CARLETON UNIVERSITY

SCHOOL OF
MATHEMATICS AND STATISTICS

HONOURS PROJECT

TITLE: Factoring Polynomials over Finite Fields

AUTHOR: Melissa Scott

SUPERVISOR: Daniel Panario

DATE: May 2019

Honours Project
Factoring Polynomials over Finite Fields

Melissa S. Scott

May 2019

Contents

Abstract

We discuss factoring polynomials over finite fields using Elimination of Repeated Factors (or Squarefree Factorization), Distinct Degree Factorization, and Equal Degree Factorization. Of these three algorithms, the second, Distinct Degree Factorization is the least efficient. It has a bottleneck where we compute the gcd of two polynomials. In order to decrease overall computational complexity, we focus on optimizing this area. We do this in two ways. Firstly by reducing the number of such calculations we require and secondly by optimizing how we compute one of the polynomials. Since these two methods are distinct we can benefit from both improvements.

1 Introduction

Factoring a polynomial f over a finite field means finding all irreducible polynomials in the field that divide f. These irreducible polynomials are factors of f and the product of them is the entirety of f. This factorization is of interest in algebraic coding theory, computational number theory, computer algebra, and cryptography among other areas [9].

In this project we discuss polynomial factorization using a 3 step method. Step 1 has two algorithms that can be used: Elimination of Repeated Factors and Squarefree Factorization. Both of these algorithms take a polynomial $f \in \mathbb{F}_q[X]$ and output its squarefree part(s). The algorithm for step 2 is Distinct Degree Factorization which takes a squarefree polynomial $g \in \mathbb{F}_q[X]$ and splits it into polynomials that each are the product of irreducible factors of g with a certain degree. Step 3 is Equal Degree Factorizationtakes a product of irreducible polynomials with the same degree and outputs these irreducible factors.

Once we outline the details of these algorithms, we observe an area in step 2 that has a large computational cost. This bottleneck comes from computing the gcd of $X^{q^i} - X$, $1 \leq i \leq n$, and the polynomial we are factoring. In order to decrease the cost of these operations we first look at minimizing the number of them we need to perform. After this we examine ways to calculate $X^{q^i} - X$ (or more specifically X^{q^i}) modulo a polynomial more efficiently. Lastly, we illustrate an algorithm for step 2 that uses these optimizations. This gives us an overall improved factorization method.

2 Background

Definition 2.1. *A* **finite field** $\mathbb{F}_q$ *is a ring* $(\mathbb{F}_q, +, \cdot)$ *with* $q < \infty$ *elements, such that* $(\mathbb{F}_q^*, \cdot)$ *forms a commutative group. We call* q *the* **order** *of* $\mathbb{F}_q$ *[3].*

Definition 2.2. *Suppose* F *and* K *are fields such that* $K \subseteq F$ *then* K *is a* **subfield** *of* F *and* F *is an* **extension field** *of* K*.*

Definition 2.3. *Let* $\mathbb{F}_q$ *be a finite field. If there exists* $m \in \mathbb{Z}^+$ *such that* $m\alpha = 0$ *for all* $\alpha \in \mathbb{F}_q$*, then the smallest such* m *is the* **characteristic** *of* $\mathbb{F}_q$*, and* $\mathbb{F}_q$ *has characteristic* m *[3].*

Lemma 2.4. *If* $\mathbb{F}_q$ *is a finite field with order* $q = p^n$ *for prime* p*, then* $\mathbb{F}_q$ *has characteristic* p*.*

Definition 2.5. *A* **polynomial over** $\mathbb{F}_q$ *is an expression of the form* $f(X) = \sum_{i=0}^{n} a_i X^i$ *where* $a_i \in \mathbb{F}_q$*. We denote this as* $f \in \mathbb{F}_q[X]$*. If* $a_n \neq 0$*, then* a_n *is the* **leading coefficient** *of* f *and* f *has* **degree** n*. When* $f(X) = 0$*, we set* $\deg(f(X)) = -\infty$ *by convention [3].*

Definition 2.6. *Let* f *be a polynomial in* $\mathbb{F}_q[X]$*. Then* f *is* **monic** *if it has leading coefficient* 1*. If* f *has positive degree and for all* $a, b \in \mathbb{F}_q[X]$ *such that* $f = ab$ *we have* a *or* b *is a constant polynomial then* f *is* **irreducible** *[3].*

Theorem 2.7. *Any polynomial* $f \in \mathbb{F}_q[X]$ *of positive degree can be written as* $f = a f_1^{a_1} \cdots f_k^{e_k}$ *where* $a \in \mathbb{F}_q$*,* $f_1, \ldots, f_k \in \mathbb{F}_q[X]$ *are distinct irreducible polynomials and* $e_1, \ldots, e_k \in \mathbb{Z}^+$ *[3].*

Definition 2.8. *Let* f *be a polynomial in* $\mathbb{F}_q[X]$*. Then* f *is* **monic** *if it has leading coefficient* 1*. If* f *has positive degree and for all* $a, b \in \mathbb{F}_q[X]$ *such that* $f = ab$ *we have* a *or* b *is a constant polynomial then* f *is* **irreducible** *[3].*

Definition 2.9. *Let* f *be a polynomial over a field* K*, with* $\deg(f) > 0$*. Suppose* F *is a field extension of* K*, then* f **splits** *in* F *if there exist* $\alpha_1, \ldots, \alpha_n \in F$ *and* $a \in K$ *such that* $f(X) = a(X - \alpha_1) \cdots (X - \alpha_n)$*. The field* F *is the* **splitting field** *of* K *if* f *splits in* F *and* $F = K(\alpha_1, \ldots, \alpha_n)$*.*

Lemma 2.10. *Let* $\mathbb{F}_q$ *be a finite field with characteristic* p *then* $(x + y)^p = x^p + y^p$ *for all* $x, y \in \mathbb{F}_q$ *[3].*

Proof. Let $\mathbb{F}_q$ be a finite field with characteristic p then

$$(x+y)^p = \binom{p}{0} x^p + \binom{p}{1} x^{p-1} y + \cdots + \binom{p}{p-1} x y^{p-1} + \binom{p}{p} y^p$$

by binomial expansion. Since $p \mid \binom{p}{i}$ for $i \in 1, \ldots, p-1$ we have

$$(x+y)^p = x^p + y^p$$

as $\mathbb{F}_q$ has characteristic p. □

Lemma 2.11. *$\mathbb{F}_{q^n}$ is a subfield of $\mathbb{F}_{q^m}$ if and only if $n \mid m$.*

Definition 2.12. *The* **gcd** *of $f, g \in \mathbb{F}_q[X]$ is the monic polynomial $d \in \mathbb{F}_q[X]$ of the largest degree such that $d \mid f$ and $d \mid g$.*

Lemma 2.13. *Let $f, g \in \mathbb{F}_q[X]$ then $\gcd(f, g) = \gcd(f - g, g)$.*

Proof. Let $d = \gcd(f, g)$ then there exits $a, b \in \mathbb{F}_q[X]$ such that $f = ad$ and $g = bd$. So $f - g = ad - bd = (a - b)d$, and we have $d \mid f - g$ as well as $d \mid g$. Assume for sake of contradiction that $h = \gcd(f - g, g)$ with $\deg(h) > \deg(d)$. Then $h \mid f - g$ and $h \mid g$ which implies $h \mid f$. This is a contradiction with $d = \gcd(f, g)$. Therefore $d = \gcd(f - g, g) = \gcd(f, g)$. □

Theorem 2.14. *Let $f, g \in \mathbb{F}_q[X]$ then $\gcd(f, g) = \gcd(f \pmod g, g)$.*

Proof. Using Lemma 2.13 repeatedly yields the result. □

Theorem 2.15. *Let $f \in \mathbb{F}_q[X]$ be an irreducible polynomial of degree n. Then $f(X) \mid X^{q^i} - X$ if and only if $n \mid i$. The product of all irreducible polynomials in $\mathbb{F}_q[X]$ of degree dividing i is given by $X^{q^i} - X$*

Proof. Let $f \in \mathbb{F}_q[X]$ be a irreducible polynomial of degree n. Suppose $f(X)$ divides $X^{q^i} - X$ and α is a root of f in the splitting field over $\mathbb{F}_q$. Then we have $f(\alpha) = 0$ which gives $\alpha^{q^i} - \alpha = 0$. Thus $\alpha \in \mathbb{F}_{q^i}$ and we have

$$i = [\mathbb{F}_{q^i} : \mathbb{F}_q] = [\mathbb{F}_{q^i} : \mathbb{F}_q(\alpha)][\mathbb{F}_q(\alpha) : \mathbb{F}_q] = [\mathbb{F}_{q^i} : \mathbb{F}_q(\alpha)]n$$

which implies $n \mid i$. Conversely, suppose $n \mid i$, then $\mathbb{F}_{q^n}$ is a subfield of $\mathbb{F}_{q^i}$. Let α be a root of f then $[\mathbb{F}_q(\alpha) : \mathbb{F}_q] = n$ implies $\alpha \in \mathbb{F}_{q^n} \subset \mathbb{F}_{q^i}$. This gives $\alpha^{q^i} = \alpha$, i.e. $\alpha^{q^i} - \alpha = 0$, and so α is a root of $X^{q^i} - X$. Therefore $f(X) \mid X^{q^i} - X$. □

Theorem 2.16. ***(Chinese Remainder Theorem)***
Let $m_1, \ldots, m_k \in \mathbb{Z}^+$ be relatively prime and $a_1, \ldots, a_k \in \mathbb{Z}$. Then there exits $N \in \mathbb{Z}$ such that

$$\begin{aligned} N &\equiv a_1 \pmod{m_1} \\ &\vdots \\ N &\equiv a_k \pmod{m_k} \end{aligned}$$

and N is unique modulo $m_1 \cdots m_k$.

Proof. Let $M_i = \dfrac{m_1 \cdots m_k}{m_i}$, then we have $\gcd(M_i, m_i) = 1$ since $m_1, \ldots, m_k$ are relatively prime. Thus M_i has an inverse modulo m_i, say x_i. Suppose

$$N \equiv \sum_{i=1}^{k} a_i M_i x_i \pmod{m_1 \ldots m_k},$$

then

$$\begin{aligned} N &\equiv \sum_{i=1}^{k} a_i M_i x_i \pmod{m_j} \\ &\equiv a_j M_j x_j \pmod{m_j} && \text{since } m_j \mid M_i \text{ for } i \neq j \\ &\equiv a_j \pmod{m_j} && \text{since } M_j x_j \equiv 1 \pmod{m_j} \end{aligned}$$

and N is a solution. Assume N and M are 2 solutions modulo $m_1 \dots m_k$. Then $M \equiv N \pmod{m_i}$ for $i = 1, \dots, k$ which means $m_i \mid N - M$. As $m_1, \dots, m_k$ are relatively prime, this gives $m_1 \dots m_k \mid N - M$ or equivalently, $N \equiv M \pmod{m_1 \cdots m_k}$. Therefore the solution is unique moduluo $m_1 \cdots m_k$. □

3 Factoring Polynomials over Finite Fields

In order to factor a polynomial $f \in \mathbb{F}_q[X]$ we use the 3 part method outlined below [6].

Step 1 Find the Squarefree part(s) of f using Elimination of Repeated Factors (Algorithm 1) or Squarefree Factorization (Algorithm 2)

Step 2 Split the squarefree part(s) of f into factors made up of irreducible polynomials of the same degree using Distinct Degree Factorization (Algorithm 3)

Step 3 Split the polynomials found in Step 2 into their irreducible factors using Equal Degree Factorization (Algorithm 6)

We observe that of these 3 algorithms, the Distinct Degree Factorization Algorithm is the most time consuming, so in Section 4 we focus on finding improvements for it. The bottleneck in the Distinct Degree Factorization Algorithm can be largely attributed to calculating the gcd of various polynomials. Therefore, we examine ways to speed up or eliminate these gcd computations.

3.1 Finding the Squarefree Part(s) of a Polynomial

One approach to removing duplicate factors from a polynomial $f \in \mathbb{F}_q[X]$ is to find a polynomial with the same factors as f that does not have any duplicates. Once this squarefree polynomial is factored we can look at which factors were duplicates if necessary.

Algorithm 1: Elimination of Repeated Factors (ERF)

Input: $f \in \mathbb{F}_q[X]$ monic of degree n
$p = \text{char}(\mathbb{F}_q)$
Output: squarefree part of f

```
u = gcd(f, f')
if u = 1 then
    return f
end
v = f/u
w = u / gcd(u, v^n)
z = w^{1/p}
return v·ERF(z)
```

Example 3.1. *We compute **ERF**(f) where* $f(X) = X^5(X+2)^3(X^2+X+2)$ *is a polynomial over* $\mathbb{F}_3$. *Taking the derivative we have*

$$f'(X) = 2X^4(X+2)^3(X^2+X+2) + X^5(X+2)^3(2X+1).$$

This gives $u = X^4(X+2)^3$, *which isn't* 1. *So we have*

$$v = X(X^2+X+2)$$

and

$$w(X) = \frac{X^4(X+2)^3}{X^4} = (X+2)^3.$$

Thus we must apply the algorithm to $z(X) = X+2$. *Since* $z'(X) = 1$, *this returns* z. *Therefore our original call to the algorithm returns*

$$v(X)z(X) = X(X^2+x+2)(X+2)$$

which is the squarefree part of f.

Theorem 3.2. *Algorithm 1 returns the squarefree part of* $f \in \mathbb{F}_q[X]$.

Proof. Let $f = \prod_{i=1}^{r} f_i^{e_i}$ for f_i irreducible with $f_i \neq f_j$ when $i \neq j$, then we have

$$f' = \sum_{j=1}^{r} e_j f^{e_j - 1} f_j' \prod_{i \neq j} f_i^{e_i}.$$

This gives

$$u = \gcd(f, f') = \prod_{i=1, p \nmid e_i}^{r} f_i^{e_i - 1} \prod_{i=1, p \mid e_i}^{r} f_i^{e_i}.$$

since $\gcd(f_i, f_i') = 1$ for irreducible f_i and if $p \mid e_i$ the term with the f_i' is zero. If $u = 1$ we must have $f' = 1$ which gives $e_i = 1$ for all i and f is already squarefree so we are done. Otherwise,

$$v = f/u = \prod_{i=1, p \nmid e_i}^{r} f_i \quad \text{implies} \quad \gcd(u, v^n) = \prod_{i=1, p \nmid e_i}^{r} f_i^{e_i - 1},$$

which implies

$$w = \frac{u}{\gcd(u, v^n)} = \prod_{i=1, p \mid e_i}^{r} f_i^{e_i},$$

and we can take the p^{th} root of w to get z. By induction we can assume that the algorithm works for polynomials with smaller e_i and thus $\mathbf{ERF}(z)$ gives the product of f_is such that $p \mid e_i$. Multiplying this by v we have

$$v\mathbf{ERF}(z) = \prod_{i=1, p \nmid e_i}^{r} f_i \prod_{i=1, p \mid e_i}^{r} f_i = \prod_{i=1}^{r} f_i$$

which is the squarefree part of f. Therefore, for any $f \in \mathbb{F}_q[X]$ Algorithm 1 outputs the squarefree part of f. □

The second method (Algorithm 2) to remove duplicate factors from a polynomial $f \in \mathbb{F}_q[X]$ is to find g_i the product of all factors that are duplicated i times. This gives us the additional benefit of knowing the number of duplicates and having (potentially not irreducible) factors of f.

Algorithm 2: Squarefree Factorization (SFF)

Input: $f \in \mathbb{F}_q[X]$ monic, $p = \text{char}(\mathbb{F}_q)$
Output: list of pairs (g_i, i) where $f = g_1 g_2^2 \dots g_n^n$

$i = 1$
output = [ ]
$g = f'$
if $g \neq 0$ **then**
  $h := \gcd(f, g)$
  $r := f/h$
  while $r \neq 1$ **do**
    $s := \gcd(r, h)$
    $t := r/s$
    add (t, i) to output
    $i := i + 1$
    $r := s$
    $h := h/s$
  end
  if $h \neq 1$ **then**
    $h := h^{1/p}$
    for (g_j, j) *in* ***SFF****(h)* **do**
      if (g_k, k) *such that* $k = j * p$ *is in output* **then**
        change (g_k, k) to $(g_k g_j, k)$
      else
        add $(g_j,\ jp)$ to output
      end
    end
  end
else
  $f = f^{1/p}$
  for (g_j, j) *in* ***SFF****(f)* **do**
    if (g_k, k) *such that* $k = j * p$ *is in output* **then**
      change (g_k, k) to $(g_k g_j, k)$
    else
      add $(g_j,\ jp)$ to output
    end
  end
end
return output

Example 3.3. *We compute* ***SFF****(f) where* $f(X) = X^5(X+2)^3(X^2+X+2)$ *is a polynomial over* $\mathbb{F}_3$. *The algorithm begins with* $i = 1$, *utput* = *[ ], and* $g = f'$. *We note*

$$f'(X) = 2X^4(X+2)^3(X^2+X+2) + X^5(X+2)^3(2X+1)$$

is not zero. This gives $h := X^4(X+2)^3$ *and* $r := X(X^2+X+2)$ *which*

is not 1. *We remain in the while loop for* 5 *iterations, resulting in output*= $[(X^2+X+2,1),(1,2),(1,3),(1,4),(X,5)]$ *and* $h=(X+2)^3$. *We then call* ***SFF****(X+2) which returns* $[(X+2,1)]$. *So in output, we change* $(1,3)$ *to* $(X+2,3)$ *before returning it.*

Theorem 3.4. *Algorithm 2 returns a list of pairs* (g_i,i) *when given an input polynomial such that* $f=g_1g_2^2\dots g_n^n$.

Proof. We first prove that the k^{th} iteration of the while loop ends with

$$i:=k+1, \qquad r:=\prod_{i=k+1,p\nmid i}^{n} g_i,$$

$$t:=\begin{cases} g_k & p\nmid k,\\ 1 & p\mid k,\end{cases} \quad \text{and} \quad h:=\prod_{i=k+1,p\nmid i}^{n} g_i^{i-(k+1)}\prod_{i=1,p\mid i}^{n} g_i^i.$$

Clearly, the loop ends with $i=k+1$ as i is incremented by one on each iteration. The first loop starts with

$$h:=\gcd(f,f')=\prod_{i=1,p\nmid i}^{n} g_i^{i-1}\prod_{i=1,p\mid i}^{n} g_i^i \quad \text{and} \quad r:=f/h=\prod_{i=1,p\nmid i}^{n} g_i$$

as shown in the previous algorithm. This gives

$$s:=\gcd(r,h)=\prod_{i=2,p\nmid i}^{n} g_i \quad \text{and} \quad t:=\begin{cases} g_1 & p\nmid 1\\ 1 & p\mid 1\end{cases}$$

which shows

$$r:=s=\prod_{i=2,p\nmid i}^{n} g_i \quad \text{and} \quad h:=h/s=\prod_{i=2,p\nmid i}^{n} g_i^{i-2}\prod_{i=1,p\mid i}^{n} g_i^i$$

as claimed for $k=1$. We assume by induction that our claim holds for k. Then if there is another iteration of the while loop we have

$$s:=\gcd(r,h)=\prod_{i=k+2,p\nmid i}^{n} g_i, \qquad t:=r/s=\begin{cases} g_{k+1} & p\nmid k+1,\\ 1 & p\mid k+1,\end{cases}$$

$$r:=s=\prod_{i=k+2,p\nmid i}^{n} g_i, \quad \text{and} \quad h:=h/s=\prod_{i=k+2,p\nmid i}^{n} g_i^{i-(k+1)}\prod_{i=1,p\mid i}^{n} g_i^i.$$

Thus our claim is true for any $k\geq 1$. With this claim it is clear that (g_k,k) is included in the output for $p\nmid k$. When the loop exits we have

$$h=\prod_{i=k+2,p\nmid i}^{n} g_i^{i-(k+1)}\prod_{i=1,p\mid i}^{n} g_i^i$$

which we can take the p^{th} root of and repeat the algorithm to add (g_k,k) with $p\mid k$ to the output. This repetition terminates as there are finitely many factors of p in any k. □

3.2 Distinct Degree Factorization

The next step in our 3 part factorization method is Distinct Degree Factorization. This splits f into factors g_i which are the product of all irreducible factors of f with degree i. The gcd calculation used here gives the desired result because of Theorem 2.15.

Algorithm 3: Distinct Degree Factorization (DDF)

Input: $f \in \mathbb{F}_q[X]$ monic squarefree of degree n
Output: $g_1, \dots g_n \in \mathbb{F}_q[X]$ such that each g_i is the product of all irreducible factors of f with degree i

$h_i = x$
$f_i = f$
for $i = 1, \dots, \lfloor n/2 \rfloor$ **do**
  $h_i = h_i^q \pmod f$
  Record that $g_i = \gcd(h_i - x, f_i)$
  $f_i = f_i / g_i$
end
if $\deg(f_i) > \lfloor n/2 \rfloor$ **then**
  $g_k = f_i$
end
All other g_i are assumed to be 1
return $g_1, \dots, g_n$

Example 3.5. *We compute* ***DDF****(f) where* $f(X) = X(X+2)(X^2+X+2)$ *is a polynomial over* $\mathbb{F}_3$*. The for loop at* $i = 1$ *gives*

$$g_1 = \gcd(X^3 \pmod{f(X)} - X, f(X)) = X(X+2)$$

and $f_1(X) = f(X)/g_1(X) = X^2 + X + 2$*. For* $i = 2$ *we have*

$$g_2 = \gcd(\left(X^3\right)^2 \pmod{f(X)} - X, f_1(X)) = X^2 + X + 2$$

and $f_2(X) = f_1(X)/g_2(X) = 1$*. Since* $\deg(f_2) = 0 \leq \lfloor 4/2 \rfloor$ *we assume* $g_3 = 1$ *and* $g_4 = 1$*. Therefore the algorithm returns* $X(X+2), X^2+X+2, 1, 1$*.*

Theorem 3.6. *Algorithm 3 outputs* $g_1, \dots g_n \in \mathbb{F}_q[X]$ *such that each* g_i *is the product of all irreducible factors of* f *with degree* i*.*

Proof.

$$\begin{aligned} g_1 &= \gcd(h_1 - x, f_1) \\ &= \gcd(x^q - x \pmod f, f) \\ &= \gcd(x^q - x, f) \\ &= \text{product of irreducible polynomials of degree 1 that divide } f. \end{aligned}$$

Assume $f_i = \frac{f}{g_1 \cdots g_{i-1}}$, g_i is the product of irreducible polynomials of degree i that divide f and $h_i = x^{q^i} \pmod f$. Then by induction

$$\begin{aligned}
g_{i+1} &= \gcd(h_{i+1} - x, f_{i+1}) \\
&= \gcd(h_i^q \pmod f - x, f_{i+1}) \\
&= \gcd\left(\left(x^{q^i}\right)^q \pmod f - x, f_i/g_i\right) \\
&= \gcd\left(x^{q^{i+1}} - x, \frac{f}{g_1 \cdots g_i}\right) \\
&= \text{product of irreducible polynomials of degree } d \mid i+1 \\
&\quad \text{that divide } \frac{f}{g_1 \cdots g_i} \\
&= \text{product of irreducible polynomials of degree } i+1 \text{ that divide } f,
\end{aligned}$$

for $i+1 \leq \lfloor n/2 \rfloor$. For $i+1 > \lfloor n/2 \rfloor$ we have $k = \deg(f_i)$. Note $k \notin \{1, \ldots, \lfloor n/2 \rfloor\}$ since all irreducible polynomials of these degrees have been removed. Thus $k = 0$ or $k > \lfloor n/2 \rfloor$. If $k = 0$ then $f_i = 1$ since f is monic and we have no more irreducible factors left. If $k > \lfloor n/2 \rfloor$, we must have 1 irreducible factor of degree k left and so we set $g_k = f_i$. We note that we can not have 2 or more factors since that would require $\deg(f_i)$ to be greater than or equal to $2(\lfloor n/2 \rfloor + 1)$ which is larger than n. □

There are three strategies for when to stop the loop in Algorithm 3. The one we give is *half-degree* because it improves the algorithm by stopping at $\lfloor n/2 \rfloor$ instead of n as in the *basic* strategy. The third and best strategy is called *early abort*. It stops the loop when $\deg(f_i)$ is less than $2i$ since the remaining f_i must be irreducible in this case [1].

This algorithm is slowed down by many gcd computations, most of which are equal to 1. To reduce these computations we can combine them, then split them up later if necessary. For example instead of computing $\gcd(a, f)$ and $\gcd(b, f)$ we compute $\gcd(ab, f)$. If $\gcd(ab, f) = 1$ we do not need a or b to find factors and we saved a calculation. On the rare occasion that $\gcd(ab, f) \neq 1$ we can then look closer to find the factors that a and/or b reveal. This strategy is called *baby-step giant-step* [6] and was presented in an Algorithm by von zur Gathen and Shoup [10] and Kaltofen and Shoup [4].

Algorithm 4: Coarse Distinct Degree Factorization (CDDF)

Input: $f \in \mathbb{F}_q[X]$ monic squarefree of degree n
Output: $H_1, \ldots, H_n$ where H_j is the product of all monic irreducible factors of f with degree $i \in \{s_{j-1}+1, \ldots, s_j\}$

$g = f$
for $j = 1, \ldots, n$ **do**
  $g_j = \prod_{s_{j-1}+1 \le i \le s_j} x^{q^i} - x \quad$ where $s_j = 2j^2$
  $H_j = \gcd(q_j \pmod g), g)$
  $g = g/H_j$
end
return $H_1, \ldots, H_n$

The value of s_j in the above algorithm was presented by von zur Gathen and Gerhard [8] when factoring polynomials over $\mathbb{F}_2$. Kaltofen and Shoup [4] use intervals of the same size, which is not as effective. Starting with smaller intervals is better as we expect there to be more irreducible factors of lower degree than of higher degree.

Algorithm 5: Fine Distinct Degree Factorization (FDDF)

Input: H_j the product of all monic irreducible factors of $f \in \mathbb{F}_q[X]$ with degree $s_{j-1}+1, \ldots,$ or s_j
Output: $h_{s_{j-1}+1}, \ldots, h_{s_j}$ where h_i all monic irreducible factors of f with degree i

$h = H_j$
for $i = s_{j-1}+1, \ldots, s_j$ **do**
  $h_i = \gcd(h, x^{q^i} - x \pmod h)$
  $h = h/h_i$
end
return $h_{s_{j-1}+1}, \ldots, h_{s_j}$

Theorem 3.7. *Applying Algorithm 5 to the results of Algorithm 4 gives $h_1, \ldots, h_n$ where each h_i is the product of irreduible factors of degree i that divide f.*

Proof. The proof follows from Theorem 3.6. □

The table below allows us to see the benefits of this method. Here $M(n)$ is the cost of multiplying two polynomials of degree less than or equal to n and by Schönhage and Strassen we can assume $M(n) \in O(n \log n \log\log n)$ [8].

Algorithm	Operations over $\mathbb{F}_q$
DDF	$O(M(n)(n \log q + n \log n))$
Coarse and Fine DDF (k same size intervals)	$O(M(n)(n \log q + k \log n))$
Coarse and Fine DDF (increasing size intervals)	$O(M(n)(n \log q))$

Table 1: Cost of various Distinct Degree Factorization algorithms [8]

3.3 Equal Degree Factorization

The third and final step of the 3 part factorization method is Equal Degree Factorization. This takes the products of irreducible factors with degree i found in Distinct Degree Factorization and factors them into those irreducible polynomials. Of the 3 algorithms in the factorization method, this is the only one that is not deterministic.Unfortunately, known deterministic algorithms for Equal Degree Factorization do not run in polynomial time in the degree n and $\log(q)$, where q is the order of the field [1].

Algorithm 6: Equal Degree Factorization (EDF) [2]

Input: $d \in \mathbb{N}$
$f \in \mathbb{F}_q[X]$ a squarefree product of $r \geq 2$ irreducible factors each of degree d where q is odd
confidence parameter ϵ

Output: list of monic irreducible factors or a failure message

```
factors = [f]
k = 1
t = 2⌈log r²/ε⌉
while k ≤ t do
    randomly choose h ∈ F_q[X] with deg(h) < n
    g = gcd(h, f)
    if g = 1 then
        g = h^((q^d − 1)/2) − 1 (mod f)
    end
    for u ∈ factors with deg(u) > d do
        if gcd(g, u) ≠ 1 and gcd(g, u) ≠ u then
            remove u from factors
            factors = factors ∪{gcd(g, u), u/ gcd(g, u)}
        end
    end
    if size(factors) = r then
        return factors
    end
    k = k + 1
end
return failure message
```

Algorithm 6 is due to Cantor and Zassenhaus [2]. In it we consider q odd, though variants exist for even q. We now look at ways to decrease the runtime of this 3 part method by focusing on Distinct Degree Factorization.

4 Using Composition of Functions to Optimize Factorization

The most expensive part of the Distinct Degree Factorization algorithm is computing $\gcd(X^{q^i} - X, f(X))$. Previously we looked at reducing the number of these calculations, but now we focus on making them less expensive. We do this by using modular composition to reduce the time complexity of computing $X^{q^i} - X$ or more specifically X^{q^i}. Once we have looked at both improvements we describe an algorithm for Distinct Degree Factorization that combines them.

We first discuss algorithms for evaluating polynomials at a set of points. This is used to evaluate a composition $f(g(X))$ at many points while only knowing

the polynomials f and g. These evaluations can then be used to interpolate the composition itself.

Algorithm 7: Multivariate Multi-point Evaluation (MME) [5]

Input: $f \in \mathbb{F}_p[X_0, \ldots, X_{m-1}]$ with $\deg_{X_i}(f) < d$ for $0 \leq i \leq m-1$
$\alpha_0, \ldots, \alpha_{N-1} \in \mathbb{F}_p^m$
p prime

Output: $f(\alpha_0), \ldots, f(\alpha_{N-1})$

1 Compute the reduction g of f modulo $X_j^p - X_j$ for $j = 0, \ldots, N-1$
2 Use an FFT algorithm [7] to compute $g(\beta) = f(\beta)$ for each $\beta \in \mathbb{F}_p^m$
3 **return** $f(\alpha_0), \ldots, f(\alpha_{N-1})$

We need the following lemma for the proof of the next theorem.

Lemma 4.1. *The product of the primes less than or equal to* $16\log(N)$ *is greater than* N *for any integer* $N \geq 2$ *[5].*

We now discuss another algorithm which uses the lemma above to evaluate a polynomial at a series of points. This algorithm will use multiple iterations of Algorithm 7 with smaller primes to produce the same result. In order to work over fields of smaller order we first treat our polynomial and its evaluation points as if they were over $\mathbb{Z}$. To do this we lift elements in $\mathbb{F}_p$ to the integers $0, \ldots, p-1$ and use integer operations on them from then on. Once everything is over the integers we can then do operations modulo various smaller primes. Evaluating the polynomial at each point modulo these primes will give us enough information to solve for the polynomial evaluation over the original field using the Chinese Remainder Theorem (2.16).

Algorithm 8: Multi-Modular [5]

Input: $f \in \mathbb{F}_p[X_0, \ldots, X_{m-1}]$ with $\deg_{X_i}(f) < d$ for $0 \leq i \leq m-1$
$\alpha_0, \ldots, \alpha_{N-1} \in (\mathbb{Z}/p\mathbb{Z})^m$
p prime
$t \in \mathbb{Z}^+$ which is the number of rounds

Output: $f(\alpha_0), \ldots, f(\alpha_{N-1})$

Let $\tilde{f} \in \mathbb{Z}[X_0, \ldots, X_{m-1}]$ be constructed from f by replacing coefficients with their lift in $\{0, 1, \ldots, p-1\}$
for $i = 0, \ldots, N-1$ **do**
  Let $\tilde{\alpha_i} \in \mathbb{Z}^m$ be constructed from α_i by replacing coordinates with their lift in $\{0, 1, \ldots, p-1\}$
end
let $l = 16 \log(d^m (p-1)^{md})$
Let $p_1, \ldots, p_k$ be the primes less than or equal to l
for $h = 1, \ldots, k$ **do**
  Let $f_h \in \mathbb{F}_{p_h}[X_0, \ldots, X_{m-1}]$ be given by $f_h = \tilde{f} \pmod{p_h}$
  Let $\alpha_{h,i} \in \mathbb{F}_{p_h}^m$ be given by $\alpha_{h,i} = \tilde{\alpha_i} \pmod{p_h}$
end
if $t = 1$ **then**
  Compute $f_h(\alpha_{h,i})$ for $i = 0, \ldots, N-1$ using $\mathbf{MME}(f_h, \alpha_{h,0}, \ldots, \alpha_{h_N-1}, p_h)$
else
  Compute $f_h(\alpha_{h,i})$ for $i = 0, \ldots, N-1$ using $\mathbf{Multi-Modular}(f_h, \alpha_{h,0}, \ldots, \alpha_{h,N-1}, p_h, t-1)$
end
for $i = 0, \ldots, N-1$ **do**
  Find $z_i \in \{0, \ldots, (p_1 \ldots p_k) - 1\}$ such that $z_i = f_h(\alpha_{h,i}) \pmod{p_h}$ for all $h = 1, \ldots, k$, using Theorem 2.16
end
return $z_0 \pmod{p}, \ldots, z_{N-1} \pmod{p}$

Theorem 4.2. *Algorithm 8 returns* $f(\alpha_0) \ldots, f(\alpha_{N-1})$.

Proof. When $t = 1$ we have $f_h(\alpha_{h,i})$, for $0 \leq i \leq N-1$, by Algorithm 7. Since $p_1, \ldots, p_k$ are relatively prime z_i is unique modulo $p_1 \cdots p_k$ by the Chinese Remainder Theorem (2.16). Let

$$\tilde{f}(X_0, \ldots, X_{m-1}) = \sum_{i_0=0}^{d-1} \cdots \sum_{i_{m-1}=0}^{d-1} f_{i_0,\ldots,i_{m-1}} X_0^{i_0} \cdots X_{m-1}^{i_{m-1}}$$

where $f_{i_0,\ldots,i_{m-1}} \in \mathbb{F}_p$ could be zero, and $\tilde{\alpha_i} = (a_0, \ldots, a_{m-1})$ for $a_j \in \{0, 1, \ldots, p-1\}$. We note that $z_i = \tilde{f}(\tilde{\alpha_i})$ since,

$$\begin{aligned}
0 &\leq \tilde{f}(\tilde{\alpha_i}) \\
&= \sum_{i_0=0}^{d-1} \cdots \sum_{i_{m-1}=0}^{d-1} f_{i_0,\ldots,i_{m-1}} a_0^{i_0} \cdots a_{m-1}^{i_{m-1}} \\
&\leq \sum_{i_0=0}^{d-1} \cdots \sum_{i_{m-1}=0}^{d-1} (p-1)(p-1)^{i_0} \cdots (p-1)^{i_{m-1}} \\
&\leq \sum_{i_0=0}^{d-1} \cdots \sum_{i_{m-1}=0}^{d-1} (p-1) \left((p-1)^{d-1}\right)^m \\
&\leq \sum_{i_0=0}^{d-1} \cdots \sum_{i_{m-1}=0}^{d-1} (p-1)^{dm} \\
&= d^m (p-1)^{dm} \\
&< p_1 \cdots p_k \qquad \text{by Lemma 4.1}
\end{aligned}$$

and z_i is unique modulo $p_1 \cdots p_k$. Therefore

$$z_i \pmod p = \tilde{f}(\tilde{\alpha_i}) \pmod p = f(\alpha_i)$$

and the algorithm holds. When $t > 1$ we have $f_h(\alpha_{h,i})$ and the result follows by induction. □

The algorithm above is only valid for polynomials in $\mathbb{F}_p$ where p is prime. Therefore we must expand it to the case when $\mathbb{F}_q$ where q is a prime power.

Algorithm 9: Multi-Modular Extension [5]

Input: $f \in \mathbb{F}_q[X_0, \ldots, X_{m-1}]$ with individual degrees at most $d-1$ and
$\mathbb{F}_q \simeq \mathbb{F}_p[Z]/g(Z)$
$\alpha_0, \ldots, \alpha_{N-1} \in \mathbb{F}_q^m$
p, n where $q = p^n$ and p prime
$t \in \mathbb{Z}^+$ which is the number of rounds

Output: $f(\alpha_0), \ldots, f(\alpha_{N-1})$

Let $M = d^m(n(p-1))^{(d-1)m+1} + 1$
Let $r = M^{(n-1)dm+1}$
Let $\tilde{f} \in \mathbb{Z}[Z][X_0, \ldots, X_{m-1}]$ be constructed from f by replacing coefficients with their lift in $\{0, 1, \ldots, p-1\}$
for $i = 1, \ldots, N-1$ **do**
  Let $\tilde{\alpha_i} \in \mathbb{Z}[Z]^m$ be constructed from α_i by replacing coordinates with their lift in $\{0, 1, \ldots, p-1\}$
end
Let $\bar{f} \in (\mathbb{Z}/r\mathbb{Z})[X_0, \ldots, X_{m-1}]$ be the reduction of $\tilde{f}$ modulo $Z - M$ **for** $i = 1, \ldots, N-1$ **do**
  Let $\overline{\alpha_i}$ be the reduction of $\tilde{\alpha_i}$ modulo $Z - M$
end
Compute $\beta_i = \bar{f}(\bar{\alpha})$ for $i = 0, \ldots, N-1$ using $\mathbf{Multi-Modular}(\bar{f}, \overline{\alpha_0}, \ldots, \overline{\alpha_{N-1}}, r, t)$ for some $t \in \mathbb{Z}^+$ **for** $i = 0, \ldots, N-1$ **do**
  Find $Q_i[Z] \in \mathbb{Z}[Z]$ of degree at most $(n-1)dm$ with coefficients in $\{0, \ldots, M-1\}$ such that $Q_i[Z] \pmod r = \beta_i$
  Let $\bar{Q_i}$ be the reduction of Q_i modulo p and $g(Z)$
end
return $Q_0[Z], \ldots, Q_{N-1}[Z]$

We now state a definition required for the next algorithm.

Definition 4.3. *The map* $\psi_{h,l} : \mathbb{F}_q[X_0, X_1, \ldots, X_{m-1}] \to \mathbb{F}_q[Y_{0,0}, \ldots, Y_{m-1,l-1}]$ *is defined by*

$$\psi_{h,l}(X_i^a) = Y_{i,0}^{a_0} Y_{i,1}^{a_1} \ldots Y_{i,l-1}^{a_{l-1}}$$

extended linearly, where a_j *are taken from* $a = \sum_{j \geq 0} a_j h^j$.

The next algorithm uses Algorithm 9 to evaluate a polynomial at a series of points. Then we evaluate another polynomial at these points. This gives us the result of evaluating the composition of these two polynomials at the original set of points. These evaluations can then be used to interpolate the composition of the polynomials using Lagrange Interpolation Formula.

Algorithm 10: Modular Composition [5]

Input: $f \in \mathbb{F}_q[X_0, \ldots, X_{m-1}]$ with individual degrees at most $d-1$
$g_0(X), \ldots, g_{m-1}(X), h(X) \in \mathbb{F}_q[X]$ with degree at most $N-1$
$d, m, N \in \mathbb{N}$
$d_0 \in \mathbb{Z}$ such that $2 \leq d_0 \leq d$

Output: $f(g_0(X), \ldots, g_{m-1}(X)) \pmod{h(X)}$

Let $l = \lceil \log_{d_0}(d) \rceil$
Let $\bar{f} = \psi_{d_0,l}(f)$
Let $g_{i,j}(X) = g_i(X)^{d_0^j}$ for $i = 0, \ldots, m-1$ and $j = 0, \ldots, l-1$
Let $N' = Nmld_0$
Select N' distinct elements of $\mathbb{F}_q$ say $\beta_0, \ldots, \beta_{N'-1}$
Let $p, n \in \mathbb{Z}$ such that $p^n = q$
for $i = 0, \ldots, m-1$ **do**
  for $j = 0, \ldots, l-1$ **do**
    Compute $\alpha_{i,j,k} = g_{i,j}(\beta_k)$ for $k = 0, \ldots, N'-1$ using **Multi-ModularExtension**($g_{i,j}$, $\beta_0, \ldots, \beta_{N'-1}$,$p$,$n$, t) for some $t \in \mathbb{Z}^+$
  end
end
Compute $\bar{f}(\alpha_{0,0,k}, \ldots, \alpha_{m-1,l-1,k})$ for $k = 0, \ldots, N'-1$ using **Multi-ModularExtension**($\bar{f}$,$\alpha_{0,0,k}, \ldots, \alpha_{m-1,l-1,k}$, p,n,t) for some $t \in \mathbb{Z}^+$
Interpolate to recover $\bar{f}(g_{0,0}(X), \ldots, g_{m-1,l-1}(X))$
return $\bar{f}(g_{0,0}(X), \ldots, g_{m-1,l-1}(X)) \pmod{h(X)}$

Theorem 4.4. *Algorithm 10 returns* $f(g_0(X), \ldots, g_{m-1}(X)) \pmod{h(X)}$.

Proof. Let

$$f(X_0, \ldots, X_{m-1}) = \sum_{i_0=0}^{d-1} \cdots \sum_{i_{m-1}=0}^{d-1} f_{i_0,\ldots,i_{m-1}} X_0^{i_0} \cdots X_{m-1}^{i_{m-1}}$$

for $f_{i_0,\ldots,i_{m-1}} \in \mathbb{F}_q$. Suppose $i_j = \sum_{k \geq 0} i_{j,k} d_0^k$ for $i_{j,k} \in \mathbb{Z}^+$. Then $i_{j,k} = 0$ for all $k \geq l$. Indeed if $i_{j,k} \neq 0$ for some $k \geq l$ then

$$d - 1 \geq i_j \geq d_0^j \geq d_0^l = d_0^{\lceil \log_{d_0}(d) \rceil} \geq d_0^{\log_{d_0}(d)} = d$$

which is a contradiction. We have $\bar{f} = \psi_{d_0,l}(f)$ which gives

$$\begin{aligned}\bar{f}(X_0, \ldots, X_{m-1}) &= \sum_{i_0=0}^{d-1} \cdots \sum_{i_{m-1}=0}^{d-1} f_{i_0,\ldots,i_{m-1}} \psi_{d_0,l}(X_0^{i_0}) \cdots \psi_{d_0,l}(X_{m-1}^{i_{m-1}}) \\ &= \sum_{i_0=0}^{d-1} \cdots \sum_{i_{m-1}=0}^{d-1} f_{i_0,\ldots,i_{m-1}} Y_{0,0}^{i_{0,0}} \cdots Y_{0,l-1}^{i_{0,l-1}} \cdots Y_{m-1,0}^{i_{m-1,0}} \cdots Y_{m-1,l-1}^{i_{m-1,l-1}}\end{aligned}$$

for $i_{j,k}$ as defined above. Thus

$$\begin{aligned}
&\bar{f}(g_{0,0}(X), \ldots, g_{0,l-1}(X), \ldots, g_{m-1,0}(X), \ldots, g_{m-1,l-1}(X)) \\
&\quad = \bar{f}(g_0(X)^{d_0^0}, \ldots, g_0(X)^{d_0^{l-1}}, \ldots, g_{m-1}(X)^{d_0^0}, \ldots, g_{m-1}(X)^{d_0^{l-1}}) \\
&\quad = \sum_{i_0=0}^{d-1} \cdots \sum_{i_{m-1}=0}^{d-1} f_{i_0,\ldots,i_{m-1}} \prod_{k=0}^{m-1} g_k(X)^{i_{k,0} d_0^0 \cdots g_k(X)^{i_{k,l-1} d_0^{l-1}}} \\
&\quad = \sum_{i_0=0}^{d-1} \cdots \sum_{i_{m-1}=0}^{d-1} f_{i_0,\ldots,i_{m-1}} \prod_{k=0}^{m-1} g_k(X)^{i_{k,0} d_0^0 + \cdots + i_{k,l-1} d_0^{l-1}} \\
&\quad = \sum_{i_0=0}^{d-1} \cdots \sum_{i_{m-1}=0}^{d-1} f_{i_0,\ldots,i_{m-1}} \prod_{k=0}^{m-1} g_k(X)^{i_k} \\
&\quad = f(g_0(X), \ldots, g_{m-1}(X))
\end{aligned}$$

and the algorithm returns $\bar{f} \pmod{h} = f(g_0, \ldots, g_{m-1}) \pmod{h}$ as claimed. □

Now that we have an algorithm for computing the composition of polynomials, we can use it to calculate polynomials of the form $X^{q^i} - X$. Polynomials of this form are required for Distinct Degree Factorization, so it is beneficial to find algorithms of lower complexity that can be used to calculate them. We now look at an algorithm that computes the product of multiple polynomials of this form.

Algorithm 11: Splitting Polynomial [5]

Input: A monic, squarefree polynomial $f(X) \in \mathbb{F}_q[X]$
$X^q \in \mathbb{F}_q[X]$
$m \in \mathbb{Z}^+$

Output: $s(X) = (X^q - X)^{a_1}(X^{q^2} - X)^{a_2} \dots (X^{q^m} - X)^{a_m} \pmod{f(X)}$
where a_i are powers of q

1 Let $n = \deg(f(X))$ and $k = m - \lfloor\sqrt{m}\rfloor^2 \leq 2\sqrt{m}$

2 Compute $X^{q^i} \pmod{f(X)}$ for $i = 0, 1, 2, \dots, \lfloor\sqrt{m}\rfloor - 1$

3 Compute $X^{q^{j\lfloor\sqrt{m}\rfloor}} \pmod{f(X)}$ for $j = 1, 2, \dots, \lfloor\sqrt{m}\rfloor$

4 Compute $X^{q^{m-i}} \pmod{f(X)}$ for $i = k-1, k-2, \dots, 1, 0$

5 Form $Q_0 = \prod_{i=0}^{k-1} (X^{q^{m-i}} - X) \pmod{f(X)}$

6 Form $P(Z) = \prod_{i=0}^{\lfloor\sqrt{m}\rfloor - 1} (Z - X^{q^i}) \pmod{f(X)}$

7 Evaluate $P\left(X^{q^{j\lfloor\sqrt{m}\rfloor}}\right)$ for $j = 1, \dots, \lfloor\sqrt{m}\rfloor$ using
ModularComposition$(P, X^{q^{j\lfloor\sqrt{m}\rfloor}}, f(X), \lfloor\sqrt{m}\rfloor + 1, 1, q^{j\lfloor\sqrt{m}\rfloor} + 1, d_0)$
for $2 \leq d_0 \leq d$

8 Take the product of polynomials above to form
$Q_1(X) = \prod_{j=1}^{\lfloor\sqrt{m}\rfloor} \prod_{i=0}^{\lfloor\sqrt{m}\rfloor - 1} (X^{q^{j\lfloor\sqrt{m}\rfloor}} - X^{q^i}) \pmod{f(X)}$

9 **return** $Q_0(X)Q_1(X)$

Theorem 4.5. *Algorithm 11 returns*

$$s(X) = (X^q - X)^{a_1}(X^{q^2} - X)^{a_2} \dots (X^{q^m} - X)^{a_m} \pmod{f(X)}$$

where a_i are powers of q.

Proof. Let

$$a_i = \begin{cases} q^{i \pmod{\lfloor\sqrt{m}\rfloor}} & 1 \leq i \leq \lfloor\sqrt{m}\rfloor^2, \\ q^0 & \lfloor\sqrt{m}\rfloor^2 + 1 \leq i \leq m, \end{cases}$$

then

$$
\begin{aligned}
&Q_0(X)Q_1(X) \\
&= \left(\prod_{i=0}^{k-1}(X^{q^{m-i}} - X)\right)\left(\prod_{j=1}^{\lfloor\sqrt{m}\rfloor}\prod_{i=0}^{\lfloor\sqrt{m}\rfloor-1}(X^{q^{j\lfloor\sqrt{m}\rfloor}} - X^{q^i})\right) \pmod{f(X)} \\
&= \left(\prod_{i=0}^{k-1}(X^{q^{m-i}} - X)\right)\left(\prod_{j=1}^{\lfloor\sqrt{m}\rfloor}\prod_{i=0}^{\lfloor\sqrt{m}\rfloor-1}(X^{q^{j\lfloor\sqrt{m}\rfloor-i}} - X)^{q^i}\right) \pmod{f(X)} \\
&= \left(\prod_{l=\lfloor\sqrt{m}\rfloor^2+1}^{m}(X^{q^l} - X)^{a_i}\right)\left(\prod_{l=1}^{\lfloor\sqrt{m}\rfloor}(X^{q^l} - X)^{a_i}\right) \pmod{f(X)} \\
&= \prod_{l=1}^{m}(X^{q^l} - X)^{a_i} \pmod{f(X)}.
\end{aligned}
$$

□

The product produced in this algorithm can be used in a Distinct Degree Factorization algorithm [5] that is similar to Algorithms 4 and 5. The powers do not affect the result of the gcd calculation since we have taken a squarefree polynomial.

Algorithm 12: Distinct Degree Factorization using Composition (DDFC)

Input: squarefree polynomial $f \in \mathbb{F}_q[X]$
$S = \{s, \ldots, s+k\}$ is a range which contains all degrees of irreducible factors of f
$X^q \in \mathbb{F}_q[X]$

Output: $g_s, \ldots g_{s+k} \in \mathbb{F}_q[X]$ such that each g_i is the product of all irreducible factors of f with degree i

if $|S| = 1$ **then**
  return f
end
let $m = s + \lfloor k/2 \rfloor$ be the midpoint of S
Compute $s(X) = \textbf{SplittingPolynomial}(f, X^q, m)$
Compute $d = \gcd(s, f)$
if $d = f$ **then**
  Let $S = \{x \in S \mid x \leq m\}$
  return $\textbf{DDFC}(f, S, X^q)$, $g_{m+1} = 1, \ldots g_n = 1$
end
if $\deg(d) = 1$ **then**
  Let $S = \{x \in S \mid x > m\}$
  return $g_1 = 1, \ldots, g_m = 1, \textbf{DDFC}(f, S, X^q)$
end
Let $f_{\text{lower}} = d(X)$ and $f_{\text{upper}} = f(X)/d(X)$
Let $S_{\text{lower}} = \{x \in S \mid x \leq m\}$ and $S_{\text{upper}} = \{x \in S \mid x > m\}$
return $\textbf{DDFC}(f_{\text{lower}}, S_{\text{lower}}, X^q)$, $\textbf{DDFC}(f_{\text{upper}}, S_{\text{upper}}, X^q)$

We note that this algorithm uses a different method than Algorithms 4 and 5. This can be seen as a binary search for each degree of irreducible. However this is not ideal, since we expect there to be more irreducible factors of lower degrees. Applying the increasing intervals from Algorithm 4 should give a more efficient algorithm. Further work can be also be done to reduce repeated work when computing the splitting polynomial for multiple values of m.

Theorem 4.6. *When a valid S is given to Algorithm 12, it returns $g_s, \ldots, g_m, g_{m+1}, \ldots, g_{s+k}$.*

Proof. When $|S| = 1$, i.e. $S = \{s\}$ then f only contains irreducible factors of one degree s and so $g_s = f$. So, the algorithm returns g_s in this case. Assume the algorithm holds for $|S| \leq k$. Suppose $|S| = k + 1$ then

$$s(X) = \prod_{i=1}^{m} (X^{q^i} - X)^{a_i} \pmod{f(X)}$$

for some a_i which are powers of q. Thus d is the product of irreducible factors of f with degrees $1, \ldots, m$ by Theorems 2.14 and 2.15. If $d = f$ we know that all irreducible factors of f have degree $s, \ldots, m$. This gives us

$$g_{m+1} = 1, \ldots, g_{s+k} = 1$$

and by induction

$$g_s, \ldots, g_m = \mathbf{DDFC}(f, \{x \in S \mid x \leq m\}, X^q)$$

as $|\{x \in S \mid x \leq m\}| \leq k$. Therefore the algorithm returns

$$g_s, \ldots, g_m, g_{m+1}, \ldots, g_{s+k},$$

as desired. If d is a constant, we know that all irreducible factors of f have degree $m, \ldots, s+k$. This gives us $g_s = 1, \ldots, g_m = 1$, and by induction

$$g_{m+1}, \ldots, g_{s+k} = \mathbf{DDFC}(f, \{x \in S \mid x > m\}, X^q)$$

as $|\{x \in S \mid x > m\}| \leq k$. Therefore the algorithm returns

$$g_s, \ldots, g_m, g_{m+1}, \ldots, g_{s+k}$$

as desired. If $d \neq f$ and d is not constant, then we have irreducible factors in both $1, \ldots, m$ and $m+1, \ldots, n$. By induction

$$g_s, \ldots, g_m = \mathbf{DDFC}(d), \{x \in S \mid x \leq m\}, X^q)$$

as $|\{x \in S \mid x \leq m\}| \leq k$ and d only has irreducible factors of degree less than m. Similarly, by induction

$$g_{m+1}, \ldots, g_{s+k} = \mathbf{DDFC}(f/d, \{x \in S \mid x > m\}, X^q)$$

and so the algorithm returns $g_s, \ldots, g_m, g_{m+1}, \ldots, g_{s+k}$ as desired. □

Observe that if we start with $S = \{1, \ldots, n\}$ where $n = \deg(f)$ then the above algorithm returns $g_1, \ldots, g_n$.

Using Algorithm 12 we obtain an algorithm for factoring a random polynomial of degree n over $\mathbb{F}_q$ that requires $O(n^{1.5+o(1)} \log^{1+o(1)} q + n^{1+o(1)} \log^{2+o(1)} q)$ bit operations. If $\log q$ is less than n, then this is asymptotically faster than the best previous algorithms [5]. For comparison, other algorithms require $(n^{2+o(1)} + n^{1+o(1)} \log q) log^{1+o(1)} q$ bit operations [10] and $n^{1.815} log^{2+o(1)} q$ bit operations [4].

5 Conclusion

The 3 step method for factoring a polynomial f over a finite field consists of:

1. Finding the squarefree part(s) of f using Elimination of Repeated Factors or Squarefree Factorization
2. Splitting f into factors that contain irreducible factors of the same degree using Distinct Degree Factorization
3. Split each of factors from 2 into irreducible polynomials using Equal Degree Factorization

We describe algorithms for each of these 3 steps, and give multiple versions of Distinct Degree Factorization. We give multiple versions of Distinct Degree Factorization since it is the bottleneck of factorization and we want to improve it so that our overall method is more efficient. Calculating $\gcd(X^{q^i} \pmod f, f)$ is what slows this algorithm down, so we focus on two ways to improve this computation. The first way is by minimizing the number of these calculations that we require. The second way is by decreasing the computational complexity of computing X^{q^i}. We do this using modular composition. Since these two approaches are distinct we can use them both at the same time to create a faster algorithm. Further work can be to implement these algorithms on a computer in an efficient way. Once we have implementations, we can perform tests to look at under what circumstances each version of Distinct Degree Factorization performs the best. It would be interesting to see how the degree of the polynomial and the distribution of its factors effect how each algorithm performs. If we had multiple polynomials in the same finite field that we want to factor at the same time, there could be ways to share the computational cost between them. For example the steps in computing X^{q^i}, before we take any modulus, is a calculation we could do only once for a set of polynomials.

References

[1] Philippe Flajolet, Xavier Gourdon, and Daniel Panario. The complete analysis of a polynomial factorization algorithm over finite fields, 2001.

[2] David G. Cantor and Hans Zassenhaus. A new algorithm for factoring polynomials over finite fields. *Mathematics of Computation*, 36:587–592, 04 1981.

[3] Sololmon W. Golomb and Guang Gong. *Signal Design for Good Correlation*. Cambridge University Press, 2005.

[4] Erich Kaltofen and Victor Shoup. Subquadratic-time factoring of polynomials over finite fields. *Math. Comp*, pages 398–406, 1998.

[5] Kiran S. Kedlaya and Christopher Umans. Fast polynomial factorization and modular composition. *SIAM Journal on Computing*, 40:1767–1802, 2011.

[6] Prof. Daniel Panario. Factoring polynomials over finite fields.

[7] Joachim von zur Gathen and Jürgen Gerhard. *Modern Computer Algebra*. Cambridge University, 1999.

[8] Joachim von zur Gathen and Jürgen Gerhard. Polynomial factorization over $\mathbb{F}_2$, 2002.

[9] Joachim von zur Gathen and Daniel Panario. Factoring polynomials over finite fields: A survey. *Symbolic Computation*, 2001.

[10] Joachim von zur Gathen and Victor Shoup. Computing frobenius maps and factoring polynomials. *Comput. Complexity*, 2:187–224, 1992.