Analysis of the coefficients of the polynomials produced in the polynomial selection phase of the Number Field Sieve

Gaspard Charvy
ENS Rennes, Bruz, France
firstname.lastname@ens-rennes.fr

Emmanuel Thomé
Université de Lorraine, CNRS, Inria, LORIA, Nancy, France
firstname.lastname@inria.fr

***Abstract*—In this report, we present an analysis of the coefficients of the polynomials produced in the polynomial selection phase of the Number Field Sieve algorithm. Our work focused especially on the collisions used to produce the polynomials, as well as on the size of the coefficients of resulting polynomials.**

***Index Terms*—RSA, NFS, polynomial selection, cado-nfs**

I. Introduction

The RSA cryptosystem relies on the difficulty to factor a product of two big primes, and the size of the keys must change to adapt to current and future progress on this problem. Since 2020, the state of the art has been the factorization of a 250 digits RSA key [1], using the current best known algorithm for large integer factorization: the Number Field Sieve.

However, the computational resources used at an academic level in this record, although very significant, are far inferior to those that a state adversary could mobilize. As a consequence, in order to figure out the real difficulty of integer factorisation, one area of work is to simulate as accurately as possible the Number Field Sieve to have a precise estimation of time needed to factor integers.

The first step to model is the polynomial selection (detailed in Section III), as the quality of the polynomial pair used in the algorithm greatly impacts the time spent in the next steps.

One way to study the produced polynomials is to consider the whole polynomial selection as a black box, and look at the output. This has already been done in [2], focusing especially on the logarithmic L^2-norm, and the murphy-E score, two indicators of the quality of the produced polynomials.

During this internship, we used a different approach, consisting in studying every step of the algorithm one by one, in order to figure out what is happening. Our main contributions are the prediction of the distribution of values we found in collisions (used to produce polynomials), as well as an analysis of the coefficients of the polynomials. This work also led to the discovery of a bug in cado-nfs [3].

In Section II, we will briefly introduce the Number Field Sieve. Then, we will present the method currently used to find polynomial pairs in Section III. Sections IV and V present our estimates of collected collisions. Finally, Section VI analyzes the size of the coefficients of produced polynomials.

II. The Number Field Sieve

The Number Field Sieve (NFS) is an algorithmic framework, that can be used to solve the problem of Integer Factorization: given N a composite integer, find a non-trivial factorization of N. This algorithm is described in [4].

There are several parts in NFS, as schematized in Figure 1, presented in [1]

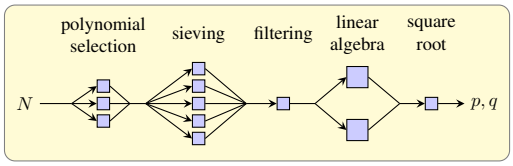


Fig. 1: Main steps of NFS

In the first step, the polynomial selection, we look for two irreducible polynomials f_0 and f_1 in $\mathbb{Z}[x]$, with a common root m modulo N. These polynomials define two algebraic number fields $K_0 = \mathbb{Q}(\alpha_0)$ and $K_1 = \mathbb{Q}(\alpha_1)$, where $f_i(\alpha_i) = 0$. This determines the mathematical setup of NFS, presented in Figure 2 (cf [1]).

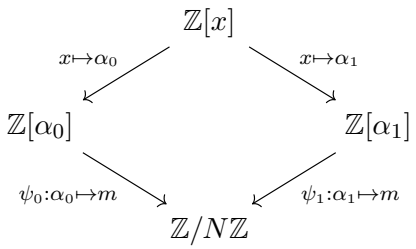


Fig. 2: The mathematical setup for NFS

These polynomials are then used to collect relations of the following form:

We look for coprime pairs $(a, b) \in \mathbb{Z}^2$ such that both values $f_i(\frac{a}{b})b^{\deg f_i}$ $(= \mathrm{Res}(a - bx, f_i))$ for $i = 1, 2$ are smooth with respect to a certain bound B (i.e., can be written as a product of primes lower than B). This allows to find a decompositions of $a - b\alpha_i$ in "small factors" in $\mathbb{Q}[\alpha_i]$, which then gives us two different decomposition of $a - bm$ modulo N (we deliberately omit some number theoretic fine points).

By combining many of these relations, we can construct a congruence of squares: $x^2 = y^2 \mod N$, with $x \neq y$. This gives us the relation $(x + y)(x - y) = 0 \mod N$, meaning that either $(x + y)$ or $(x - y)$ is a zero divisor. Since $N = pq$ with p, q two prime numbers, the only zero divisors modulo N are multiples of p and multiples q. By computing the greatest common divisor (gcd) of N and $(x - y)$ or $(x + y)$, we can find a factor of N, and deduce its factorization (with probability at least $\frac{1}{2}$).

Parameters of this algorithm—e.g., the degree of the polynomials—are chosen according to asymptotic considerations, and enable the following sub-exponential time and space complexity:

$$L_N(1/3, (64/9)^{1/3})^{1+o(1)}$$
$$=$$
$$\exp((64/9)^{1/3}(\log N)^{1/3}(\log\log N)^{2/3}(1 + o(1)))$$

III. Polynomial selection

During this internship, we focused on the polynomial selection process, especially on the implementation in cado-nfs [3], used in the lastest factorization records [1].

There are several methods to find polynomials with a common root modulo N, with coefficients sufficiently small. All competitive methods require one of the polynomials to have degree 1.

A. Base-m method

The textbook approach consists in choosing a degree d for a polynomial, choosing m close to $N^{1/d}$, $f_0 = x - m$, and $f_1 = \sum_{j=0}^{d} a_j x^j$, where $\sum_{j=0}^{d} a_j m^i$ is the base-m expansion of N, i.e., $f_1(m) = N$. This way, we have indeed f_0 and f_1 with a common root m modulo N.

B. Base (ℓ, m) method

Instead of using a monic polynomial $f_0 = x - m$, we can use a non monic polynomial: $f_0 = \ell x - m$. This idea was introduced in 2005 by Kleinjung in [5].

With $f_1(x) = \sum_{j=0}^{d} a_j x^j$, we want $\mathrm{Res}(f_0, f_1) = N$, i.e., $N = \sum_{j=0}^{d} a_j m^j \ell^{d-j} = \ell^d f_1(m/\ell)$.

Kleinjung's lemma 2.1 [5] works as follows (Algorithm 1):

Algorithm 1 Kleinjung's lemma 2.1

Input: N, a_d, m, ℓ such that $N = a_d m^d \mod \ell$
Output: $a_0, \ldots, a_{d-1}$ such that $N = \sum_{j=0}^{d} a_j m^j \ell^{d-j}$

```
r_d ← N
for j = d − 1, ..., 0 do
    r_j ← (r_{j+1} − a_{j+1} m^{j+1})/ℓ
    a_j ← r_j/m^j + δ_j with 0 ≤ δ_j ≤ ℓ, such that r_j = a_j m^j mod ℓ
end for
```

Algorithm 1 takes a_d as a parameter: we must choose a leading coefficient for the polynomial. We can also provide more than one of the leading coefficients, as long as the invariant $N = a_d m^d + a_{d-1} m^{d-1} \ell + \cdots + a_{d-j} m^{d-j} \ell^j \mod \ell^{j+1}$ holds.

Note: In cado-nfs, we choose $-\ell/2 \leq \delta_j \leq \ell/2$ *instead of* $0 \leq \delta_j \leq \ell$.

C. Finding exceptionally good ℓ and m values

To find ℓ and m values used in Kleinjung's lemma 2.1, the method used in cado-nfs is the one presented by Kleinjung at the CADO workshop [6].

1) A useful tool: $\tilde{a_d} = 1, \tilde{a}_{d-1} = 0$*:* Let us first examine how we can force Algorithm 1 to output polynomials with two prescribed leading coefficients. This is only possible for specific inputs, and we use the tilde notation to reflect this particular situation.

In this first situation, we want to find $\tilde{f}_1(x) = x^d + \tilde{a}_{d-2} x^{d-2} + \cdots + \tilde{a}_0, \tilde{f}_0(x) = \ell x - \tilde{m}$, with $\tilde{N} = \mathrm{Res}(\tilde{f}_0, \tilde{f}_1) (= \sum_{j=0}^{d} \tilde{a}_j \tilde{m}^j \ell^{d-j})$.

Write $\tilde{R} = \frac{\tilde{N} - \tilde{m}^d}{\ell^2} = \sum_{j=0}^{d-2} \tilde{a}_j \tilde{m}^j \ell^{d-2-j}$.

Given $(\tilde{N}, d)$, we want to find $(\ell, \tilde{m})$ such that $\tilde{R}$ is an integer, i.e., $\ell^2 | (\tilde{N} - \tilde{m}^d)$, and $\tilde{a}_{d-2}$ small. We first choose $\tilde{m_0} = \lceil \tilde{N}^{1/d} \rceil$. We want to find $\tilde{m}$ close to $\tilde{m_0}$.

Algorithm 2 Case $\tilde{a}_d = 1, \tilde{a}_{d-1} = 0$

Input: $\tilde{N}, \tilde{m_0}, P, d$

Output: $\ell, \tilde{m}$ such that $\tilde{R} \in \mathbb{Z}$ and Algorithm 1 can be used

1: Let $\mathbf{P} = [P, 2P]$ be a range of prime numbers, $M = (2P)^2$
2: List pairs $(p \in \mathbf{P}, i \in [-M, M])$, such that $p^2 | (\tilde{N} - (\tilde{m_0} + i)^d)$
3: Find collisions on i (we need pairs with the same i value to have the same $\tilde{m_0} + i$)
4: Each collision $(p_1, i), (p_2, i)$ yields $\ell = p_1 p_2, \tilde{m} = \tilde{m_0} + i$

Algorithm 2 provides values ℓ and $\tilde{m}$ such that $\ell^2 | (\tilde{N} - \tilde{m}^d)$, and allows us to find f_1 thanks to a small modification of Algorithm 1.

The analysis of the size of coefficients is done in Section VI.

2) More general situation: In the more general situation, we want to find $f_1(x) = a_d x^d + a_{d-1} x^{d-1} + \cdots + a_0, f_0(x) = \ell x - m$, with $N = \mathrm{Res}(f_0, f_1)$.

We want to achieve $N = \ell^d f_1(m/\ell) = a_d m^d + a_{d-1} m^{d-1} \ell + \ell^2 R$, and $|a_{d-2}| \approx \frac{R}{m^{d-2}}$ small.

To find values of ℓ and m, we reduce this problem to the situation in III-C1 with the translation $x \mapsto x - \frac{a_{d-1}}{d a_d}$, and multiplication by $d^d a_d^{d-1}$:

$$d^d a_d^{d-1} N = (d a_d m + a_{d-1} \ell)^d + \ell^2 (d^d a_d^{d-1} R - \dots)$$

We can then define $\tilde{N} = d^d a_d^{d-1} N$, $\tilde{m} = (d a_d m + a_{d-1} \ell)$, and use the tool in Algorithm 2 to find $\ell, \tilde{m}$. With a few operations modulo $d a_d$, we can find an appropriate a_{d-1} value, and then m.

Once again, an analysis of the size of produced coefficients can be found in Section VI.

Once we have found ℓ, m, a_{d-1} such that $\ell^2 | N - a_d m^d - a_{d-1} m^{d-1} \ell$, we apply Algorithm 1 to find the other coefficients.

3) Special-q: We are now focusing on the particular situation from Subsection III-C1, as we are using it to find polynomials.

Finding the roots of $\tilde{N} - (\tilde{m_0} + X)^d$ modulo p^2 for every $p \in \mathbf{P}$ is quite slow. In order to amortize the cost of these computations, we can use a technique to re-use them:

We define $\mathfrak{Q} = \{q_1, q_2, ...\}$ as a set of small primes that we are considering, and index them by increasing order: $j_1 < j_2 \Leftrightarrow q_{j_1} < q_{j_2}$ (in the cado-nfs program, we use primes from 2 to 271).

- Instead of looking for $\ell = p_1 p_2$, we aim for $\ell = p_1 p_2 q$, where q is a product of a few small prime numbers in $\mathfrak{Q}$
- Find roots of $\tilde{N} - (\tilde{m_0} + X)^d$ modulo squares of primes in $\mathfrak{Q}$
- Select a few of these primes, and use the Chinese Remainder Theorem to find a root r_q modulo q^2, where q is the product of the selected primes.

This way, we have $\tilde{N} - (\tilde{m_0} + r_q + i q^2)^d = 0 \mod q^2$ for any $i \in [-M, M]$. We can then proceed as seen in Algorithm 2, except searching for pairs (p, i) where $p^2 | \tilde{N} - (\tilde{m_0} + r_q + i q^2)^d$. This yields $\ell = p_1 p_2 q$, $\tilde{m} = \tilde{m_0} + r_q + i q^2$, with once again $\ell^2 | (\tilde{N} - \tilde{m}^d)$, and we can recover a polynomial pair as in Section III-C2.

Since p^2 and q^2 are coprime (q is a product of small primes, much smaller than p), i' is a root of $\tilde{N} - (\tilde{m_0} + X)^d = 0 \mod p^2$ is equivalent to $(i' - r_q)/q^2$ is a root of $\tilde{N} - (\tilde{m_0} + r_q + X q^2)^d = 0 \mod p^2$, which allows to reuse computation of roots modulo p^2.

IV. Collisions on (i, p) pairs

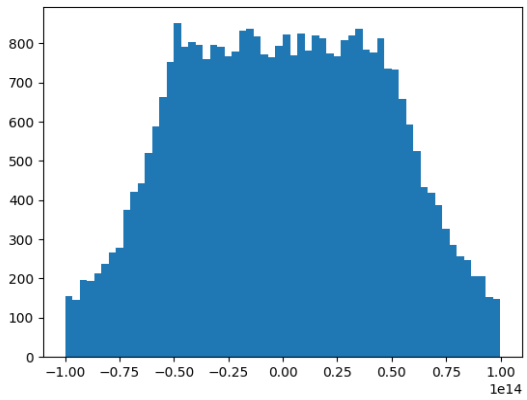
The first step of the polynomial selection is to find collisions on pairs (p, i) such that $\tilde{N} - (\tilde{m_0} + i)^d = 0 \mod p^2$ (or $\tilde{N} - (\tilde{m_0} + r_q + i q^2)^d = 0 \mod p^2$). We studied the distribution of the different elements we obtained from these collisions, starting with i and p values.

The figures presented in this Section were produced using 180-digit integers, with $P = 5 \times 10^6$, and $M = (2P)^2 = 10^{14}$

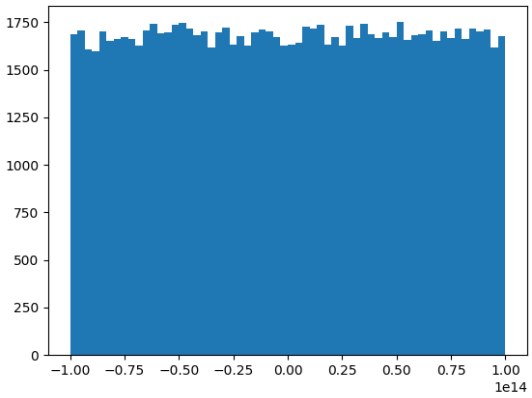
A. i distribution

First, we assumed that the distribution of i values was uniform in $[-M, M]$. However, during testing, the i values we recovered from the collisions were not at all uniform (as shown in Figure 3a). After a thorough examination of the cado-nfs code, it appeared that the function in charge of detecting the collisions during the special-q portion of the polynomial selection was not doing its job properly. The i values inserted in the hash table were about uniform, but not the values appearing in the collisions. When replacing this quite unreadable function with a readable (and correct !) version, we then got a much more satisfactory i distribution, matching with the

hypothesis of a uniform distribution (see Figure 3b). This was reported and fixed in cado-nfs bug #30089.



(a) before removing the ugly function



(b) after removing the ugly function

Fig. 3: Histograms of i values appearing in collisions

B. p distribution

We then want to estimate the probability for a given prime number p to be selected:

Let us first assume that there is exactly one root to the equation $\tilde{N} - (\tilde{m}_0 + X)^d = 0 \mod p^2$ (which, in particular, is the case whenever d is coprime to $p-1$).

If i_0 is a root of this equation, then so is every i congruent to i_0 modulo p^2 (since we are considering an equation modulo p^2). As a consequence, the number of pairs (p, i) found is the number of i values congruent to i_0 modulo p^2 that are in the interval $[-M, M]$, so there are about $\frac{2M}{p^2}$ pairs (p, i). What's interesting to us so far is that it is roughly proportional to $\frac{1}{p^2}$.

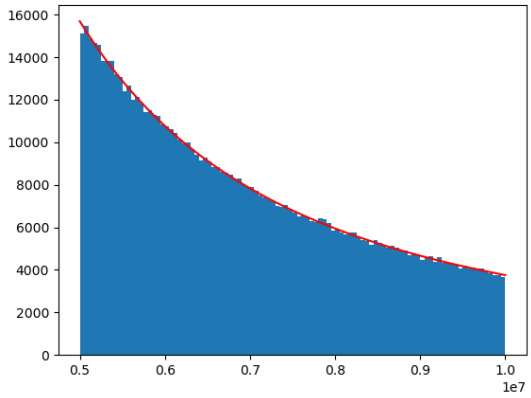


Fig. 4: Histogram of p values appearing in collisions, and plot of $f_P(x)$

In order to get the probability density function of the p values, we have to multiply this probability with the density of prime numbers around that value. We can therefore model the p distribution by the following probability density function

$$f_P(x) = \frac{1}{x^2 \log(x)} \times \frac{1}{C}$$

where $C = \int_P^{2P} \frac{1}{x^2 \log(x)} \mathrm{d}x$ is a normalizing factor.

When we collect the p values obtained in collisions, we indeed find a distribution that is very close to that estimate (see Figure 4).

If we take a closer look, there isn't always exactly 1 root to the equation $\tilde{N} - (\tilde{m}_0 + X)^d = 0 \mod p^2$. If $\gcd(p-1, d) = r$, then this equation either has r roots (with probability $\frac{1}{r}$), or none (see Appendix A, Theorem 2 [1]). But on average, we do have one root per equation, and the equations with no root or more than 1 roots are well spread, which does not impact the overall distribution.

V. q DISTRIBUTION

In this Section, we want to explain values of q that we obtained in collisions, more specifically the factors appearing in values of q.

[1] $\gcd(p, \tilde{N}) = 1$ because p is a prime bigger than d and than factors of a_d, and p cannot be a factor of N, since $p \ll \sqrt{N} \approx$ order of magnitude of factors of N

A. choice of q value

In the cado-nfs program, when the parameter `nq` (which is the number of couples (r_q, q) that we want to try) is not zero, we proceed as presented in Algorithm 3:

Algorithm 3 Choice of q values

Input: $\tilde{N}, \tilde{m_0}, \mathfrak{Q}$

Output: a subset of $\mathfrak{Q}$ from which we will form q values

1: Compute the roots of $\tilde{N} - (\tilde{m_0} + X)^d$ modulo q_j^2, for every $q_j \in \mathfrak{Q}$, skipping those dividing d or a_d
2: Sort the q_j values by decreasing number of roots
3: Fix $k = \lceil \log_d(\mathtt{nq}) \rceil$
4: Select the first `lq` values of q_j, such that we can form at least `nq` combinations (r_q, q), where q is the product of k primes among the `lq` selected, and r_q is a root of $\tilde{N} - (\tilde{m_0} + X)^d$ modulo q^2

In Algorithm 3, values of a_d tested in the polynomial selection algorithm are multiples of an increment $incr$, which is a parameter of the polynomial selection. As a consequence, prime factors dividing $d \times incr$ are always skipped. We can thus exclude them from $\mathfrak{Q}$.

The variable `lq` is the smallest value such that we can form at least `nq` combinations. It depends on the number of roots of the primes selected.

B. distribution of q factors

As in subsection IV-B (once again, Appendix A, Theorem 2) [2], with $r = \gcd(d, p-1)$, $\tilde{N} - (\tilde{m_0} + X)^d \mod q_j^2$ either has r roots with probability $\frac{1}{r}$, or none otherwise.

We studied the cases $d = 5$ and $d = 6$ ($d = 6$ was used to factor rsa-240 and rsa-250 [1]). With $d = 5$ (or any other prime), r can only be equal to 1 or d. With $d = 6$, r can be either 6 or 2 (for primes bigger than 2, $p-1$ is always even). We define r_B as the other possible value of r (case d prime: $r_B = 1$, case $d = 6$: $r_B = 2$.) The case $d = 7$ and other prime numbers should follow the same laws as $d = 5$.

It is useful here to divide $\mathfrak{Q}$ according to the number of possible roots, since we sort primes by number of roots. We will define $\mathfrak{Q}_1 = \{q_j \in \mathfrak{Q} \mid q_j = 1 \mod d\}$, and $\mathfrak{Q}_* = \{q_j \in \mathfrak{Q} \mid q_j \neq 1 \mod d\}$.

We make the assumption that the equation having r roots and q_j dividing a_d are independent.

[2]We have d and a_d coprime with q_j because we skipped q_j values that weren't suitable.

We define the following events:

- L_j: event "q_j is selected among the `lq`"
- Q_j: event "q_j appears as a factor of q".

$Q_j \subseteq L_j$, because we form combinations with selected primes.

For $q_j \in \mathfrak{Q}_1$, let A_j be a random variable that equals 1 if there are d roots to the equation $\tilde{N} - (\tilde{m_0} + X)^d \mod q_j^2$, and q_j is not skipped (i.e., q_j does not divide d nor a_d), 0 otherwise. A_j follows a Bernoulli law of parameter $\frac{1}{d} \times (1 - \frac{1}{q_j})$

Let A be the random variable corresponding to the number of primes for which there are d roots, and that are not skipped. We have: $A = \sum_j A_j$

For $q_j \in \mathfrak{Q}_*$, with $r_B = \gcd(p-1, q_j)$, let B_j be a random variable that equals 1 if q_j is not skipped and there are r_B roots to the equation $\tilde{N} - (\tilde{m_0} + X)^d \mod q_j^2$, 0 otherwise. B_j follows a Bernoulli law of parameter $\frac{1}{r_B} \times (1 - \frac{1}{q_j})$.

Due to the sort (step 2 in Algorithm 3), the primes where we get d roots are the first selected.

If $A \geq k$, i.e., there are at least k primes where we get d roots, then $\mathtt{lq} = k$ is sufficient (we can choose between d roots for every prime, so we have $d^k \geq \mathtt{nq}$ by choice of k).

Otherwise, the number of combinations (r_q, q) that we can make by choosing k primes among `lq`, with a primes giving d roots (and $\mathtt{lq} - a$ primes giving r_B roots) is:

$$\text{nb\_comb}(d, k, \mathtt{lq}, a) = \sum_{u=0}^{a} \binom{a}{u} \binom{\mathtt{lq} - a}{k - u} d^u r_B^{k-u}$$

where u corresponds to how many primes giving d roots are factors of q.

Note: the number of combinations involving a factor q_j giving d roots is $d\times$ nb_comb $(d, k-1, \mathtt{lq}-1, a-1)$*, and the number of combinations involving a factor q_j giving r_B roots is $r_B\times$* nb_comb $(d, k-1, \mathtt{lq}-1, a)$

Using the law of total probability, for $q_j \in \mathfrak{Q}$:

$$\mathbb{P}(Q_j) = \sum_a \mathbb{P}(Q_j \cap (A = a))$$

Since $Q_j \subseteq L_j$, $Q_j = Q_j \cap L_j$, and for any a,

$$\begin{aligned}\mathbb{P}(Q_j \cap (A = a)) &= \mathbb{P}(Q_j \cap L_j \cap (A = a)) \\ &= \mathbb{P}(Q_j | L_j \cap (A = a)) \times \mathbb{P}(L_j \cap (A = a))\end{aligned}$$

$\mathbb{P}(Q_j | L_j \cap (A = a))$ is the ratio of combinations that we can form using q_j over the total of combinations that

we can form, knowing that there are a primes with d roots, and that q_j is selected.

For more detailed calculations and the Sagemath [7] code used to compute these probabilities, see appendix B (page 9).

C. Experimental results

When comparing the theory to the factors of q obtained in collisions, we observed a significant difference when it comes to $q_j \in \mathfrak{Q}_*$.

This was caused by the sorting algorithm used: a poorly implemented insertion sort. Although it did indeed sort by number of roots, it disrupted the sorting from smallest to biggest that we expected and based our analysis upon. For instance, Figure 5a is what we obtained with 180-digit integers, $d = 5$ and $\texttt{nq} = 10000$. We can see that the model is quite far from experimental results. In particular, the peak for primes $q_j \in \mathfrak{Q}_*$ is not located around the same value. The explanation is that the insertion sort was swapping primes where we have d roots with the first values with r_B root, often causing them not to be selected.

After removing the insertion sort, we obtained much more satisfactory results (see Figure 5b). With d = 6, there seems to be something that we didn't take into account in our model, as we can see a small gap between experimental and predicted distributions of q factors in Figure 6.

VI. Coefficients of the produced polynomials

A. a_d, a_{d-1}

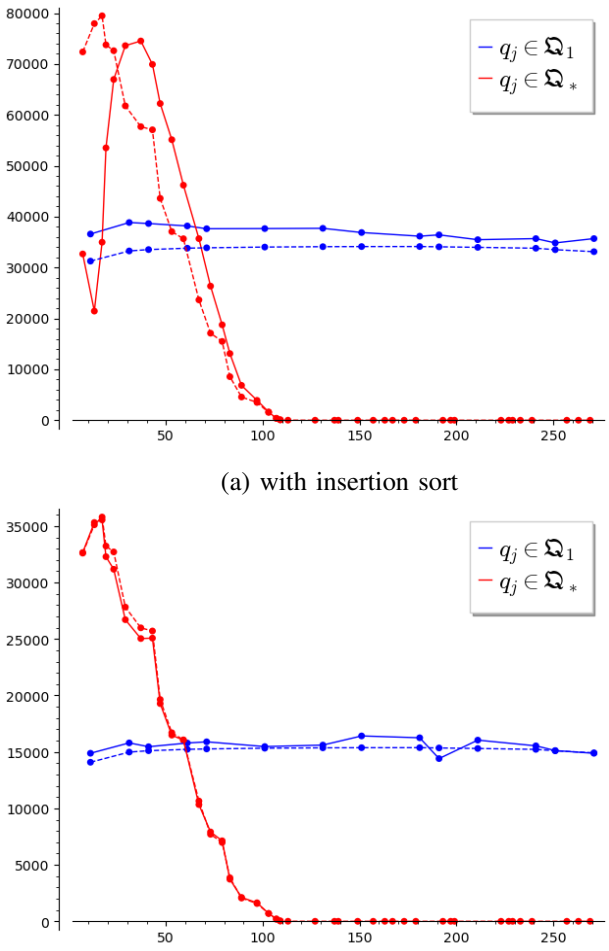
The leading coefficient a_d is a parameter of the algorithm: we can decide its value.

To recover a_{d-1} and m from $\tilde{m} = da_d m + a_{d-1}\ell$, we compute $\tilde{m} \mod da_d$, then multiply by ℓ^{-1} modulo da_d to find a_{d-1}. If $a_{d-1} > \frac{da_d}{2}$, we subtract da_d to its value, in order to have $-\frac{da_d}{2} \leq a_{d-1} \leq \frac{da_d}{2}$.

B. a_{d-2}

In this Section, we will do the analysis in the case $\ell = p_1 p_2 q$. The analysis remains the same if $\ell = p_1 p_2$ (we just apply $q = 1$). These approximations are almost those presented by Kleinjung in [6], except that we didn't use an upper bound for $|i|$.

This analysis is based on the assumption $\ell \leq \ell q \ll m \approx m_0$ (and also $\ell \leq \ell q \ll \tilde{m} \approx \tilde{m}_0$). First, since r_q is a root of $\tilde{N} - (\tilde{m}_0 + x)^d$ modulo q^2, we have $0 \leq r_q \leq q^2$, and $iq^2 \leq r_q + iq^2 \leq (i+1)q^2$. We can thus approximate $r_q + iq^2$ to iq^2.



(a) with insertion sort

(b) with a sort preserving the order of primes

Fig. 5: Number of apparitions of factors of q predicted (dashed line) and found in collisions (full line), $d = 5, \texttt{nq} = 10000$

1) Case $a_d = 1, a_{d-1} = 0$: In this first situation, we have $|\tilde{a}_{d-2}| = \left|\frac{\tilde{R}}{\tilde{m}^{d-2}} + \mathcal{O}(\ell)\right|$. The $\mathcal{O}(\ell)$ term comes from δ_{d-2} in Algorithm 1

Since we have $\tilde{N} \approx {\tilde{m}_0}^d$,

$$\begin{aligned}\tilde{N} - (\tilde{m}_0 + r_q + iq^2)^d &\approx {\tilde{m}_0}^d - (\tilde{m}_0 + r_q + iq^2)^d \\ &\approx -id\tilde{m}_0^{d-1} q^2\end{aligned}$$

(the other terms when we develop $(\tilde{m}_0 + r_q + iq^2)^d$ are negligible, because $|iq^2| < (2P)^2 q^2 \approx \ell q \ll \tilde{m}_0$)

This leads to the approximation

$$\begin{aligned}\left|\frac{\tilde{R}}{\tilde{m}^{d-2}}\right| \approx \left|\frac{id\tilde{m}_0 q^2}{\ell^2}\right| &\approx \left|\frac{id\tilde{m}_0 q^2}{p_1^2 p_2^2 q^2}\right| \\ &\approx \left|\frac{id\tilde{m}_0}{p_1^2 p_2^2}\right|\end{aligned}$$

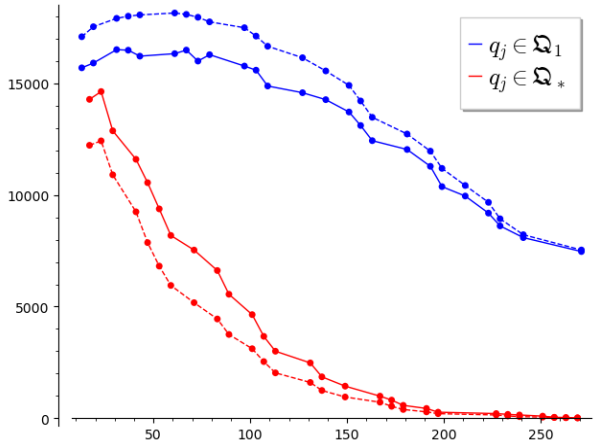


Fig. 6: Number of apparitions of factors of q predicted (dashed line) and found in collisions (full line), with rsa240 ($d = 6, \texttt{nq} = 1296$)

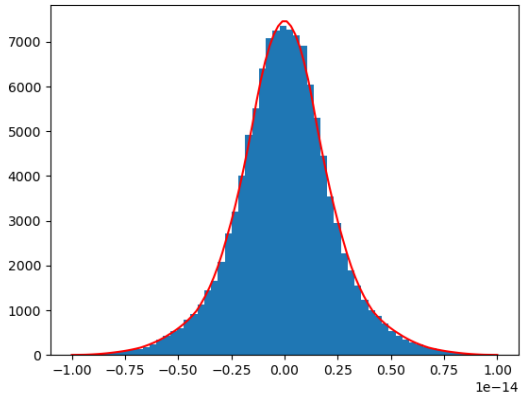


Fig. 7: Histogram of $\frac{a_{d-2}}{m_0}$ values, and estimation of the probability density function of $-\frac{i}{p_1^2 p_2^2} + \frac{\delta_{d-2}}{m_0}$

2) General case: Recall that we have $d^d a_d^{d-1} N = (da_d m + a_{d-1}\ell)^d + \ell^2(d^d a_d^{d-1} R - \dots)$,
with $\tilde{m} = da_d m + a_{d-1}\ell \approx da_d m$,

$$\begin{aligned}\tilde{R} &= d^d a_d^{d-1} R - \sum_{j=2}^{d} \binom{j}{d} (da_d m)^{d-j} a_{d-1}^j \ell^{j-2} \\ &\approx d^d a_d^{d-1} R\end{aligned}$$

We can then write

$$\begin{aligned}\left|\frac{R}{m^{d-2}}\right| &\approx \left|\frac{\tilde{R}}{d^d a_d^{d-1}} \times \frac{(da_d)^{d-2}}{\tilde{m}^{d-2}}\right| \\ &\approx \left|\frac{id\tilde{m}_0}{d^2 a_d p_1^2 p_2^2}\right| \approx \left|\frac{im_0}{p_1^2 p_2^2}\right|\end{aligned}$$

This approximation is very good in practice.

Moreover, δ_{d-2} seems to follow a uniform distribution in $[-\ell/2, \ell/2]$.

To simulate the effect that δ_{d-2} has on the size of the coefficients, we supposed that δ_{d-2} was uniform between $-\mathbb{E}(\ell)/2$ and $\mathbb{E}(\ell)/2$, where $\mathbb{E}(\ell)$ is the expected value of ℓ, and we plotted an estimation of the probability density function of $\frac{i}{p_1^2 p_2^2} + \frac{\delta_{d-2}}{m_0}$ over an histogram of the values of $\frac{a_{d-2}}{m_0}$. As we can see in Figure 7, it looks like this simple estimation can predict fairly accurately the distribution of $\frac{a_{d-2}}{m_0}$ values.

C. other coefficients

In Algorithm 1, for $1 \leq j \leq d-2$, we have: $a_j = \frac{r_j}{m^j} + \delta_j, r_{j-1} = \frac{r_j - a_j m^j}{\ell} = -\frac{\delta_j m^j}{\ell}$. As a consequence, if δ_j values are uniform in $[-\ell/2, \ell/2]$, r_{j-1} is uniform in $[-m^j/2, m^j/2]$.

With the hypothesis that every δ_j is uniform in $[-\ell/2, \ell/2]$ for $j = 1, ..., d-2$, the coefficients $a_0, ..., a_{d-3}$ are all the sum of a uniform law in $[-\ell/2, \ell/2]$ and a uniform law in $[-m/2, m/2]$, which is almost a uniform law in $[-m/2, m/2]$ (because $\ell \ll m$).

D. Importance of the $\mathcal{O}(\ell)$

Since a $\mathcal{O}(\ell)$ appears in every coefficient a_j, with $j \leq d-2$ (the δ_j value we add to ensure divisibility by ℓ), we have to be careful about ℓ not being too big. In case we have $\ell \gg \left|\frac{im_0}{p_1^2 p_2^2}\right|$, then the $\mathcal{O}(\ell)$ completely hides the expected value.

In cado-nfs, there isn't anything preventing this from happening. For instance, when we modified the nq parameter from 1000 (recommended parameter for 180-digit integers) to 10000, about 80% of polynomials had an a_{d-2} at least 10 times bigger than expected.

Assuming that q is the product of k independent primes, selected according to the distribution we calculated in Section V-B, we can estimate the expected value of q as the expected value of a q factor (we will name it q_0), raised to the power of k:

$$\begin{aligned}\mathbb{E}(q) &\approx q_0^k \\ &\approx \left(\frac{\sum_{q_j \in \mathfrak{Q}} q_j \mathbb{P}(Q_j)}{k}\right)^k \end{aligned} \quad (1)$$

The expected ℓ value grows a lot faster than nq: expected q value is close to q_0^k, where k is $\lceil \log_d(\texttt{nq}) \rceil$. q_0 is a lot bigger than d (the biggest d used for now is 6, and the parameters used for the factorization of rsa-240 produces $q_0 \approx 100$). Moreover, as k increases, so does q_0.

This rough estimation in Equation (1) could be used to have an idea of the impact that the $\mathcal{O}(\ell)$ has on the size of coefficients.

VII. Conclusion

In this report, we presented a detailed analysis of the coefficients of polynomials produced by the polynomial selection in cado-nfs [3]. Along the way, we also modeled the distribution of values found in collisions, and discovered a bug in cado-nfs. This is a first important step in order to simulate the Number Field Sieve algorithm. However, there is still a lot of work to do about the polynomial selection: using these estimations of the coefficients of produced polynomials, we want to determine the quality of these polynomials, using indicators like the logarithmic L^2-norm, and the murphy-E score. The analysis of phases of size and root optimization in the polynomial selection (that transform a raw polynomial pair in order to increase sieving properties of the polynomial) could also bring a precise understanding of the quality of the polynomials that the Number Field Sieve is using, and thus a good estimation of the time required for integer factorization.

References

[1] F. Boudot, P. Gaudry, A. Guillevic, N. Heninger, E. Thomé, and P. Zimmermann, "Comparing the difficulty of factorization and discrete logarithm: A 240-digit experiment," in *CRYPTO 2020, Part II*, D. Micciancio and T. Ristenpart, Eds., ser. Lecture Notes in Comput. Sci. Vol. 12171, Springer, Aug. 2020, pp. 62–91. DOI: 10.1007/978-3-030-56880-1_3.

[2] S. Bai, C. Bouvier, A. Kruppa, and P. Zimmermann, "Better polynomials for GNFS," *Math. Comput.*, vol. 85, no. 298, pp. 861–873, 2016. DOI: 10.1090/mcom3048.

[3] The CADO-NFS Development Team, *CADO-NFS, an implementation of the number field sieve algorithm*, development version, 2024.

[4] A. K. Lenstra and H. W. Lenstra, Jr., Eds., *The development of the number field sieve*, vol. 1554, Lecture Notes in Math. Springer, 1993. DOI: 10.1007/BFb0091534.

[5] T. Kleinjung, "On polynomial selection for the general number field sieve," *Math. Comput.*, vol. 75, no. 256, pp. 2037–2047, 2006. DOI: 10.1090/S0025-5718-06-01870-9.

[6] T. Kleinjung, *Polynomial selection*, Slides presented at the CADO workshop on integer factorization, 2008.

[7] The Sage Developers, *Sagemath, the Sage Mathematics Software System (Version 9.5)*, `https://www.sagemath.org`, 2024.

APPENDIX A
NUMBER OF ROOTS OF THE EQUATION $X^d = \tilde{N} \mod p^2$

Theorem 1. *If $r|p-1$ with p a prime number, with*

$$\begin{array}{rcl} \psi_r \colon \left((\mathbb{Z}/p\mathbb{Z})^\times, \times\right) & \longrightarrow & \left((\mathbb{Z}/p\mathbb{Z})^\times, \times\right) \\ x & \longmapsto & x^r \end{array}$$

then $\mathrm{card}(\mathrm{Ker}(\psi_r)) = r$

Proof. We define $s = \frac{p-1}{r}$. There are at most r roots to the equation $X^r = 1 \mod p$ (and at most s roots to the equation $X^s = 1 \mod p$), i.e., $\mathrm{card}(\mathrm{Ker}(\psi_r)) \leq r$, and $\mathrm{card}(\mathrm{Ker}(\psi_s)) \leq s$.

However, for any x in $(\mathbb{Z}/p\mathbb{Z})^\times$, $(x^s)^r = x^{sr} = x^{p-1} = 1$ (using Lagrange's theorem). This means that $\psi_s\left((\mathbb{Z}/p\mathbb{Z})^\times\right) \subset \mathrm{Ker}(\psi_r)$. Since $\mathrm{card}(\mathrm{Ker}(\psi_s)) \leq s$, $\mathrm{card}\left(\psi_s\left((\mathbb{Z}/p\mathbb{Z})^\times\right)\right) \geq \frac{p-1}{s} = r$ using the first isomorphism theorem.

By inclusion, this gives $\mathrm{card}(\mathrm{Ker}(\psi_r)) \geq r$, which concludes this proof.

□

Theorem 2. *If $\gcd(\tilde{N}, p) = 1$, i.e., $\tilde{N} \in (\mathbb{Z}/p\mathbb{Z})^\times$, the equation $X^d = \tilde{N} \mod p^2$ has either r roots with probability $\frac{1}{r}$ or none with probability $\frac{r-1}{r}$, where $r = \gcd(d, p-1)$ and p is a prime number*

Proof. We are looking for roots of $X^d = \tilde{N} \mod p$, and then using Hensel's lifting lemma to obtain roots of $X^d = \tilde{N} \mod p^2$.
If we have a root of $X^d = \tilde{N} \mod p$, we can multiply it by a root of $X^d = 1 \mod p$ to construct a root of $X^d = \tilde{N} \mod p$

Define $\psi_r : x \mapsto x^d$, which is an endomorphism of $\left((\mathbb{Z}/p\mathbb{Z})^\times, \times\right)$.
With $r = \gcd(d, p-1)$, $d = d'r$, we have:
Since $\gcd(d', p-1) = 1, \exists e$ such that $d'e = 1 \mod p-1$, and we have:

$$\begin{array}{lll} x^d = 1 \mod p \Leftrightarrow x^{de} = 1 \mod p & & \text{because } \gcd(e, p-1) = 1 \\ \Leftrightarrow x^{d'er} = 1 \mod p & & \\ \Leftrightarrow x^r = 1 \mod p & & \text{because } d'e = 1 \mod p-1 \end{array}$$

Using Theorem 1, there are exactly r roots to the equation $X^d = 1 \mod p$. Using the first isomorphism theorem, the image of ψ_r has a cardinal $\frac{p-1}{r}$, which means that if $\tilde{N}$ is part of $\psi_r\left((\mathbb{Z}/p\mathbb{Z})^\times\right)$ (happening with probability $\frac{\frac{p-1}{r}}{p-1} = \frac{1}{r}$), then $X^d = \tilde{N} \mod p^2$ has d roots, and 0 otherwise.

□

APPENDIX B
q FACTORS

We develop the calculations presented in Section V-B (page 5)
Reminder:

- L_j: event "q_j is selected among the `lq`"
- Q_j: event "q_j appears as a factor of q".
- For $q_j \in \mathfrak{Q}_1$, A_j is a random variable that equals 1 if there are d roots to the equation $\tilde{N} - (\tilde{m_0} + r_q + xq^2)^d \mod q_j^2$, and q_j is not skipped, 0 otherwise.
- $A = \sum_j A_j$
- For $q_j \in \mathfrak{Q}_*$, let B_j be a random variable that equals 1 if q_j is not skipped, 0 otherwise ($B_j \sim \mathcal{B}(\mathbb{P}(\overline{S_j}))$)

The A_j are independent, so are the B_j

For $q_j \in \mathfrak{Q}$,

$$\begin{aligned}\mathbb{P}(Q_j) &= \sum_a \mathbb{P}(Q_j \cap (A=a)) \\ &= \sum_a \mathbb{P}(Q_j | L_j \cap (A=a)) \times \mathbb{P}(L_j \cap (A=a))\end{aligned}$$

A.

For $q_j \in \mathfrak{Q}_1$, q_j is selected if and only if $A_j = 1$ and there are at most $k-1$ factors already selected, i.e., $\sum_{j'<j} A_{j'} < k$. This leads to

$$\mathbb{P}\left(L_j \cap (A=a)\right) = \mathbb{P}\left((A_j=1) \cap (\sum_{j'<j} A_{j'} < k) \cap (A=a)\right)$$

If $A \geq k$, then every combination formed contains every selected factor (case $\texttt{lq} = k$), i.e., $\mathbb{P}(Q_j | L_j \cap (A=a)) = 1$ when $a \geq k$:

$$\begin{aligned}\mathbb{P}\left(Q_j \cap (A=a)\right) &= \mathbb{P}\left((A_j=1) \cap (\sum_{j'<j} A_{j'} < k) \cap (\sum_{j'\neq j} A_{j'} = a)\right) \\ &= \mathbb{P}\left((A_j=1)\right) \times \mathbb{P}\left((\sum_{j'<j} A_{j'} < k) \cap (\sum_{j'\neq j} A_{j'} = a)\right)\end{aligned}$$

Otherwise, the conditions $A=a$ and $A_j=1$ ensure that $\sum_{j'<j} A_{j'} < k$, we can thus remove this event when computing probabilities in the case $A < k$:

$$\begin{aligned}\mathbb{P}\left(Q_j \cap (A=a)\right) &= \mathbb{P}(Q_j | L_j \cap (A=a)) \times \mathbb{P}\left((A_j=1) \cap (\sum_{j'!=j} A_{j'} = a)\right) \\ &= \frac{d \times \text{nb\_comb}(d, k-1, \texttt{lq}-1, a-1)}{\text{nb\_comb}(d, k, \texttt{lq}, a)} \times \mathbb{P}\left((A_j=1)\right) \times \mathbb{P}\left((\sum_{j'!=j} A_{j'} = a)\right)\end{aligned}$$

B.

For $q_j \in \mathfrak{Q}_*$, we have:

$$\mathbb{P}\left(L_j \cap (A=a)\right) = \mathbb{P}\left(L_j | (A=a)\right) \times \mathbb{P}\left((A=a)\right)$$

and $\mathbb{P}\left(L_j | (A=a)\right)$ is the probability to be in the first $\texttt{lq}-a$ factors with r_B roots that are not skipped, i.e.,

$$\begin{aligned}\mathbb{P}\left(L_j | (A=a)\right) &= \mathbb{P}\left((B_j=1) \cap (\sum_{j'<j} B_j < \texttt{lq}-a) | (A=a)\right) \\ &= \mathbb{P}\left((B_j=1)\right) \times \mathbb{P}\left((\sum_{j'<j} B_j < \texttt{lq}-a) | (A=a)\right)\end{aligned}$$

If $a \geq k$, this probability is 0 (because in that case, we only need factors with d roots).

If $a < k$, then

$$\mathbb{P}(Q_j | L_j \cap (A=a)) = \frac{r_B \times \text{nb\_comb}(d, k-1, \texttt{lq}-1, a)}{\text{nb\_comb}(d, k, \texttt{lq}, a)}$$

C. Practical considerations and sage code

To compute the probability law of a sum of variables following binomial laws, we can use polynomials:

Given E_j a family of binomial laws (of parameter $\mathbb{P}(E_j = 1)$), $\mathbb{P}(\sum_j E_j = s)$ is the coefficient of degree s of the polynomial.

$$\prod_j (1 + (X - 1)\mathbb{P}(E_j = 1))$$

To get $\mathbb{P}(\sum_j E_j \leq s)$ for instance, we can truncate this polynomial at degree s, and evaluate in 1. In the following Sagemath code, we are using truncations, products and evaluations of polynomials to compute probabilities.

To compute the probability of appearance of q factors, we used the following SAGEMATH [7] code:

```
from collections import defaultdict
from sage.functions.other import ceil
from sage.rings.real_mpfr import RR
from sage.functions.other import binomial
from sage.misc.misc_c import prod
from sage.misc.functional import log
from sage.rings.fast_arith import prime_range
from sage.rings.polynomial.polynomial_ring_constructor import PolynomialRing

def nb_comb(d,k,lq,a):
    if d == 6:
        return sum([binomial(a, u) * binomial(lq-a, k-u) * d^u * 2^(k-u)
                for u in range(a+1)])
    else:
        return sum([binomial(a, u) * binomial(lq-a, k-u) * d^u
                for u in range(a+1)])


def find_lq (d,nq,a):
    k = ceil(log(nq,d))
    lq = k
    if a > k:
        a = k
    while (nb_comb(d,k,lq,a) < nq):
        lq += 1
    return lq


Q = prime_range(2, 272)
RP = PolynomialRing(RR, 't')
t = RP.gen()


def q_distribution(d,nq,incr):
    Q1 = [q for q in Q if q % d == 1 and incr % q != 0]
    Qstar = [q for q in Q if q % d != 1 and incr % q != 0]
    # Q_skipped = [q for q in Q if incr % q == 0]


    polynomial_a = prod([1 + (1-1/q) * 1/d * (t-1) for q in Q1])
```

```
D = defaultdict(RR)

k = ceil(log(nq, d))
for q in Q1:
    pol_inf = prod([1+(1/d)*(1-1/q1)*(t-1) for q1 in Q1 if q1 < q])
    pol_sup = prod([1+(1/d)*(1-1/q1)*(t-1) for q1 in Q1 if q1 > q])
    for a in range(len(Q1)):
        lq = find_lq(d,nq,a)

        if a <= k:
            comb_ratio = d*nb_comb(d,k-1,lq-1,a-1)/nb_comb(d,k,lq,a)
            D[q] += (pol_inf * pol_sup)[a-1] * ((1/d) * (1-1/q)) * comb_ratio
        else:
            D[q] += ((pol_inf % t^k)* pol_sup)[a-1] * (1/d) * (1-1/q)

for a in range(k):
    pr_a = polynomial_a[a]
    lq = find_lq(d, nq, a)
    polynomial_b = RP(1)
    for q in Qstar:
        pr_noskip = (1 - 1/q)
        if d == 6:
            pr_noskip *= 1/2
        pr_select = (polynomial_b % t**(lq-a))(1) * pr_noskip
        polynomial_b *= (1 + pr_noskip * (t-1))
        comb_ratio = nb_comb(d, k-1, lq-1, a) / nb_comb(d, k, lq, a)
        if d==6:
            comb_ratio *= 2
        D[q] += pr_a * pr_select * comb_ratio
return D
```