We are on the same side. Alternative sieving strategies for the number field sieve

Charles Bouillaguet, Ambroise Fleury, Pierre-Alain Fouque, Paul Kirchner

▶ **To cite this version:**

Charles Bouillaguet, Ambroise Fleury, Pierre-Alain Fouque, Paul Kirchner. We are on the same side. Alternative sieving strategies for the number field sieve. ASIACRYPT 2023 - 29th International Conference on the Theory and Application of Cryptology and Information Security, Dec 2023, Guangzhou, China. pp.138-166, 10.1007/978-981-99-8730-6_5 . hal-04112671

HAL Id: hal-04112671
https://inria.hal.science/hal-04112671v1

Submitted on 31 May 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons CC0 - Public Domain Dedication 4.0 International License

We Are on the Same Side. Alternative Sieving Strategies for the Number Field Sieve

Charles Bouillaguet[1][0000−0001−9416−6244], Ambroise Fleury[2], Pierre-Alain Fouque[3], and Paul Kirchner[3]

[1] Sorbonne Université, CNRS, LIP6, F-75005 Paris, France
charles.bouillaguet@lip6.fr
[2] Université Paris-Saclay, CEA, List, F-91120, Palaiseau, France
ambroise.fleury@cea.fr
[3] Univ Rennes, CNRS, IRISA
{pierre-alain.fouque,paul.kirchner}@irisa.fr

Abstract. The Number Field Sieve (NFS) is the state-of-the art algorithm for integer factoring, and sieving is a crucial step in the NFS. It is a very time-consuming operation, whose goal is to collect many relations. The ultimate goal is to generate random smooth integers mod N with their prime decomposition, where smooth is defined on the rational and algebraic sides according to two prime factor bases.
In modern factorization tool, such as Cado-NFS, sieving is split into different stages depending on the size of the primes, but defining good parameters for all stages is based on heuristic and practical arguments. At the beginning, candidates are sieved by small primes on both sides, and if they pass the test, they continue to the next stages with bigger primes, up to the final one where we factor the remaining part using the ECM algorithm. On the one hand, first stages are fast but many false relations pass them, and we spend a lot of time with useless relations. On the other hand final stages are more time demanding but outputs less relations. It is not easy to evaluate the performance of the best strategy on the overall sieving step since it depends on the distribution of numbers that results at each stage.
In this article, we try to examine different sieving strategies to speed up this step since many improvements have been done on all other steps of the NFS. Based on the relations collected during the RSA-250 factorization and all parameters, we try to study different strategies to better understand this step. Many strategies have been defined since the discovery of NFS, and we provide here an experimental evaluation.

1 Introduction

The RSA cryptosystem was one of the first invented primitives of public-key cryptography. It still plays a dominant role in the computer security ecosystem, even though post-quantum alternatives are gaining traction. These primitives are connected to hard number-theoretic questions such as integer factorization,

which thus underpin most of modern cryptography, now and also for the next years.

The major cryptanalytic tool to assess the hardness of integer factoring, and therefore the security of RSA-based cryptography, is the Number Field Sieve algorithm (NFS). While the asymptotic complexity of this algorithm is well-known, it is often difficult to estimate the time and resources that are needed to factor an integer. As such, all regulatory bodies recommend that people either avoid RSA entirely, or prefer large RSA key sizes for safety, *e.g.* at least 2048 bits until 2030, and at least 3072 bits after this date. In environments where computing power is plentiful, this recommendation is most often followed. Yet, we do rely on cryptography that uses smaller key sizes. The first authors credit card has a 1152-bit RSA public key which is valid until 2026. In some countries, the complete certification chain credit cards hinges on the security of 1408-bit RSA. This is well below recommended key sizes, but also well above the latest published academic records (829 bits [6]).

The regular publication of computational records [14,6] enables the cryptographic community and the greater public to gain a better understanding of the actual security level offered by the RSA cryptosystem. For this purpose, high-quality implementations of the NFS algorithm are required. The most recent records, the factorization of a 250-digit RSA challenge key, was done using Cado-NFS. It is an open-source software, whose source code is publicly available.

The most time-consuming step of a large factorization using the NFS is the collection of many *relations*: in the recent RSA-250 records, it required 2450 core-years, which is 90% of the total computation time. A "relation" in the NFS is a pair of small integers (a, b) such that two homogeneous polynomials $F_0(a, b)$ and $F_1(a, b)$ are sufficiently smooth. $F_0(a, b)$ is often called the "rational norm" and $F_1(a, b)$ is the "algebraic norm".

Collecting relations can be done using many algorithms; among these, *sieving* is one of the most efficient. Cado-NFS uses a highly optimized sieve. It follows that improving sieving algorithms would have a practical impact on the performance of integer factoring algorithms, and in turn on the security of RSA.

The "batch smooth part" algorithm of Bernstein [4] has also been used with practical success in the recent records [6]. It has been used in combination with sieving: sieving happened on the algebraic side, while Bernstein's algorithm was used on the surviving (a, b) pairs on the rational side.

In this paper, we explore the idea of combining sieving and batch smooth part extraction *on the same side*. Our guiding principle is that sieving small primes is costly because they "hit" more often than large ones. We explore the possibility of not sieving primes less than, say about 100000, and instead recover the missing small factors using Bernstein's algorithm.

Our contribution is two-fold. First, we describe statistical properties of the relations collected during the factorization of RSA-250. While it was widely assumed that a prime factor p occurs in a "random" relation with probability $1/p$, we observe (and justify theoretically) that p occur with probability $1/p^\alpha$ for some $\alpha < 1$ that we characterize. We also propose an empirical model of

the number of relations found per special-$\mathfrak{q}$, which enables us to reason about alternative relation collection strategies; in particular, we are able to assess the actual effective of a relation collection algorithm that finds less relations but faster. These might need to sieve more special-$\mathfrak{q}$, with diminishing returns.

Second, we implemented the combination of sieving and batch smooth part extraction described above inside Cado-NFS, thanks to its open-source nature. Exploiting the availability of the relations collected during the record factorization of RSA-250, we carefully selected parameters to obtain an alternative relation collection procedure that finds 90% of all relations found by the "original" Cado-NFS implementation, using only 80% of the time. Combined with the previous reasoning about, we expect that this procedure will have to process 16% more special-$\mathfrak{q}$ to collect as many relations as Cado-NFS did during the factorization of RSA-250. As such, we expect our implementation to yield a $\approx 5\%$ speedup over the whole factorization of a 250-digit integer.

This paper is organized as follows: in section 2, we describe Bernstein's batch smooth part algorithm. In section 3, we give some background on the NFS and some implementation aspects of Cado-NFS that enable us to control the size of the integers $F_1(a, b)$ (the "algebraic norms") that are sieved. In section 4 we describe the relation collection algorithm of Cado-NFS in more details. In section 5, we present a statistical analysis of the relations collected during the factorization of RSA-250. In section 6, we discuss the combination of sieving and batch smoothness detection on the same side, and we present practical results in section 7.

All experiments described in this paper were conducted on a cluster of identical nodes equipped with two Intel Xeon Gold 6130 CPUs.

2 Bernstein's Batch Smooth Part Algorithm

Given a set of integers $N = \{n_1, \ldots, n_k\}$ and a set of prime numbers P, Bernstein's batch smooth part algorithm [4] finds simultaneously for each integer in N the product of its prime factors that are in P. When P contains all the primes up to a given bound, this extracts the *smooth part* of these integers. Its time complexity is $\mathcal{O}\left(b \log^{2+o(1)} b\right)$, b being the total number of bits in N and P.

This algorithm uses product and remainder trees (see [9] for more details). The product tree is a binary tree used to compute the product of many integers. Its leaves are labelled with the input integers while each internal node is labelled with the product of the labels of its children. The computation thus proceeds from the bottom up. The root is therefore labelled with the product of all the leaves. This structure enables the faster computation of $x_1 x_2 \ldots x_n$ as often multiplies integers of similar sizes.

The remainder tree is a tree built upon a product tree to compute $z \bmod n_i$ efficiently, for a given z and a large set of n_i's. First compute the product tree of the n_i's. Then the computation proceeds from the top down as shown in Fig. 1. The root initially receives z; when an internal node receives a value x, it reduces

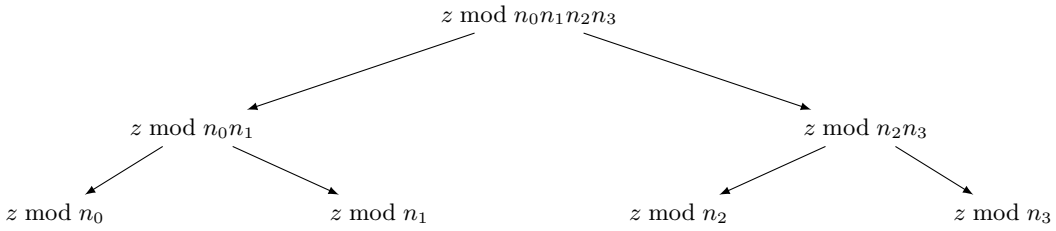

Fig. 1. Remainder tree as used in Cado-NFS

it modulo its label and sends the result to its children. The leaf labelled by n_i then outputs $z \bmod n_i$.

Bernstein later proposed "scaled remainder trees" [5] that trade divisions for multiplications, thus saving a constant factor. This variant is implemented and used in Cado-NFS. In our experiments, the scaled version is always faster than the simple version, when both are implemented using the GMP.

The batch smooth part algorithm works as follows, for a set of integers $N = \{n_1, \ldots, n_k\}$ and a set of prime numbers $P = \{p_1, \ldots, p_\ell\}$:

1. Using a product tree, compute $z \leftarrow p_1 \times \cdots \times p_\ell$
2. Using a remainder tree, compute $r_i \leftarrow z \bmod n_i$ (for all $1 \le i \le k$)
3. For $1 \le i \le k$, do:
   (a) Set $s_i \leftarrow 1$
   (b) Set $g \leftarrow \gcd(r_i, n_i)$
   (c) If $g = 1$, then s_i is the smooth part of n_i
   (d) Otherwise, set $s_i \leftarrow s_i \times g$ and $n_i \leftarrow n_i/g$, then return to step 3b

The computation of the smooth part of a huge number of integers N can be done by invoking the algorithm multiple times with small chunks of N. Because the first operation in the remainder tree algorithm is the computation of $z \bmod \prod_{i=1}^{k} n_i$, it would be a waste of resources to have z much smaller than the product of the n_i's. It is more efficient to split N in batches of about $0.5 \log_2 z$ bits, and to process them separately, keeping only z across all iterations.

Some performance measurements (using the implementation in Cado-NFS) are shown in Fig. 1.

3 The Number Field Sieve

The Number Field Sieve (NFS) is the state-of-the-art algorithm for integer factoring and discrete logarithm modulo p. Completely describing the NFS is out of the scope of this article; the interested reader can consult the books [16,8,13] or the recent computational record [6].

The full algorithm consists of several steps executed in sequence: polynomial selection, sieving, filtering, linear algebra and square root. Each of this step can be performed by a variety of sub-algorithms. Because we are mostly concerned

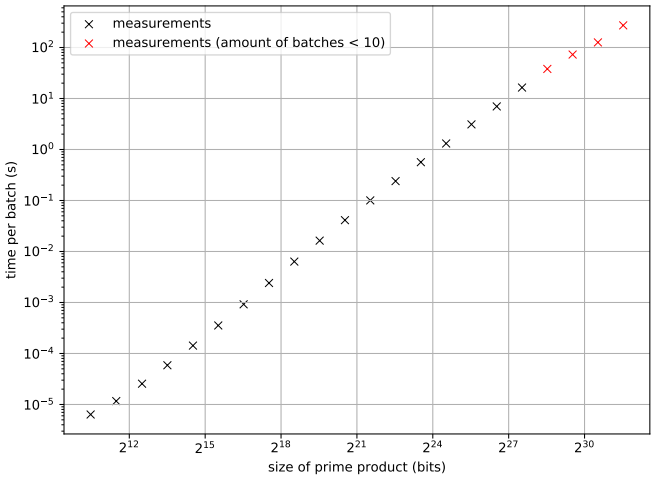

Fig. 2. Performance of batch smooth paty algorithm of [4]. The implementation is taken from Cado-NFS. This shows the time needed to process $n/2$ bits with a prime product of n bits

with sieving strategies, we only very briefly describe the global mathematical setup and we will not discuss other steps beyond the strict necessary. We also focus on the case of integer factoring.

3.1 Mathematical Background

We briefly present some mathematical background below; for more details about number fields, the reader may refer to [17].

Let $f = f_d x^d + \cdots + f_1 x + f_0$ denote a polynomial with integer coefficients, degree d, irreducible over $\mathbb{Q}$, and let $f(\alpha) = 0$ be a complex root of f. We denote by $F(x, y) = f(x/y)y^d$ the "homogenized version" of f. We work inside the number field $K := \mathbb{Q}(\alpha)$ defined by f. Any element $A \in \mathbb{Q}(\alpha)$ can be represented as $A = P(\alpha)$ for some polynomial P with rational coefficients.

A complex number is an *algebraic integer* if it is the root of a *monic* polynomial (leading coefficient 1) with integer coefficients. The set R of all algebraic integers contained in K forms the *ring of integers* of K (equivalently, the maximal order of K). In general, α is not an algebraic integer (unless f is monic, *i.e.* $f_d = 1$). However, $\hat{\alpha} = f_d \cdot \alpha$ is always an algebraic integer because $\hat{f}(x) = F(x, f_d)/f_d = x^d + f_{d-1}x^{d-1} + f_{d-2}f_d x^{d-2} + \ldots + f_0 f_d^{d-1}$ is a monic polynomial with integer coefficients and $\hat{f}(\hat{\alpha}) = {f_d}^{d-1} f(\alpha) = 0$.

The ring of integers R is often strictly larger than $\mathbb{Z}[\hat{\alpha}]$. In addition, it is well-known that R is usually not a unique factorization domain (elements do not

always have a unique factorization as a product of other irreducible elements). However R is a *Dedekind domain*: any ideal of R factors uniquely as a product of prime ideals of R. This property is not necessarily true in $\mathbb{Z}[\hat{\alpha}]$, but we have a good enough substitute (see [8, Proposition 5.4] for details).

The *norm* of $u \in K$, denoted by $N(u)$, is the product of all its conjugates in $\mathbb{Q}$. Concretely, if $u = P(\alpha)$, then $N(u) = (1/f_d)^{\deg P}\mathrm{Res}(f, P)$ — because the resultant is the product of the evaluations of P at all the complex roots of f. In general, $N(u) \in \mathbb{Q}$ and when u is an algebraic integer, then $N(u)$ is an integer. In the special case where $u = a + \alpha b$ with $a, b \in \mathbb{Z}$, then $N(f_d u) = {f_d}^{-1} \cdot F(a, b)$.

For a non-zero ideal I of $\mathbb{Z}[\hat{\alpha}]$, denote by $\|I\|$ the "norm" of I, namely the cardinality of the quotient ring $|\mathbb{Z}[\hat{\alpha}]/I|$ (it is always finite). For any two ideals I and J of $\mathbb{Z}[\hat{\alpha}]$, we have $\|IJ\| = \|I\| \times \|J\|$; it follows that I is a prime ideal of $\mathbb{Z}[\hat{\alpha}]$ whenever $\|I\|$ is a prime integer. For a principal ideal spanned by $u \in \mathbb{Z}[\hat{\alpha}]$, we have $\|\langle u \rangle\| = |N(u)|$. It follows that if $u \in \mathbb{Z}[\hat{\alpha}]$ and the principal ideal spanned by u is factored as a product of ideals: $\langle u \rangle = I_1 \dots I_k$, then we have a factorization of $N(u)$ over $\mathbb{Z}$: $N(u) = \|I_1\| \times \cdots \times \|I_k\|$. In this case, finding the factorization of $\langle u \rangle$ can be done by factoring the integer $N(u)$; from its prime factors, the (prime) ideals that appear in the factorization of $\langle u \rangle$ over $\mathbb{Z}[\hat{\alpha}]$ can usually be found relatively easily.

If I is a *prime* ideal of $\mathbb{Z}[\hat{\alpha}]$ (recall that $I \neq \mathbb{Z}[\hat{\alpha}]$ is prime iff $rs \in I$ implies that $r \in I$ or $s \in I$ for any $r, s \in \mathbb{Z}[\hat{\alpha}]$ — in particular, a product of ideals is not prime), then $I \cap \mathbb{Z}$ is also a prime ideal of $\mathbb{Z}$. As such, I contains a unique prime number p (I "lies over" p). When I is prime, the quotient $\mathbb{Z}[\hat{\alpha}]/I$ is a (finite) integral domain, hence a finite field, and therefore $\|I\| = p^k$ for some k (the *inertial degree* of I).

Suppose that p does not divide f_d. Under this assumption, we describe the prime ideals contained in $\mathbb{Z}[\hat{\alpha}]$ of inertial degree one that lie over p. These ideals are of the form $\langle p, \hat{\alpha} - r \rangle$ where $f(r) \equiv 0 \bmod p$. Here is why: suppose that I is such an ideal; then because $\mathbb{Z}[\hat{\alpha}]/I \simeq \mathbb{Z}_p$ and $\hat{\alpha}$ is a root of $\hat{f}$, the canonical ring homomorphism $\mathbb{Z}[\hat{\alpha}] \to \mathbb{Z}[\hat{\alpha}]/I$ sends $\hat{\alpha}$ to a root r of $\hat{f}$ modulo p (it follows that there are prime ideals that lie over p only when $\hat{f}$ has roots modulo p, and therefore when f does). Hence, a prime ideal gives rise to a pair (p, r) with $f(r) \equiv 0 \bmod p$. Conversely, consider such a pair (p, r) and consider the ring homomorphism $\mathbb{Z}[\hat{\alpha}] \to \mathbb{Z}_p$ that sends $\hat{\alpha}$ to r (and p to 0). Its kernel is an ideal of $\mathbb{Z}[\hat{\alpha}]$ of norm p, therefore it is a prime ideal of inertial degree one. These prime ideals can be fully described by a pair of integers (p, r).

In the context of NFS, we are interested in the factorization of ideals $J \times \langle a - b\alpha \rangle$ when $a, b \in \mathbb{Z}$ are coprime. It turns our that only prime ideals of inertial degree 1 can appear in the factorization (see [8] for a proof). When f is monic, testing if $\langle p, \alpha - r \rangle$ divides $\langle a - b\alpha \rangle$ amounts to testing if the latter is contained in the former, and this comes down to checking if $a - b\alpha$ can be written as an integer linear combination of p and $\alpha - r$. In turn, this is equivalent to $a \equiv br \bmod p$.

The theory of the NFS is often presented in the simpler situation where f is monic. In practice this is not the case: there is a performance incentive to choose a non-monic polynomial f (this gives more freedom in the choice of

f and enables the use of polynomials with smaller coefficients, and this is an advantage). The main mathematical hurdle is that α is not an algebraic integer in that case. The principal ideals $\langle a - b\alpha \rangle$ are therefore *fractional ideals*, *i.e.* they can be written as $u^{-1}I$ where u is an algebraic integer and I is an ideal of R. Indeed, $\langle a - b\alpha \rangle = {f_d}^{-1} I$ with $I = \langle f_d a - b(f_d \alpha) \rangle$, and this is a (usual) ideal of R. More precisely, I is an ideal of $\mathbb{Z}[\hat{\alpha}]$.

Montgomery introduced a clever way of dealing with the fact that α is not an algebraic integer [18]. Let $J = \{x \in R \ : \ x\alpha \in R\}$. It is easy to check that J is an ideal of R. Because $f_d \alpha \in R$, then $f_d \in J$; in addition, a complete set of generators of J is always available. In fact, $J\langle 1, \alpha \rangle = R$ and $\|J\| = f_d$. The point is that $J\langle a - b\alpha \rangle$ is an ideal of R; Its norm is precisely $\mathrm{Res}(a - bx, f) = F(a, b)$. In addition, $J \times \langle p, r - \alpha \rangle$ is a prime ideal of norm exactly p.

Up to minor details, we can multiply all ideals by J when f is not monic and work in $\mathbb{Z}[\hat{\alpha}]$ instead of $\mathbb{Z}[\alpha]$. In both cases, the norm of $\langle a - b\alpha \rangle$ (or $J \times \langle a - b\alpha \rangle$ when f is not monic) is exactly $\mathrm{Res}(a - bx, f)$, and the norm of $\langle p, \alpha - r \rangle$ (or $J \times \langle p, \alpha - r \rangle$) is exactly p.

3.2 Overview of the Algorithm

In order to factor an integer N, the first thing to do is to find two irreducible polynomials f_0 and f_1 in $\mathbb{Z}[x]$ with a common root m modulo N. These two polynomials define two algebraic number fields $\mathbb{Q}(\alpha_i)$, with $f_i(\alpha_i) = 0$. This leads to the commutative diagram shown in Fig. 3.

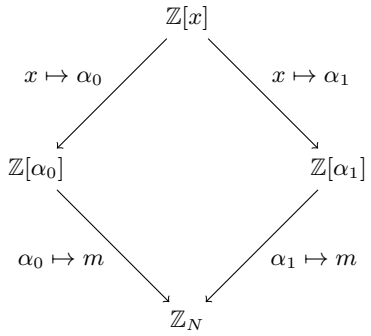

Fig. 3. The mathematical setup of NFS.

Finding two polynomials with a common root modulo a composite N of unknown factorization is difficult in general, and therefore f_0 is usually only of degree one. It follows that $\mathbb{Z}[\alpha_0]$ is in fact a subring of $\mathbb{Q}$. This leads to a commonly used terminology that distinguishes the "rational side" (f_0) and the "algebraic side" (f_1).

The main idea, and the most time-consuming operation in the NFS, consists in finding *relations*. A relation is a pair (a, b) of coprime and preferably small

integers, such that the two principal ideals $\langle a - b\alpha_i \rangle \in \mathbb{Z}[\alpha_i]$ completely factor as a product of prime ideals of small norm contained in some predetermined finite sets $\mathcal{F}_0$ and $\mathcal{F}_1$. (this presentation implicitly assumes that f_1 is monic; see the discussion above for the complications when it is not the case).

The intuitive idea is then to multiply a subset of these relations to obtain a square in $\mathbb{Z}$ on the rational side and a square in $\mathbb{Z}[\alpha_1]$ on the algebraic side. Then, walking down the commutative diagram, we "transfer" these to $\mathbb{Z}_N$ in order to obtain a congruence of squares modulo N — this reveals the factorization of N with probability 1/2.

For a pair (a, b) to yield an actual relation, the above conditions mean that $a - mb$ factors over primes less than a given bound (on the rational side) and that the principal ideal $\langle a - \alpha_1 b \rangle$ factors prime ideals (necessarily of inertial degree 1) of norm less than a given bound (on the algebraic side). As explained above, these ideals are described by pairs (p, r) where $f_1(r) \equiv 0 \bmod p$. Concretely, this means that the integer $F_1(a, b)$ has to split into a set of precomputed primes integers less than the chosen bound — this holds true even when f is not monic.

In the next subsections, we give enough background to control the size of the integers $F_1(a, b)$ that are sieved for smoothness. These integers are often called the "norms".

3.3 Polynomial Selection

For the factorization of RSA-250, the following polynomials were chosen:

$$
\begin{aligned}
f_0 &= 1851129688186382928819 13X - 3256571715934047438664355774734330386901 \\
f_1 &= 86130508464000X^6 - 8158351307642904883773378143837698 4122961112000 \\
&\quad - 66689953322631501408X^5 - 1721614429538740120011760034829385792019395X \\
&\quad - 527332210349663339661 98X^4 - 3113627253613202265126907420550648326X^2 \\
&\quad + 46262124564021437136744523465879X^3
\end{aligned}
$$

It can be observed that the coefficients of f_1 approximately form a geometric progression of reason $s = 354109.861$; s is the *skewness* of the polynomial. If $f = \sum_{i=0}^{d} f_i X^i$, write $\|f\|_1 = \sum |f_i|$ the ℓ_1-*norm* of f (also for multivariate polynomials) and write:

$$
\begin{aligned}
F_1'(X, Y) &= \frac{1}{f_d} F_1(X, Y/s) \\
&\approx X^6 - 2.19X^5Y - 0.0049X^4Y^2 + 12.1X^3Y^3 \\
&\quad - 2.3X^2Y^4 - 3.59XY^5 - 0.48Y^6
\end{aligned}
$$

It follows that if $|a| \leq x$ and $|bs| \leq x$, then $|F_1(a, b)| = f_d |F_1'(a, bs)| \leq f_d x^6 \|F_1'\|_1$. In the case of RSA-250, $\|F_1'\|_1 \approx 21.66$, so we get an upper-bound of $|F_1(a, b)| \leq 2^{50.73} x^6$.

3.4 Relation Collection

The goal of the relation collection step is to collect a large number of (a, b) pairs such that both "norms" $F_0(a, b)$ and $F_1(a, b)$ are sufficiently smooth. Several strategies can be used (and combined) for this purpose, but current implementations of the NFS all rely on a form of *sieving*. The interested reader will find an introduction and many details about sieving in general in [13].

From a high-level perspective, the process works as follows (for side i):

1. Initialize a large array S representing many (a, b) pairs.
2. For all ideals $\mathfrak{p}$ in the *factor base*, *mark* all locations in S where $\mathfrak{p} \mid \langle a - b\alpha_i \rangle$.
3. Discard (as probably non-smooth) all pairs with cofactor larger than a threshold T.
4. Finish factoring the remaining cofactors and check if they are B-smooth.

Concretely, the large array stores the (log of the) "norm" associated to the ideal $\langle a - b\alpha_i \rangle$, which is the result of the evaluation of $F_i(a, b)$. On the rational side, the factor base is composed of all prime integers less than some bound. On the algebraic side, the factor base is composed of all prime ideals (of inertial degree 1) described by pairs (p, r) with p a prime integer less than some other bound and $f_1(r) \equiv 0 \bmod p$. The locations where $\mathfrak{p}$ divides $\langle a - b\alpha_i \rangle$ are those where $a \equiv br \bmod p$. Marking the ideal amounts to subtracting the log of p from the log of $F_1(a, b)$. After all primes have been sieved, the large array contains the log of the norms of the cofactors.

Implementing this procedure requires choosing the range of sieved primes and the threshold T. A simple option consists in sieving all prime (and prime powers) up to B, which makes the fourth step ("cofactorization") trivial. In practice, it is usually more efficient to sieve only a subset of those primes. Cado-NFS sieves all primes up some bound (denoted as $\texttt{lim}_i$ for side i), which is less than the final smoothness bound $2^{\texttt{lpb}_i}$ (`lpb` stands for large prime bound), and uses the elliptic curve method in the cofactorization step. Primes that are in $[\texttt{lim}_i, 2^{\texttt{lpb}_i}]$ are called "large primes".

Implementing this strategy is a matter of trade-offs. If the threshold T is too low, many potential relations will be discarded by the filter (false negatives). If T is too high, many non-smooth numbers will proceed to the cofactorization step (false positives) and increase its cost.

Sieving more primes makes this filter more "precise" (reduce both false rates), but obviously makes sieving more expensive. Choosing these parameters is a non-trivial balancing act.

It is possible to either sieve on both side to identify (a, b) pairs where both norms are smooth (this requires choosing two sets of parameters), or to sieve on one side and process the "surviving" (a, b) pairs using an other strategy for the other side. In particular, because algebraic norms are larger than their rational counterparts, the proportion of pairs where $F_1(a, b)$ is sufficiently smooth may potentially be small. The number of surviving pairs after sieving on the algebraic side may therefore be small enough that checking the smoothness of $F_0(a, b)$ using the product tree algorithm (for the survivors only) gets faster than sieving

again all (a, b) pairs on the rational side. Both strategies were used during the factorization of RSA-250. In the sequel, we focus on sieving and assume that sieving happens on both sides.

Lattice Sieving Cado-NFS uses the technique of *lattice sieving* or *special-*$\mathfrak{q}$ *sieving* [19]. The idea consists in picking a particular prime ideal $\mathfrak{q}$ with description (p, r) —it will be "special"— and restricting our attention to (a, b) pairs such that $\mathfrak{q}$ divides $\langle a - b\alpha_1 \rangle$. We then know in advance that $F_1(a, b)$ is a multiple of p, and this reduces the problem to testing if $F_1(a, b)/p$ is smooth. Because this is a smaller number, it is a bit more likely.

Relation collection can then work by allocating a large array that holds A pairs, filling it with (a, b) such that the special-$\mathfrak{q}$ divides the principal ideal on the algebraic side and detecting all actual relations in the array. The process can then be repeated for all special-$\mathfrak{q}$ ideals of norm inside a specific interval, the "special-$\mathfrak{q}$" range. In the factorization of RSA-250, this was the interval [1G; 12G].

For each special-$\mathfrak{q}$ described by (p, r), the set all of (a, b) pairs such that $a - br \equiv 0 \bmod p$, or, in other terms, such that $a - br = kp$ for $k \in \mathbb{Z}$, forms a Euclidean lattice. More precisely, we have:

$$(a, bs) = (b, k) \begin{pmatrix} r & s \\ p & 0 \end{pmatrix}$$

where s is the skewness of f_1. Let $V = ps$ the module of the determinant of the basis matrix; in this case, it is also the volume of the lattice. This shows that the larger the special-$\mathfrak{q}$, the farthest apart are the corresponding (a, b) pairs.

Using Lagrange reduction, a reduced basis of this lattice can be computed in quadratic time. This yields two integer vectors $\mathbf{u} = (a_0, b_0)$ and $\mathbf{v} = (a_1, b_1)$ such that $\|\mathbf{u}\| \leq \|\mathbf{v}\|$ and $\|\mathbf{u}\| \cdot \|\mathbf{v}\| \leq \sqrt{4/3}V$. A *sieving area* A is chosen depending on the available amount of memory ($A = 2^{33}$ for RSA-250). A total of A pairs (a, b) are sieved, with

$$(a, bs) = i \cdot \mathbf{u} + j \cdot \mathbf{v} \qquad \text{where} \qquad |i| \leq \frac{I}{2}, \quad 0 \leq j \leq J \quad \text{and} \quad IJ = A.$$

Choice of Sieve Regions The bounds I and J can be chosen so as to minimize the maximum norm of F_1 over the corresponding parallelogram. It follows from the properties of Lagrange-reduction that $\|\mathbf{u}\|^2 \leq \sqrt{4/3}V$, and therefore both a_0 and b_0 are less than $B := (4/3)^{1/4}\sqrt{V}$ in absolute value. Next, write $t = B/\max(|a_1|, |b_1|)$ so that

$$|a_1| \cdot t \leq B \qquad \text{and} \qquad |b_1| \cdot t \leq B.$$

t is a measure of how "thin" the sieved parallelogram is, because $t = \Theta\left(\sqrt{\|\mathbf{u}\|/\|\mathbf{v}\|}\right)$. In general, the ratio $\|\mathbf{u}\|/\|\mathbf{v}\|$ could be as low as $1/V$, with $\mathbf{u} = (1, 0)$ and

$\mathbf{v} = (0, V)$. However, this is quite unlikely if the input basis is "sufficiently random". Let $\mathbf{v}^\star = \mathbf{v} - \mu\mathbf{u}$ with $\mu = \mathbf{uv}/\|\mathbf{u}\|^2$; $\mathbf{v}^\star$ is the orthogonal projection of $\mathbf{v}$ onto the orthogonal of $\mathbf{u}$. Define γ as $\|\mathbf{u}\|/\|\mathbf{v}^\star\|$. It is shown in [20] that $\Pr[\gamma \leq x]$ tends to $3x/\pi$ for "random" inputs. Because $\|\mathbf{v}^\star\| \leq \|\mathbf{v}|$, it follows that $|\mathbf{u}\|/\|\mathbf{v}\| \leq \gamma$. As a consequence, the probability to obtain a seriously unbalanced basis is small.

In any case, very thin sieve regions are unfavorable for a variety of reasons, and special-$\mathfrak{q}$'s are discarded if t is too small (say less than 1/100). It follows that, concretely, t is lower-bounded by some constant.

Finally, observe that

$$|a| = |ia_0 + ja_1| \leq \frac{I}{2}a_0 + Ja_1 \leq B\left(\frac{I}{2} + J/t\right),$$
$$|bs| = |ib_0 + jb_1| \leq \frac{I}{2}b_0 + Jb_1 \leq B\left(\frac{I}{2} + J/t\right).$$

It remains to choose I and J such that $I/2 + J/t$ is minimal under the constraint that $IJ = A$. It is not difficult to see that the optimal choice is $I = \sqrt{2A/t}$. In that case we have that both $|a|$ and $|bs|$ are less than $2\sqrt{Aps/(t\sqrt{3})}$. This choice of I and J thus guarantees that $|F_1(a, b)|$ is less than $f_d 2^6 A^3 p^3 s^3 t^{-3} \sqrt{3}^{-3} \|F_1'\|_1$ over the sieved region.

Plugging in the numerical constants of the RSA-250 factorization yields an upper-bound of about $2^{208}p^3$, assuming $t \approx 1$. Example of sieved zones are shown in fig. 4.

Random (a, b) Pairs Processed by Cado-NFS For our purpose, we need to observe statistical properties of the norms of "random" (a, b) pairs processed during a factorization. Assuming that $F_1(a, b)$ behaves like a random number is not reasonable, if only because $F_1(a, b) \equiv 0 \bmod 6$ in RSA-250 for the chosen polynomial.

However, it is possible to accumulate a realistic sample using the following simple-minded procedure:

1. Pick a random special-$\mathfrak{q}$ (in the full range used in the factorization)
2. Compute the sieve region as discussed above.
3. Pick a random (a, b) pair that would have been sieved with this special-$\mathfrak{q}$
4. Compute the "norms", *i.e.* $F_0(a, b)$ and $F_1(a, b)$.

We accumulated a bit more than 100M samples using this procedure, mostly to observe empirically the smoothness properties of the norms "under real-life conditions".

4 Relation Collection in Cado-NFS

In this section, we give a succinct description of the algorithm implemented in Cado-NFS. Its goal is to find pairs (a, b) such that both $F_0(a, b)$ and $F_1(a, b)$ are

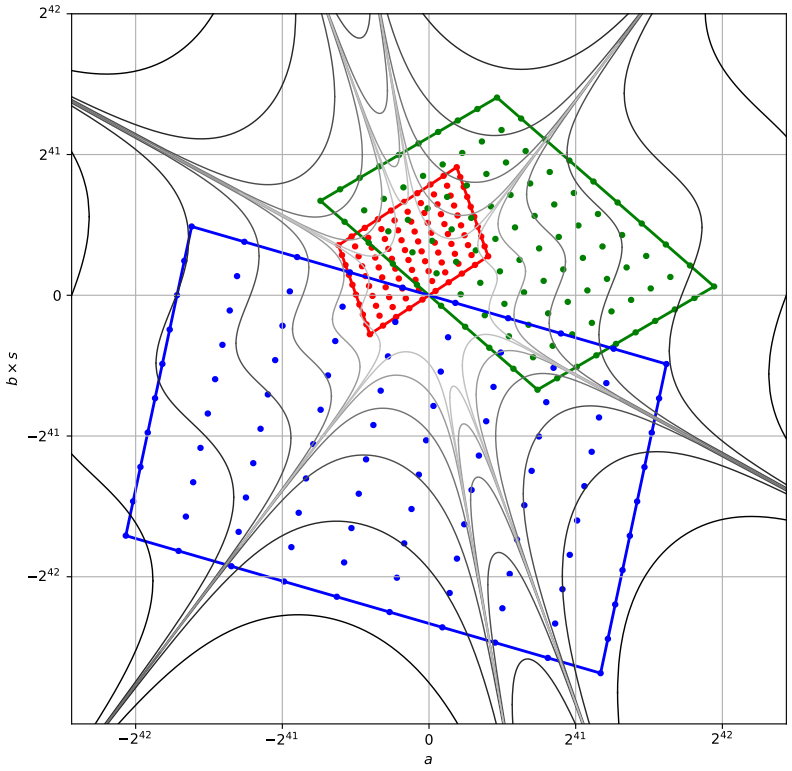

Fig. 4. Sieved regions for $q = 1G$ (red), $q = 4G$ (green), $q = 12G$ (blue). The level sets of $|F_1(a, bs)|$ are shown in black. From outer to inner, they correspond to the following sizes (in bits): 300, 294, 288, 282, 274 and 268.)

sufficiently smooth. The smoothness bound on the rational side is $2^{\texttt{lpb}_0}$, and it is $2^{\texttt{lpb}_1}$ on the algebraic side (`lpb` stands for "large prime bound"). In RSA-250, $\texttt{lpb}_0 = 36$ and $\texttt{lpb}_1 = 37$.

The global process follows the outline given in section 3.4 and uses special-q sieving. Given a large collection of pairs (a, b), the *sieve* finds "small" prime divisors of $F_i(a, b)$ up to some bounds. In Cado-NFS, the smoothness bounds used by the sieve are called $\texttt{lim}_0$ (on the rational side) and $\texttt{lim}_1$ (on the algebraic side). In the factorization of RSA-250, their values are: $\texttt{lim}_0 = \texttt{lim}_1 = 2^{31}$.

Factoring with a sieve is akin to finding primes with the sieve of Eratosthenes. It is a very efficient method and works best on large sets of numbers. However, sieving requires the target set of integers to have a specific structure. In particular, given a polynomial $S \in \mathbb{Z}[X]$, it is easy to sieve $S(0), S(1), S(2), \ldots$. Indeed, if $S(n) \equiv 0 \bmod p$, then $r \leftarrow n \bmod p$ is a root of S modulo p. It follows that $S(r + kp)$ is always a multiple of p. Finding all the $S(i)$ that are divisible by p requires repeating this procedure with all roots of S modulo p. This generalizes without problem to multivariate polynomials.

Sieving can be seen as a form of sorting: for each prime p, "emit" pairs $(r + kp, p)$ [a pair (n, p) means that p divides n]; sort these pairs according to their first coordinate [this collects together all $(n, \ldots)$ pairs]; scan the list of sorted pairs, and read in $(n, \ldots)$ all the prime factors of consecutive integers n. This idea is at the heart of "bucket sieving" [1].

Sieving is much faster than the batch smooth part algorithm of [4]. For instance, in the factorization of RSA-250, on the rational side, the product of all primes less than 2^{31} has size 3.1Gbit. Figure 1 shows that processing a batch of $\approx$ 5M numbers of about 300 bits takes 271s. Therefore processing 2^{33} of them using the batch smooth part algorithm would require about 125 hours. On the other hand, sieving completes the process in about 216s.

However, sieving only applies to structured sets of integers. The strength of the batch smooth part algorithm is that it applies to any set of integers.

4.1 Surviving Pairs

In Cado-NFS, the step after sieving is called "cofactorization". It attempts to find "large" prime divisors above $\texttt{lim}_i$ and below $2^{\texttt{lpb}_i}$. As this part is sequential and costly, only promising pairs are processed. There are called *survivors*. What makes a pair a survivor is the size of the *cofactor* (the non-factored part) of $F_i(a, b)$ after the sieve has found all prime divisors less than $\texttt{lim}_i$. A pair (a, b) "survives" if the size the cofactors on both sides are less than $2^{\texttt{mfb}_i}$. As discussed above, choosing $\texttt{lim}_i$ and $\texttt{mfb}_i$ is a matter of compromises. The goal of the sieve is to discard a huge proportion of (a, b) pairs without removing too many would-be relations.

Sieving is thus done as follows :

1. **Sieve** norms on both algebraic and rational sides.
2. Keep only promising pairs (= survivors).
3. Find tiny factors with trial division on both sides.

4. Filter survivors once more.
5. Send them to cofactoring.

Product tree factoring is preferred on the rational side for large special-qs. Indeed, as opposed to sieving, product tree (batch) factoring doesn't need its inputs to follow any sequence and can thus be used directly on surviving pairs in any order. This makes it possible to filter promising pairs right after sieving on the algebraic side, effectively swapping sieving the entire rational side with batch factoring a small subset of these pairs. A second filter is then applied after batch factoring. Sieving is thus done as follows :

1. **Sieve** norms on the algebraic side.
2. Keep only promising pairs (= survivors).
3. Find tiny factors with trial division in survivors on the algebraic side.
4. Filter survivors once more.
5. **Batch factor** survivors on the rational side.
6. Filter survivors once more.
7. Send them to cofactoring.

4.2 Different Sieves

Inside Cado-NFS sieving, different techniques are used depending on the side and targeted ideals. An overview of these are shown on table 1.

Small primes, that is primes in $[2, 2^I]$, are sieved using the *small sieve* (the parameter I is the same as the sieving region bound introduced in 3.4). The small sieve is done in two passes. The first one is approximate, a base L is first picked in order for all values of $log_L \text{norm}_i(a, b)$ for all pairs (a, b) to fit in $[0, 255]$. Logs of norms are then rounded to the nearest integer so they fit in an 8-bit integer. This approximation is then sieved by primes from the factor base. Newly found factors are not written next to norms they divide — or their logarithmic approximation —. Instead, every small prime is reduced through log_L to an 8-bit integer as well. This integer is then subtracted to all norms they are a factor of. The approximate information gathered on the first pass is enough to pick survivors. Following this, a second pass, lossless, is done on all small primes but tiny ones. Primes sieved are registered only as factors if ticking a survivor. Tiny factors are found after all sieving is done with trial division.

Primes above 2^I are sieved using a bucket sieving technique, similar to a bucket sort, with one pass for medium-sized (below `bkthresh1`) and two for larger ones. As opposed to the small sieve, this is lossless.

Factor base range	$[2, 2^I]$	$[2^I, \texttt{bkthresh1}]$	$[\texttt{bkthresh1}, \texttt{lim}]$
Sieving algorithm	Small sieve	1-level bucket sieve	2-level bucket sieve

Table 1. Repartition of sieving algorithms

5 Relations Collected During the Factorisation of RSA-250

During the factorization of RSA-250, 8.4G relations were produced in total and were kept after the factorization completed. We were kindly provided access to this dataset, hereafter denoted as "RSA-250 relations".

The relations are stored in gzipped text files totaling 786GB (1.5TB uncompressed), in a straightforward format. Analyzing this dataset enables us to simulate various algorithmic strategies and choose parameters for the proposed improvement without having to run costly exploratory experiments. This process also uncovered minor discrepancies between the actual data and what is announced on the web page of the record[4].

This section describes simple statistics about the relations. As a foreword, parsing 780GB of compressed text data is an interesting "big-data" engineering problem. The relations are spread over many gzipped files. While we could have used off-the-shelf MapReduce-style solutions [11] such as Apache Hadoop or Apache Spark [21] for this purpose, we found it easier to write a collection of multi-threaded C programs that parse the relations and accumulate various statistics. A single file is processed by a single thread — this is imposed by the sequential nature of access to the content gzipped files. This programs reads from the network file system at 370MB/s and parses about 3.8M relations per second using 32 cores. The whole dataset is then processed in about 40 minutes.

The special-q range is split in two: the "small" ones (less than 4G) and the "large" ones (greater than 4G), with different algorithmic strategies and different parameters on both ranges. Table 2 shows basic information about the collected relations.

	small	large
special-q range	[1G; 4G)	[4G; 12G)
# relations	3.9G	4.5G
Algorithm for the rational side	Sieve	Product tree
Avg. Rational norms (bits), stdev	151.8 ± 2.0	152.6 ± 2.0
Algebraic norms (bits)	283 ± 8.6	288 ± 8.4
$\texttt{mfb}_1$	111	74

Table 2. Basic statistics about the RSA-250 relations.

The relation-collection process is controlled by several parameters described in section 4. The most important for our purposes are `lim` (largest sieved prime), `lpb` (large prime bound — size of the largest prime allowed in a relation) and `mfb` (size of the largest residue after sieving). These parameters usually have different values on both sides, and `mfb` changes over the special-q range. These values can be found on the web page of the record and are summarized in table 3.

[4] https://gitlab.inria.fr/cado-nfs/records/-/tree/master/rsa250

The choices of `lpb` and `mfb` allow for two large primes on the rational side and three large primes on the algebraic side (this may include the special-`q`).

Side	0 (rational)	1 (algebraic)
`lim`	2^{31}	2^{31}
`lpb`	36	37
`mfb`	72	111 (small `q`) or 74 (large `q`)

Table 3. Parameters used to collect the RSA-250 relations.

It follows that in a collected relation:

- The rational norm (≈ 152 bits) is 2^{36}-smooth, and contains a relatively large part of size $\approx 152 - 72 = 80$ bits which is 2^{31}-smooth.
- The algebraic norm (≈ 285 bits) is 2^{37}-smooth. Over the large special-`q` range, it contains a relatively large part of size at least $285 - 32 - 74 = 179$ bits which is 2^{31}-smooth.

This is another indication that the norms in the collected relations are *not* uniformly random B-smooth integers. It must be noted in addition that in the "small" special-`q` range, half of the special-`q` fall into the range of sieved primes.

5.1 Frequency of Primes

While this is not directly related to our primary objective (improving sieving strategies), we take the opportunity to discuss an interesting phenomenon about the collected relations. Earlier works about structured Gaussian elimination applied to integer factoring (including notably [2,7]), assume that the prime factor p occur with probability $1/p$ in the collected relations, as it would in random integers.

The distribution of primes in the RSA-250 relations can be observed in Fig. 5. It is clearly visible that p occurs with frequency $1/p^\alpha$ with $\alpha < 1$. This phenomenon is not surprising; intuitively, smooth numbers should have more small factors than random integers. This can be quantified as follows. Let $\Psi(x, y)$ denote the number of y-smooth integers less than x. Take a y-smooth integers less than x which is a multiple of p; divide it by p; this yields a y-smooth number less than x/p (this is in fact a bijection). It follows that the number of such integers is exactly $\Psi(x/p, y)$.

The probability that a random y-smooth number less than x is divisible by p is therefore $\Psi(x/p, y)/\Psi(x, y)$. Tenenbaum and Hildebrand have shown in 1986 [12, theorem 3] that $\Psi(cx, y)/\Psi(x, y) \approx c^\alpha$ with $c \leq y$ and α is the unique positive solution of

$$\sum_{p \leq y} \frac{\log p}{p^\alpha - 1} = \log x.$$

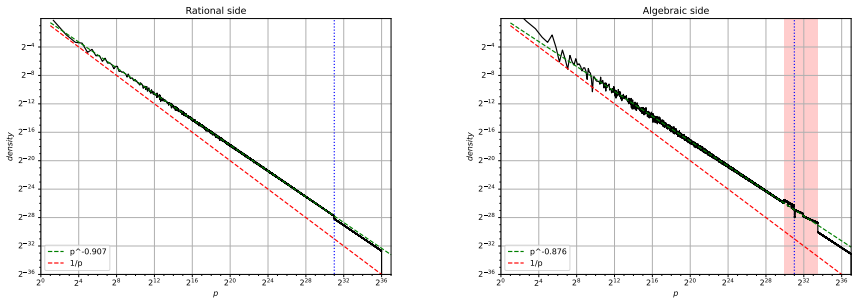

Fig. 5. Density of primes in the RSA-250 relations (log-log scale). The vertical dotted blue line marks the largest sieved prime. The special-$\mathfrak{q}$ range [1G; 12G) appears in pink.

It follows that:

$$\frac{\Psi(x/p, y)}{\Psi(x, y)} \approx \frac{1}{p^\alpha}.$$

Because of the "classical" Mertens-like asymptotic evaluation $\sum_{p \leq x} (\log p)/(p-1) = \log x - \gamma + o(1)$ when $x \to +\infty$, the value of α is necessarily less than 1 ($\gamma = 0.56...$ denotes the Euler-Mascheroni constant).

It is shown in [12] that $\alpha = \frac{\log(1+y/\log x)}{\log y}\left(1 + \mathcal{O}\left(\frac{\log\log y}{\log y}\right)\right)$. This is quite consistent with Fig. 5. When y is large compared to $\log x$, this can be simplified to $\alpha \approx 1 - \log u/(\log y)$, where u is defined as usual as $u := \log x/\log y$, or even to $\alpha \approx 1 - (\log\log x)/(\log y)$. See [10] for more details.

Beyond statistics properties of smooth numbers, the particular algorithm used to collect relations also has visible effects: sieved primes appear more frequently than large primes (slight drop on the right of the dashed vertical blue line). Special-$\mathfrak{q}$'s appear with a frequency boost because lattice sieving favors relations that contain them. This highlights again that the RSA-250 relations are not uniformly random smooth numbers.

5.2 Yield per special-$\mathfrak{q}$

In this section, we turn our attention to the number of relations found per special-$\mathfrak{q}$. A faster sieving procedure that finds less relations will need to examine more special-$\mathfrak{q}$'s. Our goal here is to provide a quantitative model.

Table 2 shows that the yield is higher for small special-$\mathfrak{q}$'s: the average "density" of relations is 1.3 for small special-$\mathfrak{q}$ *versus* 0.56 for large special-$\mathfrak{q}$ — this means that over a range $[a, b)$ of small special-$\mathfrak{q}$'s, the average number of relations found is $1.3(b-a)$. Note that this numbers do not account for duplicate relations; the proportion of duplicates is expected to be higher for small special-$\mathfrak{q}$'s. In total, 70% of these relations are unique.

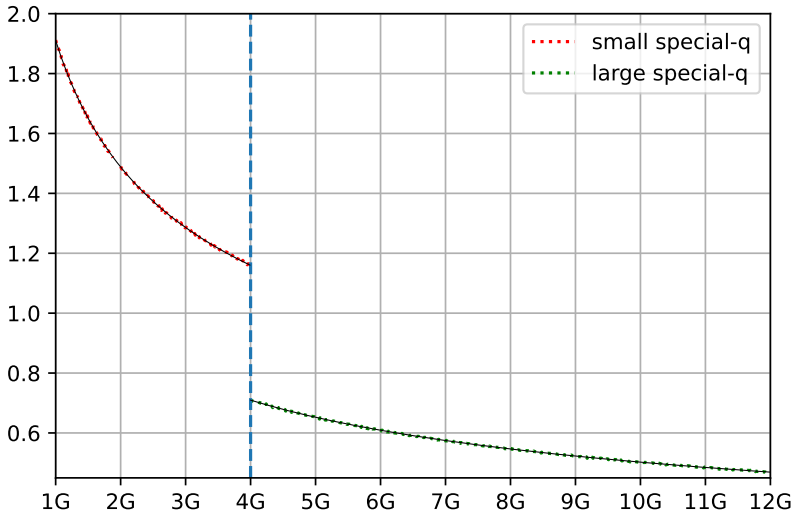

Fig. 6. Density of relations per special-$\mathfrak{q}$.

The sieving parameters are different for small and large special-$\mathfrak{q}$'s, and this is sufficient to explains a large drop in yield at the transition between the two ranges. However, the number of relations found per special-$\mathfrak{q}$ decreases with q. Fig. 6 shows this well-known phenomenon more precisely. Two causes may account for this.

Firstly, because a special-$\mathfrak{q}$ contains a prime integer, their density is about $1/\log q$ thanks to the prime number theorem. This means that the density of primes is about 4.5% at for $q = 4$G, and it drops to 4.30% at the end of the special-$\mathfrak{q}$ range ($Q = 12$G). Even if the number of relations found per special-$\mathfrak{q}$ were strictly constant, the rarefaction of primes would accounts for a 5% drop in the "density" of relations over the large special-$\mathfrak{q}$ range.

Secondly, as argued in section 3.4, the algebraic norms are less than $\approx 2^{208}p^3$ in the sieved zones, however since the special-$\mathfrak{q}$ can be taken out, we are left with numbers of size less than $\approx 2^{208}p^2$. It seems plausible that the number of relations collected per special-$\mathfrak{q}$ is correlated with the probability that integers of this size are 2^{37}-smooth.

The proportion of y-smooth integers between x and $x + dx$ is given by

$$\frac{\Psi(x + dx, y) - \Psi(x, y)}{dx} = \rho(u) + \mathcal{O}\left(\frac{1}{\log x}\right)$$

where $u = (\log x)/(\log y)$ and ρ is the Dickman function — see [15] for details. Let $R(q)$ denote the "density" of relations at this value of q (this is what Fig. 6

shows). Our initial assumption was that $R(q)$ would be correlated with $f(q) = 1/\log(q) \cdot \rho((208+2\log_2 q)/37)$. One standard way of visually asserting the quality of this model is to plot the "residues" $R(q)/f(q)$. Fig. 7 (left) shows that $R(q)$ decreases faster than this simple model predicts.

However, tinkering a bit shows that a slightly more general model is sufficient. Define $f_\alpha(q) = 1/\log(q) \cdot \rho((208 + \alpha \log_2 q)/37)$; then over the small special-$\mathfrak{q}$ range, $R(q)$ is about 297M times $f_{2.419245}(q)$ whereas over the large special-$\mathfrak{q}$ it is about 255M times $f_{2.53737}(q)$. Fig. 7 (right) shows that the residues are flat. The values of α given above have been found by dichotomy search with the objective of minimizing the absolute value of the slope of a linear regression performed on $R(q)/f_\alpha(q)$. This model seems empirically good; however we have no explanation as to why values of α greater than 2 (and less than 3) provide a better fit.

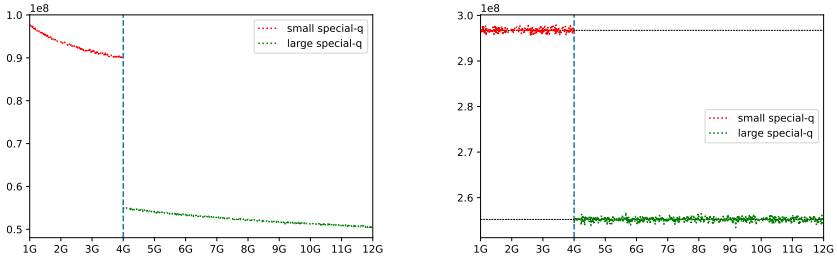

Fig. 7. Residues of potential models of the yield per special-$\mathfrak{q}$. Left: $R(q)/f_2(q)$. Right $R(q)/f_{2.419245}(q)$ (small) and $R(q)/f_{2.53737}(q)$ (large).

In any case, this yields a good model of the yield per special-$\mathfrak{q}$ (it is shown in black in Fig. 6). The multiplicative constants (297M and 255M) accounts for the sieving area, the fact that pairs where a and b are not coprime are discarded, the fact that there is a large 2^{31}-smooth part, etc. With $y = 2^{37}$ and p in $[10^9; 10^{1000}]$, we have $7 \leq u \leq 8$, where u is again $\log p/\log y$. In this range, $\rho(u)$ can be approximated to good accuracy by a power series. Set $x \leftarrow 2u - 15$ and:

$$10^7\rho(u) \approx 1.7178674920 - 2.8335703447x + 2.3000577581x^2 - 1.2233613236x^3 + 0.47877283118x^4 - 0.14657423350x^5 + 0.036375058464x^6 + \ldots$$

If one is willing to commit to these estimates, then they can be used to predict the number of relations that would have been obtained over a larger range of special-$\mathfrak{q}$'s:

$$\#\text{relations} = 3.9G + \int_{4G}^{q_{\max}} f_{2.53737}(q)dq \tag{1}$$

This can easily be estimated using a generic numerical integration algorithm. Over the range of small special-$\mathfrak{q}$, this model overestimates the number of relations found by 8%. However, the quality of the fit over the large special-$\mathfrak{q}$ range

is amazingly good: (1) predicts the actual number of collected relations when $4\text{G} \leq q \leq 12\text{G}$ with relative error 0.027%.

This has interesting consequences. For instance, assume that an alternative relation collection procedure runs faster but produces (say) only 50% of the relations compared to the current one; further assume that this proportion remain constant for all special-$\mathfrak{q}$'s. The above reasoning enables us to venture the following prediction: recovering as many relations will require sieving special-$\mathfrak{q}$'s up to $\approx 34.5\text{G}$. Taking into account the density of primes, the new procedure has to sieve about $2.75\times$ more special-$\mathfrak{q}$'s. As such, it breaks even if it is at least $2.75\times$ faster than the original. In the same way, a procedure that keeps 85% of the relations will need to sieve almost 3.3G extra special-$\mathfrak{q}$'s, i.e. do $\approx 26\%$ extra work.

6 Combining Sieving and the Batch Smooth Part Algorithm on the Same Side

During the factorization of RSA-250, sieving and the batch smooth part algorithm described in section 2 were both used — on different sides. In this section, we explore the possibility of using both on the same side. To the best of out knowledge, this has not been tested in practice.

This idea is however not new. In [4], Bernstein suggested the following strategy:

> Sieve all primes p up through, say, B; throw away the n's whose unfactored part is uncomfortably large; then apply some other method to the n's that remain.

"Some other method" could reasonably be understood as the batch smooth part algorithm and presented in the very same article. The suggestion here is to sieve *only small primes*. A potential advantage is that it would enable the use of a smaller sieve area, thus reducing memory consumption, and/or improving memory locality.

An other option would be to *skip small primes entirely*. Again this is not a new idea; it seems that it was already part of the folklore in 1993:

> Furthermore, one often does not sieve with the small primes below a certain small prime bound [...] [16]

Indeed, small primes have more "hits" and require more time than large ones.

Prime ideals below 2^I — small sieved ones — are of particular interest to speed up the sieving process. Indeed, our intuition is that comparatively more time is spent on small primes to retrieve fewer factor bits than sieving larger primes. In addition, finding the remaining small factors after sieving larger ones would be efficient as their size would be small. That is to say, sieving is more efficient on large primes while batch factoring is on small ones as it benefits from a smaller factor base.

To discuss these two ideas, we split sieved primes (below $\texttt{lim}_i$) in two subsets: extra-small primes that are below a certain bound 2^B and medium primes that are above.

1. **Sieve** using either extra-small or medium primes on both sides
2. Keep only promising pairs (= survivors) that have small cofactors enough
3. Use the **Batch smooth part** algorithm on survivors to find extra-small factors on side 1
4. Filter survivors once more on side 1 (discard those with cofactor size greater than $\texttt{mfb}_1$ bits)
5. Use the **Batch smooth part** algorithm again on survivors to find extra-small factors on side 0
6. Filter survivors again on side 0 (discard those with cofactor size greater than $\texttt{mfb}_0$ bits)
7. Send survivors to the cofactoring step

We introduce new parameters for intermediate bounds, `mfbb0` and `mfbb1`. After sieving one of the subsets of primes on both sides, a pair (a, b) is considered a survivor in step 2 if the cofactor on side i has less than $\texttt{mfbb}_i$ bits.

6.1 Sieving Only Extra-Small Primes

In this section, we try to evaluate the idea of sieving only extra-small primes. This enables the use of a smaller sieve area, which may have some benefits.

We now show that using a "large-ish" sieve region of size $A = 2^{33}$ as was done in the factorization of RSA-250 is indeed not a good strategy. Sieving primes up to 2^{31} with sieve area 2^{33} requires 213 seconds using a single-thread. Extracting the 2^{31}-smooth part of a single batch of size ≈ 1.5Gbit requires 270s. This makes about 15M cofactors of size, say, 100 bits. Therefore, rejecting pairs with "uncomfortably large" cofactors should only allow a fraction 0.1746% of all pairs to survive, otherwise applying the batch smooth part algorithm will be slower than sieving altogether. Note that this is a very conservative stance: even a lower proportion could make the combination of sieving + batch smoothness detection slower than just sieving.

For each value of the bound B that delimits the extra-small primes from the medium ones, we exhaustively try all pairs of thresholds $\texttt{mfbb}_0$ and $\texttt{mfbb}_1$. For each combination of $(B, \texttt{mfbb}_0, \texttt{mfbb}_1)$, we estimate the proportion of pairs that would survive until after step 2 of the procedure above and enter the batch smooth part algorithm (this is done using our collection of "random" pairs from section 3.4). If this proportion is larger than the threshold given above, we reject the set of parameters. Among all valid parameters set, we find the one that preserves the maximum number of relations found during the factorization of RSA-250 (this is estimated thanks to the RSA-250 dataset). The result is shown in Table 4.

From there, we are tempted to conclude that the procedure will not be practical. Unless 2^B gets quite close to the actual limit of 2^{31}, the test in step 2 is

B	mfbb$_0$	mfbb$_1$	% surviving rel.	% surviving pairs
8	123	266	0.5	0.17
9	121	258	0.8	0.17
10	119	253	1.1	0.17
11	115	255	1.6	0.17
12	113	250	2.2	0.17
13	114	239	2.9	0.17
14	112	236	4.0	0.17
15	113	228	5.2	0.17
16	115	220	6.9	0.17
17	118	212	9.2	0.17
18	117	209	12.3	0.17
19	115	207	15.8	0.17
20	113	205	19.1	0.17
21	99	216	25.9	0.17
22	96	215	33.5	0.17
23	96	210	39.8	0.17
24	100	201	48.5	0.17
25	101	195	61.8	0.17
26	99	193	72.3	0.17
27	99	189	76.7	0.17
28	123	168	82.5	0.17
29	103	177	92.6	0.17
30	103	172	98.4	0.17
31	99	155	100.0	0.11

Table 4. "Optimal" parameters for sieving only extra-small primes with a sieve area of 2^{33}. The fourth column shows the proportion of RSA-250 relations that would survive the process. The fifth column shows the proportion of pairs that survive step 2.

not precise enough to simultaneously keep the actual relations and discard sufficiently many unpromising (a, b) pairs. Therefore, restricting sieving to a small set of extra-small primes seems bound to discard a significant fraction of potential relations. In turn, as discussed in section 5.2, it seems difficult to be able to break even with pure sieving.

This does not mean that the overall strategy is doomed. In particular, the above reasoning does not rule out the possibility that it could be competitive when used with a *smaller* sieve area. Here, the argument was that if the sieve area is large, then the proportion of pairs that survive the test must be small otherwise their number will overwhelm the batch smoothness detection algorithm. Reducing the sieve area will reduce the number of surviving pairs, and may alleviate this problem.

However, we believe that we have shown that this specific strategy *requires* very different parameters than those used in the factorization of RSA-250. This, in turn, is not surprising.

6.2 The Other Way Around: Do Not Sieve Extra-Small Primes

We now turn our attention to the opposite strategy: sieving only the medium primes, on both sides. Extra-small ones are found in a second step using the batch smooth part algorithm. The intuition is that step 1 is the most efficient part of sieving. It gives us a lot of information to decide if a pair is in a good position to be a relation while being very fast. It allows us to remove a good amount of pairs in step 2 before finding small factors, using the batch smooth part algorithm, in steps 3 and 5.

This is bound to be less precise compared to "pure" sieving as it starts filtering pairs earlier, after fewer prime factors have been treated. Ideally, this loss of precision would not lead to too many bad survivors (false positives) nor lost relations (false negatives). The challenge is to find parameters that would make the alternative faster while limiting the loss of relations found by the original implementation of Cado-NFS.

We essentially had a dual approach: we imposed a proportion of RSA-250 relations that must survive the whole procedure. For each value of the bound B that delimits extra-small primes (not sieved) and medium primes, we exhaustively try all pairs of thresholds $\texttt{mfbb}_0$ and $\texttt{mfbb}_1$. For each combination of $(B, \texttt{mfbb}_0, \texttt{mfbb}_1)$, we estimate the fraction of RSA-250 relations that would be found. If this proportion is too low, we discard the parameters. Note that if $(\texttt{mfbb}_0, \texttt{mfbb}_1)$ is valid, then so does $(\texttt{mfbb}_0 + u, \texttt{mfbb}_1 + v)$. Among all valid parameter sets, we single out the ones that minimize the proportion of surviving pairs at the end of step 2. The results can be seen in Fig. 8.

In light of the discussion in section 5.2, we focus on high proportions of preserved RSA-250 relations. In addition, one particularly relevant choice of B is $B = 17$, as it means that the "small sieve" can be completely disabled (only bucket sieving remains). Reasonable parameters set are shown in table 5.

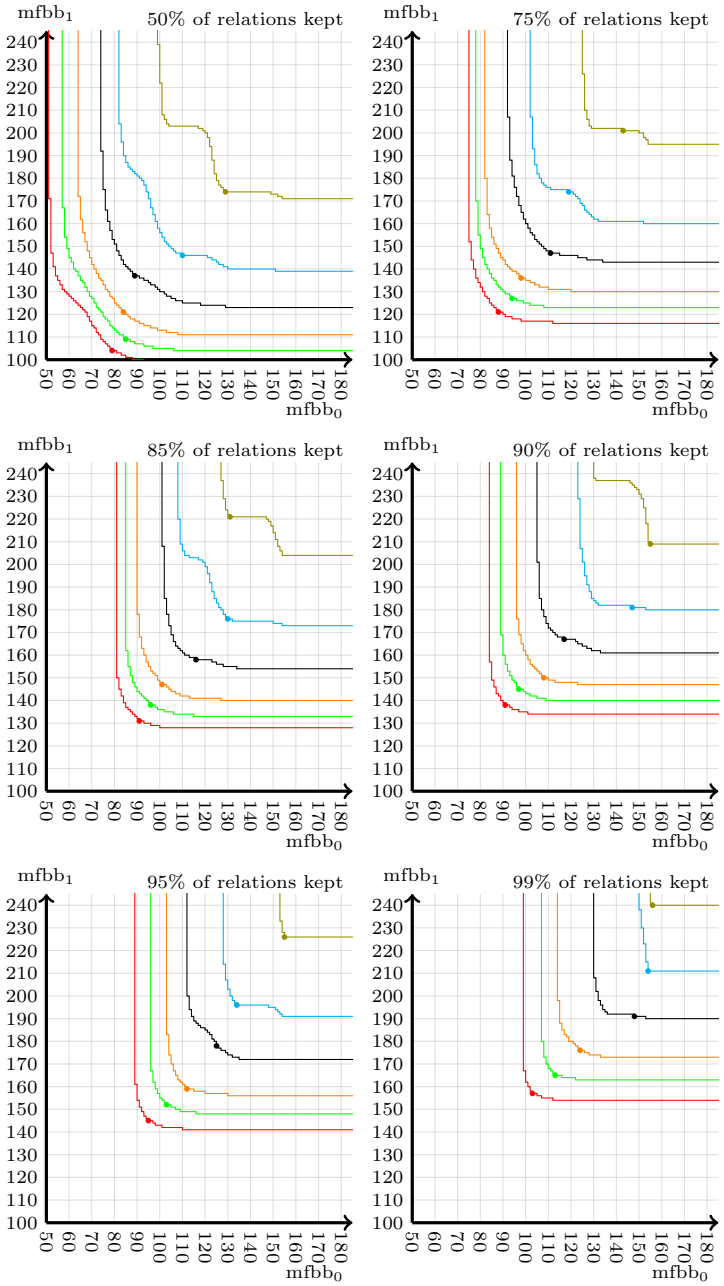

Fig. 8. Possible choices of $\texttt{mfbb}_0$ and $\texttt{mfbb}_1$ that preserve the given fraction of RSA-250 relations. Sieving all primes between 2^i and 2^{31} with $i = 10$ (red), $i = 12$ (green), $i = 14$ (orange), $i = 17$ (black), $i = 20$ (cyan) and $i = 24$ (olive). The big dot shows the values that minimize the proportion of pairs that survive step 2.

$\texttt{mfbb}_0$	$\texttt{mfbb}_1$	% surviving rel.	$\log_{10}$ proportion surviving pairs
93	135	50.3	-6.165
115	146	75.1	-5.063
116	158	85.0	-4.556
117	167	90.0	-4.233
125	178	95.0	-3.568
136	192	99.0	-2.788

Table 5. Reasonable parameters for sieving only primes between $2^B = 2^{17}$ and $\texttt{lim}_i = 2^{31}$ with a sieve area of 2^{33}. The third column shows the proportion of RSA-250 relations that would survive the process. The fourth column shows the proportion of pairs that survive step 2 (the column shows x, the actual proportion is 10^x).

7 Experiments and Practical Results

We implemented the strategy discussed in section 6.2 inside Cado-NFS. This has been a non-trivial programming effort, as this required modifying a complex program made of 29K lines of C++ code spread over nearly 50 files.

The modifications we performed can be summarized as follows:

- We altered the piece of code that holds the factor base for sieving to remove primes less than 2^B, where B is a new command-line parameter. We also inserted there a precomputation of the product of extra-small prime factors.
- The sieve area is divided into "sieve regions" of size about 64KB. We altered the piece of code that fully processes a sieve region. The original searches "survivors" (cofactor of less than $\texttt{mfb}_i$ bits) after sieving and launches an asynchronous cofactorization task for each surviving pair. We inserted a modified survivor detection procedure using the new $\texttt{mfbb}_i$ parameters; the extra-small primes factors of survivors are recovered by the batch smooth part algorithm; then pairs are fed back to the preexisting mechanism.

Our implementation is not extraordinarily well optimised. In particular, because we operate inside a single sieve region, we may or may not not have enough survivors to completely fill one batch. Regardless of its performance, the sheer availability of our implementation enabled us to empirically validate the results of the simulations presented in Fig. 8 and in Table 5.

Actual performance results are shown in Fig. 9 and 10. With $B = 17$ (*i.e.* disabling the small sieve entirely), our code manages to get faster than the original Cado-NFS implementation, while targeting 85% or 90% of the RSA-250 relations. figures strongly suggest that $B = 17$ is the optimal choice.

Tables 6 and 7 show the result of slightly longer experiments. We processed several special-$\texttt{q}$ near the beginning and the two-thirds of the whole range. The "local speedup" is the ratio between the speed-up and the proportion of relations found (it is equal to one if the performance of the new code is strictly proportional to the original).

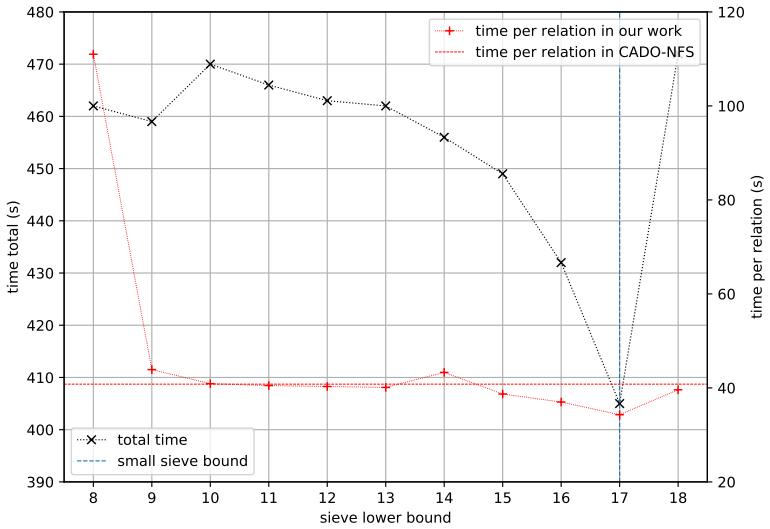

Fig. 9. Results of processing a large special-q with different bounds with optimal parameters targeting 85% of the RSA-250 relations

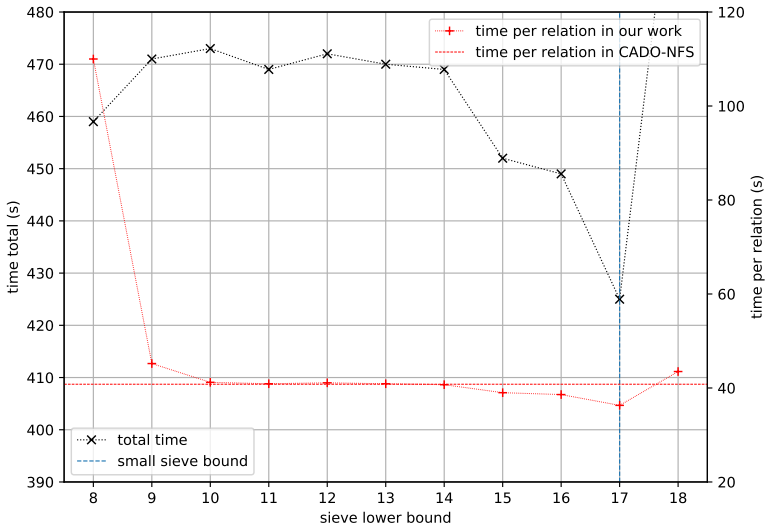

Fig. 10. Results of processing a large special-q with different bounds with optimal parameters targeting 90% of the RSA-250 relations

The observed speed-up may seem modest; we nevertheless consider it a significant achievement in view of the fact that Cado-NFS is a complex, highly optimized piece of code that holds the current computational records.

B	$\texttt{mfbb}_0$	$\texttt{mfbb}_1$	# relations found	× original	Time (s)	× original	local speed-up
Original Cado-NFS			390	-	8619	-	-
17	89	137	232	0.59	6589	0.76	0.78
17	108	143	328	0.84	6691	0.78	1.08
17	111	147	347	0.89	6940	0.81	1.10
17	114	152	361	0.93	7292	0.85	1.09
17	116	158	367	0.94	7450	0.86	1.09
17	117	167	371	0.95	8088	0.94	1.01

Table 6. Results when processing 34 special-$\mathfrak{q}$'s in the range $[8204724066, 8204725068]$.

B	$\texttt{mfbb}_0$	$\texttt{mfbb}_1$	# relations found	× original	Time (s)	× original	local speed-up
Original Cado-NFS			674	-	6942	-	-
17	114	152	519	0.77	5242	0.76	1.02
17	116	158	561	0.83	5442	0.78	1.06
17	117	167	606	0.90	5684	0.82	1.10
17	125	178	646	0.96	7558	1.01	0.88
17	148	191	667	0.99	20077	2.89	0.34

Table 7. Results when processing 25 special-$\mathfrak{q}$'s in the range $[2500000000, 2500000500]$.

Tables 6 and 7 hints at the possibility of collecting 90% of the relations found by the "original" Cado-NFS, in roughly 82% of the time. Assume that this is feasible over the whole special-$\mathfrak{q}$range. According to the estimate given in section 5.2, such a procedure has to sieve 16% more special-$\mathfrak{q}$than the original. Under these assumptions, the new relation collection procedure would yield a 5% speedup over the original Cado-NFS.

8 Conclusion

This work introduces an alternative sieving method in Cado-NFS and shows it can be more efficient than what was used to factor the current RSA-250 record. We achieve acceleration factors of up to 1.1 on sampled sieved regions. These results come however at a cost : the loss of precision at the heart of our proposition leads to finding fewer relations for the same amount of explored regions. As sieving gets less efficient for larger special-$\mathfrak{q}$'s, compensating lost relations gets costlier.

Further experiments are needed in order to find better parameters, focusing not only on the local speed-up but also on the proportion of found relations and the added work they imply.

The way we integrated Bernstein's batch factoring algorithm within Cado-NFS sieving might not be optimal. Batch factoring happens twice for each sieved region, once on each side. It is more efficient to fill one batch. In our implementation, this is however usually not the case on the first side (algebraic) and far from it on the second side (rational) as most norms are discarded between both steps. Dissociating batch factoring from sieved regions might then lead to further acceleration.

We showed experiments on the bound B separating sieved primes and those in batch factoring. We have yet to try experimenting different bounds for each side (B_0 and B_1) although it appears disabling entirely the small sieve for both sides, as we did, is the straightforward approach.

Finally, data collected during RSA-250 allowed us to draw early conclusions to pick optimal parameters such as `mfbb0` and `mfbb1` for each targeted proportion of relations. This brings us closer to an answer to the question "would the RSA-250 record have been beaten quicker using our alternative?". Exploring parameters for yet-to-be factored sized numbers would bring an answer to the more interesting one "will the next RSA record be faster with it?".

Acknowledgments

We thank the authors of [6] for giving us access to the relations collected during the factorization of RSA-250, as well as answering our many questions.

We acknowledge financial support from the French Agence Nationale de la Recherche under projects "GORILLA" (ANR-20-CE39-0002) and "KLEPTOMANIAC" (ANR-21-CE39-0008-02).

Experiments presented in this paper 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. Aoki, K., Ueda, H.: Sieving using bucket sort. In: Lee, P.J. (ed.) Advances in Cryptology - ASIACRYPT 2004, 10th International Conference on the Theory and Application of Cryptology and Information Security, Jeju Island, Korea, December 5-9, 2004, Proceedings. Lecture Notes in Computer Science, vol. 3329, pp. 92–102. Springer (2004). `https://doi.org/10.1007/978-3-540-30539-2_8`, `https://doi.org/10.1007/978-3-540-30539-2_8`
2. Bender, E.A., Canfield, E.: An approximate probabilistic model for structured gaussian elimination. Journal of Algorithms **31**(2), 271–290 (1999). `https://doi.org/https://doi.org/10.1006/jagm.1999.1008`, `https://www.sciencedirect.com/science/article/pii/S0196677499910088`

3. Bernstein, D.J.: How to find small factors of integers (September 2002), https://cr.yp.to/papers.html#sf, (Apparenly) unpublished manuscript. Available online (https://cr.yp.to/papers.html#sf)
4. Bernstein, D.J.: How to find smooth parts of integers (May 2004), https://cr.yp.to/papers.html#smoothparts, unpublished manuscript. Available online (https://cr.yp.to/papers.html#smoothparts)
5. Bernstein, D.J.: Scaled remainder trees (August 2004), https://cr.yp.to/papers.html#scaledmod, unpublished manuscript. Available online (https://cr.yp.to/papers.html#scaledmod)
6. Boudot, F., Gaudry, P., Guillevic, A., Heninger, N., Thomé, E., Zimmermann, P.: Comparing the difficulty of factorization and discrete logarithm: A 240-digit experiment. In: Micciancio, D., Ristenpart, T. (eds.) Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part II. Lecture Notes in Computer Science, vol. 12171, pp. 62–91. Springer (2020). https://doi.org/10.1007/978-3-030-56880-1_3, https://doi.org/10.1007/978-3-030-56880-1_3
7. Bouillaguet, C., Zimmermann, P.: Parallel structured gaussian elimination for the number field sieve. Mathematical Cryptology **0**(1), 22–39 (Jan 2021), https://journals.flvc.org/mathcryptology/article/view/126033
8. Buhler, J.P., Lenstra, H.W., Pomerance, C.: Factoring integers with the number field sieve. In: Lenstra, A.K., Lenstra, H.W. (eds.) The development of the number field sieve. pp. 50–94. Springer Berlin Heidelberg, Berlin, Heidelberg (1993)
9. Buhler, J., Stevenhagen, P.: Algorithmic Number Theory: Lattices, Number Fields, Curves and Cryptography. Mathematical Sciences Research Institute Publications, Cambridge University Press (2011), https://books.google.fr/books?id=fjzgYQEACAAJ
10. De La Bretèche, R., Tenenbaum, G.: Propriétés statistiques des entiers friables. The Ramanujan Journal **9**, 139–202 (03 2005). https://doi.org/10.1007/s11139-005-0832-6, https://doi.org/10.1007/s11139-005-0832-6
11. Dean, J., Ghemawat, S.: Mapreduce: Simplified data processing on large clusters. Commun. ACM **51**(1), 107–113 (jan 2008). https://doi.org/10.1145/1327452.1327492, https://doi.org/10.1145/1327452.1327492
12. Hildebrand, A., Tenenbaum, G.: On integers free of large prime factors. Transactions of The American Mathematical Society - TRANS AMER MATH SOC **296** (07 1986). https://doi.org/10.2307/2000573
13. Joux, A.: Algorithmic cryptanalysis. CRC Press (2009)
14. 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.J.J., Timofeev, A., Zimmermann, P.: Factorization of a 768-bit RSA modulus. In: Rabin, T. (ed.) Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings. Lecture Notes in Computer Science, vol. 6223, pp. 333–350. Springer (2010). https://doi.org/10.1007/978-3-642-14623-7_18, https://doi.org/10.1007/978-3-642-14623-7_18
15. Kruppa, A.: Speeding up Integer Multiplication and Factorization. Theses, Université Henri Poincaré - Nancy 1 (Jan 2010), https://theses.hal.science/tel-01748662
16. Lenstra, A.K., Lenstra, H.W., Manasse, M.S., Pollard, J.M.: The number field sieve. In: Lenstra, A.K., Lenstra, H.W. (eds.) The development of the number field sieve. pp. 11–42. Springer Berlin Heidelberg, Berlin, Heidelberg (1993)
17. Marcus, D.: Number Fields. Universitext, Springer New York (2012), https://books.google.fr/books?id=kmnlBwAAQBAJ

18. Montgomery, P.L.: Square roots of products of algebraic numbers. In: Gautschi, W. (ed.) Mathematics of Computation 1943–1993: a half-century of computational mathematics. pp. 567–571. American Mathematical Society, Providence (1994)
19. Pollard, J.M.: The lattice sieve. In: Lenstra, A.K., Lenstra, H.W. (eds.) The development of the number field sieve. pp. 43–49. Springer Berlin Heidelberg, Berlin, Heidelberg (1993)
20. Vallée, B., Vera, A.: Probabilistic analyses of lattice reduction algorithms. In: Nguyen, P.Q., Vallée, B. (eds.) The LLL Algorithm - Survey and Applications, pp. 71–143. Information Security and Cryptography, Springer (2010). https://doi.org/10.1007/978-3-642-02295-1_3, https://doi.org/10.1007/978-3-642-02295-1_3
21. Zaharia, M., Xin, R.S., Wendell, P., Das, T., Armbrust, M., Dave, A., Meng, X., Rosen, J., Venkataraman, S., Franklin, M.J., Ghodsi, A., Gonzalez, J., Shenker, S., Stoica, I.: Apache spark: A unified engine for big data processing. Commun. ACM **59**(11), 56–65 (oct 2016). https://doi.org/10.1145/2934664, https://doi.org/10.1145/2934664