Rotations and Translations of Number Field Sieve Polynomials

Jason E. Gower*

CERIAS and Department of Mathematics, Purdue University
West Lafayette, IN 47907-2067, USA, jgower@math.purdue.edu

Abstract. We present an algorithm that finds polynomials with many roots modulo many primes by rotating candidate Number Field Sieve polynomials using the Chinese Remainder Theorem. We also present an algorithm that finds a polynomial with small coefficients among all integral translations of X of a given polynomial in $\mathbb{Z}[X]$. These algorithms can be used to produce promising candidate Number Field Sieve polynomials.

1 Introduction

The Number Field Sieve (NFS) [1] is the fastest (asymptotically) known general integer factorization algorithm. When attempting to factor an integer N with NFS, we must first choose a polynomial $f \in \mathbb{Z}[X]$ with a known root m modulo N. When f has many roots modulo many small primes, then we say f has good *root properties*. If the magnitude of values taken by f are small, then we say that f has small *size*. It can be shown (heuristically) that if f has good root properties and has small size, then NFS should run faster than when f does not have these properties.

Procedures for generating candidate NFS polynomials with good root properties and small size are described in [2]. Specifically, through the use of *rotations* and *translations*, we hope to generate polynomials with better than average root properties and size. In Sect. 2 we recall some basic facts about homogeneous polynomials and their roots modulo primes. In Sect. 3 we then recall the standard method for generating candidate NFS polynomials. In Sect. 4 we describe a method for rotating candidate NFS polynomials to generate new candidate NFS polynomials with many distinct roots modulo many primes. We discuss how to find potentially small polynomials among polynomials of the form $f(X - \alpha)$, where $f \in \mathbb{Z}[X]$ is fixed and $\alpha \in \mathbb{Z}$ in Sect. 5. We present an algorithm in Sect. 6 that finds candidate NFS polynomials with good root properties and small size based on the methods discussed in Sect. 3-5. Finally, we conclude in Sect. 7 with a discussion of how "good" candidate NFS polynomials generated by the algorithms presented in this paper should be.

* This work was supported in part by grants from the CERIAS Center at Purdue University and from the Lily Endowment Inc.

2 Root Properties

Suppose $f = a_d X^d + \cdots + a_0 \in \mathbb{Z}[X]$ is a polynomial of degree d and $p \in \mathbb{Z}$ is prime. The *homogenization* of f is the polynomial $F \in \mathbb{Z}[X, Y]$ defined by $F(X, Y) = Y^d f(X/Y)$. A co-prime pair (a, b) is a *root* of F modulo p if $F(a, b) \equiv 0 \bmod p$. We shall sometimes refer to (a, b) as simply a root of F if the prime p is understood. Thinking of (a, b) as a point on the projective line $\mathbb{P}^1(\mathbb{F}_p)$, we follow the language of [2] and divide roots into two classes:

- **Projective Roots**: A root (a, b) where p divides b is called *projective*. Note that F will have projective roots if and only if p divides a_d.
- **Regular Roots**: A root (a, b) where p does not divide b is called *regular*. Here, (a, b) is a regular root iff $f(ab^{-1}) \equiv 0 \bmod p$, where b^{-1} is calculated in $\mathbb{F}_p$. A regular root (a, b) with $p \mid a$ is sometimes called a *zero* root.

3 Base-m Method

Given positive integers m, N with $m \leq N$, it is not difficult to find a polynomial $f \in \mathbb{Z}[X]$ such that $f(m) \equiv 0 \bmod N$. A well-known method for doing this is the *base-m method* described in [1]. If $N = a_d m^d + \cdots + a_0$ is the base-m representation of N, where $0 \leq a_i < m$, then by taking $f(X) = a_d X^d + \cdots + a_0$ we have $f(m) \equiv 0 \bmod N$. Given d, the degree of f can be chosen to be d by taking

$$\left\lfloor N^{\frac{1}{d+1}} \right\rfloor < m \leq \left\lfloor N^{\frac{1}{d}} \right\rfloor$$

and constructing f as above. Furthermore, suppose we want to construct a polynomial with leading coefficient L and degree d. If $1 \leq L < N^{1/(d+1)} - 1$, then it is not hard to see that a base-m polynomial with

$$\left\lfloor \left(\frac{N}{L+1}\right)^{\frac{1}{d}} \right\rfloor < m \leq \left\lfloor \left(\frac{N}{L}\right)^{\frac{1}{d}} \right\rfloor$$

will have leading coefficient $a_d = L$.

Finally, we can arrange $-\lfloor m/2 \rfloor < a_i \leq \lfloor m/2 \rfloor$ for $0 \leq i < d$ by using the transformation

$$\begin{aligned} &\text{if } \ a_i > \lfloor m/2 \rfloor, \text{then} \\ &\quad a_i \leftarrow a_i - m \\ &\quad a_{i+1} \leftarrow 1 + a_{i+1} \end{aligned}$$

for $i = 0, 1, \ldots, d-1$. It should be noted that this transformation may change the leading coefficient. This happens precisely when $a_{d-1} > \lfloor m/2 \rfloor$, after applying the transformation. If $a_{d-1} \approx \lfloor m/2 \rfloor$, then $|a_{d-1}| \approx |a_{d-1} - m|$ so we can leave a_{d-1} alone; otherwise, we can change the value of m and start over.

We summarize the above with the following algorithm:

Algorithm 1. (Modified base-m method) Let i, d, L, and N be positive integers with $1 \leq L < N^{1/(d+1)} - 1$. This algorithm attempts to find an integer m and a polynomial $f = a_d X^d + \cdots + a_0 \in \mathbb{Z}[X]$ with $a_d = L$ and $|a_j| \leq m/2$. for $0 \leq j < d-1$, such that $f(m) \equiv 0 \bmod N$. The parameter i allows the user to vary the value of m.

1. [Generate m] Set $m \leftarrow i + \left\lfloor \left(\frac{N}{L+1}\right)^{\frac{1}{d}} \right\rfloor$. If $m > \left\lfloor \left(\frac{N}{L-1}\right)^{\frac{1}{d}} \right\rfloor$, then print "$i$ is too big" and terminate the algorithm.
2. [Build base-m representation of N] Set $temp \leftarrow N$. For $j = 0, \ldots, d$, do
   $a_j \leftarrow temp \bmod m$
   $temp \leftarrow (temp - a_j)/m$.
3. [Adjust a_j] For $j = 0, 1, \ldots, d-2$, do
   If $a_j > \lfloor m/2 \rfloor$, then
   $a_j \leftarrow a_j - m$
   $a_{j+1} \leftarrow 1 + a_{j+1}$.
4. [Build polynomials] Set
   $f_1(X) \leftarrow a_d X^d + \cdots + a_0$.
   If $a_{d-1} > \lfloor m/2 \rfloor$ then set
   $a_{d-1} \leftarrow a_{d-1} - m$
   $a_d \leftarrow 1 + a_d$
   $f_2(X) \leftarrow a_d X^d + \cdots + a_0$;
   otherwise set
   $f_2(X) \leftarrow f_1(X)$.
5. [Output and Terminate] If the leading coefficient of $f_2(X)$ is L, then return m and $f_2(X)$ and terminate the algorithm. Otherwise, if the leading coefficient of $f_1(X)$ is L, then return m and $f_1(X)$ and terminate the algorithm. Finally, if neither leading coefficient is L, then print "i is too big" and terminate the algorithm.

Note that the homogenization of the polynomial generated by Algorithm 1 will have projective roots modulo each prime dividing L.

4 Rotations

Suppose $f \in \mathbb{Z}[X]$ is a polynomial of degree d with root m modulo N. Then $g = f + (b_r X^r + \cdots + b_0)(X - m)$, with $0 \leq r < d$, is a polynomial of degree d (unless $r = d-1$ and $b_{d-1} = -a_d$) with root m modulo N. We call the polynomial $b_r X^r + \cdots + b_0$ a *rotation* of f. Given a finite set of powers of distinct primes $\mathcal{S}$, we look for a rotation that yields a polynomial with good root properties with respect to $\mathcal{S}$. In [2], linear rotations ($r = 1$) are found using a sieve-like procedure. We present an algorithm that finds promising higher degree rotations using the Chinese Remainder Theorem (CRT). The basic idea is to first choose roots $k_{ij} \bmod p_i^{e_i}$. Then for each i find a rotation that yields a polynomial with roots $k_{ij} \bmod p_i^{e_i}$, and finally use CRT to find a single rotation that yields a polynomial with roots k_{ij} for all i, j.

Suppose $\mathcal{S} = \{p_1^{e_1}, \ldots, p_s^{e_s}\}$, where $p_i \neq p_j$ unless $i = j$, and $e_i \geq 1$ for all i. For each p_i and each $0 \leq j \leq r$, choose k_{ij} such that $0 \leq k_{ij} < p_i$, $k_{ij} \neq k_{il}$ unless $j = l$, and p_i does not divide $m - k_{ij}$. This requires $r \leq p_i - 2$ for all i. Now set $z_{ij} = (m - k_{ij})^{-1} f(k_{ij}) \bmod p_i^{e_i}$. If we set $g = f + (b_r X^r + \cdots + b_0)(X - m)$, then k_{ij} will be a root of g modulo $p_i^{e_i}$ for $0 \leq j \leq r$ if

$$b_r k_{ij}^r + \cdots + b_1 k_{ij} + b_0 \equiv z_{ij} \bmod p_i^{e_i} \quad .$$

To determine the b_i modulo $p_i^{e_i}$, we must solve the matrix congruence

$$\begin{pmatrix} 1 & k_{i0} & k_{i0}^2 & \cdots & k_{i0}^r \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & k_{ir} & k_{ir}^2 & \cdots & k_{ir}^r \end{pmatrix} \begin{pmatrix} b_0 \\ \vdots \\ b_r \end{pmatrix} \equiv \begin{pmatrix} z_{i0} \\ \vdots \\ z_{ir} \end{pmatrix} \bmod p_i^{e_i} \quad . \tag{1}$$

We have chosen the k_{ij} so that we may solve this system uniquely. Let $(b_{i0}, \ldots, b_{ir})^T$ denote the unique solution vector modulo $p_i^{e_i}$. Finally, we solve the system of linear congruences

$$\begin{aligned} b_j &\equiv b_{1j} \bmod p_1^{e_1} \\ b_j &\equiv b_{2j} \bmod p_2^{e_2} \\ &\vdots \\ b_j &\equiv b_{sj} \bmod p_s^{e_s} \end{aligned}$$

using CRT, for each $0 \leq j \leq r$.

We now have a polynomial $g = f + (b_r X^r + \cdots + b_0)(X - m)$ such that $g(k_{ij}) \equiv 0 \bmod p_i^{e_i}$ for $0 \leq j \leq r$ and $1 \leq i \leq s$. We should note that the coefficients of g may be larger than the coefficients of f. Explicitly, if $f = a_d X^d + \cdots + a_0$, then $g = c_d X^d + \cdots + c_0$, where

$$c_i = \begin{cases} a_d + b_{d-1} & \text{if } \ i = d \\ a_i + b_{i-1} - m b_i & \text{if } \ 1 \leq i < d \\ a_0 - m b_0 & \text{if } \ i = 0 \end{cases} \quad , \tag{2}$$

where $b_i = 0$ if $i > r$.

Now let $C = \prod_{i=1}^{s} p_i^{e_i}$. We would usually take the b_i as the least positive residue modulo C, but it should be noted that we may as well take $b_i + lC$, where $l \in \mathbb{Z}$, if it suits our purposes. In the best case scenario, we can choose the b_i so that g is a skewed polynomial with coefficients that grow geometrically (roughly) from c_d to c_0. If this is not the case, then it may be possible by using a suitable translation (see Sect. 5). Finally we note that if f has many roots modulo many primes, then its homogenization F will have many regular roots modulo many primes.

We summarize this discussion with the following algorithm:

Algorithm 2. (Rotation) Let $f \in \mathbb{Z}[X]$ be a polynomial of degree d, with root m modulo N. Let $\mathcal{S}$ be a finite set of powers of distinct primes $\mathcal{S} = \{p_1^{e_1}, \ldots, p_s^{e_s}\}$

and $0 \le r < d$. This algorithm finds a polynomial $g \in \mathbb{Z}[X]$ with root m modulo N and at least $r+1$ distinct roots modulo each $p_i^{e_i} \in \mathcal{S}$. If $r = d-1$, then the degree of g will be either d or $d-1$; otherwise the degree of g will be d.

1. [Check parameters] Order the primes so that $p_1 < p_2 < \cdots < p_s$. If $r > p_1 - 2$, then print "Either r is too big or p_1 is too small" and terminate the algorithm; otherwise proceed to the next step.
2. [Pick roots and build z_{ij}] For $i = 1, \dots, s$, do
   For $j = 0, \dots, r$, do
   $k_{ij} \leftarrow j$
   $z_{ij} \leftarrow (m - k_{ij})^{-1} f(k_{ij}) \bmod p_i^{e_i}$.
   If $k_{ij} \equiv m \bmod p_i^{e_i}$ for some j, then set $k_{ij} \leftarrow r+1$ and recalculate z_{ij}.
   Note: there will be at most one such j for each i.
3. [Build b_{ij}] For $i = 1, \dots, s$, calculate $(b_{i0}, \dots, b_{ir})^T$ from (1).
4. [Build b_i using CRT] For $j = 0, 1, \dots, r$, solve
   $$\begin{aligned} b_j &\equiv b_{1j} \bmod p_1^{e_1} \\ b_j &\equiv b_{2j} \bmod p_2^{e_2} \\ &\vdots \\ b_j &\equiv b_{sj} \bmod p_s^{e_s} \end{aligned}$$
   using CRT.
5. [Build $g(X)$] Define c_i as in (2) and set
   $g(X) \leftarrow c_d X^d + \cdots + c_0$.
6. [Output and Terminate] Return $g(X)$ and $\{k_{ij}\}$ and terminate the algorithm.

5 Translations

Let us fix a polynomial $f(X) = a_d X^d + \cdots + a_0 \in \mathbb{Z}[X]$ with $a_d \neq 0$. Note that for $\alpha \in \mathbb{Z}$, the roots of $f(X - \alpha) \in \mathbb{Z}[X]$ will just be the roots of f translated by α. However, the coefficients of $f(X-\alpha)$ will not (in general) be the coefficients of f. So $f(X-\alpha)$ has the same root properties as f, but perhaps differs in size. We now examine the effect of translation on the coefficients of f.

We define $T_f(U) = \{f(X-\alpha) \mid \alpha \in U\}$, where we will be interested in the cases $U = \mathbb{Z}, \mathbb{R}$. Also, fix $\omega = (\omega_0, \dots, \omega_d) \in \mathbb{R}^{d+1}$ and let

$$\|a_d X^d + \cdots + a_0\|_{\omega,\infty} = \max_{0 \le i \le d} |\omega_i a_i|$$

$$\|a_d X^d + \cdots + a_0\|_{\omega,k} = \left(\sum_{i=0}^{d} |\omega_i a_i|^k \right)^{\frac{1}{k}}$$

We will use the more covenient notation $\|\cdot\|_\infty$ and $\|\cdot\|_k$ and drop ω from the notation. Since polynomials with small coefficients tend to have small size, we will refer to $\|f\|_\infty$ as the *size* of f. For our fixed f and ω, we seek $h \in T_f(\mathbb{Z})$ with minimal size. The following proposition is the first step in finding such an h.

Proposition 1. *Fix $f \in \mathbb{Z}[X]$ with $deg(f) = d$, and $\omega \in \mathbb{R}^{d+1}$. Let $k \geq 1$ and take $g_k, h \in T_f(\mathbb{Z})$ with*

$$\|g_k\|_k = \min_{p \in T_f(\mathbb{Z})} \|p\|_k$$

$$\|h\|_\infty = \min_{p \in T_f(\mathbb{Z})} \|p\|_\infty \ .$$

Then

$$\|g_k\|_\infty \leq (d+1)^{\frac{1}{k}} \|h\|_\infty \ .$$

Proof. Let $g_k = b_d X^d + \cdots + b_0$, and $h = c_d X^d + \cdots + c_0$. It is easy to see that $\|g_k\|_\infty \leq \|g_k\|_k$. Let $r_k = \|g_k\|_k$ and suppose that $|\omega_i c_i| < r_k/(d+1)^{\frac{1}{k}}$ for all i. Then

$$\|h\|_k^k = \sum_{i=0}^{d} |\omega_i c_i|^k < (d+1) \left(\frac{r_k}{(d+1)^{\frac{1}{k}}} \right)^k = r_k^k = \|g_k\|_k^k$$

which implies that $\|h\|_k < \|g_k\|_k$, a contradiction since $\|g_k\|_k$ is minimal in $T_f(\mathbb{Z})$. So there must be some i such that $|\omega_i c_i| \geq r_k/(d+1)^{\frac{1}{k}}$. But this means that $\|h\|_\infty \geq r_k/(d+1)^{\frac{1}{k}}$, which immediately implies $\|g_k\|_k \leq (d+1)^{\frac{1}{k}} \|h\|_\infty$.

Proposition 1 gives us the tool we need to find h.

Corollary 1. *If $k > \frac{\ln(d+1)}{\ln(1+\|f\|_\infty^{-1})}$, then $\|g_k\|_\infty = \|h\|_\infty$.*

Proof. For $k \geq 1$, Proposition 1 says that

$$0 \leq \|g_k\|_\infty - \|h\|_\infty \leq ((d+1)^{\frac{1}{k}} - 1)\|h\|_\infty \ .$$

But since $\|h\|_\infty \leq \|f\|_\infty$, we have

$$0 \leq \|g_k\|_\infty - \|h\|_\infty \leq ((d+1)^{\frac{1}{k}} - 1)\|f\|_\infty \ .$$

Now $((d+1)^{\frac{1}{k}} - 1)\|f\|_\infty \to 0$ as $k \to \infty$. But since $\|g_k\|_\infty - \|h\|_\infty$ is a nonnegative integer, as soon as $((d+1)^{\frac{1}{k}} - 1)\|f\|_\infty < 1$, we must have $\|g_k\|_\infty = \|h\|_\infty$.

Notice that although Corollary 1 gives us a way of finding h with minimal size in $T_f(\mathbb{Z})$ in theory, there is little hope of using this result in practice. In fact, $\ln x \approx x - 1$ when $x \approx 1$, so the denominator of the lower bound will be approximately $\|f\|_\infty^{-1}$, when $\|f\|_\infty$ is large. If we were to try to use Corollary 1, we would end up having to find the critical points of a degree kd polynomial, as we shall see shortly, which is clearly unreasonable when k is very large. With that said, Proposition 1 says that even for small k we can generate a polynomial with size equal to a rather small constant times $\|h\|_\infty$. With this in mind, let us now consider how we can find g_k as defined in Proposition 1. Observe that

$$\begin{aligned} f(X-\alpha) &= \sum_{i=0}^{d} a_i (X-\alpha)^i \\ &= \sum_{i=0}^{d} a_i \sum_{j=0}^{i} \binom{i}{j} (-\alpha)^{i-j} X^j \\ &= \sum_{j=0}^{d} \left[\sum_{i=j}^{d} a_i \binom{i}{j} (-\alpha)^{i-j} \right] X^j \\ &= \sum_{j=0}^{d} p_j(\alpha) X^j \quad , \end{aligned}$$

where

$$p_j(\alpha) = \sum_{i=j}^{d} a_i \binom{i}{j} (-\alpha)^{i-j} \quad .$$

Now define $m_k(\alpha) = \sum_{i=0}^{d} (\omega_i p_i(\alpha))^k$ and change variables to get $m_k(X) = \sum_{i=0}^{d} (\omega_i p_i(X))^k$. Let k be even. Then $m_k(X)$ is a polynomial of degree kd, with $m_k(X) \geq 0$ and $m_k(\alpha) = \|f(X-\alpha)\|_k^k$. Finding the value in $\mathbb{Z}$ at which $m_k(X)$ achieves its absolute minimum is a straightforward task for small k.

We summarize this discussion with the following algorithm:

Algorithm 3. (Translation) Let $f \in \mathbb{Z}[X]$ be a polynomial of degree d, let k be a positive even integer and let $\omega \in \mathbb{R}^{d+1}$. This algorithm finds a polynomial $g_k \in \mathbb{Z}[X]$ and $\alpha_k \in \mathbb{Z}$, with $g_k(X) = f(X-\alpha_k)$ and $\|g_k\|_\infty$ less than or equal to $(d+1)^{\frac{1}{k}}$ times the size of a polynomial in $T_f(\mathbb{Z})$ of minimal size. In the process of computing g_k, the algorithm will compute the critical points of a polynomial of degree kd. If $k \geq \kappa := \left\lceil \frac{\ln(d+1)}{\ln(1+\|f\|_\infty^{-1})} \right\rceil$, then the algorithm will instead compute the critical points of a polynomial of degree κd, and g_k will have minimal size in $T_f(\mathbb{Z})$.

1. [Generate κ] Set
   $\kappa \leftarrow \left\lceil \frac{\ln(d+1)}{\ln(1+\|f\|_\infty^{-1})} \right\rceil$
   $minimal? \leftarrow false$.
2. [Is k too big?] If $k \geq \kappa$, then set
   $k \leftarrow \kappa$ if κ is even;
   otherwise set
   $k \leftarrow \kappa + 1$
   $minimal? \leftarrow true$.
3. [Generate translate coefficients] For $j = 0, 1, \ldots, d$, set
   $p_j(X) \leftarrow \sum_{i=j}^{d} a_i \binom{i}{j} (-X)^{i-j}$.
4. [Build $m_k(X)$] Set
   $m_k(X) \leftarrow (\omega_d p_d(X))^k + \cdots + (\omega_0 p_0(X))^k$.
5. [Find critical numbers] Find $\alpha_{k1}, \ldots, \alpha_{kl}$ such that $m_k'(\alpha_{ki}) = 0$ for all i.
6. [Identify α_k] Find $\alpha_k \in \{\lfloor \alpha_{ki} \rfloor, \lceil \alpha_{ki} \rceil\}_{i=1}^{l}$ such that
   $m_k(\alpha_k) \leq m_k(\lfloor \alpha_{ki} \rfloor), m_k(\lceil \alpha_{ki} \rceil)$ for all i.
7. [Build g_k] Set $g_k \leftarrow f(X-\alpha_k)$.
8. [Output and Terminate] Return α_k and g_k. If $minimal? = true$, print "This polynomial has minimal size in $T_f(\mathbb{Z})$." In either case, terminate the algorithm.

6 Candidate NFS Polynomials

We can use Algorithms 1-3 to generate candidate NFS polynomials. More precisely, let us fix a positive integer N that we wish to factor and $d \geq 1$. Pick a suitable leading coefficient L, divisible by many small primes. We use Algorithm 1 to generate a polynomial f_1 of degree d with leading coefficient L and root m modulo N. We can then use Algorithm 2 to rotate f_1 by a polynomial of degree $r = d - 2$. This generates a polynomial f_2 with at least $d - 1$ roots modulo each element in some fixed set $\mathcal{S}$ of small powers of distinct primes. Also, f_2 has leading coefficient L and root m modulo N. Finally, we use Algorithm 3 with a suitable choice for ω to produce a polynomial f_3 which has all the root properities as f_2, with perhaps minimal size in $T_{f_2}(\mathbb{Z})$.

At this point, we have a candidate NFS polynomial with good root properties and small size. However, if this polynomial is not satisfactory for some reason, adjustments can be made to generate more polynomials. For example, we may generate many candidate NFS polynomials by varying i and L in Algorithm 1, $\mathcal{S}$ in Algorithm 2, or ω in Algorithm 3. The following algorithm combines Algorithms 1-3 to produce candidate NFS polynomials:

Algorithm 4. (NFS candidate polynomial) Let $N \geq 1$ be a number that we wish to factor, L, d, and i be positive integers, $\mathcal{S} = \{p_1^{e_1}, \ldots, p_s^{e_s}\}$ be a finite set of small powers of distinct primes, and $\omega \in \mathbb{R}^{d+1}$. This algorithm attempts to produce a candidate NFS polynomial with at least $d - 1$ roots modulo every $p_i^{e_i} \in \mathcal{S}$.

1. [Generate a base-m polynomial] Generate m and f_1 from Algorithm 1, using inputs i, d, L and N. If Algorithm 1 returns an error message, print the error message and terminate the algorithm.
2. [Rotate f_1] Generate f_2 and $\{k_{ij}\}$ from Algorithm 2, using inputs f_1, d, m, N, $\mathcal{S}$, and $r = d-2$. If Algorithm 2 returns an error message, print the error message and terminate the algorithm.
3. [Translate f_2] Generate f_3 and α from Algorithm 3, using inputs f_2, d, and ω. If Algorithm 3 generates a message, print the message.
4. [Translate k_{ij}] For all i, j set:
   $k_{ij} \leftarrow k_{ij} + \alpha$.
5. [Output and Terminate] Return f_3 and $\{k_{ij}\}$ and terminate the algorithm.

7 Conclusion

One may wonder how "good" candidate NFS polynomials generated by Algorithm 4 will be. Let $f \in \mathbb{Z}[X]$ have degree d and $F \in \mathbb{Z}[X, Y]$ be the homogenization of f. One measure of "goodness" is

$$\alpha_B(F) = \sum_{p \leq B} \left(1 - q_p \frac{p}{p+1}\right) \frac{\ln p}{p-1}$$

where the sum is over all well-behaved primes (primes that do not divide the discriminant of f) less than or equal to some bound B, and q_p is the number of roots (regular and projective) of F modulo p, as defined in [2]. Heuristically and roughly speaking, we expect a typical value $F(x, y)$ to behave like a random integer of size $F(x, y) \cdot e^{\alpha_B(F)}$. So the more negative $\alpha_B(F)$ is, the "better" F should be. But clearly $\alpha_B(F)$ will be more negative whenever q_p is large for small primes p. Now $0 \le q_p \le d + 1$. However, by using Algorithm 4 we can force $q_p \ge d$ for each $p_i | L$ with $p_i^{e_i} \in \mathcal{S}$. If $\mathcal{S} = \{p_1^{e_1}, \ldots, p_s^{e_s}\}$ with $p_1 < p_2 < \cdots < p_s \le B$, with $r \le p_1 - 2$, then we will have a polynomial F which very likely has $\alpha_B(F) \ll 0$. Finally, by adjusting the coefficients after the CRT-step, or by using Algorithm 3, one hopefully has a suitable polynomial for factoring N using the Number Field Sieve. Future work will be devoted to identifying optimal parameters (i.e. $L, \mathcal{S}, \omega$) for a given N.

Acknowledgements

The author wishes to thank Samuel S. Wagstaff, Jr. for valuable conversations concerning this work, as well as the anonymous referees for their useful comments on the paper.

References

1. A.K. Lenstra and H.W. Lenstra, Jr., editors. *The Development of the Number Field Sieve*, volume 1554 of *Lecture Notes in Mathematics*. Springer-Verlag, Berlin, 1993.
2. Brian Murphy. *Polynomial Selection for the Number Field Sieve Integer Factorisation Algorithm*. PhD thesis, Australian National University, July 1999.