Polynomial Selection for the Number Field Sieve

Shi Bai

白石

Sep 2011

A thesis submitted for

the degree of Doctor of Philosophy

of the Australian National University

Declaration

The work in this thesis is my own except where otherwise stated.

Dedicated to my parents for their love and inspiration.

Acknowledgements

It is difficult to overstate my gratitude to my Ph.D. supervisor, Richard Brent. Over the three-and-a-half years, Richard has provided tremendous help and encouragement, lots of interesting ideas and insightful advice, without which my Ph.D study and this thesis would not have been possible. His patience, valuable comments and careful proofreading for the thesis is deeply appreciated.

Part of my research has been carried out during my visits to the CARAMEL group of INRIA Nancy. I was fortunate to be able to spend a few months working with Paul Zimmermann, from whom I learned much about the subject and to whom I must say thank you. I am also very grateful to Pierrick Gaudry and Emmanuel Thomé for many suggestions and help.

Many thanks go out to Thorsten Kleinjung, Peter Montgomery, Jason Papadopoulos and Herman te Riele for various discussions and instructions regarding the number field sieve, either face-to-face or via email.

Many people in the Mathematical Sciences Institute (MSI) and the Research School of Computer Science (RSCS) of the Australian National University (ANU) assisted and encouraged me in various ways during my studies. I am particularly grateful to Jörg Arndt, Markus Hegland, Chin Khoo, Paul Leopardi, Weifa Liang, Jiakun Liu, Brendan Mckay, Sudi Mungkasi, Judy-anne Osborn, Srinivasa Subramanya Rao, Alistair Rendell, Peter Strazdins, Vikram Sunkara and Bin Zhou for various discussions and chatting; Nick Guoth, Joshua Rich and Warren Yang for helping me play around with the cluster.

I must thank the MSI for providing me with office space and computing facilities. I also thank ANU, the Research School of Computer Science at ANU, the Centre of

Excellence for Mathematics and Statistics of Complex Systems and the INRIA Nancy, who generously provided a scholarship and funding for my research.

Last but not least, a big thank you to my parents for their love, support and encouragement.

Abstract

The number field sieve is asymptotically the most efficient algorithm known for factoring large integers. It consists of several stages, the first one being polynomial selection. The running time of subsequent steps depends on the quality of the chosen polynomials. In the thesis, we discuss the current state of the art in polynomial selection.

Polynomial selection can be divided into three stages: polynomial generation, size optimization and root optimization. We give some analysis of polynomial generation algorithms. We then describe some improvements for polynomial size optimization and root optimization. The size optimization is based on determining a translation and then locally optimizing the polynomial. The root optimization is based on Hensel's lifting lemma and a root sieve on congruences classes modulo small prime powers. The improvements described here have been implemented and used to obtain some good polynomials in practice. We also discuss some recent progress on polynomial selection using lattice reduction.

Notation

The following notations are used in this thesis.

$\mathbb{Z}$	The ring of rational integers.
$\mathbb{Q}$	The field of rational numbers.
$\mathbb{C}$	The field of complex numbers.
$\mathbb{Z}/n\mathbb{Z}$	The ring of integers modulo n.
$\mathbb{F}_p$	The finite field of p elements.
$\mathbb{Z}[x]$	The polynomial ring with integer coefficients.
$\mathbb{Z}[\alpha]$	The ring generated by a zero α of a given polynomial $f \in \mathbb{Z}[x]$.
$\mathbb{Q}[\alpha]$	The field of fractions of $\mathbb{Z}[\alpha]$.
$\mathcal{O}_K$	The ring of algebraic integers of $K = \mathbb{Q}(\alpha)$.
$a \mid b$	a divides b.
$a \nmid b$	a does not divide b.
$\log x$	The natural logarithm of x.
$\log_b x$	The logarithm of x to base b.
$\exp(x)$	The exponential function of x.
$\lfloor x \rfloor$	The floor function of x.

$\lceil x \rceil$	The ceiling function of x.
$f(x) = O(g(x))$	There exist constants x_0 and c such that $\lvert f(x)\rvert \leq c\,g(x)$ for all $x \geq x_0$.
$f(x) = o(g(x))$	$\lim_{x\to\infty} f(x)/g(x) = 0$.
B-smooth	An integer is B-smooth if none of its prime factors is greater than B.
Factor base	A set of (relatively) small prime numbers.
$\rho(x)$	Dickman-de Bruijn "rho" function, see §2.3.
$\Psi(x, B)$	Number of B-smooth integers $\leq x$, see §2.3.
$\left(\frac{x}{p}\right)$	Legendre symbol, see §2.2.
$\nu_p(x)$	The exponent of the largest power of p dividing $x \in \mathbb{Z}$. We define $\nu_p(0) = \infty$.
$L_n(\alpha, c)$	$\exp\big((c + o(1))(\log n)^\alpha(\log\log n)^{1-\alpha}\big)$, see §1.3.5.
$l^1(F)$	l^1-norm on the coefficients of f, see §3.2.2.
$l^\infty(F)$	l^∞-norm on the coefficients of f, see §3.2.2.
$L^2(F)$	L^2-norm of F, see §3.2.2.

Contents

Chapter 1

Introduction

Public-key cryptography such as RSA [89] has been widely used in modern communication technologies. It allows users to communicate confidently without acquiring a secret key in advance. The difficulty of some number-theoretical problems such as integer factorization and discrete logarithm is relevant to the security of public-key cryptosystems.

In this chapter, we describe public-key cryptography and some underlying mathematical problems. We also discuss some algorithms for integer factorization and discrete logarithms, without going into technical details which are the subject of later chapters.

1.1 Public-key cryptography

Modern cryptography is about communication in the presence of adversaries [88]. One of the major goals of cryptography is privacy: two parties want to communicate privately, so that an adversary knows nothing about the content.

Modern cryptography can be divided into two categories: symmetric-key (secret-key) cryptography and public-key cryptography. Symmetric-key cryptography uses a single key or a pair of trivially-related keys for both encryption and decryption. The communication channel needs to be secure to pass the key information from the sender to the receiver. Public-key cryptography uses a pair of keys for encryption and decryption. One of these keys (public key) is published or public and the other is kept private (private key). The two keys are not trivially-related and the acquisition of the public key does not lead to the exposure of the secret key. A public key algorithm does not need to assume a secure communication channel to pass the decryption key, although the channel should be authenticated to avoid "man-in-the-middle" attacks.

Diffie, Hellman and Merkle [30, 61] are among the pioneers to make the concept of public-key cryptography public. Diffie and Hellman described a key-exchange protocol to allow two parties to agree on a shared secret key. In recognition of Merkle's contribution to the invention of public-key cryptography, it is named as the Diffie-Hellman-Merkle key-exchange protocol [41]. The Diffie-Hellman-Merkle key-exchange algorithm depends on the assumed difficulty of the discrete logarithm problem.

Rivest, Shamir and Adleman [89] proposed a public-key cryptographic algorithm suitable for both signing and encryption, known as RSA. The RSA algorithm depends on the assumed difficulty of the large integer factorization problem.

Koblitz and Miller [53, 62] proposed to realize public-key cryptography using the algebraic structure of elliptic curves over finite fields, hence the name elliptic curve cryptography (ECC). The ECC is becoming more and more popular due to the smaller key size required in comparison to other cryptosystems such as RSA. It depends on the assumed difficulty of the elliptic curve discrete logarithm problem.

The integer factorization and discrete logarithm problems are of great research interest since they are believed to be difficult. On the other hand, their corresponding reverse operations (integer multiplication and discrete exponentiation) are easy[*]. Such properties are used in or relevant to the construction of public-key cryptosystems. For instance, the RSA cryptosystem is insecure if integer factorization is easy.

1.2 RSA and Integer factorization

The integer factorization problem is to decompose a composite integer into its prime factors. It is believed to be hard and no algorithm is known to factor large integers in polynomial time. We describe the relation between RSA and integer factorization.

[*]"Easy" means they can be performed in polynomial time on a deterministic Turing machine.

RSA, named after Rivest, Shamir and Adleman, is one of the most widely used public-key cryptosystems. It is believed to be secure, given a careful implementation with appropriate parameters. RSA uses a pair of keys (public key and private key). The public key is open and is used for encryption. The private key must be kept secret to decrypt the ciphered message. We describe the ideas underlying a basic RSA encryption algorithm.

Let $n = pq$ where p and q are two distinct, large prime numbers of similar size. Define $\varphi(n)$ to be Euler's totient function, which counts the number of integers less than or equal to n and coprime to n. Since n is a product of two prime numbers, $\varphi(n) = (p-1)(q-1)$. Choose e such that $1 < e < \varphi(n)$ and is coprime to $\varphi(n)$. e and n are published as the public key. The private key d can be computed from $d = e^{-1} \pmod{\varphi(n)}$.

The receiver publishes the public key and keeps the private key secret. The sender uses the public key from the receiver to encrypt a message. Let m be a message which is encrypted by $c = m^e \pmod{n}$. The ciphered message c is used in the communication. Given m, d, the receiver deciphers the message using $m = c^d \pmod{n}$.

There is no known practical method to break the RSA cryptosystem, assuming a careful implementation and choice of parameters. If the e-th root modulo a composite integer n can be computed efficiently, the message can be recovered. Alternatively, if n can be factored efficiently, the message can be recovered. The difficulty of the integer factorization problem is crucial to the security of RSA. It is clear that, if the size of n is small, RSA can be broken by factoring n. However, no efficient algorithm is known for factoring large general integers. In the next section, we describe some algorithms for integer factorization. For more information on integer factorization, we refer to surveys by Brent, Lenstra, Montgomery and Pomerance [16, 56, 67, 86].

1.3 Some algorithms for integer factorization

In recent years, the limits of the best integer factorization algorithms have been extended greatly, due in part to Moore's law and in part to algorithmic improvements [15].

It is now routine to factor numbers of 512 bits, and feasible to factor numbers of 768 bits [50]. We describe several integer factorization algorithms.

1.3.1 Pollard's ρ method

Pollard [81] described a Monte Carlo factorization algorithm in 1975, known as Pollard's ρ method. This method is efficient to find small factors.

Let x_0 be a random integer in $\mathbb{Z}/n\mathbb{Z}$ and $f(x) \in \mathbb{Z}[x]$ be an easily-computable polynomial. For instance, $f(x) = x^2 + a$ with $a \neq -2, 0$. We consider a sequence of pseudo-random integers $\{x_0, x_1, x_2, \cdots, x_i, \cdots\}$ formed by $x_{i+1} = f(x_i) \pmod{n}$. Let p be a non-trivial factor of n. If $x_i \equiv x_j \pmod{p}$ and $x_i \not\equiv x_j \pmod{n}$ for some i, j, we see that $p \mid \gcd(x_i - x_j, n) < n$. Therefore, $\gcd(x_i - x_j, n)$ is likely to yield some non-trivial factor of n.

Since the subring $\mathbb{Z}/p\mathbb{Z}$ is finite, there exist integers μ and λ such that $x_\mu \equiv x_{\mu+\lambda} \pmod{p}$, and hence $x_{\mu+i} \equiv x_{\mu+\lambda+i} \pmod{p}$ for all $i \geq 0$. The smallest such integers λ and μ respectively are the period and aperiodic lengths of the pseudo-random sequence. Under the assumption that the function $f(x)$ behaves like a random mapping, the expected value of $\mu+\lambda$ is about $\sqrt{(\pi p)/2}$ [40, 35]. How can we detect such i, j efficiently with feasible memory?

A cycle-detection algorithm can be applied with a small penalty in the running time. Pollard used Floyd's cycle-detection algorithm [52]. Floyd's method uses only a small, constant amount of memory. The worst case takes $\mu + \lambda$ iterations, each of which consists of three evaluations and one comparison. Brent [11] described another cycle-detection method which also uses only a small memory and is about 36 percent faster than Floyd's on average. The application of Brent's method to factoring is about 24 percent faster than Pollard's original version. Montgomery and Pollard [81, 64] also described some methods to reduce the GCD (greatest common divisor) costs in the algorithms.

Pollard's ρ method runs heuristically in average bit-complexity $O(n^{1/4} \log^2 n)$ [11]. In practice, the method is efficient for integers with small factors. The eighth Fermat number[†] was factored by Brent and Pollard [17] using a variation of the above method, finding a 16-digit factor.

1.3.2 Pollard's $p - 1$ method

Pollard [80] proposed another factorization algorithm based on Fermat's little theorem (FLT), known as Pollard's $p - 1$ method.

Let p be a prime factor of n and a be an integer coprime to p. It is clear from FLT that $a^{p-1} \equiv 1 \pmod{p}$. We may try various integers e in the hope that $(p-1) \mid e$. We may then use $\gcd(a^e - 1, n)$ to recover the factor p. The idea is to choose a suitable exponent e such that $a^e \equiv 1 \pmod{p}$, but $a^e \not\equiv 1 \pmod{n}$.

The basic way is to choose e to be a product of small prime powers under some limit B. Then we raise a to the e-th power and compute above greatest common divisor. This method fails if any prime power factor of $p - 1$ is larger than the bound B.

In practice, a large prime variant is often used, namely the two-stage method. We choose another bound $B' > B$. In the two-stage method, we require that all but one of the prime power factors of $p - 1$ are smaller than B, and the largest prime factor of $p - 1$ is smaller than B'.

Pollard [80] described a standard two-stage method that tests successive large primes q_i. Let $a_1 \equiv a^e \pmod{n}$ from the first stage. We look up a table consisting of values $a_1^{q_{i+1}-q_i} \pmod{n}$ and multiply it by $a_1^{q_i} \pmod{n}$ to form $a_1^{q_{i+1}} \pmod{n}$. Pollard [80] also suggested a Fast Fourier Transform (FFT) continuation. Some other second-stage methods have been proposed for the $p - 1$ factorization algorithm. We refer to works by Brent, Kruppa, Montgomery and Silverman [12, 64, 70, 69]. Some of these also have analogues for the elliptic curve method, described in the next subsection.

1.3.3 The elliptic curve method

Lenstra [59] proposed a sub-exponential[‡] algorithm for integer factorization that uses the group structure of elliptic curves, namely the elliptic curve method (ECM). For

[†]The i-th Fermat number is $F_i = 2^{2^i} + 1$.

[‡]Sub-exponential time means expected time $e^{o(x)}$ where x is the input size.

background on elliptic curves, we refer to the literature [92].

ECM uses a similar idea to the $p-1$ method, but replacing the multiplicative group of a finite field by the group of points on a random elliptic curve. It may have two stages, analogous to the $p-1$ method.

Let n be a composite integer. We choose a random elliptic curve E and a random point P on the curve. In the first stage, we compute a multiple eP of the point P, where e is a product of small prime powers. In the second stage, we compute various $q_i eP$ for some larger primes q_i up to some bound. We hope to meet the point at infinity on the elliptic curve, in which case we can recover the factor by taking a GCD.

Let p be a factor of n and $E(\mathbb{F}_p)$ be the group of points on the elliptic curve over the finite field $\mathbb{F}_p$. Let N be the order of $E(\mathbb{F}_p)$. We see that $NP = \infty$. If N consists of smaller prime factors and one large prime q_i, the above procedure may give the factor. The group order N is in $[p+1-2\sqrt{p},\, p+1+2\sqrt{p}]$ by Hasse's theorem [92]. We hope to see a random curve with a smooth group order.

In the $p-1$ factoring method, the multiplicative group of $\mathbb{F}_p$ of order $p-1$ is used. The method is usually successful if $p-1$ has only small prime factors. Similarly, ECM is usually successful if the group order has only small prime factors, i.e. is smooth. If one trial of ECM is unsuccessful, we can repeat the algorithm by starting with another random elliptic curve and hope its group order is smooth. Heuristically, ECM has expected running time

$$\exp\left((\sqrt{2}+o(1))(\log p)^{1/2}(\log\log p)^{1/2}\right).$$

Some notable factorizations by ECM include the complete factorization of the tenth and eleventh Fermat numbers by Brent [14, 13]. The current record[§] for ECM is a 73-digit factor found by Bos, Kleinjung, Lenstra and Montgomery [10].

We refer to the literature [7, 12, 64, 65, 94, 99] for analysis, further improvements and implementations of ECM. We also mention some recent development in using hyperelliptic curves [25] to factor integers.

[§]At the time when the thesis is written.

1.3.4 Congruence of squares

Fermat's method [86] tries to find two integers x, y such that $n = x^2 - y^2$. If such integers are found, we may recover a factor by taking $\gcd(n, x - y)$. The basic procedure is to choose many $x_i \geq \lceil \sqrt{n} \rceil$ and hope that $x_i^2 - n$ is a square for some i.

The running time of above methods depends mainly on the size of the prime factors. We now describe a class of methods (congruence of squares) whose running time depends mainly on the size of n.

Kraitchik [86] described a way to produce a congruence of squares by combining products of easily-factorisable elements $x_i^2 - n$. The basic procedure is to compute a sequence of $\{x_i^2 - n\}$ and find a set I that $\prod_{i \in I}(x_i^2 - n)$ is a square.

Morrison and Brillhart's CFRAC (continued fraction) method [71] uses the continued fraction expansion of $\sqrt{kn}$ for some small k to generate a sequence of numbers. They also described a systematic way to produce a square congruence by linear algebra. The CFRAC method has expected running-time

$$\exp\left((2 + o(1))(\log n)^{1/2}(\log\log n)^{1/2}\right).$$

The seventh Fermat number was factored by Morrison and Brillhart in 1970 using CFRAC [71].

Pomerance [85] described a sieve-based algorithm named the quadratic sieve (QS), which is asymptotically faster than the above methods. Like the above methods, QS attempts to find many x_i such that $x_i^2 - n$ factors completely into small prime powers. This is done by a sieve using the following principle. Let p be a prime such that $p \mid f(x)$ for some polynomial $f(x) \in \mathbb{Z}[x]$, e.g. $f(x) = x^2 - n$, then $p \mid f(x + jp)$ for all j. This procedure is repeated for all the primes in the factor base. In the end, we find all $f(x_i)$ which are smooth and use linear algebra to find a dependency. The sieve method is asymptotically and practically better than testing smoothness by trial division.

Montgomery and Silverman [93] gave a way to choose several different polynomials for the sieving, leading to the multiple polynomial quadratic sieve (MPQS). The sieving on various polynomials can be parallelised.

Heuristically, the quadratic sieve method has expected running time

$$\exp\left((1+o(1))(\log n)^{1/2}(\log\log n)^{1/2}\right).$$

Asymptotically, this is comparable to ECM if we assume the size of the smallest factor p is about half of the size of n.

Some notable factorizations by MPQS include the factorization of RSA-100, RSA-110, RSA-120 and RSA-129 [31, 29, 4]. All the other factored RSA numbers were done by the number field sieve.

1.3.5 The number field sieve

The general number field sieve (GNFS) [57] is the most efficient algorithm known for factoring large general integers. It uses a similar idea to the "congruence of squares" methods. The general number field sieve were developed from the special number field sieve (SNFS) [83], where the latter method can be used to factor integers of certain forms. For example, the ninth Fermat number was factored completely using SNFS in 1990 [58].

GNFS has a conjectured running time $L_n(1/3, (64/9)^{1/3})$ as $n \to \infty$. Here the L-function on input n for parameters c, α $(0 \le \alpha \le 1)$ is defined to be

$$L_n(\alpha, c) = \exp\left((c+o(1))(\log n)^{\alpha}(\log\log n)^{1-\alpha}\right).$$

If $\alpha = 0$, $L_n(0, c) = (\log n)^{c+o(1)}$ gives a polynomial function in the input size $\log n$. If $\alpha = 1$, $L_n(1, c) = n^{c+o(1)}$ gives an exponential function in the input size $\log n$. The NFS runs in sub-exponential time. In comparison, SNFS runs in $L_n(1/3, (32/9)^{1/3})$ due to smaller coefficients of the polynomial pair (see Chapter 3).

GNFS has been used in many (current and previous) record factorizations such as RSA-768 [50]. Due to the factorization of RSA-768, whose size is 768 bits, the parameter n in RSA should be at least 1024 bits. It also suggests that it is prudent to phase out usage of 1024-bit RSA within the next three to four years [50, 9]. We extend Brent's analysis [16] on curve fitting and extrapolation. Assuming Moore's law, we expect $(\log n)^{1/3}(\log\log n)^{2/3}$ is roughly a linear function of calendar year. We plot $(\log n)^{1/3}(\log\log n)^{2/3}$ of some RSA numbers[¶] against the year they were factored in Figure 1.1.

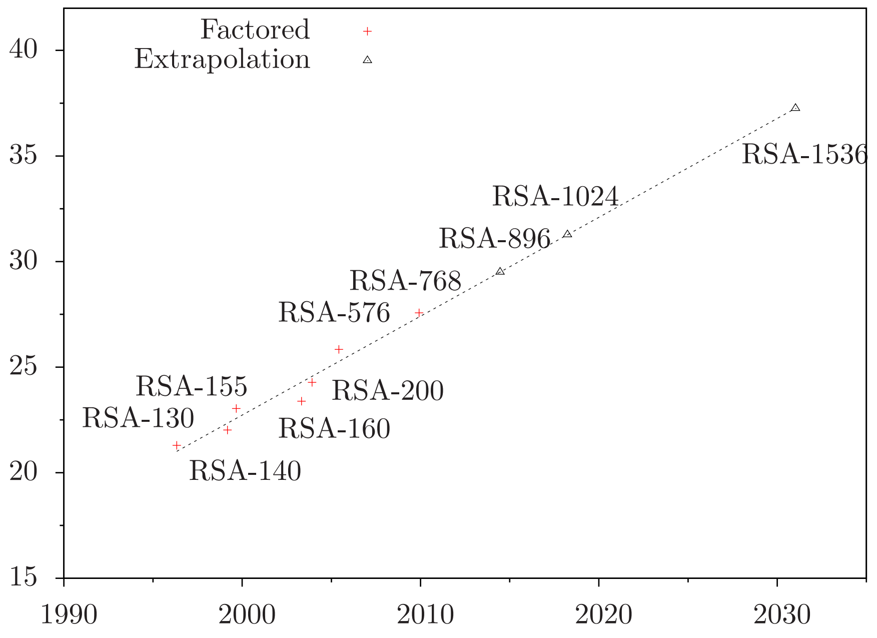

Figure 1.1: $(\log n)^{1/3}(\log\log n)^{2/3}$ versus Year

The straight line fitted by the least-squares method is

$$x = 2.15y + 1951.45$$

where x stands for Year and y stands for $(\log n)^{1/3}(\log\log n)^{2/3}$. Three projected factorization years for RSA numbers are RSA-896, RSA-1024 and RSA-1536 in order. The fitted function roughly give years 2014, 2018 and 2031 respectively. Assuming Moore's law and adequate algorithmic improvements, it may be feasible to factor RSA-1024 within the next decade.

Coppersmith [23] used multiple number fields (defined by polynomials with the

[¶]RSA-130, RSA-140, RSA-155, RSA-160, RSA-576, RSA-200, RSA-768.

same root modulo n) to reduce the asymptotic complexity of NFS for factoring from $L_n(1/3, 1.923)$[‖] to $L_n(1/3, 1.902)$. The idea is to reuse the rational-side sieve on $a - bm$. The algebraic smoothness tests are applied on pairs which pass the rational sieve. The smoothness tests are often done by ECM. In the same paper, Coppersmith also gave a method to reuse some pre-computations to factor integers close to n, namely the "factorization factory" method. The main idea is that the root of polynomials is independent from n. The individual factorization takes $L_n(1/3, 1.639)$. The pre-computation requires to save a table of $L_n(1/3, 1.639)$ space.

In this thesis, the term "number field sieve" refers to the general number field sieve [57] unless otherwise mentioned. We will describe the principle of the number field sieve for integer factorization in the next chapter.

1.4 Discrete logarithm

The discrete logarithm can be considered as the inverse operation of discrete exponentiation in a finite abelian group[**]. For instance, let g and h be elements of a finite cyclic group such that $g^x = h$. The solution x is a discrete logarithm of h to base g.

The discrete logarithm problem (DLP) is believed to be hard in certain groups. There is no known efficient algorithm to solve the DLP in the multiplicative group of a finite field or in an elliptic curve group in the general case. In the finite-field case, there is a sub-exponential algorithm but in the elliptic curve case only fully exponential algorithms are known. The presumed hardness is relevant to many cryptographic protocols/systems such as Diffie-Hellman key-exchange and ElGamal encryption [33].

The Pollard ρ method for discrete logarithm [82] and its variants [96] are probabilistic algorithms to solve DLP. The main ideas are similar to Pollard's ρ algorithm for factoring. They work by defining a pseudo-random sequence of group elements and looking for a cycle in the sequence. The elements in the pseudo-random sequence are generated by an iteration function $x_{i+1} = f(x_i)$. The procedure is completed when

[‖]The constant is $(64/9)^{1/3} (\approx 1.923)$. For convenience, we use the decimal notation in the rest of this chapter.

[**]The group operation is expressed in multiplicative notation.

it generates a collision. A collision happens when an element is drawn that has been drawn before. For the elliptic curve discrete logarithm problem (ECDLP), there is no known general sub-exponential algorithm. Van Oorschot and Wiener's parallelised Pollard's ρ method [97] is the most efficient method known for the ECDLP in the general case.

There exist sub-exponential algorithms for discrete logarithm problems over multiplicative groups of finite fields. We outline some developments.

For prime fields $\mathbb{F}_p$, Gordon [37] described a method to solve the discrete logarithm in heuristic running time $L_p(1/3, 2.080)$. Schirokauer [91] defined some easily-computable maps (named Schirokauer maps) which help to construct the linear system for solving the DLP. His algorithm runs in time $L_p(1/3, 1.923)$, which is the same as GNFS for factoring. Joux and Lercier [45] described some ways to compute individual logarithms efficiently once some pre-computation is done. They also gave a polynomial selection method to produce pair of polynomials of degrees d and $d + 1$. The individual logarithms take heuristically $L_p(1/3, 1.442)$ once a pre-computation in time $L_p(1/3, 1.923)$ is done.

Matyukhin [60], using Coppersmith [23] (multiple number fields) and Schirokauer's ideas, gave an algorithm to compute discrete logarithms in $L_p(1/3, 1.902)$. Individual logarithms require the same amount of work. Commeine and Semaev [22] improved the individual logarithm computation and adapted Coppersmith's multiple polynomials idea. The individual logarithms take $L_p(1/3, 1.442)$, after a pre-computation in time $L_p(1/3, 1.902)$ is done. Bărbulescu and Gaudry [18] used the factorization factory idea to speed up the pre-computation to $L_p(1/3, 1.639)$ and individual logarithm to $L_p(1/3, 1.232)$.

In finite fields $\mathbb{F}_{p^n}$ of small characteristic, the function field sieve (FFS) [2] is the best known algorithm for computing discrete logarithms. FFS chooses a polynomial to construct an extension field over $\mathbb{F}_p(x)$. The sieve tries to identify irreducible polynomials of small degrees. Joux and Lercier [44] described an efficient version of FFS. The FFS and its variant run in time $L_{p^n}(1/3, 1.526)$. Both of them are suitable for the case when the base field is small and the extension degree grows. For finite fields of medium characteristic, Joux, Lercier, Smart and Vercauteren [46, 47] described some variants of FFS and NFS. Their methods have running time $L_{p^n}(1/3, c)$ for various constants c.

1.5 Outline and contribution

The number field sieve starts with generating two polynomials, which share a common root modulo n. The running time of subsequent steps depends on the quality of the chosen polynomial pair. This thesis discusses algorithms for polynomial selection in the number field sieve. We mainly focus on polynomial selection with two polynomials, one of which is a linear polynomial. The chapters are organized as follows.

- Chapter 2 reviews the principle of number field sieve for integer factorization.
- Chapter 3 discusses some classic methods to generate polynomials for the number field sieve. We also examine some ways to quantify and compare the quality of polynomials.
- In Chapter 4, we review and analyze two standard techniques for polynomial generation due to Kleinjung [48] [49]. The polynomials generated can be further optimized in terms of size and root properties. The methods for size and root optimization are studied in Chapters 5 and 6.
- Chapter 5 discusses how to optimize the size of polynomials using translation, rotation and changing skewness. Jointly with Paul Zimmermann, we give some better methods for size optimization, focusing on sextic polynomials.
- In Chapter 6, we study the root optimization problem: finding polynomials that have many roots modulo small primes and prime powers. We describe a faster root sieve based on Hensel's lifting lemma. Furthermore, we describe a two-stage algorithm for the root optimization. In the first stage, we find polynomials with many roots modulo a product of tiny primes and prime powers. The procedure uses Hensel's liftings on a p^2-ary tree. In the second stage, we apply the improved

root sieve on congruence classes defined by the product (from the first stage). This chapter is joint work with Richard Brent and Emmanuel Thomé.

- Chapter 7 considers polynomial selection using lattice reduction. We review some recent progress on finding two non-linear polynomials for the number field sieve. In addition, we discuss how to generate degree-$(d, 1)$ polynomial pairs using lattice reduction.

- Chapter 8 summarises the thesis and suggests some areas for further work.

Most of the algorithms for polynomial selection described in this thesis, including polynomial generation, size optimization and root optimization, are implemented and can be found in CADO-NFS [6].

Chapter 2

The number field sieve

The number field sieve is the most efficient algorithm known for factoring large integers which do not have small factors. It consists of several stages including polynomial selection, sieving, filtering, linear algebra and finding square roots. In this chapter, we describe the principle of the number field sieve for integer factorization.

2.1 The idea of the number field sieve

Let n be the integer to be factored. The "congruence of squares" idea aims to find integers u, v such that $u^2 \equiv v^2 \pmod{n}$. A factor of n can be recovered by $\gcd(u-v, n)$, provided that $u \not\equiv \pm v \pmod{n}$. The number field sieve is based on this idea.

The number field sieve starts by choosing two irreducible, coprime polynomials f, g over $\mathbb{Z}$ which have a common root m in $\mathbb{Z}/n\mathbb{Z}$.

$$f(m) \equiv g(m) \equiv 0 \pmod{n}.$$

We assume that the polynomials are monic and one polynomial is linear:

$$f(x) = \sum_{i=0}^{d} c_i x^i, \quad g(x) = x - m.$$

For convenience, $f(x)$ is referred to as the *algebraic* polynomial and $g(x)$ is referred to as the *rational* polynomial. Let $\mathbb{Z}[\alpha] = \mathbb{Z}[X]/\langle f\rangle$ be the ring generated by a zero α of f. There is a ring homomorphism ϕ_α, that sends α to $m \pmod{n}$.

$$\phi_\alpha : \mathbb{Z}[\alpha] \to \mathbb{Z}/n\mathbb{Z}. \tag{2.1}$$

We want to find a set $\mathcal{S}$ of (a, b) pairs $(a, b \in \mathbb{Z})$ such that

$$\prod_{(a,b)\in\mathcal{S}} (a - b\alpha) = \gamma^2 \quad \text{and} \quad \prod_{(a,b)\in\mathcal{S}} (a - bm) = v^2 \tag{2.2}$$

for some $\gamma \in \mathbb{Z}[\alpha]$ and $v \in \mathbb{Z}$. Assume the set $\mathcal{S}$ is found and $\phi_\alpha(\gamma) \equiv u \pmod{n}$. We can see that

$$u^2 \equiv \prod_{(a,b)\in\mathcal{S}} \phi_\alpha(a - b\alpha) \equiv \prod_{(a,b)\in\mathcal{S}} (a - bm) \equiv v^2 \pmod{n}.$$

The congruence of squares $u^2 \equiv v^2 \pmod{n}$ may give a factor of n with probability at least 1/2.

2.2 The principle of the number field sieve

We outline how we can find the congruence of squares in number fields. For convenience, we first describe some notations that will be used in this chapter.

Let $f(x)$ be a monic, irreducible polynomial of degree d and $g(x) = x - m$. Let $\mathbb{Z}[\alpha]$ be the ring generated by a zero α of f. $K = \mathbb{Q}(\alpha)$ is the field of fractions of $\mathbb{Z}[\alpha]$, with K^* being its multiplicative group. Let $\mathcal{O}_K$ be the ring of algebraic integers of $\mathbb{Q}(\alpha)$.

An order is a subring of $\mathcal{O}_K$ which is finitely generated as a $\mathbb{Z}$-module of rank d. The ring $\mathbb{Z}[\alpha] \subseteq \mathcal{O}_K$ is an order. It follows that the ring of integers $\mathcal{O}_K$ in K is the unique maximal order. All other orders in K are contained in $\mathcal{O}_K$ [21].

The norm $N(\gamma)$ $(\gamma \in K)$ is the determinant of the $\mathbb{Q}$-linear map that sends $x \in K$

to $\gamma x \in K$. It is the product of all conjugates $\sigma_i(\gamma)$:

$$N(\gamma) = \prod_{i=1}^{d} \sigma_i(\gamma)$$

where σ_i's are the d-embeddings of $\mathbb{Q}(\alpha)$ into $\mathbb{C}$. The norm is a multiplicative function whose image lies in $\mathbb{Q}$. In particular, $N(\gamma) \in \mathbb{Z}$ if $\gamma \in \mathcal{O}_K$.

An integer is B-smooth if none of its prime factors is larger than B. An element $\gamma \in \mathbb{Z}[\alpha]$ is B-smooth if its norm $N(\gamma)$ is B-smooth. In the number field sieve, we are interested in elements of the form $a - b\alpha$ whose norm is given by

$$N(a - b\alpha) = b^d f(a/b) = \sum_{i=0}^{d} c_i a^i b^{d-i}.$$

If $\prod_{(a,b)\in\mathcal{S}}(a - b\alpha)$ is a square in $\mathbb{Z}[\alpha]$, its norm is a square in $\mathbb{Z}$.

The norm $\mathfrak{N}(\mathfrak{p})$ of an ideal $\mathfrak{p} \subseteq \mathbb{Z}[\alpha]$ is the cardinality of the quotient ring $\mathbb{Z}[\alpha]/\mathfrak{p}$. Let $\mathfrak{p}$ be a non-zero prime ideal of $\mathbb{Z}[\alpha]$. The quotient $\mathbb{Z}[\alpha]/\mathfrak{p}$ is a finite field. $\mathfrak{p}$ is called a first degree prime ideal if $\mathbb{Z}[\alpha]/\mathfrak{p}$ is isomorphic to $\mathbb{F}_p$ for some prime integer p. A prime ideal $\mathfrak{p}$ divides an element γ if $\mathfrak{p}$ contains γ.

Let r be a root of the polynomial $f(x) \pmod{p}$. There is a 1-1 correspondence between the pairs (p, r) and the first degree prime ideals $\mathfrak{p}$ of $\mathbb{Z}[\alpha]$, where p is the characteristic of $\mathbb{Z}[\alpha]/\mathfrak{p}$. There is a canonical ring homomorphism $\pi : \mathbb{Z}[\alpha] \to \mathbb{F}_p$ that maps α to r.

Let $\nu_p(x)$ $(x \in \mathbb{Z})$ be the exponent of the largest power of p dividing x. We define $\nu_p(0) = \infty$. Given a prime integer p, a root r of $f \pmod{p}$ and $a - b\alpha$ such that $\gcd(a, b) = 1$, we define

$$e_{p,r}(a - b\alpha) = \begin{cases} \nu_p\left(N(a - b\alpha)\right), & \text{if } a = br \pmod{p}; \\ 0, & \text{otherwise.} \end{cases}$$

We get $N(a - b\alpha) = \pm \prod_{(p,r)} p^{e_{p,r}(a-b\alpha)}$. Propositions 2.1–2.3 from [19] are useful to find the congruence of squares in Equation (2.2).

Proposition 2.1. *Given a prime ideal $\mathfrak{p}$ of $\mathbb{Z}[\alpha]$, there exists a group homomorphism $l_{\mathfrak{p}} : K^* \to \mathbb{Z}$ such that*

1. *$l_{\mathfrak{p}}(\gamma) \geq 0$ for all $\gamma \in \mathbb{Z}[\alpha]$, $\gamma \neq 0$; If $\gamma \in \mathbb{Z}[\alpha]$, then $l_{\mathfrak{p}}(\gamma) > 0 \Leftrightarrow \gamma \in \mathfrak{p}$;*

2. *$l_{\mathfrak{p}}(\gamma) = 0$ for all but finitely many $\mathfrak{p}$, and*

$$|N(\gamma)| = \prod_{\mathfrak{p}} \mathfrak{N}(\mathfrak{p})^{l_{\mathfrak{p}}(\gamma)}$$

where $\mathfrak{p}$ ranges over the set of all prime ideals of $\mathbb{Z}[\alpha]$.

If $\mathbb{Z}[\alpha] = \mathcal{O}_K$, $l_{\mathfrak{p}}$ is the exponent of the largest power of $\mathfrak{p}$ that appears in the ideal factorization of $\gamma\mathcal{O}_k$. We apply Proposition 2.1 to $\gamma = a - b\alpha$ for $a, b \in \mathbb{Z}$.

Corollary 2.2. *Let (a, b) be a coprime pair and $\mathfrak{p}$ be a prime ideal of $\mathbb{Z}[\alpha]$. If $\mathfrak{p}$ is not a first degree prime ideal of $\mathbb{Z}[\alpha]$, then $l_{\mathfrak{p}}(a - b\alpha) = 0$. If $\mathfrak{p}$ is a first degree prime ideal corresponding to some pair (p, r), then $l_{\mathfrak{p}}(a - b\alpha) = e_{p,r}(a - b\alpha)$.*

If $\mathbb{Z}[\alpha] = \mathcal{O}_K$, the factorization of $(a - b\alpha)\mathcal{O}_K$ gives first degree prime ideals. Proposition 2.3 gives a necessary condition to find a square.

Proposition 2.3. *Let $\mathcal{S}$ be a finite set of coprime pairs (a, b) such that $\prod_{(a,b)\in\mathcal{S}}(a - b\alpha)$ is a square of an element in K. For each prime integer p and root r such that $f(r) \equiv 0 \pmod{p}$, the following relation holds.*

$$\sum_{(a,b)\in\mathcal{S}} e_{p,r}(a - b\alpha) \equiv 0 \pmod{2}.$$

If $\mathbb{Z}[\alpha] = \mathcal{O}_K$, $\prod_{(a,b)\in\mathcal{S}}(a - b\alpha)\mathcal{O}_K$ is a square of an ideal in $\mathcal{O}_K$. However, this may not be true in the general case when $\mathbb{Z}[\alpha] \subset \mathcal{O}_K$. There are some further obstructions as mentioned in [19]. Even if $\prod_{(a,b)\in\mathcal{S}}(a - b\alpha)\mathcal{O}_K$ is a square of an ideal $\mathfrak{q}$ in $\mathcal{O}_k$, $\mathfrak{q}$ need not be principal. Assume further $\prod_{(a,b)\in\mathcal{S}}(a - b\alpha)\mathcal{O}_K = \gamma^2\mathcal{O}_K$ for some $\gamma \in \mathcal{O}_K$. It may not be true that $\prod_{(a,b)\in\mathcal{S}}(a - b\alpha) = \gamma^2$. Finally, even if $\prod_{(a,b)\in\mathcal{S}}(a - b\alpha) = \gamma^2$ for some $\gamma \in \mathcal{O}_K$, it is not necessary that $\gamma \in \mathbb{Z}[\alpha]$.

There are some methods to tackle these obstructions in practice. The last obstruction can be resolved in the following way. If $\prod_{(a,b)\in\mathcal{S}}(a-b\alpha)=\gamma^2$ for $\gamma\in\mathcal{O}_K$, then $\gamma f'(\alpha)\in\mathbb{Z}[\alpha]$. Hence $f'(\alpha)^2\prod_{(a,b)\in\mathcal{S}}(a-b\alpha)=f'(\alpha)^2\gamma^2$ is a square in $\mathbb{Z}[\alpha]$.

Let V be the multiplicative group consisting of γ such that $\mathfrak{l}_{\mathfrak{p}}(\gamma)$ is even for all prime ideals $\mathfrak{p}$ of $\mathbb{Z}[\alpha]$. The elements of V can be produced in the sieve. Let $V_1\subset V$ be the subgroup of elements γ such that $\gamma\mathcal{O}_k$ is a square of a fractional ideal. Let $V_2\subset V_1$ be UK^{*2} where U is the unit group. The elements in $V_2\cap\mathcal{O}_K$ are free of the second obstruction. Finally, let $V_3=K^{*2}$ be the subgroup of squares. We see that $V_3\subset V_2\subset V_1\subset V$. It can be shown that the dimension of the $\mathbb{F}_2$-vector space V/V_3 is bounded by $\log n$. Therefore, the first three obstructions are not too serious. In practice, they can be solved efficiently by introducing quadratic characters [1]. For convenience, we define the Legendre symbol on x and an odd prime p to be

$$\left(\frac{x}{p}\right)=\begin{cases}1, & \text{if } x \text{ is a quadratic residue modulo } p \text{ and } p\nmid x;\\ -1, & \text{if } x \text{ is a quadratic non-residue modulo } p \text{ and } p\nmid x;\\ 0, & \text{if } p\mid x.\end{cases}$$

Proposition 2.4. *Let $\mathcal{S}$ be a set of coprime pairs (a,b) such that $\prod_{(a,b)\in\mathcal{S}}(a-b\alpha)=\gamma^2$ is a square in $\mathcal{O}_k$. Let s be a root of $f \pmod q$ for an odd prime q such that $f'(s)\not\equiv 0 \pmod q$ and*

$$a\not\equiv bs \pmod q \text{ for all } (a,b)\in\mathcal{S}.$$

Then the product of Legendre maps is

$$\prod_{(a,b)\in\mathcal{S}}\left(\frac{a-bs}{q}\right)=1.$$

The converse of the proposition also holds. Let $\mathfrak{q}$ be a first degree prime ideal in $\mathbb{Z}[\alpha]$. The isomorphism $\mathbb{Z}[\alpha]/\mathfrak{q}\cong\mathbb{F}_q$ induces the map $G:\mathbb{Z}[\alpha]\setminus\mathfrak{q}\to\mathbb{F}_q\setminus\{0\}$. Define the character map $\chi_{\mathfrak{q}}:\mathbb{Z}[\alpha]\setminus\mathfrak{q}\to\{\pm1\}$ to be the composition of G and the Legendre map $\mathbb{F}_q\setminus\{0\}\to\{\pm1\}$. Let (q,s) be the pair corresponding to a first degree prime ideal

$\mathfrak{q}$. It follows that

$$\chi_{\mathfrak{q}}(a - b\alpha) = \left(\frac{a - bs}{q}\right).$$

The converse of the proposition shows that, if $\gamma \in \mathbb{Z}[\alpha] \setminus \{0\}$ satisfying $\chi_{\mathfrak{q}}(\gamma) = 1$ for all but finitely many first degree prime ideals $\mathfrak{q}$ not lying above γ, then γ is a square in $\mathbb{Q}(\alpha)$. A probabilistic version of the proposition shows, if the statement is true for sufficiently many such prime ideals, the element is a square with high probability. In practice, extra columns of character maps for such q's are added in the linear algebra stage to overcome the above obstructions.

Given a polynomial pair f, g, we want to find many coprime pairs $(a, b) \in \mathbb{Z}^2$ such that $N(a - b\alpha)$ and $a - bm$ are both smooth with respect to some integers B_F, B_G. Line sieving and lattice sieving [84] are commonly used to identify such pairs (a, b). The running-time of the sieving and subsequent steps depend on the quality of the chosen polynomial pair. For the moment, we assume that a polynomial pair is better if its coefficients are smaller.

Homogeneous polynomials. In practice, a non-monic, homogeneous polynomial pair is often used since its coefficients are expected to be smaller than in the monic case. We outline the resulting modifications in the number field sieve. Let the polynomial pair be

$$F(x, y) = \sum_{i=0}^{d} c_i x^i y^{d-i} \text{ and } G(x, y) = m_2 x - m_1 y.$$

The dehomogenized polynomials f, g share the common root $m_1/m_2 \pmod{n}$ and hence

$$F(m_1, m_2) = \sum_{i=0}^{d} c_i m_1^i m_2^{d-i} \equiv 0 \pmod{n}.$$

In sieving, we want to find coprime pairs $(a, b) \in \mathbb{Z}^2$ such that $F(a, b)$ and $G(a, b)$ are smooth. Let $\alpha \in \mathbb{C}$ be a root of $f(x)$. In the linear algebra stage, we want to find a set $\mathcal{S}$ such that $\prod_{(a,b)\in\mathcal{S}}(a - b\alpha)$ is a square in $\mathcal{O}_K$.

The dehomogenized polynomial f is non-monic. The root α of f is not an algebraic integer in general and hence $\mathbb{Z}[\alpha]$ may not be an order. A homomorphism (similar to that in Proposition 2.1) can be defined from an order $\mathbb{Z}[\alpha]\cap\mathbb{Z}[\alpha^{-1}]$ to $\mathbb{Z}$, which depends on the prime ideals of $\mathbb{Z}[\alpha] \cap \mathbb{Z}[\alpha^{-1}]$. We relate the norm of $a - b\alpha$ and the norm of prime ideals. The pairs p and $(r_1 : r_2)$ such that $p \mid F(r_1, r_2)$ are in 1-1 correspondence with the first degree prime ideals of $\mathbb{Z}[\alpha] \cap \mathbb{Z}[\alpha^{-1}]$. In this case, the set $\mathcal{S}$ of coprime pairs (a, b) needs to have even cardinality (see Equation (2.3)).

Congruence of squares. We describe how to construct the congruence of squares. Let $f_{c_d}(x) = F(x, c_d)$. A root $\omega \in \mathbb{C}$ of the polynomial $f_{c_d}(x)$ is an algebraic integer. For the moment, suppose we have found a set $\mathcal{S}$ such that the product $\prod_{(a,b)\in\mathcal{S}}(a - b\alpha)$ is a square. Hence

$$c_d^{\#\mathcal{S}} \prod_{(a,b)\in\mathcal{S}} (a - b\alpha) = \prod_{(a,b)\in\mathcal{S}} (c_d a - b\omega)$$

is also a square where the square root lies in $\mathcal{O}_K$. Let $f_1 = \partial f/\partial x$ and $F_1(x, y)$ be the homogeneous polynomial of $f_1(x)$. We see that

$$(F_1(\omega, c_d)/c_d)^2 \prod_{(a,b)\in\mathcal{S}} (c_d a - b\omega) = \beta^2$$

where $\beta \in \mathbb{Z}[\omega]$. In the square root stage, β is represented in the basis form $\sum_{i=0}^{d-1} \beta_i \omega^i$. On the other hand, let $u \in \mathbb{Z}$ be defined by

$$\prod_{(a,b)\in\mathcal{S}} (am_2 - bm_1) = u^2$$

and $h \in \mathbb{Z}$ be

$$h \equiv c_d^{d-2+\#\mathcal{S}/2} F_1(m_1, m_2) \pmod{n}.$$

We also define

$$l \equiv m_2^{\#\mathcal{S}/2} \tag{2.3}$$

and

$$v \equiv \sum_{i=0}^{d-1} \beta_i c_d^i m_1^i m_2^{d-1-i} \pmod{n}.$$

The congruence of the squares is given by $(hu)^2 \equiv (lv)^2 \pmod{n}$. We outline the equivalence.

Let the homomorphism $\phi_\alpha : \mathbb{Z}[\alpha] \to \mathbb{Z}/n\mathbb{Z}$ be defined by $\phi_\alpha(\alpha) = m_1/m_2 \pmod{n}$. It follows that $\phi_\alpha(m_2\omega) = c_d m_1 \pmod{n}$. The map $\phi_\alpha(\beta^2 m_2^{2(d-1)+\#\mathcal{S}})$ equals

$$\begin{aligned}
&\phi_\alpha(m_2^{2(d-1)+\#\mathcal{S}}(F_1(\omega, c_d)/c_d)^2 \prod_{(a,b)\in\mathcal{S}} (c_d a - b\omega)) \\
&= \phi_\alpha((m_2^{(d-1)} F_1(\omega, c_d)/c_d)^2 \prod_{(a,b)\in\mathcal{S}} m_2(c_d a - b\omega)).
\end{aligned}$$

We apply the homomorphism to the product. The above is equivalent to

$$\begin{aligned}
&\phi_\alpha(F_1^2(\omega m_2, c_d m_2)/c_d) \prod_{(a,b)\in\mathcal{S}} c_d(am_2 - bm_1) \\
&\equiv (c_d^{d-2} F_1(m_1, m_2))^2 \prod_{(a,b)\in\mathcal{S}} c_d(am_2 - bm_1) \\
&\equiv (c_d^{d-2+\#\mathcal{S}/2} F_1(m_1, m_2))^2 u^2 \pmod{n}
\end{aligned}$$

and this is $h^2u^2 \pmod{n}$. On the other hand, $\beta = \sum_{i=0}^{d-1} \beta_i w^i$ gives

$$\phi_\alpha(\beta^2 m_2^{2(d-1)+\#\mathcal{S}}) \equiv (\sum_{i=0}^{d-1} \beta_i c_d^i m_1^i m_2^{(d-1-i)})^2\, l^2 \equiv v^2 l^2 \pmod{n}.$$

Therefore, we have the congruence $(hu)^2 \equiv (lv)^2 \pmod{n}$.

2.3 Complexity and parameters

We explain the heuristic running-time of the number field sieve for factoring. In sieving, we want to find pairs whose norm is smooth[*]. The first degree prime ideals of norm bounded by B comprise the algebraic factor base. The primes bounded by B comprise the rational factor base. The factor base has size $B^{1+o(1)}$. In linear algebra, we work over $\mathbb{F}_2$ and try to find a non-trivial linear dependency. This requires that the number of smooth pairs is comparable to the size of the factor base. We consider the probability

[*]Let the algebraic and rational smoothness bounds be the same. We also assume f, g are monic.

that a number of size x is B-smooth.

Let $\Psi(x, x^{1/u})$ be the number of $x^{1/u}$-smooth integers below x for some u. The Dickman-de Bruijn function $\rho(u)$ [39] is often used to estimate $\Psi(x, x^{1/u})$. It can be shown that

$$\lim_{x\to\infty} \frac{\Psi(x, x^{1/u})}{x} = \rho(u).$$

The Dickman-de Bruijn function satisfies the differential equation

$$u\rho'(u) + \rho(u-1) = 0, \quad \rho(u) = 1 \text{ for } 0 \le u \le 1.$$

It may be shown that ρ has the asymptotic estimate

$$\log(\rho(u)) = -(1 + o(1))u \log u \text{ as } u \to \infty.$$

For practical purposes, the frequency of smooth numbers can be approximated by the following theorem [43].

Theorem 2.5. *For any fixed $\epsilon > 0$, we have*

$$\Psi(x, x^{1/u}) = xu^{-u(1+o(1))}$$

as $x^{1/u}$ and u tends to infinity, uniformly in the region $x \ge u^{u/(1-\epsilon)}$.

In sieving, we find smooth polynomial values $|N(a - b\alpha)(a - bm)|$. We assume a, b are bounded by $\{(a, b) \mid |a| \le U,\ 0 < b \le U\}$. The polynomial values are bounded by $2dn^{2/d}U^{d+1}$ since

$$|N(a - b\alpha)| \le (d + 1)n^{1/d}U^d, \quad |a - bm| \le n^{1/d}U.$$

We also make the key assumption that polynomial values behave like random integers of similar size (though see §3.2.3 and Chapter 6). We have $2U^2$ integers bounded by $2dn^{2/d}U^{d+1}$. Buhler, Lenstra and Pomerance [19] show that, when U^2 integers of size $2dn^{2/d}U^{d+1}$ are generated with the property that the number $\Psi(x, B)$ of B-smooth integers ($x = 2dn^{2/d}U^{d+1}$) satisfies the bound

$$\frac{U^2\Psi(x,B)}{x} \geq g(B)$$

for some g such that $g(B) = B^{1+o(1)}$ as $B \to \infty$, a lower bound for U is given by

$$U = \exp\left[(1/2 + o(1))\left(d\log d + \sqrt{(d\log d)^2 + 4\log(n^{1/d})\log\log(n^{1/d})}\right)\right]$$

and $B \approx U$. The function $g(B)$ is roughly the size of factor base. The theorem gives a lower bound on U, when the number of smooth integers is more than the size of the factor base.

The sieve across region $2U^2$ takes $U^{2+o(1)}$ time. In the linear algebra stage, the sparse matrix has $B^{1+o(1)}$ non-zero entries. It takes $B^{2+o(1)}$ time to find some dependencies. Sieving and linear algebra are the dominating steps in the number field sieve and their asymptotic running-time is similar. Therefore, the number field sieve runs in time $U^{2+o(1)}$ when U, B are

$$\exp\left[(1/2 + o(1))\left(d\log d + \sqrt{(d\log d)^2 + 4\log(n^{1/d})\log\log(n^{1/d})}\right)\right]$$

as $n \to \infty$. The optimal degree d is

$$d = (3^{1/3} + o(1))\left(\frac{\log n}{\log\log n}\right)^{1/3}$$

which gives

$$U = B = L_n(1/3, (8/9)^{1/3})$$

and running-time

$$L_n(1/3, (64/9)^{1/3})$$

as $n \to \infty$. The sieving region $2U^2$ is about $L_n(1/3, (64/9)^{1/3})$.

2.4 Stages of the number field sieve

We outline the main stages in the number field sieve: polynomial selection, sieving, filtering, linear algebra and finding square roots.

2.4.1 Polynomial selection

The polynomial selection generates two irreducible and coprime polynomials f and g over $\mathbb{Z}$ which have a common root m modulo n. In practice, non-monic, homogenized polynomials $F(x, y)$ and $G(x, y)$ are often used.

In the next stage (sieving), we want to find many coprime pairs $(a, b) \in \mathbb{Z}^2$ such that the integers $F(a, b)G(a, b)$ are smooth with respect to some smoothness bound B. The running-time of sieving depends on the smoothness of the polynomial values $F(a, b)G(a, b)$. We assume that the polynomial values behave like random integers of size $|F(a, b)G(a, b)|$. In practice, the chance of smoothness may be better since two factors $F(a, b)$ and $G(a, b)$ are already known. The number of relations (smooth coprime pairs) can be approximated by

$$\frac{6}{\pi^2} \iint\limits_{\Omega} \rho\left(\frac{\log|F(x,y)|}{\log B}\right) \rho\left(\frac{\log|G(x,y)|}{\log B}\right) \mathrm{d}x\,\mathrm{d}y$$

where Ω is the sieving region of (a, b). Therefore, we want to choose a polynomial pair such that the size of $|F(a, b)|$ and $|G(a, b)|$ is small on average over all (a, b). This requires that the coefficients of the polynomials are small in absolute value. Since the running-time of the sieving depends on the quality of the chosen polynomial pair, many polynomial pairs will be considered in order to find a good one.

The standard method for polynomial selection is to expand kn in base (m_1, m_2) so $kn = \sum_{i=0}^{d} c_i m_1^i m_2^{d-i}$ where k is a small constant. There are various methods to find such an expansion [19, 73, 48, 49]. We will describe the details in Chapter 3.

2.4.2 Sieving

We describe some methods to find smooth pairs (a, b). The straightforward way is to use line sieving (or classical sieving). We fix b and solve for a in $a/b \equiv r \pmod{p}$, where r is a root of $f(x) \pmod{p}$. Then we sieve across $a' \equiv a \pmod{p}$ in a range. We consider all p's in the factor base and then move to the next b.

Pollard [84] described lattice sieving methods by rows and vectors. He only sieves over pairs (a, b) where $f(a/b)$ is known to be divisible by some special large prime q's. Lattice sieving is more efficient than line sieving (for large integers) in practice.

We first find a set of primes q's for which $f(x) \equiv 0 \pmod{q}$ have solutions. q is often chosen to be larger than the smoothness bound. Let s be a root of $f(x) \pmod{q}$. The pairs (a, b) such that q divides $f(a/b)$ form a lattice generated by vectors $e_1 = (q, 0)$ and $e_2 = (s, 1)$. We compute a reduced basis $e'_1 = (e'_{11}, e'_{12})$ and $e'_2 = (e'_{21}, e'_{22})$.

Lattice sieving by rows is similar to line sieving on the q-lattice. Let r be a solution of $f(x) \pmod{p}$ for p in the factor base. We fix j_0 and solve for i_0 in

$$\frac{i_0 e'_{11} + j_0 e'_{21}}{i_0 e'_{12} + j_0 e'_{22}} \equiv r \pmod{p}.$$

The sieve is done for $i \equiv i_0 \pmod{p}$ in a range. As for line sieving, we consider all p's in the factor base and then proceed to next j.

Lattice sieve by vectors considers the points on a p-sublattice of the q-lattice. Let r be a solution of $f(x) \pmod{p}$ for p in the factor base. A pair (i, j) on the q-lattice leads to a root of $f(x) \pmod{p}$ if

$$\frac{i e'_{11} + j e'_{21}}{i e'_{12} + j e'_{22}} \equiv r \pmod{p}.$$

It can be expressed as

$$\frac{i}{j} \equiv -\frac{e'_{21} - e'_{22} r}{e'_{11} - e'_{12} r} \equiv \tilde{r} \pmod{p}.$$

All points (a, b) on the p-sublattice can be generated by $v_1 = (p, 0)$ and $v_2 = (\tilde{r}, 1)$. We compute a reduced basis v'_1 and v'_2. The linear combinations of v'_1 and v'_2 give points

(a, b) such that $pq \mid f(a/b)$. For large p's, the initialization cost could be more expensive than the sieve. Franke and Kleinjung [51] described an efficient way to compute the indices in the sieve. One can also use the bucket sieving [3] to reduce cache misses for larger p's.

2.4.3 Filtering

After sieving, the relations contain many duplicate (a, b) pairs. We want to remove the duplicates. Standard hash-coding techniques can be used to identify duplicates. For a large relation file, the memory requirement of the hash table could be huge. We can partition the relation file and detect duplications on each partition.

We are interested in an even number of occurrences of the first degree prime ideals. The prime ideals which appear only once (singletons) can be discarded. In practice, removing singletons may lead to new singletons. Hence several passes of singleton-removal are often applied. The number of remaining relations should be more than the number of prime ideals after singleton-removal. For large integer factorization, we could also reduce the memory usage by partitioning the prime ideals and taking several passes.

In sieving, we often use large prime variants that allow relations with a few large primes. We need to combine the relations with matched large prime ideals. Further, we can reduce the matrix size by combining relations, at the price of increasing the density of the matrix [20].

2.4.4 Linear algebra and square root

Linear algebra is currently a major bottleneck for factoring large integers. The standard Gaussian elimination can be modified for use by the number field sieve. However, the elimination process can introduce many nonzero entries (fill-in's) during row operations. Structured Gaussian elimination [55] is a more practical way to solve the (very) sparse system in number field sieve. The columns corresponding to small prime ideals are dense, while those for large primes are sparse. The idea is to start the elimination on the sparse parts. If the sparse parts are eliminated first, one can get a reduced matrix, which is slightly more dense but much smaller.

Coppersmith [24] described the block Lanzcos algorithm. It has comparable running time as the structured Gaussian elimination and much smaller space requirements. Montgomery [68] gave another block Lanczos method, which constructs the orthogonal vectors differently. It is one of the most efficient methods known for finding nullspaces of sparse matrices. In practice, a parallelised version of Montgomery's method can be used for large integers. We distribute the block Lanczos solution over several processors. However, the communication overheads could dominate. Another algorithm proposed by Coppersmith, the block Wiedemann method, is easier to parallelise than the block Lanczos method. The block Wiedemann method generates a sequence by linear recursion (on a random vector) and finds relations in Krylov subspaces using the Berlekamp-Massey algorithm. One can parallelise the algorithm by computing sequences with different random vectors.

In the final step, we want to find the square roots (of products) of algebraic numbers. Couveignes [26] described a method based on the Chinese Remainder Theorem. It requires that the degree of f is odd. Montgomery [66] gave an algorithm which works for general polynomials. It uses the known ideal factorization of the algebraic integers. Nguyen [76] discussed a variant of Montgomery's method. Overall, the square root appears to be an easy step in the number field sieve due to the algorithms described above.

Chapter 3

Polynomial selection: general

The running-time of sieving and subsequent steps depends on the quality of the chosen polynomial pair. Many polynomials are generated to find a good one. In this chapter, we give a brief discussion of polynomial selection.

We describe some classic ways to generate polynomials. We also discuss some methods to quantify and compare the quality of polynomials. Finally, we consider how to further optimize the polynomials in the number field sieve.

3.1 Base-m expansion

The base-m expansion [19, 73] of n can give a polynomial $f(x)$ such that $f(m) \equiv 0 \pmod{n}$. In the simple case, the linear polynomial is given by a monic polynomial $g(x) = x - m$. Given n and d, we choose m to be close to $n^{1/d}$. We obtain a monic $f(x)$ by expanding n in base m where

$$n = m^d + c_{d-1}m^{d-1} + \cdots + c_1 m + c_0$$

such that each $|c_i| \leq m/2$. Then we define

$$f(x) = x^d + c_{d-1}x^{d-1} + \cdots + c_1 x + c_0.$$

We attempt to reduce the size of the coefficients of $f(x)$, while keeping the property $f(m) \equiv 0 \pmod{n}$. Polynomials with smaller (in size) coefficients heuristically give smaller polynomial values.

A better way is to use a non-monic $f(x)$ so

$$n = c_d m^d + c_{d-1} m^{d-1} + \cdots + c_1 m + c_0$$

where c_d is small and has only (very) small prime factors. Murphy [73] also described a method to generate polynomials with small c_{d-1}'s. Given n, c_d, the coefficient c_{d-1} can be controlled using

$$(n - c_d m^d)/m^{d-1} \approx c_{d-1} + c_{d-2}/m.$$

We can choose a root m which is close to $(n/c_d)^{1/d}$. We leave the size of c_{d-2} to chance. If c_{d-2} is not small, we try another c_d or m. Otherwise, we keep the polynomial pair for further inspection and optimization. We will discuss some better methods for polynomial selection in Chapters 4–6.

Remark 3.1. In Chapters 4–6, we study the polynomial selection with two polynomials, one of which is linear. In Chapter 7, we discuss the polynomial selection with two non-linear polynomials based on the papers [34, 74, 73, 87]. We do not cover Coppersmith's multiple polynomial number field sieve (MNFS) [23] in this thesis.

3.2 Quantifying the quality of polynomials

In this section, we discuss some methods to estimate and compare the quality of polynomials*.

It is desirable that the polynomial pair can produce many smooth integers across the sieve region. This heuristically requires that the size of polynomial values is small in general. In addition, one can choose an algebraic polynomial $f(x)$ which has many

*The "quality of polynomials" means the same as the "quality of polynomial pairs".

roots modulo small prime powers. Then the polynomial values are likely to be divisible by small prime powers. This may increase the smoothness chance for polynomial values.

3.2.1 Sieving test

A sieving experiment over short intervals is a relatively accurate method to compare polynomial pairs. It is often used to compare several polynomial candidates in the final stage of the polynomial selection.

Ekkelkamp [32] also described a method for predicting the number of relations needed in the sieving. The method conducts a short sieving test and simulates relations based on the test results. Experiments show that the prediction of the number of relations is within 2% of the number of relations needed in the actual factorization.

3.2.2 Size property

Let (a, b) be pairs of relatively prime integers in the sieving region Ω. For the moment, we assume that a rectangular sieving region is used where $|a| \leq U$ and $0 < b \leq U$. We also assume that polynomial values $|F(a, b)|$ and $|G(a, b)|$ behave like random integers of similar size. The polynomial values are B-smooth with probability

$$\rho\left(\frac{\log|F(a,b)|}{\log B}\right) \text{ and } \rho\left(\frac{\log|G(a,b)|}{\log B}\right).$$

An approximation for the number of sieving reports (coprime pairs that lead to smooth polynomial values) is given by

$$\frac{6}{\pi^2}\iint\limits_{\Omega} \rho\left(\frac{\log|F(x,y)|}{\log B}\right) \rho\left(\frac{\log|G(x,y)|}{\log B}\right) \mathrm{d}x\,\mathrm{d}y.$$

The multiplier $6/\pi^2$ accounts for the probability of a, b being relatively prime.

Since G is a linear polynomial, we may assume that $\log(|G(a, b)|)$ does not vary much across the sieving region. Hence, we omit G in the size estimate for the moment.

A simplified approximation to compare polynomials is

$$\iint\limits_{\Omega} \rho\left(\frac{\log|F(x,y)|}{\log B}\right) \mathrm{d}x\,\mathrm{d}y. \tag{3.1}$$

Skewness. The base-m method gives polynomials whose coefficients are $O(n^{1/(d+1)})$. The leading coefficients c_d and c_{d-1} are much smaller than $n^{1/(d+1)}$. The coefficient c_{d-2} is slightly smaller than $n^{1/(d+1)}$. It is often better to use a skewed sieving region where the sieving bounds for a, b have ratio s. We consider the cases for $d = 5$ and 6.

Let the sieving bounds for a, b be $|a| \leq U$ and $0 < b \leq U$. Hence $s \approx 1$. The polynomial values are dominated by $c_i U^d$ for $i < d-2$, where $c_i = O(n^{1/(d+1)})$. It is often better to choose s to be larger than 1, while keeping the area of the sieving region $2U^2$. The sieving bounds become $|a| \leq U\sqrt{s}$ and $0 < b \leq U/\sqrt{s}$. Each monomial in the polynomial is bounded by $c_i U^d s^{i-d/2}$.

The polynomial values are dominated by terms $c_{d-3}U^d s^{d/2-3}$ and (or) $c_i U^d s^{i-d/2}$ for $i \geq d-2$. As an example, we can control the terms involving c_d and c_{d-3}, forcing $c_{d-3}s^{d/2-3} \approx c_d s^{d/2}$. Hence $s \approx (c_{d-3}/c_d)^{1/3}$.

If $d = 5$, $c_{d-3}U^d s^{d/2-3} = c_2 s^{-1/2} U^5$ and $c_d U^d s^{d/2} = c_5 U^5 s^{5/2} \approx c_2 s^{-1/2} U^5$. The terms involving c_3 and c_4 are both bounded by $c_3 s^{1/2} U^5$, since $c_4 \approx c_5$. If c_3 is small, the term $c_2 s^{-1/2} U^5$ (bounding the terms involving c_5 and c_2) is smaller if we choose $s \approx (c_2/c_5)^{1/3}$.

If c_3 is large, we can equalize the terms involving c_3 and c_2. The dominating terms are bounded by $c_2 s^{-1/2} U^5$ where $s \approx c_2/c_3$.

If $d = 6$, the term $c_3 U^6$ dominates the polynomial values. We will discuss some ways to optimize degree six polynomials in Chapter 5.

Size estimates. In the integral (3.1), computing ρ is time-consuming, especially if there are many candidates. We can use some coarser approximations.

$\rho(u)$ is a decreasing function of u. We can choose the polynomial pair such that the average size of $|F(a,b)|$ is small. This roughly requires that the coefficients of the polynomials are small in absolute value. We discuss several estimates to compare polynomials.

We can compare polynomials by the logarithmic average of polynomial values across the sieving region.

$$\log \left(\iint_{\Omega} |F(x, y)| \, \mathrm{d}x \, \mathrm{d}y \right).$$

For computational convenience, one can use the logarithmic L^2-norm for polynomial $F(x, y)$.

$$\log L^2(F) = \frac{1}{2} \log \left(\iint_{\Omega} F^2(x, y) \, \mathrm{d}x \, \mathrm{d}y \right). \tag{3.2}$$

The logarithmic L^2-norm is influenced by the skewness and the location of real roots. The integral in (3.2) can be expressed as a polynomial in the coefficients of $F(x, y)$.

One can also change the range and shape of the integral region (the domain Ω), while keeping the skewness. We consider a modified logarithmic L^2-norm defined by

$$\frac{1}{2} \log \left(s^{-d} \int_{-1}^{1} \int_{-1}^{1} F^2(xs, y) \, \mathrm{d}x \, \mathrm{d}y \right) \tag{3.3}$$

where s is the skewness of sieving region.

If computing the integral is still too expensive, we can use the logarithmic l^p-norm on the (absolute values of) coefficients of $F(x, y)$. For instance, the logarithmic l^∞ and l^1-norm are defined by

$$\log l^\infty(F) = \log \left(\max_{0 \le i \le d} |c_i s^{i - \frac{d}{2}}| \right)$$

and

$$\log l^1(F) = \log \left(\sum_{i=0}^{d} |c_i s^{i - \frac{d}{2}}| \right).$$

In practice, c_{d-3} to c_0 are often similar and hence we can also compare individual coefficients c_i for $i \ge d - 2$. However, the l^1 and l^∞-norm are not sensitive to the location of real roots.

A more accurate estimate generally takes a longer time. Various levels of approximations can be used in different stages of polynomial selection, depending on the efficiency-accuracy tradeoff.

Polynomial selection often starts with generating a large number of polynomials. We want to quickly compare the polynomials and discard the worst ones. The l^1-norm can be used to rapidly identify such polynomials. For instance, let $d = 6$ and allow coefficients c_0 to c_3 to be in $O(n^{1/(d+1)})$. If c_6, c_5 are trivially small, we can focus on the size of c_4.

The polynomial norm also depends on the skewness. We can optimize the skewness with respect to a particular norm. If we assume that the optimal skewness with respect to various norms does not differ too much, the optimized l^1 and l^∞-norms roughly correlate with each other. We can bound the two norms by $l^\infty \leq l^1 \leq (d+1)\, l^\infty$.

3.2.3 Root property

If a polynomial $f(x)$ has many roots modulo small prime powers, the polynomial values may behave more smoothly than random integers of about the same size. Boender, Brent, Montgomery and Murphy [8, 72, 73, 74] described some quantitative measures of this effect (root property).

Expected p-valuation. Let p be a fixed prime. Let $\nu_p(x)$ be the exponent of the largest power of p dividing x and $\nu_p(0) = \infty$. Let S be a set of integers. We use (the same) notation $\nu_p(S)$ to denote the expected p-valuation of $s \in S$.

Let S be a set of uniformly distributed random integers. For a fixed prime p, the expected p-valuation $\nu_p(S)$ is

$$1 \cdot \left(\frac{1}{p} - \frac{1}{p^2}\right) + 2 \cdot \left(\frac{1}{p^2} - \frac{1}{p^3}\right) + \cdots = \frac{1}{p-1}.$$

Thus, in an informal (logarithmic) sense, an integer s in S contains an expected power $p^{1/(p-1)}$.

Let S be a set of polynomial values $f(x)$. We use (the same) notation $\nu_p(S)$ (or $\nu_p(f)$) to denote the expected p-valuation of the polynomial values S. We assume the polynomial values behave like uniformly random integers. Hensel's lemma gives conditions when a root of $f \pmod{p^e}$ can be lifted to a root of $f \pmod{p^{e+1}}$.

Lemma 3.2 (Hensel's lemma)**.** *Let r_1 be a root of $f(x)$ modulo an odd prime p.*

1. *If r_1 is a simple root, $f(x) \pmod{p^e}$ has an unique root $r_e \equiv r_1 \pmod{p}$ for each $e > 1$.*
2. *If r_e is a multiple root*[†] *of $f \pmod{p^e}$ for $e \geq 1$, there are two possible cases. If $p^{e+1} \mid f(r_e)$, then $\forall\, i \in [0, p)$, $p^{e+1} \mid f(r_e + i\, p^e)$. If $p^{e+1} \nmid f(r_e)$, r_e cannot be lifted to a root modulo p^{e+1}.*

There are two cases. First, suppose $p \nmid \Delta$, the discriminant of $f(x)$. p is an unramified[‡] prime. Then $f(x) \pmod{p}$ has only simple roots. Let n_p be the number of roots. The expected p-valuation of polynomial values is $\nu_p(f) = n_p/(p-1)$.

The second case is when $p \mid \Delta$. Here one may get multiple roots. We can count the number of lifted roots for the expected p-valuation. For the moment, we consider p to be unramified.

In the number field sieve, we want to know the expected p-valuation of homogeneous polynomial values $F(a, b)$, where (a, b) is a pair of coprime integers, and $F(x, y)$ is the homogenous polynomial corresponding to $f(x)$. We discuss the roots of $F(x, y)$. Let $p \mid F(a, b)$ for some coprime integers a and b. Then there are two cases: either $p \nmid b$ and $f(a/b) \equiv 0 \pmod{p}$, or $p \mid b$ and $p \mid c_d$.

If $p \mid F(a, b)$ and $p \nmid b$, there exists $r \equiv a/b \pmod{p}$ that that $p \mid f(r)$. We can map these pairs (a, b) to the points on a projective line $\mathbf{P}^1(\mathbb{F}_p)$. The points on $\mathbf{P}^1(\mathbb{F}_p)$ are defined by the equivalence classes $(x : y) = (x/y \pmod{p} : 1)$ when $p \nmid y$, and $(x : y) = (1 : 0)$ (the point at infinity) when $p \mid y$ and $p \nmid x$. We call the point $(a : b)$ on $\mathbf{P}^1(\mathbb{F}_p)$ an *affine* root of $F(x, y) \pmod{p}$ if $p \mid F(a, b)$ and $p \nmid b$. All points except the point at infinity on $\mathbf{P}^1(\mathbb{F}_p)$ are the affine roots of $F(x, y) \pmod{p}$.

[†]We say that r_e is a multiple root of $f \pmod{p^e}$ if $f'(r_e) \equiv 0 \pmod{p}$.

[‡]A prime number p is said to be unramified in the number field K if the prime ideal factorization of p has no repeated prime ideal factors.

If $p \mid F(a,b)$ and $p \mid b$, it can be shown that $p \mid c_d$ and $p \nmid a$ since a is coprime to b. In this case, these pairs (a,b) correspond to the point at infinity $(1:0)$ on $\mathbf{P}^1(\mathbb{F}_p)$ and is referred to as the *projective* root of $F(x,y) \pmod p$.

Let the sample space be pairs $(a,b) \in \mathbb{F}_p \times \mathbb{F}_p$ excluding those (a,b) for which $p \mid a,\ p \mid b$ simultaneously. $F(x,y)$ can have p possible affine roots and 1 projective root. Each affine root $(r:1)$ (or simply r) relates to $p-1$ equivalent pairs (a,b) where $a/b \equiv r \pmod p$. The projective root $(1:0)$ relates to $p-1$ equivalent pairs $(a,0)$ for $a \in \mathbb{F}_p \setminus \{0\}$. A coprime pair (a,b) chosen at random falls into a particular class of roots with probability $1/(p+1)$.

We extend the sample space to pairs $(a,b) \in \mathbb{Z}/p^e\mathbb{Z} \times \mathbb{Z}/p^e\mathbb{Z}$ excluding those (a,b) such that $p \mid a,\ p \mid b$ simultaneously. Then (a,b) maps to an equivalence class $(a:b)$ on the projective line over the ring $\mathbb{Z}/p^e\mathbb{Z}$. Let $p^e \mid F(a,b)$ for some coprime integers a,b and some integer e. Then there are two cases: either $p \nmid b$ and $f(a/b) \equiv 0 \pmod{p^e}$ or $p \mid b$ and $h(b/a) \equiv 0 \pmod{p^e}$ where $h(x) = x^d f(1/x)$. $F(x,y)$ can have p^e possible affine roots, each of which relates to $p^e - p^{e-1}$ equivalent (a,b) pairs. Given $p \mid b,\ p \nmid a$, we partition (a,b) into equivalent classes defined by $(1, b/a \pmod{p^e})$ and call each class a projective root. There are p^{e-1} possible projective roots. Each projective root relates to $p^e - p^{e-1}$ equivalent (a,b) pairs.

Hence there are $p^e + p^{e-1}$ roots where each relates to $p^e - p^{e-1}$ equivalent (a,b) pairs. A coprime pair (a,b) chosen at random maps to a particular root with probability $1/(p^{e-1}(p+1))$.

Given an unramified p, let $F(x,y) \pmod p$ have n_p affine and projective roots. The expected p-valuation $\nu_p(F)$ is

$$\nu_p(F) = \sum_{e=1}^{\infty} e \, \frac{n_p}{p^{e-1}(p+1)} \left(1 - \frac{1}{p}\right) = \frac{n_p p}{p^2 - 1}. \tag{3.4}$$

The primes p's we discussed here only give simple roots (of $F(x,y) \pmod p$) and hence the average p-valuation admits a geometric representation. For multiple roots, the contribution of the lifted roots is more complicated. One can count the number of roots modulo p^e for $e \leq d$. The number of (multiple) roots over small primes can have strong impacts on the expected p-valuation. Therefore, the above geometric formula should be used with caution for small primes. We describe an example.

Let f, g be the polynomial pair A_{768} in Appendix A. A_{768} is generated by Msieve [78] (followed by optimization carried out in CADO-NFS [6]) in an online collaborative search [77]. It is an alternative (but slightly worse) polynomial for RSA-768 compared to the polynomial actually used in the factorization [50]. For each prime between 2 and 17, we generate 10^5 random pairs (a, b). We only consider the coprime pairs (about $(6/\pi^2) \cdot 10^5$). We compare the total p-valuation between the estimates and the actual values in Table 3.1.

Primes	$\#(a,b)$	Actual	Estimate	Better Estimate
2	60774	235509	81032	235500
3	60673	121858	68258	121346
5	60906	51140	38067	50755
7	61005	41058	26690	40670
11	60888	22367	22326	22326
13	60474	13829	14039	13679
17	60815	7114	7180	7180

Table 3.1: Total p-valuation

The second column of Table 3.1 denotes number of coprime pairs. The third column is the actual total p-valuation: the sum of p-valuation over all $F(a, b)$'s. Equation (3.4) assumes that roots are simple. The column "Estimate" is the estimate in Equation (3.4) multiplied by $\#(a, b)$. In the column "Better Estimate", we consider $n_{p,e}$, the number of simple and multiple roots for each p^e for $e \leq d$. The better estimate is

$$\nu_p(F) = \frac{1}{p+1} \sum_{e=1}^{d} \frac{n_{p,e}}{p^{e-1}}. \tag{3.5}$$

In Equation (3.5), we assume the lifting pattern becomes stationary for $e \geq d$. The primes $2, 3, 5, 7, 13$ ramify. Table 3.1 shows that, for ramified small primes, it is necessary to examine the lifted roots instead of simply using Equation (3.4).

Remark 3.3. Given a set of coprime (a, b) pairs, we assume that $(a \pmod{p^e} : b$

(mod p^e)) are distributed uniformly over the set of (affine or projective) roots. In practice (sieving), the distribution is not uniform (the pairs (a, b) corresponding to the polynomial roots occur more frequently). We give some experimental data. Let $f(x)$ be the algebraic polynomial used in Table 3.1. We consider a sieving test consisting of 300000 relations. The (a, b) pairs in the relations give various classes of roots a/b (mod p) in Table 3.2.

For each (a, b) in the relation file and a prime p, we record either a/b (mod p) or ∞ if $p \mid b$. We count the occurrences of a/b (mod p) and ∞. The bottom row "Roots of $f(x)$" gives the roots of $f(x)$ modulo each p. For instance, the polynomial $f(x)$ has roots $0, 3, 5$ modulo 7. The (a, b) pairs in the relation file occur more frequently at $0, 3, 5$.

a/b (mod p)	2	3	5	7	11
∞	90385	82352	39073	29588	21467
0	134962	66265	51927	52608	30749
1	74653	63133	52118	29490	21377
2	-	88250	39513	29695	28416
3	-	-	68165	50118	21280
4	-	-	49204	38077	30297
5	-	-	-	40535	21459
6	-	-	-	29889	30186
7	-	-	-	-	21513
8	-	-	-	-	21399
9	-	-	-	-	21571
10	-	-	-	-	30286
Roots of $f(x)$	0	$0, 2$	$0, 1, 3$	$0, 3, 5$	$4, 6, 10$

Table 3.2: Distribution of roots in sieving

In this thesis, we assume that the pairs fall into each class (of root) uniformly.

Cumulative expected p-valuation. Murphy defined the $\alpha(F)$ function to compare the cumulative expected p-valuation of polynomial values to random integers of similar size. $\alpha(F)$ can be considered as the logarithmic benefit of using polynomials values compared to using random integers.

$$\alpha(F) = \sum_{\substack{p \le B \\ p \text{ prime}}} \left(\frac{1}{p-1} - \frac{n_p p}{p^2-1} \right) \log p = \sum_{\substack{p \le B \\ p \text{ prime}}} \left(1 - \frac{n_p p}{p+1} \right) \frac{\log p}{p-1}.$$

For ramified p, we can replace $(n_p p)/(p^2-1)$ by $\nu_p(F)$, the actual p-valuation. On average, n_p is 1 for random, monic polynomials for unramified p's, due to the Chebotarev density theorem [75]. In the number field sieve, $\alpha(F)$ is often negative since we are interested in the case when $F(x, y)$ has more than one root.

Expected minimum $\alpha(F)$. We want to estimate the expected minimum $\alpha(F)$ after K polynomials are chosen. Emmanuel Thomé described a method (personal communication) to estimate the expected minimum of $\alpha(F)$ using order statistics. We assume that the $\alpha(F)$ values of random polynomials follow a standard Gaussian (normal) distribution $N(\mu, \sigma^2)$ (see Figure 3.1).

Let $\Phi(x)$ be the cumulative density function for the standard $N(0,1)$ normal distribution $\phi(x)$ where

$$\phi(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}, \quad \Phi(x) = \frac{1}{2} \left(1 + e\left(\frac{x}{\sqrt{2}} \right) \right).$$

In $\Phi(x)$, $e(x)$ is the error function [36] defined by

$$e(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} \, \mathrm{d}t.$$

The probability distribution for the minimum order statistic is given by

$$p_K(x) = K \, (1 - \Phi(x))^{K-1} \, \phi(x)$$

where K is the cardinality of the sample set. To compute the expected value of the minimum order statistic, we can use numerical integration. Alternatively, we can use an asymptotic approximation [27, 79] for the expected value of the minimum order statistic of the normal distribution (a more accurate estimate can be found in [90]).

$$\mu - \sigma\left(\sqrt{2\log K} - \frac{\log(\log K) + 1.3766}{2\sqrt{2\log K}}\right). \tag{3.6}$$

In practice, we need to estimate the parameters μ, σ of the actual distribution. We examine a data set of 10^7 polynomials for RSA-768. The polynomials are generated by CADO-NFS [6] and Msieve [78] [77]. They are optimized in terms of size, as discussed later.

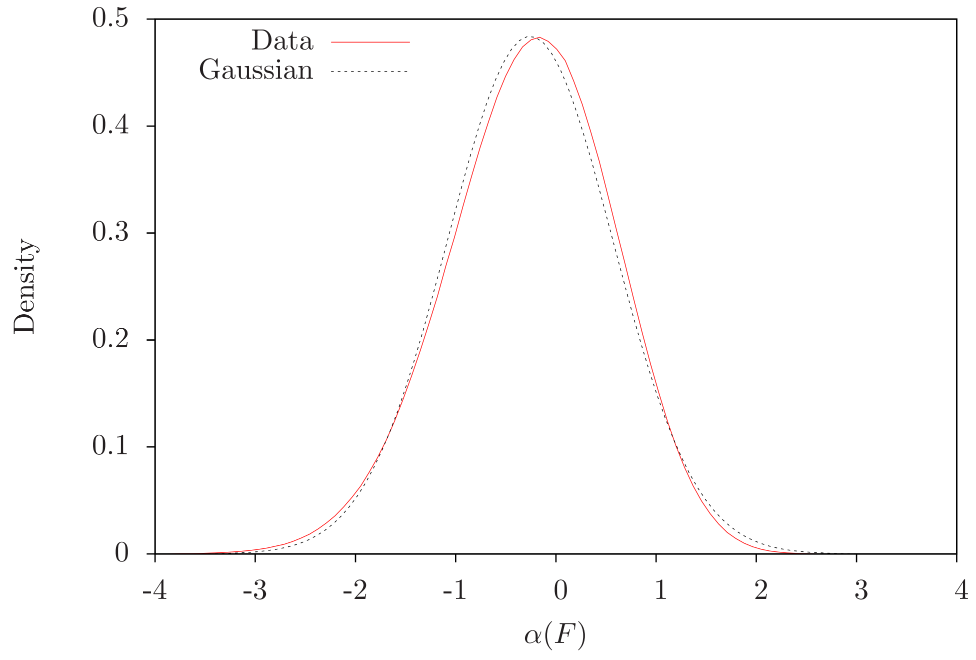

Figure 3.1: Distribution of α

The data has mean $\mu = -0.257$ and standard deviation $\sigma = 0.824$. In Figure 3.1, we show the density estimate of the data. The estimated distribution of $\alpha(F)$ is close to a Gaussian distribution with parameters μ, σ. We can use this Gaussian distribution to compute the expected value of the minimum order statistic.

Combined L^2 score. The logarithmic L^2-norm in Equation (3.2) can be modified to take the root property into account. Since the $\alpha(F)$ function affects the polynomial size on logarithmic scale, the combined function is defined as follows:

$$\alpha(F) + \frac{1}{2}\log\left(\iint\limits_{\Omega} F^2(x,y)\,\mathrm{d}x\,\mathrm{d}y\right).$$

In practice, the combined score seems to be inaccurate (a more accurate score is given by Equation (3.7)). Polynomials with similar combined scores can differ much in the test sieving. Thus, the combined score is often used as a threshold to decide which polynomials should be retained for further optimization.

3.2.4 Murphy's $\mathbb{E}$ function

After polynomial optimization, there might be too many polynomials to conduct sieving experiments. Murphy described a way to identify the best polynomials with a reliable ranking function.

For accuracy, both the algebraic polynomial $f(x)$ and the linear polynomial $g(x)$ are considered. An estimate for the number of sieving reports is given by

$$\frac{6}{\pi^2}\int_{\Omega}\rho\left(\frac{\log|F(x,y)|}{\log B_1}\right)\rho\left(\frac{\log|G(x,y)|}{\log B_2}\right)\mathrm{d}x\,\mathrm{d}y.$$

The root property is considered to give a refined estimate

$$\frac{6}{\pi^2}\int_{\Omega}\rho\left(\frac{\log|F(x,y)|+\alpha(F)}{\log B_1}\right)\rho\left(\frac{\log|G(x,y)|+\alpha(G)}{\log B_2}\right)\mathrm{d}x\,\mathrm{d}y. \tag{3.7}$$

The Dickman-de Bruijn function $\rho(x)$ does not admit a closed form solution. An asymptotic expansion can be used to approximate its values [5].

Murphy used a summation over a set of uniformly distributed sample points to compare the polynomials. An elliptic region for the sieving area can also be used. For comparison purposes, we can discard the constant multiplier $6/\pi^2$. Murphy [73] defined $\mathbb{E}(F,G)$ to be the following summation over K sample points ($\Omega_A\cos\theta_i$, $\Omega_B\cos\theta_i$):

$$\sum_{i=1}^{K}\rho\left(\frac{\log|F(\Omega_A\cos\theta_i,\Omega_B\sin\theta_i)|+\alpha(F)}{\log B_1}\right)\rho\left(\frac{\log|G(\Omega_A\cos\theta_i,\Omega_B\sin\theta_i)|+\alpha(G)}{\log B_2}\right).$$

The Ω_A and Ω_B are the major and minor axes of the elliptic sieving region. The angles θ_i determine uniform points on the boundary of the elliptic region. Murphy's $\mathbb{E}$ function provides a better estimate compared to the L^2-norm since it uses the Dickman-de Bruijn function. Stahlke and Kleinjung [95] showed that the Murphy score function correlates well with the scores of polynomials in sieving experiments.

3.3 Optimizing the quality of polynomials

Given a pair of polynomials f, g, we can generate some related pairs of polynomials using translation and rotation. Translation and rotation are useful to optimize the size and root properties. Let $f(x) = \sum_{i=0}^{d} c_i x^i$ and $g(x) = m_2 x - m_1$ where $m_1/m_2 \pmod{n}$ is the common root.

Translation of $f(x)$ by k gives a new polynomial $f_k(x)$ defined by $f_k(x) = f(x+k)$. The root of $f_k(x)$ is $m_1/m_2 - k \pmod{n}$. The linear polynomial $g_k(x)$ is $m_2 x - m_1 + k m_2$. Given a raw polynomial, we want to use translation and rotation to find a polynomial of smaller norm.

We consider some impacts of translation. Translation does not alter the root property in general. It is often used in the final stage of polynomial selection to optimize the size. However, translation only will often increase the polynomial size. We want to apply a combination of translation and rotation to reduce the polynomial size. If the skewness of the polynomial is larger than k, the translation does not affect the norm significantly. This can be seen from the coefficients of $f(x+k)$. Applying rotation on suitable $f_k(x)$ can help reduce the polynomial size.

Rotation by a polynomial $\lambda(x)$ gives a new polynomial $f_{\lambda(x)}(x)$ defined by $f_{\lambda(x)}(x) = f(x) + \lambda(x)\,(m_2 x - m_1)$. The linear polynomial is unchanged $g_{\lambda(x)}(x) = g(x) = m_2 x - m_1$. The root is unchanged. $\lambda(x)$ is often a linear or quadratic polynomial, depending on n and the skewness of $f(x)$. Rotation can affect both size and root properties.

A good choice of $\lambda(x)$ can improve the root property. Further, if $\lambda(x)$ only has small coefficients, the norm of $f_\lambda(x)$ is close to that of $f(x)$. Therefore, the (root) quality of the polynomial can be improved without significantly increasing the size.

3.4 Steps in polynomial selection

We describe a common procedure for polynomial selection. It involves three steps: polynomial generation, size optimization and root optimization. Given n and d, we can choose c_d which has small prime factors.

In polynomial generation, the l^∞ or l^1-norm is used to filter good polynomials. We decide m_2 and m_1 first, and then examine the coefficients c_{d-1}, c_{d-2}. If their size is admissible, we proceed to complete the other coefficients of the polynomial. If some coefficients are too large, we discard the polynomial and proceed to the next one.

We want to reduce the polynomial size in size optimization. To compare polynomials, we often use a better approximation, such as the L^2-norm.

In root optimization, we rotate the polynomial and attempt to generate a polynomial which has many roots modulo small prime powers. The new polynomial is expected to have similar or slightly larger size. We can optimize the size by translation since translation does not change affine roots.

Finally, we rank the best polynomials using Murphy's $\mathbb{E}$ function. For the top polynomials, we use some sieving tests to identify the best one.

Chapter 4

Polynomial generation

Murphy's base-m expansion [73] generates two polynomials, one of which is a linear monic polynomial. In this chapter, we discuss some methods which can generate polynomial pairs with non-monic linear polynomials. In these methods, the size of the polynomials' leading coefficients can be controlled.

4.1 Kleinjung's first algorithm

Kleinjung [48] described a way to generate non-monic linear polynomials in polynomial selection using the base-(m_1, m_2) expansion of n. Given integers m_1 and m_2, the base-(m_1, m_2) expansion of n is given by

$$n = \sum_{i=0}^{d} c_i m_1^i m_2^{d-i}.$$

He also gave an efficient algorithm to enumerate and check the size of the polynomial coefficients. We describe the method in this section.

Let an integer n, degree d and leading coefficient c_d be fixed. Let $m_0 = (n/c_d)^{1/d}$. We fix an integer m_2 having only small prime factors and choose an integer $m_1 \approx m_0$ such that $c_d m_1^d \equiv n \pmod{m_2}$. This is a necessary condition for the existence of a base-(m_1, m_2) expansion of n. We compute the polynomial coefficients c_i from the base-(m_1, m_2) expansion. The algebraic polynomial is given by $f(x) = \sum_{i=0}^{d} c_i x^i$, while the linear polynomial is $g(x) = m_2 x - m_1$. We assume that $m_0 \lesssim m_1$ and $m_2 \ll m_1$.

The existence of the base-(m_1, m_2) expansion can be shown in a constructive way. Let $r_d = n$. We define r_i, c_i for $i = d-1$ to 0 by

$$r_i = \frac{r_{i+1} - c_{i+1} m_1^{i+1}}{m_2},$$

$$c_i = \frac{r_i}{m_1^i} + \delta_i \text{ where } 0 \le \delta_i < m_2 \text{ such that } r_i \equiv c_i m_1^i \pmod{m_2}.$$

Each r_i can be expressed as

$$r_i = \sum_{j=0}^{i} c_j m_1^j m_2^{i-j}.$$

Coefficients. We discuss the size of coefficients in the base-(m_1, m_2) expansion. Let $\sigma = m_1 - m_0$. The coefficient c_{d-1} can be computed from

$$c_{d-1} = \frac{n - c_d m_1^d}{m_2 m_1^{d-1}} + \delta_{d-1}.$$

By construction, $(n/c_d)^{1/d} = m_0 = m_1 - \sigma$, which shows that

$$n = c_d (m_1 - \sigma)^d = c_d m_1^d + d c_d m_1^{d-1} \sigma + O(m_1^{d-2} m_2^2).$$

Therefore, c_{d-1} is about

$$c_{d-1} \approx \frac{d c_d \sigma}{m_2} + \delta_{d-1}.$$

If the difference between m_1 and m_0 is small, the size of c_{d-1} is also likely to be small.

We choose n, d and a small m_2 consisting of a product of very small factors. $m_1 \pmod{m_2}$ is found by solving the congruence equation $n = c_d m_1^d \pmod{m_2}$. We choose m_1 satisfying that congruence equation and $m_1 \gtrsim m_0$. The difference σ is smaller than m_2. The other coefficients c_i for $i \le d-2$ are controlled by

$$c_i = \frac{r_{i+1} - c_{i+1} m_1^{i+1}}{m_2 m_1^i} + \delta_i = \frac{\delta_{i+1} m_1}{m_2} + \delta_i.$$

Since δ_i is bounded by m_2, the coefficients c_i are bounded by $m_1 + m_2$. Note that we may also choose $\delta_i \in [-m_2/2, m_2/2)$ and/or $m_1 \leq m_0$.

Parameters. In the algorithm, we first choose some norm bound M depending on n, d. The permissible values for c_d then depend on n, d and M. We fix m_2 such that $m_2 \ll m_0$ and then choose $m_1 \approx m_0$ to ensure that c_{d-1} is small. We describe how to choose parameters to control c_d, c_{d-1}, c_{d-2}.

The l^∞-norm on the coefficients can be used to control the size of the coefficients. In practice, we often choose an upper bound M for the l^∞-norm where $M < n^{1/(d+1)}$. We allow c_0 and c_1 to be close to m_0. Considering the skewness, we may assume that the term c_1 dominates the l^∞-norm. On the other hand, $c_d s^{d/2} \leq M$. The optimal skewness lies in range

$$\left(\frac{m_0}{M}\right)^{\frac{2}{d-2}} \leq s \leq \left(\frac{M}{c_d}\right)^{\frac{2}{d}}.$$

This gives the range for the possible leading coefficients:

$$c_d \leq \left(\frac{M^{2d-2}}{n}\right)^{\frac{1}{d-3}}.$$

We also want to control the size of c_{d-1} and c_{d-2} . For $d/2 \leq i < d$, the l^∞-norm bound shows that $|c_i s^{i-d/2}| \leq M$. If we choose skewness to be the minimum in the permitted range, an upper bound on c_{d-1} is given by M^2/m_0. Similarly, an upper bound on c_{d-2} is given by

$$c_{d-2} \leq \left(\frac{M^{2d-6}}{m_0^{d-4}}\right)^{\frac{1}{d-2}}. \tag{4.1}$$

The upper bound on c_{d-2} is used in the algorithm to filter out bad polynomials. We discard polynomials with c_{d-2} larger than this bound. The other coefficient c_{d-1} is already small due to the choice of m_1 in the base-(m_1, m_2) expansion.

Checking the size of c_{d-2}. Kleinjung proposed an efficient way to check the coefficients c_{d-2} for many polynomials in a batch. The algorithm chooses m_2 as a product of small primes $\prod_{i=1}^{l} m_{2,i}$ such that $n \equiv c_d x^d \pmod{m_2}$ has d^l solutions. Each root can be expressed as $x_\mu = \sum_{i=1}^{l} x_{i,\mu_i}$. The $\mu = (\mu_1, \cdots, \mu_l)$ for $\mu_i \in \{1, \cdots, d\}$ is a multi-index notation of d^l possible values. Given a fixed i, $x_{i,j}$ for $1 \le j \le d$ are the d solutions of $n \equiv c_d x^d \pmod{m_{2,i}}$ satisfying

$$0 \le x_{i,j} < m_2, \quad \frac{m_2}{m_{2,i}} \mid x_{i,j}.$$

Each root of $n \equiv c_d x^d \pmod{m_2}$ can be obtained from the individual roots modulo $m_{2,i}$ using the Chinese Remainder Theorem.

Let $\tilde{m}_0$ be an integer close to m_0 and $m_2 \mid \tilde{m}_0$. We define $m_{1,\mu}$ to be $m_{1,\mu} = \tilde{m}_0 + x_\mu$. We consider the base-$(m_{1,\mu}, m_2)$ expansions of n, where the linear polynomial is given by $g(x) = m_2 x - m_{1,\mu}$. It can be seen that the coefficients c_{d-1} and c_{d-2} depend on some base pair $(m_{1,\mu}, m_2)$ and hence we denote them as $c_{d-1,\mu}$ and $c_{d-2,\mu}$. We want to express $c_{d-1,\mu}$ in a similar way as we express $m_{1,\mu}$ in terms of $x_{i,j}$. Kleinjung described the following method.

Lemma 4.1. *Given n, m_2, d, c_d, μ and $m_{1,\mu}$, there exist integers $0 \le e_{i,j} \le m_2$ for $1 \le i \le l$ and $1 \le j \le d$ such that*

$$c_{d-1,\mu} = \sum_{i=1}^{l} e_{i,\mu_i}$$

satisfies

$$c_{d-1,\mu}\, m_{1,\mu}^{d-1} \equiv \frac{n - c_d m_{1,\mu}^d}{m_2} \pmod{m_2}.$$

The $e_{i,j}$'s can be given by

$$\begin{aligned}
&e_{1,j} = c_{d-1,(j,1,\cdots,1)} \pmod{m_2},\\
&e_{i,1} = 0 \ \ \textit{for } i > 1,\\
&e_{i,j} = c_{d-1,(1,\cdots,1,j,1,\cdots 1)} - c_{d-1,(1,\cdots,1)} \pmod{m_2} \ \ \textit{for } i > 1,\ j > 1.
\end{aligned}$$

Let $1 \le k \le l$ be fixed. Assume that $\mu = (\mu_1, \cdots, \mu_l)$ and $\mu' = (\mu'_1, \cdots, \mu'_l)$ satisfy $\mu_i = \mu'_i$ for all $i \ne k$. It can be shown that $c_{d-1,\mu} - c_{d-1,\mu'} \pmod{m_2}$ only depends on the difference $x_{k,\mu_k} - x_{k,\mu'_k}$. We can express d^l coefficients $c_{d-1,\mu}$ in terms of linear combinations of $O(ld)$ variables.

The algorithm compares the size of $c_{d-2,\mu}$ to that of $m_{1,\mu}$. If $c_{d-2,\mu}/m_{1,\mu}$ is very close to an integer, then $c_{d-2,\mu}$ can be made small since we can add multiples of polynomial $(m_2 x - m_{1,\mu})x^{d-2}$ to remove the integral part. Since $m_0, \tilde{m}_0$ and $m_{1,\mu}$ are similar, we have the following estimate

$$\begin{aligned}\frac{c_{d-2,\mu}}{m_{1,\mu}} &\approx \frac{c_{d-2,\mu}}{\tilde{m}_0} \approx \frac{n - c_d m_{1,\mu}^d - c_{d-1,\mu} m_{1,\mu}^{d-1} m_2}{\tilde{m}_0^{d-1} m_2^2} \\ &\approx \frac{n - c_d \tilde{m}_0^d}{\tilde{m}_0^{d-1} m_2^2} + \frac{-c_d d(m_{1,\mu} - \tilde{m}_0) - c_{d-1,\mu} m_2}{m_2^2}.\end{aligned}$$

Let $f_0 = \dfrac{n - c_d \tilde{m}_0^d}{\tilde{m}_0^{d-1} m_2^2}$ and

$$f_{i,j} = -\frac{dc_d\, x_{i,j}}{m_2^2} - \frac{e_{i,j}}{m_2} \text{ for } 1 \le i \le l,\ 1 \le j \le d.$$

Since $m_{1,\mu} = \tilde{m}_0 + x_\mu = \tilde{m}_0 + \sum_{i=1}^{l} x_{i,\mu_i}$, the estimate satisfies

$$\frac{c_{d-2,\mu}}{m_{1,\mu}} \approx f_0 + \sum_{i=1}^{l} f_{i,\mu_i}.$$

Therefore, d^l such values $c_{d-2,\mu}/m_{1,\mu}$ can be estimated from $O(ld)$ values $x_{i,j}$ and $e_{i,j}$.

Kleinjung also described an efficient way to identify those μ's which lead to polynomials with small $c_{d-2,\mu}$'s. It takes $O(ld^{l/2} \log d)$ time to check d^l polynomials.

Let $\epsilon = c_{d-2,max}/m_0$ be a real number bounding the size of c_{d-2}, where $c_{d-2,max}$ is computed in the RHS of Equation (4.1). We partition the d^l polynomials into two lists of similar size. Let $l' = \lfloor l/2 \rfloor$. We compute two lists comprised of the fractional parts of $f_0 + \sum_{i=1}^{l'} f_{i,\mu_i}$ and $-\sum_{l'+1}^{l} f_{i,\mu_i}$. We sort the two lists and then search linearly in one list to see whether any element is within the $\epsilon-$neighbourhood of an element from the other list.

Computing $x_{i,j}, m_{i,j}, e_{i,j}$ and $f_{i,j}$ takes time $O(ld)$. Preparing, sorting and checking two sorted lists take time $O(ld^{l/2} \log d)$. The overall running-time is bounded by $O(ld^{l/2} \log d)$. On average, we spent $O(ld^{-l/2} \log d)$ for each polynomial. For efficiency, we want l to be large.

Algorithm. In summary, the algorithm consists of the following steps.

1. Given n, we determine the degree d and an upper bound $M \le n^{1/(d+1)}$ for the sup-norm of $f(x)$. We also choose l as the number of prime factors in m_2 and choose an upper bound on their size.

2. We compute the upper bound for c_d such that $c_{d,max} \le \left(M^{2d-2}/n\right)^{1/(d-3)}$. For each c_d in the range, we find a set S of primes p such that $c_d x^d \equiv n \pmod{p}$ has d solutions. We also calculate $m_0 = (n/c_d)^{1/d}$ and upper bounds for c_{d-1} and c_{d-2} where $c_{d-1,max} = M^2/m_0, \quad c_{d-2,max} = \left(M^{2d-6}/m_0^{d-4}\right)^{1/(d-2)}$.

3. For all (or some) subsets of S of length l whose product is smaller than $c_{d-1,max}$, we compute $x_{i,j}, m_{i,j}, e_{i,j}, f_0$ and $f_{i,j}$.

4. Let $\epsilon = c_{d-2,max}/m_0$. We find those μ such that $f_0 + \sum_{i=1}^{l} f_{i,\mu_i} \left(\approx \dfrac{c_{d-2,\mu}}{m_{1,\mu}}\right)$ lies within the ϵ-neighborhood of an integer.

The polynomials have small c_d, c_{d-1} due to construction in Steps 2 and 3, and small c_{d-2} due to the ϵ-bound in Step 4.

Kleinjung [49] gave another algorithm for polynomial selection which generates highly skewed polynomials whose coefficients c_{d-2} can be controlled. We call it Kleinjung's second algorithm.

4.2 Kleinjung's second algorithm

The aim of the algorithm is to find a good polynomial with large skewness. If the skewness is similar to the sieving area, the memory use in sieving can be reduced [49]. The algorithm also gives an efficient way to control the size of c_{d-2}.

Let the number n and degree d be fixed. We want to choose m_1 and m_2 such that the size of c_{d-1}, c_{d-2} can be controlled. Let the expansion of n be $n = \sum_{i=0}^{d} c_i m_1^i m_2^{d-i}$.

In Kleinjung's first algorithm, we partially (in an informal sense) compute c_{d-1}, c_{d-2} in the expansion and then check the size of c_{d-2}. To estimate and control c_{d-1}, we use the estimate $(n - c_d m_1^d)/(m_1^{d-1} m_2)$. To check c_{d-2}, we use information on c_{d-1} in the expansion to estimate c_{d-2}.

In Kleinjung's second algorithm, we want to bound c_{d-2} without explicitly expanding terms c_{d-1} and c_{d-2}. The idea is to write $n = c_d m_1^d + c_{d-1} m_1^{d-1} m_2 + m_2^2 R$. We want to choose m_1, m_2 such that R/m_1^{d-2} is expected to be small.

Coefficients. We describe how the coefficients are generated and controlled in Kleinjung's second algorithm.

Let the linear polynomial be $g(x) = m_2 x - m_1$. m_1 is chosen to be close to $(n/c_d)^{1/d}$ so that c_{d-1} is small. We find an equivalent expansion of n which has zero c_{d-1} by translation.

Let $n = \sum_{i=1}^{d} c_i m_1^i m_2^{d-i}$ be the expansion. We express it as

$$n = c_d \left(m_1^d + \frac{c_{d-1}}{c_d} m_1^{d-1} m_2 \right) + \sum_{i=0}^{d-2} c_i m_1^i m_2^{d-i}.$$

We complete the d-th power to remove the $(d-1)$-th term. It can be seen that

$$c_d \left(m_1 + \frac{c_{d-1}}{d c_d} m_2 \right)^d = c_d m_1^d + c_{d-1} m_1^{d-1} m_2 + m_2^2 R_0, \tag{4.2}$$

where R_0 is

$$c_d \sum_{i=2}^{d} \binom{d}{i} m_1^{d-i} \left(\frac{c_{d-1}}{d c_d} m_2 \right)^i.$$

The first two terms in Equation (4.2) are in the expansion of n, and hence n can be expressed as

$$n = c_d \left(m_1 + \frac{c_{d-1}}{d c_d} m_2 \right)^d - m_2^2 R_0 + \sum_{i=0}^{d-2} c_i m_1^i m_2^{d-i}.$$

It follows that

$$d^d c_d^{d-1} n = (c_d d m_1 + c_{d-1} m_2)^d - d^d c_d^{d-1} m_2^2 R_0 + d^d c_d^{d-1} \sum_{i=0}^{d-2} c_i m_1^i m_2^{d-i}.$$

Denote $\tilde{n} = d^d c_d^{d-1} n$ and $\tilde{m} = dc_d m_1 + c_{d-1} m_2$. Then we see that

$$\tilde{n} = \tilde{m}^d + m_2^2 \tilde{R} \tag{4.3}$$

for some $\tilde{R}$. c_d is fixed in the beginning of the algorithm and hence $\tilde{n}$ is a constant. It can be also seen that $\tilde{m} \approx c_d d m_1 \approx \tilde{n}^{1/d}$ since $m_2 \ll m_1$. Therefore we want to choose $\tilde{m}$ to be close to $\tilde{n}^{1/d}$. We expand Equation (4.3).

$$\tilde{n} = \tilde{m}^d + m_2^2 \sum_{i=2}^{d} m_1^{d-i} m_2^{i-2} \left(d^d c_d^{d-1} c_{d-i} - \binom{d}{i} (c_d d)^{d-i} c_{d-1}^i \right).$$

The coefficient c_{d-2} appears in the second term in the expansion. We will estimate it in terms of $\tilde{n}$ and $\tilde{m}$.

Equation $\tilde{m} = dc_d m_1 + c_{d-1} m_2$ shows $m_1 = (\tilde{m} - c_{d-1} m_2)/(dc_d)$. The remainder term $\tilde{R}$ is

$$\begin{aligned}
\tilde{R} &= \frac{\tilde{n} - \tilde{m}^d}{m_2^2} \\
&= \sum_{i=2}^{d} m_1^{d-i} m_2^{i-2} \left(d^d c_d^{d-1} c_{d-i} - \binom{d}{i} (c_d d)^{d-i} c_{d-1}^i \right) \\
&= \sum_{i=2}^{d} m_2^{i-2} \left(\frac{\tilde{m} - c_{d-1} m_2}{dc_d} \right)^{d-i} \left(d^d c_d^{d-1} c_{d-i} - \binom{d}{i} (c_d d)^{d-i} c_{d-1}^i \right) \\
&= \sum_{i=2}^{d} m_2^{i-2} (\tilde{m} - c_{d-1} m_2)^{d-i} \left(d^i c_d^{i-1} c_{d-i} - \binom{d}{i} c_{d-1}^i \right).
\end{aligned}$$

We want to estimate the size of c_{d-2} in $\tilde{R}$. The coefficient of c_{d-2} is dominated by $d^2 c_d \tilde{m}^{d-2}$. We divide $\tilde{R}$ by the dominating coefficient.

$$\begin{aligned}
\frac{\tilde{R}}{d^2 c_d \tilde{m}^{d-2}} &= \frac{\tilde{n} - \tilde{m}^d}{m_2^2 d^2 c_d \tilde{m}^{d-2}} \\
&= \sum_{i=2}^{d} \frac{m_2^{i-2} (\tilde{m} - c_{d-1} m_2)^{d-i}}{\tilde{m}^{d-2}} \left((dc_d)^{i-2} c_{d-i} - \binom{d}{i} c_{d-1}^i d^{-2} c_d^{-1} \right) \\
&= \sum_{i=2}^{d} \left(1 - \frac{c_{d-1} m_2}{\tilde{m}} \right)^{d-i} \left(\frac{m_2}{\tilde{m}} \right)^{i-2} \left((dc_d)^{i-2} c_{d-i} - \binom{d}{i} c_{d-1}^i d^{-2} c_d^{-1} \right).
\end{aligned}$$

We assume that $m_2 \ll m_1 \leq \tilde{m}$. An approximation of $\tilde{R}/(d^2 c_d \tilde{m}^{d-2})$ is

$$\begin{aligned}
&\sum_{i=2}^{d} \left(\frac{m_2}{\tilde{m}}\right)^{i-2} \left((dc_d)^{i-2} c_{d-i} - \binom{d}{i} c_{d-1}^{i} d^{-2} c_d^{-1} \right) \\
&\approx \sum_{i=2}^{d} \left(\frac{m_2}{\tilde{m}}\right)^{i-2} (dc_d)^{i-2} c_{d-i} \\
&\approx c_{d-2} + \frac{m_2}{m_1} c_{d-3} + \left(\frac{m_2}{m_1}\right)^2 c_{d-4}.
\end{aligned}$$

In practice, c_{d-3} and c_{d-4} are $O(m_1)$. If m_2 is smaller than c_{d-2}, $(\tilde{R})(d^2 c_d \tilde{m}^{d-2})$ is about c_{d-2}.

Given $\tilde{n}$ and c_d, we search for some $\tilde{m}$ which is close to $\tilde{m}_0 = \tilde{n}^{1/d} = dc_d^{1-1/d} n^{1/d}$. Let $\tilde{\sigma}$ be the difference between $\tilde{m}_0$ and $\tilde{m}$. Denote $m_0 = (n/c_d)^{1/d}$. An approximation of $|\tilde{R}|/(d^2 c_d \tilde{m}^{d-2})$ is

$$\begin{aligned}
\frac{|\tilde{R}|}{d^2 c_d \tilde{m}^{d-2}} &= \frac{|\tilde{n} - \tilde{m}^d|}{m_2^2 d^2 c_d \tilde{m}^{d-2}} = \frac{|\tilde{m}_0^d - \tilde{m}^d|}{m_2^2 d^2 c_d \tilde{m}^{d-2}} \\
&= \frac{|(\tilde{m}_0 + \tilde{\sigma})^d - \tilde{m}_0^d|}{m_2^2 d^2 c_d \tilde{m}^{d-2}} \approx \frac{\tilde{m}_0 \tilde{\sigma}}{dc_d m_2^2} \\
&= m_0 \frac{\tilde{\sigma}}{m_2^2}
\end{aligned}$$

We want to choose m_2 to be large and the difference $\tilde{\sigma}$ to be small.

Algorithm. Kleinjung gave a way to find such m_2 and m_1 efficiently. Given n, d, c_d (and hence $\tilde{n}$), it is sufficient to find $\tilde{m}$ and m_2 such that $\tilde{n} = \tilde{m}^d + m_2^2 \tilde{R}$. Let $\tilde{m}_0$ be the integral part of $\tilde{n}^{1/d}$. Given $\tilde{n}$, we first compute roots of the following equations for primes $p \in [B, 2B]$.

$$\tilde{n} \equiv (\tilde{m}_0 + r)^d \pmod{p}$$

The roots r are lifted to modulo p^2 where $r' \equiv r \pmod{p}$.

$$\tilde{n} \equiv (\tilde{m}_0 + r')^d \pmod{p^2} \tag{4.4}$$

Secondly, we search for collisions on r' among equations modulo different prime powers. If some r' satisfies both equations for p_1, p_2, then

$$\tilde{n} \equiv (\tilde{m}_0 + r')^d \pmod{p_1^2 p_2^2}.$$

We choose $m_2 = p_1 p_2$ and $\tilde{m} = \tilde{m}_0 + r'$. Given $\tilde{m}, d, c_d, m_2$ and the expression $\tilde{m} = dc_d m_1 + c_{d-1} m_2$, we compute c_{d-1} in

$$c_{d-1} \equiv \frac{\tilde{m}}{m_2} \pmod{dc_d}$$

provided that $\gcd(dc_d, m_2) = 1$. Hence $0 \le c_{d-1} < dc_d$. The size of c_{d-1} can often be smaller than in Kleinjung's first algorithm, where $|c_{d-1}|$ was about $dc_d\sigma/m_2 + m_2$.

Finally, m_1 can be calculated from $m_1 = (\tilde{m} - c_{d-1} m_2)/(dc_d)$. It is close to $(n/c_d)^{1/d}$. The difference between m_1 and $(n/c_d)^{1/d}$ is small. Therefore the coefficients c_i for $0 \le i < d-2$ can be bounded by $O(m_1)$ using Lemma 4.1.

The size of m_2 depends on the size of its factors p. We want p to be large so that the ratio $\tilde{\sigma}/m_2^2$ is small to keep c_{d-2} small. On the other hand, the size of p has to be moderate to facilitate the solutions of $\tilde{n} \equiv (\tilde{m}_0 + r')^d \pmod{p^2}$. In the general case, r' is bounded by $O(B^2)$. If there is a collision, r' is $O(B^2)$ while m_2 is $O(B^4)$. The coefficient c_{d-2} is $O(m_0/B^2)$.

Performance. We estimate the expected number of collisions for each c_d. Let k integers bounded by X be drawn randomly from a uniform distribution. For the i-th integer α being drawn, the chance that α collides with some previously drawn integer is about

$$1 - \left(\frac{X-1}{X}\right)^{i-1}.$$

The expected number of collision after k selections is

$$\sum_{i=1}^{k} \left(1 - \left(\frac{X-1}{X}\right)^{i-1}\right) = k - X + X\left(1 - \frac{1}{X}\right)^k.$$

If X is large and $k \ll X$, we consider an approximation by the truncated expansion

$$k - X + X\left(1 - \frac{1}{X}\right)^k = k - X + X\left(1 - \frac{k}{X} + \frac{k^2 - k}{2X^2} + O\left(\frac{k^3}{X^3}\right)\right) = \frac{k^2 - k}{2X} + O\left(\frac{k^3}{X^2}\right).$$

Since k/X is small, the equation is dominated by $k^2/(2X)$.

Solutions r' in Equation (4.4) are in the range $[0, 4B^2)$ (or $[-2B^2, 2B^2)$). We consider roots $r' \in [-M, M]$ for some M. For each c_d, we solve Equation (4.4) for all primes $p \in [B, 2B]$. On average, each equation has one root modulo p. We assume that the roots can be uniquely lifted to modulo p^2 and behave like uniform random integers of that size. Hence, it is similar to the situation where we draw $O(B/\log B)$ random integers from a uniform distribution of size $2M$.

Let (p, r) be a pair satisfying Equation (4.4). For each p, the range $2M$ allows $2M/p^2$ such roots r^*. The expected total number of pairs (p, r) per c_d (for all primes between B and $2B$) about

$$\sum_{\substack{p=B \\ p \text{ prime}}}^{2B} \frac{2M}{p^2} \approx 2M \int_B^{2B} \frac{1}{x^2 \log x} dx \approx \frac{M}{B \log B}.$$

Therefore, the expected number of collisions per c_d is

$$\frac{(M/(B\log B))^2}{4M} = \frac{M}{4B^2 \log^2 B}.$$

We consider the time spent per collision. The algorithm generates $M/(B\log B)$ pairs which we expect to give $M/(4B^2 \log^2 B)$ collisions. We use a hash table to record all pairs for which the amortized time per insertion is a constant. The total time spent on generating and recording pairs is about

$$O\left(\frac{M}{B\log B} + \frac{B}{\log B}\right).$$

*It is about 1 if $M \approx 2B^2$.

The expected time per collision is

$$O\left(\frac{B^3 \log B}{M} + B \log B\right).$$

If $r \in \mathbb{Z}/p^2\mathbb{Z}$, then $M = O(B^2)$. The expected number of collisions per c_d is $O(1/(\log^2 B))$. About $O(\log^2 B)$ trials of c_d give one collision. The expected time per collision is $O(B \log B)$.

When B is large, finding a collision becomes difficult. We may set M, the permissible range of r, to be larger. The disadvantage is that it gives a worse bound on c_{d-2} due to $|c_{d-2}| \approx m_0 M / m_2^2$.

4.2.1 Some variants

We review some variants of the algorithm. For convenience, the method discussed above is referred as the original method.

Factors of m_2. Kleinjung [49] described some alternate ways of choosing m_2. We can replace $m_2 = p_1 p_2$ by $m_2 = cp$ $(c \in C, p \in P)$ where $C = [P_1, P_2]$ and P is a set of primes in $[P_2, P_3]$. The collision can be detected within the P-pairs and between C-pairs and P-pairs. More generally, the collision can be detected between integers c_1 and c_2 provided that they have no common factor. Otherwise, the size $|c_{d-2}|$ could be worse due to a smaller $m_2 = c_1 c_2 / \gcd(c_1, c_2)$.

Paul Zimmermann (personal communication) described a variant which chooses $m_2 = p_1 p_2 q_1 q_2$ such that m_2 consists of four primes factors. Let B be the prime bound used in the original method. We consider two sets of primes P and Q, where P contains primes in $[\sqrt{B}, 2\sqrt{B}]$ and Q contains primes in $[2\sqrt{B}, 3\sqrt{B}]$. For all primes p (and q), we solve r in the following equations.

$$\tilde{n} \equiv (\tilde{m}_0 + r)^d \pmod{p^2}.$$

For distinct prime pairs $p_1, p_2 \in P$ (or $q_1, q_2 \in Q$), we find roots r' for the following equations using Chinese Remainder Theorem.

$$\tilde{n} \equiv (\tilde{m}_0 + r')^d \pmod{p_1^2 p_2^2}.$$

We search for collisions on the roots between equations modulo $p_1^2 p_2^2$ and $q_1^2 q_2^2$. If there is a collision on root r', we can see that

$$\tilde{n} \equiv (\tilde{m}_0 + r')^d \pmod{p_1^2 p_2^2 q_1^2 q_2^2}.$$

There are $O(B/\log^2 B)$ pairs $p_1 p_2$ (or $q_1 q_2$) and hence $O(B^2/\log^4 B)$ such $p_1^2 p_2^2 q_1^2 q_2^2$. $m_2 = p_1 p_2 q_1 q_2$ which is about $O(B^2)$. The coefficient c_{d-2} is about m_0/B^2. The expected number of collisions per c_d is

$$O\left(\frac{\left(B/(\log^2 B)\right)^2}{B^2}\right) = O(1/\log^4 B).$$

The time spent on solving equations is smaller than the original method due the smaller size of the primes ($O(\sqrt{B})$ versus $O(B)$). We need to solve $O(\sqrt{B}/(\log B))$ equations and then generates $B/\log^2 B$ pairs using CRT. The practical performance depends on the efficiency on solving equations, CRTs and hash table operations.

Special-q. Solving congruence equations is expensive. Kleinjung [49] gave a way to reuse the roots. In the first step, we search for collisions on r_p for pairs (p, r_p) where $p \in [B, 2B]$. We denote the set of such primes p as P.

$$\tilde{n} \equiv (\tilde{m}_0 + r_p)^d \pmod{p^2}.$$

In the second step, we choose a set Q of primes not in $[B, 2B]$ as the special-q's. We solve the equations

$$\tilde{n} \equiv (\tilde{m}_0 + r_q)^d \pmod{q^2}.$$

For each q and all p, we calculate $i_p \in [0, q^2)$ from

$$r_q + i_p q^2 \equiv r_p \pmod{p^2}$$

such that

$$\tilde{n} \equiv (\tilde{m}_0 + r_q + i_p q^2)^d \pmod{p^2}.$$

We record the pairs (p, i_p) in the hash table. If there is a collision on i between (p_1, i) and (p_2, i), then

$$\tilde{n} \equiv (\tilde{m}_0 + r_q + iq^2)^d \pmod{p_1^2 p_2^2 q^2}.$$

Let $m_2 = p_1 p_2 q$ and $r = r_q + iq^2$. We see that

$$\tilde{n} \equiv (\tilde{m}_0 + r)^d \pmod{m_2^2}.$$

We consider the performance of the special-q variant. Let B_q be an upper bound on the special-q primes. For the first step, the analysis remains the same as before. The range for r_p is $O(B^2)$ and the number of pairs generated is $O(B/\log B)$. The expected collisions per c_d is $O(1/\log^2 B)$.

For each special-q pair (q, r_q), we generate $O(B/\log B)$ pairs (p, i_p) where i_p is bounded by $O(B^2)$. The expected collisions per pair (q, r_q) is $O(1/\log^2 B)$. The expected total number of collisions for all special-q's is $O(B_q/(\log^2 B \log B_q))$ where B_q is the bound for q. The special-q variant is efficient in practice since solving equations is traded by computing modulo inverses.

As a summary, we describe Kleinjung's second algorithm in Algorithm 4.1.

In practice, Kleinjung's second algorithm is found to be more efficient than Kleinjung's first algorithm. First, it has a better control on the size of coefficients c_{d-2}. The c_{d-2} in the second algorithm size is often smaller. In addition, the first algorithm restricts the search space much more than the second algorithm.

Algorithm 4.1: Kleinjung's second algorithm

```
input  : $n, d, c_d$, prime sets $P$ and $Q$;
output: $(m_1, m_2)$ for the base expansion of $n$;
1  compute $\tilde{n}, \tilde{m}_0$ from $n, d, c_d$;
2  for $p \in P$ do
3      solve $r_p$ in $\tilde{n} \equiv (\tilde{m}_0 + r_p)^d \pmod{p^2}$;
4      record $(p, r_p)$ and detect collisions on $r_p$;
5  for $q \in Q$ do
6      solve $r_q$ in $\tilde{n} \equiv (\tilde{m}_0 + r_q)^d \pmod{q^2}$;
7      for each $r_q$ do
8          for $p \in P$ do
9              for each $r_p$ do
10                 solve $i_p$ in $\tilde{n} \equiv (\tilde{m}_0 + r_q + i_p q^2)^d \pmod{(pq)^2}$;
11                 record $(p, i_p)$ and detect collisions on $i_p$;
```

Chapter 5

Size optimization

Polynomial generation (e.g. using Kleinjung's methods [48, 49]) can yield many raw polynomials with small leading coefficients. The raw polynomials have very small $|c_d|, |c_{d-1}|$ and small $|c_{d-2}|$. The coefficients $|c_{d-3}|, \cdots, |c_0|$ are comparable to $(n/c_d)^{1/d}$. In this chapter, we discuss how to optimize the size of raw polynomials by changing the skewness, translating and rotating.

5.1 Objective function

In size optimization, we want to use translation and rotation to produce polynomials with smaller L^2-norm. The logarithmic L^2-norm given in Equation (3.3) is defined on a square domain.

$$\log L^2(F) = \frac{1}{2} \log \left(s^{-d} \int_{-1}^{1} \int_{-1}^{1} F^2(xs, y) \, \mathrm{d}x \, \mathrm{d}y \right).$$

For computational convenience, this chapter uses a variant L^2-norm defined on an elliptic domain. We change to polar coordinates where $x = r \cos\theta$ and $y = r \sin\theta$.

$$\log L^2(F) = \frac{1}{2} \log \left(s^{-d} \int_{0}^{2\pi} \int_{0}^{1} F^2(s \cos\theta, \sin\theta) \, r^{2d+1} \, \mathrm{d}r \, \mathrm{d}\theta \right). \tag{5.1}$$

This chapter is joint work with Paul Zimmermann.

The logarithmic L^2-norm in Equation (3.3) is not exactly the same as the logarithmic L^2-norm in Equation (5.1), because the integrals are over different domains (rectangle and ellipse). They are both (but slightly different) approximations to the size of polynomials.

For sextic polynomials, the logarithmic L^2-norm in Equation (5.1) can be expressed as

$$\begin{aligned}\log L^2(F) = \frac{1}{2}\log\Big(\frac{\pi}{7168}\big(&231\,\tilde{c}_0^2 + 42\,\tilde{c}_0\tilde{c}_2 + 14\,\tilde{c}_0\tilde{c}_4 + 10\,\tilde{c}_0\tilde{c}_6 + 21\,\tilde{c}_1^2 + 14\,\tilde{c}_1\tilde{c}_3 \\ &+ 10\,\tilde{c}_1\tilde{c}_5 + 7\,\tilde{c}_2^2 + 10\,\tilde{c}_2\tilde{c}_4 + 14\,\tilde{c}_2\tilde{c}_6 + 5\,\tilde{c}_3^2 + 14\,\tilde{c}_3\tilde{c}_5 \\ &+ 7\,\tilde{c}_4^2 + 42\,\tilde{c}_4\tilde{c}_6 + 21\,\tilde{c}_5^2 + 231\,\tilde{c}_6^2\big)\Big)\end{aligned} \tag{5.2}$$

where $\tilde{c}_i = c_i s^{i-d/2}$.

In our analysis, it is often tedious to work the with L^2-norm. We make the assumption that the L^2-norm and l^∞-norms are correlated. Then we can find polynomials with good l^∞-norm and hope that they also have good L^2-norm.

If the optimal skewness of L^2-norm and l^∞-norm does not differ too much, the correlation roughly follows from the expression in Equation (5.2). We also give some empirical evidence here.

B_{768} in Appendix A is a raw polynomial generated by Kleinjung's second algorithm [49] that could be used for RSA-768. Let f, g be the polynomial pair in B_{768}. We translate f by some uniformly-distributed sample points $k \in [-10^{10}, 10^{10}]$. For each translated polynomial, we compute the optimized (in terms of skewness) L^2-norm and l^∞-norm. Figure 5.1 shows that the logarithmic L^2 and l^∞-norms of the translated polynomials have a similar trend.

In size optimization, we want to produce polynomials with smaller logarithmic L^2-norm (e.g. Equation (5.1)) by changing the skewness, translating and rotating.

For raw quintic polynomials, coefficients $|c_5|, |c_4|$ and $|c_3|$ are small. The next non-controllable coefficient is c_2. We consider the l^∞-norm (e.g. assuming correlation between L^2 and l^∞-norms) on polynomial coefficients. Let the sieving bounds be

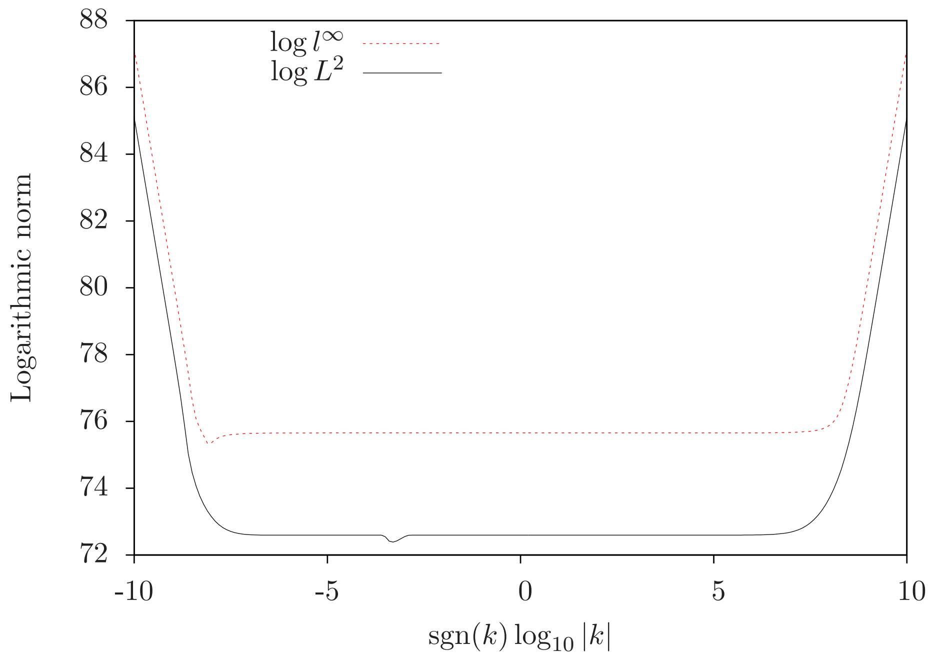

Figure 5.1: Logarithmic L^2, l^∞-norm of translation

$|a| \leq U\sqrt{s}$ and $0 < b \leq U/\sqrt{s}$. The l^∞-norm is bounded below by $|c_2|s^{-1/2}U^5$. As $s \geq 1$, the contribution of c_2 on the polynomial value is already reduced by a factor of $s^{-1/2}$.

For raw sextic polynomials, the polynomial values are bounded below by terms $|c_3|U^6$. As c_3 is not controlled in the polynomial generation step, we do not naturally get a small norm in the raw polynomials. Therefore, it is important to size-optimize them before trying to optimize the root properties. In this chapter, we focus on the size optimization of raw, sextic polynomials.

5.2 Local descent optimization

Let $f(x)$ be a sextic polynomial. We can use quadratic rotations since $c_3, \cdots, c_0$ have order $(n/c_d)^{1/d}$. Rotation by $wx^2 + ux + v$ is given by

$$f_{w,u,v}(x) = f(x) + (wx^2 + ux + v)\, g(x). \tag{5.3}$$

Murphy [73] used the classical multivariable optimization technique to optimize the L^2-norm. For sextic polynomials, there are five variables w, u, v, k, s, where k is the translation amount and s is the skewness. w, u, v, k are integers and s is real.

The allowed range of these parameters is huge. Standard iterative methods, such as gradient descent, are slow and tend to get stuck in local minima. For efficiency, we use a local descent method to optimize the size. In each iteration, we attempt some translations k and rotations w, u, v, and descend into the local minimum in the direction determined by some k, w, u, v. During the procedure, we need to re-optimize the skewness of the polynomial. We describe the method in Algorithm 5.1.

Algorithm 5.1: Local descent method

```
input  : polynomial pair $f(x) = \sum_{i=0}^{d} c_i x^i$ and $g(x) = m_2 x - m_1$;
output: polynomial pair $f', g'$ of smaller $L^2$-norm;
1  $k = w = u = v = 1$;
2  while *local minimum is found or loop limit is reached* do
3      $f'(x) = f(x \pm k), g'(x) = g(x) \pm k m_2$;
4      if *either* $L^2(f') < L^2(f)$ then
5          $f = f',\ g = g',\ k = 2k$;
6      else
7          $k = \lceil k/2 \rceil$;
8      $f'(x) = f(x) \pm w\, x^2\, g(x)$;
9      if *either* $L^2(f') < L^2(f)$ then
10         $f = f',\ w = 2w$;
11     else
12         $w = \lceil w/2 \rceil$;
13     Search similarly (e.g. Lines 8-12) for linear and constant rotations;
14 return $f(x), g(x)$;
```

The method seems to work for quintic polynomials, when the searching space is not too huge. However, it performs badly for sextic polynomials. Many iterations get stuck at local minima without giving much reduction in size. We demonstrate this situation.

We examine a data set consisting of 10^5 raw sextic polynomials for RSA-768. The polynomials are generated by Kleinjung's second algorithm [49]. Figure 5.2 shows the (discrete) density distribution of logarithmic L_2-norm for the raw and optimized (by local descent) polynomials.

In Figure 5.2, the raw polynomials have average logarithmic L^2-norm 80.75 and

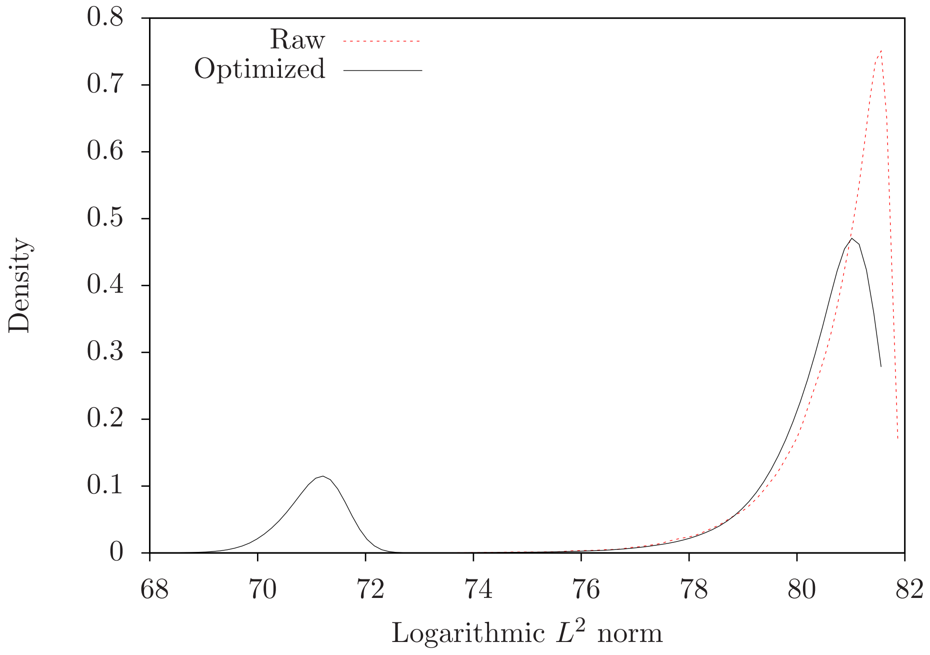

Figure 5.2: Local descent optimization

standard deviation 1.00. The optimized polynomials have average logarithmic L^2-norm 79.06 and standard deviation 3.55. It can be seen that only a few polynomials are optimized well by the local descent procedure. Many of them seem to descend to a local minimum rapidly and then get stuck. We discuss some better methods to optimize such polynomials.

5.3 Some better methods

To overcome local minima, we could try some standard global optimization methods such as simulated annealing. However, they do not seem to work well in our experiments, perhaps due to the huge search space and large coefficients.

Instead, we first translate the algebraic polynomial to increase the skewness. Heuristically, it moves away from the starting point and decreases the chance to get stuck in a local minimum. If the skewness of the polynomial is larger than the translation amount k, the translation does not affect the norm significantly. This can be seen from the coefficients of $f(x + k)$.

5.3.1 Use of translation

One question is how to decide the translation amount. We first describe a heuristic method.

The aim is to produce a polynomial with a better shape, such that each ratio c_i/c_{i-1} correlates with the skewness. In the meantime, we want to keep the polynomial size from changing too much during the translation.

One way is to increase k successively until an increase of the L^2-norm occurs. Empirically, the norm of translated polynomials stays steady initially and then increases rapidly as k increases further (see Figure 5.1). We can choose k to be the turning point of the norm. We then use a local descent method to search for a local minimum.

The method works better than a local descent method. For comparison, we consider the same data set of 10^5 polynomials used in Figure 5.2. Figure 5.3 shows the (discrete) density distribution of the logarithmic L_2-norm for the raw and optimized polynomials. The optimized polynomials have average logarithmic L^2-norm 76.83 and standard deviation 4.64.

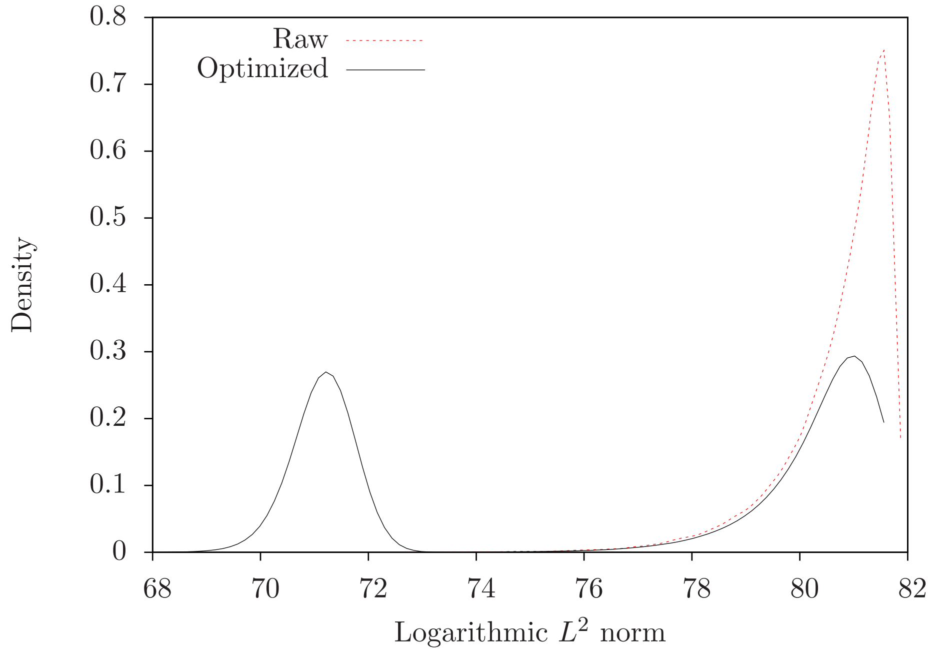

Figure 5.3: Translation and local descent

5.3.2 Better use of translation

The heuristic method in Subsection 5.3.1 still does not work for many polynomials. Figure 5.3 shows that more than half of the polynomials are not optimized well. We describe a better method.

In the raw polynomial, c_0, c_1, c_2, c_3 have similar size and are much larger than c_4, c_5, c_6. In Equation (5.2), the $\tilde{c}_0, \tilde{c}_1, \tilde{c}_2$ are bounded by $\tilde{c}_3$. Therefore, the L^2-norm can be controlled by terms involving $\tilde{c}_3, \tilde{c}_4, \tilde{c}_6$. A lower bound, not depending on skewness, is dominated by the term $\tilde{c}_3^2 = c_3^2$. Since a small c_3 is a necessary condition for a small L^2-norm, an idea is to minimize c_3 by translation.

Translation by k gives a polynomial whose coefficients are functions of k:

$$\begin{aligned} f(x+k) = {} & c_6x^6 \\ & + (6c_6k + c_5)x^5 \\ & + (15c_6k^2 + 5c_5k + c_4)x^4 \\ & + (20c_6k^3 + 10c_5k^2 + 4c_4k + c_3)x^3 \\ & + \cdots . \end{aligned}$$

Let $c_i(k)$ be the coefficients of the i-th term in the translated polynomial. $c_3(k)$ of $f(x+k)$ is a cubic polynomial in k. The coefficients $c_0(k), c_1(k), c_2(k)$ will increase due to translation. We can use rotation to reduce them, if needed.

Minimizing $c_3(k)$. The cubic polynomial $c_3(k)$ has either one or three real roots. For each real root, we choose k to be the nearest integer to the root. We translate $f(x)$ by k. The optimization is expected to work for all sextic polynomials since there exists at least one real root for a cubic polynomial.

Translation causes a larger $c_5(k)$ and $c_4(k)$, compared to c_5 and c_4. In the cubic polynomial $c_3(k)$, the constant term c_3 is about $O(m_1)$. The real root has approximate order $O((m_1/c_6)^{1/3})$. Hence $c_5(k)$ is bounded by $O((m_1/c_6)^{1/3})$ and $c_4(k)$ is bounded by $O((m_1/c_6)^{2/3} + c_4)$. Empirically, c_4 is comparable to $(m_1/c_6)^{2/3}$. This shows that the coefficient $c_5(k)$ can increase significantly.

The coefficients $c_2(k), c_1(k), c_0(k)$ are also increased during translation. We can reduce them using rotation (e.g. in a local descent optimization). After translation, $c_3(k)$ is minimized and often smaller than the original c_3. This also gives some room to reduce $c_2(k), c_1(k), c_0(k)$ by rotation.

Compared to the raw polynomial, $c_4(k), c_2(k), c_1(k), c_6$ are comparable (or the same), while $c_5(k)$ and $c_0(k)$ are increased. If the gain from a smaller $c_3(k)$ exceeds the deterioration from a larger $c_5(k)$, the L^2-norm can be reduced. In translated polynomials, the L^2-norm is dominated by terms involving $c_3(k), c_4(k), c_5(k)$ and $c_6(k)$. Once k is fixed in minimizing $c_3(k)$, we can further optimize the polynomial locally by the descent method.

Example and statistics. We give an example for a polynomial that could be used for RSA-768. B_{768} in Appendix A is a raw polynomial for RSA-768 generated by Kleinjung's second algorithm. It has logarithmic L_2-norm 72.59. The coefficient of x^3 in $f(x + k)$ is

$$71727600k^3 + 190647000k^2 + 1129504938822234180339372k +$$
$$71869370113024022527461281418 8142.$$

The cubic polynomial has a real root near $k = -191352410$. We translate $f(x)$ by k and then apply the local descent optimization to the translated polynomial. The optimized polynomial C_{768} in Appendix A has L_2-norm 67.60. Empirically, some sieving tests over short intervals show C_{768} sieves about 1.75 times faster than B_{768}.

The method works better on average than previous methods. We consider the same data set of 10^5 polynomials used in Figure 5.2 and Figure 5.3. Figure 5.4 shows the (discrete) density distribution of the logarithmic L_2-norm for the raw and optimized polynomials. The improved method can reduce the average logarithmic L_2-norm to 70.34 with a standard deviation 0.60.

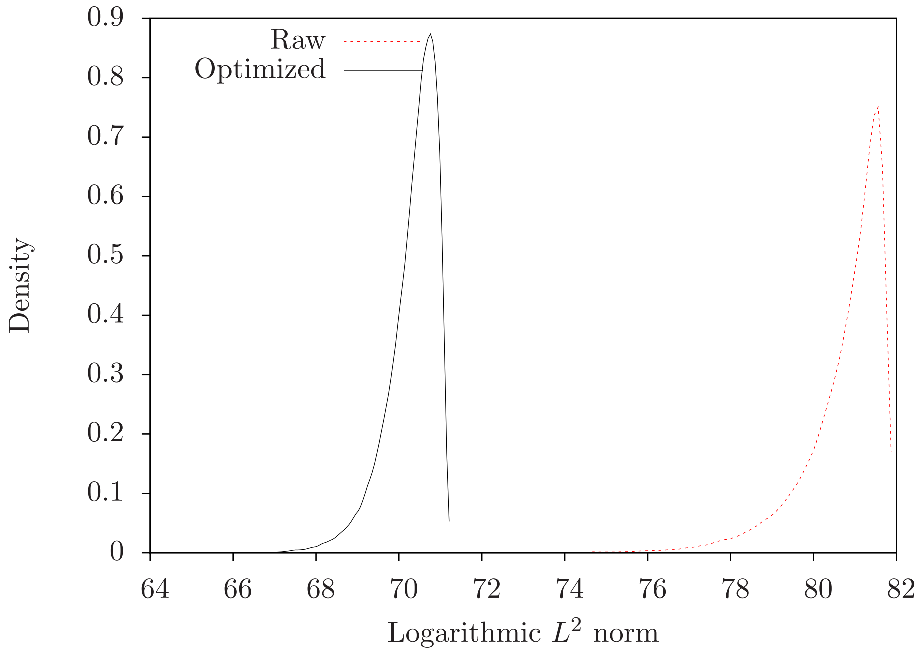

Figure 5.4: Translation and local descent: better

Further improvement. Thorsten Kleinjung (personal communication) describes a method which can further reduce the norm. Before translation, we attempt several cubic rotations by $f(x) + \delta x^3 g(x)$ for small δ's on the raw polynomial $f(x)$. This gives some variation during the optimization. For each rotated polynomial, we repeat the optimization procedure and record the minimum norm found.

The variation gives some benefits in practice. We consider the same data set of 10^5 polynomials used in Figure 5.2. In experiments, we rotate polynomials by $|\delta| \leq 256$ and optimize the size using the above method. Figure 5.5 shows the (discrete) density distribution of logarithmic L_2-norm for the raw and optimized polynomials. This method can further reduce the average logarithmic L_2-norm to 69.84 with a standard deviation 0.56.

Trade-off between size and root

The raw polynomial often has a small c_5, which permits a larger rotation bound in root optimization. The size optimization procedure leads to a much larger c_5 due to translation. This may lead to a smaller rotation space. We could have optimized the

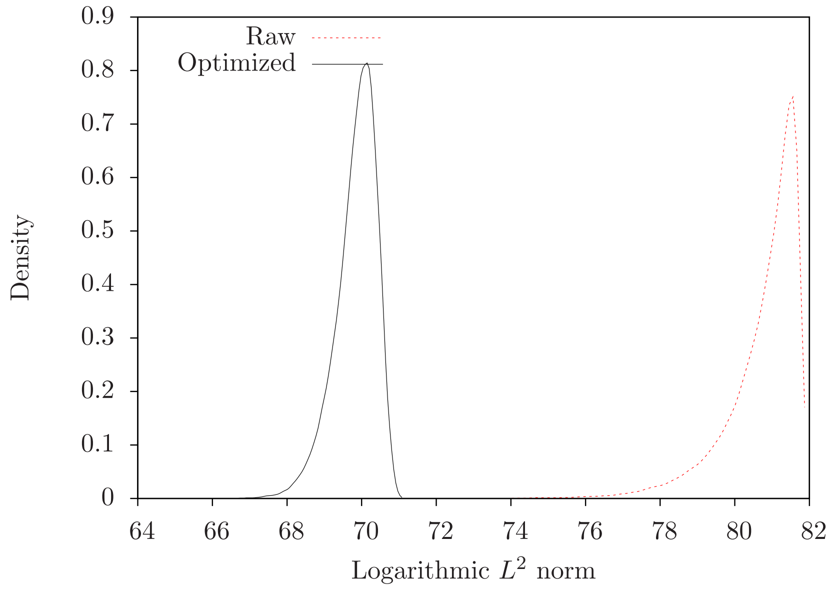

Figure 5.5: Translation and local descent: further improvement

root property (of the raw polynomial) first and then optimized the size by translating and changing the skewness. If the root property is outstanding, we might expect that it can better than the size-root (in order) optimization. However, we give a heuristic argument that this is difficult in practice.

Let $f_{u,v,w}(x)$ be the rotated polynomial in Equation (5.3). If $c_3 \sim O(m_1)$ and $c_0 \sim c_3 s^3$, $c_1 \sim c_3 s^2$, $c_2 \sim c_3 s$, we have an upper bound $O(s^6)$ for the rotation space. We want to estimate the expected minimum $\alpha(F)$ after K polynomials are chosen where K is about s^6.

Assume $\alpha(F)$ follows a normal distribution as in Figure 3.1. The data has mean $\mu = -0.257$ and deviation $\sigma = 0.824$. Equation (3.6) shows that the expected minimum α is about $-0.257 - 1.648\sqrt{3 \log s}$. Hence the expected minimum α is proportional to the square root of logarithmic scale of skewness.

We consider the situation of sextic polynomials for RSA-768. Let $s = 10^{10}$, which is reasonably large for raw polynomials. Equation (3.6) gives an expected minimum $\alpha = -13.80$ after s^6 polynomials are generated. In an ideal situation, we can expect to find such α without affecting the size. In practice, a rotation space of s^6 is very likely to destroy the size and it is very hard to find polynomials with such α while keeping the size constrained. We can also apply a size optimization of two variables (translation and skewness) afterwards. However, such optimization is restricted as no rotation can be used and is likely to be ineffective.

On the other hand, if we first conduct size optimization, Figure 5.5 shows that a reduction of 10 in norm is common. A following root optimization can further reduce the combined norm by 7–11, despite increasing the size. Put together, size-root (in order) optimization often behaves much better than root-size optimization in experiments. Therefore, it is suggested to optimize the size property first and then the root property.

Chapter 6

Root optimization

Many polynomials can have comparable size after size optimization. We produce and choose the best polynomials in terms of good α-values. This roughly requires that the polynomials have many roots modulo small prime and prime powers. This step is referred to as root optimization and involves polynomial rotation. We discuss some algorithms for root optimization in this chapter.

6.1 Root sieve

We focus on root optimization for sextic polynomials in this chapter. Given a polynomial pair (f, g), we want to find a rotated polynomial with similar size but better root properties. For sextic polynomials, one can use quadratic rotations defined by $f_{w,u,v}(x) = f(x) + (wx^2 + ux + v)\, g(x)$ for some integers w, u, v. Let W, U, V be upper bounds for w, u, v. W is often small (for the integers we are interested in). To reduce the search space, it is sufficient to consider linear rotations defined by $f_{u,v}(x) = f(x) + (ux + v)\, g(x)$. We want to choose (u, v) such that $f_{u,v}(x)$ has a small α-value.

The straightforward way is to look at individual polynomials $f_{u,v}(x)$ for all possible (u, v)'s and compare their α-values. This is time-consuming and impractical since the permissible bounds on U, V are often huge.

This chapter is joint work with Richard Brent and Emmanuel Thomé.

Murphy [73] described a sieve-like procedure, namely the root sieve, to find polynomials with good root properties. It is a standard method to optimize the root property in the final stage of polynomial selection. We describe Murphy's root sieve in Algorithm 6.1. Let B be the bound for small primes and U, V be bounds for the linear rotation. The root sieve fills an array with estimated α-values. The α-values are estimated from p-valuation for small primes $p \le B$. Alternatively, it is sufficient to calculate the summation of the weighted p-valuation $\nu_p(F) \log p$ for the purpose of comparison. The idea of the root sieve is that, when r is a root of $f_{u,v}(x) \pmod{p^e}$, it is also a root of $f_{u+ip^e,v+jp^e}(x) \pmod{p^e}$.

Algorithm 6.1: Murphy's root sieve

```
input : a polynomial pair $f, g$; integers $U$, $V$, $B$;
output: an array of approximated $\alpha$-values of dimension $U \times V$;
1 for $p \le B$ do
2     for $e$ where $p^e \le B$ do
3         for $x \in [0, p^e - 1]$ do
4             for $u \in [0, p^e - 1]$ do
5                 compute $v$ in $f(x) + (ux + v)\, g(x) \equiv 0 \pmod{p^e}$;
6                 update $\nu_p(f_{u+ip^e,v+jp^e})$ by sieving;
```

In general, the root sieve does not affect the projective roots significantly. It is sufficient to only consider the affine roots' contribution to the α-value. In the end, we identify slots with small α-values in the sieving array. For each such slot (polynomial), we can compute a more accurate α-value with a large bound B and then re-optimize its size (by translation only).

We consider the asymptotic complexity of Murphy's root sieve.

$$\begin{aligned}\sum_{\substack{p \le B \\ p \text{ prime}}} \left(\sum_{e=1}^{\lfloor \frac{\log B}{\log p} \rfloor} p^e p^e \left(O(1) + \frac{UV}{p^{2e}} \right) \right) &= O\left(\frac{B^3}{\log B} \right) + UV \sum_{\substack{p \le B \\ p \text{ prime}}} \left\lfloor \frac{\log B}{\log p} \right\rfloor \\ &\approx UV \log B \int_2^B \frac{1}{\log^2 p} dp \\ &= O\left(UV \frac{B}{\log B} \right).\end{aligned}$$

We are interested in small primes and hence $B/\log B$ is small. The sieving bounds U, V dominate the running-time $O(UVB/\log B)$.

The classical root sieve works for small bounds U, V. For sextic polynomials, the permissible ranges for U, V are huge. We discuss some better methods for root optimization based on the root sieve.

6.2 A faster root sieve

In the root sieve, we identify the number of roots of rotated polynomials $f_{u,v}(x)$ for small primes and prime powers. In most cases, the roots are simple, and hence their average p-valuation follows Equation (3.4). There is no need to count the lifted roots for them. We describe a faster root sieve based on this idea.

We use the following facts based on Hensel's lemma. Suppose r_1 is a simple root of $f(x) \pmod p$. There exists a unique lifted root r_e of $f(x) \pmod{p^e}$ for each $e > 1$. In addition, each lifted root r_e is a simple root of $f(x) \pmod p$. For convenience, we say r_e is a simple root of $f(x) \pmod{p^e}$ if $f'(r_e) \not\equiv 0 \pmod p$.

Let r_e be a simple root of a rotated polynomial $f_{u,v}(x) \pmod{p^e}$ for $e \geq 1$. It is clear that $f_{u+ip^e,v+jp^e}(x) \equiv f_{u,v}(x) \pmod{p^e}$ for integers i, j. It follows that r_e is also a simple root of the rotated polynomials $f_{u+ip^e,v+jp^e}(x) \pmod{p^e}$. Given a simple root r_1 of a polynomial $f_{u,v}(x) \pmod p$, the contribution of the root r_1 to $\nu_p(F_{u,v})$ is $p/(p^2-1)$. We can update the score for all rotated polynomials $f_{u+ip,v+jp}(x)$ in a sieve.

If r_e is a multiple root of $f(x) \pmod{p^e}$ for some $e \geq 1$, there are two possible cases. If $f(r_e) \equiv 0 \pmod{p^{e+1}}$, then $\forall l \in [0, p)$, $f(r_e + lp^e) \equiv 0 \pmod{p^{e+1}}$. There are p lifted roots r_{e+1} satisfying $r_{e+1} = (r_e + lp^e) \equiv r_e \pmod{p^e}$, $\forall l \in [0, p)$. In addition, the lifted roots r_{e+1} are multiple since $f'(r_{e+1}) \equiv 0 \pmod p$. On the other hand, if $f(r_e) \not\equiv 0 \pmod{p^{e+1}}$, r_e cannot be lifted to a root modulo p^{e+1}.

Let r_e be a multiple root of a rotated polynomial $f_{u,v}(x) \pmod{p^e}$ for $e \geq 1$. It is also a multiple root for all rotated polynomials $f_{u+ip^e,v+jp^e}(x) \pmod{p^e}$.

Let r be a fixed integer modulo p. We discuss the case when r is a multiple root for some rotated polynomial $f_{u,v}(x) \pmod p$. We see that $f(r) + (ur+v)g(r) \equiv 0 \pmod p$

and $f'(r) + ug(r) + (ur + v)g'(r) \equiv 0 \pmod p$. Since $(ur + v) \equiv -f(r)/g(r) \pmod p$, we get $ug^2(r) \equiv f(r)g'(r) - f'(r)g(r) \pmod p$.

Therefore, only 1 in p u's admits a multiple root at $r \pmod p$. For the other u's, we can compute v and update the simple contribution $p/(p^2 - 1)$ to slots in the sieve array. If r is a multiple root of $f_{u,v}(x) \pmod p$, we have to lift to count the lifted roots. We discuss the details of the lifting method in the following sections. For the moment, we describe the improved root sieve in Algorithm 6.2.

Algorithm 6.2: A faster root sieve

```
input  : a polynomial pair $f, g$; integers $U$, $V$, $B$;
output: an array of approximated $\alpha$-values of dimension $U \times V$;
1  for $p \leq B$ do
2      for $x \in [0, p-1]$ do
3          compute $\tilde{u}$ such that $\tilde{u}g^2(x) \equiv f(x)g'(x) - f'(x)g(x) \pmod p$;
4          for $u \in [0, p-1]$ do
5              compute $v$ such that $f(x) + uxg(x) + vg(x) \equiv 0 \pmod p$;
6              if $u \neq \tilde{u}$;
7              then
8                  update $\nu_p(f_{u+ip,v+jp})$ in sieving;
9              else
10                 lift to count multiple roots of $f_{\bar{u},\bar{v}}(x) \pmod{p^e}$ such that
                   $(\bar{u}, \bar{v}) \equiv (u, v) \pmod p$, $\bar{u}, \bar{v} \leq p^e$, $p^e \leq B$ and then sieve;
```

Let $r = x$ be fixed in Line 2. In Line 3, we compute $\tilde{u}$ such that r is a multiple root of $f_{\tilde{u},v}(x)$ for some v. If $u \neq \tilde{u}$, r is a simple root for this u, and some v which will be computed in Line 5. If $u = \tilde{u}$, $f_{u,v}(x)$ admits r as a multiple root. We need to lift (up to degree d) to count the roots. The running-time to do this is about

$$\sum_{\substack{p \leq B \\ p \text{ prime}}} \left(p \left((p-1)\frac{UV}{p^2} + O\left(\frac{UV}{p^2}\right)\right)\right) = O\left(UV \frac{B}{\log B}\right).$$

The asymptotic running-time of the improved root sieve has the same magnitude as Algorithm 6.1. In practice, however, we can save arithmetic by avoiding considering the prime powers in many cases.

For comparison, Murphy's root sieve takes about $UV \sum_{\substack{p \leq B \\ p \text{ prime}}} (\log B)/(\log p)$ operations, while Algorithm 6.2 takes about $UV \sum_{\substack{p \leq B \\ p \text{ prime}}} 1$ operations. Taking $B = 200$ for instance. $\sum_{\substack{p \le 200 \\ p \text{ prime}}} (\log 200)/(\log p) \approx 2705$ and $\sum_{\substack{p \le 200 \\ p \text{ prime}}} 1 = 46$.

6.3 A two-stage method

We give a two-stage algorithm for the root optimization. The algorithm is motivated by previous work by Gower [38], Jason Papadopoulos (personal communication), Stahlke and Kleinjung [95], which suggested to consider congruence classes modulo small primes.

If the permissible rotation bounds U, V are large, the root sieve can take a long time for each polynomial. This is even more inconvenient if there are many polynomials. We describe a faster method for root optimization based on the following ideas.

A polynomial with only a few roots modulo small prime powers p^e is less likely to have a good α-value. Therefore, rotated polynomials with many roots modulo small prime powers are first detected. A further root sieve for larger prime powers can then be applied.

In the first stage, we find a (or some) good rotated pair $(u_0, v_0) \pmod{p_1^{e_1} \cdots p_m^{e_m}}$ such that the polynomial $f_{u_0,v_0}(x)$ has many roots modulo (very) small prime powers $p_1^{e_1}, \cdots, p_m^{e_m}$. Let B_s be an upper bound for $p_m^{e_m}$. In the second stage, we apply the root sieve in Algorithm 6.2 to the polynomial $f_{u_0,v_0}(x)$ for larger prime powers up to some bound B.

6.3.1 Stage 1

Given $f(x)$, we want to find a rotated polynomial $f_{u_0,v_0}(x)$ which has many roots modulo small primes and small prime powers. Let the prime powers be $p_1^{e_1}, \cdots, p_m^{e_m}$. There are several ways to generate $f_{u_0,v_0}(x)$.

First, we can root-sieve a matrix of pairs (u, v) of size $(\prod_{i=1}^{m} p_i^{e_i})^2$ and pick up the best (u, v) pair(s) as (u_0, v_0). If the matrix is small, there is no need to restrict the bound in the root sieve to be B_s. We can use the larger bound B. If the matrix is large, however, the root sieve might be slow. We describe a faster strategy.

We first find m (or more[*]) individual polynomials $f_{u_i,v_i,p_i}(x)$ $(1 \le i \le m)$ each of

[*]For each p_i, we may generate several polynomials. In Stage 2 we consider multi-sets of combinations.

which has many roots modulo small $p_i^{e_i}$. The values u_i and v_i are bounded by $p_i^{e_i}$. We combine them to obtain a polynomial $f_{u_0,v_0}(x) \pmod{\prod_{i=1}^m p_i^{e_i}}$ using the Chinese Remainder Theorem. The polynomial $f_{u_0,v_0}(x) \pmod{p_i^k}$ has the same number of roots as the individual polynomials $f_{u_i,v_i,p_i}(x) \pmod{p_i^k}$ for $1 \le k \le e_i$. Hence the combined polynomial is likely to have many roots modulo small prime powers $p_1^{e_1}, \cdots, p_m^{e_m}$.

Individual polynomials. To find individual polynomials $f_{u_i,v_i,p_i}(x)$ that have many roots modulo small prime powers $p_i^{e_i}$, we can root-sieve a square matrix $p_i^{e_i} \times p_i^{e_i}$ and pick up the good pairs.

Alternatively, we use a lifting method together with a p_i^2-ary tree data structure. This seems to be more efficient when $p_i^{2e_i}$ is large. For each $p_i^{e_i}$, we construct a tree of height e_i and record good (u, v) pairs during the lift. The lift is based on Hensel's lemma. For convenience, we fix $f(x) \pmod p$ where $p = p_i$ and $e = e_i$ for some i. We describe the method.

We create a root node. In the base case, we search for polynomials $f_{u,v}(x) \pmod p$ $(u, v \in [0, p))$ which have many roots and record them in the tree. There could be at most p^2 level-1 leaves for the root node.

Let a level-1 leaf be $(u, v) \pmod p$. A simple root is uniquely lifted. If the polynomial $f_{u,v}(x) \pmod p$ only gives rise to simple roots, we already know the exact p-valuation of $f_{u,v}(x)$. In case of multiple roots, we need to lift and record the lifted pairs. Assume that $f_{u,v}(x) \pmod p$ has some multiple root r_m and some simple root r_s. We want to update the p-valuation for rotated polynomials

$$f(x) + \left(\left(u + \sum_{k=1}^{e-1} i_k p^k\right)x + \left(v + \sum_{k=1}^{e-1} j_k p^k\right)\right) g(x) \pmod{p^e} \tag{6.1}$$

where each $i_k, j_k \in [0, p)$. We give the following procedure for the lifting.

1. For a simple root r_s, we find out which of the rotated polynomials $f_{u+ip,v+jp}(x) \pmod{p^2}$ admit r_s as a root. If $f_{u+ip,v+jp}(r_s) \equiv 0 \pmod{p^2}$ for some i, j, then

$$(ir_s + j)g(r_s) + f_{u,v}(r_s)/p \equiv 0 \pmod p. \tag{6.2}$$

Hence the set of (i, j)'s satisfies a linear congruence equation. For simple roots, there is no need to compute the lifted root. It is sufficient to update the p-valuation contributed by r_s to polynomials $f_{u+ip,v+jp}(x)$.

2. Let r_m be a multiple root of $f_{u,v}(x) \pmod p$. If a rotated polynomial $f_{u+ip,v+jp}(x) \pmod{p^2}$ admits r_m as a root for some (i, j), all the $\{r_m + lp\}$ $(0 \le l < p)$ are also roots for the polynomial. In addition, $f'_{u+ip,v+jp}(r_m + lp) \equiv 0 \pmod p$. We record the multiple roots $\{r_m + lp\}$ together with the $\{(u + ip, v + jp)\}$ pairs. The procedure also works for the lift from p^e to p^{e+1} for higher e's.

We consider the memory usage of the p^2-ary tree. If r is a root of $f_{u,v}(x) \pmod p$, Equation (6.2) shows a node $(u, v) \pmod p$ gives p lifted nodes $(u+ip, v+jp) \pmod{p^2}$ for some (i, j)'s. Since $f_{u,v}(x) \pmod p$ can potentially have other roots besides r, there could be at most p^2 pairs $(u + ip, v + jp) \pmod{p^2}$. The procedure also needs to record the multiple roots for each node. We are mainly interested in the bottom level leaves of the tree, those $(u, v) \pmod{p^e}$. It is safe to delete the tree path which will not be used anymore. Hence a depth-first lifting method can be used. In practice, the memory usage is often smaller than a sieve array of size p^{2e}.

For each p, we find a polynomial that either has many simple roots or many multiple roots which can be lifted further. Tiny primes p's are more likely to be ramified. Hence we are more likely to meet multiple roots for tiny p.

CRTs. For each p, we have generated some polynomial(s) rotated by $(ux + v)g(x) \pmod{p^e}$ which have comparably good expected p-valuation. For convenience, we identify the rotated polynomial by pair (u, v).

Stage 1 repeats for prime powers $p_1^{e_1}, \cdots, p_m^{e_m}$. We generate the multi-sets combinations of pairs $\{(u, v)\}$ and recover a set of $\{(u_0, v_0)\} \pmod{\prod_{i=1}^m p_i^{e_i}}$. We fix such a pair (rotated polynomial) $(u_0, v_0) \pmod{\prod_{i=1}^m p_i^{e_i}}$.

The whole search space is an integral lattice of $\mathbb{Z}^2$. In Stage 2, we want to root-sieve on the sublattice points defined by $(u_0 + \gamma \prod_{i=1}^m p_i^{e_i}, v_0 + \beta \prod_{i=1}^m p_i^{e_i})$ where $(\gamma, \beta) \in \mathbb{Z}^2$. The sublattice points are expected to give rotated polynomials with good root properties, since the polynomials have many roots modulo $p_1^{e_1}, \cdots, p_m^{e_m}$.

We often choose the p_i's to be the smallest consecutive primes since they are likely to contribute most to the α-value. The exponents e_i in prime powers $p_i^{e_i}$ need some more inspection. If e_i is too small, the sieving range $(\gamma, \beta) \in \mathbb{Z}^2$ can be large. If e_i is too large, $\prod_{i=1}^m p_i^{e_i}$ is large and hence some polynomials which have good size property might be omitted in the root sieve. One heuristic is to choose $p_i^{e_i} \lesssim p_j^{e_j}$ for $i > j,\ i, j \in [1, m]$. To determine m, one can choose $\prod_{i=1}^m p_i^{e_i}$ to be comparable to the sieving bound U. We can discard those (u_0, v_0)'s such that $u_0 > U$. If u_0 is comparable to U, it is sufficient to use a line sieve for constant rotations.

Remark 6.1. In the implementation, we may want to tune the parameters by trying several sets of parameters such as various p_i's and e_i's. We can run a test root sieve in short intervals. The set of parameters which generates the best score is then used.

6.3.2 Stage 2

In Stage 2, we apply the root sieve in Algorithm 6.2 to polynomial $f_{u_0,v_0}(x)$, perhaps with some larger prime bound. In the root sieve, one can reuse the code from Stage 1, where the updates of α-values can be batched. We describe the method as follows.

Sieve on sublattice. Let $M = \prod_{i=1}^m p_i^{e_i}$ and (u_0, v_0) be fixed from Stage 1. In the second stage, we do the root sieve for (larger) prime powers on the sublattice defined by $\{(u_0 + \gamma M, v_0 + \beta M)\}$ where $\gamma, \beta \in \mathbb{Z}$. Let p be a prime and $r_k \pmod{p^k}$ be a root of

$$f(x) + \Big((u_0 + \gamma M)x + (v_0 + \beta M)\Big)g(x) \pmod{p^k}$$

for some fixed integers γ, β. The sieve on the sublattice follows from

$$f(r_k) + \Big(\big(u_0 + M(\gamma + ip^k)\big)r_k + \big(v_0 + M(\beta + jp^k)\big)\Big)\, g(r_k) \equiv 0 \pmod{p^k}$$

for integers $i, j \in \mathbb{Z}$. We consider the root sieve for a fixed prime p in Algorithm 6.2.

Let $f(x), g(x), M, u_0, v_0$ be fixed from Stage 1. In Algorithm 6.2, we assume u, r

are fixed for the moment. Let p be a prime not dividing M. The sieve array has approximate size $\lfloor U/M \rfloor \times \lfloor V/M \rfloor$. Each element (γ, β) in the sieve array stands for a point $(u_0 + \gamma M, v_0 + \beta M)$ in $\mathbb{Z}^2$. We solve for v in $f(r) + urg(r) + vg(r) \equiv 0 \pmod{p}$. Knowing (u, v), we can solve for (γ, β) in $u \equiv u_0 + \gamma M \pmod{p}$ and $v \equiv v_0 + \beta M \pmod{p}$, provided that $p \nmid M$.

For the moment, we fix integers γ, β. If r is a simple root, it is sufficient to sieve $(\gamma + ip, \beta + jp)$ for various (i, j)'s and update the p-valuation $p \log p/(p^2 - 1)$ to each slot. If r is a multiple root, we can use a similar lifting procedure as in Stage 1. We describe the recursion to deal with multiple roots in Algorithm 6.3.

Algorithm 6.3: Recursion for multiple roots

```
input : a polynomial pair $f, g$; integers $U$, $V$, $B$; node $(u, v)$, tree height $e$,
        current level $k$, prime $p$;
output: updated $\alpha$-values array;
1 for multiple roots $r$ of $f_{u,v}(x) \pmod{p}$ do
2     for $k < e$ do
3         compute $(i, j)$'s in $(ir + j)g(r) + f_{u,v}(r)/p^k \equiv 0 \pmod{p}$;
4         create child nodes $(u + ip^k, v + jp^k)$ with roots $\{r + lp^k\}$, $\forall l \in [0, p)$;
5         recursively call Algorithm 6.3 on $(u, v)$'s leftmost child node;
6         change coordinates for current node $(u, v)$ and sieve;
7         delete current node and move to its sibling node or parent node;
```

From Stage 1, we know u_0, v_0. In Algorithm 6.2, we fix u, r and solve for v. Given a multiple root r of $f(x) \pmod{p^k}$, we find pairs (u', v') such that $f_{u',v'}(r) \equiv 0 \pmod{p^{k+1}}$ where $u' \equiv u \pmod{p^k}$ and $v' \equiv v \pmod{p^k}$. We can construct nodes representing the (u', v') pairs together with their roots. In the recursion, we compute the lifted nodes in a depth-first manner. Once the maximum level p^e is reached, we do the root sieve for the current nodes and delete the nodes which have been sieved.

When a lifted tree node $(u', v') \pmod{p^k}$ is created, the number of roots for $f_{u',v'}(x) \pmod{p^k}$ is known. In the root sieve, the α-scores can be updated in a batch for all the roots of $f_{u',v'}(x) \pmod{p^k}$. For each node (u', v'), we also need to compute the corresponding coordinates in the sieve array.

Primes p dividing M. We have assumed that p is a prime not dividing M. From Stage 1, M is a product of prime powers $p_i^{e_i}$ for $1 \le i \le m$. For accuracy, we can also consider primes powers $p_i^{e'_i}$ with $e'_i \ne e_i$ such that p_i appears in the M. Let r be root of $f_{u,v}(x) \pmod p$. If r is a simple root, there is no need to consider any liftings. Hence we consider polynomials $f_{u,v}(x) \pmod p$ which have a multiple root.

We fix some $p = p_i$ and $e = e_i$, which are used in Stage 1. Let u, v, p be fixed in Algorithm 6.2. Let e' be the exponent of p that we want to consider in Stage 2. There are two cases depending on e'.

If $e' \le e$, the points on the sublattice have equal scores contributed by roots modulo $p^{e'}$. It is sufficient to look at the multiple roots modulo p^k for $k \le e'$. In Algorithm 6.2, we either sieve all slots of the array or do not sieve at all. Given u, v, p, k, if $v \equiv v_0 \pmod{p^k}$ in $v \equiv v_0 + \beta M \pmod{p^k}$, we need to sieve the whole array. This can be omitted because it will give the same result for each polynomial and we only want to compare polynomials. If $v_0 \not\equiv v \pmod{p^k}$, no slot satisfies the equation. Therefore, it is safe to skip the current iteration when $e' \le e$.

If $e' > e$, the rotated polynomials $(u, v) \pmod{p^k}$ for $e < k \le e'$ may have different behaviors. We describe some modifications in the lifting procedure. Let $u \equiv u_0 \pmod{p^k}$, r, v_0 be fixed in Algorithm 6.2. We compute v. We want to know which points (polynomials) on the sieve array are equivalent to $(u, v) \pmod{p^k}$.

For $k \le e$, the situation is similar to the case when $e' \le e$. If the equation $v \equiv v_0 \pmod{p^k}$ is satisfied, we record the node $(u, v) \pmod{p^k}$ for further liftings. There is no need to sieve since all slots on the sieve array have equal scores for roots modulo p^k. If $v_0 \not\equiv v \pmod p$ where $k = 1$, we have neither to root-sieve nor record the node. Let $e < k \le e'$. If $(u', v') \pmod{p^k}$ satisfies $u' \equiv u \pmod p$ and $v' \equiv v \pmod p$, we need

$$v_0 + \beta M \equiv v' \pmod{p^k}.$$

The equation is solvable for β only if

$$v_0 \equiv v' \pmod{p^k}.$$

Hence it is safe to discard those $(u, v) \pmod p$ such that $u \equiv u_0 \pmod p$ but $v \not\equiv v_0 \pmod p$.

On the other hand, we consider some k in $e < k \leq e'$. In the lifting procedure, we record nodes without sieving until we reach the level-$(e+1)$ nodes. Starting from a node (u, v) modulo p^{e+1}, that is $k > e$, we want to solve the equation

$$v_0 + \beta M \equiv v \pmod{p^k}.$$

The depth-first lifting procedure shows that

$$v_0 \equiv v \pmod{p^e}.$$

Hence β is solvable in the following equation

$$\frac{v_0 - v}{p^e} + \frac{M}{p^e}\beta \equiv 0 \pmod{p^{k-e}}$$

since $\gcd(M/p^e, p) = 1$. In the root sieve, we step the array by $\beta + jp^{k-e}$ for various j.

6.3.3 Further remarks and improvements

Let (U, V) be the rotation bounds for the polynomial. The root sieve in Algorithm 6.2 runs asymptotically in time $UVB/\log B$ (ignoring constant factors). In Stage 2, the searching space is restricted to a sublattice determined by $M = \prod_{i=1}^{m} p_i^{e_i}$, where the parameters p_i's depend on Stage 1. Hence, the root sieve in Stage 2 runs in time about $UVB/(M^2 \log B)$.

In Stage 2, the points not on the sublattice are discarded since compared to points on the sublattice they have worse p-valuation for those p's in Stage 1. We assumed that they were unlikely to give rise to polynomials with good root properties. However, a polynomial could have good $\alpha(F)$ while some p in M gives a poor p-valuation. This often happens when some p'-valuation of $p' \nmid M$, those ignored in Stage 1, is exceptionally good, and hence mitigates some poor p-valuation where $p \mid M$.

Alternatively, we can use a root sieve to identify good rotations in Stage 1 for some small sieving bounds (U', V'). Then we examine the pattern of p-valuation of these polynomials and decide the congruence classes used in Stage 2.

We have ignored the size property of polynomials in the algorithms. We have assumed that polynomials rotated by similar (u, v)'s have comparable size. In practice, some trials are often needed to decide the sieving bounds (U, V). We give some further remarks regarding the implementation.

Block sieving. The root sieve makes frequent memory references to the array. However, there is only one arithmetic operation for each array element. The time spent on retrieving memory often dominates. For instance, the root sieve may cause cache misses if the sieve on p steps over a large sieve array. A common way to deal with cache misses is to sieve in blocks.

We partition the sieving region into multiple blocks each of whose size is at most the cache size. In the root sieve, we attempt to keep each block in the cache while many arithmetic operations are applied. The fragment of the block sieving is described in Algorithm 6.4.

Algorithm 6.4: Block sieving

```
input : a polynomial pair $f, g$; integers $U$, $V$, $B$;
output: an array of approximated $\alpha$-values in dimension $U \times V$;
1 for $x \le B$ do
2     for each block do
3         for $p$ where $x < p \le B$ do
4             ······
```

We have also changed the order of iterations to better facilitate the block sieving. This might give some benefits due to the following heuristic. In Algorithm 6.2, when p is small, polynomial roots x modulo p are small. The number of roots $x \le p$ blocked for sieving is also limited. Instead we block primes p. If x is small, there are still many p's which can be blocked.

For multiple roots, we might need to sieve in steps p^k for $k \ge 1$. When p^k is not too small, each block has only a few (or none) references. In this case, we may use a sorting-based sieving procedure like the bucket sieve [3].

Arithmetic. The coefficients of the rotated polynomials are multiple precision numbers. Since p^e can often fit into a single precision integer, it is sufficient to use single precision in most parts of the algorithms.

The algorithms involve arithmetic on p^k for all $k \leq e$. It is sufficient to store polynomial coefficients modulo p^e and do the modulo reduction for arithmetic modulo p^k. Let D be a multiple precision integer. In the algorithm, we use a single precision integer S instead of D where $S = D \pmod{p^e}$. If $x \equiv D \pmod{p^k}$ for $k \leq e$, it is clear that $x \equiv S \pmod{p^k}$. Hence we can use the S in the root optimization.

In addition, the range of possible α-values is small. We may use short integers to approximate the α-values instead of storing floating point numbers. This might save some memory.

Quadratic rotation. Sextic polynomials have been used in the factorizations of many large integers such as RSA-768. Rotations by quadratic polynomials can be used for sextic polynomials if the coefficients and skewness of the polynomials are large. We have assumed that W is small in $f_{w,u,v}(x)$ and we use linear rotations in this section. If the permissible bound for W is large, we can use a similar idea to that in Stage 1 to find good sublattices in three variables. At the end of Stage 1, a set of polynomials having good α-values are found which are defined by rotations of (w_0, u_0, v_0)'s. In Stage 2, we root-sieve on the sublattice $\{(w_0 + \delta M, u_0 + \gamma M, v_0 + \beta M)\}$ where $\delta, \gamma, \beta \in \mathbb{Z}$.

Chapter 7

Polynomial selection using lattice reduction

In Chapters 4–6, we discussed the polynomial selection with two polynomials, one of which is linear. In the number field sieve, two non-linear polynomials can also be used and this may give some benefits. In this chapter, we review some methods to generate two non-linear polynomials based on lattice reduction. In addition, we discuss some similar (lattice-based) methods for generating degree-$(d, 1)$ polynomial pairs.

7.1 Use of two non-linear polynomials

We want to find two irreducible, coprime, non-linear polynomials f_1, f_2 which have a common root m modulo n. For convenience, we assume that both polynomials are monic and have the same degree d.

$$f_1(x) = \sum_{i=0}^{d} a_i x^i, \quad f_2(x) = \sum_{i=0}^{d} b_i x^i.$$

Let $\mathbb{Z}[\alpha] = \mathbb{Z}[X]/\langle f_1 \rangle$ and $\mathbb{Z}[\beta] = \mathbb{Z}[X]/\langle f_2 \rangle$ be the rings generated by zeros of f_1, f_2 respectively. K_1 and K_2 denote their fields of fractions. There exist ring homomorphisms

ϕ_α and ϕ_β such that

$$\phi_\alpha : \mathbb{Z}[\alpha] \to \mathbb{Z}/n\mathbb{Z}, \quad \phi_\beta : \mathbb{Z}[\beta] \to \mathbb{Z}/n\mathbb{Z}.$$

We want to find a set $\mathcal{S}$ of (c, e) pairs $(c, e \in \mathbb{Z})$ such that

$$\prod_{(c,e)\in\mathcal{S}} (c - e\alpha) = \gamma^2 \text{ and } \prod_{(c,e)\in\mathcal{S}} (c - e\beta) = \delta^2$$

for some $\gamma \in \mathbb{Z}[\alpha]$ and $\delta \in \mathbb{Z}[\beta]$. Assume that the set $\mathcal{S}$ is found. $\phi_\alpha(\gamma) \equiv u \pmod{n}$ and $\phi_\beta(\delta) \equiv v \pmod{n}$. It follows that

$$u^2 \equiv \prod_{(c,e)\in\mathcal{S}} \phi_\alpha(c - e\alpha) \equiv \prod_{(c,e)\in\mathcal{S}} (c - em) \equiv \prod_{(c,e)\in\mathcal{S}} \phi_\beta(c - e\beta) \equiv v^2 \pmod{n}.$$

The congruence of squares $u^2 \equiv v^2 \pmod{n}$ may give a factor of n.

Crandall and Pomerance [28] have shown that it is advantagenous to use two non-linear polynomials. Let F_1, F_2 be the homogeneous polynomials of f_1, f_2. In sieving, we find smooth polynomial values $F_1(c, e)F_2(c, e)$. The polynomial values $F_1(c, e)F_2(c, e)$ are known to consist of two factors of similar size. Let X be an upper bound on the values $F_1(c, e)F_2(c, e)$ and B be the smoothness bound. The B-smoothness change for polynomial values is $\rho^2(\log(X)/2\log(B)) = \rho^2(u/2) \approx u^{-u}2^u$ where $u = \log(X)/\log(B)$. This is 2^u times better than $\rho(u) \approx u^{-u}$ if X is not known to consist of two factors. Let the pair (d_1, d_2) be the degree of polynomials in number field sieve. In the number field sieve with polynomials of degree $(d - 1, 1)$, the factor from the linear polynomial is much smaller than the factor from the algebraic polynomial. It is beneficial to use two polynomials of degree $(d/2, d/2)$, so that the polynomial values have similar size.

For convenience, let the polynomials have degree (d, d). The resultant of two polynomials is the determinant of the Sylvester matrix of the two polynomials. It is a homogeneous polynomial of degree $2d$ in the coefficients of the polynomials, and hence may be used to measure the size property. The resultant is a multiple of n. We prefer it to be as small as possible, so we want to find polynomials of degree (d, d) with resultant $\pm n$. Prest and Zimmermann [87] described some heuristic evidence that such polynomials exist. Polynomials of degree (d, d) have $2d+2$ coefficients. We choose $2d+2$ random integers (as coefficients) bounded by $O(n^{1/(2d)})$. There are about $(n^{1/(2d)})^{2d+2} = n^{1+1/d}$ possible values for the resultant. It is reasonable to expect that about $n^{1/d}$ of them equal n.

Montgomery [63] showed that finding polynomials of degree (d, d) with coefficients $O(n^{1/(2d)})$ was equivalent to find a geometric progression modulo n of length $2d - 1$ bounded by $O(n^{1-1/d})$. He gave a good way to do this for $d = 2$ (see Section 7.2).

Given a set of non-linear polynomials F_1, F_2, we want to compare and optimize these polynomials. We may use the logarithmic L^2-norm for the product of polynomials F_1, F_2

$$\frac{1}{2}\log\left(\iint\limits_{\Omega} \left(F_1(x,y)F_2(x,y)\right)^2 \,\mathrm{d}x\,\mathrm{d}y\right) \tag{7.1}$$

where Ω is the sieving region. Murphy's $\mathbb{E}$ function can also be used as a better estimate to rank the top polynomials before the sieving test.

At the moment, we don't know how to optimize the root and size properties for two non-linear polynomials of similar degree. For polynomials of degree $(d, 1)$, the rotations by the linear polynomials are critical to reduce the coefficients and improve the roots without greatly increasing the size. For two non-linear polynomials, a rotation is likely to increase the size significantly.

7.2 Two quadratic polynomials

Montgomery's two-quadratic method [34, 73] is a standard way for generating two quadratic polynomials. Given n, it finds a pair of quadratic polynomials with a common root $m \pmod n$, each of whose coefficients are $O(n^{1/4})$. We find a small geometric progression $\mathbf{c} = (c_0, c_1, c_2)$ of ratio $m \pmod n$. It is not a geometric progression over $\mathbb{Z}$, since otherwise the two polynomials are not coprime. Explicitly, the vector $\mathbf{c}$ can be constructed as follows.

Given n, we fix a prime $p < \sqrt{n}$ for which n is a quadratic residue. We choose r such that $|r - n^{1/2}| \le p/2$ and $r^2 \equiv n \pmod{p}$. The geometric progression is given by $\mathbf{c} = \big(p,\, r,\, (r^2 - n)/p\big)$ in which the ratio m is $r/p \pmod{n}$. It can also be expressed as $\mathbf{c} = (p,\, pm,\, pm^2) \pmod{n}$. The terms in the geometric progression are bounded by $O(n^{1/2})$.

Let $f_1(x) = a_2x^2 + a_1x + a_0$ and $f_2(x) = b_2x^2 + b_1x + b_0$ be two polynomials having a common root m modulo n. Then (a_0, a_1, a_2) and (b_0, b_1, b_2) are orthogonal to the vector $(p,\, pm,\, pm^2)$ modulo n. Two such orthogonal vectors can be given by $(r,\, -p,\, 0)$ and $((rt - c_2)/p,\, -t,\, 1)$ where $t \equiv c_2/r \pmod{p}$ and $c_2 \equiv (r^2 - n)/p \pmod{n}$. We apply the LLL reduction to get two reduced vectors and use them as $(a_0,\, a_1,\, a_2)$ and $(b_0,\, b_1,\, b_2)$.

We consider the size of reduced vectors (coefficients). The two basis vectors have coefficients $O(n^{1/2})$. Let the matrix formed by two basis vectors be M. The determinant (volume) of the lattice spanned by the basis vectors is given by $\sqrt{\det(MM^T)} \approx pn^{1/2}$. Therefore, the shortest vector is about $\sqrt{p}n^{1/4}$. If p is small, the coefficients are $O(n^{1/4})$. The resultant is $O(n)$.

In the two-quadratic method, we use a small geometric progression of length 3. To generate two cubic polynomials (of small coefficients), one need to find a small geometric progression of length 5. It is not known how to generate such geometric progression of length $2d - 1$ efficiently. Alternatively, there exist some methods to produce small geometric progressions of length $d + 1$.

7.3 Two cubic polynomials

Williams' method. Williams [98] described a variant of the two-quadratic method to produce two cubic polynomials. We start with a four-term geometric progression $(1,\, r,\, r^2,\, r^3 - n)$ where $r = n^{1/3} + \delta$ for some δ. The terms r^2 and $r^3 - n$ are bounded by $O(n^{2/3})$ when δ is small. For convenience, we write $r_2 = r^2$ and $r_3 = r^3 - n$. We construct the following basis matrix. The basis (row) vectors represent independent polynomials having r as a root:

$$M = \begin{pmatrix} r & -1 & 0 & 0 \\ r_2 & 0 & -1 & 0 \\ r_3 & 0 & 0 & -1 \end{pmatrix}. \tag{7.2}$$

We apply LLL reduction to the matrix. The reduced vectors give coefficients of the form (a_0, a_1, a_2, a_3). The lattice has determinant $O(n^{2/3})$ and hence the shortest vector is $O(n^{2/9})$. The coefficients of the reduced polynomials are $O(n^{2/9})$. Since the resultant is a homogeneous polynomial of degree 6, it is $O(n^{4/3})$.

Prest and Zimmermann's method. Prest and Zimmermann [87] described an improved method which could further reduce the size and resultant. It considers a geometric progression of length $d+1$ and takes skewed polynomials into account. We consider the case of generating two cubic polynomials.

Let r be an integer close to $n^{1/3}$. We start with a four-term geometric progression $(1, r, r^2, r^3-n)$ as before. To capture the skewed polynomials, we consider the skewness matrix S.

$$S = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & s & 0 & 0 \\ 0 & 0 & s^2 & 0 \\ 0 & 0 & 0 & s^3 \end{pmatrix}.$$

We multiply the skewness matrix S by M in Equation (7.2).

$$MS = \begin{pmatrix} r & -s & 0 & 0 \\ r_2 & 0 & -s^2 & 0 \\ r_3 & 0 & 0 & -s^3 \end{pmatrix}. \tag{7.3}$$

Two reduced vectors give skewed coefficients of the form $(a_0, a_1s, a_2s^2, a_3s^3)$ and $(b_0, b_1s, b_2s^2, b_3s^3)$. We apply the inverse S^{-1} to recover the coefficients.

Let K be the expected length of the shortest vector. Prest and Zimmermann suggested to minimize the "median" size of the coefficients, more precisely $\sqrt{a_0 a_3} \approx K s^{-3/2}$. Here K is about $\left(\det(MS(MS)^T)\right)^{1/6} = s\left((r^3-n)^2 + r^4 s^2 + r^2 s^4 + s^6\right)^{1/6}$. Assume that $s \ll n^{1/3}$. The dominant term is $r^4 s^2$ and hence $K \approx s^{4/3} n^{2/9}$.

The median coefficient $\sqrt{a_0 a_3} \approx K s^{-3/2} \approx s^{-1/6} n^{2/9}$. We want s to be large to minimize the median coefficient. It is reasonable to assume that K is smaller than the norm of the linear polynomial $sx - r$. Therefore, an upper bound on s is given by $K \ll n^{1/3}$, which shows $s \ll n^{1/12}$. We choose the maximum value $s \approx n^{1/12}$. The median coefficient is $O(n^{5/24})$, while the resultant is $O(n^{5/4})$.

Koo, Jo and Kwon's method. Koo, Jo and Kwon [54] extended Montgomery's analysis [63] by considering geometric progressions of length $d+k$ for $1 \le k \le d-1$.

Let $d = 3$ for example. Given n, we fix a prime $p < n^{1/3}$ for which $x^3 \equiv n \pmod{p}$ is solvable. We choose a root r of $x^3 \equiv n \pmod{p}$ to be close to $n^{1/3}$. Denote $r_2 = r^2$ and $r_3 = (r^3 - n)/p$. The geometric progression is given by $\mathbf{c} = \left(p^2, pr, r_2, r_3\right)$ in which the ratio m is $r/p \pmod{n}$. The terms in the geometric progression is bounded by $O(n^{2/3})$. The basis matrix is

$$M = \begin{pmatrix} Kp^2 & 1 & 0 & 0 & 0 \\ Kpr & 0 & s & 0 & 0 \\ Kr_2 & 0 & 0 & s^2 & 0 \\ Kr_3 & 0 & 0 & 0 & s^3 \end{pmatrix}. \tag{7.4}$$

where K is a sufficiently large integer. The reduced vectors have the form $(0, a_0, a_1 s, a_2 s^2, a_3 s^3)$, from which the polynomial coefficients can be recovered. We can also use three polynomials having root $r/p \mod n$ as the basis vectors:

$$M' = \begin{pmatrix} r & -p & 0 & 0 \\ r_2 & 0 & -p^2 & 0 \\ r_3 & 0 & 0 & -p^3 \end{pmatrix}.$$

The skewed version of the basis matrix is

$$M'S = \begin{pmatrix} r & -ps & 0 & 0 \\ r_2 & 0 & -p^2s^2 & 0 \\ r_3 & 0 & 0 & -p^3s^3 \end{pmatrix}.$$

After lattice reduction, we apply the inverse S^{-1} to recover the coefficients. The resultant for the two recovered polynomials is $O(n^{5/4})$.

Koo, Jo and Kwon [54] also gave a method to generate geometric progressions of length $d+2$. Let $d=3$. A five-term geometric progression is given by

$$\mathbf{c} = \left(p^2,\, pr,\, r^2,\, (r^3-n)/p,\, r(r^3-n)/p^2\right),$$

provided that $r(r^3-n)/p^2$ is an integer. To control the size of the geometric progression, we want that $x^3 \equiv n \pmod{p^2}$ is solvable with a root $r \approx p$. At present, it is not known how to find such p's efficiently when n is large.

7.4 Degree-$(d, 1)$ polynomials

We can also use lattice-based techniques to generate polynomial pairs of degree $(d, 1)$. Herrmann, May and Ritzenhofen [42] described a method to generate degree-$(d, 1)$ polynomial pairs.

We construct a $(d+1)$-dimensional square matrix of basis vectors and reduce it using LLL. For example, let $d=5$. We choose $m \approx n^{1/(d+1)} = n^{1/6}$. The row (basis) vectors represent independent polynomials having root m modulo n. We consider a skewed version of the basis matrix:

$$MS = \begin{pmatrix} n & 0 & 0 & 0 & 0 & 0 \\ m & -s & 0 & 0 & 0 & 0 \\ m^2 & 0 & -s^2 & 0 & 0 & 0 \\ m^3 & 0 & 0 & -s^3 & 0 & 0 \\ m^4 & 0 & 0 & 0 & -s^4 & 0 \\ m^5 & 0 & 0 & 0 & 0 & -s^5 \end{pmatrix}.$$

We apply the LLL reduction on MS and then recover a vector by multiplying S^{-1}. The linear polynomial is given by $x - m$. Herrmann, May and Ritzenhofen [42] also suggested to modify the norm used in LLL to capture the L^2-norm for polynomials.

We can further modify the matrix to find non-monic linear polynomials. Let the skewness $s = 1$. The linear polynomial $lx - m$ gives root $r = m/l \pmod{n}$. It is reasonable to consider the following matrix.

$$M'' = \begin{pmatrix} n & 0 & 0 & 0 & 0 & 0 \\ r \pmod{n} & -1 & 0 & 0 & 0 & 0 \\ r^2 \pmod{n} & 0 & -1 & 0 & 0 & 0 \\ r^3 \pmod{n} & 0 & 0 & -1 & 0 & 0 \\ r^4 \pmod{n} & 0 & 0 & 0 & -1 & 0 \\ r^5 \pmod{n} & 0 & 0 & 0 & 0 & -1 \end{pmatrix}.$$

As before, we can apply the skewness matrix and then do the LLL reduction. The determinant of the lattice is the same. However, the resultant might increase due to the linear polynomial. In the end, we can use various optimization techniques from Chapters 5, 6 to optimize the polynomial pair.

Chapter 8

Conclusions and further work

In this chapter, we summarise the main findings of this thesis and suggest some topics for further work.

8.1 Conclusions

The thesis discussed polynomial selection for the number field sieve. We mainly focused on polynomial selection with two polynomials, one of which is a linear non-monic polynomial. Polynomial selection with such polynomials can be divided into three stages: polynomial generation, size optimization and root optimization.

Polynomial selection often starts with generating many polynomials with small leading coefficients. We reviewed two algorithms for generating such polynomials due to Kleinjung [48] [49]. In practice, Kleinjung's second algorithm [49] and its variants perform better.

Size optimization aims to further reduce the size of the raw polynomials by changing skewness, translating and rotating. Traditional local optimization techniques fail for many sextic polynomials when the integers to be factored are large. We described some methods to optimize the size by better determining the translation.

Root optimization aims to produce polynomials that have many roots modulo small primes and prime powers. The traditional root sieve is time-consuming on sextic polynomials for factoring large integers. We discussed some faster algorithms for root optimization. The algorithms are based on Hensel's lemma and a root sieve on congruences classes modulo small prime powers.

The size and root optimization have been tested empirically in CADO-NFS [6] and found to work for sextic polynomials.

8.2 Further work

We suggest the following areas for further research.

The size and root optimization techniques described in this thesis focus on sextic polynomials. Larger integers such as RSA-1024 may use septic polynomials. It is not known whether the methods for sextic polynomials can be modified to cater for septic polynomials.

Polynomial selection with two non-linear polynomials can be beneficial, if we can find such polynomials with small coefficients. At the moment, two cubic polynomials and two non-linear polynomials of degree $(4, 3)$ are the most interesting cases. It is not yet clear how to generate such polynomials efficiently to compete with degree $(d, 1)$ polynomial pairs ($d = 5$ or 6 respectively).

Furthermore, it is not known whether it is necessary or how to optimize the root and size properties for the two non-linear polynomials of similar degree.

Appendix A

Some polynomials

This appendix lists some polynomial pairs discussed in the thesis.

Polynomial A_{768}**:**

$$\begin{aligned}
f(x) = {}& \quad 90901008\,x^6 \\
& + 2258124423259704\,x^5 \\
& + 28187497745151468960 9641\,x^4 \\
& - 18039521792786550991302371778 24\,x^3 \\
& - 35583065959892224664209208391851390580\,x^2 \\
& + 41994172518178515646523609392516698968894816\,x \\
& + 1006065007607534520153196224714870304977201610 7520 \\
g(x) = {}& \quad 9051269666431942968619\,x \\
& - 15437147462287145670372578820920041741
\end{aligned}$$

Polynomial B_{768}:

$$\begin{aligned} f(x) = &\ \ 3586380\,x^6 \\ &+ 19064700x^5 \\ &+ 2823762347055585450848 43x^4 \\ &+ 7186937011302402252746128141881 42x^3 \\ &+ 43402001628933397612599912223809 11282x^2 \\ &- 12541568233611627968693736065307030120x \\ &+ 9008374174467563445936947139641332877 \\ g(x) = &\ \ 5336205483258201922 5383\,x \\ &- 264577222515141490879113842490445208 30 \end{aligned}$$

Polynomial C_{768}:

$$\begin{aligned} f(x) = &\ \ 3586380\,x^6 \\ &- 4117247962908300x^5 \\ &+ 22518332252355341901098 43x^4 \\ &+ 1362209300404696707841386105 16x^3 \\ &- 17501466895317212327777571690641007037x^2 \\ &- 26110703038255899947787642868830402747 6731x \\ &+ 761551516028003977492805501977631103604836365 7612 \\ g(x) = &\ \ 5336205483258201922 5383\,x \\ &- 26457732461661641051994527730477303005 \end{aligned}$$

Bibliography

[1] L. M. Adleman. Factoring numbers using singular integers. In *Proceedings of STOC '91*, pages 64–71. ACM, 1991.

[2] L. M. Adleman and M.-D. A. Huang. Function field sieve method for discrete logarithms over finite fields. *Information and Computation*, 151(1-2):5–16, 1999.

[3] K. Aoki and H. Ueda. Sieving using bucket sort. In *Proceedings of ASIACRYPT '04*, volume 3329 of *Lecture Notes in Computer Science*, pages 92–102. Springer, 2004.

[4] D. Atkins, M. Graff, A. K. Lenstra, and P. C. Leyland. The magic words are Squeamish Ossifrage. In *Proceedings of ASIACRYPT '94*, volume 917 of *Lecture Notes in Computer Science*, pages 263–277. Springer, 1994.

[5] E. Bach and R. Peralta. Asymptotic semismoothness probabilities. *Mathematics of Computation*, 65:1701–1715, 1996.

[6] S. Bai, P. Gaudry, A. Kruppa, F. Morain, L. Muller, E. Thomé, and P. Zimmermann. CADO-NFS, an implementation of the number field sieve. `http://cado-nfs.gforge.inria.fr`, 2011.

[7] D. J. Bernstein, P. Birkner, T. Lange, and C. Peters. ECM using Edwards curves. `http://cr.yp.to/papers.html#eecm`, 2010.

[8] H. Boender. *Factoring large integers with the quadratic sieve.* PhD thesis, Leiden University, 1997.

[9] J. W. Bos, M. E. Kaihara, T. Kleinjung, A. K. Lenstra, and P. L. Montgomery. On the security of 1024-bit RSA and 160-bit elliptic curve cryptography. Cryptology ePrint Archive, Report 2009/389, `http://eprint.iacr.org/2009/389`, 2009.

[10] J. W. Bos, T. Kleinjung, A. K. Lenstra, and P. Montgomery. A 73-digit prime factor by ECM. `http://www.loria.fr/~zimmerma/records/p73`, 2010.

[11] R. P. Brent. An improved Monte Carlo factorization algorithm. *BIT Numerical Mathematics*, 20(2):176–184, 1980.

[12] R. P. Brent. Some integer factorization algorithms using elliptic curves. In *Australian Computer Science Communications 8, 149–163*, 1986.

[13] R. P. Brent. Factorization of the tenth and eleventh Fermat numbers. Technical report, TR-CS-96-02, CSL, ANU, 1996.

[14] R. P. Brent. Factorization of the tenth Fermat number. *Mathematics of Computation*, 68(225):429–451, 1999.

[15] R. P. Brent. Some parallel algorithms for integer factorisation. In *Proceedings of the Euro-Par '99*, volume 1685 of *Lecture Notes in Computer Science*, pages 1–22. Springer, 1999.

[16] R. P. Brent. Recent progress and prospects for integer factorisation algorithms. In *Proceedings of COCOON '00*, volume 1858 of *Lecture Notes in Computer Science*, pages 3–22. Springer, 2000.

[17] R. P. Brent and J. M. Pollard. Factorization of the eighth Fermat number. *Mathematics of Computation*, 36:627–630, 1981.

[18] R. Bărbulescu and P. Gaudry. Improvements on the discrete logarithm problem in GF(p). Technical report, CARAMEL, INRIA Nancy, 2011.

[19] J. Buhler, H. Lenstra, and C. Pomerance. Factoring integers with the number field sieve. In Lenstra and Lenstra [57], pages 50–94.

[20] S. Cavallar. *On the number field sieve integer factorisation algorithm*. PhD thesis, Leiden University, 2002.

[21] H. Cohen. *A Course in computational algebraic number theory*, volume 138 of *Graduate Texts in Mathematics*. Springer, 1993.

[22] A. Commeine and I. Semaev. An algorithm to solve the discrete logarithm problem with the number field sieve. In *Proceedings of PKC '06*, volume 3958 of *Lecture Notes in Computer Science*, pages 174–190. Springer, 2006.

[23] D. Coppersmith. Modifications to the number field sieve. *Journal of Cryptology*, 6:169–180, 1993.

[24] D. Coppersmith. Solving linear equations over GF(2): block Lanczos algorithm. *Linear Algebra and its Applications*, 192:33 – 60, 1993.

[25] R. Cosset. Factorization with genus 2 curves. *Mathematics of Computation*, 79(270):1191–1208, 2010.

[26] J.-M. Couveignes. Computing a square root for the number field sieve. In Lenstra and Lenstra [57], pages 95–102.

[27] H. Cramér. *Mathematical Methods of Statistics*. Princeton University Press, 1999.

[28] R. Crandall and C. Pomerance. *Prime numbers: a computational perspective*. Springer, second edition, 2005.

[29] T. Denny, B. Dodson, A. Lenstra, and M. Manasse. On the factorization of RSA-120. In *Proceedings of CRYPTO '93*, volume 773 of *Lecture Notes in Computer Science*, pages 166–174. Springer, 1994.

[30] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[31] B. Dixon and A. Lenstra. Factoring integers using SIMD sieves. In *Proceedings of EUROCRYPT '93*, volume 765 of *Lecture Notes in Computer Science*, pages 28–39. Springer, 1994.

[32] W. Ekkelkamp. Predicting the sieving effort for the number field sieve. In *Proceedings of ANTS-VIII*, volume 5011 of *Lecture Notes in Computer Science*, pages 167–179. Springer, 2008.

[33] T. Elgamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[34] M. Elkenbracht-Huizing. An implementation of the number field sieve. *Experimental Mathematics*, 5:231–253, 1996.

[35] P. Flajolet and A. Odlyzko. Random mapping statistics. In *Proceedings of EUROCRYPT '89*, volume 434 of *Lecture Notes in Computer Science*, pages 329–354. Springer, 1990.

[36] W. Gautschi. Chapter 7. Error function and Fresnel integrals. In M. Abramowitz and I. A. Stegun, editors, *Handbook of Mathematical Functions: with Formulas, Graphs, and Mathematical Tables*, page 1046. Dover Publications, 1972.

[37] D. M. Gordon. Discrete logarithms in GF(p) using the number field sieve. *SIAM Journal on Discrete Mathematics*, 6(1):124–138, 1993.

[38] J. E. Gower. Rotations and translations of number field sieve polynomials. In *Proceedings of ASIACRYPT '03*, volume 2894 of *Lecture Notes in Computer Science*, pages 302–310. Springer, 2003.

[39] A. Granville. Smooth numbers: computational number theory and beyond. In *Proc. MSRI Conf. Algorithmic Number Theory: Lattices, Number Fields, Curves and Cryptography.* MSRI Publications, Volume 44, 2008.

[40] B. Harris. Probability distribution related to random mappings. *The Annals of Mathematical Statistics*, 31:1045–1062, 1960.

[41] M. Hellman. An overview of public key cryptography. *IEEE Communications Magazine*, 40(5):42 –49, 2002.

[42] M. Herrmann, A. May, and M. Ritzenhofen. Polynomial selection using lattices. In *Workshop on Factoring Large Integers*, Ruhr-University Bochum, 2009.

[43] A. Hildebrand and G. Tenenbaum. Integers without large prime factors. *Journal de Théorie des Nombres de Bordeaux*, 5(2):411–484, 1993.

[44] A. Joux and R. Lercier. The function field sieve is quite special. In *Proceedings of ANTS-V*, volume 2369 of *Lecture Notes in Computer Science*, pages 431–445. Springer, 2002.

[45] A. Joux and R. Lercier. Improvements to the general number field sieve for discrete logarithms in prime fields. A comparison with the Gaussian integer method. *Mathematics of Computation*, 72(242):953–967, 2003.

[46] A. Joux and R. Lercier. The function field sieve in the medium prime case. In *Proceedings of EUROCRYPT '06*, volume 4004 of *Lecture Notes in Computer Science*, pages 254–270. Springer, 2006.

[47] A. Joux, R. Lercier, N. P. Smart, and F. Vercauteren. The number field sieve in the medium prime case. In *Proceedings of CRYPTO '06*, volume 4117 of *Lecture Notes in Computer Science*, pages 326–344. Springer, 2006.

[48] T. Kleinjung. On polynomial selection for the general number field sieve. *Mathematics of Computation*, 75(256):2037–2047, 2006.

[49] T. Kleinjung. Polynomial selection. In *CADO workshop on integer factorization*, INRIA Nancy, 2008. `http://cado.gforge.inria.fr/workshop/slides/kleinjung.pdf`.

[50] T. Kleinjung, K. Aoki, J. Franke, A. K. Lenstra, E. Thomé, J. W. Bos, P. Gaudry, A. Kruppa, P. L. Montgomery, D. A. Osvik, H. J. J. te Riele, A. Timofeev, and P. Zimmermann. Factorization of a 768-bit RSA modulus. In *Proceedings of CRYPTO '10*, volume 6223 of *Lecture Notes in Computer Science*, pages 333–350. Springer, 2010.

[51] T. Kleinjung and J. Franke. Continued fractions and lattice sieving. In *Proceedings of SHARCS 2005*, 2005. `http://www.ruhr-uni-bochum.de/itsc/tanja/SHARCS/talks/FrankeKleinjung.pdf`.

[52] D. E. Knuth. *The Art of Computer Programming*, volume 2. Addison-Wesley, Reading, Mass., 3rd edition, 1997.

[53] N. Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48:203–209, 1987.

[54] N. Koo, G. H. Jo, and S. Kwon. On nonlinear polynomial selection and geometric progression (mod N) for number field sieve. Cryptology ePrint Archive, Report 2011/292, `http://eprint.iacr.org/2011/292`, 2011.

[55] B. A. LaMacchia and A. M. Odlyzko. Solving large sparse linear systems over finite fields. In *Proceedings of CRYPTO '90*, volume 537 of *Lecture Notes in Computer Science*, pages 109–133. Springer, 1990.

[56] A. K. Lenstra. Integer factoring. *Designs, Codes and Cryptography*, 19, 2000.

[57] A. K. Lenstra and H. W. Lenstra, Jr., editors. *The Development of the Number Field Sieve*, volume 1554 of *Lecture Notes in Mathematics*. Springer, 1993.

[58] A. K. Lenstra, H. W. Lenstra Jr., M. S. Manasse, and J. M. Pollard. The factorization of the ninth Fermat number. *Mathematics of Computation*, 61(203):319–349, 1993.

[59] H. W. Lenstra, Jr. Factoring integers with elliptic curves. *Annals of Mathematics*, 126:649–673, 1987.

[60] D. V. Matyukhin. On the asymptotic complexity of computing discrete logarithms in the field GF(p). *Diskretnaya Matematika*, 15(1):28–49, 2003.

[61] R. C. Merkle. Secure communications over insecure channels. *Communications of the ACM*, 21:294–299, 1978.

[62] V. S. Miller. Use of elliptic curves in cryptography. In *Proceedings of CRYPTO '85*, volume 218 of *Lecture Notes in Computer Science*, pages 417–426. Springer, 1986.

[63] P. L. Montgomery. Small geometric progressions modulo N. Unpublished note, December 1993, revised 1995 and 2005.

[64] P. L. Montgomery. Speeding the Pollard and elliptic curve methods of factorization. *Mathematics of Computation*, 48(177):243–264, 1987.

[65] P. L. Montgomery. *An FFT extension of the elliptic curve method of factorization.* PhD thesis, University of California at Los Angeles, 1992.

[66] P. L. Montgomery. Square roots of products of algebraic numbers. In *Proceedings of Symposia in Applied Mathematics, Mathematics of Computation, 1943-1993: a Half-Century of Computational Mathematics*, volume 48, pages 567–571, 1994.

[67] P. L. Montgomery. A survey of modern integer factorization algorithms. *CWI Quarterly*, 7:337–366, 1994.

[68] P. L. Montgomery. A block Lanczos algorithm for finding dependencies over GF(2). In *Proceedings of EUROCRYPT '95*, volume 921 of *Lecture Notes in Computer Science*, pages 106–120. Springer, 1995.

[69] P. L. Montgomery and A. Kruppa. Improved stage 2 to $p \pm 1$ factoring algorithms. In *Proceedings of ANTS-VIII*, volume 5011 of *Lecture Notes in Computer Science*, pages 180–195. Springer, 2008.

[70] P. L. Montgomery and R. D. Silverman. An FFT extension to the $p-1$ algorithm. *Mathematics of Computation*, 54(190):839–854, Apr. 1990.

[71] M. A. Morrison and J. Brillhart. A method of factoring and the factorization of F_7. *Mathematics of Computation*, 29:183–205, 1975.

[72] B. A. Murphy. Modelling the Yield of Number Field Sieve Polynomials. In *Algorithmic Number Theory - ANTS III, LNCS 1443*, pages 137–147, 1998.

[73] B. A. Murphy. *Polynomial selection for the number field sieve integer factorisation algorithm*. PhD thesis, The Australian National University, 1999.

[74] B. A. Murphy and R. P. Brent. On quadratic polynomials for the number field sieve. In *Proceedings of the CATS '98*, volume 20 of *Australian Computer Science Communications*, pages 199–213. Springer, 1998.

[75] J. Neukirch. *Algebraic number theory*, volume 322 of *Fundamental Principles of Mathematical Sciences*. Springer, 1999.

[76] P. Q. Nguyen. A Montgomery-like square root for the number field sieve. In *Proceedings of ANTS-III*, volume 1423 of *Lecture Notes in Computer Science*, pages 151–168. Springer, 1998.

[77] J. Papadopoulos. Call for volunteers: RSA768 polynomial selection, 2011. `http://www.mersenneforum.org/showthread.php?t=15540`.

[78] J. Papadopoulos. Msieve, 2011. `http://sourceforge.net/projects/msieve/`.

[79] M. Petzold. A note on the first moment of extreme order statistics from the normal distribution. Working paper, Department of Statistics, Göteborg University, 2000.

[80] J. M. Pollard. Theorems on factorization and primality testing. *Proceedings Cambridge Philosophical Society*, 76:521–528, 1974.

[81] J. M. Pollard. A Monte Carlo method for factorization. *BIT Numerical Mathematics*, 15:331–334, 1975.

[82] J. M. Pollard. Monte Carlo methods for index computation mod p. *Mathematics of Computation*, 32:918–924, 1978.

[83] J. M. Pollard. Factoring with cubic integers. In Lenstra and Lenstra [57], pages 4–10.

[84] J. M. Pollard. The lattice sieve. In Lenstra and Lenstra [57], pages 43–49.

[85] C. Pomerance. Analysis and comparison of some integer factoring algorithms. In *Computational Methods in Number Theory*, pages 89–139. Math. Centrum Tract 154, Amsterdam, 1982.

[86] C. Pomerance. A tale of two sieves. *Notices of the American Mathematical Society*, 43:1473–1485, 1996.

[87] T. Prest and P. Zimmermann. Non-linear polynomial selection for the number field sieve. Technical report, CARAMEL, INRIA Nancy, 2010.

[88] R. L. Rivest. Cryptography. In *Handbook of Theoretical Computer Science (Volume A: Algorithms and Complexity)*, chapter 13, pages 717–755. Elsevier and MIT Press, 1990.

[89] R. L. Rivest, A. Shamir, and L. M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[90] J. P. Royston. Algorithm AS 177: Expected normal order statistics (exact and approximate). *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 31(2):pp. 161–165, 1982.

[91] O. Schirokauer. Discrete logarithms and local units. *Philosophical Transactions: Physical Sciences and Engineering*, 345(1676):pp. 409–423, 1993.

[92] J. H. Silverman. *The Arithmetic of Elliptic Curves*, volume 106 of *Graduate Texts in Mathematics*. Springer, 1986.

[93] R. D. Silverman. The multiple polynomial quadratic sieve. *Mathematics of Computation*, 48:329–339, 1987.

[94] R. D. Silverman and S. S. Wagstaff. A practical analysis of the elliptic curve factoring algorithm. *Mathematics of Computation*, 61(203):445–462, 1993.

[95] C. Stahlke and T. Kleinjung. Ideas for finding better polynomials to use in GNFS. In *Workshop on Factoring Large Numbers, Discrete Logarithmes and Cryptanalytical Hardware*, Institut für Experimentelle Mathematik, Universität Duisburg-Essen, 2008.

[96] E. Teske. On random walks for Pollard's rho method. *Mathematics of Computation*, 70(234):809–825, 2001.

[97] P. C. van Oorschot and M. J. Wiener. Parallel collision search with cryptanalytic applications. *Journal of Cryptology*, 12(1):1–28, 1999.

[98] R. S. Williams. Cubic polynomials in the number field sieve. Master's thesis, Texas Tech University, 2010.

[99] P. Zimmermann and B. Dodson. 20 years of ECM. In *Proceedings of ANTS-VII*, volume 4076 of *Lecture Notes in Computer Science*, pages 525–542. Springer, 2006.