A New Ranking Function for Polynomial Selection in the Number Field Sieve

Nicolas David, Paul Zimmermann

To cite this version:

Nicolas David, Paul Zimmermann. A New Ranking Function for Polynomial Selection in the Number Field Sieve. Contemporary mathematics, American Mathematical Society, 2020, 75 Years of Mathematics of Computation, 754, pp.315-325. hal-02151093v4

HAL Id: hal-02151093

https://hal.inria.fr/hal-02151093v4

Submitted on 17 Jun 2020

A New Ranking Function for Polynomial Selection in the Number Field Sieve

Nicolas David and Paul Zimmermann

Abstract. This article explains why the classical Murphy-E ranking function might fail to correctly rank polynomial pairs in the Number Field Sieve, and proposes a new ranking function.

1. Introduction

The General Number Field Sieve (GNFS) is the best algorithm currently known to factor integers, and was used for the RSA-768 factorization record [**7**]. The first stage of GNFS, called *polynomial selection*, chooses two polynomials $f(x)$ and $g(x)$ to factor the target integer n. The first step of polynomial selection, called *size optimization*, selects many polynomial pairs with small norm, while the second step, called *root optimization*, optimizes the root properties of the most promising pairs. The current state-of-the-art of root optimization is described in [**3**].

The ultimate goal of polynomial selection is to find the best polynomial pair to factor the given integer n. To compare two polynomial pairs, one uses a *ranking function*: it associates to each polynomial pair a real number, and the best polynomial pair is assumed to correspond to the largest number. Usually a first ranking function is used during size optimization: it has to be fast, but does not need to be very accurate. A second ranking function is used during root optimization, to select a few polynomial pairs, which are compared by a *sieving test.* A sieving test consists in looking for real relations over a sample of the whole sieving domain. Figure 1 summarizes the polynomial selection workflow.

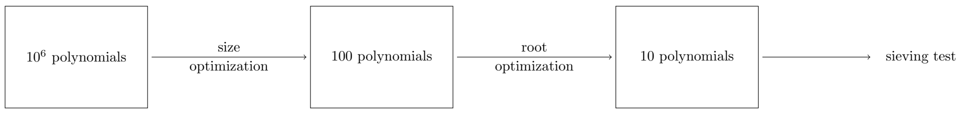

Figure 1. Polynomial selection workflow.

Since sieving tests are expensive, one keeps few polynomial pairs after root optimization, therefore the ranking function used for that step should be very accurate. In this article, we consider the extreme case where the sieving test is skipped, i.e., one chooses the polynomial pair with the best ranking after root optimization. This case is quite important, since it corresponds to what software tools usually do to factor integers of moderate size.

The Murphy-E value, introduced in [**8**], is commonly used as this ultimate ranking function. We demonstrate in this article that it can fail —by a non-negligible factor— to identify the best polynomial pair that would be found by a sieving test. We explain why it fails, and we propose an alternative ranking function.

The article is organized as follows. Section 2 describes the classical ranking function used, namely Murphy-E, and explains on one example why it can fail to identify the best polynomial pair. Section 3 introduces the new ranking function E', and explains how to compute it. Finally, Section 4 compares the classical and new ranking functions. An appendix gives details to reproduce the examples and experiments of this article.

2. Classical Polynomial Selection

Remember that to factor an integer n with GNFS, the polynomial selection step selects two polynomials with integer coefficients, $f(x)$ and $g(x)$, having a common root modulo n. With the current state-of-the-art of polynomial selection [**6, 2**], the polynomial $g(x)$ is linear, while $f(x)$ has degree 6 for current factorization records [**7**]. In the first stage (size optimization), millions of polynomial pairs are generated, and a few are kept (say 100, see Figure 1). In the second stage (root optimization), each polynomial pair kept in stage 1 is *root-optimized* (see [**3**]), and yields a unique *root-optimized* polynomial pair. Finally, the root-optimized polynomial pair with the best ranking function is selected for the actual factorization.

2.1. The skewness.

In classical linear polynomial selection, the algebraic polynomial $f(x) = f_d x^d + \cdots + f_0$ is usually *skewed*, i.e., the leading coefficient f_d is much smaller than f_0. One defines the *skewness* of f as a real $s > 1$ such that the $|f_i|s^i$ have the same magnitude [**6**, Definition 3.1]. The *sieving* phase identifies values $F(a, b), G(a, b)$ which are simultaneously smooth, for a, b small integers, and $F(x, y), G(x, y)$ the homogeneous polynomials corresponding to $f(x)$ and $g(x)$ respectively. The a, b values are chosen so that $|a| < A$, $b < B$, with $A \approx sB$. In such a way, the terms $f_i a^i b^{d-i}$ are all of the same magnitude since $|f_i|A^i B^{d-i} \approx |f_i|s^i B^d$. The optimal skewness depends on the choice of norm for f: in this article we consider the circular norm, defined in [**2**, Equation (2.1)].

2.2. The α value.

A commonly used measure to rank polynomials during root-optimization is the α value. It was introduced by Murphy [**8**], we use here the definition from [**3**]. Given a polynomial $f(x)$, and an integer bound B,

$$\alpha(f,B) = \sum_{\substack{p \le B \\ p \text{ prime}}} \left(\frac{1}{p-1} - \nu_p(F) \right) \log p, \tag{1}$$

where $\nu_p(F)$ is the expected p-valuation of $F(a,b)$, for a,b coprime integers. When the context is clear, we simply write $\alpha(f)$.

Equation (1) can be written $\alpha = \sum_{p \le B} \alpha_p$ —the summation being intended over primes only—, where $\alpha_p = (1/(p-1) - E[X_p]) \log p$, and X_p is the random variable corresponding to the p-valuation of $F(a,b)$ for coprime random integers a and b. The term $1/(p-1)\log p$ corresponds to the expected p-valuation of a random integer. Thus α measures the expected logarithm benefit for the B-smooth part: if $\alpha > 0$ (resp. $\alpha < 0$), then $F(a,b)$ has a B-smooth part which is smaller (resp. larger) on average than random integers.

The α value cannot be used alone to compare two polynomials, since it does not take into account the polynomial norms, i.e., the values of $|F(a,b)|$. Murphy proposes to use the following ranking function during size optimization [**8**]:

$$\log \operatorname{norm}(f) + \alpha(f),$$

where $\operatorname{norm}(f)$ is any norm taking into account the skewness. This ranking function makes sense, since $\log \operatorname{norm}(f)$ relates to the logarithm of the norms $|F(a,b)|$, while $\alpha(f)$ takes into account the logarithm benefit for the smooth part.

2.3. The Murphy-E value. The number of real roots of f influences the yield of the polynomial: since $F(a,b) = b^d f(a/b)$ —where d is the degree of f—, $F(a,b)$ will be smaller than expected when a/b is near a root of f. As a consequence, a polynomial with a large number of real roots is more likely to yield smooth $F(a,b)$ values. Since the influence of real roots is not measured by $\alpha(f)$, a better ranking function was introduced in [**8**], namely the Murphy-E value. Moreover, Murphy-E takes into account both polynomials f and g. We consider here the definition from [**6**]:

$$E = \frac{6}{\pi^2} \int_{\mathcal{A}} \rho\left(\frac{\log(F(x,y)) + \alpha(f)}{\log B_f} \right) \rho\left(\frac{\log(G(x,y)) + \alpha(g)}{\log B_g} \right) \mathrm{d}x\mathrm{d}y, \tag{2}$$

where B_f is the smoothness bound on the algebraic side (polynomial f), B_g the smoothness bound on the rational side (polynomial g), and $\mathcal{A}$ the sieving area. The Murphy-E value defined by Equation (2) grows with the probability that both $F(a,b)$ and $G(a,b)$ are smooth, thus the larger, the better.

2.4. Notations. In the whole article, p denotes a prime, thus when used as a summation index, it is implicit that the sum is over primes only; $X_p(f)$ —or simply X_p if clear from the context— is the random variable corresponding to the p-valuation of $F(a, b)$, for coprime random integers a and b; B is an integer bound (a commonly used value is $B = 2000$); $Y(f)$ —or simply Y— is the random variable corresponding to the logarithm of the B-smooth part of $F(a, b)$, for coprime random integers a and b:

$$Y = \sum_{p \le B} X_p \log p.$$

We will also occasionnally write $Y_p := X_p \log p$.

2.5. A Motivating Example. While comparing releases 2.2.0 and 2.3.0 of CADO-NFS [**9**], Pierrick Gaudry found a regression in the polynomial selection for the RSA-155 number from the RSA Factoring Challenge [**1**]. The (f_1, g_1) polynomial pair found by CADO-NFS 2.2.0 is:

$$\begin{aligned} f_1 &= 2420600x^5 - 3336940896058x^4 + 4864742815969149671x^3 \\ &+ 1290870867692888810959166x^2 - 5527137948673643281698832 69654x \\ &- 5996185195483610727482217583 9388980 \\ g_1 &= 1037889490142461625 79x - 5012761682008448923162 35022847 \end{aligned}$$

while the (f_2, g_2) pair found by CADO-NFS 2.3.0 is:

$$\begin{aligned} f_2 &= 745920x^5 - 2076894693938x^4 - 681801484930531955x^3 \\ &+ 1614628025120092091914179x^2 + 18890487216790826593939581 8184x \\ &- 5878691920285948613382134329 8647600 \\ g_2 &= 7756938953438894260924 7x - 5479738059625962381416 89365703 \end{aligned}$$

Both polynomial pairs were found using Kleinjung's algorithm [**6**], with post-processing from [**2**].[1] The values of α are respectively $\alpha(f_1) = -6.452$ and $\alpha(f_2) = -5.685$ (with $B = 2000$). The Murphy-E value of (f_1, g_1) is $E_1 = 1.02 \cdot 10^{-10}$, and that of (f_2, g_2) is $E_2 = 1.11 \cdot 10^{-10}$. Thus one can expect that (f_2, g_2) is significantly better than (f_1, g_1).

However, sieving experiments disagree with that ranking: on a similar test, one obtains 26214 relations with (f_1, g_1), against only 24715 for (f_2, g_2) (see appendix for more details).

Altogether, the discrepancy between the Murphy-E values (1.02 against 1.11) and the sieving test (26214 against 24715) is about 15%, which is not negligible at all. We try to explain this discrepancy in the rest of the article.

2.6. Why Murphy-E fails. Figure 2 shows the distribution of the logarithm benefit on *actual relations*, for both f_1 and f_2. For random $F(a, b)$ values —instead of actual relations—, those distributions would have an average of $-\alpha(f_1)$ and $-\alpha(f_2)$ respectively, by definition of α. One notices that: (i) the logarithm benefit is larger than $|\alpha|$ for both polynomials; and (ii) the difference between the average logarithm benefit and $|\alpha|$ is much larger for f_1 than for f_2. In the rest of this article we explain this difference, and we provide a new ranking function E' that captures this difference.

[1] The pair (f_1, g_1) could not be found by CADO-NFS 2.3.0 since some polynomial selection parameters changed between releases 2.2.0 and 2.3.0.

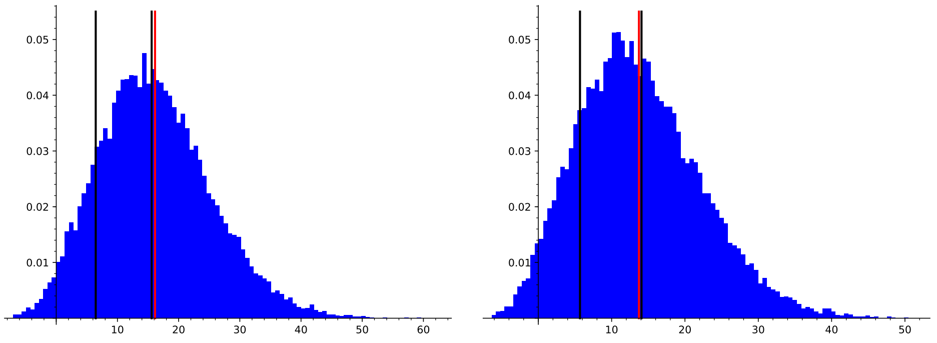

FIGURE 2. Distribution of the logarithm benefit for actual relations for f_1 (left) and f_2 (right), the left black line representing $-\alpha$, i.e., the average for random $F(a,b)$, the red line the average logarithm benefit for actual relations, and the right black line $-\alpha + \sigma_i$ where σ_i, $1 \le i \le 2$, is the corresponding standard deviation.

3. A New Ranking Function

Remember that X_p is the random variable corresponding to the p-valuation of $F(a,b)$ for coprime integers a, b, where $F(x,y)$ is the homogenous polynomial associated to f. The mean of X_p can be computed algorithmically, from which one can deduce the value of α_p,

$$\alpha_p(f) = \left(\frac{1}{p-1} - E[X_p]\right) \log p,$$

and in turn the value of $\alpha = \sum_{p \le B} \alpha_p$. One thus has

$$\alpha(f) = \sum_{p \le B} \frac{\log p}{p-1} - E[Y],$$

where Y is the random variable corresponding to the logarithm of the B-smooth part of $F(a,b)$. Up to sign and a constant, $\alpha(f)$ represents the first moment of the random variable Y.

We propose to also take into account the *second moment* of the random variable Y. The rationale is that the first moment is not enough to rank two polynomials, as demonstrated by the example from §2.5 and Figure 2. If f_1 and f_2 have the same first moment, but f_2 has a larger variance, then f_2 is more likely to produce smooth relations, since most smooth relations correspond to the tail of the distribution of the random variable Y. More precisely, we propose to replace the constant $\alpha(f)$ by a measure μ in Equation (2), where μ stands for $\sum_{p \le B} \frac{\log p}{p-1} - Y$:

$$E' = \frac{6}{\pi^2} \int_{\mathcal{A}} \rho\left(\frac{\log(F(x,y)) + \mu}{\log B_f}\right) \rho\left(\frac{\log(G(x,y)) + \alpha(g)}{\log B_g}\right) \mathrm{d}x\mathrm{d}y\mathrm{d}\mu. \tag{3}$$

Remark. One might wonder why only $\alpha(f)$ is replaced by a distribution in Equation (3), and not also $\alpha(g)$. The reason is that for linear polynomial selection, where $g(x)$ has degree 1, it has exactly one simple root for every prime p (either affine or projective), thus the random variable $X_p(g)$ has the same distribution for every polynomial $g(x)$. However, for non-linear polynomial selection, as for example when using Joux-Lercier algorithm for the discrete logarithm [**5**], it might make sense to replace also $\alpha(g)$ by a distribution.

The rest of this section is organized as follows. In §3.1, we explain how to compute the mean and variance of the random variable X_p. Then in §3.2, we explain how to model the distribution Y, that will enable one to numerical integrate the measure μ in Equation (3).

Algorithm 1 DistValuationAffine

```
Input: a polynomial $f(x)$, a prime $p$, an integer $w \geq 0$
Output: $E[X_p(f)], E[X_p^2(f)]$ (when $w = 0$)
 1: $v \leftarrow \mathrm{val}_p(f)$
 2: $f \leftarrow f/p^v$
 3: $E, F, w \leftarrow v, w^2, w + v$
 4: for $r$ in roots($f$ mod $p$) do
 5:     if $f'(r) \bmod p \neq 0$ then
 6:         $E \leftarrow E + 1/(p-1)$
 7:         $F \leftarrow F + 2w/(p-1) + (p+1)/(p-1)^2$
 8:     else
 9:         $(E_r, F_r) \leftarrow$ DistValuationAffine($f(r + px), p, w$)
10:         $E \leftarrow E + E_r/p, \quad F \leftarrow F + F_r/p - w^2/p$
11: return $E, F$
```

3.1. Computing the mean and variance of X_p. Algorithm 1, when called with $w = 0$, returns the first two moments $E[X_p(f)]$ and $E[X_p^2(f)]$ of its input $f(x)$. More generally, when called with some integer $w \geq 0$, it returns the first two moments $E[Z_p(f)]$ and $E[Z_p^2(f)]$, with $Z_p = w + X_p$. Indeed, consider the tree formed by the recursive calls of Algorithm 1: each multiple root of $f \bmod p$ yields a recursive call. Assume the current valuation is w after line 3, i.e., we have to add w to the p-valuation of the current polynomial f:

- each simple root lifts up to infinity, i.e., we have valuation $w + 1$ with probability $1/p$, valuation $w+2$ with probability $1/p^2$, ... This explains the increment $1/(p-1)$ to E. For the second moment, we have w^2 with probability $1 - 1/p$, $(w + 1)^2$ with probability

$1/p - 1/p^2$, $(w+2)^2$ with probability $1/p^2 - 1/p^3$, ... This yields:

$$\begin{aligned}\sum_{i\geq 0}(w+i)^2\left(\frac{1}{p^i}-\frac{1}{p^{i+1}}\right) &= w^2\sum_{i\geq 0}\left(\frac{1}{p^i}-\frac{1}{p^{i+1}}\right)+2w\sum_{i\geq 0} i\left(\frac{1}{p^i}-\frac{1}{p^{i+1}}\right)\\ &+ \sum_{i\geq 0} i^2\left(\frac{1}{p^i}-\frac{1}{p^{i+1}}\right)\\ &= w^2+\frac{2w}{p-1}+\frac{p+1}{(p-1)^2}.\end{aligned}$$

Since the w^2 term was already stored in F at line 3, it remains to add the two other terms;

- in case of a multiple root, the recursive call at line 9 computes the mean and variance associated to the polynomial $f(r+px)$, which need to be multiplied by $1/p$, the probability of that root; finally, we have a $-w^2/p$ correction, since this was already taken into account at line 3 for this root.

This algorithm, for the computation of $E[X_p(f)]$ only, can also be found in [**4**, Algorithm A.2].

Algorithm 2 DistValuation

```
Input: a polynomial $f(x)$, a prime $p$
Output: mean $E[X_p]$ and variance $\mathrm{Var}[X_p]$ of $X_p(f)$, as defined in §3
  $E_{\mathrm{aff}}, F_{\mathrm{aff}} \leftarrow \mathrm{DistValuationAffine}(f, p, 0)$
  $E_{\mathrm{proj}}, F_{\mathrm{proj}} \leftarrow \mathrm{DistValuationAffine}(\mathrm{rev}(f)(px), p, 0)$
  $E \leftarrow (pE_{\mathrm{aff}} + E_{\mathrm{proj}})/(p+1)$
  $F \leftarrow (pF_{\mathrm{aff}} + F_{\mathrm{proj}})/(p+1)$
  return $E, F - E^2$
```

Remember that X_p is the random variable corresponding to the p-valuation of $F(a,b)$ for coprime integers a, b. We claim that Algorithm 2 returns the mean $E[X_p]$ and variance $\mathrm{Var}[X_p]$ of X_p. It is based on Algorithm 1, which returns the first two moments of X_p, taking into account only the affine roots (not the projective ones). By looking at those algorithms, it can be seen that $E[X_p]$ and $\mathrm{Var}[X_p]$ are rational numbers. From those algorithms, one deduces the value of $\alpha_p = (1/(p-1) - E[X_p])\log p$, and the corresponding standard deviation. Figure 3 gives α_p and the corresponding standard deviation $\sigma_p := \sqrt{\mathrm{Var}[Y_p]}$ for both f_1 and f_2, for $p \leq 13$. One sees on this figure that the values of σ_p are consistently larger for f_1 than those for f_2.

3.2. Estimating the Distribution Y. We are interested in the distribution of the random variable:

$$Y = \sum_{p\leq B} X_p \log p.$$

p	2	3	5	7	11	13
α_p	-0.924	-0.549	-0.604	-0.967	-0.819	-0.779
σ_p	1.424	1.453	1.190	1.787	1.641	1.409
α_p	-1.444	-0.275	-0.872	-1.013	-0.619	-0.183
σ_p	1.113	1.064	1.112	1.521	1.280	1.015

FIGURE 3. Value of α_p and corresponding standard deviation of the random variable Y_p for the polynomial f_1 (top) and f_2 (bottom), for $p \leq 13$.

If p and q are distinct primes, the p-valuation and q-valuation of $F(a,b)$ are independent, thus their mean and variance do sum up. It follows:

$$E[Y] = \sum_{p\leq B} E[X_p] \log p, \qquad \mathrm{Var}[Y] = \sum_{p\leq B} \mathrm{Var}[X_p] \log^2 p. \tag{4}$$

As demonstrated in Figure 4, Y is nicely approximated by a non-central chi-squared distribution, whose mean (resp. variance) is the sum of the means (resp. variances) of the Y_p. This phenomenon can be motivated as follows. Remember that the non-central chi-squared distribution is obtained as the sum of squares of k independent and normally distributed random variables. Here, the random variables $X_p \log p$ are indeed independent and positive, but they are not the squares of normally distributed random variables. Moreover the distribution X_p is discrete, not continuous. However, when many random variables $X_p \log p$ are accumulated, their sum looks like a continuous distribution, which resembles a non-central chi-squared distribution, as seen on Figure 4.

Figure 4 shows the distribution of Y for random coprime pairs (a,b), both for the polynomial $F_1(a,b)$ found by CADO-NFS 2.2.0, and $F_2(a,b)$ found by CADO-NFS 2.3.0. For each polynomial, one computes the mean and variance of Y using the algorithms from §3.1, and one deduces the parameters k and λ of a non-central chi-squared distribution, knowing that such a distribution has mean $k+\lambda$ and variance $2(k+2\lambda)$. The figure shows a very good match of the histogram with the non-central chi-squared distribution (solid black line).

In summary the new ranking function E' is computed as follows:

(1) use Algorithms 1 and 2 to compute $E[X_p]$ and $\mathrm{Var}[X_p]$ for primes $p \leq B$;
(2) compute $E[Y]$ and $\mathrm{Var}[Y]$ using Equations (4), and deduce $\alpha = \sum_{p\leq B} \log p/(p-1) - E[Y]$, the corresponding variance being the same as $\mathrm{Var}[Y]$;
(3) deduce the parameters k and λ for the corresponding non-central chi-squared distribution Y using:

$$\alpha = k + \lambda, \qquad \mathrm{Var}[Y] = 2(k+2\lambda);$$

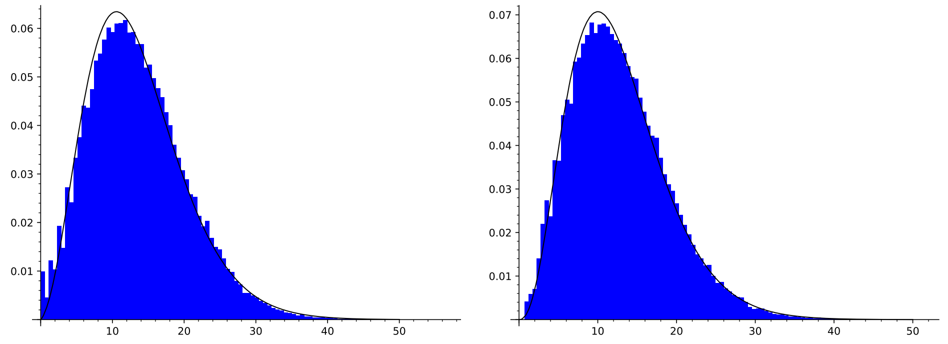

FIGURE 4. Distribution (blue histogram) of the logarithm Y of the B-smooth part of $F_1(a,b)$ (left) and $F_2(a,b)$ (right) for random coprime (a,b) pairs, and corresponding non-central chi-squared distribution (black line) deduced from Algorithm 2.

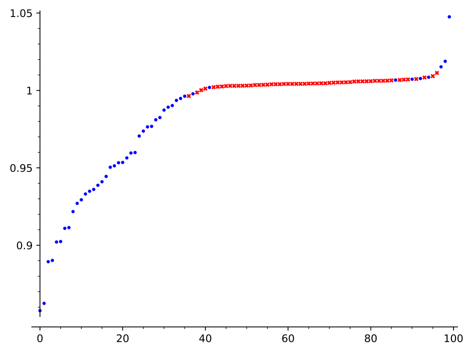

FIGURE 5. Ratio t'/t to factor 100 integers of 120 digits where t is the cpu time using the classical Murphy-E ranking function, and t' with the new ranking function E'. Cases where both polynomial pairs are the same are shown in red crosses.

(4) use Equation (3) to compute E', where $\mu = \sum_{p \le B} \log p/(p-1) - Y$, and Y is a non-central chi-squared distribution of parameters k and λ.

4. Experimental Results

To compare the new ranking function E' to the classical one E, we have done the following experiment. We have generated 100 RSA-like inputs of 120 digits as follows. Let p and q be the prime factors of the RSA-120 challenge number. We define $p_0 = p$, and $p_{k+1} = \text{nextprime}(p_k)$, similarly for q_k, and $N_k = p_k q_k$. We have factored each of those 100 numbers with CADO-NFS, first with the classical Murphy-E ranking function, and then with the new function E'. Both factorizations were done on the same processor (32-core Intel Xeon Gold 6130 at 2.10GHz), with hyper-threading and turbo-boost disabled.

Figure 5 shows the ratio t'/t of the cumulated cpu time needed to factor those 100 numbers, where for t the selected polynomial pair was the one with the best classical ranking function E, and for t' it was with the new ranking function E'. For clarity, the numbers are sorted by increasing value of t'/t.

For 56 numbers out of 100, the selected polynomial pairs are the same; this corresponds to red crosses on Figure 5, which are all near 1 as expected. We see on that figure that, apart from a few outliers, the new function E' either finds the same polynomial pair or a better one. Analyzing in detail the outlier that gives a ratio $t'/t \approx 1.044$ on Figure 5, one finds that if the bound B for computing α is increased from 2000 to 10000, the timings become very close. This suggest that $B = 2000$ might be too small to accurately estimate $\alpha(f)$; for the top ten polynomials coming out of the rootsieve — either with the classical ranking function or the new one —, one could use for B the factor base bound used on the algebraic side.

Going back to our motivating example (§2.5), the value of E' — still with $B = 2000$ — is $2.47 \cdot 10^{-10}$ for the (f_1, g_1) polynomial pair, and $2.28 \cdot 10^{-10}$ for (f_2, g_2), thus the new ranking function would select (f_1, g_1). With $B = 10000$, we get $E_1 = 1.03 \cdot 10^{-10}$ and $E_2 = 1.12 \cdot 10^{-10}$, thus even with a larger value of B, the classical ranking function would select (f_2, g_2); while with $E'_1 = 3.63 \cdot 10^{-10}$ and $E'_2 = 3.37 \cdot 10^{-10}$ the new ranking function would select (f_1, g_1).

Acknowledgements. The authors thank Pierrick Gaudry who pointed out the regression in CADO-NFS 2.3.0, which was the main motivation for this work, and Anne Gégout-Petit for discussions about the non-central chi-squared distribution. They also thank the anonymous referee for her/his very useful comments. Experiments presented in this article were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (see `https://www.grid5000.fr`).

References

[1] RSA factoring challenge. `https://en.wikipedia.org/wiki/RSA_Factoring_Challenge`.

[2] Bai, S., Bouvier, C., Kruppa, A., and Zimmermann, P. Better polynomials for GNFS. *Mathematics of Computation 85* (2016), 861–873.

[3] Bai, S., Brent, R., and Thomé, E. Root optimization of polynomials in the number field sieve. *Mathematics of Computation 84*, 295 (2015), 2447–2457.

[4] Guillevic, A., and Singh, S. On the alpha value of polynomials in the tower number field sieve algorithm. Cryptology ePrint Archive, Report 2019/885, 2019. `https://eprint.iacr.org/2019/885`.

[5] Joux, A., and Lercier, R. 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 (2003), 953–967.

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

[7] Kleinjung, T., Aoki, K., Franke, J., Lenstra, A. K., Thomé, E., Bos, J. W., Gaudry, P., Kruppa, A., Montgomery, P. L., Osvik, D. A., te Riele, H., Timofeev, A., and Zimmermann, P. Factorization of a 768-bit RSA modulus. In *CRYPTO 2010 Advances in Cryptology - CRYPTO 2010* (Santa Barbara, USA, 2010), T. Rabin, Ed., vol. 6223 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 333–350.

[8] Murphy, B. A. *Polynomial Selection for the Number Field Sieve Integer Factorisation Algorithm.* PhD thesis, Australian National University, 1999. 144 pages.

[9] The CADO-NFS Development Team. CADO-NFS, an implementation of the number field sieve algorithm. `http://cado-nfs.gforge.inria.fr/`, 2017. Release 2.3.0.

Reproducibility Appendix

This appendix provides elements that were skipped to avoid obfuscating the article with too many details, but which are nevertheless needed to reproduce our results.

In §2.5, the polynomial pair (f_1, g_1) has skewness 426112, while (f_2, g_2) has skewness 608078. The Murphy-E values were computed with $B_f = 30940618$, $B_g = 17246818$, and area $\mathcal{A} = 2^{27} B_f \approx 4.15 \cdot 10^{15}$.

Still in §2.5, the sieving experiments were done with lattice sieving, using the first 1,000 special-q's up from 20,000,000, with factor base bound 10,000,000 on the rational side and 20,000,000 on the algebraic side, large prime bounds of 2^{29}, with two large primes on each side. One obtains 26214 relations for (f_1, g_1) against only 24715 for (f_2, g_2).

To produce Figure 4, we used 10^5 coprime pairs (a, b) randomly drawn from $-10^6 \le a < 10^6$ and $1 \le b < 10^6$.

To produce Figure 5, we used $B = 2000$ as smoothness bound (cf. Equation (1)). In Equation (3), we have $\mu = c - t$, where $c = \sum_{p \le B} \log p/(p-1)$, and t is the logarithm of the B-smooth part (horizontal axis on Figure 4). In Equation (3), we integrated from $t = 0$ to $t = 6c$, where $6c \approx 42.29$ with $B = 2000$. Taking a larger upper integration bound for t gives worse results, most probably because the right tails on Figure 4 do not match the non-central chi-squared distribution. For the classical Murphy-E, we used CADO-NFS, branch `master`, revision `f20905d4d`; for the new ranking function, we used branch `dist-alpha`, revision `d9711a70e`, in both cases with GCC 8.3.0 and `CFLAGS=-O3 -funroll-loops -DNDEBUG`.

Figure 6 shows how the integrand of Equations (2) for E and (3) for E' compares to the actual relations found. On the right graph, one sees that the red curve is above the actual relations found in several places, which concurs with the fact that E overestimates the relations found for (f_2, g_2), whereas the green curve (E') is often below the histogram. On the left graph, on the contrary the red curve is below the actual relations found in several places,

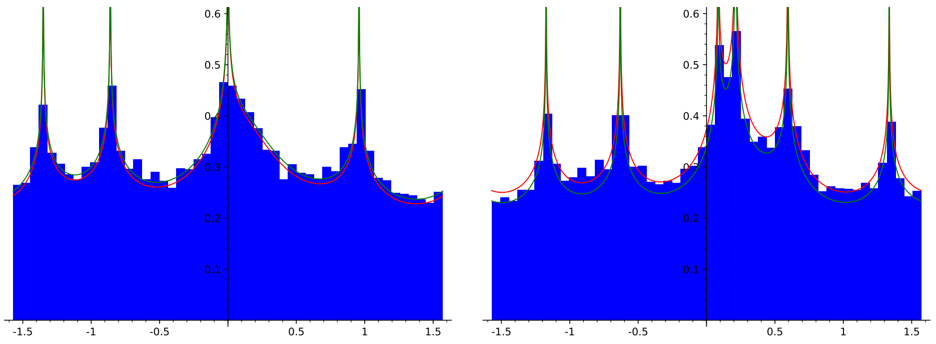

Figure 6. Plot over $[-\pi/2, \pi/2]$ of the integrand of Equation (2) in red, together with the histogram of $\operatorname{atan}(sa/b)$ for the relations found, where s is the skewness, for (f_1, g_1) (left) and for (f_2, g_2) (right), and in green the same for Equation (3).

which concurs with the fact that E underestimates the relations found for (f_1, g_1).

École Normale Supérieure Paris-Saclay, Cachan, France

Université de Lorraine, CNRS, Inria, LORIA, F-54000, Nancy, France