thus, if (4) holds for all maximal proper divisors d_i of n, it follows that

$$r(x) \mid \mathrm{C}_{p,d}\left(x^{\mathrm{M}_{p,n}/\mathrm{M}_{p,d}}\right) \tag{5}$$

for all proper divisors d of n. Of course, this observation reduces the number of compatibility conditions to check per candidate polynomial.

3.2 An Algorithm Based on Elements

We present our first new algorithm for generating Conway polynomials. It is inspired by the proofs of the results in Section 2. To find the Conway polynomial $\mathrm{C}_{p,n}$, we must know inductively the Conway polynomials C_{p,d_i}, for $1 \leq i \leq s$. First note that the cardinalities of the multiplicative groups of the maximal subfields of $\mathrm{GF}(p^n)$ are $f_i = \mathrm{M}_{p,d_i}$. We choose a root x_i for each C_{p,d_i}. We know that $o(x_i) = f_i$. In the multiplicative group $G = \mathrm{GF}(p^n)^*$, we can find an element $x_{1,2}$ of order $\mathrm{lcm}\{f_1, f_2\}$ that is compatible with x_1 and x_2. In another step, we can find an element $x_{1,2,3}$ of order $\mathrm{lcm}\{f_1, f_2, f_3\}$ that is compatible with x_1, x_2, and x_3. Iterating, we can find an element $x_{1,2,\ldots,s}$ of order

$$f = \mathrm{lcm}\{f_1, f_2, \ldots, f_s\}$$

that is compatible with $x_1, x_2, \ldots, x_s$. Note that $g = \mathrm{M}_{p,n}/f$. Finally, all gth roots of $x_{1,2,\ldots,s}$ that are primitive elements of $\mathrm{GF}(p^n)$ are candidates for being the roots of the Conway polynomial $\mathrm{C}_{p,n}$.

The algorithm appears in Figure 1. The time complexity of the algorithm is dominated by the loop in steps 16-19 that searches through g values in $\mathrm{GF}(p^n)$. Hence the algorithm is almost linear as a function of g.

```
GenerateConway($p, n$)
1   let $n = q_1^{e_1} \cdots q_s^{e_s}$ be the prime factorization of $n$
2   $r \leftarrow \mathrm{M}_{p,n}$
3   for $i \leftarrow 1$ to $s$
4       do $d_i \leftarrow n/q_i$
5          $f_i \leftarrow \mathrm{M}_{p,d_i}$
6          $m_i \leftarrow r/f_i$
7          $x_i \leftarrow$ a root of $\mathrm{C}_{p,d_i}$
8   $x \leftarrow x_1$
9   $v \leftarrow f_1$
10  for $i \leftarrow 2$ to $s$
11      do find $\alpha, \beta$ such that $\alpha\, v + \beta f_i = \gcd\{v, f_i\}$
12         $x \leftarrow x^{\alpha} x_i^{\beta}$
13         $v \leftarrow \mathrm{lcm}\{v, f_i\}$
14  $g \leftarrow r/v$
15  min_poly $\leftarrow \infty$
16  for $z$ a $g$th root of $x$ in GF($p^n$)
17      do poly $\leftarrow$ minimum polynomial of $z$
18         if poly is primitive and poly $<$ min_poly
19            then min_poly $\leftarrow$ poly
20  return min_poly
```

Figure 1: First algorithm for generating Conway polynomials.

3.3 An Algorithm Based on Polynomials

Recall that, by definition, the Conway polynomials must satisfy the compatibility conditions

$$\mathrm{C}_{p,n}(x) \mid \mathrm{C}_{p,d}\left(x^{\mathrm{M}_{p,n}/\mathrm{M}_{p,d}}\right) \tag{6}$$

for all proper divisors d of n. By the transitivity relation (5), it suffices to check the condition (6) for the maximal proper divisors of n, namely $d_1, \ldots, d_s$. Clearly, the compatibility condition holds for all these divisors if and only if

$$\mathrm{C}_{p,n}(x) \mid \gcd_{1 \leq i \leq s} \{\mathrm{C}_{p,d_i}(x^{m_i})\}. \tag{7}$$

Thus, if we know $\mathrm{C}_{p,d_i}(x)$ for each i, we can find $\mathrm{C}_{p,n}(x)$ in principle by simply computing the GCD of the polynomials $\mathrm{C}_{p,d_i}(x^{m_i})$, factoring the resulting polynomial, and picking the lexicographically smallest primitive, irreducible factor of degree n. Unfortunately, the degree of the polynomial

$$f(x) = \gcd_{1 \leq i \leq s} \{\mathrm{C}_{p,d_i}(x^{m_i})\}$$

is typically very large compared to n, making it difficult to factor f.

To obtain a viable algorithm, we introduce a new unknown $z = x^g$, where $g = \gcd_{1 \leq i \leq s}\{m_i\}$, as before. Note that each polynomial $\mathrm{C}_{p,d_i}(x^{m_i})$ in x can be written as a polynomial $\mathrm{C}_{p,d_i}(z^{m_i/g})$ in z. Hence, $f(x) = \gcd_{1 \leq i \leq s}\{\mathrm{C}_{p,d_i}(x^{m_i})\}$ can also be written as a polynomial $r(z)$ in the unknown z. The polynomial $r(z)$ has some useful properties, given by the following theorem.

In proving this theorem, we need the following result from finite field theory (pages 49–50 of Lidl and Niederreiter [5]).

Lemma 8. *If f is an irreducible polynomial of degree d over* $\mathrm{GF}(p)$*, then all the roots of f are in* $\mathrm{GF}(p^d)$*. If α is one root of* $\mathrm{GF}(p^d)$*, then all the roots of f are $\alpha, \alpha^p, \alpha^{p^2}, \ldots, \alpha^{p^d-1}$ and these are all distinct.*

In general, if $\alpha \in \mathrm{GF}(p^d)$ but not necessarily primitive, then the elements α^{p^i-1}, where $0 \leq i \leq d-1$, are called the *conjugates* of α in $\mathrm{GF}(p^d)$.

Theorem 9. *Let $n > 1$. Using the notation above, the polynomial*

$$r(z) = \gcd_{1 \le i \le s} \{\mathrm{C}_{p,d_i}(z^{m_i/g})\}$$

is a monic irreducible polynomial of degree n, provided that s, the number of distinct primes dividing n, is at least 2. If $s = 1$, then we have $r(z) = \mathrm{C}_{p,n/q_1}(z)$.

Moreover, if $z_0 \in \mathrm{GF}(p^n)$ is any root of $r(z)$, then z_0 has exactly g distinct gth roots $x_1, \ldots, x_g \in \mathrm{GF}(p^n)$, and these roots satisfy the compatibility property

$$\mathrm{C}_{p,d_i}(x_j^{m_i}) = 0 \text{ for } 1 \le i \le s \text{ and } 1 \le j \le g.$$

Among these roots, one that is primitive and whose minimal polynomial is lexicographically smallest has the Conway polynomial $C_{p,n}$ as its minimal polynomial.

Proof: If $s = 1$, it is obvious that $r(z) = \mathrm{C}_{p,n/q_1}(z)$. So assume $s > 1$. Clearly, $r(z)$ is monic. Write $r(z)$ as the product of t monic, non-constant, irreducible polynomials over $\mathbb{Z}_p$, say

$$r(z) = f_1(z) \cdots f_t(z).$$

By the remarks preceding the theorem, we know that the Conway polynomial $\mathrm{C}_{p,n}(x)$ must divide $r(z) = r(x^g)$. Thus, $r(z)$ is non-constant, and $t > 0$.

Next, since each polynomial $\mathrm{C}_{p,d_i}(x)$ is irreducible, we have $\gcd(\mathrm{C}_{p,d_i}(x), \frac{d}{dx}\mathrm{C}_{p,d_i}(x)) = 1$ for all i. Setting $u = z^{m_i/g} = z^{n_i}$, we see that

$$\begin{aligned} \gcd\left(\mathrm{C}_{p,d_i}(z^{n_i}), \frac{d}{dz}\mathrm{C}_{p,d_i}(z^{n_i})\right) &= \gcd\left(\mathrm{C}_{p,d_i}(u), n_i z^{n_i-1}\frac{d}{du}\mathrm{C}_{p,d_i}(u)\right) \\ &= 1. \end{aligned}$$

Thus, $\mathrm{C}_{p,d_i}(z^{n_i})$ has no repeated factors, for each i, implying that $r(z)$ cannot have any repeated factors either. Thus, the factors $f_i(z)$ of $r(z)$ are all distinct.

Fix j with $1 \le j \le t$, and let d be the degree of $f_j(z)$. Let z_j be a root of $p_j(z)$ in the extension field $\mathrm{GF}(p^d)$ of $\mathbb{Z}_p$. Since $r(z_j) = 0$, we have $\mathrm{C}_{p,d_i}(z_j^{n_i}) = 0$. Thus, some power of

z_j, namely $z_j^{n_i}$, lies in the field $\mathrm{GF}(p^{d_i})$, so that $\mathrm{GF}(p^{d_i}) \subset \mathrm{GF}(p^d)$ and hence $d_i \mid d$. Since $s \geq 2$, we have $n = \mathrm{lcm}(d_i) \mid d$. Moreover, since

$$z_j^{\mathrm{M}_{p,n}} = \left(z_j^{m_i/g}\right)^{g\mathrm{M}_{p,d_i}} = 1^g = 1,$$

$z_j \in \mathrm{GF}(p^n)$ and so $d \mid n$. Hence, $d = n$, and all irreducible factors of $r(z)$ must have degree n.

Finally, we claim that $r(z)$ has only one such factor, i.e., that $t = 1$. To prove this claim, by Lemma 8 it suffices to show that any two roots of $r(z)$ are conjugates in $\mathrm{GF}(p^n)$. So let z_0 and z_1 be arbitrary roots of $r(z)$. Note that both the elements $z_0^{n_i}$ and $z_1^{n_i}$ have C_{p,d_i} as their minimal polynomial; it follows that $z_1^{n_i}$ must be a conjugate of $z_0^{n_i}$ in $\mathrm{GF}(p^{d_i})$, say $z_1^{n_i} = (z_0^{n_i})^{p^{c_i}}$ for some c_i with $0 \leq c_i < d_i$. We claim that there exists an integer c such that $c \equiv c_i \pmod{d_i}$ for all i. This follows from the Extended Chinese Remainder Theorem, provided that $c_i \equiv c_j \pmod{n/(q_i q_j)}$ for $1 \leq i < j \leq s$. To verify these congruences, fix i and j satisfying $1 \leq i < j \leq s$, and define $e = n/(q_i q_j)$. Further define $b_i = \mathrm{M}_{p,d_i}/\mathrm{M}_{p,e}$ and $b_j = \mathrm{M}_{p,d_j}/\mathrm{M}_{p,e}$. After computing

$$\begin{aligned} n_i b_i &= \frac{\mathrm{M}_{p,n}}{g\mathrm{M}_{p,d_i}} \frac{\mathrm{M}_{p,d_i}}{\mathrm{M}_{p,e}} \\ &= \frac{\mathrm{M}_{p,n}}{g\mathrm{M}_{p,d_j}} \frac{\mathrm{M}_{p,d_j}}{\mathrm{M}_{p,e}} \\ &= n_j b_j, \end{aligned}$$

we have that

$$z_0^{n_i b_i} = z_0^{n_j b_j}$$

is a primitive element of $\mathrm{GF}(p^e)$ (because it is a root of $\mathrm{C}_{p,e}$), as is

$$z_1^{n_i b_i} = z_1^{n_j b_j}.$$

We can now compute

$$\begin{aligned}
\left(z_0^{n_i b_i}\right)^{p^{c_i}} &= \left(z_0^{n_i p^{c_i}}\right)^{b_i} \\
&= z_1^{n_i b_i} \\
&= z_1^{n_j b_j} \\
&= \left(z_0^{n_j p^{c_j}}\right)^{b_j} \\
&= \left(z_0^{n_j b_j}\right)^{p^{c_j}} \\
&= \left(z_0^{n_i b_i}\right)^{p^{c_j}} .
\end{aligned}$$

Hence $c_i \equiv c_j \pmod{e}$, as desired.

We now have an integer c such that $z_1^{n_i} = (z_0^{n_i})^{p^c}$ for all i. Since $\gcd\{n_1, \ldots, n_s\} = 1$, there exist integers a_i such that $\sum_{i=1}^{s} a_i n_i = 1$. Therefore, we obtain

$$\begin{aligned}
\prod_{i=1}^{s} (z_1^{n_i})^{a_i} &= \prod_{i=1}^{s} \left(z_0^{n_i p^c}\right)^{a_i} \\
z_1^{\sum_{i=1}^{s} a_i n_i} &= z_0^{p^c \sum_{i=1}^{s} a_i n_i},
\end{aligned}$$

implying that

$$z_1 = z_0^{p^c}.$$

Hence, z_0 and z_1 are conjugates in $\mathrm{GF}(p^n)$, forcing $t = 1$. This completes the proof of the first part of the theorem.

For the rest of the proof, we assume that $s \geq 1$. Let z_0 denote any fixed root of $r(z)$ in $\mathrm{GF}(p^n)$. Since

$$z_0^{\mathrm{M}_{p,n}/g} = (z_0^{m_1/g})^{\mathrm{M}_{p,d_1}} = 1,$$

the order of z_0 must divide $\mathrm{M}_{p,n}/g$. By Lemma 2, z_0 has g distinct gth roots in $\mathrm{GF}(p^n)$. If x_j is such a root, then for every i

$$\mathrm{C}_{p,d_i}(x_j^{m_i}) = \mathrm{C}_{p,d_i}(z_0^{m_i/g}) = 0,$$

so that the roots are compatible with previously chosen Conway polynomials. Conversely, by the definition of $r(z)$, any compatible field element x_0 must have a minimal polynomial $m(x)$ that divides $r(z) = r(x^g)$; hence, x_0 appears among the gth roots of z_0. It follows immediately that one of the roots x_j has $\mathrm{C}_{p,n}(x)$ as its minimal polynomial. The correct root must clearly be the primitive root whose minimal polynomial is lexicographically smallest; the existence of a primitive root is guaranteed by Theorem 7. $\square$

This theorem immediately leads to the following algorithm for finding the Conway polynomial $\mathrm{C}_{p,n}$. We begin by finding the prime factorization $q_1^{e_1} \cdots q_s^{e_s}$ of n. The algorithm splits into three cases, depending on the number of distinct primes s in the factorization of n.

- Case I: $s \geq 2$.

    1. Look up (or recursively calculate) the Conway polynomials $\mathrm{C}_{p,d_i}(x)$ for each maximal proper divisor $d_i = n/q_i$.
    2. Find $g = \gcd_{1 \leq i \leq s}\{\mathrm{M}_{p,n}/\mathrm{M}_{p,d_i}\}$. Setting $z = x^g$, compute
    $$r(z) = \gcd_{1 \leq i \leq s} \{\mathrm{C}_{p,d_i}(z^{m_i/g})\}$$
    using any standard algorithm for finding polynomial greatest common divisors.
    3. Let z_0 denote a root of $r(z)$ in $\mathrm{GF}(p^n)$. Find any gth root α of z_0 in $\mathrm{GF}(p^n)$, and let ζ be a primitive gth root of unity in $\mathrm{GF}(p^n)$. Many well-known algorithms exist to compute the field elements α and ζ; see, for example, Tonelli's algorithm in Section 7.3 of Bach and Shallit [1]. Since $r(z)$ is an irreducible polynomial of degree n over $\mathbb{Z}_p[z]$ (by Theorem 9), it is convenient to perform the root extraction algorithm using the field representation
    $$\mathrm{GF}(p^n) = \mathbb{Z}_p[z]/(r(z)).$$
    It is easy to define $\mathrm{GF}(p^n)$ in this way using Magma.

4. Observe that all the gth roots of z_0 in $\mathrm{GF}(p^n)$ are of the form $\alpha\zeta^k$, for $0 \le k < g$. Consider each of these roots in turn. Compute the minimal polynomials corresponding to each *primitive* root, and return the lexicographically smallest polynomial so computed. By Theorem 6, there are exactly $\tau(M, \mathrm{M}_{p,n})$ primitive roots among the gth roots of z_0; note that $\tau(M, \mathrm{M}_{p,n}) > 0$.

   More generally, we can obtain a compatible system of polynomials for any given p by stopping the search through the gth roots of z_0 as soon as we find the first primitive element that is compatible with the previously chosen primitive elements (or polynomials). This possibility is discussed in more detail later.

- Case II: $s = 1$. This case is really a degenerate form of the previous one. Note that $n = q_1^{e_1}$, $g = \mathrm{M}_{p,n}/\mathrm{M}_{p,n/q_1}$, and $r(z) = \mathrm{C}_{p,n/q_1}(z)$. As before, let z_0 denote any root of q, considered as an element of $\mathrm{GF}(p^n) \supset \mathrm{GF}(p^{n/q_1})$. Hence, as above, we cycle through all g of these roots, and return the lexicographically smallest minimal polynomial of degree n that corresponds to a *primitive* root.

- Case III: $s = 0$, i.e., $n = 1$. In this case, we are simply looking for $\mathrm{C}_{p,1}(x)$, the Conway polynomial for the prime field $\mathbb{Z}_p$. We simply test each element $1, 2, \ldots$ of $\mathbb{Z}_p$ for primitivity until we find the first primitive element γ. (Testing an element for primitivity can be done in polynomial time. See Bach and Shallit [1], Exercise 5.8.) Then, by definition, $\mathrm{C}_{p,1}(x) = x - \gamma$.

Theorem 9 proves the correctness of this algorithm for Cases I and II, and the algorithm is obviously correct for the last case.

Example 10. Suppose we wish to find $\mathrm{C}_{2,6}(x)$. In this case, $p = 2$, $n = 6$, $q_1 = 2$, $q_2 = 3$, $s = 2$, $d_1 = 3$, $d_2 = 2$, $m_1 = (2^6 - 1)/(2^3 - 1) = 9$, $m_2 = (2^6 - 1)/(2^2 - 1) = 21$, and $g = \gcd(9, 21) = 3$. We look up $\mathrm{C}_{2,3}(x) = x^3 + x + 1$ and $\mathrm{C}_{2,2}(x) = x^2 + x + 1$. Setting $z = x^3$, we have

$$\begin{aligned} \mathrm{C}_{2,3}(x^9) &= x^{27} + x^9 + 1 &= z^9 + z^3 + 1 \\ \mathrm{C}_{2,2}(x^{21}) &= x^{42} + x^{21} + 1 &= z^{14} + z^7 + 1. \end{aligned}$$

The greatest common divisor of these two polynomials is

$$f(x) = 1 + x^6 + x^{12} + x^{15} + x^{18} = 1 + z^2 + z^4 + z^5 + z^6 = r(z).$$

This is an irreducible polynomial in the unknown z, but factors as a polynomial in x as

$$f(x) = (1 + x + x^3 + x^4 + x^6)(1 + x^5 + x^6)(1 + x + x^2 + x^5 + x^6).$$

Letting z_0 denote a root of $r(z)$ in $\mathrm{GF}(2^6)$, it is easy to check that the three irreducible factors of $f(x)$ are the minimal polynomials of the three cube roots of z_0 in $\mathrm{GF}(2^6)$. The smallest of these factors relative to $<_{\mathrm{lex}}$, namely $x^6 + x^4 + x^3 + x + 1$, is $\mathrm{C}_{2,6}(x)$. In this case, since n was small, we found the candidate polynomials directly by factoring $f(x)$. In practice, of course, these polynomials are found one at a time by taking a cube root α of z_0 in $\mathrm{GF}(2^6)$, and then checking the minimal polynomials of each primitive cube root of z_0 in this field.

Assuming $n > 1$, the running time of the root-checking phase of the algorithm clearly grows linearly with g. In fact, for $s < 3$ the size of g determines those values of p and n for which it is practical to compute $\mathrm{C}_{p,n}$ with this algorithm. Unfortunately, for any fixed p, g grows in a highly irregular and choppy manner as n is increased. If n is itself a prime power, g is particularly "large", and our algorithm is very slow. Ironically, the brute force algorithm performs better in this case, since the search space contains more primitive, compatible candidates (hence the *first* acceptable candidate will be found more quickly). See Section 4.2 for a more detailed comparison.

4 Implementation and Results

Here we discuss the implementation of the second of our new algorithms in the computer algebra system Magma and report on new Conway polynomials that we have obtained with this implementation. We also discuss the challenges faced by the algorithm and compare its performance to that of the brute force algorithm.

4.1 Implementation in Magma

Some subtleties arise when implementing the algorithm of Section 3.3 in computer algebra systems such as Magma. In particular, when computing the Conway polynomial for GF(p^n), it is usually necessary to perform extensive arithmetic operations (taking roots, etc.) in the finite field GF(p^n).[2] When n has two or more distinct prime factors, Theorem 9 furnishes a very convenient representation for GF(p^n), since we can define GF(p^n) by adjoining a root z_0 of $r(z)$ to the prime field $\mathbb{Z}_p$. On the other hand, if n is a prime power $q_1^{e_1}$, the polynomial $r(z)$ in the algorithm is simply $\mathrm{C}_{p,n/q_1}(z)$. In this case, we first define GF(p^{n/q_1}) in Magma using this polynomial and then create a degree p extension of this field to obtain a representation for GF(p^n).

Another subtlety arises in the computation of polynomial GCD's in the first stage of the algorithm. Ironically, for values of n with three or more distinct prime factors, our algorithm often fails because there is not enough memory to store the huge polynomials $\mathrm{C}_{p,d_i}(z^{m_i/g})$ that occur when computing $r(z)$. If n has two prime factors, the following trick proves very useful when finding the polynomial GCD of $g(z) = \mathrm{C}_{p,d_1}(z^{m_1/g})$ and $h(z) = \mathrm{C}_{p,d_2}(z^{m_2/g})$. Assume that $n = q_1^{e_1} q_2^{e_2}$, where $q_1 < q_2$. Then the degree of $g(z)$ will be dramatically smaller than the degree of $h(z)$. Indeed, for moderate n, we can factor $g(z)$ directly using standard polynomial factoring algorithms. For each irreducible factor $r(z)$ of degree n that divides $g(z)$, we can indirectly test whether that factor also divides $h(z)$ as follows. Let z_0 be a root of $r(z)$ in GF(p^n). Since $r(z)$ divides $g(z)$, it must be the case that the minimal polynomial of $z_0^{m_1/g}$ is $\mathrm{C}_{p,d_1}(z)$. However, the minimal polynomial of $z_0^{m_2/g}$ equals $\mathrm{C}_{p,d_2}(z)$ if and only if $r(z)$ divides $h(z)$. Since large powers of finite field elements, as well as their minimal polynomials, can be efficiently computed, we can now compute $r(z)$ without ever storing or using $h(z)$, provided that we can compute and fully factor $g(z)$.

[2]This becomes quite awkward in GAP v3.4.4, since one cannot readily define or use finite fields whose Conway polynomials are not known.

Unfortunately, although this technique can be generalized to more than two polynomials, the exponents involved quickly become so large that even the polynomial $\mathrm{C}_{p,d_i}(z^{m_i/g})$ of lowest degree is too big to store.

Table 2 lists a number of new Conway polynomials. Each of these polynomials corresponds to a gap in the list of known Conway polynomials in version 2.3-1 of Magma. Appendix A contains the Magma routines used to compute the Conway polynomials in Table 2. The initial call to find $\mathrm{C}_{p,n}(x)$ is `conwaypol(p,n);`.

4.2 Comparison of Old and New Algorithms

As discussed earlier, the time consumed by our new algorithms to determine a particular Conway polynomial $\mathrm{C}_{p,n}$ depends critically on the quantity

$$g = \gcd_{1 \le i \le s} \left\{ \frac{\mathrm{M}_{p,n}}{\mathrm{M}_{p,n/q_i}} \right\}.$$

If g is reasonably small (say, eight digits or less), then $\mathrm{C}_{p,n}$ can be computed in a moderate amount of time. For example, on a Sun Ultra Sparc 30 workstation we computed $\mathrm{C}_{2,42}$ in only 59 seconds.[3] In this case, g was only 5419; nearly all of the computing time was spent calculating the polynomial GCD $r(z)$. In contrast, the brute force algorithm in GAP for computing Conway polynomials ran for days on $\mathrm{C}_{2,42}$ without completing.

Table 3 contrasts the running time required by our new algorithm with that required by the brute force search. We consider the computation of the Conway polynomials $\mathrm{C}_{2,n}$, for $40 \le n < 70$. For each value of n, we tabulate the quantities g, $c = \tau(M, \mathrm{M}_{p,n})$, $h = p^n/c$ (rounded to the nearest integer), and the amount of CPU time t needed by the Magma implementation of our algorithm to compute the Conway polynomial (when available). Recall that g is the number of field elements compatible with previously chosen polynomials. Since our algorithm must check each of these elements for primitivity, the time required by the algorithm increases at least linearly with g. The quantity c gives the number of compatible field elements that are also primitive. Thus, p^n/c gives a rough

[3]Times cited refer to total CPU time, as reported by Magma at the end of each session.

Table 2: Selected Conway polynomials.

p	n	$C_{p,n}(x)$
2	46	$x^{46}+x^{23}+x^{21}+x^{20}+x^{17}+x^{14}+1$
2	50	$x^{50}+x^{29}+x^{28}+x^{27}+x^{19}+x^{17}+x^{16}+x^{14}+x^{13}+x^{12}$ $x^{10}+x^{9}+x^{8}+x^{6}+x^{4}+x^{2}+1$
2	52	$x^{52}+x^{28}+x^{27}+x^{26}+x^{25}+x^{23}+x^{21}+x^{17}+x^{15}+x^{14}+$ $x^{10}+x^{7}+x^{4}+x+1$
2	56	$x^{56}+x^{33}+x^{30}+x^{26}+x^{22}+x^{19}+x^{14}+x^{13}+x^{11}+x^{9}$ $+x^{8}+x^{4}+x^{3}+x^{2}+1$
2	60	$x^{60}+x^{45}+x^{44}+x^{42}+x^{41}+x^{39}+x^{36}+x^{34}+x^{33}+x^{32}$ $+x^{30}+x^{26}+x^{25}+x^{22}+x^{19}+x^{17}+x^{12}+x^{8}+x^{5}+x^{4}+$ $x^{3}+x^{2}+1$
3	26	$x^{26}+x^{13}+2x^{12}+2x^{11}+2x^{10}+2x^{9}+2x^{8}+2x^{7}$ $+2x^{6}+x^{3}+2x^{2}+x+2$
3	28	$x^{28}+2x^{14}+x^{13}+x^{12}+2x^{11}+x^{10}+x^{9}+x^{8}$ $+2x^{6}+2x^{4}+x^{3}+2$
3	34	$x^{34}+x^{18}+2x^{17}+2x^{16}+2x^{14}+2x^{12}+x^{11}+2x^{9}$ $+x^{7}+2x^{6}+2x^{4}+2$
3	36	$x^{36}+x^{21}+2x^{20}+x^{17}+x^{16}+2x^{14}+2x^{13}+2x^{11}+2x^{10}$ $+x^{9}+2x^{8}+2x^{6}+2x^{5}+x^{3}+x^{2}+x+2$

Table 2: Selected Conway polynomials (continued).

p	n	$C_{p,n}(x)$
5	20	$x^{20} + 3x^{12} + 4x^{10} + 3x^9 + 2x^8 + 3x^6 + 4x^3 + x + 2$
5	22	$x^{22} + x^{12} + 3x^{11} + 4x^9 + 3x^8 + 2x^6 + 2x^5$ $+4x^3 + 3x^2 + 3x + 2$
5	24	$x^{24} + 2x^{16} + 4x^{15} + 4x^{13} + 2x^{12} + x^{11} + 3x^{10} + 4x^8$ $+2x^7 + 4x^6 + 2x^4 + 3x^3 + 3x^2 + x + 2$
7	20	$x^{20} + x^{12} + 6x^{11} + 2x^{10} + 5x^9 + 2x^8 + 3x^7$ $+x^6 + 3x^5 + 3x^3 + x + 3$
7	24	$x^{24} + 6x^{15} + 5x^{14} + 5x^{13} + x^{12} + 2x^{11} + x^{10} + 3x^8$ $+4x^7 + 5x^6 + 2x^5 + 2x^4 + 6x^3 + 4x^2 + 3x + 3$
11	14	$x^{14} + 2x^7 + 9x^6 + 6x^5 + 4x^4 + 8x^3 + 6x^2 + 10x + 2$
11	18	$x^{18} + 3x^{12} + 8x^{11} + 10x^{10} + 8x^9 + 3x^8 + 9x^7 + x^6$ $+3x^4 + 9x^3 + 8x^2 + 2x + 2$
13	14	$x^{14} + 4x^7 + 6x^5 + 11x^4 + 7x^3 + 10x^2 + 10x + 2$
13	18	$x^{18} + 10x^{11} + 4x^{10} + 11x^9 + 11x^8 + 9x^7 + 5x^6$ $+3x^5 + 5x^4 + 6x^3 + 9x + 2$
17	14	$x^{14} + x^8 + 11x^7 + x^6 + 8x^5 + 16x^4 + 13x^3 + 9x^2 + 3x + 3$

estimate of the number of elements that the brute force algorithm must check before it finds the first primitive, compatible one. Using Theorem 6, it is easy to calculate c given values for p, n, and g.

It is instructive to compare the relative magnitudes of g and p^n/c for different values of n. If n is a prime, then $g = \mathrm{M}_{p,n}/(p-1)$ and p^n/c is quite small. In this case, the brute force algorithm works quite well, since there are quite a few primitive, compatible polynomials in the search space, so the first one will be found quickly. In contrast, our algorithm performs horribly in this instance, since it must check all g compatible candidates to find the lexicographically smallest primitive one.

On the other hand, for composite n, the value of g tends to be small, often much smaller than p^n/c. For example, consider the cases $n \in \{40, 42, 44, 48, 50, 52, 54, 60, 66\}$ from Table 3 for dramatic differences that favor our algorithm over the brute force algorithm. In this case, our algorithm succeeds where the brute force algorithm fails. On the Sun Ultra Sparc 30 workstation, Conway polynomials can be computed in three days or less for values of g up to 10^8. Our algorithm is still viable, of course, for problems with nine-digit or ten-digit g's, but the computation will take proportionately longer (weeks or months, respectively).

5 Alternative Directions

Consider the algorithm described in Section 3.3, which finds $r(z)$ by finding the greatest common divisor of several sparse polynomials of large degree. A major deficiency in this algorithm is its inability to compute this polynomial GCD when p or n gets large. In particular, this stage of the algorithm is especially prone to failure when n has three or more prime factors. The version of the algorithm from Section 3.2 (which does computations with field elements rather than polynomials) addresses this problem.

Assuming that enough memory is available to compute the polynomial GCD $r(z)$, the major time expense incurred by the algorithm occurs when it checks all g of the gth roots of z_0 to find the primitive root whose minimal polynomial is lexicographically smallest.

Table 3: Comparison of the efficiency of two algorithms for computing $C_{p,n}$.

n	g	$c = \tau(M, \mathrm{M}_{p,n})$	$h = p^n/c$	CPU time t (sec.) (if available)
40	61681	61680	17826064	125
41	2199023255551	2198858730832	1	N/A
42	5419	5418	811747233	59
43	8796093022207	8774777333880	1	N/A
44	838861	836352	21034428	1947
45	14709241	14685300	2395890	34396
46	2796203	2796202	25165830	16336
47	140737488355327	140646443289600	1	N/A
48	65281	64512	4363141380	203
49	4432676798593	4432676798592	127	N/A
50	1016801	1012500	1111999907	2844
51	2454285751	2429105112	927007	N/A
52	13421773	13076544	344402896	38880
53	9007199254740991	9005653101120000	1	N/A
54	261633	261630	68854483467	1000
55	567767102431	566942112000	63549	N/A
56	15790321	15790320	4563403024	51350
57	39268347319	39267102096	3670125	N/A
58	178956971	175923744	1638382458	N/A
59	576460752303423487	576457548871463200	1	N/A
60	80581	79200	14557089704631	318

Table 3: Comparison of Algorithms' Efficiency (continued).

n	g	$c = \phi(N)g/N$	$h = p^n/c$
61	2305843009213693951	2305843009213693950	1
62	715827883	715827882	6442450950
63	60247241209	60246498816	153093909
64	4294967297	4288266240	4301678823
65	145295143558111	145295143558110	253921
66	1397419	1376496	53604933319703
67	147573952589676412927	147573951827644447920	1
68	3435973837	3407185152	86625144220
69	10052678938039	10052678938038	58720249

Because so many of the gth roots of z_0 are primitive, we can find a primitive root with a compatible minimal polynomial very quickly, by stopping at the first primitive root we find. The polynomial so obtained is not, in general, the Conway polynomial. However, it does have all the desirable algebraic properties of the Conway polynomial, namely primitivity and compatibility with previously chosen polynomials. Hence, for each p, one can quickly generate a large set of compatible polynomials to represent fields of characteristic p.

Indeed, one can *define* a new set of polynomials via the modified version of our algorithm. The only difficulty in postulating such a definition is that certain portions of our algorithm — specifically, taking roots in finite fields — rely on randomized subroutines; hence, the algorithm produces different polynomials each time it is executed. To obtain one standard set of polynomials, it is necessary to remove all randomness from the algorithm used to define these polynomials.

Appendix B contains the Magma code for this alternate algorithm that generates random sets of compatible polynomials for $p = 2$. The routine that generates these polynomials skips all values of n with three or more prime factors, since Magma tends to run out of memory when computing the polynomial GCD $r(z)$ for such n.

In summary, we have demonstrated two new algorithms for generating Conway polynomials, or more general sets of compatible polynomials, over finite fields. These algorithms are much more efficient for finding $\mathrm{C}_{p,n}$ than the brute force algorithm when the parameter g is small compared to the parameter p^n/c. We have also shown the practical significance of these new algorithms by generating numerous Conway polynomials that had not previously been identified.

References

[1] E. Bach and J. Shallit, *Algorithmic Number Theory*, The MIT Press, Cambridge, Massachusetts, 1996.

[2] The GAP Group, *GAP – Groups, Algorithms, and Programming, Version 4*, Lehrstuhl D für Mathematik, RWTH Aachen, Germany and School of Mathematical and Computational Sciences, U. St. Andrews, Scotland, 1997.

[3] J. Grabmeier and A. Scheerhorn, *Finite fields in AXIOM*, 1993. Online document available at `http://extweb.nag.co.uk/doc/TechRep/NP1513.html`.

[4] C. Jansen, K. Lux, R. Parker, and R. Wilson, *An Atlas of Brauer Characters*, Clarendon Press, Oxford, 1995.

[5] R. Lidl and H. Niederreiter, *Introduction to Finite Fields and Their Applications*, Cambridge University Press, Cambridge, 1994.

[6] W. Nickel, *Endliche Körper in dem gruppentheoretischen Programmsystem GAP*. Diploma thesis, RWTH Aachen, 1988.

[7] R. Parker, *Finite fields and Conway polynomials*, 1990. Attributed in [8] to a talk given at IBM Heidelberg Scientific Center.

[8] A. Scheerhorn, *Trace- and norm-compatible extensions of finite fields*, Applicable Algebra in Engineering, Communication and Computing, 3 (1992), pp. 199–209.

A Magma Implementation of Algorithm

Here is the Magma source code for our implementation of the algorithm of Section 3.3.

```
// File "conway": Magma function to compute Conway polynomials.
load "getroot";
load "tryall0";
// implements the "factor the smallest C_{p,d_i}(x^{m_i}) and check"
// technique to get the polynomial GCD q(z)

substitute := function(p,pr,x,exp)
// return polynomial p over polynomial ring pr with x replaced by x^exp

   list := Coefficients(p);
   answer := pr ! 0;

   j := 1;
   while (j le #list) do
     if (list[j] ne 0) then
       answer := answer + list[j]*x^((j-1)*exp);
     end if;
     j := j + 1;
   end while;

   return answer;
end function;


conwaypol := function(p,n)
// return the Conway Polynomial for GF(p^n) if possible

// check validity of arguments
  if (not IsPrime(p)) then
    print "Error: ",p," is not a prime!";
    return 0;
  end if;
  if (n le 0) then
    print "Error: ",n," is not positive!";
    return 0;
  end if;

  pr<x> := PolynomialRing( GF(p) );

  if (n eq 1) then // base case: GF(p)
```

```
    beta := 1;
    while (not IsPrimitive( GF(p) ! beta )) do
       beta := beta + 1;
    end while;
    return x - beta;
  end if;

  facs := Factorization(n);
  s := #facs;

  if (s eq 1) then // n is a prime power: special case
    print "prime power n";
    q := facs[1][1];
    d := n div facs[1][1];
    m := (p^n - 1) div (p^d - 1);
    g := m;
    if (not ExistsConwayPolynomial(p,d)) then
      //print "Finding polynomial with p=",p," d=",d," recursively.";
      cpd := Self(p,d);
    else
      cpd := ConwayPolynomial(p,d);
    end if;

// one more special case if d=1 (i.e., n itself is a prime)
    if (d gt 1) then
       gfpd<z> := ext< GF(p) | cpd >;
    else
       gfpd := GF(p);
       z := gfpd ! (GF(p) ! -Coefficients(cpd)[1]);
    end if;
    gfpn := ext< gfpd | q >;
    root, zeta := getanyroot(z, g, gfpn);
    print "root = ",root," and zeta = ",zeta;
    return tryall2(root,zeta,g,gfpn,p,n);
  end if;

  ii := 1; d:=[* *]; m:=[* *]; cpd:=[* *];
  while (ii le s) do
    d[ii] := n div facs[ii][1];
    m[ii] := (p^n - 1) div (p^d[ii] - 1);
    if (not ExistsConwayPolynomial(p,d[ii])) then
      //print "Finding polynomial with p=",p," d=",d[ii]," recursively.";
      cpd[ii] := Self(p,d[ii]);
    end if;
    cpd[ii] := ConwayPolynomial(p,d[ii]);
    print "ii=",ii,"; d[ii]=",d[ii],"; m[ii]=",m[ii],"; cpd[ii]=",cpd[ii];
    ii := ii + 1;
  end while;

// Calculate the integer and polynomial GCD's of interest.
  g := m[1]; ii:=2;
  while (ii le s) do
    g:=GCD(g,m[ii]);
    ii:=ii+1;
  end while;
  print "g=",g;

  q := substitute(cpd[1],pr,x,m[1] div g);
  plist := Factorization(q);
  print "Have to check ",#plist," polynomial factors.";

  foundGCD := false;
  ii := 1;
  while (ii le #plist) and (not foundGCD) do
    pcand := plist[ii][1];
    if (Degree(pcand) eq n) then
      gf<zz> := ext< GF(p) | pcand >;
      if (p2comp(MinimalPolynomial(zz^(m[1] div g)),cpd[1]) ne 0) then
        print "Bad factor!";
      end if;
      kk := 2; OK:=true;
      while (kk le s) and OK do
        if (p2comp(MinimalPolynomial(zz^(m[kk] div g)),cpd[kk]) ne 0)
             then OK:=false; end if;
        kk := kk + 1;
      end while;
      if (OK) then
         q:=pcand;
         foundGCD:=true;
      end if;
    end if;
    ii := ii+1;
  end while;

  print "OK, g=",g,"; and q(x) [really z] =",q;
  if (not IsIrreducible(q)) then  // consistency check
     print "Error: q(x) is reducible!!!";
     return 0;
  end if;
  if (Degree(q) ne n) then
     print "Error: q(x) does not have degree n!!";
     return 0;
  end if;

  gf<z> := ext< GF(p) | q >;
  root, zeta := getanyroot(z, g, gf);
  return tryall(root,zeta,g,gf,p,n);

end function;
// end of file "conway"
////////////////////////////////////////////////////////////////////////
// File "getroot": Magma functions to take roots of field elements.

getroot := function(a,r,fq)
// Returns an r'th root of element a in field fq (if possible)
// along with a primitive r'th root of unity.
// This function implements a version of Tonelli's algorithm
// as given in Chapter 7 of Bach and Shallit.
// Here, r must be prime; the function "getanyroot"
// later generalizes to non-prime r.

  q := #fq;
//  print "a =",a,"; r =",r,"; q =",q,"; fq =",fq;
  ii := 0;
  if ((q-1) mod r) ne 0 then
    print "Error: r does not divide q-1";
    return 0,0;
  elif (not IsPrime(r)) then
    print "Error: r is not a prime";
    return 0,0;
  end if;

  repeat
    h:=Random(fq);
    ii := ii + 1;
  until (h ne 0) and ((h^((q-1) div r)) ne 1);
//  print "Found h =",h,"after",ii,"iteration(s).";
//  print "";

  t := (q-1) div r;
  s := 1;
  while (t mod r) eq 0 do
```

```
    t := t div r;
    s := s + 1;
  end while;
  gg, alpha, beta := XGCD(t,r^s);
  if (gg ne 1) then
    print "Error: r^s and t were not relatively prime!";
    return 0,0;
  end if;
  gg, rprime, tprime := XGCD(r,t);

//  print "q-1 = r^s t, for q-1 =",q-1,"; r =",r,"; s =",s,"; t =",t;
//  print "r^(-1) modulo t is: ",rprime;
//  print "alpha *",t,"+ beta *",r^s,
//    "= 1  for alpha =",alpha,"and beta =",beta;

  ar := a^t;
  at := a^(r^s);
  g  := h^t;
  zeta := g^((q-1) div r);
  e  := 0;

  ii := 0;
  while (ii lt s) do
     jj := 0;
     while ((g^(e+jj*r^ii))*ar)^(r^(s-ii-1)) ne 1 do
       jj := jj + 1;
       if (jj eq r) then
          print "Error: could not find digit ",ii;
          return 0,0;
       end if;
     end while;
     e := e + jj*r^ii;
     ii := ii+1;
   end while;

  if (e mod r ne 0) then
    print "Error: final value of e is not divisible by r";
    print "  i.e., a is NOT an r'th power in the field!";
    return 0;
  end if;
  br := g^(-(e div r));
  bt := at^rprime;
  b  := (br^alpha)*(bt^beta);
  return b,zeta;
```

```
end function;

getanyroot := function(a,r,fq)
// "a" is a perfect "r"th power in the field "fq"
// return an "r"th root of "a" in "fq" along with a primitive
//   "r"th root of unity in "fq".
  factors := Factorization(r);

  pe := PrimitiveElement(fq);
  zeta := pe^((#fq-1) div r);
  root := a;
  ii:=1;
  while (ii le #factors) do
    jj:=1;
    while (jj le factors[ii][2]) do
      root1, zeta1:= getroot(root,factors[ii][1],fq);
      root := root1;
      jj := jj + 1;
    end while;
    ii:=ii+1;
  end while;

  return root, zeta;
end function;

// end of file "getroot"
////////////////////////////////////////////////////////////////////////
// File "tryall0": routines to check all g g'th roots of z
// and pick out the best primitive one.

p2comp := function(p1,p2)
 // uses ordering in "Brauer character" book
  c1 := Coefficients(p1);
  c2 := Coefficients(p2);
  if (#c1 ne #c2) then
     print "p2comp warning: polynomials are of unequal degrees!";
     return 0;
  end if;

  ii := #c1; sign := +1;
  while (ii gt 0) and (c1[ii] eq c2[ii]) do
    ii := ii-1;
    sign := -sign;
  end while;
  if (ii eq 0) then return 0;  // equal polynomials
  end if;

  a1:=IntegerRing() ! (sign*c1[ii]);
  a2:=IntegerRing() ! (sign*c2[ii]);
  if (a1 gt a2) then return 1;
  else return -1;
  end if;
end function;

tryall := function(root,zeta,r,fq,p,n)

  b := root;
  q := #fq;

  best := MinimalPolynomial(b);
  print "b =",b;
  print "Minimal polynomial: ",best;
  isprim := IsPrimitive(best);
  print "Initially, isprim = ",isprim;

  ii:=1;
  while ii lt r and (not isprim) do
    b := b * zeta;
    best:=MinimalPolynomial(b);
    isprim:=IsPrimitive(best);
    ii := ii + 1;
  end while;

  if not isprim then
    print "Error: No polynomial was primitive!!!";
    return 0;
  end if;
  print "First primitive one:",best;


  while ii lt r do
    b := b * zeta;
      tp:=MinimalPolynomial(b);
      if (p2comp(tp,best) eq -1) then
        if IsPrimitive(tp) then best:=tp;
        end if;
      end if;
    ii := ii + 1;
    if (ii mod 1000 eq 0) then
      print "ii is now:",ii;
    end if;
  end while;

  print "Best one was:",best;
  return best;
end function;


tryall2 := function(root,zeta,r,fq,prime,dd)

  b := root;
  q := #fq;

  best := MinimalPolynomial(b,GF(prime));
  print "b =",b;
  print "Minimal polynomial: ",best;
  cc := Coefficients(best); print "cc = ",cc;
  if (#cc lt (dd+1)) then isprim:=false;
  else isprim := IsPrimitive(best);
  end if;
  print "Initially, isprim = ",isprim;

  ii:=1;
  while ii lt r and (not isprim) do
    b := b * zeta;
    best:=MinimalPolynomial(b,GF(prime));
    if (#Coefficients(best) lt (dd+1)) then isprim:=false;
    else isprim:=IsPrimitive(best);
    end if;
    ii := ii + 1;
  end while;

  if not isprim then
    print "Error: No polynomial was primitive!!!";
    return 0;
  end if;
  print "First primitive one:",best;

  while ii lt r do
    b := b * zeta;
    tp:=MinimalPolynomial(b,GF(prime));
    if (#Coefficients(tp) eq #Coefficients(best)) then
     if (p2comp(tp,best) eq -1) then
      if IsPrimitive(tp) then best:=tp;
      end if;
     end if;
    end if;
    ii := ii + 1;
    if (ii mod 1000 eq 0) then
      print "ii is now:",ii;
    end if;
  end while;

  print "Best one was:",best;
  return best;
end function;

// end of file "tryall0"
```

B Modified Algorithm

Here is the modified algorithm of Section 5.

```
// File "compatible_sets":
// Magma code to generate a random set of mutually compatible
// polynomials that are not Conway polynomials.

load "getroot";
load "tryall4";
// implement the "factor the smallest C_{p,d_i}(x^{m_i}) and check"
// trick to get the polynomial GCD q(z)
// Take the first primitive, compatible polynomial you can get

substitute := function(p,pr,x,exp)
// return polynomial p over polynomial ring pr with x replaced by x^exp

   list := Coefficients(p);
   answer := pr ! 0;

   j := 1;
   while (j le #list) do
     if (list[j] ne 0) then
       answer := answer + list[j]*x^((j-1)*exp);
     end if;
     j := j + 1;
   end while;

   return answer;
end function;

compatiblepol := function(p,n,table)
// check validity of arguments
  if (not IsPrime(p)) then
    print "Error: ",p," is not a prime!";
    return 0;
  end if;
  if (n le 0) then
    print "Error: ",n," is not positive!";
    return 0;
  end if;

  pr<x> := PolynomialRing( GF(p) );
  if (n eq 1) then // base case: GF(p)
    beta := 1;
    while (not IsPrimitive( GF(p) ! beta )) do
       beta := beta + 1;
    end while;
    return x - beta;
  end if;

  facs := Factorization(n);
  s := #facs;

  if (s eq 1) then // n is a prime power: special case
    //print "prime power n";
    q := facs[1][1];
    d := n div facs[1][1];
    m := (p^n - 1) div (p^d - 1);
    g := m;
    cpd := table[d];

// one more special case if d=1 (i.e., n itself is a prime)
    if (d gt 1) then
       gfpd<z> := ext< GF(p) | cpd >;
    else
       gfpd := GF(p);
       z := gfpd ! (GF(p) ! -Coefficients(cpd)[1]);
    end if;
    gfpn := ext< gfpd | q >;
    root, zeta := getanyroot(z, g, gfpn);
    //print "root = ",root," and zeta = ",zeta;
    return tryall2(root,zeta,g,gfpn,p,n);
  end if;

  ii := 1; d:=[* *]; m:=[* *]; cpd:=[* *];
  while (ii le s) do
    d[ii] := n div facs[ii][1];
    m[ii] := (p^n - 1) div (p^d[ii] - 1);
    cpd[ii] := table[d[ii]];
    // print "ii=",ii,"; d[ii]=",d[ii],";
    //   m[ii]=",m[ii],"; cpd[ii]=",cpd[ii];
    ii := ii + 1;
  end while;

// Calculate the integer and polynomial GCD's of interest.
  g := m[1]; ii:=2;
```

```
while (ii le s) do
  g:=GCD(g,m[ii]);
  ii:=ii+1;
end while;
//print "g=",g;

q := substitute(cpd[1],pr,x,m[1] div g);
plist := Factorization(q);
//print "Have to check ",#plist," polynomial factors.";

foundGCD := false;
ii := 1;
while (ii le #plist) and (not foundGCD) do
  pcand := plist[ii][1];
  if (Degree(pcand) eq n) then
    gf<zz> := ext< GF(p) | pcand >;
    if (p2comp(MinimalPolynomial(zz^(m[1] div g)),cpd[1]) ne 0) then
      print "Bad factor!";
    end if;
    kk := 2; OK:=true;
    while (kk le s) and OK do
      mp := MinimalPolynomial(zz^(m[kk] div g));
      if (Degree(mp) ne Degree(cpd[kk])) then
          OK:=false;
      elif (p2comp(mp,cpd[kk]) ne 0) then
          OK:=false;
      end if;
      kk := kk + 1;
    end while;
    if (OK) then
       q:=pcand;
       foundGCD:=true;
    end if;
  end if;
  ii := ii+1;
end while;

//print "OK, g=",g,"; and q(x) [really z] =",q;
if (not IsIrreducible(q)) then  // check answer
   print "error: q(x) is reducible!!!";
   return 0;
end if;
if (Degree(q) ne n) then
   print "error: q(x) does not have degree n!!";
```

```
      return 0;
   end if;

   gf<z> := ext< GF(p) | q >;
   root, zeta := getanyroot(z, g, gf);
   return tryall(root,zeta,g,gf);

end function;


poltable := [* *];
p := 2;
u := 1;
print "Table of random compatible polynomials for p = ",p;
while (u le 200) do
   fu := Factorization(u);
   if (#fu le 2) then
      /* don't bother for n's with 3 or more prime factors */
      /* (Magma usually runs out of memory for those n.) */
      poltable[u] := compatiblepol(p,u,poltable);
      print "poltable[",u,"] := ",poltable[u];
   else poltable[u] := 0;
   end if;
   u := u+1;
end while;
quit;

// end of file "compatible_sets"
///////////////////////////////////////////////////////////
// File "tryall4": modified version of "tryall0" that
// stops at the first primitive g'th root of z.

p2comp := function(p1,p2)
 // uses ordering in "Brauer character" book
   c1 := Coefficients(p1);
   c2 := Coefficients(p2);
   if (#c1 ne #c2) then
      print "p2comp warning: polynomials are of unequal degrees!";
      return 0;
   end if;

   ii := #c1; sign := +1;
   while (ii gt 0) and (c1[ii] eq c2[ii]) do
      ii := ii-1;
```

```
    sign := -sign;
  end while;

  if (ii eq 0) then return 0;  // equal polynomials
  end if;

  a1:=IntegerRing() ! (sign*c1[ii]);
  a2:=IntegerRing() ! (sign*c2[ii]);
  if (a1 gt a2) then return 1;
  else return -1;
  end if;
end function;

tryall := function(root,zeta,r,fq)

  b := root;
  q := #fq;

  best := MinimalPolynomial(b);
//  print "b =",b;
//  print "Minimal polynomial: ",best;
  isprim := IsPrimitive(best);
//  print "Initially, isprim = ",isprim;

  ii:=1;
  while ii lt r and (not isprim) do
    b := b * zeta;
    best:=MinimalPolynomial(b);
    isprim:=IsPrimitive(best);
    ii := ii + 1;
  end while;

  if not isprim then
    print "Error: No polynomial was primitive!!!";
    return 0;
  end if;
//  print "First primitive one:",best;
  return best;
end function;


tryall2 := function(root,zeta,r,fq,prime,dd)

  b := root;
  q := #fq;

  best := MinimalPolynomial(b,GF(prime));
//  print "b =",b;
//  print "Minimal polynomial: ",best;
  cc := Coefficients(best); //print "cc = ",cc;
  if (#cc lt (dd+1)) then isprim:=false;
  else isprim := IsPrimitive(best);
  end if;
//  print "Initially, isprim = ",isprim;

  ii:=1;
  while ii lt r and (not isprim) do
    b := b * zeta;
    best:=MinimalPolynomial(b,GF(prime));
    if (#Coefficients(best) lt (dd+1)) then isprim:=false;
    else isprim:=IsPrimitive(best);
    end if;
    ii := ii + 1;
  end while;

  if not isprim then
    print "Error: No polynomial was primitive!!!";
    return 0;
  end if;
//  print "First primitive one:",best;
  return best;

end function;
```