Higher dimensional sieving for the number field sieve algorithms

Laurent Grémy

HIGHER DIMENSIONAL SIEVING FOR THE NUMBER FIELD SIEVE ALGORITHMS

LAURENT GRÉMY

Abstract. Since 2016 and the introduction of the exTNFS (extended Tower Number Field Sieve) algorithm, the security of cryptosystems based on non-prime finite fields, mainly the paring and torus-based one, is being reassessed. The feasibility of the relation collection, a crucial step of the NFS variants, is especially investigated. It usually involves polynomials of degree one, i.e., a search space of dimension two. However, exTNFS uses bivariate polynomials of at least four coefficients. If sieving in dimension two is well described in the literature, sieving in higher dimension received significantly less attention. We describe and analyze three different generic algorithms to sieve in any dimension for the NFS algorithms. Our implementation shows the practicability of dimension four sieving, but the hardness of dimension six sieving.

1. Introduction

Nowadays, an important part of the deployed asymmetric cryptosystems bases its security on the hardness of two main number theory problems: the factorization of large integers and the computation of discrete logarithms in a finite cyclic group. In such a group $(G, \cdot)$ of order ℓ and generator g, the *discrete logarithm problem* (DLP) is, given $a \in G$, to find $x \in [0, \ell)$ such that $g^x = a$. Usual choices of group are groups of points on elliptic curves or multiplicative subgroups of finite fields.

In this article, we focus on discrete logarithms in finite fields of the form $\mathbb{F}_{p^n}$, where p is a prime and n is relatively small, namely the medium and large characteristics situation studied in [22]. Computing discrete logarithms in this type of field can affect torus-based [29, 36] or pairing-based [12] cryptography. The best known algorithm to achieve computations in such groups is the *number field sieve* (NFS) algorithm. It has a subexponential complexity, often expressed with the $L(\alpha)$ notation: $L_{p^n}(\alpha, c) = \exp\left[(c + o(1)) \log(p^n)^\alpha \log\log(p^n)^{1-\alpha}\right]$, where $\alpha = 1/3$ for all the variants of NFS. For the general setting in medium characteristic, the first $L(1/3)$ algorithm was reached with $c = 2.43$ [22], improved to 2.21 [4] and now to 1.93 with exTNFS [23], the same complexity as NFS in large characteristic. In some specific context, exTNFS even reaches a lower complexity. However, theoretical complexities are not enough to estimate what would cost a real attack, since practical improvements can be hidden in the $o(1)$ term [1, 30, 7]. Experimental results are then needed to assess the concrete limits of known algorithms.

On the practical side, there has been a lot of effort to compute discrete logarithms in prime fields, culminating in a 768-bit record [27]. Although the records for $\mathbb{F}_{p^2}$ are smaller than the ones in prime fields, the computations turned out to be

Laurent Grémy was supported by the ERC Starting Grant ERC-2013-StG-335086-LATTAC. His work was started in the CARAMBA team of Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France and completed in the AriC team.

faster than expected [4]. However, when n is a small composite and p fits for $\mathbb{F}_{p^n}$ to be in the medium characteristic case (typically $n = 6$ [16] and $n = 12$ [18]), the records are smaller, even with a comparable amount of time spent during the computation. A way to fill the gap between medium and large characteristics is to implement exTNFS, since the computations in medium characteristic were, until now, performed with a predecessor of exTNFS.

Since exTNFS is a relatively new algorithm, there remain many theoretical and practical challenges to be solved before a practical computation can be reached. One of the major challenges concerns the sieve algorithms which efficiently perform the relation collection, one of the most costly steps of NFS. However, if there exist sieve algorithms in dimension two and three, these sieves are not efficient for higher dimension and exTNFS needs to sieve in even dimension larger or equal to four.

Our contributions. We describe three new generic sieve algorithms which deal with any dimension, especially those addressed by exTNFS. Instantiating these algorithms in dimension two or three may allow to recover the existing sieve algorithms. Since these new sieves do not ensure completeness of the enumeration, unlike most of the existing sieve algorithms, we describe workarounds to ensure a trade-off between the completeness and the running time efficiency. Finally, we analyze some quality criteria of these sieve algorithms and show the feasibility of sieving in dimension four, but the hardness of dimension six sieving.

2. Overview of the NFS algorithms

Let ℓ be a large prime factor of the order $\Phi_n(p)$ of $\mathbb{F}_{p^n}^*$ that is coprime to $\Phi_k(p)$ for all prime factors k of n: the Pohlig–Hellman algorithm allows to reduce the DLP in $\mathbb{F}_{p^n}^*$ to the DLP in all its subgroups, especially the one of order ℓ. The NFS algorithms can be split into four main steps: *polynomial selection*, *relation collection*, *linear algebra* and *individual logarithm*. The first step defines in a convenient way the field $\mathbb{F}_{p^n}$. The next two steps find the discrete logarithms of a subset of *small to medium* elements of $\mathbb{F}_{p^n}$, where sizes of the elements will be defined later. The last step computes the discrete logarithm of a *large* element of $\mathbb{F}_{p^n}$. The overall complexity of NFS is dominated by the relation collection and the linear algebra.

2.1. Polynomial selection.

Let $n = \eta\kappa$: the field $\mathbb{F}_{p^n}$ can be represented as a degree-κ extension of $\mathbb{F}_{p^\eta}$. Let h be an integer polynomial of degree η irreducible over $\mathbb{F}_p$ and ι be a root of h. Let $\mathbb{F}_{p^\eta}$ be defined by R/pR, where R is the ring $\mathbb{Z}[y]/h(y)$. There exist two ring homomorphisms from $R[x] = \mathbb{Z}[\iota][x]$ to $\mathbb{F}_{p^n}$: one of them involves a number field K_0 (respectively K_1) defined by f_0 (respectively f_1). The polynomials f_0 and f_1 are irreducible over R and share a common irreducible factor ϕ of degree κ modulo p. This setting allows to define $\mathbb{F}_{p^n} = \mathbb{F}_{(p^\eta)^\kappa} \approx (R/pR)[x]/\phi(x)$. This provides the commutative diagram of Figure 1. The different polynomial selections defining f_0 and f_1 are given in Figure 2.

2.2. Relation collection.

Since the diagram of Figure 1 is the same for all the NFS variants, we use in the following the name NFS_η to cover all the variants (see Table 1 for their names) or NFS when the parameter η does not matter.

2.2.1. *Relation.*

A relation in NFS is given by a polynomial $a(x, y)$ in $R[x]$ of degree μ in x, often set to $\mu = 1$ to reach the best complexity (see Table 1), and $\eta - 1$ in y. Since there are $t = (\mu + 1)\eta$ coefficients to define a polynomial a, the relation collection is done in *dimension* t. A polynomial a gives a relation when the ideal

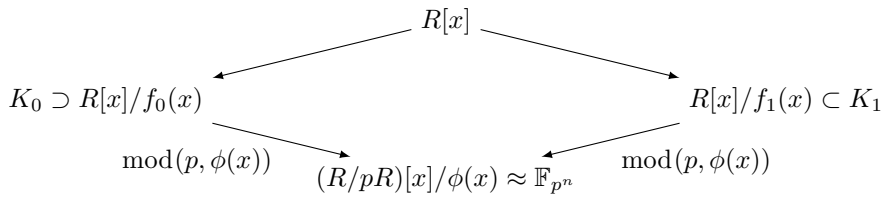

FIGURE 1. The NFS diagram to compute discrete logarithms in $\mathbb{F}_{p^n}$.

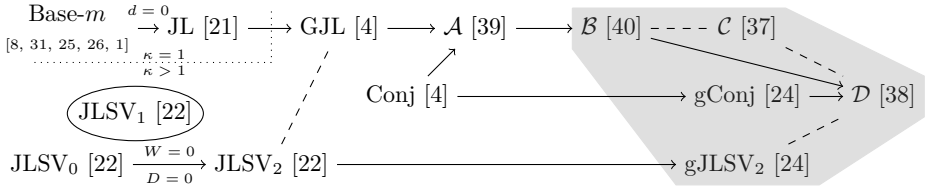

FIGURE 2. Polynomial selections: a link $a \to b$ means that a is a particular case of b (getting a from b is written if this is not explicit in the articles); a dashed link means that the selection strategies in a and b strongly resemble. Polynomial selections in the gray area allow to build polynomial with algebraic coefficients.

factorizations of a mapped in both number fields involve prime ideals of norms smaller than two $L(1/3)$ smoothness bounds B_0 and B_1 respectively: such ideals are elements of the so-called factor bases $\mathcal{F}_0$ and $\mathcal{F}_1$ respectively, see [22, 5, 23].

Since the factorization of a in prime ideals and the factorization of the norm of a are strongly linked, the relation collection looks for polynomials a of norms B_0-smooth in K_0 and B_1-smooth in K_1. To ensure the best probability of smoothness, the t coefficients $\boldsymbol{a}$ of a are taken into a *t-search space* $\mathcal{S}$ containing $L(1/3)$ elements. Since an upper bound of the norm of a involves its infinity norm [6], the search spaces are usually cuboids of form $\mathcal{S} = [S_0^m, S_0^M) \times [S_1^m, S_1^M) \times \cdots \times [S_{t-1}^m, S_{t-1}^M)$, where $\mathbf{0}$ is in $\mathcal{S}$, all the $[S_i^m, S_i^M)$ are integer intervals and $S_i^M = -S_i^m$, where i is in $[0, t)$. Theoretically, all the S_i^M are equal but practically, the skewness of the polynomials f_0 and f_1 must be taken into account [31, 25, 26, 1], implying a skew search space. Since $-a$ and a give the same relation, $S_{t-1}^m = 0$. By abuse of notation, we denote by a both the polynomial and the list $\boldsymbol{a}$ of its t coefficients.

	$\kappa = 1$	$\kappa \geq 1$
$\eta = 1$	NFS	NFS-HD
$\eta \geq 1$	TNFS	exTNFS

(A) Name of the NFS variants.

	$\kappa = 1$	$\kappa \geq 1$
$\eta = 1$		$\mu \geq 1$
$\eta \geq 1$	$\mu = 1$	

(B) Optimal degree.

TABLE 1. The different variants of NFS.

2.2.2. *Practical considerations.* To ensure the best running time for the relation collection, the polynomials f_0 and f_1 must be chosen carefully. However, the two

usual quality criteria, especially the α but also the Murphy-E functions [31], are only defined for NFS_1 and $\mu \leq 3$ [14]. Finding good polynomials for $\text{NFS}_{>1}$, even by settling for integer coefficients to define f_0 and f_1, is yet a challenge.

The goal of the relation collection is to produce more relations than the number of ideals in both factor bases. A classical algorithm, used to analyze theoretically NFS, tests the smoothness of the norms of a in $\mathcal{S}$ by using the *elliptic curve method* (ECM) algorithm. However, if this algorithm is almost memory-free, the practical running time of such a task is prohibitive.

Instead, the common practical way is to perform ECM only on promising polynomials a, i.e., polynomials whose norms have many small factors. Finding these small factors is efficiently performed thanks to arithmetic sieve algorithms. However, sieve algorithms need a huge memory-footprint, since they need to store the norms of all the elements of $\mathcal{S}$. This problem was tackled in [33], allowing moreover a high-level parallelization, by considering many subsets of polynomial: in one number field, say K_0, the factorization into prime ideals of these polynomials involved at least an enforced ideal of medium size. Let $\mathfrak{Q}$ be such an ideal, called special-$\mathfrak{Q}$. Polynomials a such that $\mathfrak{Q}$ appears into their ideal factorization in K_0 are elements of a lattice, called $\mathfrak{Q}$-lattice, a basis of which is given by the rows of the matrix $M_{\mathfrak{Q}}$. To consider only polynomials fitting into $\mathcal{S}$, sieves look for elements $\boldsymbol{c}$ in the intersection of the $\mathfrak{Q}$-lattice and a t-search space $\mathcal{H} = [H_0^m, H_0^M) \times [H_1^m, H_1^M) \times \cdots \times [0, H_{t-1}^M)$: a is obtained from $\boldsymbol{c}M_{\mathfrak{Q}}$. If theoretically $\mathcal{H}$ should depend on $\mathfrak{Q}$, it is often the same for all the special-$\mathfrak{Q}$s. In this intersection, sieve algorithms remove the contribution of small ideals. Let $\mathfrak{R}$ be such an ideal of prime norm r. Except for a tiny number of such ideals, a basis of the $\mathfrak{R}$-lattice in the $\mathfrak{Q}$-lattice can be of the form

$$\begin{aligned}(1)\quad \{(r,0,0,\ldots,0),(\lambda_{0,\mathfrak{Q},\mathfrak{R}},1,0,0,\ldots,0),(\lambda_{1,\mathfrak{Q},\mathfrak{R}},0,1,0,0,\ldots,0),\ldots,\\ (\lambda_{t-2,\mathfrak{Q},\mathfrak{R}},0,0,\ldots,0,1)\} = \{\boldsymbol{b_0},\boldsymbol{b_1},\boldsymbol{b_2},\ldots,\boldsymbol{b_{t-1}}\},\end{aligned}$$

where the $\lambda_{i,\mathfrak{Q},\mathfrak{R}}$ are integers in $[0,r)$. Briefly, the different steps of the relation collection with the *special-$\mathfrak{Q}$-method* and sieving algorithms are as follows:

- For all the possible special-$\mathfrak{Q}$s
  1. For both sides i in $[0,1]$
     (a) Compute the norms $N_i[\boldsymbol{c}]$ of $a = \boldsymbol{c}M_{\mathfrak{Q}}$, where $\boldsymbol{c}$ is in $\mathcal{H}$.
     (b) For all the ideals $\mathfrak{R}$ to be sieved, enumerate the elements $\boldsymbol{c}$ in $\mathcal{H} \cap \Lambda_{\mathfrak{Q}\mathfrak{R}}$ and remove the contribution of $\mathfrak{R}$ from $N_i[\boldsymbol{c}]$.
  2. If both $N_0[\boldsymbol{c}]$ and $N_1[\boldsymbol{c}]$ are sufficiently small to have a chance to give a relation, factorize the norms of a and report a if a gives a relation.

However, if there exist generic sieve algorithms in any dimension (see Section 3), they are not very efficient when $t \geq 4$ that especially arises with $\text{NFS}_{>1}$. We propose algorithms for these cases in Section 4. Note that we will use the term sieve algorithms, but we only focus on the enumeration part of them, which is Step 1b without updating the array N_i. Step 1a is briefly addressed in Section 5.

2.3. **Linear algebra and individual logarithm.** Let θ_0 (respectively θ_1) be a root of f_0 (respectively f_1). Let a be a polynomial that gives a relation, i.e., $\langle a(\theta_k,\iota)\rangle = \prod_{\mathfrak{R}\in\mathcal{F}_k} \mathfrak{R}^{\text{val}_{\mathfrak{R}}(a(\theta_k,\iota))}$, where k is in $[0,1]$ and val denotes the valuation: the factorizations of the norms of a must be translated into such a factorization of ideals [9]. A relation can be transformed into a linear relation involving the virtual logarithms (vlog) of the ideals [42]. To be valid, this linear relation must

involve the Schirokauer maps ϵ_k [41], as $\sum_{\mathfrak{R}\in\mathcal{F}_0} \mathrm{val}_{\mathfrak{R}}(a(\theta_k,\iota))\,\mathrm{vlog}(\mathfrak{R}) + \epsilon_0(a) = \sum_{\mathfrak{R}\in\mathcal{F}_1} \mathrm{val}_{\mathfrak{R}}(a(\theta_k,\iota))\,\mathrm{vlog}(\mathfrak{R}) + \epsilon_1(a) \mod \ell$. In this equation, the virtual logarithms are unknowns, the valuations are small integers and the Schirokauer maps are large integers, close to ℓ. These large elements negatively impact the usual algorithms to solve sparse systems, but the cost of these heavy parts can be significantly decreased thanks to a modification of the block Wiedemann algorithm [10, 20, 13].

The last step of the computation is the computation of a large, say $L(1)$, unknown logarithm. This computation is achieved by rewriting the (virtual) logarithm of the target in terms of logarithms of smaller elements; these smaller elements are again rewritten in terms of smaller elements until the logarithm of the target has been rewritten using only the precomputed logarithms given by the relation collection and the linear algebra. This *descent step* uses different algorithms depending on the size of the rewritten element: the target is rewritten in elements up to $L(2/3)$ thanks to the so-called initial splitting (booting step) [34, 21, 2, 17, 45]; for elements in $[L(1/3), L(2/3))$, the special-$\mathfrak{Q}$-method is used. The theoretical analysis of [13, Appendix A.2] shows that the descent by special-$\mathfrak{Q}$ may be more efficient by considering polynomials of degree not restricted to $\mu = 1$.

3. A framework to study existing sieve algorithms

Let $\mathfrak{Q}$ be a special-$\mathfrak{Q}$ and $\mathfrak{R}$ be an ideal to be sieved such that the lattice $\Lambda_{\mathfrak{Q}\mathfrak{R}}$ is given by a basis as in Equation (1)[1]. There exist different sieve algorithms proposed for NFS that allow to enumerate the elements in the intersection of $\Lambda_{\mathfrak{Q}\mathfrak{R}}$ and a search space $\mathcal{H}$. Their efficiency depends on the dimension of $\Lambda_{\mathfrak{Q}\mathfrak{R}}$ and the density of the lattice in $\mathcal{H}$. This density is formally defined thanks to the *level* of a sieve algorithm in Definition 3.1, a key notion for the rest of the description and especially for Section 4. All the existing sieve algorithms used in NFS are reported in Table 2. These algorithms can be described by the following two-step algorithm. The vectors produced in Step 2 will be called *transition-vectors*:

(1) Compute an adapted set $\mathcal{B}$ of spanning vectors of $\Lambda_{\mathfrak{Q}\mathfrak{R}}$ with respect to $\mathcal{H}$.
(2) Start from $\mathbf{0}$ and use the vectors of $\mathcal{B}$ or a (often small) linear combination of them to enumerate elements in the intersection of $\Lambda_{\mathfrak{Q}\mathfrak{R}}$ and $\mathcal{H}$.

Definition 3.1 (Level)**.** Let Λ be a lattice and $\mathcal{H}$ be a search space. The level of a sieve algorithm with respect to Λ and $\mathcal{H}$ is the minimal integer value $\ell < t$ such that the intersections of the cuboids $[H_0^m, H_0^M) \times [H_1^m, H_1^M) \times \cdots \times [H_\ell^m, H_\ell^M) \times \{c_{\ell+1}\} \times \{c_{\ell+2}\} \times \cdots \times \{c_{t-1}\}$, where $(c_{\ell+1}, c_{\ell+2}, \ldots, c_{t-1})$ are in $[H_{\ell+1}^m, H_{\ell+1}^M) \times [H_{\ell+2}^m, H_{\ell+2}^M) \times \cdots \times [H_{t-1}^m, H_{t-1}^M)$, and the lattice Λ contains more than one element on average. In case $\mathcal{H}$ contains less than one element on average, $\ell = t - 1$.

3.1. Exhaustive sieve algorithms.

The first use of a sieve algorithm in an index calculus context is attributed to Schroeppel and was successfully used by Pomerance [35, 28]. They used the one-dimensional sieve of Eratosthenes as a factoring algorithm instead of a prime detecting one. It was extended to any dimension and called *line sieve*, see for example its use in dimension three in [44]. In dimension two, the line sieve is known to be inefficient when there is at most one element in a line, an intersection of $\Lambda_{\mathfrak{Q}\mathfrak{R}}$ and $[H_0^m, H_0^M) \times \{c_1\}$ where c_1 is in $\mathbb{Z}$: the 0-level line sieve is used as a 1-level sieve algorithm. Pollard designed in this sense the *sieve by*

[1]Sieve algorithms can deal with other basis shapes of lattices, but this one occurs the most.

vectors [33], now subsumed by the *lattice sieve* of Franke and Kleinjung [11]. Based on this sieve algorithm, the *plane sieve* [14] and the *3-dimensional lattice sieve* [19] were proposed for similar densities in three dimensions. The plane sieve was turned into a generic sieve algorithm in CADO-NFS [43] (see Section 4.4).

The completeness of all these sieve algorithms comes from special procedures that compute transition-vectors. They are defined thanks to the *t-extended search spaces*: let k be in $[0, t)$ and $\mathcal{H}$ be a t-search space, the t-extended search space $\mathcal{H}_k$ is the set $[H_0^m, H_0^M) \times [H_1^m, H_1^M) \times \cdots \times [H_k^m, H_k^M) \times \mathbb{Z}^{t-(k+1)}$.

Definition 3.2 (Transition-vector)**.** Let k be in $[0, t)$ and $\mathcal{H}$ be a t-search space. A k–transition-vector is an element $\boldsymbol{v} \neq \mathbf{0}$ of a lattice Λ such that there exist $\boldsymbol{c}$ and $\boldsymbol{d}$ in the intersection of Λ and the t-extended search space $\mathcal{H}_k$, where $\boldsymbol{d} = \boldsymbol{c} + \boldsymbol{v}$ is such that the $t - 1 - k$ last coordinates of $\boldsymbol{c}$ and $\boldsymbol{d}$ are equal and the coordinate $\boldsymbol{d}[k]$ is the smallest possible larger than $\boldsymbol{c}[k]$.

With such sieve algorithms, the small factors of both norms of all the considered polynomials a are known: this allows to be close to the expected number of relations at the end of the relation collection. But, the number of relations is not the only efficiency criterion of the relation collection. Indeed, in dimension two, the lattice sieve is used since it allows to maintain the same number of relations but decrease the time per relation. The same occurs in dimension three, switching from the line to the plane or the 3D-lattice sieves. However, these sieves have some drawbacks, highlighted when there is less than one element on average in each plane $[H_0^m, H_0^M) \times [H_1^m, H_1^M) \times \{c_2\}$, where c_2 is in $[H_2^m, H_2^M)$. The plane sieve is essentially the use of the lattice sieve on each plane: even if there is no element in a plane, the lattice sieve is used to report nothing instead of using it only on non empty planes. There is no alternative to avoid these useless uses without losing completeness. The 3D-lattice sieve does not have this drawback, but the procedure to generate the spanning list $\mathcal{B}$ and the one to enumerate seem difficult to analyze and may be costly for skewed lattices or skewed search spaces.

3.2. **Heuristic sieve algorithms.** Because of these drawbacks and especially the penalty in terms of running time, the designers of the plane sieve proposed a heuristic sieve algorithm, the *space sieve* [14]. Its use allows to decrease the running time by 45% for the 240 bits example of [14], while at the same time losing less than 6% of the relations. This corresponds to decrease the time per relation by 42%.

	Line sieve	Lattice sieve [11]	3-dimensional lattice sieve [19]	Plane sieve [14]	Space sieve [14]	This work
$t = 2$	✓	✓	✗	✗	✗	✓
$t = 3$	✓	✗	✓	✓	✓	✓
$t > 3$	✓	✗	✗	✓	✗	✓
Level	$\ell = 0$	$\ell = 1$	$\ell = 1$ and $\ell = 2$	$\ell = 1$	$\ell = 2$	Any
Completeness of enumeration	✓	✓	✓	✓	✗	✗

TABLE 2. Characteristics of the sieve algorithms proposed for NFS.

The space sieve focuses on enumerating a large portion of the elements instead of all of them, which is helpful for multiple reasons. First, all the sieve algorithms, both exhaustive and heuristic, allow to enumerate the t-extended search space $\mathcal{H}_{t-2}$

instead of the search space $\mathcal{H} = \mathcal{H}_{t-1}$. For exhaustive sieves, it implies that the spanning set $\mathcal{B}$ is qualitatively too accurate because it allows to generate transition-vectors that will never be used. If this accuracy implies a costly computation to find an adapted set $\mathcal{B}$, the time per relation can be drastically impacted. Secondly, completeness is not always useful, since this reports hits on polynomials a that will give relations or not: missing some hits may not affect the number of relations in some circumstances. Furthermore, if the computation can be completed, the expected gain in the time per relation must be considered to compare heuristic and exhaustive sieves, even if the relation collection misses some relations. Finally, in dimension larger than three, the use of heuristic sieve seems unavoidable: to the best of our knowledge, producing all the transition-vectors can only be computed by the exhaustive sieve algorithms, all of them being inefficient when there is less than one element in $[H_0^m, H_0^M) \times [H_1^m, H_1^M) \times [H_2^m, H_2^M) \times \{c_3\} \times \{c_4\} \times \cdots \times \{c_{t-1}\}$, where c_i is in $[H_i^m, H_i^M)$. Yielding to produce some transition-vectors can be done by computing Graver basis of the lattice: these transition-vectors may lead to build a generic exhaustive sieve algorithm from the heuristic one described in Section 4. However, computing Graver basis is often too costly in our context [15, 32].

In the following, we propose `globalntv`, `localntv` and `sparsentv`, three heuristic sieves which perform the enumeration in any dimensions and levels.

4. Sieve algorithms in higher dimensions

Using transition-vectors implies that the sieve enumerations are exhaustive. Since completeness is not the main target of `globalntv`, `localntv` and `sparsentv`, the vectors used in Step 2 of Section 3, called here *nearly-transition-vectors*, will be weakened by removing from Definition 3.2 the strong condition about $\boldsymbol{d}[k]$.

Definition 4.1 (Nearly-transition-vector)**.** Let k be in $[0, t)$ and $\mathcal{H}$ be a t-search space. A k–nearly-transition-vector is an element $\boldsymbol{v} \neq \boldsymbol{0}$ of a lattice Λ such that there exist $\boldsymbol{c}$ and $\boldsymbol{d}$ in the intersection of Λ and the t-extended search space $\mathcal{H}_k$, where $\boldsymbol{d} = \boldsymbol{c} + \boldsymbol{v}$ is such that the $t-1-k$ last coordinates of $\boldsymbol{c}$ and $\boldsymbol{d}$ are equal and the coordinate $\boldsymbol{d}[k]$ is larger than $\boldsymbol{c}[k]$.[2]

The three generic sieve algorithms will take place in a general framework, described by allowing the report of duplicated elements for simplicity in Algorithm 1. It is purposely vague, to be as general as possible: instantiation examples of **Initialization**, Step 1c and Step 1d will be given in the following.

The addition of a possible nearly-transition-vector (Step 1c) is likewise performed for all the three sieve algorithms. Like the addition of a 2–nearly-transition-vector in the space sieve [14], a loop iterate the list of k–nearly-transition-vectors, beforehand sorted by increasing coordinate k (see Section 4.3). We also choose to use the same fall-back strategy (Step 1d): this choice is justified in Section 4.2. Therefore, the difference between the three sieve algorithms only comes from the initialization processes, described in Section 4.1.

4.1. Initializations.

To define the three initialization processes, we introduce two new notions: the *shape* of the nearly-transition-vectors and the *skew-small-vectors*.

[2]Note that transition vectors of [14, Definition 5] are 2–nearly-transition-vectors.

Initialization: Call a procedure that returns nearly-transition-vectors with respect to a search space $\mathcal{H}$ and a lattice $\Lambda_{\mathfrak{Q}\mathfrak{R}}$ described as in Equation (1). Set $\boldsymbol{c}$ to $\mathbf{0}$ and k to $t-1$.

Enumeration:

(1) While $\boldsymbol{c}[k] < H_k^M$:
   (a) Report $\boldsymbol{c}$.
   (b) If $k > 0$, call this enumeration procedure recursively with inputs $\boldsymbol{c}$ and $k-1$.
   (c) Find a k–nearly-transition-vector $\boldsymbol{v}$ from the one computed during **Initialization**, such that adding $\boldsymbol{v}$ to $\boldsymbol{c}$ lands in the extended search space $\mathcal{H}_{k-1}$ and $\boldsymbol{c}[k]$ is the smallest possible.
   (d) If there does not exist such a k–nearly-transition-vector $\boldsymbol{v}$, call a *fall-back strategy* that tried to produce a new element $\boldsymbol{c}$ in $\Lambda_{\mathfrak{Q}\mathfrak{R}} \cap \mathcal{H}$, and therefore a new k–nearly-transition-vector.
(2) Recover $\boldsymbol{c}$ as it was when the procedure was called.
(3) While $\boldsymbol{c}[k] \geq H_k^m$, perform Step 1a, Step 1b, Step 1c and Step 1d by considering $\boldsymbol{c} - \boldsymbol{v}$ instead of $\boldsymbol{c} + \boldsymbol{v}$.

ALGORITHM 1. Framework for `globalntv`, `localntv` and `sparsentv`.

4.1.1. *Preliminaries.* Even if the three initialization processes are different, the shapes of the nearly-transition-vectors are the same. The shape represents the expected magnitude of the coefficients of the nearly-transition-vectors with respect to a search space $\mathcal{H}$ and $\Lambda_{\mathfrak{Q}\mathfrak{R}}$. In this paragraph, the $O(j)$ notation will denote a value smaller or almost equal to j. Let us recall the shape of the transition-vectors of the ℓ-level sieve algorithms in three dimensions. Let I_i be the length of the interval $[H_i^m, H_i^M)$. When $\ell = 0$, the shape is equal to $(O(r), O(1), O(1))$; the one for $\ell = 1$ is $(O(I_0), O(r/I_0), O(1))$; the one for $\ell = 2$ is $(O(I_0), O(I_1), O(r/(I_0 I_1)))$. This shape is generalized, as $(I_0, I_1, \ldots, I_{\ell-1}, r/(I_0 \times I_1 \times \cdots \times I_{\ell-1}), 1, 1, \ldots, 1)$, given a level ℓ of a sieve algorithm and removing the $O(\cdot)$ for clarity.

The initialization processes of the three sieve algorithms does not ensure that the produced vectors are nearly-transition-vectors. They build skew-small-vectors, that are lattice vectors whose coefficients try to follow the shape. Even if Definition 4.2 will not capture it, skew-small-vectors are build to be almost nearly-transition-vectors: a k–skew-small-vector $\boldsymbol{v}$ is a k–nearly-transition-vector if $|\boldsymbol{v}[i]| < I_i$.

Definition 4.2 (Skew-small-vector)**.** Let k be in $[0, t)$. A k–skew-small-vector is an element $\boldsymbol{v} \neq \mathbf{0}$ of a lattice Λ such that there exist $\boldsymbol{c}$ and $\boldsymbol{d}$ in Λ, where $\boldsymbol{d} = \boldsymbol{c} + \boldsymbol{v}$ is such that the $t-1-k$ last coordinates of $\boldsymbol{c}$ and $\boldsymbol{d}$ are equal and the coordinate $\boldsymbol{d}[k]$ is larger than $\boldsymbol{c}[k]$.

4.1.2. *Three initialization processes.* The three initialization processes try to generate a large number of nearly-transition-vectors, given the level ℓ of the sieve algorithms. They begin by building a basis $\mathcal{B}$ of $\Lambda_{\mathfrak{Q}\mathfrak{R}}$ whose basis vectors are skew-small-vectors. Nearly-transition-vectors are afterwards build thanks to small linear combination of the basis vectors. The major difference between `globalntv` on the one hand, and `localntv` and `sparsentv` on the other is about the coefficients of the k–skew-small-vectors, where $k > \ell$. In `localntv` and `sparsentv`, the coordinate k is enforced to 1, and even to 0 for the coordinates $\ell + 1$ to $k - 1$ in `sparsentv`.

This comes from a crude interpretation of the magnitude of the coefficients given by the shape. To build the k–skew-small-vectors, where $k \leq \ell$ for `localntv` and `sparsentv` or all of them for `globalntv`, the initialization processes compute a skew basis of a (sub)lattice, that is a basis formed by skew-small-vectors. The basis $\mathcal{B}$ is build thanks to:

- a skew basis reduction of $\{\boldsymbol{b_0}, \boldsymbol{b_1}, \ldots, \boldsymbol{b_{t-1}}\}$ for `globalntv`;
- a skew basis reduction of $\{\boldsymbol{b_0}, \boldsymbol{b_1}, \ldots, \boldsymbol{b_\ell}\}$ followed by, for k in $[\ell+1, t)$, a reduction of $\boldsymbol{b_k}$ by its closest vector in $\{\boldsymbol{b_0}, \boldsymbol{b_1}, \ldots, \boldsymbol{b_{k-1}}\}$ for `localntv`;
- a skew basis reduction of $\{\boldsymbol{b_0}, \boldsymbol{b_1}, \ldots, \boldsymbol{b_\ell}\}$ followed by, for k in $[\ell+1, t)$, a reduction of $\boldsymbol{b_k}$ by its closest vector in $\{\boldsymbol{b_0}, \boldsymbol{b_1}, \ldots, \boldsymbol{b_\ell}\}$ for `sparsentv`.

To build possible nearly-transition-vectors, linear combinations of the skew basis vectors are performed, as well as computations of some vectors close to $\boldsymbol{b_k}$ in the corresponding sublattice instead of one for `localntv` and `sparsentv`. The patterns of the skew-small-vectors produced by the different initializations follow necessarily the ones reported in Table 3. Note that, when $\ell = t-2$, `localntv` and `sparsentv` have the same initialization processes. When $\ell = t-1$, the three initialization processes are the same.

k	`globalntv`	`localntv`	`sparsentv`
0	$(>0,0,0,0,0)$	$(>0,0,0,0,0)$	$(>0,0,0,0,0)$
1	$(\cdot,>0,0,0,0)$	$(\cdot,>0,0,0,0)$	$(\cdot,>0,0,0,0)$
2	$(\cdot,\cdot,>0,0,0)$	$(\cdot,\cdot,>0,0,0)$	$(\cdot,\cdot,>0,0,0)$
3	$(\cdot,\cdot,\cdot,>0,0)$	$(\cdot,\cdot,\cdot,1,0)$	$(\cdot,\cdot,\cdot,1,0)$
4	$(\cdot,\cdot,\cdot,\cdot,>0)$	$(\cdot,\cdot,\cdot,\cdot,1)$	$(\cdot,\cdot,\cdot,0,1)$

TABLE 3. Patterns of the k–skew-small-vectors, where $\ell = 2$ and $t = 5$.

4.2. **A common fall-back strategy.** At this step, all the additions to $\boldsymbol{c}$ in $\Lambda_{\mathfrak{Q}\mathfrak{R}} \cap \mathcal{H}$ of a k–nearly-transition-vector fail to land in $\mathcal{H}_{k-1}$. The additions of $\boldsymbol{v}$, a k–skew-small-vector, are necessarily out of $\mathcal{H}_{k-1}$. Since no k–skew-small-vector allows to stay in $\mathcal{H}_{k-1}$, a possible k–nearly-transition-vector must have some smaller coordinates. Vectors close to $\boldsymbol{c}+\boldsymbol{v}$ in the sublattice formed by $\{\boldsymbol{b_0}, \boldsymbol{b_1}, \ldots, \boldsymbol{b_{k-1}}\}$ may allow from $\boldsymbol{c}+\boldsymbol{v}$ to obtain such a k–nearly-transition-vector. Let $\boldsymbol{e}$ be such a vector: subtract $\boldsymbol{e}$ to $\boldsymbol{c}+\boldsymbol{v}$ will shrink the k first coefficients of $\boldsymbol{c}+\boldsymbol{v}$. If $\boldsymbol{c}+\boldsymbol{v}-\boldsymbol{e}$ fits in the search space, $\boldsymbol{v}-\boldsymbol{e}$ is a new k–nearly-transition-vector. If not, set $\boldsymbol{c}$ to $\boldsymbol{c}+\boldsymbol{v}-\boldsymbol{e}$ and rerun this procedure, until $\boldsymbol{c}+\boldsymbol{v}-\boldsymbol{e}$ fits in $\mathcal{H}$ or its coordinate k is larger than H_k^M. The different steps of this fall-back strategy are, $\boldsymbol{c}$ in $\Lambda_{\mathfrak{Q}\mathfrak{R}} \cap \mathcal{H}$ and k in $[0, t)$:

(1) While $\boldsymbol{c}[k] < H_k^M$
  (a) For all k–skew-small-vectors $\boldsymbol{v}$
    (i) Compute some vectors close to $\boldsymbol{c}+\boldsymbol{v}$ in the sublattice generated by $\{\boldsymbol{b_0}, \boldsymbol{b_1}, \ldots, \boldsymbol{b_{k-1}}\}$ and store them in the list E.
    (ii) For all $\boldsymbol{e}$ in E, return $\boldsymbol{c}+\boldsymbol{v}-\boldsymbol{e}$ if $\boldsymbol{c}+\boldsymbol{v}-\boldsymbol{e}$ is in $\mathcal{H}$.
  (b) Set $\boldsymbol{c}$ to one of the vector $\boldsymbol{c}+\boldsymbol{v}-\boldsymbol{e}$ computed previously.
(2) Return fail.

If this procedure does not fail, the new element in $\mathcal{H}$ is the output of this procedure and $\boldsymbol{v}-\boldsymbol{e}$ is the new k–nearly-transition-vector, computed by the difference

between the output and the input vectors of the fall-back procedure and inserted in the lists of k–nearly-transition-vectors and k–skew-small-vectors for further use.

This fall-back strategy is costly since it requires to solve a or multiple closest vector problem instances in Step 1(a)i, iterate all the k–skew-small-vectors and loop while H_k^M is not reached. The condition to use such a strategy must therefore be carefully studied. If $k \leq \ell$, the average number of elements with the same $(t-k-1)$th last coordinate is equal to one, from the Definition 3.1 of the level. If no precomputed k–nearly-transition-vectors allows to find a new element in $\mathcal{H}$, then, most likely, there do not exist such elements. However, if $k > \ell$, there are generically more chances that such an element exists. The fall-back strategy is therefore applied only when $k > \ell$. This condition must be study a little bit more carefully. If $\ell = t-1$, the $t-1$ first coordinates of $\boldsymbol{c}+\boldsymbol{v}$ out of $\mathcal{H}$ must be shrunk, where $\boldsymbol{v}$ is a ℓ–skew-small-vector. Therefore, when $k = t-1$, the close vector $\boldsymbol{e}$ is a linear combination of $\{\boldsymbol{b_0}, \boldsymbol{b_1}, \ldots, \boldsymbol{b_{t-2}}\}$. Since this strategy allows to modify the maximal number of coordinates without changing the last non-zero one, the strategy allows to increase the chance to find a new element. Another strategy is proposed in Section 4.4, but is specific to `sparsentv`.

4.3. **Formal algorithms.** The pseudo-code of the addition of a nearly-transition-vector and the fall-back strategy are given respectively in Function `add` and in Function `fbadd`. They return an element in the intersection of $\Lambda_{\Omega\mathfrak{R}}$ and $\mathcal{H}$ or an element out of $\mathcal{H}$ to stop the enumeration of a subset of $\mathcal{H}$. The lists T and S consist of t lists containing respectively nearly-transition-vectors and skew-small-vectors (e.g., k–nearly-transition-vectors are stored in $T[k]$). Each list $T[k]$ and $S[k]$ are sorted by increasing coordinate k. Given an element $\boldsymbol{c}$ of $\Lambda_{\Omega\mathfrak{R}}$ and an integer i, the function `CVA` (Close Vectors Around a targeted element) returns a list of some lattice vectors close to $\boldsymbol{c}$ in the sublattice of $\Lambda_{\Omega\mathfrak{R}}$ generated by $\{\boldsymbol{b_0}, \boldsymbol{b_1}, \ldots, \boldsymbol{b_i}\}$.

```
FUNC. add(c, k, H, T, S, Λ_ΩR, ℓ)
  for v ∈ T[k] do
    if c + v ∈ H_{k-1} then return c + v;
  if k > ℓ or k = t − 1 then
    e ← fbadd(c, k, H, S, Λ_ΩR);
    if e ∈ H then
      T[k] ← T[k] ∪ {e − c};
      S[k] ← S[k] ∪ {e − c};
    c ← e;
  else
    c ← (H_0^M, H_1^M, ..., H_{t-1}^M);  // ∉ H
  return c;
```

```
FUNC. fbadd(c, k, H, S, Λ_ΩR)
while c[k] < H_k^M do
    L ← ∅;
    for v ∈ S[k] do
      E ← CVA(c + v, k − 1, Λ_ΩR);
      for e ∈ E do
        if c + v − e ∈ H then
          return c + v − e;
        L ← L ∪ {c + v − e};
    set c to an element of L;
return c;                          // ∉ H
```

4.4. **A specific fall-back strategy.** Unlike the previous fall-back strategy, we describe here a specific one which allows to recover all the sieve algorithms of Section 3. This specific fall-back strategy is designed for `sparsentv` by exploiting the specific patterns of the skew-small-vectors of `sparsentv`. It can be more costly but can report a larger number of elements. To completely recover exhaustive sieve algorithms, the k–skew-small-vectors used in the sieve algorithms must have their coordinate k equal to 1, when $k > \ell$.

When the fall-back strategy is called, the coefficient of $\boldsymbol{c}+\boldsymbol{v}$, where $\boldsymbol{c}$ is in $\Lambda_{\Omega\mathfrak{R}} \cap \mathcal{H}$ and $\boldsymbol{v}$ is a k–skew-small-vector, are shrunk with vectors close to $\boldsymbol{c}+\boldsymbol{v}$ in

the sublattice generated by $\{\boldsymbol{b_0}, \boldsymbol{b_1}, \ldots, \boldsymbol{b_\ell}\}$ instead of $\{\boldsymbol{b_0}, \boldsymbol{b_1}, \ldots, \boldsymbol{b_{k-1}}\}$, to keep unchanged the coordinates $\ell+1$ to $t-1$ of $\boldsymbol{c}+\boldsymbol{v}$. Let $\boldsymbol{e}$ be a vector subtracted to $\boldsymbol{c}+\boldsymbol{v}$ to shrink its coefficients. If $\boldsymbol{c}+\boldsymbol{v}-\boldsymbol{e}$ fits in $\mathcal{H}$, a new element in the intersection of $\Lambda_{\mathfrak{Q}\mathfrak{R}}$ and $\mathcal{H}$ is found, as well as a new k–nearly-transition-vector.

If $k > \ell+1$, the coordinates $\ell+1$ to $k-1$ of $\boldsymbol{c}$ have not been modified, and therefore, some cuboids of dimension $\ell+1$ were not explored to try to find a new starting point: to explore them, this procedure must be called with inputs one of the vectors generated previously and $k-1$. If all the recursions fail to find a new element in the intersection of the lattice and the search space, $\boldsymbol{c}$ is set to $\boldsymbol{c}+\boldsymbol{v}-\boldsymbol{e}$ and this procedure is redone with inputs $\boldsymbol{c}$ and k, until a generated element fits in the $\mathcal{H}$ or its coordinate k is larger than H_k^M. The different steps of this generation are the same as the ones described in Section 4.2, except that after Step 1b, the following instruction is added:

(1) While $\boldsymbol{c_t}[k] < H_k^M$
    ...
    (c) If $k-1 > \ell$, use this fall-back procedure (additive or subtractive case) with $\boldsymbol{c}$ and $k-1$ as inputs and return the result if it does not fail.
(2) Return fail.

5. Analyzes of the generic sieves

Practical generic sieve algorithms are of two types: exhaustive for the levels $\ell = 0$ and $\ell = 1$, and heuristic for all levels[3]. For levels $\ell = 0$ and $\ell = 1$, using heuristic algorithms make almost no sense, since generically, the exhaustive algorithms are optimal in term of running time. For larger levels, the practical gain obtained by using the space sieve lets us expect an improvement since exhaustive sieves are not adapted to such levels. However, heuristic sieves do not ensure completeness of the enumeration: if substantially many relations are not reported, the time per relation can negatively be impacted and can eventually be worse than with exhaustive sieves.

To evaluate the practicability of the three new sieve algorithms, we analyze them practically thanks to a Sage implementation of the three sieves named `ntv.sage` (provided in CADO-NFS), mainly implemented to test accuracy of the enumeration processes, see Section 5.1. Even if the implementation is not optimized to test running time, we can extrapolate some tendency about the efficiency of the sieves, see Section 5.2. The practical experiments were done on random lattices[4] having the shape of Equation (1), whose volume is adapted to fit for the tested levels.

5.1. **Accuracy.**

The quality criteria to test accuracy reported in Table 4 are:

- the number of produced skew-small-vectors, adjusted thanks to the number of the small linear combinations and close vectors,
- the number of iterations of the while loop in the fall-back strategy and
- the relative error between the expected number of elements to be enumerated ($\#\mathcal{H}/r$) and the number of reported elements.

The relative error informs about the accuracy of the algorithm. A large relative error likely means that the nearly-transition-vectors have too large coordinates. A

[3]Combining the 3D-lattice sieve [19] and Section 4.4 may lead to obtain an 2-level exhaustive generic sieve algorithm, but we did not manage to fully implement the 3D-lattice sieve.

[4]From the point of view of a practical sieving procedure, lattices describing ideals of a same or different factor bases, or random lattices, are treated similarly.

few more linear combinations during the initialization may solve this problem. The criterion about the fall-back strategy informs about the global effort on discovering new nearly-transition-vectors or stopping regularly the enumeration process, as the number of generated skew-small-vectors about the global effort on the initialization. The combination of these three criteria is needed since, e.g., generating a huge amount of skew-small-vectors will decrease quantitatively the two other criteria by putting solely too much effort on the initialization.

Since the patterns of the skew-small-vectors of `localntv` and `sparsentv` are constrained, their relative errors are expected to be better (i.e., smaller) than the one with `globalntv`. Since the initialization is less under control with `globalntv`, the number of skew-small-vectors may be often (much) larger for `globalntv`; however, the number of calls to the fall-back strategy is expected to be lower.

	`globalntv` ($\ell = 2$)				`localntv` ($\ell = 2$)				`globalntv` ($\ell = 3$)			
	min	med	max	mean	min	med	max	mean	min	med	max	mean
# ssvs	40				41				40			
# fbs	0	2.0	61	3.1	0	3.0	61	4.3	0	12.0	65	20.0
rel. err.	0.0	2.6	95.7	10.1	0.0	1.2	96.7	5.8	0.0	0.0	75.0	2.0

(A) Experiments on 2^{14} lattices where $\mathcal{H} = [-2^6, 2^6)^3 \times [0, 2^6)$ ($t = 4$, $\#\mathcal{H} = 2^{27}$).

	`globalntv` ($\ell = 2$)				`localntv` ($\ell = 2$)				`sparsentv` ($\ell = 2$)			
	min	med	max	mean	min	med	max	mean	min	med	max	mean
# ssvs	364				69				37			
# fbs	0	5.0	712	18.3	0	9.0	591	20.0	0	13.0	332	22.0
rel. err.	0.0	1.5	36.1	6.4	0.0	1.6	50.0	5.6	0.0	2.0	49.0	5.9

(B) Experiments on 2^7 lattices where $\mathcal{H} = [-2^4, 2^4)^5 \times [0, 2^4)$ ($t = 6$, $\#\mathcal{H} = 2^{29}$).

	`globalntv` ($\ell = 3$)				`localntv` ($\ell = 3$)				`sparsentv` ($\ell = 3$)			
	min	med	max	mean	min	med	max	mean	min	med	max	mean
# ssvs	364				88				72			
# fbs	0	8.0	142	13.3	0	12.0	186	16.8	0	14.0	161	18.1
rel. err.	0.0	2.7	54.4	7.7	0.0	3.3	47.7	6.9	0.0	3.2	48.8	6.8

(C) Experiments on 2^7 lattices where $\mathcal{H} = [-2^4, 2^4)^5 \times [0, 2^4)$ ($t = 6$, $\#\mathcal{H} = 2^{29}$).

	`globalntv` ($\ell = 4$)				`localntv` ($\ell = 4$)				`globalntv` ($\ell = 5$)			
	min	med	max	mean	min	med	max	mean	min	med	max	mean
# ssvs	364				153				364			
# fbs	0	1.0	10	1.8	0	3.0	10	3.3	0	8.5	17	8.3
rel. err.	0.0	6.4	60	11.3	0.0	5.2	52.9	9.8	0.0	0.0	66.7	2.0

(D) Experiments on 2^7 lattices where $\mathcal{H} = [-2^4, 2^4)^5 \times [0, 2^4)$ ($t = 6$, $\#\mathcal{H} = 2^{29}$).

TABLE 4. Experiments on the three sieves: "# ssvs" corresponds to Criterion 5.1, "# fbs" to Criterion 5.1 and "rel. err." to Criterion 5.1.

The accuracy of the algorithms seems more than sufficient for the majority of the lattices, both in four and six dimensions. The maximal values of all the tables can be impressive, but occur only for a sufficiently small number of skewed lattices: since the enumeration in such lattices may be costly, it can be better to avoid them or at least, to be not too accurate.

In four dimensions, the accuracy is combined with a reasonable number of produced skew-small-vectors. The criteria do not help to determine which of the 2-level

`localntv` and `globalntv` is the most suitable algorithm. The running time estimations may help to decide. At level $\ell = 3$, the number of calls to the fall-back strategy can be an issue but may be under control in a careful implementation.

The situation is mitigated in dimension six. Except for the 2-level `sparsentv`, the number of skew-small-vectors is huge, which disqualify with this setting all the sieves at any level. In addition, the number of calls to the fall-back strategy at level $\ell = 2$ and $\ell = 3$ indicates that the produced nearly-transition-vectors are of poor quality. If dimension six sieving would be feasible, it will need more investigation; however, using cuboid search spaces is probably a too hard constraint that implies a hardness, or even an impossibility, for the sieving process. In addition, the initialization of the norms in higher dimensions implemented in CADO-NFS [43] is actually too slow for dimension larger than six by preserving a relative accuracy. It confirms the hardness of the relation collection above dimension four.

5.2. **Running time.** From the previous section, only four-dimensional sieving seems an option. We compare, at levels $\ell = 2$ and $\ell = 3$, the new sieves with the state-of-the-art sieve algorithms and also between themselves.

Comparison with the plane sieve. The 2-level `globalntv` and `localntv` are compared with the most efficient existing sieve algorithm, which is the (generalized) plane sieve. Our implementation of the plane sieve is however a bit incomplete: we implement the fall-back strategy of Section 4.4 without enforcing the coordinate k of the k–skew-small-vectors to be equal to 1. This implementation may be a bit faster than a complete plane sieve. On 2^{10} lattices, `globalntv` and `localntv` are faster than our generalized plane sieve, with `localntv` slightly faster than `globalntv`. Since the accuracy of the two heuristic sieve algorithms is quite good, both sieves must be consider as an alternative to the plane sieve.

Comparison of the new sieves. The 3-level `globalntv` is also compared with the 2-level `globalntv` and `localntv` on 2^{10} lattices. Unlike the previous comparisons, the results can be puzzling. Indeed, for lattices where the 3-level `globalntv` is expected to be efficient, the 2-level `localntv` is less than 1.5 time faster. Furthermore, the 2-level `localntv` is more than 3 time faster than the 2-level `globalntv`. Before explaining these results, we first remark that, in this situation, the three studied sieves algorithms share the same condition to use or not the fall-back strategy. The second remark comes from a detail of our implementation. Since accuracy is our main concern, Step 1b of the fall-back strategy in Section 4.2 sets $\boldsymbol{c}$ to one of the computed elements with the smallest coordinate k (i.e., the first element, since the list of k–nearly-transition-vectors is sorted by increasing coordinate k).

The 2-level `globalntv` and `localntv` produce more or less the same nearly-transition-vectors, despite different produced skew-small-vectors. The 3–skew-small-vectors are less numerous and have smaller coordinates with `localntv` than with `globalntv`. Then, if the for loop on the k–skew-small-vectors (Step 1a) fails to find an element in $\mathcal{H}$ in both sieves, and if the coordinate k of the first k–skew-small-vectors is the same for both sieves (this two situations often occur), `localntv` is faster than `globalntv`.

Between the 3-level `globalntv` and the 2-level `localntv`, the situation shares some of the observations made previously. However, this time, `globalntv` produces nearly-transition-vectors and skew-small-vectors of better quality than `localntv`: in some cases, `globalntv` is faster than `localntv`, but if the situations become the same as in the previous analysis, `localntv` stays faster. We believe that a careful

study of the different parts (especially how the linear combinations can produce useful vectors during the initialization of `globalntv` specialized in dimension four) of the algorithms will lead to an efficient implementation of the 3-level `globalntv`.

6. Conclusion

In this article we propose algorithms to sieve in any dimensions in the intersection of a lattice and a cuboid, which is one of the challenges we list to have a practical implementation of the $\text{NFS}_{>1}$ algorithms. These algorithms allow to report a large portion of the elements in the intersection, faster than the previous generic sieve algorithms. We provide a reference implementation of these algorithms, allowing us to highlight their advantages and drawbacks for the accuracy and efficiency of the enumeration, and demonstrate the practicability of these sieves for dimension four, and the hardness of sieving in dimension six and above.

In a near future, we plan to integrate these algorithms, specialized in dimension four, in the existing implementations of NFS_1 in CADO-NFS [43] and extend it to $\text{NFS}_{>1}$. It will help key size estimations for pairings [30, 3]. However, since a practical computation of the relation collection with $\text{NFS}_{>1}$ will be possible only with good polynomials f_0 and f_1, we also plan to study quality criteria for such NFS algorithms. Further work includes also enumeration in non-cuboid search space.

Acknowledgments. The authors are grateful to Pierrick Gaudry and Marion Videau for numerous discussions and reviews of preliminary versions of this work, as well as Aurore Guillevic and Shashank Singh for numerous discussions about NFS. We also thank Damien Stehlé and the referees whose remarks helped us to improve the presentation of our results.

References

1. S. Bai, C. Bouvier, A. Kruppa, and P. Zimmermann, *Better polynomials for GNFS*, Math. Comp. **85** (2016), no. 298, 861–873.
2. R. Barbulescu, *Algorithmes de logarithmes discrets dans les corps finis*, Ph.D. thesis, Université de Lorraine, 2013.
3. R. Barbulescu and S. Duquesne, *Updating key size estimations for pairings*, J. Cryptology **31** (2018), 1–39.
4. R. Barbulescu, P. Gaudry, A. Guillevic, and F. Morain, *Improving NFS for the discrete logarithm problem in non-prime finite fields*, EUROCRYPT 2015 (E. Oswald and M. Fischlin, eds.), LNCS, vol. 9056, Springer, 2015, pp. 129–155.
5. R. Barbulescu, P. Gaudry, and T. Kleinjung, *The Tower Number Field Sieve*, ASIACRYPT 2015 (T. Iwata and J. Cheon, eds.), LNCS, vol. 9453, Springer, 2015, pp. 31–55.
6. Y. Bistritz and A. Lifshitz, *Bounds for resultants of univariate and bivariate polynomials*, Linear Algebra and its Applications **432** (2010), no. 8, 1995–2005.
7. F. Boudot, *On Improving Integer Factorization and Discrete Logarithm Computation using Partial Triangulation*, Cryptology ePrint Archive, Report 2017/758, 2017.
8. J. Buhler, H. Lenstra, and C. Pomerance, *Factoring integers with the number field sieve*, The Development of the Number Field Sieve (A. Lenstra and H. Lenstra, eds.), Lecture Notes in Mathematics, vol. 1554, Springer, 1993, pp. 50–94.
9. H. Cohen, *A course in algorithmic algebraic number theory*, Graduate Texts in Mathematics, vol. 138, Springer, 2000, fourth printing.
10. D. Coppersmith, *Solving homogeneous linear equations over $GF(2)$ via block Wiedemann algorithm*, Math. Comp. **62** (1994), no. 205, 333–350.
11. J. Franke and T. Kleinjung, *Continued fractions and lattice sieving*, SHARCS 2005, 2005.
12. D. Freeman, M. Scott, and E. Teske, *A Taxonomy of Pairing-Friendly Elliptic Curves*, J. Cryptology **23** (2010), 224–280.

13. J. Fried, P. Gaudry, N. Heninger, and E. Thomé, *A Kilobit Hidden SNFS Discrete Logarithm Computation*, EUROCRYPT 2017 (J.-S. Coron and J. Nielsen, eds.), LNCS, vol. 10210, Springer, 2017, pp. 202–231.
14. P. Gaudry, L. Grémy, and M. Videau, *Collecting relations for the number field sieve in* $GF(p^6)$, LMS J. Comput. Math **19** (2016), no. A, 332–350.
15. J. Graver, *On the foundations of linear and integer linear programming I*, Mathematical Programming **9** (1975), no. 1, 207–226.
16. L. Grémy, A. Guillevic, F. Morain, and E. Thomé, *Computing discrete logarithms in* $GF(p^6)$, SAC 2017 (C. Adams and J. Camenisch, eds.), LNCS, vol. 10719, Springer, 2018, pp. 85–105.
17. A. Guillevic, *Faster individual discrete logarithms with the QPA and NFS variants*, Cryptology ePrint Archive, Report 2016/684, 2017.
18. K. Hayasaka, K. Aoki, T. Kobayashi, and T. Takagi, *An Experiment of Number Field Sieve for Discrete Logarithm Problem over* $GF(p^{12})$, Number Theory and Cryptography (M. Fischlin and S. Katzenbeisser, eds.), LNCS, vol. 8260, Springer, 2013, pp. 108–120.
19. ———, *A construction of 3-dimensional lattice sieve for number field sieve over* $\mathbb{F}_{p^n}$, Cryptology ePrint Archive, 2015/1179, 2015.
20. A. Joux and C. Pierrot, *Nearly sparse linear algebra and application to discrete logarithms computations*, Contemporary Developments in Finite Fields and Applications (A. Canteaut, G. Effinger, S. Huczynska, D. Panario, and L. Storme, eds.), World Scientific Publishing Company, 2016, pp. 119–144.
21. A. Joux and R. Lercier, *Improvements to the General Number Field Sieve for discrete logarithms in prime fields*, Math. Comp. **72** (2003), no. 242, 953–967.
22. A. Joux, R. Lercier, N. Smart, and F. Vercauteren, *The Number Field Sieve in the Medium Prime Case*, CRYPTO 2006 (C. Dwork, ed.), LNCS, vol. 4117, Springer, 2006, pp. 326–344.
23. T. Kim and R. Barbulescu, *Extended Tower Number Field Sieve: A New Complexity for the Medium Prime Case*, CRYPTO 2016 (M. Robshaw and J. Katz, eds.), LNCS, vol. 9814, Springer, 2016, pp. 543–571.
24. T. Kim and J. Jeong, *Extended Tower Number Field Sieve with Application to Finite Fields of Arbitrary Composite Extension Degree*, PKC 2017 (S. Fehr, ed.), LNCS, vol. 10174, Springer, 2017, pp. 388–408.
25. T. Kleinjung, *On polynomial selection for the general number field sieve*, Math. Comp. **75** (2006), no. 256, 2037–2047.
26. ———, *Polynomial Selection*, Slides presented at the CADO workshop on integer factorization, 2008, `http://cado.gforge.inria.fr/workshop/slides/kleinjung.pdf`.
27. T. Kleinjung, C. Diem, A. Lenstra, C. Priplata, and C. Stahlke, *Computation of a 768-Bit Prime Field Discrete Logarithm*, EUROCRYPT 2017 (J.-S. Coron and J. Nielsen, eds.), LNCS, vol. 10210, Springer, 2017, pp. 185–201.
28. A. Lenstra, *General purpose integer factoring*, Topics in Computational Number Theory Inspired by Peter L. Montgomery (J. Bos and A. Lenstra, eds.), Cambridge University Press, 2017, pp. 116–160.
29. A. Lenstra and E. Verheul, *The XTR public key system*, CRYPTO 2000 (M. Bellare, ed.), LNCS, vol. 1880, Springer, 2000, pp. 1–19.
30. A. Menezes, P. Sarkar, and S. Singh, *Challenges with Assessing the Impact of NFS Advances on the Security of Pairing-Based Cryptography*, Mycrypt 2016 (R. Phan and M. Yung, eds.), LNCS, vol. 10311, Springer, 2017, pp. 83–108.
31. B. Murphy, *Polynomial selection for the number field sieve integer factorisation algorithm*, Ph.D. thesis, The Australian National University, 1999.
32. S. Onn, *Theory and Applications of n-Fold Integer Programming*, Mixed Integer Nonlinear Programming (J. Lee and S. Leyffer, eds.), IMA, vol. 154, Springer, 2012, pp. 559–593.
33. J. Pollard, *The lattice sieve*, The Development of the Number Field Sieve (A. Lenstra and H. Lenstra, eds.), Lecture Notes in Mathematics, vol. 1554, Springer, 1993, pp. 43–49.
34. C. Pomerance, *Analysis and comparison of some integer factoring algorithms*, Computational Methods in Number Theory (H. Lenstra and R. Tijdeman, eds.), Mathematical Centre Tracts, vol. 154, Mathematish Centrum, 1982, pp. 89–139.
35. ———, *A Tale of Two Sieves*, Notices Amer. Math. Soc **43** (1996), 1473–1485.
36. K. Rubin and A. Silverberg, *Torus-based cryptography*, CRYPTO 2003 (D. Boneh, ed.), LNCS, vol. 2729, Springer, 2003, pp. 349–365.

37. P. Sarkar and S. Singh, *A General Polynomial Selection Method and New Asymptotic Complexities for the Tower Number Field Sieve Algorithm*, ASIACRYPT 2016 (J. Cheon and T. Takagi, eds.), LNCS, vol. 10031, Springer, 2016, pp. 37–62.
38. ———, *A Generalisation of the Conjugation Method for Polynomial Selection for the Extended Tower Number Field Sieve Algorithm*, Cryptology ePrint Archive, Report 2016/537, 2016.
39. ———, *New Complexity Trade-Offs for the (Multiple) Number Field Sieve Algorithm in Non-Prime Fields*, EUROCRYPT 2016 (M. Fischlin and J.-S. Coron, eds.), LNCS, vol. 9665, Springer, 2016, pp. 429–458.
40. ———, *Tower number field sieve variant of a recent polynomial selection method*, Cryptology ePrint Archive, Report 2016/401, 2016.
41. O. Schirokauer, *Discrete logarithms and local units*, Philos. Trans. Roy. Soc. A **345** (1993), no. 1676, 409–423.
42. ———, *Virtual logarithms*, J. Algorithms **57** (2005), 140–147.
43. The CADO-NFS Development Team, *CADO-NFS, an implementation of the number field sieve algorithm*, 2018, development version, `http://cado-nfs.gforge.inria.fr/`.
44. P. Zajac, *Discrete logarithm problem in degree six finite fields*, Ph.D. thesis, Slovak University of Technology, 2008, `http://www.kaivt.elf.stuba.sk/kaivt/Vyskum/XTRDL`.
45. Y. Zhu, J. Zhuang, C. Lv, and D. Lin, *Improvements on the individual logarithm step in extended tower number field sieve*, Cryptology ePrint Archive, Report 2016/727, 2016.

Univ Lyon, CNRS, ENS de Lyon, Inria, Université Claude Bernard Lyon 1, LIP UMR 5668, F-69007 Lyon, France

Email address: `firstname.lastname@ens-lyon.fr`