



Symbolic Computation and Complexity Analyses for Number Theory and Cryptography

Aude Le Gluher

► To cite this version:

Aude Le Gluher. Symbolic Computation and Complexity Analyses for Number Theory and Cryptography. Cryptography and Security [cs.CR]. Université de Lorraine, 2021. English. NNT : 2021LORR0245 . tel-03564208

HAL Id: tel-03564208

<https://hal.univ-lorraine.fr/tel-03564208>

Submitted on 10 Feb 2022

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.



AVERTISSEMENT

Ce document est le fruit d'un long travail approuvé par le jury de soutenance et mis à disposition de l'ensemble de la communauté universitaire élargie.

Il est soumis à la propriété intellectuelle de l'auteur. Ceci implique une obligation de citation et de référencement lors de l'utilisation de ce document.

D'autre part, toute contrefaçon, plagiat, reproduction illicite encourt une poursuite pénale.

Contact : ddoc-theses-contact@univ-lorraine.fr

LIENS

Code de la Propriété Intellectuelle. articles L 122. 4

Code de la Propriété Intellectuelle. articles L 335.2- L 335.10

http://www.cfcopies.com/V2/leg/leg_droi.php

<http://www.culture.gouv.fr/culture/infos-pratiques/droits/protection.htm>

Symbolic Computation and Complexity Analyses for Number Theory and Cryptography

THÈSE

présentée et soutenue publiquement le 7 décembre 2021

pour l'obtention du

Doctorat de l'Université de Lorraine
(mention informatique)

par

Aude Le Gluher

Composition du jury

<i>Président :</i>	Karim BELABAS	Professeur des Universités, Université de Bordeaux
<i>Rapporteurs :</i>	Karim BELABAS Laurent IMBERT	Professeur des Universités, Université de Bordeaux Directeur de recherche CNRS, Université de Montpellier
<i>Examineurs :</i>	Cécile DARTYGE Vanessa VITSE	Maîtresse de conférences, Université de Lorraine Maîtresse de conférences, Université Grenoble-Alpes
<i>Invité :</i>	Thorsten KLEINJUNG	Chercheur, Ecole polytechnique fédérale de Lausanne
<i>Encadrants :</i>	Pierre-Jean SPAENLEHAUER Emmanuel THOMÉ	Chargé de recherche, INRIA Nancy - Grand Est Directeur de recherche, INRIA Nancy -Grand Est

Remerciements

Le nombre de personnes impliquées dans la réussite d'une thèse est grand devant le nombre de synonymes de "remercier" en français. Par conséquent, le discours qui suit sera répétitif. Compte tenu d'autre part que la mémoire d'une Aude est hautement faillible, il oubliera également de mentionner de nombreuses personnes, qui, je l'espère, n'en prendront pas ombrage.

Tout d'abord, je remercie du fond du coeur mes deux directeurs de thèse, Emmanuel et Pierre-Jean, sans qui le présent document n'aurait pas vu le jour. Pour avoir observé d'autres styles d'encadrement, je me considère extrêmement privilégiée d'avoir pu bénéficier du votre, remarquable tant pour les discussions intellectuellement nourrissantes qui ont émaillé ces trois années que pour votre bienveillance et votre disponibilité.

Merci à Laurent Imbert et Karim Belabas pour avoir accepté de relire cette thèse, vos retours ont grandement contribué à améliorer la qualité de ce manuscrit. I would also like to thank Vanessa Vitse, Cécile Dartyge and Thorsten Kleinjung who agreed to be on my thesis comitee.

Merci ensuite à tous les membres de l'équipe Caramba, collectivement responsables de l'excellente ambiance dans laquelle s'est déroulée ma thèse et ce malgré les années coviesques dont nous avons dû faire les frais. Vous m'avez fait grandir, à la fois intellectuellement et humainement. Le canal troll du mattermost me manquera beaucoup. Merci entre autres à tous ceux qui ont pris le temps de répondre à mes questions diverses, variées et souvent impromptues, en particulier Paul pour tes réponses précises sur les arcanes du fonctionnement du filtre et Pierrick pour celles sur l'implémentation de l'algorithme de simulation dudit filtre. Merci à tous les doctorants et stagiaires de Caramba avec qui j'ai pu cheminer plus ou moins longtemps et en particulier à ma cobureau Gabrielle qui a dû supporter plus d'une fois mes discours grandilogogants.

Un grand merci à Anne, Horatiu et Marine pour m'avoir donné l'opportunité d'enseigner dans vos classes. Ces expériences me sont d'autant plus précieuses qu'elles me permettent aujourd'hui de construire moi-même mes propres cours en classes préparatoires. Merci aux doctorants et stagiaires de l'équipe Pesto pour les moments passés ensemble lors de nos pique-niques des doctorants du mercredi et les goûters avec pâte à tartiner artisanale. Merci aussi à tous les non-permanents de Larsen - entre autres - pour les moments de détente autour d'une partie d'Avalon sur les temps de midi (je ne suis pas Merlin).

Je mesure la chance que j'ai eue de pouvoir effectuer ma thèse tout proche de mes parents. Les dimanches midi passés ensemble pendant ces trois années sont des moments que je chérirai longtemps. Merci papa pour les coups de main sur les questions conkitu... c'est dur l'analyse. J'espère que tes élèves ne découvriront jamais qui est à l'origine de certaines questions de tes devoirs, je risquerais des représailles.

Merci enfin à toi, Vlad, pour tout. Au delà du fait qu'une partie des preuves techniques et des scripts de simulation de cette thèse te doit beaucoup, merci de m'avoir aidé à garder le cap dans mes moments de découragement, de m'avoir soutenue même lorsque le stress me rendait exaspérante et d'avoir toujours été à mes côtés. Je souhaite pouvoir te rendre la pareille.

Contents

Introduction	1
I Preliminaries	7
1 Background on the Number Field Sieve	9
1.1 Integer factorization and discrete logarithms	9
1.1.1 Factorization of large integers	9
1.1.2 Computation of discrete logarithms in finite fields	11
1.2 Overview of the number field sieve (NFS)	12
1.2.1 Mathematical background	12
1.2.2 Main steps of NFS	13
1.2.3 NFS in practice: implementations and records	19
1.2.4 NFS in theory: asymptotic complexity	20
1.3 Focus on the filtering step in NFS	21
1.3.1 Purge step	23
1.3.2 Merge step	24
2 Background on Riemann-Roch spaces	27
2.1 Riemann-Roch spaces	27
2.1.1 Mathematical background	27
2.1.2 Riemann-Roch spaces, Riemann-Roch theorem	30
2.1.3 Uses of Riemann-Roch spaces	31
2.2 Computation of Riemann-Roch spaces	32
2.2.1 Algorithmic state-of-the-art	32
2.2.2 Geometric algorithms and the residue theorem	33
II Complexity and simulation of the Number Field Sieve	35
3 Asymptotic complexity of NFS	37
3.1 A minimization problem	37
3.2 First order resolution	40
3.3 Expansion of smoothness probabilities	44
3.3.1 Useful classes of functions	44
3.3.2 Hildebrand and De Bruijn formulas	46
3.3.3 Asymptotic development of smoothness probabilities	47
3.4 Further terms in the asymptotic expansion of NFS complexity	53
3.4.1 Two new proven terms in the asymptotic development	53
3.4.2 Generalization of the method, algorithms	57
3.4.3 General form of the asymptotic expansion of NFS complexity	60
3.5 Relevance of the use of asymptotic expansions for practical estimates	61

4	Simulation of the filtering step in NFS	65
4.1	Overview of the ESTIMATEMATSIZE algorithm	65
4.1.1	Generation of fake relations	66
4.1.2	Use of shrink factors	68
4.1.3	Summary of the ESTIMATEMATSIZE algorithm	69
4.2	Base results for ESTIMATEMATSIZE experiments	70
4.3	Impact of the number of example special- q	72
4.4	Impact of the shrinking step	75
4.4.1	Estimations based on real shrunk relations	75
4.4.2	Suggested improvements for the shrinking step	80
4.5	Conclusion	84
III	Computation of Riemann-Roch spaces	85
5	A fast geometric algorithm to compute Riemann-Roch spaces	87
5.1	Overview of the algorithm	87
5.1.1	Hypothesis on the curve and divisors	87
5.1.2	Main steps of the algorithm	88
5.1.3	Correction of the algorithm	89
5.2	Arithmetic on divisors	90
5.2.1	Data structures	90
5.2.2	Existence of primitive element representations and nodal representations . .	94
5.2.3	Operations on smooth divisors	98
5.3	Subroutines of the main algorithm	103
5.3.1	Interpolation subroutine	103
5.3.2	Computing the smooth part of the principal divisor associated to a regular function on the curve	104
5.3.3	Computing the linear space of regular functions of bounded degree having prescribed zeros	107
5.4	Analyses of the algorithm	108
5.4.1	Asymptotic complexity	108
5.4.2	Lower bounds on the probability of success	110
5.4.3	Implementation and experimental results	115
5.4.4	Conclusion	116
	Bibliography	119
	Résumé en français	127
	Abstract	133
	Résumé	133

Introduction

Public-key cryptography

One main goal of cryptography is to build and analyze protocols that allow two entities to communicate securely, even when they do so through an insecure channel that could be eavesdropped. The general framework of such a protocol is the following: The sender, traditionally called Bob, encrypts his message using a key K_0 , sends the now unintelligible message to the receiver, Alice, who decrypts the message using a key K_1 . Symmetric cryptography designs and analyzes protocols in which $K_0 = K_1$ (or at least, K_1 can easily be deduced from K_0). Anyone knowing K_0 can thus encrypt messages instead of Bob or decrypt messages instead of Alice, which means that if Alice and Bob want to communicate secretly, they must make sure that they are the only ones to know the key K_0 . This leads to the uncomfortable situation where Alice and Bob must already share a secret, their key, before exchanging secrets.

In the 70s, Merkle [Mer78] and Diffie and Hellman [DH76] described an elegant workaround to this problem which led to the birth of asymmetric cryptography, also known as public-key cryptography. In asymmetric cryptographic protocols, the keys K_0 and K_1 are mathematically linked but recovering K_1 from K_0 is extremely difficult. Thus, in order to send a message to Alice, Bob encrypts it with Alice's so-called public key K_0 , which Alice previously broadcasted, and Alice can decrypt it using her private key K_1 that no other than her should know. An attacker does know the public key K_0 , but contrary to what happens in a symmetric setup, he cannot use it to retrieve the private key K_1 . In order to make these protocols practical, we now need functions that are easy to compute, so that encrypting a message is easy, but hard to invert except with some special knowledge (the key K_1), so that no one can decrypt a message except its intended receiver. Such functions are called trapdoor one-way functions and they are at the heart of asymmetric protocols.

Historically, two candidate functions emerged: the multiplication of prime integers and modular exponentiation. The latter consists in computing $g^x \bmod N$ when g, x and a prime N are known. The inversion of these functions is respectively integer factorization and discrete logarithm computation. The latter means recovering x from the knowledge of N, g and g^x using the same notations as above. In fact, the notion of discrete logarithm can be extended to a more general setup: If G is a finite cyclic group and g is a generator of G , computing a discrete logarithm of $h \in G$ means finding an element x such that $h = g^x$. In the following, we will focus on discrete logarithm computations when G is a multiplicative subgroup of a finite field.

Both of these operations are mathematical problems that have been studied for a while and are believed to be difficult to perform. This motivated the use of both functions in now widely-spread asymmetric cryptosystems, such as RSA [RSA78], the Diffie-Hellman key exchange [DH76] or ElGamal encryption and signature schemes [ElG85]. The interest for discrete logarithm computations has also been renewed lately with the development of pairing-based protocols [BF01, PS06, BBS04]. In order to assess the safety of these cryptosystems, the following question must be answered: How hard is it to factor integers or compute discrete logarithms in finite fields? The answer to this question is of crucial importance to regulatory bodies as it allows them to publish recommendations on key sizes for a variety of deployed protocols. Indeed, the size of the key K_0

used in the aforementioned cryptosystems dictates the size of the input that must be factored (or of which the discrete logarithm must be computed) to retrieve K_1 . To guarantee the security of these cryptosystems, they must thus be used with keys whose sizes ensure that the underlying factorization or discrete logarithm computation cannot be easily solved.

To answer the aforementioned question, we can turn to the study of algorithms designed to factor integers or compute discrete logarithms in finite fields. In this thesis, we will mostly be focused on the time complexity of these algorithms, which depends on the size of its input. In the 90's, Shor introduced quantum algorithms to solve both problems in polynomial time [Sho97a] which would indicate that these problems are actually easy to tackle in a quantum setup. But despite recent progress, we are quite far from reaching quantum computers capable of large scale computations. In this thesis, we choose not to leave the classical setup.

In this context, the Number Field Sieve (NFS for short) is, to this day, the fastest algorithm that allows to address both integer factorization and discrete logarithm computations in finite fields. Several implementations of this algorithm have been made. To evaluate the resources needed for this algorithm to solve either of the two problems, several approaches are possible. One, is to use NFS implementations to actually measure the computing time needed on inputs of a given size. But this method roughly comes down to feeding NFS implementations with larger and larger inputs: For each input for which the computation finishes, we can positively say that the input had a size that made integer factorization or discrete logarithm computation feasible, while for input sizes that have not yet been tested or for which the computation did not end for now, we cannot conclude. This can be mitigated by trying and estimating the hardness of our problems on larger inputs from the results on smaller inputs but such extrapolations can be complex. Another approach is to *simulate* NFS. The idea is to design an efficient algorithm that will estimate the time required by an implementation of NFS to finish a computation on a given input, without actually performing the computation. Though appealing, many obstacles lie in the way of this method as finding good models of the operations performed in NFS is tricky and in practice the algorithm involves many parameters — in addition, they are often not independent — that can greatly influence the computing time. This explains why the present standard to estimate NFS computing times, hence to say for which input sizes factoring and discrete logarithm computations is hard, is to fall back on the asymptotic complexity of the algorithm, which seems more manageable.

We talked at length about the Number Field Sieve, whose name comes partly from the number field algebraic structure. Number fields have a lot in common with function fields of algebraic curves over finite fields. While a number field is a finite extension of \mathbb{Q} which can always be built by quotienting $\mathbb{Q}[X]$ by an irreducible polynomial, function fields are finite extensions of $\mathbb{F}_q(T)$, the field of rational functions in one variable over the finite field with q elements, and can be built based on a given algebraic curve [Ros02, MW83]. In fact the similarities between these two algebraic structure led in the 30s to the idea of uniting them under a common denomination: global fields. Despite this deep analogy, the vocabulary and techniques used in both settings are quite different. Still, many parallels can be drawn between theorems and structures in both settings and it is often fruitful, when a result holds for one of these two families of fields, to try and develop parallel methods in the other.

Part of this thesis will focus on the study of some vector spaces in function fields: Riemann-Roch spaces. They are spaces of rational functions on a curve whose poles and zeros are constrained. Among the many uses of these spaces, they allow the effective computation of the group law in the Jacobian of a curve which has many interests in number theory and algebraic geometry. Interestingly, Riemann-Roch spaces are also at the heart of some cryptographic constructions. Indeed, they can be used to build efficient algebraico-arithmetic error-correcting codes as Goppa first envisioned [Gop83, Gop77]. Practical applications however require a way to efficiently build bases of these vector spaces.

Contributions

Refined analysis of the asymptotic complexity of NFS.

It is known (see e.g. [BLP93]) that the complexity of NFS to factor an integer N , under several classical heuristics is given by

$$\exp\left(\sqrt[3]{\frac{64}{9}}(\log N)^{1/3}(\log \log N)^{2/3}(1 + \xi(N))\right)$$

where $\xi(N)$ is an unknown function such that $\xi(N) \in o(1)$. A similar formula holds in the context of discrete logarithm computation with N replaced by p , the size of the finite field in which the discrete logarithm computation takes place. To this day, the widely accepted standard to assess the computational power needed to factor integers and compute discrete logarithm in finite fields relies on a simplification of this formula: It assumes that $\xi(N) = 0$, and uses a computational record to set a proportionality ratio. Though the brutal simplification $\xi(N) = 0$ is mathematically dubious and was originally considered as such, it gradually became a completely acceptable assumption, mostly because of a lack of a better alternative to evaluate NFS computing times on a given input. In particular, we stress that recommendations for key sizes in deployed RSA-based cryptography are based on this method [NIS19, Sec. 7.5], [ANS14, Sec. B.2.2] [ENI14, Table 3.1].

In this thesis, we compute precise asymptotic expansions for the function ξ , leading to refined formulas for the asymptotic complexity of NFS. More precisely, we:

1. Re-establish from scratch the minimization problem of which NFS complexity is solution and re-prove that solving this problem yields the above first-order formula. The latter result is already common knowledge but its proofs often rely on unneeded hypotheses that we all avoid here.
2. Establish new terms for the asymptotic expansion of the function ξ . We prove that:

$$\xi(N) \times \log \log N = \frac{4}{3} \log \log \log N + \left(-2 \log 2 + \frac{\log 3}{6} - 2\right) + o(1).$$

3. Show more generally that the asymptotic expansion of ξ can actually be extended to a bivariate series evaluated at $\log \log \log N / \log \log N$ and $1 / \log \log N$ and provide algorithms that allow to output the coefficients of a bivariate series $\mathbf{A} \in \mathbb{Q}(\log 2, \log 3)[[X, Y]]$ with constant term 1 such that the heuristic asymptotic complexity of NFS $C(N)$ satisfies :

$$\frac{\log C(N)}{\sqrt[3]{\frac{64}{9}}(\log N)^{\frac{1}{3}}(\log \log N)^{\frac{2}{3}}} = \mathbf{A}^{(n)}\left(\frac{\log \log \log N}{\log \log N}, \frac{1}{\log \log N}\right) + o\left(\frac{1}{(\log \log N)^n}\right)$$

where $\mathbf{A}^{(n)}$ denotes the truncation of the series \mathbf{A} to total degree n . An implementation of these algorithms, that allowed to compute more than a hundred terms of the asymptotic expansion of NFS complexity, is available at:

https://gitlab.inria.fr/NFS_asymptotic_complexity/simulations

4. Study the converging behaviour of the series associated to ξ and find that the range in which $\xi(N)$ can be replaced by one of its truncations is beyond $N > \exp(\exp(22)) \approx 2^{5171935985}$ which is far greater than the usual values used in practical cryptographic applications (where $N \leq 2^{20000}$ at the very most). Thus, replacing ξ by any of its truncations in the cryptographically relevant range comes down to replacing a series by its first terms in a range where the series diverges. It means caution is required when using truncated formulas for the complexity of NFS in order to extrapolate key sizes for cryptography and stresses the importance of simulation tools to estimate this complexity.

In summary, this work shows that it is possible to know the asymptotic complexity of NFS with as much precision as desired, though it unfortunately does not translate into more precision for estimations of practical computing times. The results presented here expand those of the following paper, precise some technicalities, and fix the arguments needed to establish the minimization problem of which NFS complexity is solution using GRH.

[LGST21] Refined Analysis of the Asymptotic Complexity of the Number Field Sieve, with Pierre-Jean Spaenlehauer and Emmanuel Thomé, published in Mathematical Cryptology in 2021.

Study of a simulation tool for the filtering step of NFS.

As emphasized in the previous paragraph, using truncated formulas for NFS asymptotic complexity in order to deduce practical computing times for this algorithm is a quick and easy method to produce estimations that, regrettably, are possibly very far off. This obviously motivates the idea to find other ways to produce more accurate estimations of NFS computing times on an input of a given size. Actually using an implementation of NFS on this input does produce very precise estimations, but this method is by definition unworkable for sufficiently large inputs. So in addition to the precision of its estimations we would like a method that takes a reasonable time to produce them. This leads to the idea of simulating NFS: We would like to model the main steps of this algorithm to try and estimate the time required to run them without doing a full computation.

Simulation tools for NFS are few, but recently one of them was designed and used during the record computations of RSA-240 and DLP-240 presented in [BGG⁺20a]. It was implemented using the NFS implementation CADO-NFS as a base. This simulation algorithm, called ESTIMATEMATSIZE throughout this thesis, takes as an input a set of NFS parameters, simulates the so-called filtering step of NFS, and outputs a matrix whose size and density partly rules the computing time of the so-called linear algebra step of NFS. Thus, it allows to simulate part of the NFS algorithm. The authors in [BGG⁺20a] used this simulation tool to test sets of parameters and pinpoint which ones were best suited to speed up their record computations.

In this thesis, we study the ESTIMATEMATSIZE algorithm from an experimental viewpoint. The end goal of this work is to build a fast prediction tool for part of the NFS algorithm which is ensured to be reliable. More precisely we:

1. Give a precise description of the modelization for the filtering step of NFS that is used in the preexisting algorithm ESTIMATEMATSIZE.
2. Develop efficient tools and methods to retrospectively analyze this algorithm, and in particular to uncouple as best as possible the impact of each modelization simplification on the accuracy of the predictions.
3. Investigate the performance of this simulation algorithm using large numbers of diversified simulation-related parameters. This allowed to better assess its reliability and to reveal trends in the behaviour of the ESTIMATEMATSIZE algorithm, some of which remain unexplained to this day.
4. Propose modifications — some of which have been implemented — to the original modelization, in particular to mitigate some brutal approximations in the original implementation.

This work led to contributions to the CADO-NFS software. For the time being, this study remains mostly descriptive. It further highlights the fact that simulating NFS, even partly, is complex, as a wide range of interdependent parameters influence computing times: Together with the simulation-related parameters, it makes it difficult to explain the unexpected behaviours of the simulation algorithm that have been uncovered during this thesis. However this study also showed that thousands of estimations done with the ESTIMATEMATSIZE implementation available in the CADO-NFS software always fell within a 20% margin of the expected values. Thus, it allows to assert with some confidence that this simulation algorithm is already quite reliable, which is good news for the future of NFS simulation.

Design and implementation of a fast algorithm to compute Riemann-Roch spaces.

If \mathcal{C} is a projective curve of genus g defined over a field K , a divisor of \mathcal{C} is a formal finite sum of points of \mathcal{C} with integer coefficients. Any divisor D can be written $D = D_+ - D_-$ where the integer coefficients of D_+ and D_- are positive. Simply put, the Riemann-Roch space $L(D)$ associated to a divisor $D = D_+ - D_- = \sum_{i=1}^n n_i P_i - \sum_{j=1}^k m_j Q_j$ is the K -vector space of functions f on \mathcal{C} — that is, rational fractions whose numerators and denominators are considered modulo the polynomial describing \mathcal{C} — such that f can only have poles in a point P_i with multiplicity at most n_i and necessarily has zeros in all the points Q_j with multiplicity at least m_j .

The computation of a basis for these vector spaces is a subroutine used in several areas of computer science and computational mathematics, most prominently in the construction of algebraico-geometric error-correcting codes and in the computation of the group law in the Jacobian of a curve. Two families of algorithms were designed to compute such bases.

Arithmetic approaches rely on fast algorithms for algebraic function fields. The reference algorithm in this category is by Hess [Hes02]: It allows to compute Riemann-Roch spaces associated to a divisor $D = D_+ - D_-$ for any curve \mathcal{C} regardless of its singularities. It is polynomial in the input size, that is in $\deg \mathcal{C}$ and $\deg D_+$ where $\deg \mathcal{C}$ is the degree of the polynomial that defines the curve \mathcal{C} and $\deg D_+$ is the number of points in the divisor D_+ counted with multiplicities, which is a relevant measure of the size of D as $L(D) = \{0\}$ if $\deg D_- > \deg D_+$. It has been implemented in the MAGMA computer algebra system.

Geometric approaches, whose correction rests on the residue theorem of Brill and Noether [Ful08, Section 8.1], have been initiated by Goppa [Gop83]. During the 90's, several versions of this algorithm have emerged, notably an algorithm by Huang and Ierardi [HI94] which deterministically computes Riemann-Roch spaces of plane ordinary curves within $O(\deg(\mathcal{C})^6 \deg(D_+)^6)$ operations in K and a first implementation of a Brill-Noether based algorithm by Haché [Hac95] a year later. The important special case of algorithms geared towards computing the group law on Jacobians of curves was also tackled with geometric methods. Volcheck [Vol94] described an algorithm with complexity $O(\max(\deg(\mathcal{C}), g)^7)$ to solve this problem and Khuri-Makdisi [KM07] later achieved a complexity in $O(g^\omega)$ where ω is a feasible exponent for matrix multiplication. At the beginning of this thesis, this was the best known complexity for computing the group law on Jacobians of general curves.

In this thesis, we design a fast probabilistic geometric algorithm for computing Riemann-Roch spaces on nodal plane curves defined over sufficiently large perfect fields. Nodal curves are the simplest singular curves as their singularities are all ordinary of order two, but they are generic enough as any algebraic curve admits a nodal plane model up to a birational map when the base field is large enough [ACGH85, Appendix A]. More precisely, we:

1. Propose a polynomial representation for divisors inspired by Mumford coordinates and by representations of algebraic sets by primitive elements.
2. Use this representation, together with the Brill-Noether scheme that underlies geometric algorithms, to build and prove an algorithm for computing Riemann-Roch spaces. We stress that the building blocks of our algorithm — namely arithmetic on bivariate polynomials and linear algebra — are both well studied topics for which fast implementations exist. Thus, our algorithm is both efficient and it can be easily implemented.
3. Prove that the complexity of this algorithm is bounded by $O(\max(\deg(\mathcal{C})^{2\omega}, \deg(D_+)^{\omega}))$ where ω is a feasible exponent for matrix multiplication. In particular, computing the group law on Jacobians of smooth curves using our algorithm takes at most $O(g^\omega)$ operations in K (in this context, $\deg(D_+) = O(g)$ and $\deg(\mathcal{C}) = O(\sqrt{g})$ by the genus-degree formula) which was the bound reached by Khuri-Makdisi.
4. Prove that its failure probability is bounded by $O(\max(\deg(\mathcal{C})^4, \deg(D_+)^2)/|\mathcal{E}|)$, where \mathcal{E} is a finite subset of K in which we pick elements uniformly at random, provided that a few mild

assumptions on the input are satisfied. We also provide a decision algorithm that checks if these assumptions — which roughly require that the impact of the singularities of \mathcal{C} during the computation is minimal — hold.

5. Give a C++/NTL implementation of our algorithm, which is available at

`https://gitlab.inria.fr/pspaenle/rrspace`

and provide experimental data which indicate that our prototype software is competitive with the reference implementation in the MAGMA computer algebra system.

Our algorithm has since been enhanced by Abelard, Lecerf and Couvreur [ACL20]. They achieved theoretical sub-quadratic complexity for the computation of Riemann-Roch spaces by replacing general linear algebra by structured linear algebra, though they do not provide an implementation for this algorithm. The results obtained during this thesis are reported in the following paper.

[LGS20] A Fast Randomized Geometric Algorithm for Computing Riemann-Roch spaces, with Pierre-Jean Spaenlehauer, published in Mathematics of Computation in 2020.

Plan of the thesis

In a first chapter, we detail some key applications of factoring integers and computing discrete logarithms before presenting the main steps of the Number Field Sieve algorithm, which allows to solve both of these problems. Special attention was given to the so-called filtering step of this algorithm. This step is at the core of the simulation tools that we investigate in Chapter 4. The second chapter lays the groundwork for Chapter 5 by introducing Riemann-Roch spaces and their uses, as well as a general geometric method attributed to Brill and Noether to compute bases of these spaces that will be one of the cornerstones of the algorithm described in this last chapter.

Chapters 3, 4 and 5 are devoted to presenting our contributions, both in the analysis of NFS computing times for the two former, and in the computation of Riemann-Roch spaces for the latter. Chapter 3 focuses on our results regarding the asymptotic complexity analysis of NFS. Chapter 4 leaves theoretical analyses behind to delve into the description and the assessment of the simulation tool ESTIMATEMATSIZE. Finally, in Chapter 5, we present our results for the computation of Riemann-Roch spaces.

Part I

Preliminaries

Chapter 1

Background on the Number Field Sieve

This chapter motivates the computation of integer factorization and discrete logarithms in finite fields by presenting a few salient applications in cryptography in which the hardness of the aforementioned problems is crucial. It then presents an algorithm that solves both problems: the number field sieve, NFS for short. The main steps of this algorithm are explained in order to provide the framework needed for the analyses in Chapter 3. Special care is finally dedicated to detailing the so called filtering step of NFS: It gives foundations to understand Chapter 4.

1.1 Integer factorization and discrete logarithms

Factoring integers and solving discrete logarithms in finite fields are two fundamental problems in computational number theory, which are core routines for a very large range of applications. Perhaps their most prominent and critical use is the fact that the security of many currently deployed cryptosystems — e.g. RSA, finite field Diffie-Hellman, ElGamal — relies directly on their computational difficulty. This is one of the reasons why the development of algorithms for solving these two problems and the analysis of their complexity are central topics in computational mathematics. In this section we introduce both problems, highlight why solving them is of crucial interest and present algorithms designed to solve them.

1.1.1 Factorization of large integers

The search for an efficient factorization technique predates its importance in cryptography. Factoring an integer N means finding all the prime divisors of N . The problem is rather simple to introduce, yet it has stood the test of time remarkably well. Besides it is linked to a number of topics such as the study of perfect numbers for instance. Perfect numbers are integers that are equal to the sum of their non trivial divisors, like 24. Questions about perfect numbers and in particular “Is there an odd perfect number ?” actually require the knowledge of the factorization of many integers of the form $b^n \pm 1$. The Cunningham Project precisely provides tables of the prime factors for these integers with small values for b . People had been publishing tables of factors of $b^n \pm 1$ for a time but all of them were assembled in a book [CW25] in 1925 by Cunningham and Woodall. The task of collecting factorizations for these numbers continued after Cunningham’s death (see [BLS⁺02]) and is still being pursued today. Besides the prominent Cunningham Project, integer factorization has many other uses in computational number theory as Wagstaff emphasizes in his book [Wag13].

The interest for factorization was greatly renewed with the introduction of the cryptosystem now called RSA. Described by Rivest, Shamir and Adleman in [RSA78] the security of this scheme rests on the difficulty to factor integers. We briefly present RSA below to stress why.

RSA encryption scheme

RSA is an asymmetric scheme that allows two individuals, classically named Alice and Bob to communicate secretly using an unsecure channel. To do so, it describes a method to compute a private and a public key for Alice and how to use them so that Bob can securely send a message to Alice.

Building the keys starts with Alice choosing two large primes p and q roughly of the same size and that Alice keeps secret. Alice computes the integers $N = pq$ and $\varphi(N) = (p-1)(q-1)$ where φ is Euler's totient function. She then chooses an exponent e coprime to $\varphi(N)$ and computes an integer d such that $ed \equiv 1 \pmod{\varphi(N)}$ thanks to Euclid's algorithm. She keeps her private key (p, q, d) secret and broadcasts her public key (N, e) in such a way that (N, e) is guaranteed to be Alice's: This usually amounts to trusting a given authority to link Alice to her public key.

If Bob wants to send her a message $m \in (\mathbb{Z}/N\mathbb{Z})^*$, he encrypts it by computing $c = m^e \pmod{N}$. The decryption proceeds as follows: Alice computes $c^d \pmod{N}$. Since d is a multiplicative inverse of e modulo $\varphi(N)$, there is an integer k such that $ed = 1 + k\varphi(N)$ and the Fermat-Euler theorem ensures that $c^d = m^{ed} = m \times m^{\varphi(N)k} \pmod{N} = m \times 1 \pmod{N}$.

A straightforward attack on this cipher is to factor the integer N in order to recover the private key. Attacks on the whole protocol are also possible, such as man-in-the-middle attacks: When retrieving Alice's public key, Bob could be fooled into thinking that an eavesdropper Eve's key is actually Alice's. He would then send his message to Eve while being convinced that he writes to Alice. Eve could decrypt the message intended for Alice, read it, forward it using Alice's actual public key, and Alice would not be aware that Eve spied on her private messages with Bob.

Factorization methods

The simplest factorization algorithm is trial division. It consists in systematically testing whether N is divisible by the primes below \sqrt{N} and has an exponential complexity in the size of N , namely $\log N$. Although it seems rudimentary, it is actually quite efficient to factor small integers. Faster algorithms allow to find factors of N in a sub-exponential time. The $p-1$ method described by Pollard in [Pol74] for instance is especially fast but only on inputs that have specific types of factors. The elliptic curve factorization method (ECM for short) introduced by Lenstra in [Len87] builds on this method to reach heuristically subexponential complexity. Those algorithms are not particularly well suited to factor random RSA-like integers however. But they can still be of great use, for instance by being used as subroutines in other factorization algorithms; besides this is also the case for trial division. In order to tackle the factorization of RSA-like integers, several methods were developed such as the Continued Fraction Factoring Algorithm (CFRAC) introduced by Lehmer and Powers in 1931 [LP31] and later enhanced by Morrison and Brillhart in 1975 [MB75], the quadratic sieve (QS for short) described by Pomerance in 1984 [Pom84] or the number field sieve (NFS for short) first suggested by Pollard in 1993 [Pol93] and more clearly described by Buhler, Lenstra and Pomerance in 1993 [BLP93] and Pomerance alone in 1994 [Pom94]. The number field sieve is to this day the best algorithm to factor large integers that are the product of two random primes of the same size and will be one of our main concerns in this document.

The algorithms designed to factor RSA-like integers such as NFS rely on the same idea: find congruences of squares modulo the integer $N = pq$ to factor. Indeed, if we find integers x, y that satisfy $x^2 \equiv y^2 \pmod{N}$, then $\gcd(x+y, N)$ or $\gcd(x-y, N)$ provides a non trivial factor of N with good probability. More precisely, if x is fixed and y is chosen randomly among all the values that satisfy $x^2 \equiv y^2 \pmod{N}$, reducing the previous equivalence modulo p (resp. q) yields $x \equiv \pm y \pmod{p}$ (resp. $x \equiv \pm y \pmod{q}$). There are four choices for y and the Chinese Remainder Theorem ensures that two of them will give nothing and two lead to a proper factorization.

If k such congruences of squares modulo N are found, the probability that none of the gcds described above yield a proper factor of N is then smaller than $1/2^k$ if N has only two distinct prime factors. It is even smaller if N has more than two factors. Once enough congruences of squares are found, factoring an integer is thus easy thanks to Euclid's algorithm. The hard part is building proper congruences, as we will see in more detail in the NFS case.

1.1.2 Computation of discrete logarithms in finite fields

A discrete logarithm is defined as follows.

Definition 1.1. *Let (G, \times) be a finite cyclic group of order m , g a generator of G and h an element of G . The discrete logarithm of h in base g is the element $x \in \mathbb{Z}/m\mathbb{Z}$ such that $g^x = h$.*

Solving a discrete logarithm problem (DLP) means finding such an element x when given G, g and h . The difficulty of this problem greatly depends on the group G . For instance, solving a discrete logarithm problem in the additive group $G = \mathbb{Z}/p\mathbb{Z}$ where p is a positive integer, is easily done thanks to Euclid's algorithm. In deployed cryptosystems such as the Diffie-Hellman key exchange or ElGamal encryption scheme described below, we would rather use groups in which solving a DLP is believed to be difficult. The groups used are either a subgroup of the multiplicative group of a finite field or a subgroup of the Jacobian of a curve defined over a finite field, in particular, elliptic curves.

It has been remarked by Pohlig and Hellman [PH78] that computing a DLP in a group of composite order m can be reduced to solving DLPs in the (smaller) subgroups of G of prime order. Thus the Pohlig-Hellman algorithm allows to break down a DLP into base DLPs: That is why we will assume in the following that the DLP we want to solve occurs in a group of prime order p . Each one of the base DLPs can then be solved separately. In generic groups — the interested reader will find a precise definition in [Sho97b] but these groups can roughly be thought of as cyclic groups in which no other structure helps in the computation of a DLP — this is usually done using the baby-step-giant-step algorithm due to Shanks [Sha71] or the Pollard-rho algorithm [Pol78]. Both algorithms have a time complexity in $O(\sqrt{p})$, which is actually the best possible time complexity for a DLP in a generic group as Shoup proved in [Sho97b]. Despite their genericity, these algorithms are used in practice to solve discrete logarithm problems when G is a subgroup of the group of rational points on an elliptic curve defined over a finite field. Indeed, such groups do not provide structures that we know how to exploit to speed up the computation of discrete logarithms as of today. In finite fields however, the square root complexity of the above algorithms can be enhanced by specific algorithms based on the index calculus method. It is especially the case of the number field sieve tweaked for DLP and of its variants. Before explaining how NFS can be used to solve DLPs in Section 1.2.2 we describe a few protocols the security of which is linked to the hardness of the discrete logarithm problem.

Diffie-Hellman key exchange protocol

This key exchange protocol described in 1976 by Diffie and Hellman in [DH76] aims at making two entities, say Alice and Bob, agree on a common secret. In order to do so, Alice and Bob publicly agree on a group G of order m and a generator g of G . Alice chooses a secret element $a \in \mathbb{Z}/m\mathbb{Z}$ and sends g^a on a possibly unsecure channel to Bob. Similarly, Bob chooses a secret integer $b \in \mathbb{Z}/m\mathbb{Z}$ and sends g^b to Alice. Both entities can now compute the secret $h = (g^a)^b = (g^b)^a$. This key exchange protocol is used in SSL/TLS and SSH protocols.

Recovering $h = g^{ab}$ from the knowledge of g, g^a and g^b is known as the computational Diffie-Hellman problem. This is not exactly a discrete logarithm problem but it is closely related (see for instance [Mau94] or [dB90]). Besides, computing discrete logarithms is the only known method today to solve such a problem: If an attacker knows how to efficiently compute discrete logarithm in G , he can also solve a computational Diffie-Hellman problem in G since he can compute a, b from g, g^a and g^b .

Notice that this version of the protocol is actually vulnerable to man-in-the-middle attacks as it does not authenticate the participants. An opponent Eve could intercept g^a before it reaches Bob and g^b before it reaches Alice. She could then send to Alice and Bob her own value g^e . Thus Eve agrees on a key g^{ae} with Alice and g^{be} with Bob, when Alice and Bob are convinced they share a secret key with one another. If Bob ever wants to send a message to Alice, he will in fact unknowingly send it to Eve and she will be able to decrypt it, read it and possibly modify it before re-encrypting it with the key she shares with Alice.

ElGamal schemes and more

The ElGamal encryption protocol described in 1985 in [ElG85] allows two entities to send and receive private messages through potentially unsecure channels. In order for Bob to send a message m to Alice, she must first compute her public and private keys. To do so, she chooses a finite cyclic group G of order m , a generator g of G and a secret exponent $a \in \mathbb{Z}/m\mathbb{Z}$. She broadcasts her public key (G, g, g^a) and keeps her private key a secret. To encrypt the message m , Bob chooses a random element r in $\mathbb{Z}/m\mathbb{Z}$ and sends to Alice the couple $(c, c') = (m(g^a)^r, g^r)$. Alice can retrieve the original message by computing $c(c')^{-a}$.

An attacker who wants to decrypt (c, c') must find g^{ar} from the knowledge of g, g^a and g^r , *i.e.*, solve a computational Diffie-Hellman problem which is closely linked to a discrete logarithm problem as we have seen above.

In [ElG85], ElGamal also describes a signature scheme, the security of which is also based on the hardness of DLP in well chosen groups. This property is shared with other deployed and standardized signature schemes such as the Digital Signature Algorithm (DSA) and its elliptic curve counterpart ECDSA [JMV01] which are actually variants of the ElGamal signature scheme.

Pairing-based protocols

A cryptographic pairing is an efficiently computable (that is, polynomial in the input size) map $e : G_1 \times G_2 \mapsto G_T$ where G_1, G_2, G_T are cyclic groups of order m that satisfy two properties: bilinearity and non-degeneracy. Pairing-based cryptography is at the heart of a number of recent schemes such as identity-based encryption schemes [BF01], identity-based signature schemes [PS06], group signature schemes [BBS04] and is also an important tool in zero-knowledge protocols [Gro10]. In practice, G_1 and G_2 are usually subgroups of a group of points of an elliptic curve and G_T is a subgroup of a finite field.

Without delving into too much detail, the security of these schemes is based on the hardness of the discrete logarithm problem in all three groups G_1, G_2, G_T . Thus, building secure pairings and secure pairing-based schemes is closely linked to the study of the difficulty of solving discrete logarithm problems.

1.2 Overview of the number field sieve (NFS)

1.2.1 Mathematical background

This section presents the main mathematical objects manipulated in the Number Field Sieve, in particular the one that gives its name to the algorithm. For more details on the structures involved and their properties we refer to [Sam97].

Definition 1.2. (*Number field*) A number field is a finite extension of \mathbb{Q} . Any number field is isomorphic to a quotient $\mathbb{Q}[X]/(f)$ where f is an irreducible polynomial in $\mathbb{Q}[X]$.

Given its definition, computing in a number field boils down to performing arithmetic operations on polynomials. The first step of NFS precisely consists in building two polynomials associated to two number fields K_0 and K_1 . It then considers principal ideals in the so called ring of integers of K_0 and K_1 .

Definition 1.3. If K is a number field, an algebraic integer in K is an element of K that is a root of a monic polynomial with integer coefficients. The set of algebraic integers in K is actually a ring: It is called the ring of integers of K and is denoted \mathcal{O}_K .

Once proper number fields are built, the NFS algorithm proceeds by computing and studying the norms of many principal ideals. The definition of the norm of an ideal is recalled below.

Definition 1.4. If K is a number field and \mathcal{O}_K its ring of integers, the norm of a nonzero ideal I is the cardinal of the quotient ring \mathcal{O}_K/I . The norm of an ideal I will be denoted $N(I)$.

In practice, norms of ideals in number fields will be computed through a resultant computation. Indeed, if $K = \mathbb{Q}(\alpha)$ where α is a root of an irreducible polynomial f and $g \in \mathbb{Z}[X]$ then the norm of $g(\alpha) \in K$ is $\text{Res}(f/f_n, g)$ where f_n is the leading term of f (see Proposition 4.3.4 in [Coh93]).

Once the norm of an ideal is computed in NFS, it is checked for smoothness with regard to a given bound. This is the last notion we will define in this section.

Definition 1.5. *Let B be a positive integer. An integer is B -smooth if all the primes in its factorization are below B . An ideal is B -smooth if all the prime ideals in its factorization have a norm below B .*

The analysis of NFS relies heavily on smoothness probabilities. We define below the notation that will be used to denote these probabilities. Although it is not required to understand the main ideas of NFS, this notation will be intensively used in Section 1.2.4 and later in Chapter 3 and we would like to introduce it early.

Definition 1.6. *We let*

$$\Psi(x, y) = \text{Card}(\{\text{integers in } [1, x] \text{ that are } y\text{-smooth}\}).$$

With this notation, $\Psi(x, y)/x$ is the probability that a uniformly random integer in $[1, x]$ is y -smooth.

Estimations, bounds and properties of $\Psi(x, y)$ play a crucial part in several number theory topics, particularly in the study of the distribution of integers without large prime factors whether globally, in short intervals or in arithmetically interesting sequences. We refer to Hildebrand and Tenenbaum's survey [HT93] as well as a more computationally oriented overview by Granville [Gra08b] for more information on this matter.

1.2.2 Main steps of NFS

The Number Field Sieve algorithm was originally designed for integer factorization. First it was introduced for integers with a specific shape by Pollard in [Pol93], then progressively extended to any integer (see for instance [LLMP93b]). It was later adapted to the computation of discrete logarithms in finite fields ([Gor93, Sch00]). In order to distinguish between these two algorithms, the number field sieve for factorization will be called NFS and the number field sieve for discrete logarithm computations will be called NFS-DL. When discussing parts of the algorithm that apply in both cases, we will overload the notation NFS. This section is devoted to the presentation of both algorithms: First we will explain its main ideas which are very similar in both cases, then we will discuss the specificities of NFS and NFS-DL. In our explanations, we consider the factorization of an integer N in the NFS case and the computation of a discrete logarithm in a subgroup of $(\mathbb{Z}/q\mathbb{Z})^*$ of order ℓ (where q is prime) in the NFS-DL case.

In a nutshell, NFS computes two compatible irreducible integer polynomials f_0 and f_1 , and searches for integer pairs (u, v) in a search space \mathcal{A} such that the integers $\text{Res}_x(u - vx, f_0(x))$ and $\text{Res}_x(u - vx, f_1(x))$ are smooth with respect to chosen smoothness bounds B_0 and B_1 , where Res_x denotes the resultant with respect to x . The notations K_i for $i = 0, 1$ denote the number fields $\mathbb{Q}[X]/(f_i)$. The number of smooth pairs that must be collected must be at least $\pi_{K_0}(B_0) + \pi_{K_1}(B_1)$, which denotes the number of prime ideals with norm at most B_0 and B_1 in the rings of integers of K_0 and K_1 . A linear algebra calculation in dimension $\pi_{K_0}(B_0) + \pi_{K_1}(B_1)$ follows. The factorization of an integer or the discrete logarithm of an element in a prime field can be recovered from this linear algebra step. We now present in more detail the main steps involved in both NFS and NFS-DL.

Polynomial selection

We describe this step for NFS. It is the same in NFS-DL except that any occurrence of N is replaced by q . The algorithm starts by selecting two irreducible integer polynomials f_0 and

f_1 of degree d_0 (resp. d_1) that share a common root m modulo N . Everything related to what happens in $K_0 = \mathbb{Q}[X]/(f_0) = \mathbb{Q}(\alpha_0)$ where α_0 is a root of f_0 (resp. $K_1 = \mathbb{Q}[X]/(f_1) = \mathbb{Q}(\alpha_1)$ where α_1 is a root of f_1) is said to be happening on *side 0* (resp. *side 1*). Usually in NFS the degree of f_0 is one so side 0 is also called the rational side while f_1 has a degree strictly greater than one making side 1 the so called algebraic side. In the following, we will consider that f_0 and f_1 have a leading coefficient equal to one. It ensures that α_0 and α_1 are algebraic integers which allows to avoid some technicalities in the following explanations. In practice though the polynomial selection algorithm used often leads to f_0 and f_1 having leading coefficients not equal to one and this has consequences as we will see for instance in Section 4.4.2.

There are many strategies to build the polynomials f_0 and f_1 . Regardless of what method is used the polynomial selection step always serves the same main goals and builds two polynomials f_0 and f_1 that satisfy the following properties:

- Both polynomials are irreducible and share a common root modulo N .
- The size of the coefficients of f_0 and f_1 and their degrees are controlled.

In practice, a third property is highly sought for: We would like to choose polynomials which generate many smooth values. This is of importance for the second step of the algorithm. Different metrics help ranking couples of polynomials in order to optimize this criterion ([Mur98] and [M⁺99] introduce some of these parameters, see also Section 3.1.2 of [Bou15] for a summary of the main parameters used to evaluate the quality of the selected polynomials).

One way to quickly build polynomials that satisfy the desired properties is to use the base- m selection. This has little to do with the way f_0 and f_1 are actually selected in practice but it is the polynomial selection classically used to assess the asymptotic complexity of the algorithm.

Definition 1.7. *In the base- m polynomial selection we set a degree d and m an integer close to $N^{1/d}$. The integer N can be written in base m : $N = n_0 + n_1m + \dots + m^d$. The polynomial f_0 is $x - m$, and f_1 is the polynomial $n_0 + n_1X + \dots + X^d$.*

Let us notice that the size of the coefficients of f_1 are all below m by construction and that f_0 and f_1 do indeed share a root modulo N since $f_1(m) = N$ by definition. Most probably, the polynomial f_1 is irreducible over \mathbb{Z} . If not, this is good news: $f_1 = gh$ with g and h non trivial so $N = f_1(m) = g(m) \times h(m)$ and we may have already found a factorization of N . The size of the degree d is tuned during the complexity analysis of this algorithm: In practice this degree stays below 10.

In practice the Kleinjung polynomial selection (see [Kle06]) or the Joux-Lercier selection (see [JL03]) in the case of NFS-DL can be used. Besides, when N has a special form, for instance when it is a Cunningham number, the coefficients of the selected polynomials can be made a lot smaller as in [LLMP93a] by directly taking advantage of this specific form instead of using more generic methods of selection. The variant of NFS used when the polynomial selection is made easy by the special form of N is called SNFS (Special Number Field Sieve).

Main idea of the NFS algorithm (resp. the NFS-DL algorithm)

We recall here that the polynomials f_0 and f_1 have a leading coefficient equal to one and a common root modulo N (resp. q) called m .

For $i \in \{0, 1\}$, we introduce the following morphisms:

$$e_i : \begin{cases} \mathbb{Z}[X] \rightarrow \mathbb{Z}[\alpha_i] \subset \mathcal{O}_{K_i} \\ X \mapsto \alpha_i \end{cases} \quad \text{and} \quad \pi_i : \begin{cases} \mathbb{Z}[\alpha_i] \rightarrow \mathbb{Z}/N\mathbb{Z} \text{ or } \mathbb{Z}/q\mathbb{Z} \\ \alpha_i \mapsto m \bmod N \end{cases}$$

Since f_0 and f_1 have a common modular root (modulo N or q), the following diagram commutes:

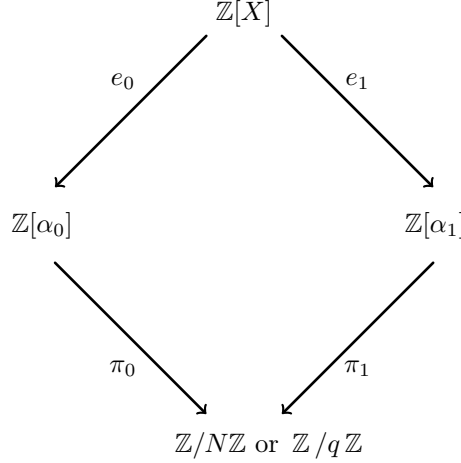


Figure 1.1: Commutative diagram summarizing the main idea of NFS

Because of the commutativity, for any polynomial $P \in \mathbb{Z}[X]$ we have $\pi_0(e_0(P)) = \pi_1(e_1(P))$ in the target ring at the bottom of the diagram. The algebraic structures on side-0 and side-1 are most likely not related so this yields non trivial equalities in the target bottom ring. Both NFS and NFS-DL take advantage of this crucial property as explained below.

Relation collection

The goal of this step is to compute relations, the definition of which is presented below.

Definition 1.8. *In both NFS and NFS-DL, a relation for smoothness bounds (B_0, B_1) is a couple (u, v) such that the ideal $(u - v\alpha_0)$ is B_0 -smooth and the ideal $(u - v\alpha_1)$ is B_1 -smooth, together with the factorization in \mathcal{O}_{K_0} (resp. \mathcal{O}_{K_1}) of the aforementioned ideals into prime ideals of K_0 (resp. K_1).*

For ease of reading a couple (u, v) satisfying the previous properties will also be called a relation.

Said otherwise, producing relations amounts to plugging linear polynomials $u - vX$ at the top of the previous diagram and trying to smooth them on both paths.

In order to produce relations we have thus two problems to solve: checking the smoothness of the considered ideals and actually factorizing them if they are smooth. If f_0 has degree one, $\mathbb{Z}[\alpha_0]$ is equal to \mathcal{O}_{K_0} and isomorphic to \mathbb{Z} so checking the smoothness of an element $u - v\alpha_0$ and factoring it in \mathcal{O}_{K_0} immediately makes sense. On side 1 (and on side 0 when f_0 is not of degree one), checking for smoothness is not too difficult: The norm of a prime ideal is a power of a prime, so checking for the B -smoothness of an ideal on side-1 roughly comes down to checking if its norm is B -smooth in \mathbb{Z} . But retrieving the actual factorization of a smooth ideal is not that straightforward as $\mathbb{Z}[\alpha_1]$ is not necessarily equal to \mathcal{O}_{K_1} which itself is not necessarily a unique factorization domain. We refer to [BLP93] regarding how to overcome these issues and emphasize that in the end it is possible to factor an ideal I in \mathcal{O}_{K_1} as a product of prime ideals even though \mathcal{O}_{K_1} itself might be unknown.

The pairs (u, v) are chosen in a so called sieving area \mathcal{S} and such that u and v are coprime. Let us remark that if (u, v) is a relation then so is $(-u, -v)$ but the latter does not yield new information if the first has been found. That is why the sieve area can safely be assumed to be $\mathcal{S} = \{(u, v) \in \mathbb{Z}^2 \mid \gcd(u, v) = 1, |u| \leq A, 0 \leq v \leq A\}$ where A is a bound tuned during the complexity analysis. For each side i , a smoothness bound B_i is fixed and the factor base \mathcal{B}_i is the set of prime ideals of \mathcal{O}_{K_i} with a norm smaller than B_i . The smoothness bounds are often considered equal, especially in the complexity analysis, but this is not a necessity, neither in practice nor for the complexity analysis. They too are adjusted by the complexity analysis.

For integers $(u, v) \in \mathcal{S}$, the norm of the ideal $(u - v\alpha_i)$ is checked for B_i -smoothness. There are several methods to achieve this goal: They will be briefly discussed later on. If its norm is smooth, the ideal $(u - v\alpha_i)$ can be written as a product of prime ideals of norm less than B_i . This is not obvious but is ensured when f_i has a leading coefficient equal to one and u and v are coprime (see [BLP93]). If a pair (u, v) yields smooth ideals on both sides *i.e.*, for $i \in \{0, 1\}$ we have,

$$(u - v\alpha_i) = \prod_{\mathfrak{p} \in \mathcal{B}_i, \mathcal{N}(\mathfrak{p}) < B_i} \mathfrak{p}^{e_{\mathfrak{p},i}}$$

for some integers $e_{\mathfrak{p},i}$, a relation has been found. Those relations are stored in a matrix whose rows are labeled by (u, v) and whose columns are labeled by the prime factors $\mathfrak{p} \in \mathcal{B}_0 \cup \mathcal{B}_1$: The coefficient on row (u, v) and in column $\mathfrak{p} \in \mathcal{B}_i$ is $e_{\mathfrak{p},i}$. Each row in this so called matrix of relations translates into two factorizations of ideals. Notice that most of the integers $e_{\mathfrak{p},i}$ are equal to zero so the matrix of relations is sparse by construction.

Several techniques exist to find doubly smooth norms. One family among them stands out and gives its name to NFS: the sieve. Sieve algorithms used to pinpoint which pairs (u, v) are relations use the same idea than the one exploited in the famous Eratosthenes' sieve. Instead of excluding integers until only primes remain, sieve algorithms exclude norms of ideals until only probably smooth ideals remain. A theoretical sieving procedure on side i proceeds as follows. For a given $v \leq A$ where we recall A is the bound on the sieving area, we initialize an array containing the norms $\mathcal{N}(u - v\alpha_i)$ for u ranging in $[-A, A]$. For each prime p below the smoothness bound B_i , the integers u such that $\mathcal{N}(u - v\alpha_i)$ is divisible by p are identified (see how in [LLMP93b, FK05, BL93]) and these norms are replaced by the quotient in the division of $\mathcal{N}(u - v\alpha_i)$ by the highest power of p possible. In the end, the cells containing ± 1 yield pairs (u, v) such that the ideal $(u - v\alpha_i)$ is smooth. Sieving can be done on both sides to find doubly smooth ideals.

In practice, several tricks are used for efficiency purposes. For instance, noticing that dividing a norm by p comes down to subtracting $\log p$ to the logarithm of the norm, we prefer to initialize our arrays with zeroes and add $\log p$ to the number stored in the cell corresponding to u if p divides $\mathcal{N}(u - v\alpha_i)$. If the value in the cell corresponding to u is above a sufficiently large threshold then $(u - v\alpha_i)$ is probably smooth. Moreover, small primes are not considered during the sieve and in practice relations can be allowed to have a small number (two or three) of so called large primes in their factorization, that is, primes above the smoothness bound but below a large prime bound. All of this means that sieving determines promising (u, v) but to actually call (u, v) a relation, the ideals $(u - v\alpha_0)$ and $(u - v\alpha_1)$ must still be checked for smoothness and completely factored: sieving yields part of the factorizations, small factors can be found using trial division and large factors with ECM for instance. In a word, sieving greatly narrows down the number of candidates (u, v) to consider.

The sieve algorithms can even be improved by the use of *special- q* (first introduced in [DH84] in the context of the quadratic sieve). A special- q is originally a (small) prime ideal that is forced to intervene in the factorization of the ideals $(u - vX)$, even though later considerations allow the use of composite special- q . Using special- q — whether prime or composite — increases smoothness probabilities and reduces the sieve area. A side effect of the use of special- q is the fact that some relations can be found several times: If a relation r involves the ideals \mathfrak{p}_1 and \mathfrak{p}_2 and both \mathfrak{p}_1 and \mathfrak{p}_2 are used as special- q then r will be found once during the sieve with \mathfrak{p}_1 and once during the sieve with \mathfrak{p}_2 . Such duplicates will need to be detected and deleted. It can be done during the filtering step (see below) but several strategies allow to remove them as soon as they are produced during the relation collection step, see for instance [BGG⁺20b] for more details.

Filtering step

This step only reduces the practical computing time of the algorithm. It aims at reducing the size of the matrix produced during the relation collection step while maintaining its sparsity in order to decrease the cost of the linear algebra step. It first removes irrelevant rows and

columns and proceeds with a structured Gaussian elimination [PS92]. More details on this step are presented in [Cav02], [Bou15] and in Section 1.3.

Linear algebra

During this step, the kernel of a large sparse matrix is computed using sparse linear algebra.

NFS case. In the NFS case, the end goal is to build congruences of squares modulo N . If we find a polynomial $P \in \mathbb{Z}[X]$ such that $e_0(P) = s_0^2$ and $e_1(P) = s_1^2$ are squares, using the morphisms π_0 and π_1 we have

$$\pi_0(s_0)^2 = \pi_1(s_1)^2 \bmod N$$

which is exactly what we want although this is no easy task to actually compute s_0 and s_1 . The idea is to build such a polynomial P as a product of linear polynomials $u - vX$. The goal is then to find a set S of pairs (u, v) such that

$$\prod_{(u,v) \in S} (u - v\alpha_i)$$

is a square in $\mathbb{Z}[\alpha_i]$ on both sides. For this property to be satisfied we must necessarily have that all the valuations of the ideals involved in the factorizations of the relations described by the set S are even, which leads to reducing the matrix of relations modulo two. While necessary, this condition is not sufficient but it is possible to make it so with good probability thanks to a few tricks that will be mentioned in the last step of the algorithm. Then we notice that all the elements in the left nullspace of the matrix of relations yield a set S which can then be exploited to find a congruence of squares modulo N and hopefully a proper factor of N . In order to guarantee finding a factor of N , the left nullspace of the matrix of relations must be large enough [BFHP20] (in practice, its dimension is around 100 – 150) which means that the relation collection step can only stop when the number of relations found exceeds the size of the factor base.

NFS-DL case As in the NFS case, the relation collection step provides us with lots of factorizations of ideals into prime ideals with a controlled norm. The goal is to build from this an injective “virtual logarithm” morphism from a subgroup of $\mathbb{Q}(\alpha_i)$ modulo ℓ -th powers to $\mathbb{Z}/\ell\mathbb{Z}$ on each side such that both morphisms are compatible with the commutativity of the NFS diagram. This is done by computing the images of elements of \mathcal{O}_{K_i} , namely the image of the elements in the factor base on each side, hoping that these images constrain the morphism enough for it to be uniquely determined — at least on the elements of which the logarithm needs to be computed. To give an intuition as to why this is our end goal, let us imagine that both sides of the diagram are rational. The relation collection step provides us with equalities of the form

$$\prod_{\mathcal{N}(p) < B_0} p^{e_{p,0}} = \prod_{\mathcal{N}(p) < B_1} p^{e_{p,1}}$$

where the products are mapped into $\mathbb{Z}/\ell\mathbb{Z}$. These multiplicative relations can be translated into additive ones by taking the logarithm:

$$\sum_{\mathcal{N}(p) < B_0} e_{p,0} \log(p) = \sum_{\mathcal{N}(p) < B_1} e_{p,1} \log(p) \bmod \ell$$

where the $\log(p)$ are unknown quantities. With enough linear equalities such as the previous one, we can solve a linear system in $\mathbb{Z}/\ell\mathbb{Z}$ that gives all the logarithms of the elements in the factor base. From this, we hope to recover any wanted logarithm.

Of course this reasoning needs some adjustments to work on an algebraic side. Indeed, the morphism π_i acts on elements and not ideals so mapping the factorizations we obtain during the relation collection step to \mathbb{F}_q^* is a challenge and was one of the main obstacles in the adaptation of NFS ideas to the computation of discrete logarithms. A workaround was found by Schirokauer. The Schirokauer maps (see [Sch93]) are characters from elements of $\mathbb{Q}[X]/(f_i)$ prime with ℓ to

$\mathbb{Z}/\ell\mathbb{Z}$ such that when all these characters vanish a factorization of ideals in $\mathbb{Z}[\alpha_i]$ can be meaningfully mapped to \mathbb{F}_q^* . Thus, the use of Schirokauer maps allows to give a sense to the logarithm of an ideal in $\mathbb{Z}[X]/(f_i)$: These objects are usually called virtual logarithms ([Sch05]). These maps can be computed in polynomial time and the number of Schirokauer maps is ruled by the number of roots of f_i . In practice, Schirokauer maps only add a few columns to the matrix of relations that are considered as if they were columns associated to prime ideals. As such, the virtual logarithms of the Schirokauer maps are computed alongside the virtual logarithms of the ideals in the factor base. We will not linger any longer on the technicalities required to give meaning to the previous equalities in an algebraic context. Similarly to what happens in the NFS case, we need enough relations (rows) compared with the number of ideals in the factor base (columns) for this kernel to yield the virtual logarithms of the elements in the factor base.

To sum up, in both cases we need to compute a kernel of a very sparse matrix: The number of nonzero coefficients per row is usually between 100 and 300. The kernel computation can be done with the same algorithm in both cases. The algorithm currently used is the block Wiedemann algorithm described by Coppersmith in [Cop94]. This algorithm main operation is the computation of matrix-vector products and its complexity depends on the size of the matrix and its weight, that is, the number of nonzero coefficients. This makes it suitable for sparse linear algebra. It is an improvement on the original Wiedemann algorithm (see [Wie86]) since it allows partial parallelization thus decreasing practical computing times. It is also easier to distribute the computations required by the block Wiedemann algorithm, thus allowing the processing of larger matrices.

The two main differences of the linear algebra step between NFS and NFS-DL are: In NFS we compute the left nullspace of a matrix over $\mathbb{Z}/2\mathbb{Z}$ while in NFS-DL we compute the nullspace of a matrix defined over $\mathbb{Z}/\ell\mathbb{Z}$. These differences have an impact in practice. Indeed the cost (and the memory cost) of each operation in the block Wiedemann algorithm is greater in $\mathbb{Z}/\ell\mathbb{Z}$ than in $\mathbb{Z}/2\mathbb{Z}$. Overall, the linear algebra step is more costly in the NFS-DL case, that is why in record DLPs, special care is given to the relation collection step in order to ensure the smoothest linear algebra step possible.

Final step

NFS case. In the NFS case, the last step is called the square root step. Indeed, the linear algebra step provided us with ideals $\prod_{(u,v) \in S} (u - v\alpha_i) \mathcal{O}_{K_i}$ in $\mathbb{Z}[\alpha_i]$ such that they factor into prime ideals with controlled norm and even valuations on both sides. From this we want to derive elements $\prod_{(u,v) \in S'} (u - v\alpha_i)$ in $\mathbb{Z}[\alpha_i]$ that are squares s_i^2 in $\mathbb{Z}[\alpha_i]$ and compute their square roots s_i , hence the name of this step.

Four obstacles, clearly described in [Pom94], lie in the way. We introduce the three most problematic issues and how to work around them below:

- The ring of integers \mathcal{O}_{K_i} is in general not equal to $\mathbb{Z}[\alpha_i]$ so a product $\prod_{(u,v) \in S} (u - v\alpha_i) \mathcal{O}_{K_i}$ is not necessarily the square of an ideal in \mathcal{O}_{K_i} .
- The ring \mathcal{O}_{K_i} is not necessarily principal so even if $\prod_{(u,v) \in S} (u - v\alpha_i) \mathcal{O}_{K_i} = I^2$ in \mathcal{O}_{K_i} the ideal I may not be principal.
- The group of units of \mathcal{O}_{K_i} is not trivial so even if $\prod_{(u,v) \in S} (u - v\alpha_i) \mathcal{O}_{K_i} = s_i^2 \mathcal{O}_{K_i}$ with $s_i \in \mathcal{O}_{K_i}$, it does not imply that $\prod_{(u,v) \in S} (u - v\alpha_i) = s_i^2$.

These three obstructions are overcome by the use of characters as proposed by Adleman in [Adl91]. A character χ_p used in NFS is a morphism from $\mathbb{Z}[\alpha_i]$ to $\{\pm 1\}$, associated to a prime ideal $p \in \mathbb{Z}[\alpha_i]$ and satisfying the following property: If $e \in \mathbb{Z}[\alpha_i]$ is the square of an element of $\mathbb{Z}[\alpha_i]$ then $\chi_p(e) = 1$. The reciprocal is false but if such an equality to one holds for

enough different characters the probability for e to be a square in $\mathbb{Z}[\alpha_i]$ becomes high. In fact this idea can even be made rigorous under GRH using the Grunwald-Wang theorem [BFHP20] as it gives a bound on how many characters are necessary for the reciprocal to be true. Said otherwise, if e is a square modulo enough primes then it is a square in $\mathbb{Z}[\alpha_i]$ with good probability. Thus, the raw elements of the kernel computed in the previous step are multiplicatively combined to create elements of which the evaluation of many characters yields one. For more details, we refer to [Pom94].

Finally, the square root step yields elements x, y such that $x^2 \equiv y^2 \pmod{N}$ which gives the factorization of N with good probability thanks to a gcd computation.

NFS-DL case. In the NFS-DL case, the last step is the individual logarithm computation, also named “descent”. The linear algebra step provided us with the image through compatible morphisms from \mathcal{O}_{K_i} to $\mathbb{Z}/\ell\mathbb{Z}$ of the ideals in the factor base. Said otherwise, it yielded the virtual logarithms of all the ideals with norms smaller than B_0 and B_1 and of the Schirokauer maps. From this we want to derive the logarithm of a given element e in the target subgroup of order ℓ we chose in \mathbb{F}_q^* . The main idea is to express this element as a product of elements of which the virtual logarithm is known. In order to do so, the element e is smoothed by computing elements $g^\varepsilon e$ for many exponents ε until one element of this form turns out to be β -smooth where β is tuned to control the computing time. Let us notice that the bound β is most probably above the bounds B_0 and B_1 . The factors of the element $g^\varepsilon e$ that lie between B_0 and β (resp. B_1 and β) are then integrated into new relations that involve smaller factors until it is possible to express their logarithms in function of the already known virtual logarithms. Let us notice that once the virtual logarithms of the factor base elements are computed this descent step can be performed for as many targets as we want with no additional cost. This property is responsible for a security vulnerability of the Diffie-Hellman key exchange called Logjam [ABD⁺15].

1.2.3 NFS in practice: implementations and records

To this day, the Number Field Sieve (NFS) is the most efficient method to factor integers and to compute discrete logarithms in prime finite fields. It is an active research area, and implementations of NFS have led to several record computations which give a good idea of the computational power required to factor RSA moduli up to around 2^{800} , or to compute discrete logarithms in prime fields of the same size. Existing software can also be used to form reasonable estimates of the hardness of these problems, up to roughly 1024 bits.

Historically, one of the first implementations of NFS is described by Bernstein and Lenstra [BL93]. Gradually, several implementations of NFS appeared: GGNFS which is currently stalled [Mon], MSIEVE by Papadopoulos [Pap] and CADO-NFS [CAD17] which is still currently developed at Inria Nancy. The latter is distributed under the GNU Lesser General Public License. The latest record computations, both for integer factorization and discrete logarithm computation, have been obtained with CADO-NFS and research aiming at improving this implementation is very active.

In 2021, the record for integer factorization with NFS is the factorisation of RSA-250 which is a 250-digit integer from the RSA challenge list. It quickly followed the factorization of RSA-240 which was obtained together with the discrete logarithm record computation that took place over a 240-digit prime field in 2020 [BGG⁺20b]. Computations took about 2700 core-years for RSA-250, 1000 core-years for RSA-240 and 3200 core-years for DLP-240 using Intel Xeon Gold 6130 CPUs at 2.1GHz. Figure 1.2 shows a broader overview of record computations obtained since the 90’s.

We emphasize that Figure 1.2 lacks at least one crucial data: the time cost of the computations. Anyway, comparing the time cost of those records is tricky as both the hardware and the software may change between two points. Yet, until 2020, a quick glance at this plot could at least have us believe that in practice computing DLPs is quite harder than factoring integers as there is a ten year delay between an integer factorization and the computation of a DLP in a field of the same size. The record computations of 2020 break this trend and show that the latter problem is not much harder than the former.

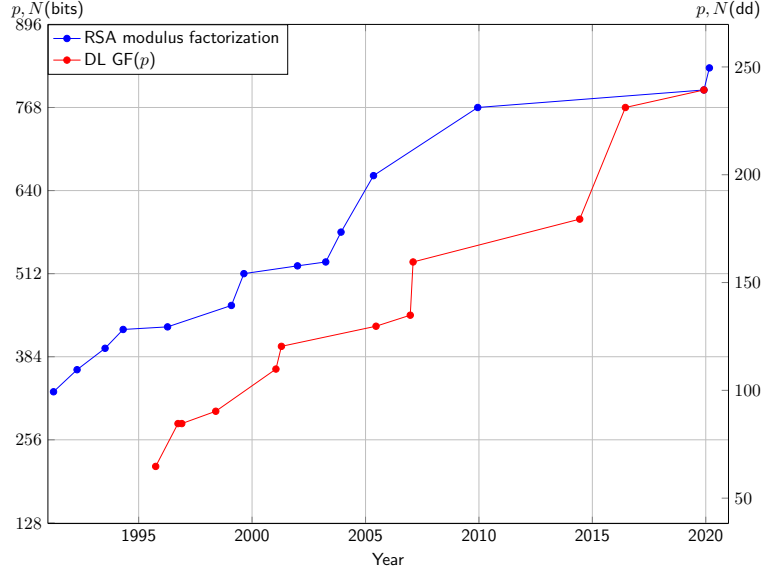


Figure 1.2: Brief history of computational records using NFS since the 90's

1.2.4 NFS in theory: asymptotic complexity

Besides record computations, another approach to estimate the computational power required by NFS is to use a theoretical complexity analysis. This analysis relies on several simplifying assumptions :

- The polynomial selection is a base- m selection (see Definition 1.7).
- The smoothness bounds B_0 and B_1 can be chosen equal to the same bound B without increasing the overall asymptotic complexity.
- The pairs (u, v) are picked from a set $\mathcal{A} = [-A, A]^2$ for some bound A .
- The norms of the ideals of which the smoothness is checked during the collection of relations step behave like random integers of the same size with regard to smoothness probabilities. This last hypothesis is detailed below:

Heuristic assumption 1.9. *In the number field sieve algorithm, the probability that the two integers $M_0 = \text{Res}_x(u - vx, f_0)$ and $M_1 = \text{Res}_x(u - vx, f_1)$ are simultaneously B -smooth, as (u, v) are picked uniformly from the search space \mathcal{A} , is given by the probability that two random independent integers of the same size are B -smooth, which is $\frac{\Psi(M_0, B)}{M_0} \cdot \frac{\Psi(M_1, B)}{M_1}$.*

We recall that $\Psi(x, y)/x$ denotes the probability for a random integer below x to be y -smooth as introduced in Definition 1.6. This last assumption is very bold, and in fact wrong in several ways. Correcting terms, which are actually used in practice, can be used to lessen the gap between $\text{Res}_x(u - vx, f_0)$ and integers of the same size [Mur99, BL17]. However, this heuristic is not that wrong asymptotically, as evidenced by [Gra08a, Eq. (1.21)]. We also note that if the number field is considered a random variable, positive results do exist [LV18].

Under these hypotheses, the asymptotic complexity of the usual variant of NFS to factor an integer N , is known to be

$$\exp \left(\sqrt[3]{\frac{64}{9}} (\log N)^{1/3} (\log \log N)^{2/3} (1 + \xi(N)) \right) \quad (1.1)$$

where $\xi(N) \in o(1)$ as N grows. This asymptotic formula is obtained by solving an optimization problem, which involves the number-theoretic function Ψ . The same formula holds for NFS-DL by replacing N by q .

The complexity of NFS and other factorization algorithms are often expressed using a standard L -notation that we introduce below:

Definition 1.10. *The L -notation is*

$$L_N(\alpha, c) = \exp(c(\log N)^\alpha (\log \log N)^{1-\alpha})$$

where $\alpha \in [0, 1]$ and $c > 0$.

The case $\alpha = 0$ corresponds to a polynomial function in the size of N (i.e., in $\log(N)$), the case $\alpha = 1$ corresponds to an exponential function and the case $\alpha \in]0, 1[$ to a sub-exponential function. The constant c is sometimes omitted. Using this notation, the asymptotic complexity of NFS is in $L_N(1/3, \sqrt[3]{64/9} + o(1))$ and in $L_q(1/3, \sqrt[3]{64/9} + o(1))$ for NFS-DL.

This theoretical complexity is used in practice to assess computing times for given values of N . In order to do so, the function ξ is replaced by zero in formula (1.1), for lack of a better alternative. The belief is that the size of the integers N for practical estimations is large enough (the order of magnitude is a thousand bits) for ξ to be close enough to zero. Yet this assumption has little to no foundation. Chapter 3 is precisely devoted to assessing the relevance of this brutal simplification by investigating the behaviour of the function ξ .

1.3 Focus on the filtering step in NFS

In this section, we elaborate on the filtering step, briefly introduced in Section 1.2.2. In particular, we describe the main substeps it involves and give examples of the operations performed. It serves two goals: introduce the vocabulary used in Chapter 4 and understand how the properties of the input matrix and the operations of the filtering step interact so as to faithfully simulate this step. The simulation of this step is tackled in Chapter 4.

In both NFS and NFS-DL, the relation collection step is the same. In both cases, elements of number fields are checked for smoothness, factored if it is the case and said factorizations are stored in a matrix. More precisely, each relation is numbered and so are elements of the factor base. The coefficient (i, j) in the matrix of relation is the valuation with respect to the j -th element in the factor base in the i -th relation. There is a strong connection between a row in the matrix of relations and a relation, that is why both terms are used indistinctively to denote a row of said matrix. Similarly, elements of the factor base are prime ideals in a number field and since a column in the matrix of relations corresponds to one such ideal, the terms ideal and column are used indistinctively to talk about a column.

Roughly speaking, the filtering step aims at balancing three quantities related to the matrix of relations: its size, its weight and its excess. These three notions are defined below.

Definition 1.11. *Let M be a matrix.*

- *The size of M is its number of rows.*
- *The weight of M is the number of nonzero coefficients in M . Similarly, the weight of a column (resp. a row) in M is the number of nonzero coefficients in this column (resp. row).*
- *The excess of M is the difference between its number of rows and its number of columns.*
- *The density of M is the quotient of its weight by its size. In particular, densities in the context of the filtering step do not lie between 0 and 1. Likewise, the density of a column is the quotient of the weight of this column by its size.*

Let us notice that the size of a non square matrix will (arbitrarily) be its number of rows even if it has more columns than rows. This is of little importance anyway since we will deal with matrices that typically have billions of rows and columns when entering the filtering step and millions when exiting while having an excess whose magnitude is in the hundreds. We now explain the goals of the filtering step in more detail in both the NFS and NFS-DL cases. We refer to Chapter 3 of [Cav02] and Chapter 5 of [Bou15] for even more details.

When using NFS to factor integers, the end goal is to find congruences of squares *i.e.*, we want to find subsets of relations that multiplied together give elements whose factorization only involve even valuations. Since only the parity of the valuations is needed to perform such a task, the elements of the matrix of relations are considered modulo 2. Solving the previous problem comes down to solving a linear algebra problem: compute the left kernel of the matrix of relations. Several elements of this kernel might be necessary to produce a congruence of squares modulo N (as explained in the square root step in Section 1.2.2) and among those congruences some will not yield a proper factor of N . To ensure that the factorization of N will be found nonetheless, the dimension of the kernel of the matrix of relations must be large enough: in practice around 100. The last sentence can be rephrased: The excess of the matrix of relations must be large enough in the case of factorization.

When using NFS to compute discrete logarithms, the next step after the relation collection is to compute virtual logarithms. This is done by computing the right kernel of the matrix of relations. The linear algebra step in NFS-DL slightly differs from the linear algebra step in NFS:

- The base field of the matrix of relations is $\mathbb{Z}/2\mathbb{Z}$ in NFS while it is $\mathbb{Z}/\ell\mathbb{Z}$ in NFS-DL where the order of magnitude for ℓ is between 160 and 256 bits in practical applications.
- A left nullspace is computed in NFS while it is a right nullspace in NFS-DL.
- The matrix in NFS must have enough excess while the matrix in NFS-DL can be square *ie* have an excess equal to zero.

Despite these differences, matrices in both cases behave similarly with respect to linear algebra: They are huge and very sparse. In both cases, sparse linear algebra algorithms are used. The complexity of these algorithms depends on the size and the weight of the input matrix. The filtering step is performed before the linear algebra step. It has three goals:

- Significantly reduce the size of the matrix of relations. For the record DLP-240, the number of relations after the relation collection step was around 8.9 billion, after the filtering step it was down to 282 million.
- Keep the sparsity of the matrix of relations. Typically, the matrix before filtering has less than 20 nonzero elements per row while the matrix after filtering has between 100 and 200 nonzero elements per row. While this is of course a significant increase, it is by no means comparable to the explosion that would be incurred by blind Gaussian eliminations.
- Ensure that the kernel of the matrix satisfies good properties, typically in the case of NFS it must be large enough.

Said otherwise, the filtering step aims at reducing the cost of linear algebra by finding a compromise between the size of the filtered matrix and its weight while controlling its excess.

The filtering step is divided in two sub-steps. The first one is called the purge step: It reduces the size of the matrix without increasing its weight. The second one is the merge step: It reduces further the size of the matrix at the cost of a slight increase in density.

1.3.1 Purge step

The purge step itself is divided in three substeps: the duplicates removal, the singletons removal and the clique removal.

The first substep is rather straightforward: Because of the sieve in the relation collection step, some relations might have been found several times. But these duplicates do not give any useful information so they can safely be removed. A variety of techniques can be used to deal with the duplicates. They can even be detected and removed during the relation collection step by maintaining hash tables that tell if a relation involving a given special- q would have been found when sieving a smaller special- q . Overall, this substep reduces the number of rows in the matrix.

A column is a singleton if it contains only one nonzero coefficient. If such a column is labeled by a prime ideal \mathfrak{p} , it means that there is only one relation r that involves \mathfrak{p} in its factorization. In the NFS case, this relation will never be used to build a congruence of squares. Indeed, the valuation of \mathfrak{p} in r is odd and is even in all the other relations so no product of relations involving r can make the valuation of \mathfrak{p} even. In the NFS-DL, if \mathfrak{p} is a singleton associated to the relation r it means that its logarithm can be deduced from the logarithms of the other ideals involved in r . In both cases, the singleton can be removed: completely in the NFS case and stored elsewhere in the NFS-DL case. This operation deletes one column and one row in the matrix of relations without increasing its weight. This operation can only increase the excess : Most of the time the excess is not modified and when a relation involves several ideals of weight one it increases as several columns are deleted for one row. In practice this means that the filtering step can even start with a negative excess as it will increase during the filter: It allows to stop the relation collection step earlier.

Let us notice that removing a singleton can produce new singletons. This step must be applied several times in order to remove all the singletons.

Example 1.12. *The following matrix M_0 has a weight of 8 and an excess of 1. Its first column is a singleton. Removing this singleton yields a matrix M_1 of weight 6 with an unchanged excess and creates a new singleton in the first column of M_1 . Removing this singleton finally produces a matrix M_2 with no singleton left and still no excess change.*

$$M_0 = \begin{pmatrix} 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad M_1 = \begin{pmatrix} 0 & 1 & 1 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad M_2 = \begin{pmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}$$

The following matrix M_0 has a negative excess (-1) and the second and fourth columns are singletons. Since the same relation is responsible for the later to be singletons, after removal we get a matrix M_1 with an excess equal to zero.

$$M_0 = \begin{pmatrix} 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix}, \quad M_1 = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}$$

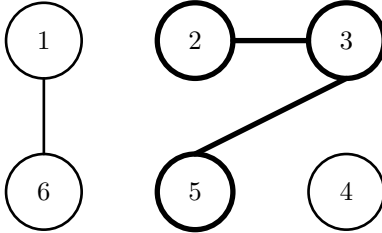
The final substep is the “clique removal”. Let us consider the following graph: Nodes are the relations of the matrix and there is an edge between two nodes if there is a ideal of weight two involved in both relations. In this context, a clique is a connected component of the previous graph. We emphasize that a clique in the context of NFS has nothing to do with a clique in its usual sense in graph theory but that it is nonetheless the accepted term to denote the aforementioned connected components because of the historical terminology in [Cav02].

The clique removal takes advantage of the following property of cliques: If any row is deleted in the matrix of relations, all the rows and all the columns in its clique can be deleted as well. Let us remove a relation r . If it was alone in its clique then removing r did indeed remove the clique

of r . If not then r is linked to another relation r' through an ideal \mathfrak{p} of weight 2. Since r was removed, r' is now the only relation that involves \mathfrak{p} : The deletion of r made r' a singleton which can in turn be deleted. The same reasoning can recursively be applied to r' until the whole clique of r is deleted.

Most of the time, removing a clique deletes n columns for $n + 1$ rows so it decreases the excess by one. Only in rare cases (which happen when the clique is cyclic) can the excess increase. As long as there is enough excess, this excess is used to remove the most heavy cliques according to a specified weight function. Roughly speaking, the weight of a clique usually depends on the number of nonzero coefficients involved in the relations that form the clique. For more precise information on the weight functions that can be used, we refer to [Bou15]

Example 1.13. *The following matrix M_0 has an excess of one. If we number the relations from top to bottom, the corresponding graph for clique removal is presented below. If the weight of a clique is the number of nonzero coefficients that are involved in its composing relations, the heaviest clique is the one in bold.*

$$M_0 = \begin{pmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \end{pmatrix}$$


We remove this clique to get a matrix M_1 whose excess did indeed decrease by one. Let us notice that removing a clique can produce new singletons as it is the case in the second column of M_1 . We remove this singleton to get the final matrix M_2 .

$$M_1 = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix}, \quad M_2 = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$$

1.3.2 Merge step

The purge step allows both to decrease the size and the weight of the matrix of relations. The merge step however increases the weight of the matrix in order to decrease further its size by performing structured Gaussian eliminations [PS92]. The Gaussian operations are made on rows and they are chosen so that the weight of the resulting rows is as small as possible. This step can be stopped at any moment: In practice, a target density for the final matrix is chosen and Gaussian elimination is performed as long as the density of the current matrix stays below the target density.

Let us explain the main idea of the merge step on simple cases. In NFS, the matrix of relations is a matrix over $\mathbb{Z}/2\mathbb{Z}$. Let \mathfrak{p} be an ideal of weight two and denote by r and r' the two relations in which \mathfrak{p} is involved. Replace the row r in the matrix of relations by $r + r'$: This operation is called a *2-merge*. Interestingly, it makes the ideal \mathfrak{p} a singleton since it only appears in r' now. This singleton can be deleted which allows to remove the row r' and the column \mathfrak{p} from the matrix of relations. The size of the matrix decreases and in this case, so does its weight: If r has weight w and r' has weight w' , the weight of $r + r'$ is at the most $w + w' - 2$ since the two nonzero coefficients in column \mathfrak{p} are deleted. Of course this idea can be extended to the case of NFS-DL. The only difference is that the matrix of relations has coefficients in $\mathbb{Z}/\ell\mathbb{Z}$ instead of $\mathbb{Z}/2\mathbb{Z}$ so instead of replacing r by $r + r'$ to make \mathfrak{p} a singleton, r must be replaced by a linear combination $ar + a'r'$ where $a, a' \in \mathbb{Z}/\ell\mathbb{Z}$. In practice in NFS-DL, 2-merges are applied only if $a, a' \in \pm 1$.

This principle can be extended to columns of weight greater than two. Let \mathfrak{p} be an ideal of weight three and r_0, r_1, r_2 the three relations involving \mathfrak{p} . Replace the row r_0 by $r_0 + r_2$ and the row r_1 by $r_1 + r_2$: This operation is called a *3-merge*. This operation does not change the kernel

while making \mathfrak{p} into a singleton since it only appears in r_2 so column \mathfrak{p} and row r_2 can be deleted. However there are two main differences between a 3-merge and a 2-merge. First, the weight of the matrix increases during a 3-merge: if w_i denotes the weight of r_i , the weight $w_0 + w_1 + w_2$ of the set of rows $\{r_0, r_1, r_2\}$ has been replaced by the weight $w_0 + 2w_2 + w_1 - 2$ of the set of rows $\{r_0 + r_2, r_1 + r_2\}$. Thus, the size of the matrix of relations decreased but its weight increased. Second, there are several ways to combine r_0, r_1, r_2 : We could have made r_0 or r_1 be the only relation in which I is involved instead of r_2 . Since we want to increase the weight of the matrix as little as possible, it is important to study which combination of these relations increases the weight of the resulting matrix the least.

The same idea can be applied to columns of weight k , in which case the operation is called a k -merge:

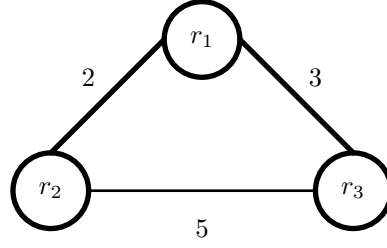
Definition 1.14. *Let \mathfrak{p} be an ideal of weight k in the matrix of relations and r_0, r_1, \dots, r_{k-1} be the k rows involving \mathfrak{p} . A k -merge is a sequence of operations of the form $r_i \leftarrow r_i + r_j$ with $i \neq j$ aiming at making the ideal \mathfrak{p} become a singleton. Performing a k -merge means replacing the rows r_i by the sums $r_i + r_j$ and deleting the column \mathfrak{p} and the row that makes I a singleton. The weight of a k -merge is the smallest number of nonzero coefficients possible in the rows involved in the merge after it has been performed.*

Similarly to a 3-merge, a $k > 2$ merge allows to delete one row and one column at the cost of an increase in weight. In order to find the best combination of rows, that is, the combination that increases the weight the least, a minimum spanning tree is computed. We consider the following complete edge-weighted graph: The nodes are the relations involving a fixed ideal of weight k and the weight on the edge linking r and r' is the weight of $r + r'$. A minimum spanning tree on this graph yields the combinations of rows that guarantee the least increase in weight [Cav02].

In practice the merge step is performed in several passes. The first passes perform all possible 2-merges: There is no reason not to do a 2-merge since it decreases both the weight and the size of the matrix. Let us notice that applying a 2-merge may produce new columns of weight two that can in turn be merged. If there are no 2-merges available, a parameter k is increased (as long as it does not exceed a parameter bound k_{max} that rules the maximum weight of a column that we allow to be merged) and from now on it will be increased with each pass in order to allow heavier merges. All the κ -merges for $2 \leq \kappa \leq k$ are stored and their weight computed. In fact computing the weight of a merge requires to compute a minimum spanning tree and would be too costly so we only compute an estimation of this weight using the Markowitz rule [Mar57]: For each merge, the lightest row is added to all the other rows. The merges' estimated weights are sorted in increasing order and corresponding merges are applied until the end of the pass. Minimum spanning trees are only computed when the merge is really applied. The density of the resulting matrix is then computed and compared with the target density: If it exceeds it, the merge step stops ; if it does not, a new pass of merge begins.

Example 1.15. *Let us consider the following matrix M_0 and denote by r_i its i -th row. There are two columns in M_0 that can be used for a 3-merge: columns 2 and 4. We estimate the weight of these three 3-merges. For column 2, this amounts to adding r_3 (which has weight 3) to r_1 and r_2 (which have weight 4) and summing the weights of $r_3 + r_1$ and $r_3 + r_2$. This estimated weight of a 3-merge on column 2 is 8. Similarly, the estimated weight of a 3-merge on column 4 is obtained by summing r_3 to r_5 and r_6 and it equals 9. The first 3-merge to be applied will then be the merge on column 2. The graph attached to these merges is presented below, with a minimal spanning tree in bold.*

$$M_0 = \begin{pmatrix} 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$



Here is an example where the estimated weight of the merge does not give the least increase in density: The operations that will indeed be performed are $r_1 + r_2$ and $r_2 + r_3$ instead of $r_3 + r_1$ and $r_3 + r_2$. This yields the matrix M_1 below:

$$M_1 = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Performing this first 3-merge created new possible 3-merges in columns 4, 5 and 6. In general, k -merges can even produce k' -merges with $k' < k$.

Chapter 2

Background on Riemann-Roch spaces

This chapter provides insight on Riemann-Roch spaces and some of their uses, as well as a brief overview on how to efficiently compute them. It introduces notations and notions that will be intensively used in Chapter 5.

2.1 Riemann-Roch spaces

2.1.1 Mathematical background

This section is devoted to presenting some notions and tools on projective curves that will be needed to introduce Riemann-Roch spaces. For further details we refer to [Ful08] and [NX09]. In this section K is a perfect field and \bar{K} is an algebraic closure of K .

We briefly recall that the projective plane over K , denoted \mathbb{P}^2 is the quotient of $K^3 \setminus \{(0, 0, 0)\}$ by the equivalence relation of collinearity. Said otherwise, the points of coordinates (x, y, z) and (kx, ky, kz) with $k \in K \setminus \{0\}$ denote the same point in this space and we will denote the so called homogenous coordinates of this point $(x : y : z)$. More intuitively, this space can be seen as the classical affine plane K^2 to which is added a line of points at infinity: the points that have homogenous coordinates $(x : y : 0)$ for some $(x, y) \in K^2 \setminus \{(0, 0)\}$.

We also recall that a polynomial with n indeterminates is homogeneous of degree d if all its monomials with nonzero coefficient have total degree d and notice that for any such polynomial P and all $\lambda \in K$ we have $P(\lambda X_1, \lambda X_2, \dots, \lambda X_n) = \lambda^d P(X_1, X_2, \dots, X_n)$. Actually, this property characterizes homogeneous polynomials of degree d . An homogeneous polynomial of degree d is also called a *form* of degree d .

With these two notions in mind, we now define a projective plane curve.

Definition 2.1. A projective plane curve defined over K is described by an element of the set

$$\bigcup_{d \in \mathbb{Z}_{>0}} K_d[X, Y, Z] / \sim$$

where $K_d[X, Y, Z]$ is the set of homogeneous polynomials of degree d with three indeterminates and \sim is an equivalence relation on this set such that $P \sim Q$ if and only if there is a $\lambda \in K \setminus \{0\}$ satisfying $P = \lambda Q$.

It is said to be absolutely irreducible if the polynomial that defines it is irreducible in \bar{K} .

In the rest of the section as well as in Chapter 5, we will only consider absolutely irreducible curves, which ensures that the curves considered are algebraic varieties [NX09, Example 2.3.10].

Similarly to Definition 2.1, an *affine curve* is described by a non constant element of $K[X, Y]/\sim$. Whether the curve is projective or affine, the *degree* of the curve is the degree of the polynomial that describes it. It is quite common to mix up a curve with its defining polynomial.

We quickly explicit below the link between projective and affine curves:

- Homogenisation is the map $\begin{cases} K[X, Y] & \rightarrow K[X, Y, Z] \\ P(X, Y) & \mapsto \tilde{P} = Z^d P(X/Z, Y/Z) \end{cases}$ where d is the degree of P

The polynomial \tilde{P} should be an element of $k(Z)[X, Y]$ but it is easy to check that it actually belongs to $k[X, Y, Z]$. It is homogeneous of degree d , hence the name of this map.

- De-homogenisation is the map $\begin{cases} K[X, Y, Z] & \rightarrow K[X, Y] \\ P(X, Y, Z) & \mapsto P(X, Y, 1) \end{cases}$

Example 2.2. The points (x, y) on a parabola satisfy $y = x^2$ which means that this affine curve is described by the polynomial $Y - X^2$. Homogenizing this polynomial yields the polynomial that describes the projective parabola: $YZ - X^2$.

More generally:

Notation 2.3. If \mathcal{C} is a projective curve described by a polynomial $Q \neq Z$, we will denote by \mathcal{C}^0 the affine curve obtained by intersecting \mathcal{C} with the Zariski open subset $\{Z \neq 0\} \subset \mathbb{P}^2$. It is described by the bivariate polynomial $q(X, Y) = Q(X, Y, 1)$ resulting from the de-homogenisation of Q .

The condition $Q \neq Z$ is actually not restrictive as it can be obtained through a linear change of coordinates.

In the following, all definitions will be given for projective curves. However, it will always be possible to consider that the curves manipulated in Chapter 5 are affine, in particular when performing algorithmic operations on them. Indeed we will see in Chapter 5 that only a finite number of points on a given curve are useful for our algorithmic purposes. If all those points are affine *i.e.*, none of them lie on the line at infinity $\{(x : y : 0) \mid (x, y) \in K^2 \setminus \{(0, 0)\}\}$ it is immediately possible to forget about the behaviour of the curve \mathcal{C} at infinity by de-homogenisation and come down to working on the affine curve \mathcal{C}^0 . If some of those points lie at infinity a linear change of coordinates allows to get back to a more favorable situation. Indeed, as shown in [Ful08], if K is large enough with regard to the number of relevant points, there is a linear change of coordinates that make the line $Z = 0$ avoid this finite set of points. This change of coordinates makes all relevant points affine and the previous reasoning can be applied. If K is not large enough for this trick to work, we can always work in a small extension of K . This might however give rise to complexity concerns.

A point of a projective curve is *singular* if its multiplicity m defined as in [Ful08, Section 3.1] is greater than one : such a point is said to have order m . If there is at least one singular point on a projective curve, the curve is said to be singular too, otherwise, the curve is said to be smooth. A singularity is ordinary if the tangents (see [Ful08, Section 3.1] again) at this singularity are all distinct. We give some visual intuition for singular points in an affine context: in Figure 2.1, both curves have a singular point in $(0, 0)$ of order two. On the left, it is an ordinary singularity, on the right it is not.

Definition 2.4. A curve is nodal if all its singularities have order 2 and are ordinary. An ordinary singularity of order two is called a node.

In this document, all the curves encountered will be nodal. This is a good compromise between simplicity and genericity. Indeed, nodal curves have the simplest singularities possible, which helps depicting the algorithms in Chapter 5. On the other hand, these curves are generic enough: Any projective algebraic curve admits a nodal plane model (up to a field extension if k is a small finite field), which can be for instance obtained by computing the image of a nonsingular projective model of the curve by a generic linear projection to \mathbb{P}^2 [ACGH85, Appendix A].

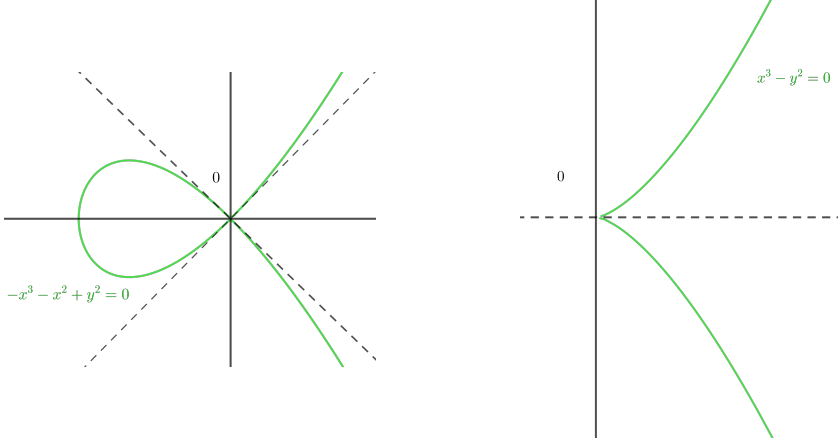


Figure 2.1: Ordinary and non ordinary singularities

Notation 2.5. If \mathcal{C} is a nodal curve, we use the notation $\tilde{\mathcal{C}}$ to denote a nonsingular model of \mathcal{C} which projects onto \mathcal{C} (as denoted by X in [Ful08, Ch. 8]). We assume that this implicit projection $\tilde{\mathcal{C}} \rightarrow \mathcal{C}$ is one-to-one on nonsingular points of \mathcal{C} and that it is two-to-one on nodes.

Now that curves have been introduced we turn to the definition of divisors. We consider an absolutely irreducible projective nodal curve \mathcal{C} defined over K .

Definition 2.6. *Divisors*

- A divisor on a curve \mathcal{C} is a formal sum of closed points (see [NX09, Section 1.2] for a precise definition) of $\tilde{\mathcal{C}}$ with integer coefficients that are all equal to zero except for a finite number. It is an effective divisor if all the integer coefficients involved are nonnegative.
- When the support of a divisor D i.e., the finite set of points with nonzero coefficients, involves only points of $\tilde{\mathcal{C}}$ which project to nonsingular points of \mathcal{C} , we call D a smooth divisor of \mathcal{C} by slight abuse of terminology. More generally, we will often identify nonsingular closed points of \mathcal{C} with their corresponding points on $\tilde{\mathcal{C}}$.
- On a nodal curve \mathcal{C} with r nodes the nodal divisor, denoted by E , is the effective divisor of degree $2r$ which is the sum of all the closed points of $\tilde{\mathcal{C}}$ which project to a node of \mathcal{C} .
- For two divisors $D = \sum_{P \in \tilde{\mathcal{C}}} n_P P, D' = \sum_{P \in \tilde{\mathcal{C}}} n'_P P$ on \mathcal{C} , we write $D \leq D'$ if for any $P \in \tilde{\mathcal{C}}$ the inequality $n_P \leq n'_P$ holds.

If $D = \sum_P n_P P$ (where only a finite number of integer coefficients n_P are nonzero) is a divisor on a curve \mathcal{C} , the degree of D is denoted $\deg(D)$ and equals $\sum_P n_P$ and its size is $\sum_P |n_P|$.

The technicalities implied by possible singular points and the fact that K is not necessarily algebraically closed must not let us lose sight of the fact that a divisor is a rather simple object: a finite formal sum of points with integer coefficients. As such, one can keep the following example in mind each time a divisor is mentioned.

Example 2.7. The two projective points $P = (0 : 1 : 1)$ and $Q = (3 : 4 : 5)$ are on the projective curve $X^2 + Y^2 - Z^2$. The formal sum $D = 3P - 2Q$ is a divisor on this curve. Its degree is 1, its size is 5 and it is greater than the divisor $-4Q$. This divisor is the difference of two effective divisors: $3P$ and $2Q$.

Remark 2.8. Actually, any divisor D can be uniquely written $D_+ - D_-$ where D_+ and D_- are effective divisors with distinct supports. As we will see in Chapter 5, we prefer manipulating effective divisors so we will generally use this decomposition.

Among all the divisors on a curve, some are special as they are linked to elements of the function field of \mathcal{C} , the definition of which is recalled below.

Definition 2.9. Let \mathcal{C} be a projective absolutely irreducible plane curve described by a polynomial Q . Its homogeneous coordinate ring is $K[\mathcal{C}] = k[X, Y, Z]/Q(X, Y, Z)$ and its function field is $K(\mathcal{C}) = \left\{ \frac{R}{S} \mid \frac{R}{S} \in \text{Frac} \left(\frac{K[X, Y, Z]}{(Q)} \right) \text{ and } R, S \text{ are homogeneous of the same degree} \right\}$.

In other words, an element of $K(\mathcal{C})$ is a rational fraction R/S where R and S are considered modulo Q . Such a function can be evaluated at any point $P \in \mathcal{C}$ that does not make S vanish. If $R(P) = 0$, P (resp. $S(P) = 0$), P is a zero of R/S (resp. a pole).

Example 2.10. If \mathcal{C} is the projective parabola $YZ - X^2$, $\frac{YZ-2X}{Y-YZ}$ represents the same element as $\frac{X^2-2X}{Y-X^2}$ in $k(\mathcal{C})$, of which $(1 : 1 : 1)$ is a pole and $(2 : 4 : 1)$ is a zero.

Definition 2.11. If $f \in K(\mathcal{C})$ is a nonzero function on \mathcal{C} , i.e. a quotient $f = g/h$ of two nonzero forms $g, h \in k[\mathcal{C}]$ of the same degree, we let (f) denote the associated divisor, as defined in [Ful08, Sec. 8.1].

Roughly speaking, the divisor associated to a function $f \in K(\mathcal{C})$ is the divisor for which the coefficient in front of a point P is the order of P in f : It is then zero for any point that is neither a pole nor a zero of f , positive and equal to the multiplicity of P in f if P is a zero and negative and equal to the opposite of the multiplicity of P in $1/f$ if P is a pole. Since f has a finite number of zeros and poles, (f) is well defined and its degree is zero because of Bezout's theorem [Ful08, Section 5.3]. A divisor D such that there is $f \in K(\mathcal{C})$ satisfying $D = K(\mathcal{C})$ is called a principal divisor. Two divisors that differ by a principal divisor are said to be equivalent and quotienting the group of degree-0 divisors on \mathcal{C} by this equivalence relation, that is, quotienting the group of degree-0 divisors on \mathcal{C} by the subgroup of principal divisors, yields the so called Jacobian of the curve \mathcal{C} .

Before moving on to the definition of a Riemann-Roch space, we first introduce notations and definitions for divisors associated to functions that do not lie in $K(\mathcal{C})$.

Notation 2.12. • If $g \in K[\mathcal{C}]$ is a nonzero form on \mathcal{C} , we denote by $(g)_+$ the associated effective divisor, as defined in [Ful08, Sec. 8.1]. It is essentially the same definition as for an element of $K(\mathcal{C})$ except that there are no poles.

- If $g \in K[\mathcal{C}^0]$ is a nonzero regular function on \mathcal{C}^0 , then by abuse of notation, we overload the notation (g) to denote the effective divisor associated to the form $Z^{\deg(g)}g(X/Z, Y/Z, 1)$. We emphasize that this divisor has no pole and that it is not the principal divisor associated to the function $g(X/Z, Y/Z, 1) \in K(\mathcal{C})$.

2.1.2 Riemann-Roch spaces, Riemann-Roch theorem

A Riemann-Roch space is a set of functions on a curve whose zeros and poles are constrained. More precisely:

Definition 2.13. Let D be a divisor on a plane absolutely irreducible projective curve \mathcal{C} . The Riemann-Roch space $L(D)$ associated to D is the K -vector space

$$L(D) = \{f \in K(\mathcal{C}) \setminus \{0\} \mid (f) \geq -D\} \cup \{0\}$$

Example 2.14. If $D = 3P - 2Q$ then a function in $L(D)$ must have a valuation in P greater than -3 and a valuation in Q greater than 2 : In other words, $L(D)$ is the set of rational functions that have a pole in P of order at most 3 and a zero in Q of order at least 2 . Let us notice that a function in $L(D)$ in this case cannot have a pole anywhere but at P , yet it can have other zeros besides Q .

This example can be generalized for any divisor D written $D = D_+ - D_-$ with D_+ and D_- effective: The divisor D_+ constrains the poles of the elements of $L(D)$ and D_- constrains its zeros.

Remark 2.15. *If $\deg(D) < 0$ then $L(D) = 0$. Indeed, if $f \in L(D) \setminus \{0\}$, then $(f) \geq -D$ and taking the degrees in this inequality we get $0 \geq -\deg(D)$ since (f) is a principal divisor: This is a contradiction.*

If D is written $D_+ - D_-$ with D_+ and D_- effective divisors with distinct supports, the degree of D is $\deg(D_+) - \deg(D_-)$ and its size is $\deg(D_+) + \deg(D_-)$. The previous remark shows that we can always assume that $\deg(D) \geq 0$ (otherwise, the Riemann-Roch space associated to D is trivial) hence that $\deg(D_+) \geq \deg(D_-)$. In this context, the size of D is bounded above by $2\deg(D_+)$ and below by $\deg(D_+)$ so the degree of D_+ is a relevant measure of the size of D . It explains why the complexities of algorithms computing Riemann-Roch spaces are expressed in terms of $\deg(D_+)$ instead of the size of D .

It is easily checked that a Riemann-Roch space is a vector space. Moreover these vector spaces are of finite dimension. Indeed, if $D = \sum_P n_P P$ where all n_P are zero except for a finite number, we obviously have $D \leq \sum_P |n_P| P = D_0$ so $L(D) \subset L(D_0)$ according to [Ful08, Proposition 3, Section 8.1]. This same proposition ensures that

$$\dim(L(D_0)/L(0)) \leq \deg(D_0 - 0) = \deg(D_0)$$

and since $L(0) = K$ by [Ful08, Corollary 1, Section 8.1], $\dim(L(D_0)) = \dim(L(D_0)/L(0)) + 1 \leq \deg(D_0) + 1$ which means that in turn $\dim(L(D)) \leq \deg(D_0) + 1 < +\infty$. This crude proof does not however provide the dimension of the Riemann-Roch space associated to a divisor D nor an effective method to compute a base of this vector space. The latter will be our main concern in Chapter 5.

The Riemann-Roch theorem is a fundamental result both in algebraic geometry and complex analysis (for the computation of the dimension of the space of meromorphic functions with prescribed zeros and allowed poles for instance). Riemann first proved a lower bound for a Riemann-Roch space and one of his students, Roch, provided the correcting term between the aforementioned lower bound and the dimension of $L(D)$ giving birth to the Riemann-Roch theorem. Originally this theorem was proved in a purely analytic context but we present below its version for algebraic curves. (which amounts to replacing “compact Riemann surface of genus g ” by “smooth projective curve of genus g ”).

Theorem 2.16. *(Riemann-Roch theorem [Ful08, Section 8.6]) Let \mathcal{C} be a smooth projective curve of geometric genus g and Δ a canonical divisor on \mathcal{C} . Then for any divisor D on \mathcal{C} :*

$$\dim(L(D)) - \dim(L(\Delta - D)) = \deg(D) + 1 - g$$

We refer to [Ful08, Section 8.5] (resp. [Ful08, Section 8.3]) for a definition of a canonical divisor (resp. a definition of the genus). The smoothness assumption can actually be relaxed. For more information on this crucial theorem, its generalizations and applications we refer to [Mir95].

2.1.3 Uses of Riemann-Roch spaces

In the following, computing a Riemann-Roch space $L(D)$ means explicitly finding a basis of $L(D)$. The computation of Riemann-Roch spaces is a subroutine used in several areas of computational mathematics. It can for instance be used for the integration of algebraic functions [Dav81] — the desired primitive is characterized by its zeros and poles and a Riemann-Roch space computation ensues — or to study diophantine equations as in [Coa70]. We detail below two of its main uses in cryptography.

Computation in the Jacobians of curves

As introduced in Section 2.1.1, the Jacobian of a curve \mathcal{C} is the quotient of the group of degree-0 divisors on \mathcal{C} by the subgroup of principal divisors on \mathcal{C} . Representing a point in the Jacobian of

a genus- g curve \mathcal{C} as $D - gO$, where D is an effective divisor of degree g and O is a fixed rational point (or more generally, a fixed divisor of degree 1) the sum of the classes of $D_1 - gO$ and $D_2 - gO$ can be computed by finding a function f in the Riemann-Roch space $L(D_1 + D_2 - gO)$. Indeed, by setting $D_3 = D_1 + D_2 - gO + (f)$, the divisor $D_3 - gO$ is linearly equivalent to $(D_1 - gO) + (D_2 - gO)$.

This was originally of cryptographic interest as the Jacobians of elliptic or hyperelliptic curves are good candidates to be discrete logarithm-resistant groups and can thus be used in cryptosystems described in Section 1.1.2. To make them practical we need an efficient way to compute the group law on the Jacobian of such curves. Riemann-Roch spaces computations are not required in this context as direct formulas already allow to perform arithmetic in Jacobians of elliptic or hyperelliptic curves, but computing in the Jacobians of curves retains its interest for applications in number theory and algebraic geometry.

Algebraico-geometric error-correcting codes

One of the most prominent applications of the computation of Riemann-Roch spaces is the construction of algebraico-geometric error-correcting codes. Introduced by Goppa [Gop83] [Gop77], encoding with such a code corresponds to evaluating a basis of a Riemann-Roch space at points of an algebraic curve. Therefore, to use such codes in practice the basis of a Riemann-Roch space must be computed. They can be used for instance in the McEliece cryptosystem introduced by McEliece in 1978 [McE78] in which errors are added to the code word corresponding to a clear message to hide it and removed during the decoding process. Although the security properties of McEliece cryptosystems are not always well understood and that some codes lead to weaknesses in these schemes [Min07] they have some advantages over more popular asymmetric cryptosystems such as RSA among which being a good candidate for post-quantum cryptography.

2.2 Computation of Riemann-Roch spaces

As emphasized in Section 2.1.3, some applications require ways to efficiently and explicitly compute bases of Riemann-Roch spaces. We present below a brief overview of algorithms designed to solve either this general problem or the more specific one that is the computation of the group law in the Jacobian of a curve, as well as the theorem on which so called geometric approaches rest to tackle the issue.

2.2.1 Algorithmic state-of-the-art

In this section \mathcal{C} denotes a projective curve of genus g and $D = D_+ - D_-$ a divisor on this curve such that $\deg(D_+) \geq \deg(D_-)$. As emphasized in Remark 2.15, this inequality is not restrictive and in this case $\deg(D_+)$ is a relevant measure of the size of D . There are two families of algorithms to compute Riemann-Roch spaces: geometric and arithmetic algorithms.

Geometric algorithms rely on the Brill-Noether approach described in more detail in Section 2.2.2 which was originally restricted to curves with ordinary singularities. Goppa was among the first to propose an algorithm using this method in [Gop83, §4] but it only worked in finite fields and some parts of this algorithm use exhaustive search. The last forty years gave birth to more and more efficient algorithms in the vein of the Brill-Noether idea by incorporating tools of modern computer algebra.

Le Brigand and Riesler first extended the Brill-Noether method to any plane curve and any divisor in 1988 [LBR88] and Haché [Hac95] proposed the first implementation of Brill-Noether's approach in a computer algebra system, using local desingularizations to handle singularities encountered during the algorithm in 1995. Roughly at the same time, Huang and Ierardi provided in [HI94] a deterministic algorithm for computing Riemann-Roch spaces of a plane curve \mathcal{C} all singularities of which are ordinary within $O(\deg(\mathcal{C})^6 \deg(D_+)^6)$ arithmetic operations in the base field while Volcheck [Vol94] described an algorithm with complexity $O(\max(\deg(\mathcal{C}), g)^7)$ to perform additions in Jacobian of curves, which we recall, involves the computation of particular Riemann-Roch spaces.

In parallel, ideas by Coates [Coe70] and Davenport [Dav81] initiated the emergence of arithmetic algorithms that rely mostly on the computation of integral closures in algebraic function fields. The forefront algorithm using this approach was designed by Hess [Hes02] in 2002. It is polynomial in the input size, does not require assumptions on the curve singularities nor specific changes of coordinates that are usually needed in geometric approaches. Despite these advantages, further magnified by the fact that Hess's algorithm has been implemented in the MAGMA computer algebra system making it a reference method, it requires to compute integral closures which is costly. In fact, this operation alone has higher complexity than some overall geometric approaches [Abe20] which explain why geometric algorithms still retain their relevance.

In 2007, Khuri-Makdisi [KM07] further enhanced the results of Volcheck by proposing a geometric algorithm computing the group law on Jacobians of general curves which requires $O(g^{\omega+\varepsilon})$ operations in the base field, where ω is a feasible exponent for matrix multiplication and ε is any fixed positive number. Actually, the ε in this complexity can be removed if the cardinality of the base field grows polynomially in g , which is the case in the aforementioned paper. We present in Chapter 5 an algorithm that achieves similar complexity in the case of smooth divisors on nodal curves using basic tools (resultants and linear algebra) together with an implementation that improves the computing times obtained with MAGMA. Our complexity bounds have since been improved upon by Abelard, Couvreur and Lecerf in [ACL20], thus achieving sub-quadratic complexity.

2.2.2 Geometric algorithms and the residue theorem

The classical geometric approach for computing Riemann-Roch spaces is due to Brill and Noether. Given a divisor D on a projective curve \mathcal{C} , this method proceeds by finding first a common denominator to all the functions in the Riemann-Roch space $L(D)$. This is done by computing a form h on the curve such that the associated effective divisor (h) satisfies $(h) \geq D$. Then the residual divisor $(h) - D$ is computed. From this, a basis of the Riemann-Roch space is found by computing the kernel of a linear map.

The correctness of this method is ensured by the residue theorem of Brill and Noether, which works even in the presence of ordinary singularities by using the technique of adjoint curves, see [Sev21, § 42][Ful08, Sec. 8.1]. This theorem is actually one of the foundations of the theory of adjoint curves.

Proposition 2.17. *[Ful08, Sec. 8.1] Let \mathcal{C} be a projective curve, absolutely irreducible over a perfect field K and whose singularities are ordinary. Let D, D' be two linearly equivalent effective divisors on \mathcal{C} and E be the adjoint divisor of \mathcal{C} as defined in [Ful08, Prop. 2, Sec. 8.1]. Let $g \in k[\mathcal{C}]$ be a form such that $(g) = D + E + A$ for some effective divisor A . Then there exists a form $g' \in k[\mathcal{C}]$ of the same degree as g such that $(g') = D' + E + A$.*

Proof. We note f the polynomial describing the curve \mathcal{C} . Since D and D' are equivalent, there are polynomials h, h' of the same degree such that $D - D' = (h/h')$ that is: $D + (h) - (h') = D'$. We apply the Max Noether theorem (see [Ful08, Sec. 7.5]) to the polynomials gh, h' and f : It ensures the existence of homogeneous polynomials g' and h' such that $gh = g'h' + ff'$ and satisfying in particular $\deg(g') = \deg(g)$. This equality implies :

$$\begin{aligned} (g') &= (g) + (h) - (h') \\ &= D + E + A + (h) - (h') \text{ by definition of } g \\ &= D' + E + A \text{ because of the equivalence of } D \text{ and } D' \end{aligned}$$

which yields the desired result. The Max Noether theorem can indeed be applied: $(gh) = (g) + (h) = D + E + A + (h) = D' + (h') + E + A \geq (h')$ because the divisors D', A and E are effective. This ensures that the multiplicity of any point in the intersection of the curves described by f and h' is greater in gh than in h' , hence Noether conditions are satisfied for all those points. \square

In the case of nodal plane curves, an adjoint curve is just a curve which goes through all the nodes of \mathcal{C} , and the adjoint divisor E is the divisor defined in 2.6. We will see in Section 5.1.3 how to use the residue theorem to prove the correctness of a Brill-Noether based algorithm such as the one depicted in Chapter 5.

Part II

Complexity and simulation of the Number Field Sieve

Chapter 3

Asymptotic complexity of NFS

This chapter is based on *Refined Analysis of the Asymptotic Complexity of the Number Field Sieve*, co-authored with Pierre-Jean Spaenlehauer and Emmanuel Thomé [LGST21]. A few amendments and expansions have been made in this thesis. In particular, Section 3.1 has been revised, Section 3.2 did not figure in the original paper and neither did Proposition 3.11, and Section 3.3.3 has been detailed for pedagogical purposes.

Recall that the overall complexity analysis of NFS relies on several heuristics, most notably on the fact that norms of random ideals in a given number field have the same smoothness probability as random integers of the same order of magnitude (see Heuristic 1.9). But even if we consider that these assumptions hold, the complexity of NFS to factor an integer N is given by

$$\exp \left(\sqrt[3]{\frac{64}{9}} (\log N)^{1/3} (\log \log N)^{2/3} (1 + \xi(N)) \right)$$

where $\xi(N) \in o(1)$ (see Section 1.2.4). This formula involves a function ξ which tends to zero as the entry N grows, yet is never spelled out explicitly.

This inaccuracy conflicts with the fact that for the last few decades, the widespread development of public-key cryptography has created a pressing need to answer the following question: Given some computational power C , what size of RSA modulus N should be considered so that the cost of NFS exceeds C ? This is of interest to regulatory bodies, see for example [NIS19, Sec. 7.5], [ANS14, Sec. B.2.2], or [ENI14, Table 3.1]. A common way to answer this is to consider the complexity formula, assume that $\xi = 0$ since we know $\xi(N) \in o(1)$ when N grows, and use a computational record to set a proportionality ratio. Since the 2000s, the validity of this approach has been considered differently. In [LV01, §2.4.4], the reader is warned that ξ should not be neglected for large extrapolations, and omitting it probably yields biased estimations. Later work gradually moved to considering $\xi = 0$ as a totally acceptable simplification assumption, not heeding the warning. We emphasize that this method for computing key sizes is the international standard for deployed RSA-based cryptography.

The goal of this chapter is to question the relevance of neglecting ξ and to give insights on what this function hides.

3.1 A minimization problem

To achieve the goal of this chapter we need first to find optimal asymptotic choices of NFS parameters as N grows, in order to minimize its heuristic asymptotic computational cost. This amounts to minimizing a function of the parameters of NFS bound together by a non-linear constraint. This section is devoted to establishing this constraint and expliciting the aforementioned function.

From now on, we use the hypotheses introduced in Section 1.2.4. Our goal is not to get rid of them, not even the one explicit in Heuristic 1.9. The usual complexity analysis of NFS already uses them as a base and we will do the same to improve the asymptotic estimate in Formula (1.1). We will also assume the generalized Riemann hypothesis (GRH). We recall that $\Psi(x, y)/x$ denotes the probability for a random integer below x to be y -smooth and introduce the following shortcut notation before proceeding.

Definition 3.1. *For convenience, we use the notation:*

$$p(u, v) = \log(\Psi(e^u, e^v)/e^u)$$

This notation is designed to remove layers of exponential or logarithm functions in the expressions involving smoothness probabilities below and clarify and shorten the expressions manipulated from now on. For the same purpose of clarity we also introduce the following notation to denote the iterates of the logarithm function. It is quite classical in smoothness-related papers, see for example [CEP83] or [Pom81].

Notation 3.2. *Throughout this chapter, $\log x$ denotes the natural logarithm to base e . We use the notation \log_k to denote the k -th iterate of the log function, so that $\log_{k+1} = \log \circ \log_k$. Therefore, we stress that \log_k does not stand for the logarithm to base k .*

It is now time to describe the mathematical problem whose solution is the asymptotic complexity of NFS. We treat the case of NFS but the same holds for NFS-DL.

According to the hypotheses in Section 1.2.4, we use the base- m polynomial selection described in Definition 1.7 to produce two polynomials f_0 and f_1 . We recall that m is close to $N^{1/d}$ where d is the degree of f_1 . We then pick pairs (u, v) from a set $\mathcal{A} = [-A, A]^2$, for some bound A and check $\text{Res}_x(u - vx, f_0(x))$ and $\text{Res}_x(u - vx, f_1(x))$ for smoothness with respect to a common bound B . Notice that the hypothesis according to which the smoothness bounds on side-0 and side-1 can be taken equal could probably be relaxed at the cost of an extra layer of technicality to the following analysis. The integers that we check for smoothness in NFS are bounded by M_0 and M_1 , which we define as follows

$$\begin{aligned} |\text{Res}_x(u - vx, f_0(x))| &\leq M_0 = (m + 1)A, \\ \text{and } |\text{Res}_x(u - vx, f_1(x))| &\leq M_1 = (d + 1)mA^d. \end{aligned}$$

Our simplified version of NFS has three main parameters that we can tune, as functions of $\nu = \log N$, in order to optimize the asymptotic complexity: the degree $d = \deg f_1$, the size of the search space A and the common smoothness bound B . The choice to express d , A and B in terms of $\log N$ instead of N is guided by the fact that it allows to avoid layers of logarithms that render the following equations more difficult to read. The relation collection and linear algebra step are the most costly. It is rather clear why we can neglect the time of polynomial selection, filtering and square root steps, but a little less so for the descent in the NFS-DL case. Explanations allowing to disregard the complexity of the descent can be found in [Sem02], [EGT11] and [FGHT17]. They can be roughly summed up by the following take home message: The parameters of the descent are precisely tuned so that the time needed for the descent is asymptotically negligible when compared to the linear algebra and relation collection steps. The point is: The time complexity of NFS is the sum of the time taken by the search for smooth pairs and the time taken by linear algebra. We write these as

$$\begin{aligned} C_{\text{search}} &= A^2 \cdot C_{\text{test}}, \\ C_{\text{linear algebra}} &= (\pi_{K_0}(B) + \pi_{K_1}(B))^2(1 + o(1)). \end{aligned}$$

In the expressions above, C_{test} is the time spent per pair (u, v) and $\pi_{K_i}(B)$ denotes the number of prime ideals in K_i with a norm below B . Several techniques can be used to bring C_{test} to an

amortized cost of $O(1)$; sieving is the most popular, but ECM-testing also works (heuristically), and it is also possible to detect smooth values with product trees as in [Ber02]. Furthermore, we counted a quadratic cost for linear algebra, as this can be carried out with sparse linear algebra algorithms such as [Wie86]. We wish to minimize the cost, subject to one constraint: We need enough smooth pairs, so that the left nullspace of the matrix of relations is non trivial. Hence we need more rows than columns in the aforementioned matrix. Taking into account the heuristic hypothesis 1.9, this translates into:

$$A^2 \cdot \frac{\Psi(M_0, B)}{M_0} \cdot \frac{\Psi(M_1, B)}{M_1} \geq \pi_{K_0}(B) + \pi_{K_1}(B). \quad (3.1)$$

Of course, we can assume that this is an equality, since otherwise decreasing A would decrease the overall complexity.

The next step of the analysis is to express C_{search} , $C_{\text{linear algebra}}$ and the factors in the constraint (3.1) in terms of the parameters d , A and B . Let $a(\nu) = \log A(\nu)$ and $b(\nu) = \log B(\nu)$. Note that all functions are expected to tend to infinity as $\nu = \log N$ tends to infinity. Moreover, for $i \in \{0, 1\}$ and by using an explicit version of Chebotarev's density theorem under GRH [GM19] (which allows to take into account the fact that K_i depends on N) we have

$$\log \pi_{K_i}(B(\nu)) = \log \left(\text{Li}(B(\nu)) + O(\sqrt{B(\nu)} \log(D_{K_i} B(\nu)^{n_{K_i}})) \right)$$

for $B(\nu) \gg (\log D_{K_i})^{2+\varepsilon}$ where D_{K_i} is the absolute discriminant of K_i and n_{K_i} is its absolute degree. Since $\log \text{Li}(x) = \log x - O(\log_2 x)$, it implies that $\log(\pi_{K_0}(B(\nu))) = b(\nu) - O(\log b(\nu))$ and $\log(\pi_{K_1}(B(\nu))) = b(\nu) - O(\log b(\nu))$ as well. As a result,

$$\begin{aligned} \log C_{\text{search}} &= O(1) + 2a(\nu), \\ \log C_{\text{linear algebra}} &= O(1) + \log((\pi_{K_0}(B(\nu)) + \pi_{K_1}(B(\nu)))^2) \\ &= 2b(\nu) - 2\varepsilon(\nu), \\ \log(C_{\text{search}} + C_{\text{linear algebra}}) &= O(1) + \log \max(C_{\text{search}}, C_{\text{linear algebra}}) \\ &= O(1) + 2 \max(b(\nu) - \varepsilon(\nu), a(\nu)). \end{aligned}$$

where $\varepsilon(\nu) = O(\log b(\nu))$. Likewise, the constraint on the probabilities can be rewritten, using the notation $p(u, v)$ from Definition 3.1:

$$\begin{aligned} 2a(\nu) + \sum_{i \in \{0, 1\}} p(\log M_i, b) &= \log(\pi_{K_0}(B(\nu)) + \pi_{K_1}(B(\nu))) \\ &= b(\nu) - \varepsilon(\nu). \end{aligned}$$

Given the inaccuracy that exists in Formula (1.1), we can profit from some simplifications before we formulate our optimization problem. It is sufficient to minimize the function $\max(a(\nu), b(\nu) - \varepsilon(\nu))$, as this would imply a cost that would be at most within a constant factor of the optimal. Using the same argument we can take into account only the important parts of the bounds M_0 and M_1 , namely mA and mA^d . In fact, we can even disregard the terms in $\varepsilon(\nu)$ both in the objective function and the constraint. Indeed, minimizing the function $\max(a, b)$ implies a cost for NFS that would be at most within a $O(\log b)$ factor of the optimal solution. We will see in Section 3.2 that the order of magnitude of this factor is necessarily $O(\log_3 N)$, which is too small to impact the results of Section 3.2 and 3.4. Our optimization problem is therefore rewritten as the following simplified problem, which is our main target here as well as in Section 3.4:

Problem 3.3 (Simplified optimization problem). *Find three functions,*

$$a(\nu), b(\nu), d(\nu) : \mathbb{R}_{>0} \rightarrow \mathbb{R}_{>0}$$

which for all $\nu \in \mathbb{R}_{>0}$ minimize $\max(a(\nu), b(\nu))$ subject to the constraint

$$p(a + \nu/d, b) + p(da + \nu/d, b) + 2a - b = 0. \quad (3.2)$$

Once this optimization problem is solved, the asymptotic complexity of NFS ensues: It is equal to $\exp(2 \max(a, b))$.

3.2 First order resolution

To deal with Optimization Problem 3.3, the classical analysis of NFS relies on the following result.

Proposition 3.4 (Canfield-Erdős-Pomerance [CEP83] and De Bruijn [DB51b]). *Let $\varepsilon \in]0, 1[$. If $3 \leq x/y \leq (1 - \varepsilon)x/\log x$ then, as $x/y \rightarrow +\infty$:*

$$p(x, y) = -(x/y) \cdot \log(x/y) \cdot (1 + o(1)).$$

Using this low-order result, we rediscover the following well-known expressions (see e.g. [BLP93, §11]) for the minimizers a, b, d :

Proposition 3.5. *Let (a, b, d) be a minimizer for Problem 3.3. Then*

$$\begin{aligned} a(\nu) &= (8/9)^{1/3} \nu^{1/3} (\log \nu)^{2/3} (1 + o(1)), \\ b(\nu) &= (8/9)^{1/3} \nu^{1/3} (\log \nu)^{2/3} (1 + o(1)), \\ d(\nu) &= (3\nu/\log \nu)^{1/3} (1 + o(1)). \end{aligned}$$

Furthermore,

$$\begin{aligned} (a(\nu) + \nu/d(\nu))/b(\nu) &= \frac{1}{2} (3\nu/\log \nu)^{1/3} (1 + o(1)), \\ (d(\nu)a(\nu) + \nu/d(\nu))/b(\nu) &= \frac{3}{2} (3\nu/\log \nu)^{1/3} (1 + o(1)). \end{aligned}$$

We emphasize that in the literature, these expressions are usually found using various hypotheses on the shape of the minimizers a, b, d . Here, we do not make any of these assumptions to show that the expressions for the minimizers only derive from the Canfield-Erdős-Pomerance formula and Optimization Problem 3.3.

Proof. We only prove the first part of Proposition 3.5, as the second is a direct consequence. Here follows the outline of the proof:

1. We show that there exists functions (a_0, b_0, d_0) in the claimed classes which satisfy the constraint of the optimization problem. It provides a base result: If (a, b, d) is solution to Problem 3.3 we will have in particular $\max(a, b) \leq \max(a_0, b_0)$. Since we know in which classes of functions a_0 and b_0 belong, it will give insight on a and b .
2. We show that any minimizer for Problem 3.3 must satisfy the above equalities.

Those two steps roughly amount to expanding Equation (3.2), called *the equation of the constraint* below, using Canfield-Erdős-Pomerance formula.

Step 1: Set

$$\begin{aligned} a_0(\nu) &= (8/9)^{1/3} \nu^{1/3} (\log \nu)^{2/3} (1 + \tilde{a}(\nu)), \\ b_0(\nu) &= (8/9)^{1/3} \nu^{1/3} (\log \nu)^{2/3}, \\ d_0(\nu) &= (3\nu/\log \nu)^{1/3}. \end{aligned}$$

where \tilde{a} is an unknown function of ν . We have:

$$\begin{aligned} \frac{a_0 + \nu/d_0}{b_0} &= \frac{3^{1/3}}{2} \nu^{1/3} (\log \nu)^{-1/3} (1 + K(\nu)) \\ \log \left(\frac{a_0 + \nu/d_0}{b_0} \right) &= \frac{\log \nu}{3} \left(1 + H(\nu) + \frac{3 \log(1 + K(\nu))}{\log \nu} \right) \\ \frac{a_0 d_0 + \nu/d_0}{b_0} &= \frac{3^{1/3}}{2} \nu^{1/3} (\log \nu)^{-1/3} (3 + 2\tilde{a}) \\ \log \left(\frac{a_0 d_0 + \nu/d_0}{b_0} \right) &= \frac{\log \nu}{3} \left(1 + H(\nu) + \frac{3 \log(3 + 2\tilde{a})}{\log \nu} \right) \end{aligned}$$

with $K(\nu) = 2/3^{1/3}\nu^{-1/3}(\log \nu)^{1/3}(1 + \tilde{a})$ and $H(\nu) = 3/(\log \nu) \times (\log(3^{1/3}/2) - (\log_2 \nu)/3)$. Using Proposition 3.4, and multiplying the whole equality by $3^{2/3}\nu^{-1/3}(\log \nu)^{-2/3}$, Equation (3.2) becomes:

$$\left[-\frac{1}{2}(1 + K) \left(1 + H + \frac{3 \log(1 + K)}{\log \nu} \right) - \frac{1}{2}(3 + 2\tilde{a}) \left(1 + H + \frac{3 \log(3 + 2\tilde{a})}{\log \nu} \right) \right] (1 + o(1)) + 4\tilde{a} + 2 = 0$$

We denote by $f(\tilde{a}, \nu)$ the left-hand term of this equality. Tedious but straightforward computations show that $f(0, \nu) < 0$ for sufficiently large ν and $f(1/(\log \nu)^{1/2}, \nu) > 0$ for sufficiently large ν . It means that there is a ν_0 such that for all $\nu \geq \nu_0$, both inequalities hold. Let $\nu \geq \nu_0$ be given. Since $t \mapsto f(t, \nu)$ is a continuous function of t , the intermediate value theorem ensures that there is a $\tilde{a}(\nu) \in]0, 1/(\log \nu)^{1/2}[$ such that $f(\tilde{a}(\nu), \nu) = 0$. Because of the range in which \tilde{a} lies, the squeeze theorem says that $\tilde{a}(\nu)$ tends to 0 when ν grows. Thus we have proved the existence of a function \tilde{a} which belongs to $o(1)$ and such that the functions a_0, b_0, d_0 satisfy the equation of the constraint.

Step 2: Let (a, b, d) be a minimizer for Problem 3.3. We can always write (a, b, d) as follows (at least for $\nu > e$ which is enough for our purposes):

$$\begin{aligned} a(\nu) &= \nu^{1/3}(\log \nu)^{2/3} \tilde{a}(\nu), \\ b(\nu) &= \nu^{1/3}(\log \nu)^{2/3} \tilde{b}(\nu), \\ d(\nu) &= \nu^{1/3}(\log \nu)^{-1/3} \tilde{d}(\nu). \end{aligned}$$

where \tilde{a}, \tilde{b} and \tilde{d} are unknown functions. By minimality, we know that $a \leq \max(a, b)$ which itself is smaller than a function in $(8/9)^{1/3}\nu^{1/3}(\log \nu)^{2/3}(1 + o(1))$ according to Step 1. This ensures that $\tilde{a} = O(1)$. The same holds for \tilde{b} .

Lemma 3.6. *The function \tilde{b} is bounded away from zero.*

Proof. The equation of the constraint (3.2) can be rewritten:

$$2a = b - p(da + \nu/d, b) - p(a + \nu/d, b)$$

We know that a, b, d, ν are all positive number and so is $-p(x, y)$ for any x, y . We can therefore derive two facts from the equation of the constraint. First $2a \geq b$ which ensures that:

$$\log \tilde{a} \geq \log \tilde{b} + O(1). \quad (3.3)$$

Furthermore $(d\sqrt{a} - \sqrt{\nu})^2 \geq 0$ so $da + \nu/d \geq 2\sqrt{a\nu}$. Since $-p(x, y)$ increases as x grows, we have:

$$2a \geq \frac{2\sqrt{a\nu}}{b} \log \left(\frac{2\sqrt{a\nu}}{b} \right). \quad (3.4)$$

We simplify this inequality by replacing a and b by their expressions. First we have:

$$\frac{2\sqrt{a\nu}}{2} = 2\nu^{1/3}(\log \nu)^{-1/3} \frac{\sqrt{\tilde{a}}}{\tilde{b}}.$$

From this expression, it follows that:

$$\begin{aligned} \log \left(\frac{2\sqrt{a\nu}}{b} \right) &= \frac{\log \nu}{3}(1 + o(1)) + \frac{\log \tilde{a}}{2} - \log \tilde{b} \\ &\geq \frac{\log \nu}{3}(1 + o(1)) + \frac{-\log \tilde{b}}{2} + O(1) \end{aligned}$$

where the last inequality is ensured by (3.3). Finally, (3.4) can be rewritten:

$$\sqrt{\tilde{a}} \geq \frac{1 + o(1)}{3\tilde{b}} + \frac{-\log \tilde{b} + O(1)}{2\tilde{b}(\log \nu)}$$

Now assume that $\liminf \tilde{b} = 0$ as ν grows. It implies that $\limsup(-\log \tilde{b} + O(1)) = +\infty$ so that the \limsup of the second term of the right-hand side of this inequality is nonnegative. Moreover, the \limsup of the first term is $+\infty$. It implies that $\limsup \tilde{a} = +\infty$ which contradicts the fact that \tilde{a} is bounded and concludes the proof of this first claim. \square

Lemma 3.7. *The functions $\tilde{a}, \tilde{b}, \tilde{d}$ satisfy the following equality:*

$$\tilde{a}\tilde{d}^2(1 + \varepsilon_1) + 2 + \varepsilon_2 + 3\tilde{b}^2\tilde{d} - 6\tilde{a}\tilde{b}\tilde{d} = 0$$

where $\varepsilon_1, \varepsilon_2$ are both functions of ν that are in $o(1)$.

Proof. We expand the equation of the constraint as a function of $\tilde{a}, \tilde{b}, \tilde{d}$ of which we omit the argument. We have:

$$\begin{aligned} \frac{a + \nu/d}{b} &= \frac{1}{\tilde{b}\tilde{d}} \nu^{1/3} (\log \nu)^{-1/3} \left(1 + \frac{\tilde{a}\tilde{d}}{\nu^{1/3} (\log \nu)^{-1/3}} \right) = \frac{1}{\tilde{b}\tilde{d}} \nu^{1/3} (\log \nu)^{-1/3} (1 + o(1)) \\ \log \left(\frac{a + \nu/d}{b} \right) &= \frac{\log \nu}{3} \left(1 - \frac{\log_2 \nu}{\log \nu} - \frac{3 \log \tilde{b}}{\log \nu} - \frac{3 \log \tilde{d}}{\log \nu} + \frac{3 \log(1 + o(1))}{\log \nu} \right) = \frac{\log \nu}{3} (1 + o(1)) \\ \frac{ad + \nu/d}{b} &= \frac{1}{\tilde{b}} \left(\tilde{a}\tilde{d} + \frac{1}{\tilde{d}} \right) \nu^{1/3} (\log \nu)^{-1/3} \\ \log \left(\frac{ad + \nu/d}{b} \right) &= \frac{\log \nu}{3} \left(1 - \frac{\log_2 \nu}{\log \nu} - \frac{3 \log \tilde{b}}{\log \nu} + \frac{3 \log(\tilde{a}\tilde{d} + 1/\tilde{d})}{\log \nu} \right) = \frac{\log \nu}{3} (1 + o(1)). \end{aligned}$$

The simplifications made in the first, second and fourth expressions are explained below:

- We obviously have $(\log_2 \nu)/(\log \nu) \in o(1)$ and $3 \log(1 + o(1))/(\log \nu) \in o(1)$.
- According to Lemma 3.6 and the fact that $\tilde{b} \in O(1)$, we know that $\log \tilde{b} \in O(1)$ so $(3 \log \tilde{b})/(\log \nu) \in o(1)$.
- The expressions that involve \tilde{d} are more tricky as Step 1 offers no guarantee about its behaviour. In fact, using similar arguments than in Lemma 3.6, the equation of the constraint together with the properties of \tilde{b} and \tilde{a} ensure that \tilde{d} is bounded below and above as well. It allows to further simplify the expressions above.

Using Proposition 3.4, we deduce that after a division by $\nu^{1/3} (\log \nu)^{2/3}$ the equation of the constraint can be rewritten:

$$-\frac{1}{3\tilde{b}\tilde{d}}(1 + o(1)) - \frac{1}{3\tilde{b}} \left(\tilde{a}\tilde{d} + \frac{1}{\tilde{d}} \right) (1 + o(1)) + 2\tilde{a} - \tilde{b} = 0.$$

Clearing the denominators yields the desired equality. \square

Remember that (a, b, d) must minimize $\max(a, b)$, so $(\tilde{a}, \tilde{b}, \tilde{d})$ must minimize $\max(\tilde{a}, \tilde{b})$. If we knew that \tilde{a}, \tilde{b} and \tilde{d} had limits $\bar{a}, \bar{b}, \bar{c}$ in $+\infty$, Lemma 3.7 would ensure that $(\bar{a}, \bar{b}, \bar{d})$ is necessarily solution of the following optimization problem:

Problem 3.8. *Find $(x, y, z) \in (\mathbb{R}_{>0})^3$ which minimize the function $f(x, y, z) = \max(x, y)$ under the constraint $g(x, y, z) = xz^2 + 2 + 3y^2z - 6xyz = 0$.*

Let us study the minimizers for the latter problem. First, we study the minimizers (x, y, z) for Problem 3.8 in the half-plane $x > y$: In this area, we have $f(x, y) = x$. If (x, y, z) is a minimizer for Problem 3.8 in the half-plane $x > y$, then according to the method of Langrange multipliers, the gradient ∇L of the Lagrangian function $L : (x, y, z, \lambda) \mapsto x + \lambda g(x, y, z)$ must be zero in (x, y, z) . In particular:

$$\begin{aligned}\frac{\partial L}{\partial x}(x, y, z) &= 1 + \lambda z(z - 6y) = 0 \\ \frac{\partial L}{\partial y}(x, y, z) &= 6\lambda z(x - y) = 0.\end{aligned}$$

The second equation implies that either λ , z or $x - y$ must be zero. But $\lambda = 0$ simplifies the first equation into $1 = 0$ so this is impossible, we know that $z > 0$ and we assumed that $x > y$. It means that there are no solutions to $\nabla L(x, y, z) = 0$ if $x > y$ so there is no solution (x, y, z) to Problem 3.8 such that $x > y$. Using the same method, we prove that there is no solution (x, y, z) to Problem 3.8 such that $x < y$. As a result, if (x, y, z) is a solution of Problem 3.8 it must necessarily satisfy $x = y$. Thus, we are left with the following problem: Find $(x, z) \in (\mathbb{R}_{>0})^2$ such that x is minimal and such that the constraint $xz^2 + 2 - 3x^2z = 0$ is satisfied.

Again, we use the method of Lagrange multipliers: If (x, z) is solution to the above optimisation problem, then the gradient of the Lagrangian function $L : (x, z, \lambda) \mapsto x + \lambda(xz^2 + 2 - 3x^2z)$ vanishes at (x, z) . It means that:

$$\begin{aligned}\frac{\partial L}{\partial x}(x, z) &= 1 + \lambda z(z - 6x) = 0 \\ \frac{\partial L}{\partial z}(x, z) &= \lambda x(2z - 3x) = 0 \\ \frac{\partial L}{\partial \lambda}(x, z) &= xz^2 + 2 - 3x^2z = 0.\end{aligned}$$

The second equation implies that either λ , x or $2z - 3x$ is equal to zero. But $x > 0$, and if $\lambda = 0$ it contradicts the first equation so we must have $z = 3x/2$. Replacing z by this expression in the third equation yields $9x^3 = 8$ so $x = (8/9)^{1/3}$. The corresponding value for z is $3^{1/3}$. Conversely, $((8/9)^{1/3}, (8/9)^{1/3}, 3^{1/3})$ does minimize x under the constraint $xz^2 + 2 - 3x^2z = 0$. Therefore, we showed that there is only one solution to Problem 3.8, namely $((8/9)^{1/3}, (8/9)^{1/3}, 3^{1/3})$.

If e_1, e_2 are two reals, let us denote by $P(e_1, e_2)$ the following optimization problem, which is a perturbation of Optimization Problem 3.8: Find $(x, y, z) \in (\mathbb{R}_{>0})^3$ which minimize the function $f(x, y, z) = \max(x, y)$ under the constraint $g_{e_1, e_2}(x, y, z) = xz^2(1 + e_1) + 2 + e_2 + 3y^2z - 6xyz = 0$. We show that the only solution of $P(e_1, e_2)$ is the triple (x, y, z) such that $x = y = \frac{2}{3}z(1 + e_1)$ and $z = \left(\frac{3(2+e_2)}{2(1+e_1)^2}\right)^{1/3}$ using exactly the same arguments than when we solved $P(0, 0)$, that is Optimization Problem 3.8.

Lemma 3.7 ensures that for a given ν , $(\tilde{a}(\nu), \tilde{b}(\nu), \tilde{d}(\nu))$ is a solution of $P(\varepsilon_1(\nu), \varepsilon_2(\nu))$ which proves that:

$$\tilde{a}(\nu) = \tilde{b}(\nu) = \frac{2}{3}\tilde{d}(\nu)(1 + \varepsilon_1(\nu)) \text{ and } \tilde{d}(\nu) = \left(\frac{3(2 + \varepsilon_2(\nu))}{2(1 + \varepsilon_1(\nu))^2}\right)^{1/3}$$

Since ε_1 and ε_2 tend to zero at infinity, it proves both that $\tilde{a}, \tilde{b}, \tilde{d}$ have limits $\bar{a}, \bar{b}, \bar{d}$ at infinity and that $(\bar{a}, \bar{b}, \bar{d}) = ((8/9)^{1/3}, (8/9)^{1/3}, 3^{1/3})$. But Step 1 ensures that $(\bar{a}, \bar{b}, \bar{d}) = ((8/9)^{1/3}, (8/9)^{1/3}, 3^{1/3})$ is indeed possible. This allows to finally conclude this proof. \square

Remark 3.9. *The classical complexity analysis of NFS often starts by assuming that*

$$\begin{aligned}d(\nu) &= D(\nu/\log \nu)^\delta(1 + o(1)), \\ a(\nu) &= A\nu^\alpha(\log \nu)^{1-\alpha}(1 + o(1)), \\ b(\nu) &= B\nu^\beta(\log \nu)^{1-\beta}(1 + o(1)).\end{aligned}$$

where $A, B, D, \alpha, \beta, \delta$ are unknown positive constants. It proceeds by plugging these expressions for a, b, d into the constraint to deduce the unknowns, leaving the reader uncomfortable as to why it is possible to assume a given shape for a, b, d and why this one in particular is chosen. Proposition 3.5 shows there is actually no need to assume a special shape for a, b, d as it is uncovered during the first order analysis and it is a direct consequence of the shape of the equation of the constraint and of Canfield-Erdős-Pomerance formula. As to why looking for a, b, d of the form $\kappa \nu^k (\log \nu)^{k'} (1 + o(1))$ in the first place, this is actually inspired by the first order formula for smoothness probabilities in Proposition 3.4 which displays the same shape and transfers it to the equation of the constraint (3.2).

Assuming that $a = b$ is also quite common in the complexity analysis. The argument for this simplification is that if $a > b$, then decreasing a and increasing b would lead to a smaller value for $\max(a, b)$. It forgets however that a and b are linked by the equation of the constraint and that it is quite possible that the latter does not allow a to decrease while b increases or vice versa. Fortunately, the hypothesis $a = b$ is actually not needed.

The way we obtain the first term of NFS asymptotic complexity inspires us a method to get more terms: We could get more precision in the expansion of the equation of Constraint (3.2) and hope to be able to apply the reasonings of the previous proof. But to expand the constraint, we first need a tool to expand the probabilities of smoothness involved in it, and this is the topic of the next section.

3.3 Expansion of smoothness probabilities

3.3.1 Useful classes of functions

In following sections we will encounter multiple times two functions, which we denote by $\mathcal{X} : \eta \mapsto \log_2 \eta / \log \eta$ and $\mathcal{Y} : \eta \mapsto 1 / \log \eta$. We will often omit the argument of the functions \mathcal{X} and \mathcal{Y} .

The notation $\mathbb{R}[[X, Y]]$ is used for bivariate formal series with real coefficients, and for $\mathbf{S} \in \mathbb{R}[[X, Y]]$ and $i \in \mathbb{Z}_{\geq 0}$, the notation $\mathbf{S}^{(i)}$ stands for the truncation of \mathbf{S} to total degree less than or equal to i .

We introduce the following class of functions, to capture the asymptotic behavior of several functions of interest at infinity.

Definition 3.10. *The class of functions \mathcal{C} is the set of functions f defined over a neighborhood of $+\infty$ with values in \mathbb{R} such that*

$$\exists \mathbf{F} \in \mathbb{R}[[X, Y]], \forall n f(\eta) = \mathbf{F}^{(n)}(\mathcal{X}, \mathcal{Y}) + o(\mathcal{Y}^n)$$

We say that \mathbf{F} is the series associated to f .

Definition 3.10 deserves several comments. First, it is important to notice that the truncation $\mathbf{F}^{(n)}$ that appears in Definition 3.10 is necessary, as the *evaluation* of \mathbf{F} at a given value need not make sense: the series \mathbf{F} is only intended to capture the asymptotic behavior of the function f at infinity. Next, the map from $f \in \mathcal{C}$ to its associated series \mathbf{F} is clearly not injective, since for example the function $1/\eta$ is in \mathcal{C} , and its associated series is zero. In fact, an extension of Borel's lemma shows that the link from $f \in \mathcal{C}$ to \mathbf{F} is surjective: Any series, regardless of any notion of convergence, can be realized as the asymptotic expansion of some function in \mathcal{C} . Even though this result is not needed to proceed, we prove it below for completeness.

Proposition 3.11. *The map from \mathcal{C} to $\mathbb{R}[[X, Y]]$ is surjective.*

Proof. Let $S = \sum_{n \geq 0} \sum_{i+j=n} a_{i,j} X^i Y^j$ be an element of $\mathbb{R}[[X, Y]]$.

Let e denote a function of class C^∞ such that $e(x) \in [0, 1]$ for all real x , $e(x) = 0$ for $x \leq 0$ and $e(x) = 1$ for all $x \geq 1$. The function e is the counterpart of the bump function used in the proof of the classical Borel's lemma.

For any $n \geq 0$, let $B_n = \sum_{i+j=n} |a_{i,j}|$, $A_n = \exp \exp B_n$ and $e_n : x \mapsto e(x - A_n)$.

We show that the function

$$f : \eta \mapsto \sum_{n \geq 0} \sum_{i+j=n} a_{i,j} e_{i+j}(\eta) \mathcal{X}(\eta)^i \mathcal{Y}(\eta)^j$$

is an antecedent of S .

First, f is well-defined for $\eta \geq e^e$ (which is enough for our purposes as f need only be defined over a neighborhood of $+\infty$). Let $\eta \geq e^e$ and for $n \geq 0$ let $u_n(\eta) = \sum_{i+j=n} a_{i,j} e_{i+j}(\eta) \mathcal{X}(\eta)^i \mathcal{Y}(\eta)^j$.

We show that the series $\sum u_n(\eta)$ is absolutely convergent.

If n is such that $\eta \leq A_n = e^{e^{B_n}}$ then $e_n(\eta) = 0$ so $u_n(\eta) = 0$ too. Otherwise, we have $B_n < \log_2(\eta)$ and then $|u_n(\eta)| \leq \frac{1}{(\log \eta)^n} \sum_{i+j=n} |a_{i,j}| \log_2(\eta)^i \leq \frac{(\log_2 \eta)^n}{(\log \eta)^n} B_n \leq (\log_2 \eta) \frac{(\log_2 \eta)^n}{(\log \eta)^n}$. As a result, for all $n \geq 0$ we have $|u_n(\eta)| \leq (\log_2 \eta) \left(\frac{\log_2 \eta}{\log \eta} \right)^n$ and $\sum \left(\frac{\log_2 \eta}{\log \eta} \right)^n$ is a convergent geometric series which concludes this first part of the proof.

Now we prove that $f \in \mathcal{C}$. Let $n_0 \geq 0$ and let us study $R(\eta) = \sum_{n > n_0} \sum_{i+j=n} a_{i,j} e_{i+j}(\eta) \mathcal{X}(\eta)^i \mathcal{Y}(\eta)^j$. For all $\eta \geq e^e$ we have:

$$|R(\eta)| \leq \sum_{n > n_0} B_n |e_n(\eta)| \mathcal{X}(\eta)^n$$

as $\mathcal{Y}(\eta) \leq \mathcal{X}(\eta)$. But $|e_n(\eta)| \leq B_n^{-1} \log_2(\eta)$ by construction. Indeed, if $\eta \leq A_n$, the left-hand side of this inequality is zero and the right-hand side is positive since $\eta \geq e^e$. And if $\eta \geq A_n$, $e_n(\eta) \leq 1$ and by definition of A_n , $B_n^{-1} \log_2(\eta) \geq 1$. In the end, after a change of indices:

$$|R(\eta)| \leq \mathcal{X}(\eta)^{n_0+1} \log_2(\eta) \underbrace{\sum_{n \geq 0} \mathcal{X}(\eta)^n}_{< +\infty \text{ since } \mathcal{X}(\eta) < 1 \text{ for } \eta \geq e^e}$$

This proves that $R(\eta) = O(\mathcal{X}(\eta)^{n_0+1} \log_2(\eta))$ when η goes to infinity and implies that $R(\eta) = o(\mathcal{Y}(\eta)^{n_0})$. As a result, we have indeed for all $n_0 \geq 0$ and for all η sufficiently large:

$$f(\eta) = \sum_{n \leq n_0} \sum_{i+j=n} a_{i,j} \mathcal{X}(\eta)^i \mathcal{Y}(\eta)^j + o(\mathcal{Y}(\eta)^{n_0})$$

□

In order to express conveniently our results on the asymptotic expansion of the function ξ , we also introduce a family of variants of the class \mathcal{C} .

Definition 3.12. Let $\alpha, \beta \in \mathbb{R}$. The class of functions $\mathcal{C}^{[\alpha, \beta]}$ is the set of functions f with values in \mathbb{R} such that $\nu \mapsto \nu^{-\alpha} (\log \nu)^{-\beta} f(\nu)$ is in \mathcal{C} .

In particular, $\mathcal{C}^{[0,0]} = \mathcal{C}$. If f is in $\mathcal{C}^{[\alpha, \beta]}$, then we denote by $\mathbf{F} \in \mathbb{R}[[X, Y]]$ the series associated to $\nu \mapsto \nu^{-\alpha} (\log \nu)^{-\beta} f(\nu)$ and call it, by extension, the series associated to f . Note however that this extension deserves some caution, as the association makes sense only relative to a given (α, β) .

The introduction of these classes $\mathcal{C}^{[\alpha, \beta]}$ is motivated by the fact that in the context of NFS, the function $u \mapsto -\log \rho(u)$ where ρ is the Dickman-De Bruijn function, is in $\mathcal{C}^{[1,1]}$ (see Corollaries 3.23 and 3.24 below). This is of importance as this function is linked to smoothness probabilities as we will see in the next sections. The following proposition establishes stability properties for elements of the classes $\mathcal{C}^{[\alpha, \beta]}$.

Proposition 3.13. *Let $f \in \mathcal{C}^{[\alpha, \beta]}$ with associated series $\mathbf{F} \in \mathbb{R}[[X, Y]]$ such that $\alpha > 0$ and $\mathbf{F}(0, 0) > 0$. Then*

1. $\log f \in \mathcal{C}^{[0, 1]}$ and its associated series is $\alpha + \beta X + (Y \log \mathbf{F})$;
2. $\mathcal{Y}(f) = (\log f)^{-1} \in \mathcal{C}$ and its associated series is $Y/(\alpha + \beta X + (Y \log \mathbf{F}))$;
3. if $\alpha > 0$, then $\mathcal{X}(f) = \log \log f / \log f \in \mathcal{C}$ and its associated series is

$$(X + Y \log(\alpha + \beta X + (Y \log \mathbf{F}))/(\alpha + \beta X + (Y \log \mathbf{F})).$$

where $\log \mathbf{F}$ is the formal series $\log \mathbf{F}(0, 0) - \sum_{i \geq 1} \frac{(1 - \mathbf{F}/\mathbf{F}(0, 0))^i}{i}$.

Proof. This result follows from direct computations. We prove the first statement as an example.

Let $n \in \mathbb{Z}_{\geq 0}$. We know that $f(\eta) = \eta^\alpha (\log \eta)^\beta (\mathbf{F}^{(n)}(\mathcal{X}(\eta), \mathcal{Y}(\eta)) + o(\mathcal{Y}(\eta)^n))$. Then

$$\begin{aligned} \log f &= \alpha \log(\eta) + \beta \log_2 \eta + \log \left(\mathbf{F}(0, 0) \left(\frac{\mathbf{F}^{(n)}}{\mathbf{F}(0, 0)} + o(\mathcal{Y}^n) \right) \right) \\ &= \log(\eta) \left(\alpha + \beta \frac{\log_2 \eta}{\log \eta} + \frac{1}{\log \eta} \left(\log \mathbf{F}(0, 0) + \log \left(1 + \underbrace{\frac{\mathbf{F}^{(n)} - \mathbf{F}(0, 0)}{\mathbf{F}(0, 0)}}_{\rightarrow 0 \text{ when } \eta \rightarrow +\infty} + o(\mathcal{Y}^n) \right) \right) \right) \\ &= \log(\eta) (\alpha + \beta \mathcal{X} + \mathcal{Y} \cdot ((\log \mathbf{F})^{(n)} + o(\mathcal{Y}^n))) \end{aligned}$$

by using a first order Taylor expansion of the logarithm. □

3.3.2 Hildebrand and De Bruijn formulas

The precision of an asymptotic expansion of the NFS complexity is tightly connected to the precision in results regarding smoothness probabilities. Canfield, Erdős and Pomerance actually prove a better result than Proposition 3.4.

Theorem 3.14 (Canfield, Erdős and Pomerance [CEP83]). *For all $x \geq 1$ and for all $u = x/y \geq 3$, we have*

$$p(x, y) \geq -u \log u \cdot p(u),$$

where $p(u) = 1 + \mathcal{X}(u) - \mathcal{Y}(u) + \mathcal{X}(u)\mathcal{Y}(u) - \mathcal{Y}(u)^2 + O(\mathcal{X}(u)^2 \mathcal{Y}(u))$.

In fact, it turns out that Theorem 3.14 is at the same time too strong and too weak for our purposes. On the one hand, it is too bad that the asymptotic expansion of the right-hand side of the inequality stops at some point. On the other hand it is a really strong result since it is true for basically all x, u without any restriction. But in the NFS context, we know the magnitudes of x, u from the first order analysis: We do not need a smoothness result that would be unconditionally true. A better approximation of the smoothness probability in a narrower range would suit us. Hildebrand proves such a result.

Theorem 3.15 (Hildebrand [Hil86]). *Let $\varepsilon > 0$. For $1 \leq u \leq \exp((\log y)^{3/5-\varepsilon})$ and $x = y^u$, we have*

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

where ρ is the Dickman–de Bruijn function.

Under the Riemann hypothesis this result even holds in a wider range. In the NFS context we are in the appropriate range to apply Theorem 3.15. Indeed, based on Proposition 3.5, we expect to use Theorem 3.15 in a context where $\log y = b = \Theta(\nu^{1/3}(\log \nu)^{2/3})$ and $u = \Theta((\nu/\log \nu)^{1/3})$: u is polynomial in $\log y$, while the bound in Theorem 3.15 is subexponential. Asymptotically, the result of Theorem 3.15 is therefore very precise. This leads us to study more precisely the expansion of ρ .

In [DB51a], De Bruijn proves the following formula :

$$\rho(u) \underset{u \rightarrow +\infty}{\sim} \frac{e^\gamma}{\sqrt{2\pi u}} \times \exp\left(-\int_0^\xi \frac{se^s - e^s + 1}{s} ds\right)$$

for all $u > 1$ and $\xi = (e^u - 1)/u$. Let $\eta = (e^s - 1)/s$, so that $s = \log(1 + s\eta)$. By substitution we have:

$$\int_0^\xi \frac{se^s - e^s + 1}{s} ds = \int_1^u s d\eta.$$

The path to expanding the probabilities of smoothness involved in equation 3.2 is now clear :

1. Study the function s .
2. Use the information on s to expand its integral.
3. Deduce the expansion of $\log \rho$ from the previous step and De Bruijn formula.
4. Trace back to the expansion of smoothness probabilities using Hildebrand formula.

3.3.3 Asymptotic development of smoothness probabilities

Using Hildebrand and De Bruijn formulas, this section is devoted to obtaining explicitly computable expansions of smoothness probabilities using the strategy depicted at the end of Section 3.3.2. Although no unusual mathematical tool is used in the following proofs, this section is technical. In an attempt to demystify the general formulas presented below and emphasize that all of them can be algorithmically computed, we will provide examples of expansions with a few terms before the presentation of the general cases. To help navigate this section, we detail its organization below:

1. First, we expand the function s defined by the equation $s(\eta) = \log(1 + s(\eta)\eta)$. Example 3.16 presents an expansion for this function with two terms together with the main ideas used in Proposition 3.17. The latter is a generalization of example 3.16. Proposition 3.18 shows that there is actually an explicit series formula for the expansion of s and Proposition 3.19 explores its convergence domain.
2. Next, we expand the function $u \mapsto \int_e^u s(\eta) d\eta$. Example 3.20 presents an expansion for this function with five terms and Proposition 3.22 generalizes the process, leaning on the technical Lemma 3.21.
3. Finally we use the general formula for the expansion of $u \mapsto \int_e^u s(\eta) d\eta$ together with De Bruijn and Hildebrand formulas to conclude about the expansion of functions in link with smoothness probabilities in Corollaries 3.23 and 3.24.

Proposition 3.17 gives a method to expand the function s defined by the equation: $s(\eta) = \log(1 + s(\eta)\eta)$. It relies on one main tool : the asymptotic expansion of the function $\eta \mapsto \log(1 + \eta)$ around 0. We show how in the following example.

Example 3.16. *We have:*

$$s(\eta) = (\log \eta)(1 + \mathcal{X}(\eta) + o(\mathcal{Y}(\eta))).$$

Proof. Let us admit that $s = \log(\eta)(1 + o(1))$ when $\eta \rightarrow +\infty$. This is properly done in Proposition 3.17 and we will take it for granted here. Once this is done, we can replace s by this first order approximation in the right-hand term that defines it:

$$\begin{aligned}
s &= \log(1 + s\eta) \\
&= \log(1 + \eta(\log \eta)(1 + o(1))) \text{ (replace in the right-hand term)} \\
&= \log \left(\eta(\log \eta) \left[1 + \frac{1}{\eta(\log \eta)} + o(1) \right] \right) \text{ (factor by leading coefficient)} \\
&= \log \eta + \log_2 \eta + \log(1 + o(1)) \text{ (use logarithm properties)} \\
&= \log \eta + \log_2 \eta + o(1) \text{ (expand the residual logarithm)} \\
&= (\log \eta)(1 + \mathcal{X}(\eta) + o(\mathcal{Y}(\eta)))
\end{aligned}$$

□

This yields an expansion for s with a new term that we can immediately plug back into the right-hand term to reiterate the process. Thus we can recursively compute the expansion of s and even prove it has a special shape.

Proposition 3.17. *The function $\eta \rightarrow s(\eta)/\log \eta$ is in \mathcal{C} .*

Proof. We prove by induction on n that there exists $P_n \in \mathbb{R}[X, Y]$ such that, as $\eta \rightarrow +\infty$, we have $s = \log \eta \cdot (P_n(\mathcal{X}, \mathcal{Y}) + o(\mathcal{Y}^n))$.

First we show that $s = (\log \eta)(1 + o(1))$ when $\eta \rightarrow +\infty$, which is to say $P_0 = 1$. Indeed $\varphi : s \mapsto (e^s - 1)/s$ is bijective on $\mathbb{R}_{>0}$ and strictly increasing, and so is φ^{-1} . Since $\varphi(\log \eta) = (\eta - 1)/\log \eta < \eta = \varphi(s)$, we have $\log \eta < s$. Let now $\varepsilon > 0$. When η is large enough, we have $\eta^{1+\varepsilon} > (1 + \varepsilon)\eta \log \eta + 1$. This leads to $\varphi(s) < \varphi((1 + \varepsilon)\log \eta)$, hence to $s < (1 + \varepsilon)\log \eta$ and proves the base case.

We now proceed with the induction step. Since $s = \log(1 + s\eta)$, we may write

$$\begin{aligned}
s &= \log s + \log \eta + \log(1 + 1/(s\eta)) \\
&= \log \eta \cdot (1 + \log s / \log \eta + o(1/\eta)) \\
&= \log \eta \cdot (1 + \mathcal{X} + \mathcal{Y} \cdot \log P_n(\mathcal{X}, \mathcal{Y}) + o(\mathcal{Y}^{n+1})),
\end{aligned}$$

where the last expression omits some terms that are swallowed by $o(\mathcal{Y}^{n+1})$. Since the constant coefficient of P_n is 1, an expansion of the inner logarithm above to n terms gives the desired result. More precisely, one can verify that P_{n+1} is the truncation to total degree at most $n + 1$ of

$$1 + X - Y \sum_{i=1}^n (-1)^i \frac{(P_n - 1)^i}{i}.$$

□

Let us notice that the proof of the previous proposition also gives a method to algorithmically compute any expansion of s using only operations on polynomials.

In fact, a more explicit version of the series associated to $s(\eta)/\log \eta$ can be computed:

Proposition 3.18. *Letting \mathbf{P} denote the series associated to $s(\eta)/\log \eta$, we have*

$$\mathbf{P}(X, Y) = 1 + X + Y \cdot \sum_{i=1}^{+\infty} \sum_{j=1}^i \frac{S(i, i-j+1)}{j!} X^j Y^{i-j}$$

where the $S(i, j)$ are signed Stirling numbers of the first kind.

Proof. By [Com70] (see also [Com74, Sec. 5, Exercise 22]), the reciprocal $g(y)$ of $f : x \mapsto e^x/x$ has the following asymptotic series expansion at the neighborhood of ∞ :

$$\log y + \log_2 y + \sum_{i=1}^{+\infty} \sum_{j=1}^i \frac{S(i, i-j+1)}{j!} \frac{(\log_2 y)^j}{(\log y)^i}.$$

By definition, $s(y)$ is the reciprocal of the function $h : x \mapsto e^x/x - 1/x$. We will show that $s(y) - g(y) \sim (y \log y)^{-1}$ as $y \rightarrow +\infty$. This will conclude our proof since it will ensure that for all $n \in \mathbb{Z}_{\geq 0}$,

$$s(y) = \log y + \log_2 y + \sum_{i=1}^n \sum_{j=1}^i \frac{S(i, i-j+1)}{j!} \frac{(\log_2 y)^j}{(\log y)^i} + o\left(\frac{1}{(\log y)^n}\right)$$

as y grows.

First, using the fact that $h(s(y)) = e^{s(y)}y^{-1} - s(y)^{-1} = y$, we get that $f(s(y)) - f(g(y)) = e^{s(y)}s(y)^{-1} - y = s(y)^{-1}$. By the mean value theorem, there exists θ_y between $g(y)$ and $s(y)$ such that $s(y)^{-1} = f(s(y)) - f(g(y)) = f'(\theta_y)(s(y) - g(y))$. Since $s(y) = \log y + \log_2 y + o(1)$ according to Example 3.16 and $g(y) = \log y + \log_2 y + o(1)$, we deduce that $\theta_y = \log y + \log_2 y + o(1)$ when $y \rightarrow +\infty$, hence $f'(\theta_y) \sim y$. Consequently, $s(y) - g(y) \sim (ys(y))^{-1} \sim (y \log y)^{-1}$. \square

Using the previous result we can even give insight on the convergence domain of the series associated to s .

Proposition 3.19. *The series*

$$\sum_{i=1}^{+\infty} \sum_{j=1}^i \frac{S(i, i-j+1)}{j!} \mathcal{X}^j \mathcal{Y}^{i-j}$$

converges uniformly for $\eta \in [176, +\infty[$.

Proof. First, for all $k \in \mathbb{Z}_{\geq 0}$ and all $\eta > e^e$ we have $\mathcal{Y} \leq \mathcal{X}$, so that

$$\sum_{i=1}^k \sum_{j=1}^i \frac{s(i, i-j+1)}{j!} \mathcal{X}^j \mathcal{Y}^{i-j} \leq \sum_{i=1}^k \sum_{j=1}^i \frac{s(i, i-j+1)}{j!} \mathcal{X}^i = \sum_{i=1}^k c_i \mathcal{X}^i$$

where $c_i = \sum_{j=1}^i \frac{s(i, i-j+1)}{j!}$. Let $a_i = i! \sum_{j=1}^i \left| \frac{s(i, i-j+1)}{j!} \right|$. According to the asymptotic formula for the sequence A138013 of the OEIS¹, we have, as $i \rightarrow +\infty$:

$$a_i \sim \sqrt{-1 - w(-e^{-2})} \times (-w(-e^{-2}))^i \times i^{i-1}/e^i$$

where $w = W_{-1}$ and W denotes the Lambert W function, so that

$$\frac{a_i/i!}{a_{i+1}/(i+1)!} \xrightarrow{i \rightarrow \infty} \frac{1}{-w(-e^{-2})}.$$

This shows that the power series with coefficients $a_i/i!$ has a finite radius of convergence equal to $-1/w(-e^{-2})$. Since $|c_i| \leq a_i/i!$, the power series $\sum c_i X^i$ also has finite radius of convergence, which is at most as large as the former. Therefore the series converges uniformly for $\frac{\log \log \eta}{\log \eta} \leq -1/w(-e^{-2})$, and this inequality is satisfied for $\log \log \eta \geq -w(1/w(-e^{-2}))$ i.e., for $\eta \geq 176$. \square

¹<https://oeis.org/A138013>

We now shift gears and study the asymptotic behavior of $\int_e^u s d\eta$ as u grows. One can notice that the lower bound of the previous integral is e instead of 1 as in De Bruijn's formula. In fact the lower bound of this integral has no importance as any constant term will be swallowed by the small o involved in its expansions: We choose e to ensure its positivity. Intuitively, we are going to replace s by its expansions to obtain expansions of its integral. The main tool to go from an expansion of s to an expansion of its integral is integration by parts, using 1 as one of the parts.

In order to better illustrate the tools involved and stress the fact that the expansions of $\int_e^u s d\eta$ can be computed explicitly, we detail the computation of the expansion of $\int_e^u s d\eta$ that can be derived from an expansion of s with three terms.

Example 3.20. *We have*

$$\int_e^u s d\eta = u(\log u) \left(1 + \mathcal{X}(u) - \mathcal{Y}(u) + \mathcal{X}(u)\mathcal{Y}(u) - \mathcal{Y}(u)^2 + o(\mathcal{Y}(u)^2) \right).$$

Proof. According to Proposition 3.18, $s(\eta) = \log \eta + \log_2 \eta + \log_2 \eta / \log \eta + o(1/\log \eta)$. As a result:

$$\int_e^u s d\eta = \left[\eta \left(\log \eta + \log_2 \eta + \frac{\log_2 \eta}{\log \eta} \right) \right]_e^u - \int_e^u \left(1 + \frac{1}{\log \eta} + \frac{1}{(\log \eta)^2} - \frac{\log_2 \eta}{(\log \eta)^2} + o\left(\frac{1}{\log \eta}\right) \right) d\eta$$

after one integration by parts using 1 as one of the parts. Some of the terms in the remaining integral are swallowed by the $o(1/\log \eta)$ and the others are integrated by parts again, using 1 as one of the parts :

$$\int_e^u s d\eta = \left[\eta \left(\log \eta + \log_2 \eta + \frac{\log_2 \eta}{\log \eta} \right) \right]_e^u - \left[\eta \left(1 + \frac{1}{\log \eta} \right) \right]_e^u + \int_e^u \left(\frac{-1}{(\log \eta)^2} + o\left(\frac{1}{\log \eta}\right) \right) d\eta$$

This time, all the terms in the residual integral are swallowed by the $o(1/\log \eta)$ and we are left with:

$$\int_e^u s d\eta = \left[\eta \left(\log \eta + \log_2 \eta + \frac{\log_2 \eta}{\log \eta} - 1 - \frac{1}{\log \eta} \right) \right]_e^u + \int_e^u o\left(\frac{1}{\log \eta}\right) d\eta$$

Let us first deal with the residual integral. As we will see in Lemma 3.21 we straightforwardly have $\int_e^u o(1/\log \eta) d\eta = o(u/\log u)$. As a result this small o will swallow all the constant terms that come from the evaluation of the expression inside the brackets in e and we finally have:

$$\int_e^u s d\eta = u \left(\log u + \log_2 u - 1 + \frac{\log_2 u}{\log u} - \frac{1}{\log u} + o\left(\frac{1}{\log u}\right) \right)$$

which can conveniently be rewritten in function of \mathcal{X} and \mathcal{Y} :

$$\int_e^u s d\eta = u(\log u) \left(1 + \mathcal{X}(u) - \mathcal{Y}(u) + \mathcal{X}(u)\mathcal{Y}(u) - \mathcal{Y}(u)^2 + o(\mathcal{Y}(u)^2) \right)$$

□

Before generalizing this proof, we prove the small lemma used to deal with the residual integral in the above example.

Lemma 3.21. *For all $n \geq 0$, if $f \in o\left(\frac{1}{(\log \eta)^n}\right)$ is continuous then,*

$$\int_e^u f(\eta) d\eta = o\left(\frac{u}{(\log u)^n}\right)$$

Proof. Let $n \geq 0$ and $\varepsilon > 0$. By definition, there is an bound $M \geq e$ such that for all $\eta \geq M$ we have $|f(\eta)(\log \eta)^n| \leq \varepsilon$.

On the one hand

$$\int_e^M f(\eta) d\eta = o\left(\frac{u}{(\log u)^n}\right)$$

since the left integral is a constant.

On the other hand, for all $u \geq M$

$$\left| \int_M^u f(\eta) d\eta \times \frac{(\log u)^n}{u} \right| \leq \int_e^u \frac{\varepsilon}{u} d\eta = \varepsilon \left(1 - \frac{M}{u}\right)$$

and the right expression can be rendered as close to ε as wanted if u is sufficiently large. It proves that

$$\int_M^u f(\eta) d\eta = o\left(\frac{u}{(\log u)^n}\right)$$

and summing both integrals concludes. \square

Let us notice that we will satisfy the continuity hypothesis of this lemma in Proposition 3.22 as all the small o involved in our expansions can in fact be expressed in terms of the continuous functions $\log \eta$, $\log_2 \eta$ and $s(\eta)$.

Proposition 3.22. *The function $u \rightarrow \frac{1}{u \log u} \cdot \int_e^u s d\eta$ is in \mathcal{C} , and the coefficients of its associated series \mathbf{Q} can be computed explicitly.*

Proof. We prove that for all $n \in \mathbb{Z}_{\geq 0}$, there is a polynomial Q_n such that, as $u \rightarrow +\infty$:

$$\int_e^u s d\eta = u \log u \cdot (Q_n(\mathcal{X}(u), \mathcal{Y}(u))) + o(\mathcal{Y}(u)^n),$$

and that for all $i, j \geq 0$, $Q_i(X, Y) \equiv Q_j(X, Y) \pmod{\langle X, Y \rangle^{\min(i, j)}}$. We emphasize that our proof provides an explicit method to compute Q_n .

As previously stated, we are free to choose the lower bound of the integral as we are looking for an asymptotic expansion of a divergent integral as $u \rightarrow \infty$, up to terms that also tend to infinity.

Let Δ be the \mathbb{R} -linear operator defined on $\mathbb{R}[X, Y]$ by $\Delta 1 = 0$, $\Delta X = Y \cdot (Y - X)$, $\Delta Y = -Y^2$, and $\Delta(UV) = U\Delta V + (\Delta U)V$. One can check that:

$$\forall T \in \mathbb{R}[X, Y], \quad \frac{d}{d\eta} T(\mathcal{X}, \mathcal{Y}) = \frac{1}{\eta} \cdot (\Delta T)(\mathcal{X}, \mathcal{Y}).$$

Notice that $\Delta \mathbb{R}[X, Y] \subset Y \mathbb{R}[X, Y]$.

We use the properties of Δ to prove an intermediate result. Let T be an arbitrary bivariate polynomial in $\mathbb{R}[X, Y]$. Using the above notation, repeated integration by parts yields:

$$\begin{aligned} \int_e^u T(\mathcal{X}, \mathcal{Y}) d\eta &= [\eta T(\mathcal{X}, \mathcal{Y})]_e^u - \int_e^u (\Delta T)(\mathcal{X}, \mathcal{Y}) d\eta \\ &= \sum_{i=0}^{n-1} (-1)^i [\eta \cdot (\Delta^i T)(\mathcal{X}, \mathcal{Y})]_e^u + (-1)^n \int_e^u (\Delta^n T)(\mathcal{X}, \mathcal{Y}) d\eta \\ &= [\eta \cdot R(\mathcal{X}, \mathcal{Y})]_e^u + \int_e^u o(\mathcal{Y}^{n-1}) d\eta \end{aligned}$$

for $R = \sum_{i=0}^{n-1} (-1)^i \Delta^i T \in \mathbb{R}[X, Y]$. Note that R is a truncation of $(1 + \Delta)^{-1} T$. This allows us to quickly conclude. Indeed, using Proposition 3.17, for all $n \in \mathbb{Z}_{\geq 0}$:

$$\begin{aligned} \int_e^u s d\eta &= [(\eta \log \eta - \eta) \mathbf{P}^{(n)}(\mathcal{X}, \mathcal{Y})]_e^u + \\ &\quad \int_e^u (1 - \log \eta) \Delta(\mathbf{P}^{(n)})(\mathcal{X}, \mathcal{Y}) d\eta \end{aligned}$$

Since $(1 - \log \eta) \Delta(\mathbf{P}^{(n)})(\mathcal{X}, \mathcal{Y})$ can be written as $T_n(\mathcal{X}, \mathcal{Y})$ for some $T_n \in \mathbb{R}[X, Y]$, the previous result shows that there exists $R_n \in \mathbb{R}[X, Y]$ such that

$$\int_e^u s d\eta = [\eta \log \eta \cdot (1 - \mathcal{Y}) \cdot \mathbf{P}^{(n)}(\mathcal{X}, \mathcal{Y})]_e^u + [\eta \cdot R_n(\mathcal{X}, \mathcal{Y})]_e^u + \int_e^u o(\mathcal{Y}^{n-1}) d\eta.$$

The claimed expression, with $Q_n = (1 - Y) \mathbf{P}^{(n)} + Y \cdot R_n$, follows from lemma 3.21 since it ensures that $\int_e^u o(1/(\log u)^{n-1}) d\eta = o(u/(\log u)^{n-1})$ as $u \rightarrow +\infty$. Finally, we notice that for all $i, j \geq 0$, $R_i(X, Y) \equiv R_j(X, Y) \pmod{\langle X, Y \rangle^{\min(i, j)}}$, which implies that $Q_i(X, Y) \equiv Q_j(X, Y) \pmod{\langle X, Y \rangle^{\min(i, j)}}$.

The series \mathbf{Q} has therefore the following expression, which makes it easy to compute \mathbf{Q} from \mathbf{P} .

$$\mathbf{Q} = (1 - Y) \mathbf{P} + Y(1 + \Delta)^{-1}(1 - Y^{-1}) \Delta \mathbf{P}.$$

□

Propositions 3.17 and 3.18 are completely explicit. In Proposition 3.22, the expression of \mathbf{Q} is straightforward to compute as well. For example, the terms of \mathbf{Q} up to degree 3, are:

$$\mathbf{Q}^{(3)}(X, Y) = 1 + X - Y + XY - Y^2 - \frac{X^2 Y}{2} + 2XY^2 - 2Y^3.$$

We now show that computing the series \mathbf{Q} immediately yields expansions of smoothness probabilities up to any precision. To do so, we prove that for any integer n the residual factors in Hidebrand formula and De Bruijn formula are swallowed by $o(\mathcal{Y}(u)^n)$.

Corollary 3.23. *We recall that \mathbf{Q} is the series introduced in Proposition 3.22. For all $n \in \mathbb{Z}_{\geq 0}$ we have, as $u \rightarrow +\infty$:*

$$\rho(u) = \exp \left(-u \log u \left(\mathbf{Q}^{(n)}(\mathcal{X}(u), \mathcal{Y}(u)) + o(\mathcal{Y}(u)^n) \right) \right).$$

Proof. The equivalent of ρ introduced at the very beginning of the section ensures that:

$$\rho(u) = \frac{e^\gamma}{\sqrt{2\pi u}} \times \exp \left(- \int_1^u s d\eta \right) \times (1 + o_{u \rightarrow +\infty}(1))$$

Taking the logarithm of this equality and using Proposition 3.22 yields when u goes to infinity:

$$\begin{aligned} \log \rho(u) &= \gamma - \frac{1}{2} \log(2\pi u) - \int_1^e s d\eta - \int_e^u s d\eta + \log(1 + o(1)) \\ &= - \int_e^u s d\eta + O(\log u) \\ &= -u \log u \cdot \left(\mathbf{Q}^{(n)}(\mathcal{X}(u), \mathcal{Y}(u)) + o(\mathcal{Y}(u)^n) + O\left(\frac{1}{u}\right) \right) \\ &= -u \log u \cdot \left(\mathbf{Q}^{(n)}(\mathcal{X}(u), \mathcal{Y}(u)) + o(\mathcal{Y}(u)^n) \right) \end{aligned}$$

as the error term that comes from the expansion of $\int_e^u s d\eta$ absorbs $O(1/u)$ at infinity.

□

Said otherwise, the function $u \mapsto -\log \rho(u)$ is in $\mathcal{C}^{[1,1]}$ and its associated series is in \mathbf{Q} .

Corollary 3.24. *In the optimal parameter range of NFS, the asymptotic expansion of the smoothness probabilities follows from the previous result. For any $n \in \mathbb{Z}_{\geq 0}$ we have, on both sides (rational and algebraic), as $N \rightarrow +\infty$:*

$$\Psi(M_i, B)/B = \exp\left(-u \log u \left(\mathbf{Q}^{(n)}(\mathcal{X}(u), \mathcal{Y}(u)) + o(\mathcal{Y}(u)^n)\right)\right),$$

where u denotes $\log M_i / \log B$, which is the size ratio on side $i \in \{0, 1\}$.

Proof. As we argued when justifying the use of Theorem 3.15, Proposition 3.5 tells us that the optimal parameter range of NFS leads to $u = \Theta((\nu / \log \nu)^{1/3})$, with $\nu = \log N$. It follows that $b = \Theta(u \log u)$, so that the right-hand side of Theorem 3.15 can be written as

$$\begin{aligned} \rho(u) \left(1 + O\left(\frac{1}{u}\right)\right) &= \rho(u) \exp\left(-u \log u \cdot O\left(\frac{1}{u^2 \log u}\right)\right) \\ &= \exp\left(-u \log u \left(\mathbf{Q}^{(n)}(\mathcal{X}(u), \mathcal{Y}(u)) + o(\mathcal{Y}(u)^n)\right)\right). \end{aligned}$$

Put otherwise, a function within $O\left(\frac{1}{\eta^2 \log \eta}\right)$ is always in \mathcal{C} , and its associated series is zero. \square

3.4 Further terms in the asymptotic expansion of NFS complexity

This section is devoted to the analysis of the asymptotic behavior of the unknown function ξ involved in the heuristic complexity of NFS. To this end, we compute the asymptotic expansions of the functions a, b, d solutions of the minimization problem introduced in 3.3: We will often call them *the minimizers*. The computation of the asymptotic expansion of ξ relies on three algorithms described in Section 3.4.2. Our implementation in SageMath of these algorithms is available at the following URL

https://gitlab.inria.fr/NFS_asymptotic_complexity/simulations.

3.4.1 Two new proven terms in the asymptotic development

In this section we begin expanding the functions of interest a, b, d . The end result of this section is an asymptotic expansion of the complexity of NFS with two new terms. We also introduce several reasonings that will be intensively used and automatized in Section 3.4.2.

Theorem 3.25. *The minimizers a, b, d satisfy :*

$$\begin{aligned} a &= (8/9)^{1/3} \nu^{1/3} (\log \nu)^{2/3} (1 + a_{10} \mathcal{X}(\nu) + a_{01} \mathcal{Y}(\nu) + o(\mathcal{Y}(\nu))), \\ b &= (8/9)^{1/3} \nu^{1/3} (\log \nu)^{2/3} (1 + a_{10} \mathcal{X}(\nu) + a_{01} \mathcal{Y}(\nu) + o(\mathcal{Y}(\nu))), \\ d &= (3\nu / \log \nu)^{1/3} (1 + d_{10} \mathcal{X}(\nu) + d_{01} \mathcal{Y}(\nu) + o(\mathcal{Y}(\nu))), \end{aligned}$$

where $a_{10} = 4/3, a_{01} = -2 \log 2 + \log 3/6 - 2, d_{10} = -2/3$ and $d_{01} = \log 2 - 5 \log 3/6 + 1$.

Before proving Thm. 3.25, we show that there exist functions a, b, d as in Thm. 3.25 which satisfy Eq. (3.2). They will serve as a baseline for our minimization problem. One could wonder where the constants a_{ij} and d_{ij} in the statement of Thm. 3.25, and also in the following lemma, come from. To obtain these constants, we used a computer algebra system to iteratively expand the constraint (3.2) and then we minimized the coefficients of the expansions of a, b, d by hand. In the rest of the section, we will always omit the argument ν of the functions \mathcal{X} and \mathcal{Y} .

Lemma 3.26. *There exist functions a, b, d which satisfy Eq. (3.2) and such that*

$$\begin{aligned} a &= (8/9)^{1/3} \nu^{1/3} (\log \nu)^{2/3} (1 + a_{10} \mathcal{X} + a_{01} \mathcal{Y} + a_{20} \mathcal{X}^2 + a_{11} \mathcal{X} \mathcal{Y} + a_{02} \mathcal{Y}^2 + o(\mathcal{Y}^2)) \\ b &= (8/9)^{1/3} \nu^{1/3} (\log \nu)^{2/3} (1 + a_{10} \mathcal{X} + a_{01} \mathcal{Y} + a_{20} \mathcal{X}^2 + a_{11} \mathcal{X} \mathcal{Y} + a_{02} \mathcal{Y}^2 + o(\mathcal{Y}^2)) \\ d &= (3\nu / \log \nu)^{1/3} (1 + d_{10} \mathcal{X} + d_{01} \mathcal{Y} + o(\mathcal{Y})), \end{aligned}$$

where

$$\begin{aligned} a_{10} &= 4/3, & a_{01} &= -2 \log 2 + \log 3/6 - 2, \\ a_{20} &= -4/9, & a_{11} &= 4 \log 2/3 - \log 3/9 + 4, \\ a_{02} &= -(\log 2)^2 + (\log 2 \cdot \log 3)/6 - 7(\log 3)^2/36 - 6 \log 2 + \log 3/2 - 5, \\ d_{10} &= -2/3, & \text{and } d_{01} &= \log 2 - 5 \log 3/6 + 1. \end{aligned}$$

Proof. Set

$$\begin{aligned} a &= (8/9)^{1/3} \nu^{1/3} (\log \nu)^{2/3} (1 + a_{10} \mathcal{X} + a_{01} \mathcal{Y} + a_{20} \mathcal{X}^2 + a_{11} \mathcal{X} \mathcal{Y} + a_{02} \mathcal{Y}^2 + \tilde{a} \mathcal{X}^3) \\ b &= (8/9)^{1/3} \nu^{1/3} (\log \nu)^{2/3} (1 + a_{10} \mathcal{X} + a_{01} \mathcal{Y} + a_{20} \mathcal{X}^2 + a_{11} \mathcal{X} \mathcal{Y} + a_{02} \mathcal{Y}^2) \\ d &= (3\nu / \log \nu)^{1/3} (1 + d_{10} \mathcal{X} + d_{01} \mathcal{Y}), \end{aligned}$$

where $a_{10}, a_{01}, a_{20}, a_{11}, a_{02}, d_{10}, d_{01}$ are as in the lemma, and \tilde{a} is an unknown function of the variable ν .

Given these expressions, we wish to rewrite Eq. (3.2) as a function of \tilde{a} . This is particularly tedious, but straightforward. The only needed tools are formulas in Prop. 3.13 over the function field $\mathbb{R}(\tilde{a})$. The code repository mentioned in the introduction of this section shows how the expansion can be carried out with a computer algebra system (and the same holds for other calculations in this section). We obtain that Equation (3.2) with the functions set above can be rephrased as:

$$(\tilde{a} - 32/81) = \varepsilon(\tilde{a}, \nu),$$

for a continuous function ε such that for all $t \in \mathbb{R}$, $\lim_{\nu \rightarrow \infty} \varepsilon(t, \nu) = 0$. Let now $t_- = 31/81$ and $t_+ = 33/81$. Since $\varepsilon(t_-, \nu)$ and $\varepsilon(t_+, \nu)$ both tend to zero, we can define ν_- and ν_+ such that

$$\begin{aligned} \forall \nu > \nu_-, & (t_- - 32/81) - \varepsilon(t_-, \nu) < 0, \\ \forall \nu > \nu_+, & (t_+ - 32/81) - \varepsilon(t_+, \nu) > 0. \end{aligned}$$

By the intermediate value theorem, we obtain that for any $\nu > \max(\nu_-, \nu_+)$, there exists $t \in [t_-, t_+]$ such that $(t - 32/81) = \varepsilon(t, \nu)$. Let now \tilde{a} be the function of ν that returns such a number t . Then the functions a , b , and d defined above satisfy by construction the desired property. \square

Proof of Theorem 3.25. The roadmap of the proof is the following:

1. We express the constraint (3.2) using a sufficiently precise asymptotic expansion of the smoothness probability (given by Corollary 3.24) and the asymptotic expansions of a, b, d known so far. Then we prove that the $o(1)$ involved in the asymptotic expansions of a, b, d are actually in the class $O(\mathcal{X}^\lambda \mathcal{Y}^\mu)$ for some $\lambda, \mu \geq 0$ (not both being zero) so that we can write these $o(1)$ as $C \cdot \mathcal{X}^\lambda \mathcal{Y}^\mu$ where C is a function bounded at a neighbourhood of ∞ .
2. We prove that $C = c + o(1)$ where c is a constant computed along the way and restart the whole process using the more precise asymptotic expansions for a, b, d that we have just obtained in order to compute the next coefficients.

Step 1: By Proposition 3.5, minimizers can be written as

$$\begin{aligned} a &= (8/9)^{1/3} \nu^{1/3} (\log \nu)^{2/3} (1 + \tilde{a}), \\ b &= (8/9)^{1/3} \nu^{1/3} (\log \nu)^{2/3} (1 + \tilde{b}), \\ d &= (3\nu / \log \nu)^{1/3} (1 + \tilde{d}), \end{aligned}$$

where $\tilde{a}, \tilde{b}, \tilde{d} = o(1)$.

Direct computations and simplifications (that involve Taylor series expansions, Prop. 3.13, and the asymptotic expansion of the smoothness probability in Corollary 3.24) that take into account the fact that $\tilde{a}, \tilde{b}, \tilde{d} = o(1)$ rephrase Eq. (3.2) as

$$\tilde{a} = \frac{2}{3}\tilde{b}^2 + \frac{1}{3}\tilde{d}^2 + O(\mathcal{X}).$$

The last equation shows that $\tilde{a}\mathcal{X}^{-1}$ is bounded below by a finite constant. Moreover, Lemma 3.26 ensures the existence of functions a_0, b_0, d_0 that can be used as substitutes for $\tilde{a}, \tilde{b}, \tilde{d}$ above, that satisfy the constraint (3.2), and such that $\lim a_0\mathcal{X}^{-1}$ exists and is finite. Since a, b, d are minimizers of Problem 3.3, $\tilde{a}\mathcal{X}^{-1}$ is also upper bounded by $a_0\mathcal{X}^{-1}$. Therefore, $\tilde{a} \in O(\mathcal{X})$, whence the same also holds for \tilde{b}^2 and \tilde{d}^2 .

Replacing $\tilde{a}, \tilde{b}, \tilde{d}$ respectively by $\bar{a}\mathcal{X}, \bar{b}\mathcal{X}^{\frac{1}{2}}, \bar{d}\mathcal{X}^{\frac{1}{2}}$ for some functions $\bar{a}, \bar{b}, \bar{d}$ bounded at a neighborhood of $+\infty$ in Eq. (3.2), we obtain that

$$-\bar{a} + \frac{2}{3}\bar{b}^2 + \frac{1}{3}\bar{d}^2 + \frac{4}{3} = o(1),$$

which means in particular that $\bar{a} \geq 4/3 + o(1)$. By minimality of a we must also have $\bar{a} \leq 4/3 + o(1)$ so as not to contradict Lemma 3.26. So $\bar{a} = 4/3 + o(1)$ and we must necessarily have $\bar{b} = \bar{d} = o(1)$. Therefore, we obtain

$$\begin{aligned} a &= (8/9)^{1/3} \nu^{1/3} (\log \nu)^{2/3} (1 + 4\mathcal{X}/3 + \tilde{a}\mathcal{X}), \\ b &= (8/9)^{1/3} \nu^{1/3} (\log \nu)^{2/3} (1 + \tilde{b}\mathcal{X}^{\frac{1}{2}}), \\ d &= (3\nu/\log \nu)^{1/3} (1 + \tilde{d}\mathcal{X}^{\frac{1}{2}}), \end{aligned}$$

for some fresh functions $\tilde{a}, \tilde{b}, \tilde{d} = o(1)$.

Step 2: We use the result obtained in Step 1 and the asymptotic expansion of the smoothness probability (Corollary 3.24) to deduce by direct computations the following equality enforced by Eq. (3.2):

$$\left(-\tilde{a} + \frac{2}{3}\tilde{b}^2 + \frac{1}{3}\tilde{d}^2\right) \cdot \mathcal{X} = O(\mathcal{Y}).$$

Following the same reasoning as in Step 1, $\tilde{a}\mathcal{X}\mathcal{Y}^{-1}$, $\tilde{b}\mathcal{X}^{\frac{1}{2}}\mathcal{Y}^{-\frac{1}{2}}$ and $\tilde{d}\mathcal{X}^{\frac{1}{2}}\mathcal{Y}^{-\frac{1}{2}}$ must be bounded at a neighborhood of $+\infty$. Replacing \tilde{a} (resp. \tilde{b}, \tilde{d}) by $\bar{a}\mathcal{X}^{-1}\mathcal{Y}$ (resp. $\bar{b}\mathcal{X}^{-\frac{1}{2}}\mathcal{Y}^{\frac{1}{2}}$ and $\bar{d}\mathcal{X}^{-\frac{1}{2}}\mathcal{Y}^{\frac{1}{2}}$) for some functions $\bar{a}, \bar{b}, \bar{d}$ bounded at a neighborhood of $+\infty$, we obtain the following equality:

$$-\bar{a} + \frac{2\bar{b}^2}{3} + \frac{\bar{d}^2}{3} - 2\log 2 + \frac{\log 3}{6} - 2 = o(1).$$

By minimality and using our baseline result Lemma 3.26 as we did in Step 1, the equalities $\bar{a} = -2\log 2 + \log 3/6 - 2 + o(1)$, $\bar{b} = o(1)$, and $\bar{d} = o(1)$ must hold, which means that:

$$\begin{aligned} a &= (8/9)^{1/3} \nu^{1/3} (\log \nu)^{2/3} (1 + 4\mathcal{X}/3 + (-2\log 2 + \log 3/6 - 2)\mathcal{Y} + \tilde{a}\mathcal{Y}), \\ b &= (8/9)^{1/3} \nu^{1/3} (\log \nu)^{2/3} (1 + \tilde{b}\mathcal{Y}^{\frac{1}{2}}), \\ d &= (3\nu/\log \nu)^{1/3} (1 + \tilde{d}\mathcal{Y}^{\frac{1}{2}}), \end{aligned}$$

for some fresh functions $\tilde{a}, \tilde{b}, \tilde{d} = o(1)$.

Step 3: Again we use the asymptotic expansion obtained in Step 2 to refine the asymptotic equality from Eq. (3.2) and obtain:

$$\begin{aligned} &\left(\left(-2\log(2) + \frac{5\log(3)}{3} - 2 + o(1)\right)\tilde{d} + \left(8\log(2) - \frac{2\log(3)}{3} + 2 + o(1)\right)\tilde{b}\right)\frac{\mathcal{Y}^{\frac{3}{2}}}{3} \\ &+ \left(-\tilde{a} + \frac{\tilde{d}^2}{3} + \frac{2\tilde{b}^2}{3}\right)\mathcal{Y} + \left(\frac{4\tilde{d}}{9} - \frac{16\tilde{b}}{9}\right)\mathcal{X}\mathcal{Y}^{\frac{1}{2}} = O(\mathcal{X}^2). \end{aligned}$$

where $o(1)$ are explicit expressions in $\tilde{a}, \tilde{b}, \tilde{d}$, which we omit for brevity.

This can be rephrased as

$$\underbrace{\frac{1}{3} \left(\tilde{d} \mathcal{Y}^{\frac{1}{2}} + \frac{2}{3} \mathcal{X} + \left(-\log(2) + \frac{5 \log(3)}{6} - 1 + o(1) \right) \mathcal{Y} \right)^2}_{=\delta(\nu)} + \underbrace{\frac{2}{3} \left(\tilde{b} \mathcal{Y}^{\frac{1}{2}} - \frac{4}{3} \mathcal{X} + \left(2 \log(2) - \frac{\log(3)}{6} + \frac{1}{2} + o(1) \right) \mathcal{Y} \right)^2}_{=\beta(\nu)} - \tilde{a} \mathcal{Y} = O(\mathcal{X}^2).$$

We prove now that $\beta(\nu) = O(\mathcal{X}^2)$ and $\delta(\nu) = O(\mathcal{X}^2)$. Assume by contradiction that this does not hold. Then $\tilde{a}(\nu)$ is positive asymptotically and it cannot belong to the class $O(\mathcal{X}^2)$. This would contradict our upper bound for the minimum given in Lemma 3.26. Consequently, $\beta(\nu)$ and $\delta(\nu)$ belong to $O(\mathcal{X}^2)$ and then so does $\tilde{a} \mathcal{Y}$. This means that $\tilde{a} = O(\mathcal{X}^2 \mathcal{Y}^{-1})$ and $\tilde{b}, \tilde{d} = O(\mathcal{X} \mathcal{Y}^{-\frac{1}{2}})$. As usual we call $\bar{a}, \bar{b}, \bar{d}$ the functions $\tilde{a} \mathcal{X}^{-2} \mathcal{Y}, \tilde{b} \mathcal{X}^{-1} \mathcal{Y}^{\frac{1}{2}}$ and $\tilde{d} \mathcal{X}^{-1} \mathcal{Y}^{\frac{1}{2}}$ bounded at ∞ , and we substitute in Eq. (3.2) to get

$$\left(-\bar{a} - \frac{4}{9} \right) + \frac{2}{3} \left(\bar{b} - \frac{4}{3} \right)^2 + \frac{1}{3} \left(\bar{d} + \frac{2}{3} \right)^2 = o(1). \quad (3.5)$$

Lemma 3.26 ensures that we must have $\bar{a} \leq -4/9 + o(1)$, otherwise it would contradict the minimality of a . The equation above ensures that we must have $\bar{a} \geq -4/9 + o(1)$ as well. So $\bar{a} = -4/9 + o(1)$, which implies that $\bar{b} = 4/3 + o(1)$ and $\bar{d} = -2/3 + o(1)$. This third step of this proof gives:

$$\begin{aligned} a &= (8/9)^{1/3} \nu^{1/3} (\log \nu)^{2/3} (1 + 4 \mathcal{X} / 3 + (-2 \log 2 + \log 3 / 6 - 2) \mathcal{Y} - 4 \mathcal{X}^2 / 9 + \tilde{a} \mathcal{X}^2), \\ b &= (8/9)^{1/3} \nu^{1/3} (\log \nu)^{2/3} (1 + 4 \mathcal{X} / 3 + \tilde{b} \mathcal{X}), \\ d &= (3\nu / \log \nu)^{1/3} (1 - 2 \mathcal{X} / 3 + \tilde{d} \mathcal{X}), \end{aligned}$$

for some fresh unknown functions $\tilde{a}, \tilde{b}, \tilde{d} = o(1)$.

Step 4: Substituting the asymptotic expansion obtained in Step 3 yields

$$\left(-\tilde{a} + \frac{2\tilde{b}^2}{3} + \frac{\tilde{d}^2}{3} \right) \cdot \mathcal{X}^2 = O(\mathcal{X} \mathcal{Y}).$$

As in Step 1, \tilde{a} must belong to $O(\mathcal{X}^{-1} \mathcal{Y})$ and \tilde{b}, \tilde{d} to $O(\mathcal{X}^{-\frac{1}{2}} \mathcal{Y}^{\frac{1}{2}})$ so as not to contradict Lemma 3.26. Using the notations $\bar{a}, \bar{b}, \bar{d}$ for the asymptotically bounded functions $\tilde{a} \mathcal{X} \mathcal{Y}^{-1}, \tilde{b} \mathcal{X}^{\frac{1}{2}} \mathcal{Y}^{-\frac{1}{2}}$ and $\tilde{d} \mathcal{X}^{\frac{1}{2}} \mathcal{Y}^{-\frac{1}{2}}$, we find by substitution in Eq. (3.2) that $\bar{b} = o(1), \bar{d} = o(1)$ and $\bar{a} = 4 \log 2 / 3 - \log 3 / 9 + 4 + o(1)$, ,

$$\begin{aligned} a &= (8/9)^{1/3} \nu^{1/3} (\log \nu)^{2/3} (1 + 4 \mathcal{X} / 3 + (-2 \log 2 + \log 3 / 6 - 2) \mathcal{Y} \\ &\quad - 4 \mathcal{X}^2 / 9 + (4 \log 2 / 3 - \log 3 / 9 + 4) \mathcal{X} \mathcal{Y} + \tilde{a} \mathcal{X} \mathcal{Y}), \\ b &= (8/9)^{1/3} \nu^{1/3} (\log \nu)^{2/3} (1 + 4 \mathcal{X} / 3 + \tilde{b} \mathcal{X}^{\frac{1}{2}} \mathcal{Y}^{\frac{1}{2}}), \\ d &= (3\nu / \log \nu)^{1/3} (1 - 2 \mathcal{X} / 3 + \tilde{d} \mathcal{X}^{\frac{1}{2}} \mathcal{Y}^{\frac{1}{2}}), \end{aligned}$$

for some fresh unknown functions $\tilde{a}, \tilde{b}, \tilde{d} = o(1)$.

Step 5: We expand the constraint for the last time and after a factorization that follows the pattern of Step 3 we get:

$$\begin{aligned} &\frac{1}{3} \left(\tilde{d} \mathcal{X}^{\frac{1}{2}} + \left(-\log 2 + \frac{5 \log 3}{6} - 1 \right) \mathcal{Y}^{\frac{1}{2}} \right)^2 \mathcal{Y} + \\ &\frac{2}{3} \left(\tilde{b} \mathcal{X}^{\frac{1}{2}} + \left(2 \log 2 - \frac{\log 3}{6} + \frac{1}{2} \right) \mathcal{Y}^{\frac{1}{2}} \right)^2 \mathcal{Y} - \tilde{a} \mathcal{X} \mathcal{Y} = O(\mathcal{Y}^2). \end{aligned}$$

Again, we must have $\tilde{a} = O(\mathcal{X}^{-1} \mathcal{Y})$ and $\tilde{b}, \tilde{d} = O(\mathcal{X}^{-\frac{1}{2}} \mathcal{Y}^{\frac{1}{2}})$, so as not to contradict Lemma 3.26. We also compute the associated limits for \tilde{b} and \tilde{d} by using the same method as in the previous steps: we let $\bar{a}, \bar{b}, \bar{d}$ denote the bounded functions $\tilde{a} \mathcal{X} \mathcal{Y}^{-1}, \tilde{b} \mathcal{X}^{\frac{1}{2}} \mathcal{Y}^{-\frac{1}{2}}, \tilde{d} \mathcal{X}^{\frac{1}{2}} \mathcal{Y}^{-\frac{1}{2}}$. Direct computations yield

$$\begin{aligned} & \left[\frac{2}{3} \left(\bar{b} + 2 \log 2 - \frac{\log 3}{6} + \frac{1}{2} \right)^2 - \frac{3}{2} \right] + \frac{1}{3} \left(\bar{d} + \frac{5 \log 3}{6} - \log 2 - 1 \right)^2 = \\ & \left(\bar{a} - \left(-(\log 2)^2 + \frac{\log 2 \log 3}{6} - \frac{7(\log 3)^2}{3} - 6 \log 2 + \frac{\log 3}{2} - 5 \right) \right) + o(1). \end{aligned}$$

Lemma 3.26 ensures that $\bar{b} \leq a_{01} + o(1)$ and $\bar{a} \leq a_{02} + o(1)$. This implies that the lefthand side of the equality is bounded below by some function in $o(1)$ and hence $\lim \bar{a} = a_{02}$. Therefore we get that

$$\begin{aligned} \lim \tilde{b} \mathcal{X}^{\frac{1}{2}} \mathcal{Y}^{-\frac{1}{2}} &= -2 \log 2 + \log 3/6 - 2, \\ \lim \tilde{d} \mathcal{X}^{\frac{1}{2}} \mathcal{Y}^{-\frac{1}{2}} &= \log 2 - 5 \log 3/6 + 1, \end{aligned}$$

which concludes the proof. \square

Corollary 3.27. *Let $\Phi(\nu) = \max(a(\nu), b(\nu))$ be the quantity minimized by Problem 3.3. The heuristic complexity $C(N)$ to factor an integer N with NFS satisfies:*

$$\log C(N) = \sqrt[3]{\frac{64}{9}} (\log N)^{1/3} (\log_2 N)^{1/3} \left(1 + a_{10} \frac{\log_3 N}{\log_2 N} + \frac{a_{01}}{\log_2 N} + o\left(\frac{1}{\log_2 N}\right) \right)$$

where $a_{10} = 4/3$ and $a_{01} = -2 \log 2 + \log 3/6 - 2$.

This follows both from Theorem 3.25 and the fact that the $O(\log b)$ that should have been taken into account in the non-simplified optimization problem of which NFS is solution (see Section 3.1) is a $O(\log_3 N)$ which end up swallowed by the $o(1/\log_2 N)$. A natural question to ask is whether the process used in the proof of Thm. 3.25 can be continued. As we will see in the next section, the answer to this question is yes.

3.4.2 Generalization of the method, algorithms

In this section, we describe how the arguments of Section 3.4.1 can be turned into three algorithms that allow to compute more precise asymptotic expansions for the minimizers a, b, d . These three algorithms take as input the precision requested for the asymptotic expansion of the complexity and they mirror the three steps used in the proof of Theorem 3.25:

- Algorithm GUESSTERMS guesses the asymptotic expansion of the minimizers of Problem 3.3.
- Algorithm PROVEEXISTENCE proves the existence of functions satisfying the constraint in Problem 3.3, and whose asymptotic expansion is the output of Algorithm GUESSTERMS. This establishes an upper bound for the minimum of the objective function in Problem 3.3.
- Algorithm PROVEMINIMALITY proves that the asymptotic expansion of the minimizers of Problem 3.3 must be the output of Algorithm GUESSTERMS.

These algorithms are used in the following way. We set a degree $n > 1$. The three algorithms are used to prove that $\xi(N) = Q(\mathcal{X}(\log N), \mathcal{Y}(\log N)) + o(\mathcal{Y}(\log N)^n)$ where Q is a bivariate polynomial of total degree n , whose coefficients are computed along the way. We emphasize that these algorithms might fail, *i.e.*, become unable to compute or prove new terms at some point. Indeed, one of the cornerstones of our algorithms is that, when expanding the equation of the constraint (3.2) using the results of Section 3.3.3, this expansion will always have a shape that

follows one of three patterns. If it does not, then it cannot proceed. We were unable to prove that our algorithms never fail, but experimentally they never did. And even if one of them does, the terms of the complexity computed up until the failure point are guaranteed to be correct. We also point out that all the bivariate polynomials that we consider in these algorithms have coefficients in $\mathbb{Q}(\log 2, \log 3)$, so they can be described exactly without having to rely on floating-point computations.

We now describe more precisely the algorithms `GUESSTERMS`, `PROVEEXISTENCE` and `PROVEMINIMALITY` assuming there is no failure.

Algorithm `GUESSTERMS` assumes that a, b, d belong respectively to the classes $\mathcal{C}^{[1/3, 2/3]}$, $\mathcal{C}^{[1/3, 2/3]}$, $\mathcal{C}^{[1/3, -1/3]}$ and even that $a = b$. We call $\mathbf{A}, \mathbf{B}, \mathbf{D}$ the bivariate series associated to a, b, d . The reason that motivates looking for a, b, d in these classes is the fact that the equation of the constraint builds on a function in $\mathcal{C}^{[1, 1]}$ and we hope that this shape spreads to the minimizers. First, we expand Eq. (3.2) based on the asymptotic expansions of a, b and d (initialized thanks to the result of Theorem 3.25). Then we minimize the leading term of the asymptotic expansion of the objective function in Problem 3.3 under the constraint that the coefficient of the main term in the asymptotic expansion of Eq. (3.2) must vanish. This is done thanks to Taylor series expansions at infinity and bivariate series computations at finite precision. From this, we deduce the next coefficients of our series. The repetition of this process provides an algorithm to iteratively compute the coefficients of the series $\mathbf{A}, \mathbf{B}, \mathbf{D}$. These series will be our candidates for the asymptotic expansion of the minimizers of Problem 3.3.

Algorithm `PROVEEXISTENCE` is the counterpart of Lemma 3.26. It checks that there exist functions α, β, δ such that:

- The asymptotic expansion of α is $\alpha = (8/9)^{1/3} \nu^{1/3} (\log \nu)^{2/3} (\mathbf{A}^{(n+1)}(\mathcal{X}, \mathcal{Y}) + o(\mathcal{Y}^{n+1}))$;
- The asymptotic expansion of β is $\beta = (8/9)^{1/3} \nu^{1/3} (\log \nu)^{2/3} (\mathbf{B}^{(n+1)}(\mathcal{X}, \mathcal{Y}) + o(\mathcal{Y}^{n+1}))$;
- The asymptotic expansion of δ is $\delta = (3\nu / \log \nu)^{1/3} (\mathbf{D}^{(n+1)/2}(\mathcal{X}, \mathcal{Y}) + o(\mathcal{Y}^{(n+1)/2}))$;
- The functions α, β, δ satisfy the constraint (3.2).

Algorithm `PROVEEXISTENCE` serves the same purpose as Lemma 3.26: Give a baseline result that ensures the existence of functions that satisfy the constraint (3.2) and whose asymptotic expansions are known up to a given degree. In particular, a solution to Problem 3.3 must be smaller than α, β, δ . The algorithm works similarly to the proof of Lemma 3.26 by setting three functions a, b, d where a depends on an unknown function \tilde{a} :

$$\begin{aligned} a &= (8/9)^{1/3} \nu^{1/3} (\log \nu)^{2/3} (\mathbf{A}^{(n+1)}(\mathcal{X}, \mathcal{Y}) + \tilde{a} \mathcal{X}^{n+2}); \\ b &= (8/9)^{1/3} \nu^{1/3} (\log \nu)^{2/3} \mathbf{B}^{(n+1)}(\mathcal{X}, \mathcal{Y}); \\ d &= (3\nu / \log \nu)^{1/3} \mathbf{D}^{(n+1)/2}(\mathcal{X}, \mathcal{Y}). \end{aligned}$$

Then the algorithm checks by using Taylor series expansions that the constraint of Problem 3.3 instantiated with these functions can be rewritten as $(\tilde{a} - \kappa) = \varepsilon(\tilde{a}, \nu)$ for some $\kappa \in \mathbb{R}$ and ε a function as in the proof of Lemma 3.26.

Algorithm `PROVEMINIMALITY` is the counterpart of the proof of Theorem 3.25 and it follows its roadmap. We proceed iteratively with the terms of the asymptotic expansion, computing and proving the expansions of the minimizers a, b, d up to degree n . To do so, we use the current (proven) knowledge of the asymptotic expansions of a, b, d , initialized thanks to Theorem 3.25, and expand the constraint by computing Taylor series expansions at infinity.

To prove that the next terms of the series guessed by Algorithm `GUESSTERMS` are correct, we proceed as follows. First, we prove that the remainder of the series expansion, which has form $o(\mathcal{X}^{\frac{i}{2}} \mathcal{Y}^{\frac{j}{2}})$, is in fact of the form $O(\mathcal{X}^{\frac{i'}{2}} \mathcal{Y}^{\frac{j'}{2}})$, where (i', j') is strictly larger than (i, j) for the graded lexicographical ordering, so that $O(\mathcal{X}^{\frac{i'}{2}} \mathcal{Y}^{\frac{j'}{2}})$ is a proper subset of $o(\mathcal{X}^{\frac{i}{2}} \mathcal{Y}^{\frac{j}{2}})$. This first

step works as long as the equation derived from the constraint follows certain patterns, which are given in Proposition 3.28 below.

In a second step, we prove that this remainder has in fact the form $\kappa \mathcal{X}^{\frac{i'}{2}} \mathcal{Y}^{\frac{j'}{2}} (1 + o(1))$, where κ is the corresponding coefficient in the guessed series. This verification involves an equation that is similar to Eq. (3.5) in the proof of Theorem 3.25. Again, the shape of the equation encountered is crucial to proceed.

Based on our experiments, we surmise (but we cannot prove) that only three patterns occur in the first step, and that the equation encountered in the second step always matches the shape of Eq. (3.5). Algorithm PROVEMINIMALITY only consists in verifying that this holds.

One could be worried that this proving algorithm has to take into account powers of \mathcal{X} and \mathcal{Y} in $\mathbb{Z}/2$. This is not a problem: Our algorithm only has to consider series evaluated in $\sqrt{\mathcal{X}}$ and $\sqrt{\mathcal{Y}}$ instead of \mathcal{X} and \mathcal{Y} . Actually, this behaviour was already encountered in the proof of Theorem 3.25. It so happens that the coefficient of $\mathcal{X}^i \mathcal{Y}^j$ in the asymptotic expansions is experimentally always zero when i or j is not an integer, although we have no proof of this fact.

Proposition 3.28. *Let $\tilde{a}, \tilde{b}, \tilde{d} = o(1)$ be three functions.*

- **Pattern (P1):** *Let $i > 0, j \geq 0$. Assume that $\tilde{a}(\nu) \leq a_0(\nu)$, $\tilde{b}(\nu) \leq b_0(\nu)$, where $a_0, b_0 = O(\mathcal{X}^{-1} \mathcal{Y})$. Assume further that*

$$\left(\frac{\tilde{d}^2}{3} - \tilde{a} - 2\tilde{b} \right) \mathcal{X}^i \mathcal{Y}^j = O(\mathcal{X}^{i-1} \mathcal{Y}^{j+1}).$$

Then $\tilde{a}, \tilde{b} = O(\mathcal{X}^{-1} \mathcal{Y})$ and $\tilde{d} = O(\mathcal{X}^{-\frac{1}{2}} \mathcal{Y}^{\frac{1}{2}})$.

- **Pattern (P2):** *Let $i > 1$ and $\kappa \in \mathbb{R}$. Assume that $\tilde{a}(\nu) \leq a_0(\nu)$, $\tilde{b}(\nu) \leq b_0(\nu)$, where $a_0, b_0 = O(\mathcal{X}^{i+1} \mathcal{Y}^{-i})$. Assume further that*

$$\left(\frac{\tilde{d}^2}{3} - \tilde{a} - 2\tilde{b}(1 + o(1)) \right) \mathcal{Y}^i + \kappa \tilde{d} \mathcal{X}^{\frac{i+1}{2}} \mathcal{Y}^{\frac{i}{2}} (1 + o(1)) = O(\mathcal{X}^{i+1}).$$

Then $\tilde{a} = O(\mathcal{X}^{i+1} \mathcal{Y}^{-i})$, $\tilde{b} \in O(\mathcal{X}^{i+1} \mathcal{Y}^{-i}) \subset O(\mathcal{Y}^{\frac{1}{2}})$, and $\tilde{d} = O(\mathcal{X}^{\frac{i+1}{2}} \mathcal{Y}^{-\frac{i}{2}})$.

- **Pattern (P3):** *Let $i > 2$. Assume that $\tilde{a}(\nu) \leq a_0(\nu)$, $\tilde{b}(\nu) \leq b_0(\nu)$, where $a_0 = O(\mathcal{X}^{-1} \mathcal{Y})$ and $b_0 = O(\mathcal{X}^{i-1} \mathcal{Y}^{-i+\frac{3}{2}})$. Assume further that*

$$\left(\frac{\tilde{d}^2}{3} - \tilde{a} \right) \mathcal{X}^i - 2\tilde{b} \mathcal{Y}^{i-\frac{1}{2}} = O(\mathcal{X}^{i-1} \mathcal{Y}).$$

Then $\tilde{a} = O(\mathcal{X}^{-1} \mathcal{Y})$ and $\tilde{b} = O(\mathcal{X}^{i-1} \mathcal{Y}^{-i+\frac{3}{2}})$, and $\tilde{d} = O(\mathcal{X}^{\frac{1}{2}} \mathcal{Y}^{-\frac{1}{2}})$.

Proof. The proof in all three cases is very similar. We only prove Pattern (P1) as an example.

Dividing the equality by $\mathcal{X}^i \mathcal{Y}^j$, we get

$$\tilde{a} + 2\tilde{b} = \frac{\tilde{d}^2}{3} + O(\mathcal{X}^{-1} \mathcal{Y}),$$

which shows that $\tilde{a} + 2\tilde{b}$ is bounded below by a function in $O(\mathcal{X}^{-1} \mathcal{Y})$. Since $\tilde{a} + 2\tilde{b} \leq a_0 + 2b_0 = O(\mathcal{X}^{-1} \mathcal{Y})$, we deduce that $\tilde{a} + 2\tilde{b} = O(\mathcal{X}^{-1} \mathcal{Y})$. Writing \bar{a}, \bar{b} for $\tilde{a} \mathcal{X} \mathcal{Y}^{-1}, \tilde{b} \mathcal{X} \mathcal{Y}^{-1}$ respectively, we get that $\bar{a} + 2\bar{b} = O(1)$. Together with the fact that \bar{a}, \bar{b} are bounded above by $\limsup a_0 \mathcal{X} \mathcal{Y}^{-1}, \limsup b_0 \mathcal{X} \mathcal{Y}^{-1}$ respectively, this implies that $\bar{a}, \bar{b} \in O(1)$, and hence that $\tilde{a}, \tilde{b} \in O(\mathcal{X}^{-1} \mathcal{Y})$. Hence \tilde{d}^2 must also belong to $O(\mathcal{X}^{-1} \mathcal{Y})$. □

In fact these three patterns do not appear at random during the proof: They appear according to the shape of the remainder $o(\mathcal{X}^i \mathcal{Y}^j)$ in the series expansion of a we are currently considering. We noticed that when $i \neq 0$ and $j \neq 0$ the equation follows pattern **(P1)**, when $i = 0$ it follows pattern **(P2)** and when $j = 0$ it follow pattern **(P3)**. In particular, pattern **(P3)** is always encountered one step after pattern **(P2)** has been encountered. We want to consider that \tilde{b} is in $O(\mathcal{Y}^{\frac{1}{2}})$ instead of the tighter class $O(\mathcal{X}^{i+1} \mathcal{Y}^{-i})$ at the end of pattern **(P2)** precisely to ensure that the next step will yield an equation that follows pattern **(P3)**, provided that the expansion of smoothness probabilities required to do so does not break the chain of patterns.

The three algorithms GUESSTERMS, PROVEEXISTENCE and PROVEMINIMALITY are used consecutively in order to compute a proven asymptotic expansion of a, b, d at a given precision n . We have implemented a function COMPUTEPROVENEXPANSION which performs this process: It takes as input an integer $n > 1$ and — if it does not fail — it returns two bivariate polynomials $\mathbf{A}^{(n+1)}, \mathbf{D}^{(n+1)/2}$ of respective degrees $n + 1$ and $(n + 1)/2$ such that the minimizers a, b, d of Problem 3.3 satisfy

$$\begin{aligned} a &= (8/9)^{1/3} \nu^{1/3} (\log \nu)^{2/3} (\mathbf{A}^{(n+1)}(\mathcal{X}, \mathcal{Y}) + o(\mathcal{Y}^{n+1})), \\ b &= (8/9)^{1/3} \nu^{1/3} (\log \nu)^{2/3} (\mathbf{A}^{(n)}(\mathcal{X}, \mathcal{Y}) + o(\mathcal{Y}^n)), \\ d &= (3\nu / \log \nu)^{1/3} (\mathbf{D}^{(n+1)/2}(\mathcal{X}, \mathcal{Y}) + o(\mathcal{Y}^{(n+1)/2})). \end{aligned}$$

The expansion for b is proven up to a degree one less than the one for a . The underlying reasons are the same as in the proof of Theorem 3.25, where this behavior was first encountered.

3.4.3 General form of the asymptotic expansion of NFS complexity

Theorem 3.29. *Using the general method described in Section 3.4.2 and assuming our algorithms never fail, the shape of the minimizers a, b, d implies that the heuristic complexity of NFS is bounded above by*

$$\exp \left[\sqrt[3]{\frac{64}{9}} (\log N)^{1/3} (\log_2 N)^{2/3} \left(\mathbf{A}^{(n)} \left(\frac{\log_3 N}{\log_2 N}, \frac{1}{\log_2 N} \right) + o \left(\frac{1}{(\log_2 N)^n} \right) \right) \right].$$

Remark 3.30. *Similarly to what happened in Corollary 3.27, the residual factor in $O(\exp(\log b))$ that should intervene in NFS complexity according to Section 3.1 is in $O(\log_3 N / (\log N)^{1/3} (\log_2 N)^{2/3})$ and gets swallowed by the term in $o(1/(\log_2 N)^n)$ in the above expression for any integer n . The series \mathbf{A} mentioned in Theorem 3.29 does correspond to the series \mathbf{A} associated to the minimizer a .*

Our algorithms thus provide an asymptotic expansion of the heuristic complexity $C(N)$ of NFS. More precisely, they output coefficients of a bivariate series $\mathbf{A} \in \mathbb{Q}(\log 2, \log 3)[[X, Y]]$ such that for all $n \geq 0$ such as our algorithms do not fail, we have:

$$\log C(N) = \sqrt[3]{\frac{64}{9}} (\log N)^{1/3} (\log_2 N)^{2/3} \left(\mathbf{A}^{(n)} \left(\frac{\log_3 N}{\log_2 N}, \frac{1}{\log_2 N} \right) + o \left(\frac{1}{(\log_2 N)^n} \right) \right).$$

Here are the first coefficients of the series $\mathbf{A}(X, Y) = \sum_{i,j \geq 0} a_{ij} X^i Y^j$ obtained via our implementation:

a_{00}	1
a_{10}	$4/3$
a_{01}	$-2\log 2 + \log 3/6 - 2$
a_{20}	$-4/9$
a_{11}	$4\log 2/3 - \log 3/9 + 4$
a_{02}	$-(\log 2)^2 + \log 2\log 3/6 - 7(\log 3)^2/36 - 6\log 2 + \log 3/2 - 5$
a_{30}	$32/81$
a_{21}	$-16\log 2/9 + 4\log 3/27 - 56/9$
a_{12}	$8(\log 2)^2/3 - 4\log 2\log 3/9 + 56\log 2/3 + 14(\log 3)^2/27 - 14\log 3/9 + 64/3$
a_{03}	$-4(\log 2)^3/3 + (\log 2)^2\log 3/3 - 14(\log 2)^2 - 7\log 2(\log 3)^2/9$
	$+7\log 2\log 3/3 - 32\log 2 + 41(\log 3)^3/648 - 49(\log 3)^2/18 + 8\log 3/3 - 85/3$

Besides, computing this series \mathbf{A} also yields precise expansions for the function ξ involved in Formula (1.1). Indeed, it can be approximated asymptotically by evaluating at $(\log_3 N/\log_2 N, 1/\log_2 N)$ the truncations of the series $\mathbf{A}(X, Y) - 1$.

Actually, using the algorithms in Section 3.4.2, we were able to compute the series \mathbf{A}, \mathbf{B} and \mathbf{D} associated to the minimizers a, b, d up to degree total 14 (more than a hundred terms). It took a few hours on a personal laptop. The fact that $\mathbf{A} = \mathbf{B}$ has been verified so far backs the claim that the patterns encountered while proving minimality are always as expected. Moreover, despite the fact the algorithms `PROVEEXISTENCE` and `PROVEMINIMALITY` regularly have to consider terms $\mathcal{X}^i \mathcal{Y}^j$ with i or j in $\frac{1}{2}\mathbb{Z}$ in the expansions of a, b, d , as previously mentioned, the coefficients of these terms always turned out to be zero in our experiments. These remarks allow us to formulate the following conjecture:

Conjecture 3.31. *The minimizers a, b, d of Problem 3.3 belong respectively to the classes of functions $\mathcal{C}^{[1/3, 2/3]}$, $\mathcal{C}^{[1/3, 2/3]}$, $\mathcal{C}^{[1/3, -1/3]}$. Moreover, the series $\mathbf{A}, \mathbf{B} \in \mathbb{R}[[X, Y]]$ associated to a, b are equal.*

The Number Field Sieve algorithm has many variants, such as the Special Number Field Sieve (SNFS) that we mentioned in Section 1.2.2, the Multiple Number Field Sieve (MNFS) [BP14, Cop93] or the Tower Number Field Sieve (TNFS) [BGK15] to cite a few. The complexity analysis of these algorithms has many similarities with the analysis of NFS and all of them have an asymptotic complexity that can be expressed as $L_N(\alpha, c + o(1))$ where α, c are constants. It would be interesting to find out if the reasonings we held so far can be adapted to the asymptotic analysis of these algorithms, thus unveiling what hides behind the $o(1)$ in these cases. Beyond NFS, the complexity of several other algorithms is linked to the asymptotics of smoothness probabilities. This is the case for example of the quadratic sieve and its variants, of class group computations in number fields, or of the elliptic curve factoring method. Again, applying the methods of this chapter could help to better understand their asymptotic complexities.

3.5 Relevance of the use of asymptotic expansions for practical estimates

Section 3.4 provides precise asymptotic estimates of the minimizers of Optimization Problem 3.3, which yields refined formulas for the asymptotic complexity of NFS and for the expansions of the function ξ . In particular, assuming the standard heuristics for the complexity of NFS, Theorem 3.25 shows that

$$\xi(N) = \frac{4\log_3 N}{3\log_2 N} + \frac{-2\log 2 + \log 3/6 - 2}{\log_2 N}(1 + o(1)).$$

This asymptotic expansion of ξ can be algorithmically extended to a bivariate series expansion evaluated at $\log_3 N/\log_2 N$ and $1/\log_2 N$. Experimentally, more than a hundred terms of this asymptotic expansion can be computed in a few hours on a personal laptop.

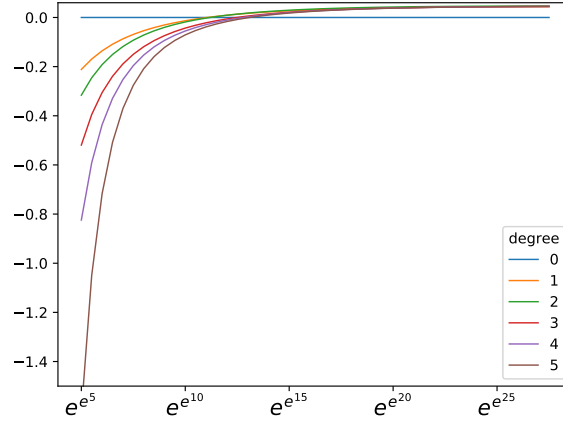


Figure 3.1: Truncations of ξ up to total degree i for $0 \leq i \leq 5$.

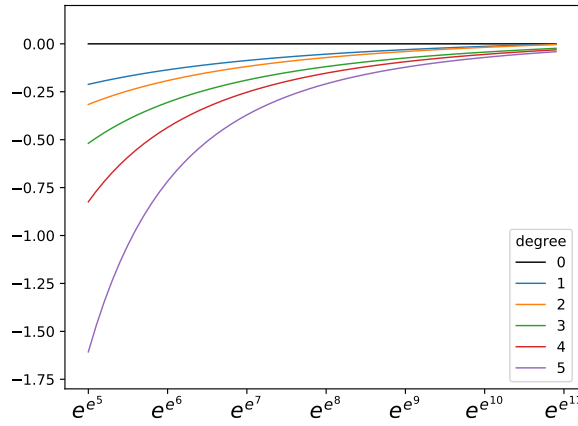


Figure 3.2: Truncations of ξ up to total degree i for $0 \leq i \leq 5$ in function of N for cryptographically relevant values of N .

Since $\xi(N) \sim 4 \log_3 N / (3 \log_2 N)$, the convergence of ξ to zero is very slow, which is somewhat worrying. But overall all of the above still sounds like good news for practical formula-based keysize computations, as it seems that this would provide more precise estimates. In fact, the contrary happens: We observe experimentally that values of N that would be useful for cryptography are far too small compared to the radius of convergence at infinity of the series involved in the expansion of ξ .

For $i \geq 0$, we let $\xi_i(N)$ denote the function $\mathbf{A}^{(i)}(\log_3 N / \log_2 N, 1 / \log_2 N) - 1$ where \mathbf{A} is the series defined in Theorem 3.29. In particular, for all $i \geq 0$, we have $\xi(N) = \xi_i(N) + o(1/(\log_2 N)^i)$. Figure 3.1 displays the behaviour of ξ_i for total degree i between zero and five. Figure 3.2 shows the behavior of ξ_i for cryptographically relevant values of N . Figure 3.3 focuses on the range where we observe experimentally the convergence of the truncations ξ_i . Notice that in all of these figures the abscissa axis is in log log scale.

These figures raise questions on the relevance of the traditional assumption $\xi = 0$ for estimating the complexity of NFS in the range which is useful for cryptographic applications — *i.e.*, $N \leq 2^{20000}$ at the very most — and using the truncated asymptotic formulas for estimating

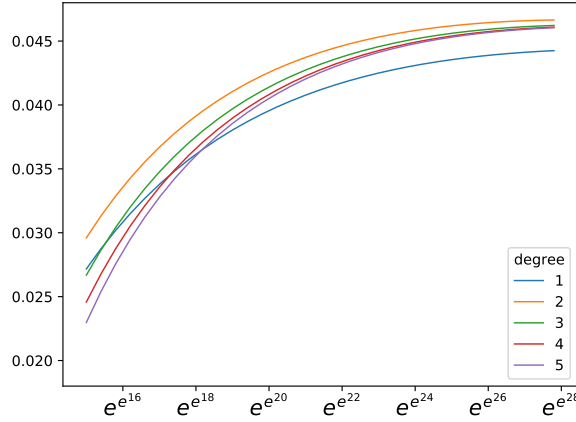


Figure 3.3: Converging behavior for ξ_i .

practical cryptographic key lengths. In particular, we obtain completely different results for the final NFS complexity depending on how many terms we consider in our series expansion of ξ , see Figure 3.2. Besides, we only start to observe convergence for $N > \exp(\exp(22)) \approx 2^{5171935985}$. Thus, it turns out that for practical values of N , replacing ξ by ξ_i for $i > 0$ is possibly even worse than replacing it by zero since the asymptotic series expansion of ξ diverges for $N \leq \exp(\exp(22))$. Said otherwise, this means that **the exponent in the classical formula used for estimating practical RSA key sizes is the first term of a divergent series**. In order to illustrate this phenomenon and its potentially disastrous consequences, let us consider the following example functions:

$$g : N \mapsto \exp\left(\frac{(\log N)^{1/3}(\log_2 N)^{2/3}}{1 + 22/\log_2 N}\right); \quad g_0 : N \mapsto \exp\left((\log N)^{1/3}(\log_2 N)^{2/3}\right);$$

$$g_2 : N \mapsto \exp\left((\log N)^{1/3}(\log_2 N)^{2/3}\left(1 - \frac{22}{\log_2 N} + \frac{22^2}{(\log_2 N)^2}\right)\right).$$

While it is true that g belongs to the class $\exp((\log N)^{1/3}(\log_2 N)^{2/3}(1 + o(1)))$, the radius of convergence of $1/(1+x) = \sum_i (-x)^i$ is 1. This implies that if $N \leq \exp(\exp(22)) \approx 2^{5171935985}$, then equalling $o(1)$ with 0 means estimating the exponent of the function g by evaluating the first term of a divergent series. Numerical computations show that $g(2^{2048}) \approx 2^{15}$, while $g_0(2^{2048}) \approx 2^{61}$, so estimating $o(1)$ by 0 in the case $N = 2^{2048}$ yields completely erroneous results. If one were to compare “projected values” between $N = 2^{512}$ and $N = 2^{2048}$ based on the simplification $o(1) = 0$, the corresponding calculation would yield $g_0(2^{2048})/g_0(2^{512}) \approx 2^{28}$, compared to $g(2^{2048})/g(2^{512}) \approx 2^8$. And approximating this $o(1)$ with more terms absolutely does not help: When $g(2^{2048}) \approx 2^{15}$, $g_2(2^{2048}) \approx 2^{435}$. Of course, in the context of NFS, $\xi(N) \neq 1/(1 + 22/\log_2 N)$ but it unfortunately exhibits a similar behaviour. Consequently:

- Carelessly neglecting the $o(1)$ term can lead to dramatic errors in the assessment of a complexity in the class given by Eq. (1.1), and of its growth as N varies.
- Knowing more terms in the asymptotic expansion of ξ does not yield more precision in practical estimations based on a truncated formula for NFS complexity.

In fact, the converging behaviour of ξ should come as no surprise as the expansion of ξ relies on the expansion of ρ , and the latter involves a series that converges only for sufficiently large values as stated in Proposition 3.19. We recall that $\rho(u) \sim \exp\left(-u \log u \left(\mathbf{Q}^{(i)}(\mathcal{X}(u), \mathcal{Y}(u)) + o(\mathcal{Y}(u)^i)\right)\right)$

where \mathbf{Q} is defined in Proposition 3.22. To experimentally assess the convergence properties of ρ estimations, Figure 3.4 plots the functions $u \mapsto \mathbf{Q}^{(i)}(\mathcal{X}(u), \mathcal{Y}(u))$ in function of u for $1 \leq i \leq 6$.

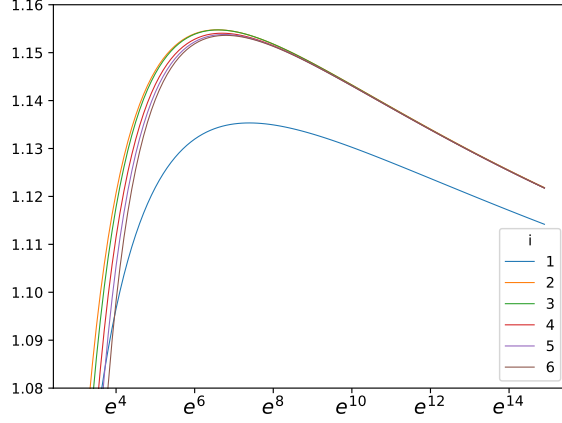


Figure 3.4: Plot of the functions $u \mapsto \mathbf{Q}^{(i)}(\mathcal{X}(u), \mathcal{Y}(u))$ for $1 \leq i \leq 6$, see Corollary 3.23.

Experimentally, we observe in Figure 3.4 that the asymptotic series expansion of ρ starts to converge around $u \approx e^7$. When assessing the complexity of NFS, we evaluate ρ for values of u that have the same order of magnitude than $(\log N)^{1/3}$. This is consistent with the observed convergence of the series expansion of ξ for $N > \exp(\exp(22))$ in Figure 3.3 since $\exp(22)^{1/3} \approx e^7$.

In summary, all the asymptotic estimations for ξ that we have at our disposal, including the brutal approximation $\xi = 0$, say little to nothing about the behavior of the complexity of NFS in the range where the algorithm can be used. In this range, it makes as much sense to say that the complexity of NFS behaves as $L_N(1/3, \sqrt[3]{64/9})$ than to say that it behaves as N^4 . We do know about the behaviour of ξ at infinity but in practical ranges this function remains a mystery. The consequence of these observations is that, unfortunately, replacing ξ by a truncation of its asymptotic expansion, even up to a high degree, in an attempt to have a better understanding of the NFS complexity may be irrelevant for practical uses. Indeed, it comes down to replacing a series by its first terms in a range where the series diverges. It means caution is definitely required when using Formula (1.1) or other truncated asymptotic expansions of the heuristic complexity of NFS in order to extrapolate key sizes for cryptography. This stresses the importance of simulation tools that rely on precise numerical evaluations of ρ , or possibly on actual smoothness tests, to estimate the complexity of NFS.

Chapter 4

Simulation of the filtering step in NFS

A crucial part in a computation using an implementation of NFS is determining a good set of parameters. This is especially true for record computations. This is not a trivial problem as dozens of parameters need to be adjusted and some interact with others. Moreover, thoroughly testing only one set of parameters would require a computing time of a same order of magnitude than the time required for the actual computation which is totally impracticable. Thus, in order to find suitable parameters we turn to simulation.

This section presents and studies the ESTIMATEMATSIZE algorithm from an experimental point of view. It partly predicts the behaviour of an NFS experiment given a set of parameters without actually running it. This algorithm is still being modified today for two main reasons: improve its predictive power and reliability and pinpoint which steps cause the gaps between predictions and reality in the hope of improving them.

The end goal of this study is to build a reliable and fast prediction tool. In this work, we experimentally investigated the behaviour of the ESTIMATEMATSIZE algorithm, we developed tools to do so more efficiently and we proposed modifications to the original code in order to mitigate brutal approximations that turned out to be excessive. We also opened several lines of questioning and suggested other experiments that could lead to a better understanding of the algorithm and in the end to better predictions. We emphasize that this chapter was fueled by discussions with Patrice Boudot, Pierrick Gaudry, Pierre-Jean Spaenlehauer, Emmanuel Thomé and Paul Zimmermann and rests on some of their preliminary works.

4.1 Overview of the ESTIMATEMATSIZE algorithm

In the following, we refer to Section 1.2.2 for all the vocabulary related to the execution of NFS. The ESTIMATEMATSIZE algorithm was first notably used during the record computation of RSA-240 and DLP-240 presented in [BGG⁺20a]. Roughly, it predicts useful information about an NFS run in a short amount of time. More precisely, its input is a set of NFS parameters and two polynomials from the polynomial selection step close enough to the optimal polynomials that will be used in a real computation. From this, it allows to estimate the size of the matrix after the filtering step and to some extent the cost of the linear algebra step. As a result, it can be used to compare sets of parameters and consequently tune the set of NFS parameters that will be used in a complete run.

Given a set of parameters for the relation collection step, the main idea of this algorithm is to sieve a few special- q in order to get a few sample relations. From these samples, so called fake relations are built. The number of fake relations generated is similar to the number of relations

that would have been produced in a real run with the same parameters. Building a fake relation is much faster than computing a real one so this step takes a lot less time than the time actually required to run the relation collection step. The filtering step is then applied to the matrix of fake relations, as if they were real. The hope is that fake relations faithfully mimic real ones so that the size and properties of the fake matrix after the filtering step are close to the ones the real matrix would have. As the complexity of the linear algebra used in NFS is closely linked to the size and density of the input matrix, this also allows to get a valuable insight on the behaviour of the linear algebra step.

In the following sections, we describe how fake relations are created in Section 4.1.1, how to further reduce the time required to run a simulation with the use of shrink factors in Section 4.1.2 and finally sum up the whole ESTIMATEMATSIZE algorithm in Section 4.1.3.

4.1.1 Generation of fake relations

Building example relations

In this chapter, we will intensively use the words “chunk” and “sample” which are rather blurry terms. We precise what they mean here in Definition 4.1.

Definition 4.1. *In the context of the simulation of the filtering step of NFS, a chunk is a range of special- q and a sample in a chunk is a special- q in this chunk.*

The first step to generate fake relations is to build a few real relations on which the fake ones will be based. This is done by sieving a few special- q . More precisely, the whole special- q range is divided in n_c chunks of the same size. In each chunk, n_s sample special- q are uniformly randomly selected. In total we sieve $n_c \times n_s$ special- q . In the example depicted below, the range of special- q is divided in nine chunks and two special- q per chunk are selected for a total number of model special- q of eighteen. By default in the original implementation, $n_c = 3$ and $n_s = 50$.

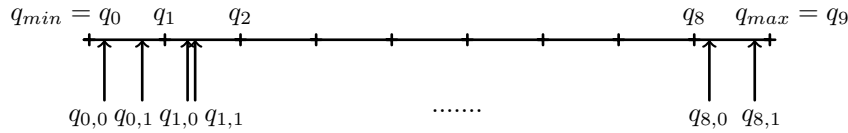


Figure 4.1: Selection of example special- q with 9 chunks and 2 samples per chunk.

Once we have selected a set S of special- q , we build the associated relations through sieving: Each special- q yields a set S_q of relations. The elements in the set of real relations $S_{rels} = \bigcup_{q \in S} S_q$ will be called *example relations* and all fake relations will be based on them. We emphasize that generating example relations is the most costly step in ESTIMATEMATSIZE by a large margin.

Building fake relations

We now explain how the example relations are used to build fake relations that mimic real ones. For the sake of simplicity, we consider a prime special- q although the following operations can be adapted for composite special- q . For each special- q in the q -range we build a corresponding set of fake relations \tilde{S}_q as follows:

1. Find a special- q $q_{ex} \in S$ in the same chunk as q .
2. Build as many fake relations for q as the number of relations in $S_{q_{ex}}$.
3. To build one relation for q , pick a relation associated to q_{ex} at random in $S_{q_{ex}}$. For all ideal I involved in this relation, if $I = q_{ex}$ remove it, else replace I by a random ideal I' on the same side (side-0 or side-1) with a norm close to the norm of I . In practice, the size of I' is

within a margin of the size of I of $\pm 20\%$. Add the ideal q to the relation to mimic the fact that q is a special- q , hence intervenes in the relation.

Remark 4.2. *The choice of the 20% margin is at the moment arbitrary. Some experiments should be conducted to assess the impact of the distortion of real relations to generate the fake ones.*

Once this has been done for all special- q in the q -range, all the unique (all duplicates are removed on the fly) fake relations are stored in a matrix that will be called the fake matrix of relations or even the fake matrix. The example relations are not added to this fake matrix. Then the filtering step can be run on the fake matrix as if it were a matrix that actually was the output of a relation collection step of NFS.

Of course, producing fake relations is extremely fast compared to generating real ones through sieving. But fake relations share properties with real ones: They have the same average number of relations per special- q and the same average number of ideal per relation for instance. Because of this proximity between fake relations and real ones, we hope that the filtering step behaves similarly on a fake matrix and on a real one generated with the same parameters in the relation collection step.

Relevance of the method to select the example special- q

One could wonder why we choose to divide the range of special- q in chunks when selecting example special- q instead of directly picking $n_c \times n_s$ special- q at random in the whole q -range $[q_{min}, q_{max}]$.

One reason is that in the current implementation of ESTIMATEMATSIZE, the fake relations associated to q are based on the real relations associated to a q' in the same chunk so tuning the number of chunks allows to keep the size of q close to the size of q' . This is of importance as large special- q and small special- q do not necessarily yield the same number of relations and their properties can slightly differ so we would like to avoid building fake relations for a large q by using real relations associated to a small q' . But this issue can be avoided for instance by basing the fake relations for q on the real relations obtained for q' such that the size of q' is the closest to the size of q among all the example special- q .

Another reason is that we hope that it will provide better example relations by ensuring some diversity in the selected special- q . Indeed, we can guarantee that for all $i \in [0, k - 1]$, n_s special- q will be chosen in the range $[q_i, q_{i+1}]$ while this does not hold when the choice of special- q is random in the whole range.

However, an interesting but as of yet not answered question about the selection of those special- q is: Can this selection be improved? This is of special interest since the selected special- q will generate relations on which all the fake relations will be based so similarity between the fake matrix and what would have been the real one is tightly connected to the example relations, hence to the choice of special- q selected to begin with. For instance the following selection can have undesired properties. Say we select 5 special- q in 10 chunks. Because of the distribution of primes, there are more primes in the first chunk than in the last, however the same number of special- q is selected in each. This can lead to an over-representation of relations corresponding to large special- q and an under-representation of relations corresponding to small special- q in the set of example relations and this imbalance could have a significant impact on the next steps of the algorithm.

One way to avoid this biased selection would be to choose our model special- q uniformly at random among the special- q in the q -range. This would require a way to efficiently choose primes at random in a given range. This can be achieved using [FT14] for instance. Overall guaranteeing an unbiased special- q selection is not a trivial task. In fact, since special- q can even be composite in practice [BGG⁺20b], the question of the selection of representative special- q is even more complicated.

4.1.2 Use of shrink factors

The fake matrix on which the filtering step is applied is expected to have the same size than a matrix resulting from a real computation with the same sieving parameters. If one wants to run several simulations, this size can become an issue, in particular memory-wise. In order to mitigate this, the ESTIMATEMATSIZE algorithm has a *shrink factor* parameter that allows to build and filter a fake matrix of a smaller size.

When using a shrink factor σ , the number of rows and of columns of the fake matrix is divided by σ . More precisely:

1. The generation of example relations is not impacted.
2. The generation of fake relations for a given special- q is very similar. The only difference is that the number of relations produced for a given special- q is the number of relations described in Section 4.1.1 divided by σ . If this number n_{rels} is smaller than one, a relation will be produced with probability n_{rels} . This reduces the number of fake relations, hence of rows, by a factor σ .
3. The columns are shrunk with respect to the side, side-0 or side-1. For a given side, ideals are sorted by size. Then, each group of σ consecutive columns are merged into one with this rule: There is a non-zero coefficient in the i -th row of the merged column if there was at least one non-zero coefficient in the i -th row in one of the σ columns before the merge. This reduces the number of columns by a factor σ .

Point 3 calls for a few remarks:

Remark 4.3. • *First, let us notice that the "merge" operation done on columns when a shrink factor is applied has nothing to do with the "merge" operation of the filtering step.*

- *In the NFS case, a nonzero coefficient is always equal to one since the base field of the matrix of relations is $\mathbb{Z}/2\mathbb{Z}$. In the NFS-DL case however, the base field of the matrix of relations is $\mathbb{Z}/\ell\mathbb{Z}$ which begs the question: If there is a nonzero coefficient in one of the merged columns, what element of $\mathbb{Z}/\ell\mathbb{Z}$ should be placed in the shrunk column? The answer is: 1. This is motivated by two facts. First, even in the NFS-DL case, most coefficients in the matrix of relations equal 1. Second, the main concern in the filtering step is whether coefficients are zero or not, the actual value of the coefficient does not matter that much.*
- *In practice, ideals are numbered according to their size. A relation is then a list of numbers instead of the list of ideals in the associated factorization. In this context, shrinking columns by a factor σ simply comes down to dividing all the numbers in each relation by σ and rounding.*
- *Because a matrix of relations is really sparse, it is reasonable to think that in each group of σ consecutive columns being merged into one, there is a really low chance that two of these columns have a nonzero coefficient on a common line. This property is central to assert that the shrinking step causes minimal distortion in the properties of the original matrix. But sparsity in a matrix of relations is not evenly distributed. While columns indexed by large ideals are indeed very sparse, the same cannot be said for columns indexed by very small ideals as small factors are a lot more likely to appear in the factorization of norms during the relation collection step, leading to overall denser columns. A first approximation is to ignore the potential impact of these dense columns on the properties of the shrunk matrix, as they are few. However, we will explore in Section 4.4.2 a few ideas designed to take them into account, and to compensate this bias.*

The resulting matrix will be called a fake shrunk matrix and there is a factor σ between its size and the size of the original fake matrix. In practice, the shrink factor usually ranges from 1 to 1000

so the fake shrunk matrix can indeed be much smaller than the original one. At the same time, a fake shrunk matrix preserves a number of properties of the fake un-shrunk matrix. In particular, the average weight and the distributions of the weights of columns and rows is preserved. As we have seen in Section 1.3, those distributions (especially for columns) rule most of the filtering step, which fuels the hope that filtering a fake shrunk matrix will give usable insight on how a real matrix of relations would have behaved.

As in the previous section, the filtering step is applied to a fake shrunk matrix as if it were genuine. The output matrix is expected to be σ times smaller than the un-shrunk matrix would have been. Usually the shrink factor σ is an integer but nothing actually prevents it from being a float as long as it is greater than one.

4.1.3 Summary of the ESTIMATEMATSIZE algorithm

We present in Figure 4.2 a schematic overview of the algorithm, describing both its steps and the main parameters involved in each step.

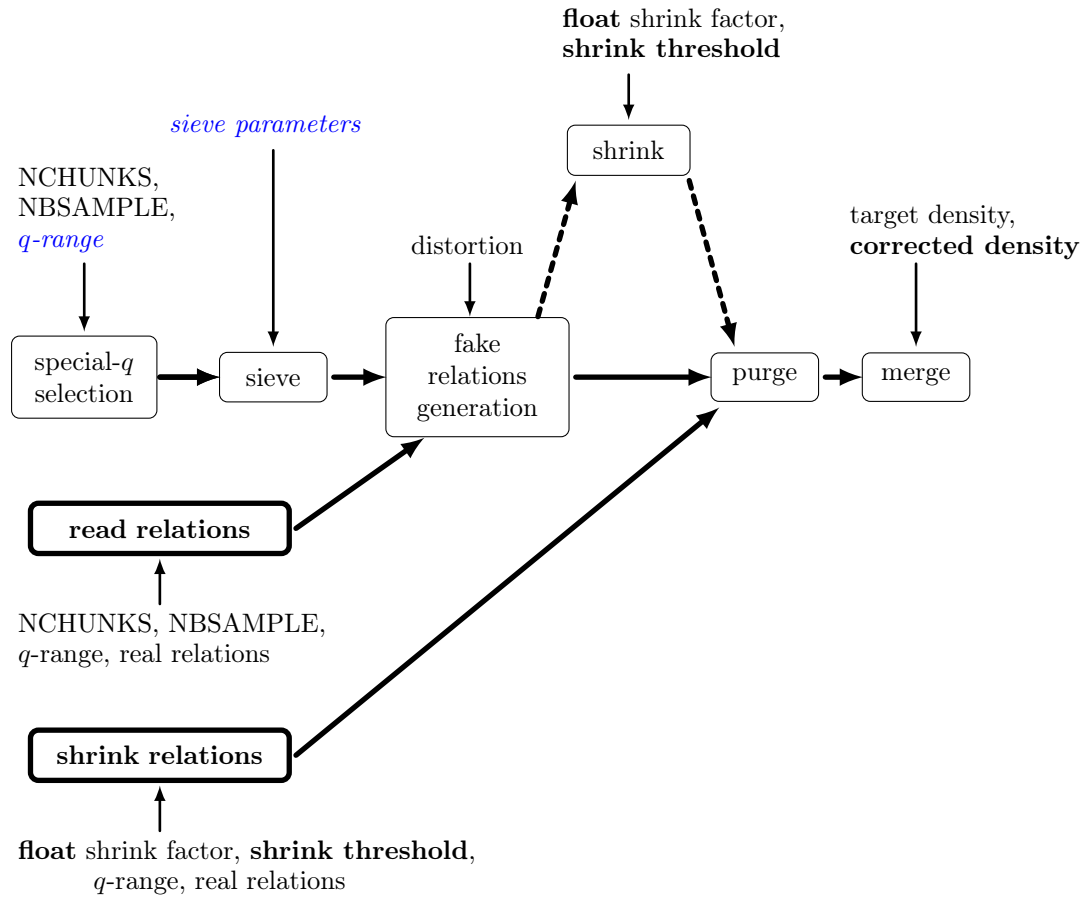


Figure 4.2: Overview of the ESTIMATEMATSIZE algorithm. Bold elements correspond to algorithmic amendments made during this thesis.

The first row in Figure 4.2 depicts the execution of ESTIMATEMATSIZE when it is used to make predictions. In italics are the main parameters that can be evaluated during the experiment, non italicized parameters are parameters specific to the simulation algorithm. The special-*q* range is divided in *NCHUNKS* chunks in each of which *NBSAMPLE* sample special-*q* are selected as explained in Section 4.1.1. The capitalized denominations for these quantities reflect the ones

used in practice in the implementation. The relations for the selected special- q are then built through sieving, providing example relations. This step is ruled by all the sieving parameters we want to assess, for instance the size and the shape of the sieving area, the limit for the size of the cofactors after the sieving step, the so called large prime bound which sets a limit on the factors' sizes in the cofactorization, all of them being tuned on side-0 as well as side-1. We refer for instance to [BGG⁺20a] for more details on which parameters are of interest in practice. The set of example relations thus computed are used to create fake relations by distorting them as explained in Section 4.1.1. Depending on the experiment, the shrinking step can be skipped altogether, in which case the filtering step is immediately applied to the matrix of fake relations as if it were genuine. Let us notice that the separation we have made in the Figure 4.2 between the generation of fake relations and the shrinking step can be misleading: In practice, when a shrink factor is applied both steps are performed at the same time as explained in Section 4.1.2. We chose to separate the two nonetheless in order to simplify the exposition.

Bold elements in the Figure 4.2 correspond to parameters and algorithms that were added to the original ESTIMATEMATSIZE algorithm during this thesis. In particular, the last two rows of this sketch depict the algorithms developed for retrospective analyses of ESTIMATEMATSIZE. They were designed to be used in the following context: We assume that a real computation already took place and that the corresponding relations computed during the sieving step were stored. The algorithm READRELATIONS allows to directly find and pick the relations corresponding to the selected special- q instead of re-computing them, thus skipping the time-consuming sieving step. The SHRINKRELATIONS algorithm is used when a shrink factor is applied and allows both to speed up computations and uncouple two phenomenons in the ESTIMATEMATSIZE algorithm: the impact of the fake relations generation on the predictions and the impact of the shrinking step on these predictions. Indeed it applies the shrinking operations presented in Section 4.1.2 directly on the real stored relations instead of fake ones which allows to study specifically the behaviour of the shrinking step as we will see in Section 4.4. Thus, we hope to independantly measure the impact of each step of ESTIMATEMATSIZE on the quality of predictions.

More details on the added parameters, their uses and the reasons for their introduction will be given in Section 4.4 but we quickly present them before moving on :

- The shrink factor was originally confined to integer values but it can now be a floating-point number. This has little interest for predictions but it is a tool to retrospectively study the algorithm.
- The shrink threshold parameter allows a less brutal shrinking step by excluding the (heavy) columns below this threshold from being shrunk.
- The corrected density parameter allows to use a more relevant quantity than the density of the matrix to control the merge step when a shrink factor is applied.

In parallel to the development of these new features, a few comfort options were added to the implementation of ESTIMATEMATSIZE in CADO-NFS. Most notably, a number of files such as the files containing the example relations are no longer re-computed if they already belong to the working directory specified for the experiment which greatly speeds up repeated simulations as building example relations requires costly sieving. Moreover, the only argument of the script at the beginning of this thesis was the file containing the polynomials from the polynomial selection step: The sieving, filtering and simulation parameters had to be individually given through environment variables. All these parameters can now be wrapped up in a single parameter file which makes running experiment easier and tidier.

4.2 Base results for ESTIMATEMATSIZE experiments

The main goal of this chapter is to report on experiments conducted with ESTIMATEMATSIZE that aim at describing the behaviour of this algorithm. This is a first step towards assessing

the reliability of this algorithm to predict useful information on a factorization or DL run, and perhaps improving it. During this exploration, a number of tools were designed to help studying the behaviour of the ESTIMATEMATSIZE algorithm. This algorithm is still being modified both to improve the quality of its predictions and to retrieve exploitable information that can help explaining its behaviour. It has been noticed in [BGG⁺20a] that the quality of the predictions of ESTIMATEMATSIZE were somewhat lower in the case of NFS-DL than in the case of NFS. As a result, all experiments have been conducted on discrete logarithm computations, since this is where there seems to be the largest room for improvement.

Before conducting our ESTIMATEMATSIZE experiments, real discrete logarithm computations were performed in prime fields of size 512 bits on the one hand and 585 bits on the other hand using the NFS implementation CADO-NFS available at <https://gitlab.inria.fr/cado-nfs/cado-nfs>. We will refer to the complete computation in a field of size 512 bits (resp. 585 bits) by “the DLP-512 experiment” or even by the DLP-512 (resp. the DLP-585). The parameters used for the DLP-512 experiment were the suggested parameters for a 155-digits discrete logarithm computation (which corresponds to 514 bits) by the CADO-NFS software. The parameters used in the DLP-585 experiment were designed for the occasion as there are no suggested parameters for this size in CADO-NFS. They result from a crude interpolation between the suggested parameters for a 155-digits computation and the parameters used for a 180-digits experiment. Both sets of parameters, together with the actual values for the size p of the number field and the size ℓ of a subgroup of $\mathbb{Z}/p\mathbb{Z}$ in which we want to compute a discrete logarithm in each case are available at https://gitlab.inria.fr/alegluhe/simulation_nfs.

These two discrete logarithm computations were run using the grvngt and grcinq clusters of Grid’5000. In both cases, the computation was interrupted just after the filtering step and the data of the filtering step stored in order to be able to compare them with the estimations provided by the experiments with ESTIMATEMATSIZE. The target density — that is, the quotient of the matrix weight by its size — for the merge step of the filtering step was $d = 100$ in both cases, the same value will always be used during the estimation experiments.

We provide in Table 4.3 the raw matrix size, the size after the purge step and the estimated size after the merge step — that is, after the whole filtering step — for a density equal to 100 in both cases. They will serve as reference points to assess the precision of the results obtained through simulations with ESTIMATEMATSIZE.

	DLP-512	DLP-585
raw relations	26 553 389	30 848 508
relations after purge	5 070 652	15 926 672
relations after merge	2 010 802	6 628 634

Figure 4.3: Number of rows in the matrix at various stages of the filtering step

The expression “estimated size after the merge step” deserves a few comments. Remember from Section 1.3.2 that the merge step is performed in passes. During each pass, several merges are applied, all of which having a similar weight. At the end of a pass, the density of the resulting matrix is compared with the target density d . If it is below, a new pass begins with heavier merges. If it is above, the merge step stops. Therefore, after the last pass, the matrix of relations has a raw size and a density d_+ such that $d_+ > d$. Similarly, the raw matrix sizes after the whole filtering step that we get from ESTIMATEMATSIZE experiments correspond to densities above d but not equal to d . This makes comparisons between all these matrix sizes difficult as those sizes do not correspond to the same density.

Fortunately, it is possible to estimate what the size of the filtered matrix would have been if the merge step had stopped exactly when the density of the matrix reached d . Besides, this value is already computed by the merge script in CADO-NFS. This value is found through linear

interpolation between the two last passes. More precisely, let us denote by d_+ the density of the matrix after the last pass of merges and by d_- the density of the matrix after the second-last pass of merges. Notice that $d_- < d \leq d_+$. We assume that the density of a matrix being merged is an affine function of its size during a given pass. This is motivated by the fact that the merges' weights during a pass are similar, say roughly equal to w , so that when a merge in a pass is applied, the size of the matrix decreases by one and its weight increases by w . In particular, the density of the matrix is an affine function of its size between d_- and d_+ and the coefficients of this function can be computed using the two pairs (density, size) resulting from the last and second-last passes. Once this function is known, all that remains is to evaluate it in d to get the so-called estimated size. Comparing estimated sizes makes a lot more sense than comparing raw sizes since it allows to compare the sizes of matrices that have the same density.

In the following, when we mention matrix sizes after the merge step (or after the whole filtering step) we will always refer to the estimated size for a density equal to 100.

Remark 4.4.

We will often refer to values in Table 4.3 by calling them “the real values for a DLP-585” (resp. for a DLP-512) or even by “the real values” without mentioning the experiment when the context is clear. This is slightly misleading as the results displayed in Table 4.3 were obtained using specific sets of parameters. Were these parameters chosen differently, we would have obtained different values for the above matrix sizes that would have been as “real” as the ones we present here. So when we talk about “the real value” we mean “the value that is output with the set of parameters mentioned in this section”.

4.3 Impact of the number of example special- q

The main difficulty in the analysis of the reliability of an ESTIMATEMATSIZE experiment rests on the fact that many parameters are involved. They fall roughly into two categories:

- How are the example relations selected? This encompasses the number of example relations which depends on the parameters NCHUNKS and NBSAMPLE and all the parameters used to sieve.
- Is the matrix of relations shrunk or not?

In order to assess the reliability and power of prediction of ESTIMATEMATSIZE, we uncoupled those two factors as best as possible. In this section, the focus is on the impact of the number of example relations on predictions. Thus, all experiments were made with a shrink factor of one, that is, no shrinking step was applied. Furthermore, the sieve parameters were the same than the ones used in the real run. The only parameters that vary between experiments in this section is the number of chunks and the number of sampled special- q in each chunk.

Protocol and results

We let \mathcal{E}_c (resp \mathcal{E}_s) be two sets of integers. In practice we chose $\mathcal{E}_c = \{2 + 2k \mid k \in \mathbb{Z}_{\geq 0} \text{ and } 2 + 2k < 50\}$ and $\mathcal{E}_s = \{10 + 10k \mid k \in \mathbb{Z}_{\geq 0} \text{ and } 10 + 10k < 200\}$ for the experiments on the DLP-585. We pushed to $\mathcal{E}_c = \{2 + 2k \mid k \in \mathbb{Z}_{\geq 0} \text{ and } 2 + 2k < 100\}$ and $\mathcal{E}_s = \{10 + 10k \mid k \in \mathbb{Z} \text{ and } 10 + 10k < 400\}$ for the DLP-512 experiments.

For each pair $(n_c, n_s) \in \mathcal{E}_c \times \mathcal{E}_s$, we divide the special- q range in n_c chunks of the same size and randomly select n_s special- q in each chunk. Normally we would then have to build the relations for all those special- q from scratch, which would require some time-consuming sieving. But, instead of sieving the selected special- q , we take advantage of the fact that all the relations for all the special- q in the q -range have already been computed during the real run. These relations were stored and can be easily accessed and retrieved in order to produce the example relations without sieving. During this thesis, we designed an algorithm READRELATIONS that precisely performs

these operations once the relations output during a real run have been stored in a convenient data structure. In practice, the algorithm ESTIMATEMATSIZE actually encompasses algorithm READRELATIONS: One simply has to specify in the options of ESTIMATEMATSIZE how to generate example relations (sieving or reading) and add the path to the aforementioned data if one wants to build example relations by reading precomputed relations.

Of course, algorithm READRELATIONS is only useful to study the ESTIMATEMATSIZE algorithm but not to do actual estimations. Indeed, in practice, we do not have access to all the real relations for all special- q : If we had, we would not need to produce fake relations in the first place. However it remains a powerful tool for retrospective analysis as it greatly speeds up the generation of example relations in an ESTIMATEMATSIZE experiment by cutting sieving computing times. No precise estimation of the magnitude of this speed-up has been made, but it is at least beyond 20.

Anyway, once the example relations are built, we proceed as described in Section 4.1.1: We build a matrix of fake relations that goes through the filtering step as if it were genuine. In the end, for each pair $(n_c, n_s) \in \mathcal{E}_c \times \mathcal{E}_s$, we retrieve the matrix size after the filtering step and compare it with the corresponding real value. Figure 4.4 displays the error percentage between the real matrix size and the estimated matrix size after the filtering step in function of the total number of example special- q $n_c \times n_s$ for all $(n_c, n_s) \in \mathcal{E}_c \times \mathcal{E}_s$. The left plot correspond to the DLP-585 experiment and the right plot to the DLP-512 experiment.

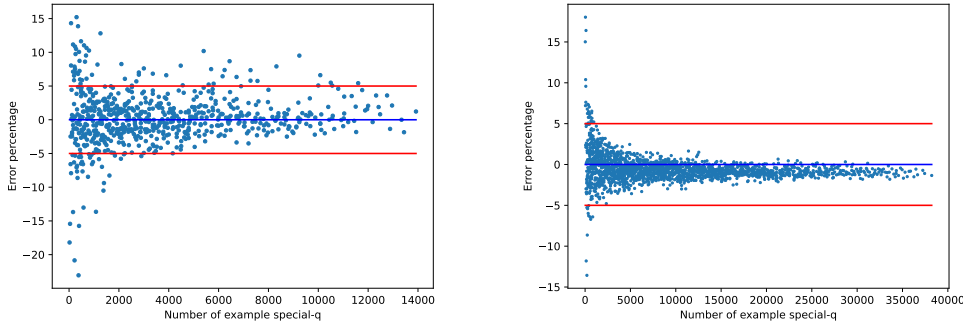


Figure 4.4: Error percentage between the real matrix size after the filtering step and the estimated matrix size after the filtering step when its input is a fake version of the real matrix of relations (for a DLP-585 on the left and a DLP-512 on the right).

Analyses of Figure 4.4

A quick glance at Figure 4.4 tells us that:

1. Estimated matrix sizes after the filtering step are on average quite close to the real one.
2. Estimated matrix sizes tend to stabilize when the total number of example special- q increases.
3. The variability of the predictions is linked to the number of example special- q .
4. Estimations in the DLP-585 case are more chaotic.

We now detail the above points.

1. A first satisfying observation is that on hundreds of experiments on two different DLPs, the value output by ESTIMATEMATSIZE is always within a 20% margin of the real value, except exceptionally for a few small values for the number of example special- q . In fact, in the DLP-512 case, errors between real and estimated values even turn below 5% rather quickly. This is not the case for the DLP-585 but notice that the matrix of relations in this context is larger than the one for a DLP-512. It means that a given number of example special- q have a greater chance to yield

typical relations in the DLP-512 case (and so, lead to a representative fake matrix and accurate estimated sizes) than in the DLP-585 case. This can explain why we need more example relations for the estimated sizes in a DLP-585 context to reliably fall within a 5% margin of the real size.

2. The matrix sizes output by ESTIMATEMATSIZE in these experiments appear to stabilize around some “asymptotic” value when the number of example special- q increases. It is particularly blatant in the DLP-512 case. What is unsettling is that in this case this asymptotic value seems slightly below the real one, while we would have expected it to be exactly equal to it. This stabilization behaviour is less obvious in the DLP-585 case. Further tests in which we could use the (larger) sets \mathcal{E}_c and \mathcal{E}_s used in the case of DLP-512 for instance can be made to both confirm this stabilization trend and check if the asymptotic value also underestimate the real one in this case.

3. The fewer the number of example special- q , the more diverse the predictions of ESTIMATEMATSIZE are. This is not very surprising: More example special- q means more and more diverse example relations on which to base the fake ones. When the number of example special- q is low, there is more chance that the example relations do not capture the whole architecture of the real matrix of relations well enough, thus leading to less accurate estimations. In particular, for a number of example special- q below 1000 in the DLP-512 experiment and below 1500 in the DLP-585 experiment, estimations can be really accurate but sometimes quite far off. When using ESTIMATEMATSIZE in the context of a DLP computation in a field of size larger than 500 bits, we would thus advise either to

- not use the default values for the parameters NCHUNKS and NBSAMPLE in the ESTIMATEMATSIZE implementation, namely 3 and 50, as they lead to a total number of example special- q of 150 which is too small to consistently avoid extreme estimations.
- or to average out several estimations done with the same small number of example special- q . This actually works quite well: Using the data retrieved in the previous experiment and averaging out the results of ten ESTIMATEMATSIZE experiments for each total number of example special- q ranging from 50 to 1500, we get an average error that stays below 5% in both cases.

4. In order to compare the dispersion of both datasets we compute the standard deviation of the estimated matrix sizes for which the number of example special- q they are based on falls between 0 and 1000, do the same when this number falls between 1000 and 2000, and so on up to 13000 and 14000. For comparable numbers of example special- q , the standard deviation of the sample in the DLP-585 experiment is always one order of magnitude larger than the one in the DLP-512 experiment (around 10^5 for the former and 10^4 for the latter). This can again be explained by the fact that the matrix of relations for a DLP-585 is larger than the one for a DLP-512, meaning that a same number of example special- q is more likely to accurately represent real relations in the second case than in the first.

Finally, we would like to warn about a potential bias introduced by the fact that example special- q are selected from chunks. We described the results of the previous set of experiments as if the only parameter that changes is the total number of examples, while in fact it is the number of chunks and the number of samples that vary. Said otherwise, for a total number of example special- q of 200, one experiment was led by selecting two special- q in each one of a hundred chunks, another by selecting five special- q in each one of 40 chunks, another by selecting 100 special- q in each one of two chunks... But there is no guarantee that this repartition of the total number of examples into chunks and samples has no influence on estimations. It could be that a large number of chunks leads to better estimations than a small one for the same total number of examples for instance.

To find out, we can conduct the following experiment: For a given number n_{ex} of example special- q , find all possible factorizations of $n_{ex} = n_c \times n_s$ into two integers n_c and n_s and run an ESTIMATEMATSIZE experiment for each pair (n_c, n_s) . Repeat for several n_{ex} . If a pattern in

the estimated matrix sizes appears, for instance if the estimated sizes increase with the number of chunks for all the tested n_{ex} , it would point to the fact that the repartition into chunks and samples influences estimations. Actually, we have led such experiments for $n_{ex} \in \{50, 250, 450\}$. We did not uncover any obvious trend but the number of n_{ex} tested is perhaps too small to conclude.

4.4 Impact of the shrinking step

The following sections report on experiments done with a non trivial shrinking step, which means that a shrink factor not equal to one was applied to the matrix of relations before it entered the filtering step. Originally, the ESTIMATEMATSIZE algorithm only allowed the shrinking step to be applied to a fake matrix of relations. But the sampling step and the generation of fake relations could well interfere with the impact of the shrinking step. In order to uncouple those two steps — namely building a fake matrix and shrinking a matrix — and try to measure only the impact of the shrinking step on predictions, the experiments with a shrink factor have been made on real relations. We detail below what we mean by this and how it was achieved.

Recall that all the relations in our DLP-585 and DLP-512 experiments have been precomputed during the real runs. The shrinking step can thus be applied to those real relations using the SHRINKRELATIONS algorithm instead of fake relations: If the shrink factor is σ , we keep one relation for every σ relations and we divide the index of the ideals involved in each relation by σ rounding up (or down) in order to shrink the columns of the (real) matrix of relations. A seed parameter rules which relations are kept: The seed is linked to a relation number n in the real matrix and the relations kept are those numbered by an element in $\{(n + k\sigma) \bmod N \mid k \in \mathbb{Z}\}$ where N is the size of the input matrix.

The SHRINKRELATIONS algorithm was designed during this thesis. It serves two main purposes:

- Evaluate the impact of the shrinking step regardless of what happens during the sampling and creation of fake relations steps.
- Significantly speed up the running times of experiments when a shrink factor is used. It is indeed much faster to shrink precomputed relations than to build example relations on which we base fake ones as it requires some costly sieving.

Of course, this algorithm is a tool only designed to assist the evaluation of the ESTIMATEMATSIZE algorithm. In practice, real relations are not yet computed when the need for estimations arises. Similarly to the READRELATIONS algorithm, it is mostly useful for retrospective analyses.

In all the following experiments, the shrinking step is directly applied to the real matrix of relations through the SHRINKRELATIONS algorithm. If M denotes this real matrix of relations, we will denote by $M(\sigma)$ the σ -shrunk matrix output by SHRINKRELATIONS when its input is M .

4.4.1 Estimations based on real shrunk relations

This section describes the behaviour of the shrinking step of the ESTIMATEMATSIZE algorithm.

Experimental protocol and results

The first experiment we made was the following. For each shrink factor σ in a given range, we shrink the real matrix of relations of the DLP-585 experiment using SHRINKRELATIONS and we plug the resulting matrix into the filtering step, which yields a final matrix of size N_σ . To estimate what would have been the size of the matrix after the filtering step if it had not been shrunk, we compute the value $\sigma \times N_\sigma$. We hope this value is close to the real matrix size after the merge step for the DLP-585 experiment provided by Table 4.3.

The choice of the relations that are kept during the shrinking step can maybe influence the final estimation. In order to mitigate this, for a given shrink factor σ the same experiment is

therefore launched with five different seed parameters in SHRINKRELATIONS so that a different set of real relations is kept each time.

Finally, we plot the error percentage between the real value given by Table 4.3 and the estimated value $\sigma \times N_\sigma$ for each shrink factor σ to get Figure 4.5. To each abscissa correspond five points, each for one of the five seeds used in SHRINKRELATIONS. The red curve links the points $(\sigma, \text{average error for a shrink factor } \sigma)$. Thus this plot reads for instance: When a shrink factor of 75 is applied on the real matrix of relations, we obtain an estimated matrix size after the filtering step that slightly underestimate the real value and is on average within a 4% margin of the real value.

The same experiment was done in the case of the DLP-512 experiment to see if the general behaviour of the shrinking step somehow depended on the matrix of real relations on which everything is based. This yields Figure 4.6

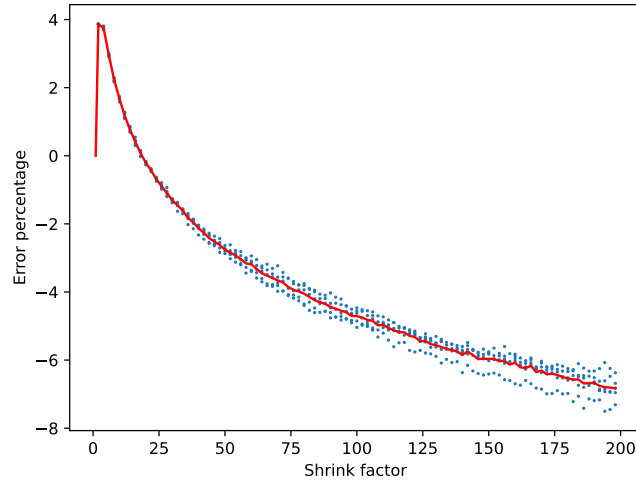


Figure 4.5: Error percentage between the real matrix size after the filtering step and the estimated matrix size after the filtering step when its input is a σ -shrunk version of the real matrix of relations for a DLP-585.

Analyses of Figures 4.5 and 4.6

First let us summarize three observations allowed by these first experiments:

1. Estimated matrix sizes stay close to the real matrix size, even when the shrink factor is large.
2. Both scatter plots look the same.
3. Something weird is going on with small shrink factors.

We now detail each one of the above points.

1. The first analysis of these figures is indeed quite promising: The estimated size for a shrink factor that ranges between 2 and 200 stays within 8% of the real value in both cases. In fact, when we push the shrink factor up to 1000 the error between real and estimated values remains below 13% in both cases. This is rather consistent with previous experiments done in [BGG⁺20b] and further supports the idea that the shrinking step of ESTIMATEMATSIZE still allows precise estimations even with a large shrink factor.

2. Although a given shrink factor does not yield exactly the same error in both cases, we also notice that the global shape of both figures is roughly the same. This makes us hope that

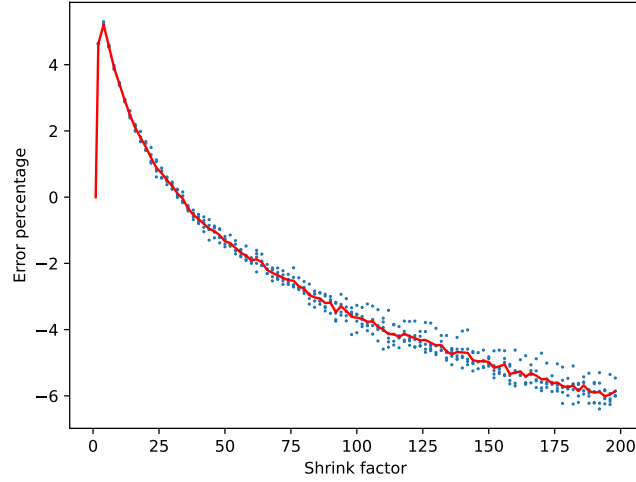


Figure 4.6: Error percentage between the real matrix size after the filtering step and the estimated matrix size after the filtering step when its input is a σ -shrunk version of the real matrix of relations for a DLP-512.

Figures 4.5 and 4.6 actually capture a general behaviour that we would get for any DLP. In particular:

- Disregarding what happens for shrink factors between 1 and 3, the average signed error is a smoothly decreasing function of the shrink factor.
- The impact of the seed – that is, the impact of which relations are kept by the shrinking step – is minimal. It does grow with the shrink factor, which is not surprising as a larger shrink factor means more deleted rows in the original matrix, hence a greater distortion of the input. But, the impact of the seed can be lessened by simply averaging the results of a few estimations, that can be independantly launched in parallel, in which the set of relations selected by the shrinking step varies.
- A quick rule of thumb can be derived from these figures: When using a “small” shrink factor, we tend to overestimate the real matrix size after the filtering step while using a “large” shrink factor, we tend to underestimate it. Notice that in [BGG⁺20b] the authors ran an ESTIMATEMATSIZE experiment with a rather large shrink factor of 100 on RSA-240 that yielded an estimated matrix size of 33 million rows, and which underestimated the size of the 36 million row matrix they obtained after the filtering step in the real run. This points to the fact that the general behaviour of the shrinking step we have uncovered in the context of DLP computations also applies in the context of factorization, although further experiments should be led to confirm this statement.

One will have noticed the quotes enclosing the words small and large in the previous point. What is a “small” shrink factor depends on the experiment: In the case of DLP-585, a small shrink factor is a shrink factor below 20 while in the case of DLP-512, a small shrink factor is a shrink factor below 35. Computing the abscissa of the intersection point between the abscissas axis and the function that links the error and the shrink factor would yield the threshold shrink factor. But to this day there is no method to compute it, estimate it or bound it, and it is not even clear which parameters influence this threshold value.

3. The impact of the shrinking step for small values of shrink factors is quite baffling. Intuitively, the larger the shrink factor, the greater the distortion of the matrix input in the filtering

step. This leads to think that the absolute value of the error between real and estimated values would increase with the shrink factor. But it is not the case: In the DLP-585 experiment, we get a more accurate estimated value with a shrink factor of 25 than with a shrink factor of 2. In fact in both experiments, small shrink factors strangely overestimate the real size of the filtered matrix.

What is even more disturbing is the discontinuity between $\sigma = 1$ (no shrink) and $\sigma = 2$. The option allowing the use of floating-point shrink factors in SHRINKRELATIONS and ESTIMATEMATSIZE was actually designed to investigate further this discontinuity. Indeed, notice that the operations involved in SHRINKRELATIONS can easily be adapted to non integer shrink factors σ : For instance if $\sigma = 1.2$, we can divide the matrix of relations into blocks of 12 relations, keep 10 relations out of each block of 12 relations and then divide the index of the ideals involved in the kept relation by 1.2 rounding up (or down). Unfortunately experiments with floating-point shrink factors between 1 and 2 did not help understanding this phenomenon: It only yielded errors increasing from 0 to the error observed for a shrink factor of 2.

Still, it would be interesting to understand what happens with small shrink factors. A first step is to pinpoint which steps are responsible for this strange behaviour. Is it there only because of the shrinking step? Is it linked to the structure of a matrix of relations? Is it caused by an operation in the purge or the merge step during the filtering step? There is also a possibility that this odd behaviour is caused by a bug in the code of ESTIMATEMATSIZE that was not picked up during this analysis despite our efforts.

We present here an experiment that can be led to clarify things. Remember that what matters the most in the purge and merge steps is the distribution of nonzero elements in columns. With this in mind, compute the density of each of the columns of the real matrix of relations. Build a random matrix of the same size, in $\mathbb{Z}/2\mathbb{Z}$ for instance, such that the density of its i -th column is the same than the density of the i -th column in the matrix of relations. Shrink and filter this random matrix as we would have the matrix of relations. The purge and merge steps should behave similarly on the real matrix and on the fake random one. So, if the same overestimation for small shrink factors appears, it would point to this overestimation being the result of the shrinking step in itself. If not, it would probably mean that a matrix of relations has hidden structures that are disrupted by the shrinking step and identifying them could have many interests. For instance, it could allow the creation of better fake relations in ESTIMATEMATSIZE.

Uncoupling the purge and the merge steps as best as possible

Recall that the filtering step is divided in two main substeps: the purge step and the merge step. In order to better understand the interactions between the shrinking and the filtering step, we can follow what happens to the shrunk matrices in the filtering step in more detail.

To study the interaction of the purge step with the shrinking step, we retrieved the matrix sizes after the purge step in all the previous experiments. Figure 4.7 plots for each shrink factor σ the rescaled size of the matrix after the purge step when the matrix input in the filtering step is shrunk with a factor σ , namely $\sigma \times N_{\text{purge},\sigma}$ where $N_{\text{purge},\sigma}$ is the raw matrix size after the purge step. To each abscissa corresponds five points, each for one of the five seeds used in SHRINKRELATIONS in the above experiment. The red curve links the points $(\sigma, \text{average rescaled size after the purge step for a shrink factor } \sigma)$. Notice that the difference between the real matrix size after the purge step and the shrunk matrix sizes after the purge step is not an “error” as the merge step that follows could fill the gap between the final real and estimated sizes.

To study the interaction of the merge step with the shrinking step, we followed the evolution of the matrices sizes during the different passes of the merge step in the case of the DLP-585 experiment in four cases:

- The input matrix is the real matrix of relations M .
- The input matrix is $M(6)$. According to Figure 4.5, the final matrix size in this case overestimates the real value.

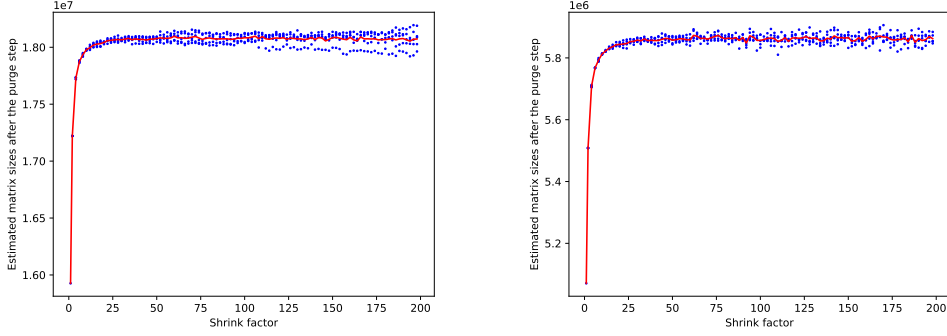


Figure 4.7: Matrix sizes (in number of rows) after the purge step in function of the shrink factor applied to the input matrix. On the left, the DLP-585 case. On the right, the DLP-512 case.

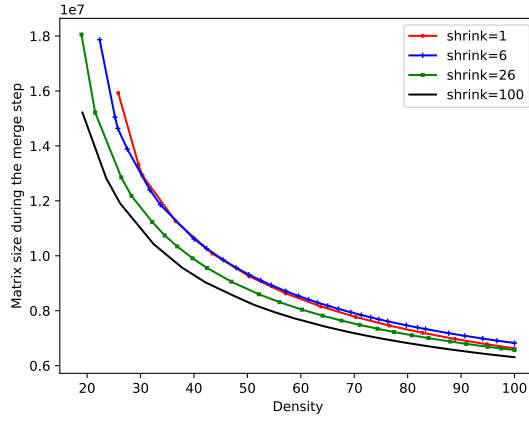


Figure 4.8: Evolution of shrunk matrix sizes during the merge step.

- The input matrix is $M(26)$. According to Figure 4.5, the estimated final matrix size in this case is almost spot on.
- The input matrix is $M(100)$. According to Figure 4.5, the final matrix size in this case underestimates the real value.

In each case, we retrieved from the logs of the merge step the pairs (matrix density, matrix size) after each pass and after rescaling the matrix sizes by multiplying them by the corresponding shrink factor, this yields Figure 4.8.

Analyses of Figures 4.7 and 4.8

Figure 4.8 does not display any peculiar behaviour. We could have conjectured that an underestimation of the matrix size when entering the merge step would translate into an underestimation of the final matrix size. But it is not the case as all experiments with a non trivial shrinking step underestimate the real value for a density equal to 30, yet one ends up underestimating the final value, one overestimates it and the third gets really close to the real value in the end. Overall, there is no obvious trend in the behaviour of the merge step on shrunk matrices.

According to Figure 4.7, the same cannot be said for the purge step. Indeed, the shrinking step interacts with the purge step in mysterious ways: The rescaled sizes after the purge step are always larger than their real counterpart and for sufficiently large shrink factors, the rescaled sizes after the purge step are remarkably stable on average. In fact, there seems to be a threshold

shrink factor below which the rescaled matrix sizes after the purge step rapidly increase with the shrink factor and above which these rescaled sizes are stable no matter the shrink factor. Neither behaviour have been explained to this day.

One will have noticed that this thresholded behaviour between “small” and “large” shrink factors had already been highlighted by Figures 4.5 and 4.6. At first we did not investigate the interactions between the purge and the shrinking steps and rather focused on improving the behaviour of the merge step when a shrink factor is applied, as we shall see in Section 4.4.2. Retrospectively however, we would advise to dig deeper into this alleged correlation, as it is possible that the shape of the curves in Figures 4.5 and 4.6 can actually be explained by the behaviour of the purge step on shrunk matrices.

4.4.2 Suggested improvements for the shrinking step

As emphasized in Section 4.1.2, shrinking a matrix of relations roughly preserves its properties. This is however a first-order evaluation that assumes that the columns of a matrix of relations are very sparse. While this is a reasonable assumption for columns indexed by large ideals, for columns corresponding to small ideals this is quite wrong (see Remark 4.3). In this section, we detail two heuristics that can lead to making the shrinking step’s distortion less significant.

Remark 4.5. *Recall that each column in a matrix of relation is indexed by an ideal. Usually these ideals are sorted by increasing norm so that the columns of the matrix of relations are sorted too. Thus, its first columns are indexed by small ideals and are quite heavy, as small ideals are more likely to intervene in relations than large ones when its last columns are indexed by large ideals and are very light.*

In this section, we will call the former columns the heavy columns, and the latter will be referred to as the light columns or sparse columns.

Notice that there is no clear boundary between light and heavy columns. One thing for sure though: The number of heavy columns is far smaller than the number of light ones as highlighted in Example 4.6

Example 4.6. *To give an intuition about the distribution of column densities in an NFS context, we retrieved the weight of each column of the matrix of relations after the purge step in the DLP-585 experiment.*

This matrix has around 15.9 million rows (see Table 4.3). The first column has no nonzero element (we explain why in Remark 4.9), and the first ten columns have a density above 0.5. After the hundredth column, column densities are already below 0.01, after the thousandth it is below 0.001. In fact, the 99.9% lighter columns have less than 1000 nonzero coefficients (which correspond to a column density of at most 6.4×10^{-5}).

Improving the shrinking step by modifying its interactions with the merge step.

The shrinking step has more consequences on heavy columns than light columns. Indeed, heavy columns contain by definition a lot of nonzero coefficients: When grouping heavy columns there is a high probability that a line has nonzero coefficients on several columns of the group. These several nonzero coefficients in the original matrix are replaced by only one nonzero coefficient in the shrunk matrix. Moreover, if the shrink factor is too large compared to the number of nonzero coefficients of heavy columns, some nonzero coefficients will necessarily be deleted. For instance, if a real matrix of size 1000 with a first column containing 500 nonzero coefficients is shrunk with a shrink factor of 10, the resulting matrix’s first column cannot contain more than 100 nonzero coefficients so 400 nonzero coefficients were deleted.

As we have seen in Definition 1.11, the density of a matrix in the context of the filtering step of NFS is the ratio between its number of nonzero coefficients and its size. As of now, we assumed that the density of a shrunk matrix was the same than the density of the original matrix. As a

result, to predict the size of the real matrix after the filtering step with a target density d , we can launch the filtering step on a shrunk matrix with the same target density d . But the two phenomenons above actually imply that the density of a shrunk matrix is likely to be less than the density of the matrix it came from. Besides, Figure 4.8 supports this claim as the density of the shrunk matrices entering the first pass of the merge step is indeed lower than the density of the real matrix at the same step of the filtering step. It means that a shrunk filtered matrix's size obtained with a target density d probably corresponds to an original filtered matrix's size obtained with a density greater than d .

To mitigate this issue, we introduce the concept of *corrected density*.

Definition 4.7. *Let S be a matrix and σ a positive shrink factor. The density of any matrix M such that $M(\sigma) = S$ is a corrected density for S .*

Notice that several original matrices can give birth to the same shrunk matrix. All these original matrices may not have the same density but all of them are likely to have a density greater than the density of S .

Recall that the merge step is done in several passes at the end of which the density of the matrix being merged is compared with a target density. The idea is to drive the merge step using corrected densities instead of densities. Thus, if $M(\sigma)$ is a shrunk matrix, M_0 the matrix output by the purge step when its input is $M(\sigma)$ and for all $i \geq 1$, M_i is the matrix at the end of the i -th pass of the merge step when its input is M_{i-1} , instead of comparing the density of M_i to the target density, we would like to compare a corrected density of M_i with the target density. Doing so, we hope to better take into account the fact that a shrunk matrix underestimates the density of the original matrix. Said otherwise, we believe that a corrected density is a better quantity to guide the merge step when a shrinking step is applied on the input matrix.

We now have to estimate a corrected density for all the M_i . A corrected density is the density of a fictional matrix \widetilde{M}_i such that $M_i = \widetilde{M}_i(\sigma)$. Notice that \widetilde{M}_i does not really exist, and in particular it is *not* the matrix at the end of the i -th path of the merge step when the matrix input in the whole filtering step is the real matrix M . Proposition 4.8 yields an approximation of a corrected density for a matrix M under the two following hypotheses:

1. For each group of σ columns $C_0, C_1, \dots, C_{\sigma-1}$ of M that are shrunk into one column of $M(\sigma)$ the densities of $C_0, C_1, \dots, C_{\sigma-1}$ are the same.
2. For all group of σ columns $C_0, C_1, \dots, C_{\sigma-1}$ of M that are shrunk into one column of $M(\sigma)$ and for all row r in M , the events in the set

$$\{\text{there is a zero coefficient at the intersection of column } C_i \text{ and row } r \mid i \in \llbracket 0, \sigma - 1 \rrbracket\}$$

are independent.

Proposition 4.8. *Let M be a matrix, σ a positive integer shrink factor and $M(\sigma)$ the matrix resulting from the shrinking of the matrix M with a factor σ . We note N the size of the matrix $M(\sigma)$ and for all $i \in [0, N - 1]$, we let w_i be the weight of the i -th column of $M(\sigma)$ i.e., the number of nonzero coefficients in its i -th column.*

Under the two previous hypotheses, a corrected density of the matrix $M(\sigma)$ is

$$\sigma \sum_{i=0}^{N-1} \left(1 - \left(1 - \frac{w_i}{N} \right)^{1/\sigma} \right)$$

Proof. By definition, one of the corrected densities of $M(\sigma)$ is $\frac{W}{\sigma N}$ where W denotes the total weight of M . We will denote by D this corrected density and will continue to call D the approximation of D that follows from the two previous hypotheses. For all $i \in [0, N - 1]$, we note W_i the total weight of the i -th group of σ columns of M so that $W = \sum_{i=0}^{N-1} W_i$.

Let $i \in [0, N-1]$. According to our first hypothesis, the σ columns in the i -th group of columns of M have the same density (as defined in Definition 1.11) that we denote d_i . Since they also have the same size, namely σN , they have the same weight: $d_i \times \sigma N$. As a result, $W_i = \sigma \times d_i \times \sigma N$ which implies that D equals $\sigma \times \sum_{i=0}^{N-1} d_i$.

Now we express d_i as a function of N and the weights w_i . On the one hand, the density of the i -th column of $M(\sigma)$ is by definition w_i/N . On the other hand, if the σ columns of the i -th block have the same density d_i , the density of the column in $M(\sigma)$ resulting from the shrinking of these columns is $1 - (1 - d_i)^\sigma$. Indeed the probability to have a nonzero coefficient in the shrunk column for a given row is $1 - P$ where P is the probability to have a zero coefficient in the shrunk column for a given row. But there is a zero coefficient in the shrunk column at a given row if and only if all the coefficients at this row in the columns of the i -th block of columns in M are zero. The probability that a column in the i -th block has a zero coefficient at a given row is $1 - d_i$ which means that $P = (1 - d_i)^\sigma$ because of the independence guaranteed by our second hypothesis. In the end, we have:

$$\frac{w_i}{N} = 1 - (1 - d_i)^\sigma$$

This implies that

$$d_i = 1 - \left(1 - \frac{w_i}{N}\right)^{1/\sigma}$$

which yields the expected expression for the (estimated) corrected density D . □

Proposition 4.8 calls for a few remarks:

- Using the notations of Proposition 4.8, we have $\sigma \left(1 - \left(1 - \frac{w_i}{N}\right)^{1/\sigma}\right) \geq \frac{w_i}{N}$ for all $i \in [0, N-1]$ so the corrected density D of $M(\sigma)$ is indeed greater than the density of $M(\sigma)$, namely $\sum_{i=0}^{N-1} w_i/N$.
- For light columns, the ratio w_i/N is small so $(1 - w_i/N)^{1/\sigma} \simeq 1 - \sigma w_i/N$. This implies that, if all columns of the original matrix are light, the density of $M(\sigma)$ and its corrected density D are very close. This was expected since a corrected density aims at taking into account the impact of heavy columns.

Both observations are coherent with the fact that shrinking a matrix underestimates the density of the real matrix because of the distortion it induces on heavy columns.

Revisiting the experiments of Section 4.4.1

We modified the merge script in the CADO-NFS software so that it allows to use corrected densities to rule merge passes if required. We then conducted the same experiment than the one described in 4.4.1 only specifying the use of corrected densities in the merge step. Plotting the average error percentage between real and shrunk matrices sizes after the merge step in function of the shrink factor yields Figure 4.9.

Figure 4.9 tells us that using corrected densities in the merge step simply translates the curves obtained in Figures 4.5 and 4.6 upward by roughly 3%. The fact that the errors are larger when using corrected densities is not surprising as the corrected density of a shrunk matrix is larger than its density, meaning that for a same shrink factor the merge step stops earlier when using corrected densities than when using densities, leading to larger matrices. The fact that the shape of the curves in Figure 4.9 is the same than in Figures 4.5 and 4.6 further supports the idea that the purge step is mostly responsible for it, as the only step impacted by the use of corrected densities is the merge step.

Moreover, if the quality of predictions increases for “large” shrink factors, it decreases for “small” shrink factors. This is not very satisfying, yet one can argue that when using the ESTIMATEMAT-SIZE algorithm with a nontrivial shrinking step, we would like to use large shrink factors anyway. Thus, Figure 4.9 sounds like good news for practical estimations.

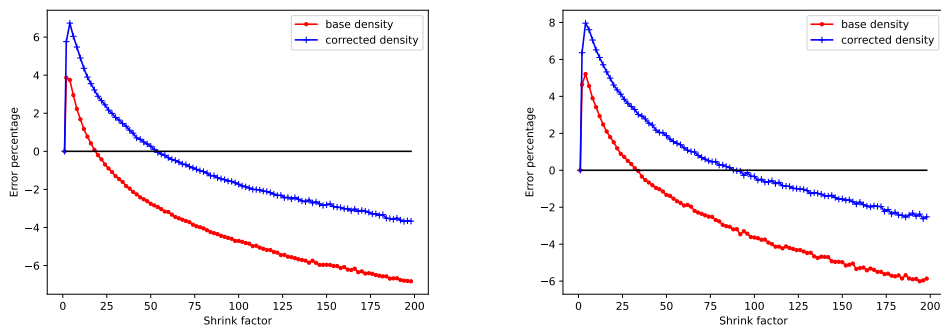


Figure 4.9: Error percentage between the size of the real matrix size after the merge step and a σ -shrunk matrix size after the merge step. Curves tagged with bullets correspond to a merge step ruled with the merged matrix density, curves tagged with crosses correspond to a merge step ruled with the merged matrix corrected density. On the left, the DLP-585 case. On the right, the DLP-512 case.

However, we definitely advise caution, should anyone want to use the ESTIMATEMATSIZE algorithm with corrected densities in the merge step. Indeed, the first hypothesis on which the proof of Proposition 4.8 relies is definitely questionable. In fact, the first σ (heavy) columns do *not* have the same density, as hinted by example 4.6. It is particularly the case in a discrete logarithm context where the first column is a column of ones (see why in Remark 4.9) while the next ones see their density decrease rapidly. As a result the first column of the shrunk matrix, no matter the shrink factor, is full of ones too. According to Proposition 4.8, this full column is considered to account for σ full columns in the real matrix, which is not the case. Consequently, for large shrink factors, the corrected density should greatly overestimate the density of the real matrix, thus stopping the merge step almost immediately which would lead to matrix sizes after the filtering step far larger than the real corresponding size. The fact that this undesired property does not appear in Figure 4.9 makes us suspect either an interaction of the corrected density with some other parameters or steps of ESTIMATEMATSIZE that could counterbalance it or a bug in our implementation. We therefore emphasize that the option allowing to use corrected densities should be tested further.

Remark 4.9. *Why is the first column of the matrix of relations full in NFS-DL? This situation occurs when a polynomial f chosen to build one of the number fields has a leading coefficient that is not equal to one. In practice, this is often the case as allowing some leeway on the leading coefficients in the polynomial step leads to overall better polynomials. In this case, the ideal $\langle 1, \alpha \rangle$ where α is a root of f is not in \mathcal{O}_K where $K = \mathbb{Q}/(f)$. The inverse of this ideal in the group of fractional ideals is denoted by J . If u, v are coprime integers, then the ideal $J(u - v\alpha)$ is in \mathcal{O}_K this time and can be factored. Thus, the factorization of $(u - v\alpha)$ always involves the ideal J^{-1} and the corresponding row in the matrix of relations presents a one in a special column labeled by J^{-1} . Let us notice that J^{-1} is not necessarily prime but this is no big deal and we consider J^{-1} as if it were part of the factor base. The column corresponding to the ideal J^{-1} is placed in first position by convention. This explains why the first column in NFS-DL is a column of ones.*

One can wonder why then it is not the case in the context of factorization. In fact, polynomials with a leading coefficient not equal to one can also be selected in NFS and this leads to a column of ones in the matrix of relations too. But in the case of NFS, the column labeled by J^{-1} — as well as any “inconvenient column” — can be ignored both by the filtering and linear algebra steps and the use of characters allows to make up for the information thus discarded. In an NFS-DL context though, this first column cannot be ignored.

Introducing more flexibility in the shrinking process

We have described above why the impact of the shrink factor is not the same on heavy columns and light columns. This gives birth to the idea of using an “adaptative” shrinking step that would respect better the discrepancy in density between heavy columns and light ones. The idea is to shrink light columns a lot as they can bear a lot of distortion because of their small density and shrink heavy columns less, perhaps not at all. The hope is that it will lead to a shrunk matrix that respect the properties of the original matrix better while still being smaller than the real one hence reducing estimations computing times.

We implemented a crude version of this adaptative shrink by introducing a threshold parameter in the ESTIMATEMATSIZE algorithm. If an experiment is launched with a shrink factor σ and a threshold τ , the first τ columns of the matrix of relations will be kept as is and the shrink factor σ will only be applied on columns with an index above τ . The shrinking of rows is not modified.

When a nonzero threshold is used together with a correction of density in the merge step, the formula of Proposition 4.8 must be modified accordingly: It becomes

$$\sum_{i=0}^{\tau-1} \frac{w_i}{N} + \sigma \sum_{i=\tau}^{N-1} \left(1 - \left(1 - \frac{w_i}{N} \right)^{1/\sigma} \right)$$

This modification has also been implemented. Besides, we hope that using a threshold when correcting densities will fix the issue of the corrected density formula being theoretically quite wrong for large shrink factors.

This threshold feature has yet to be tested. We would like to stress that the idea of an adaptative shrinking step, though appealing, is quite hard to make practical. At least two obstacles lie in the way. On the one hand, estimating how much shrinking-related distortion can a block of columns endure while still ensuring a controlled disruption of the overall results of ESTIMATEMATSIZE is a delicate problem. On the other hand, adaptatively shrinking columns does not tell how to shrink rows in accordance.

4.5 Conclusion

This study of the ESTIMATEMATSIZE simulation algorithm allows to draw two main conclusions:

1. The existing algorithm ESTIMATEMATSIZE is already a simulation tool that is quite effective, robust and reliable: Almost all the estimations made with the implementation available in CADO-NFS with various parameters and on two different DLPs fell at worst within a 20% margin of the real values.
2. Actually explaining the behaviour of this simulation tool is nightmarish, as the parameters and operations involved in NFS are many, interdependent and intricate.

Overall, this is good news for the future of NFS simulation: In practice it is already possible to simulate the filtering step of NFS quite accurately. More precisely, this study reveals that the behaviour of the generation of fake relations is rather straightforward: The more example special- q , the more accurate and stable the predictions are. The impact of a shrinking step however, is a lot less understood and we suspect that examining its interactions with the purge step might help better understanding it.

Finally, this work also gave birth to a variety of tools customized to retrospectively explore the behaviour of the ESTIMATEMATSIZE algorithm. They will probably be of great help to pursue the analysis of this algorithm and perhaps explaining and improving it further.

Part III

Computation of Riemann-Roch spaces

Chapter 5

A fast geometric algorithm to compute Riemann-Roch spaces

This chapter is based on *A Fast Randomized Geometric Algorithm for Computing Riemann-Roch Spaces*, co-authored with Pierre-Jean Spaenlehauer [LGS20].

It presents a probabilistic variant of Brill-Noether's algorithm for computing a basis of the Riemann-Roch space $L(D)$ associated to a divisor D on a projective nodal plane curve \mathcal{C} over a sufficiently large perfect field K . After a description of the algorithm and a proof of its correctness follows a complexity analysis as well as a study of its probability of failure. An implementation in C++ of this algorithm is available at

<https://gitlab.inria.fr/pspaenle/rrspace>

and experimental data show that it outperforms the reference implementation in the Magma computer algebra system [BCP97].

5.1 Overview of the algorithm

This section provides an overview of the main algorithm to compute the Riemann-Roch space associated to a divisor D on a curve \mathcal{C} , without giving yet all the details on the data structures that we use to represent mathematical objects — in particular, divisors. It also introduces some mild assumptions on \mathcal{C} and D required so that our algorithm does not always fail, as well as notations that will be used throughout this chapter.

5.1.1 Hypothesis on the curve and divisors

This section uses some of the notations introduced in Section 2.1.1. It describes part of the input of our algorithm.

Assumptions on the curve \mathcal{C} .

The curve $\mathcal{C} \subset \mathbb{P}^2$ is an absolutely irreducible projective nodal curve defined over a perfect field K with r nodes. We note $q(X, Y, Z)$ its associated polynomial. Without loss of generality (up to a linear change of coordinates) we can assume that $q \neq Z$ and that all the nodes of the curve belongs to its affine subset \mathcal{C}^0 .

Assumptions on the input divisor D .

We require that D is defined over K , *i.e.*, that it is invariant under the natural action of the Galois group $\text{Gal}(k/K)$ for any extension k of K . In this setting, smooth effective divisors on \mathcal{C}^0 can be thought of as nonzero ideals I in $k[\mathcal{C}^0]$ such that $I + \langle \partial q / \partial X, \partial q / \partial Y \rangle = k[\mathcal{C}^0]$.

If the curve \mathcal{C} is singular, then we need two mild assumptions on the input divisor D to ensure that our algorithm does not always fail. First, the divisor D should be smooth, and its support should be contained in the affine chart \mathcal{C}^0 . To describe the second assumption — which is more technical — we need some insight on the data structure that we will use: The input divisor D will be given as a pair of effective divisors (D_+, D_-) such that $D = D_+ - D_-$. Set

$$d = \begin{cases} \lfloor (\deg(D_+) + r) / \deg(\mathcal{C}) + (\deg(\mathcal{C}) - 1) / 2 \rfloor & \text{if } \binom{\deg(\mathcal{C})+1}{2} \leq \deg(D_+) + r \\ \lfloor (\sqrt{1 + 8(\deg(D_+) + r)} - 1) / 2 \rfloor & \text{otherwise.} \end{cases}$$

When $\binom{\deg(\mathcal{C})+1}{2} = \deg(D_+) + r$, those two quantities match and are equal to $\deg(\mathcal{C})$. We will see in the sequel that this value of d is in fact the smallest integer which ensures the existence of a nonzero form $h \in \overline{K}[\mathcal{C}]$ of degree d such that $(h) \geq D_+ + E$ where we recall that E is the nodal divisor of \mathcal{C} (see Definition 2.6). Our second assumption is that there exists a form h of degree d such that $(h) \geq D_+ + E$ and furthermore $(h) - E$ is a smooth divisor. This is a mild assumption which is satisfied in most cases. In the rare cases where it is not satisfied, a workaround for practical computations — for which we do not prove any theoretical guarantee of success — is to increase slightly the value of d in Algorithm 6 (INTERPOLATE) in order to increase the dimension of the space of such functions h .

Roughly speaking, the assumptions on the input require that the impact of the singularities of \mathcal{C} during the execution of the algorithm is minimal.

5.1.2 Main steps of the algorithm

Our algorithm follows the ideas of Brill and Noether introduced in Section 2.2.2. Remember that any divisor D can be written $D = D_+ - D_-$ with D_+ and D_- are both effective divisors (see Remark 2.8) and that in this context, D_+ describes the possible poles of the elements in $L(D)$ and D_- their zeros.

First, our algorithm computes a common denominator for the elements of a K -basis of the Riemann-Roch space $L(D)$. To do so, we look for a polynomial h of degree d with coefficients unknown, and that vanishes with the right multiplicities at all the points prescribed by $D_+ + E$. Thus, the coefficients of h are solution of a linear system. The degree d is tuned to be as small as possible while guaranteeing that this linear system has nontrivial solutions.

The polynomial h is a solution of an interpolation problem. Unfortunately, h is not a suitable denominator for the elements of $L(D)$ as it has undesired zeros: the nodes of \mathcal{C} , by construction (this is required to use the Brill-Noether residue theorem and prove the correctness of the algorithm) and most probably other points of \mathcal{C} as h is not an exact solution to the aforementioned interpolation problem. Said otherwise, we encounter the situation pictured in Figure 5.1, even if the curve \mathcal{C} is smooth. Let the bold points be the points of D_+ : We have indeed found a polynomial h that goes through them but it also goes through the circled points. Therefore, if h is used as a denominator for the elements of the basis of $L(D)$, it allows too many poles.

Notice that our assumptions on the divisor D (see Section 5.1.1) ensures that it is possible to choose h such that $(h) - E$ is smooth. In this context, none of the circled points in Figure 5.1 is singular. It means that we know exactly where are the undesired zeros of h : The singular undesired zeros of h are described by the divisor E and the smooth undesired zeros of h are described by the smooth divisor $((h) - E) - D_+$.

The trick now is to add this divisor to D_- , which prescribes the zeros of the elements of $L(D)$, to get a divisor D_{num} . This way, we will counterbalance the unwanted zeros of the denominator h by introducing exactly the same zeros in the numerators of the elements of the basis of $L(D)$. We finally get the numerators by computing a basis of the polynomials of degree at most $\deg(h)$ that vanish at all the points of $D_{\text{num}} + E$ with the right multiplicities, which amounts to solving a linear system again.

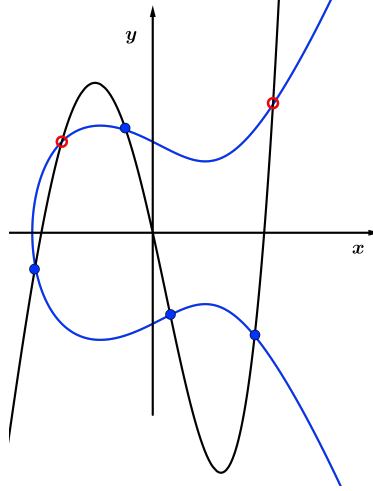


Figure 5.1: The polynomial h has undesired zeros.

We can now give a condensed bird's eye view of our algorithm for computing a Riemann-Roch space provided the hypotheses presented in Section 5.1.1 are satisfied.

Function RIEMANNROCHBASIS;
Data: A curve \mathcal{C} together with its nodal divisor E , and a divisor $D = D_+ - D_-$ on \mathcal{C} such that D_+ and D_- are smooth effective divisors.
Result: A basis of the Riemann-Roch space $L(D)$.
 $h \leftarrow \text{INTERPOLATE}(\deg(\mathcal{C}), D_+, E);$
 $D_h \leftarrow \text{COMPPRINC DIV}(\mathcal{C}, h, E);$
 $D_{\text{res}} \leftarrow \text{SUBTRACTDIVISORS}(D_h, D_+);$
 $D_{\text{num}} \leftarrow \text{ADDDIVISORS}(D_-, D_{\text{res}});$
 $B \leftarrow \text{NUMERATORBASIS}(\deg(\mathcal{C}), D_{\text{num}}, \deg(h), E);$
Return $\{f/h \mid f \in B\}$.

Algorithm 1: A bird's eye view of the algorithm.

We briefly describe what is done at each step of the algorithm. The routine `INTERPOLATE` takes as input an effective divisor D_+ , and it returns a form h such that $(h) \geq D_+ + E$. Then, `COMPPRINC DIV` computes from h a convenient representation of the divisor $(h) - E$. The routines used to perform addition and subtraction of divisors — namely, `ADDDIVISORS` and `SUBTRACTDIVISORS` — will be described in Section 5.2. Then, `NUMERATORBASIS` takes as input the effective divisor D_{num} and the degree of h , and it returns a basis of the vector space of all forms $f \in K[\mathcal{C}]$ of degree $\deg(h)$ such that $(f) \geq D_{\text{num}} + E$. Finally, we divide this basis by the common denominator h in order to obtain a basis of the Riemann-Roch space.

5.1.3 Correction of the algorithm

Assuming that all the subroutines behave correctly, we can prove the general correctness of the main algorithm. As stated in Section 2.2.2 the proof mostly relies on the Brill-Noether residue theorem.

Theorem 5.1. *If all the subroutines `INTERPOLATE`, `COMPPRINC DIV`, `SUBTRACTDIVISORS`, `ADDDIVISORS`, `NUMERATORBASIS` are correct, then Algorithm 1 is correct: It returns a basis of the space $L(D)$.*

Proof. We first prove that there exists a basis of $L(D)$ such that any basis element f belongs to the vector space spanned by the output of Algorithm 1. To this end, we must prove that f can be

written as g/h where h is the output of the subroutine INTERPOLATE and g belongs to the vector space spanned by the output of the subroutine NUMERATORBASIS. Proposition 2.17 with $D = D_+$, $D' = D_+ + (f)$ and h implies that there exists a form $g \in k[\mathcal{C}]$ such that $(g/h) = (f)$, where g has the same degree as h . Therefore, $f = \lambda g/h$ for some nonzero $\lambda \in K$. It remains to prove that g belongs to the vector space spanned by the output of NUMERATORBASIS. Since $f \in L(D)$, we must have $(g) = (f) + (h) \geq (h) - D = (D_h + E - D_+) + D_- = D_{\text{res}} + D_- + E = D_{\text{num}} + E$. But NUMERATORBASIS returns precisely a basis of the space of forms α of the same degree as h such that $(\alpha) \geq D_{\text{num}} + E$.

Conversely, let f be a function returned by Algorithm 1. Then $f \cdot h$ belongs to B , and hence $(f \cdot h) = (f) + (h) \geq D_{\text{num}} + E = (h) - D$. This implies that $(f) \geq -D$ and hence $f \in L(D)$. \square

5.2 Arithmetic on divisors

This section focuses on the data structures used to represent curves, forms and effective divisors. It also describes algorithms to perform additions and subtractions of divisors with this chosen representation.

5.2.1 Data structures

Data structure for the curve \mathcal{C}

We represent the projective curve $\mathcal{C} \subset \mathbb{P}^2$ by its affine model \mathcal{C}^0 in the affine chart $Z \neq 0$ which is described by a bivariate polynomial $q \in K[X, Y]$. We assume that the degree of q in Y equals its total degree. This condition implies that \mathcal{C} is in *projective Noether position* with respect to the projection on the line $Y = 0$, i.e., that the canonical map $K[X, Z] \rightarrow K[\mathcal{C}]$ is injective and that it defines an integral ring extension. This also implies that the map $K[X] \rightarrow K[\mathcal{C}^0]$ is an integral ring extension. We refer to [GLS01, Sec. 3.1] for more details on the projective Noether position.

We emphasize that the projective Noether position is achieved in generic coordinates. Hence this assumption does not lose any generality since it can be enforced by a harmless linear change of coordinates. More precisely, regarding a linear change of coordinate in \mathbb{P}^2 as a 3×3 matrix $M = (m_{ij})_{1 \leq i, j \leq 3}$, the invertible matrices which put the curve in projective Noether position are precisely the matrices whose 9 coefficients do not make a polynomial $P(m_{11}, \dots, m_{33})$ of degree $\deg(\mathcal{C}) + 3$ vanish. This polynomial is the product of $\det(M)$ — which has degree 3 — with the coefficient of $Y^{\deg(\mathcal{C})}$ in the new system of coordinates — which has degree $\deg(\mathcal{C})$. Using Schwartz-Zippel lemma [Sch79, Coro. 1], this implies that the probability that the curve is not in projective Noether position after a linear change of coordinates given by a random matrix whose entries are picked uniformly at random in a finite subset $\mathcal{E} \subset K$ is bounded above by $(\deg(\mathcal{C}) + 3)/|\mathcal{E}|$.

Data structure for forms.

We will represent forms on \mathcal{C} — namely elements in $K[\mathcal{C}] = K[X, Y, Z]/(Z^{\deg(q)}q(X/Z, Y/Z))$ — by their affine counterpart in the affine chart $Z = 1$. Consequently, we shall represent a form $g \in K[\mathcal{C}]$ as an element in $K[X, Y]/q(X, Y)$, given by a representative $\tilde{g} \in K[X, Y]$ such that $\deg_Y(\tilde{g}) < \deg(\mathcal{C})$, using the fact that q is monic in Y . This representation does not encode what happens on the line $Z = 0$ at infinity. In order to encode the behaviour on this line, it is enough to adjoin to \tilde{g} the degree d of the form g , since g is the class of $Z^d \tilde{g}(X/Z, Y/Z)$ in $K[\mathcal{C}]$. In the sequel of this chapter, we do not mention further this issue and we often identify g with \tilde{g} by slight abuse of notation when the context is clear.

Data structure for smooth divisors on \mathcal{C} .

For representing divisors which do not involve any node, we use a data structure strongly inspired by the Mumford representation for divisors on hyperelliptic curves and by representations of algebraic sets by primitive elements as in [Can88]. Our data structure requires a mild assumption on the divisor that we represent: None of the points in the support of the divisor should lie at infinity. In fact, this is not a strong restriction since all points can be brought to an affine chart

via a projective change of coordinate. If one does not wish to change the coordinate system, another solution is to maintain three representations, one for each of the three canonical affine charts covering \mathbb{P}^2 .

We shall represent a smooth divisor D as a pair of smooth effective divisors (D_+, D_-) such that $D = D_+ - D_-$. One crucial point for the representation of effective divisors is that the 0-dimensional algebraic set corresponding to the support (*i.e.* without considering the multiplicities) of an effective divisor D can be described by a finite étale algebra which is a quotient of $k[\mathcal{C}^0]$ by a nonzero ideal. This étale algebra is isomorphic to a quotient of a univariate polynomial ring if it admits a primitive element. Using primitive elements to represent 0-dimensional algebraic sets is a classical technique in computer algebra, see e.g. [Can88, Sec. 2][GLS01, Sec. 3.2].

Lemma 5.2. *Let R be a finite étale K -algebra, i.e., a finite product of finite extensions of K . Let $z \in R$ be an element, and let m_z denote the multiplication by z in R , seen as a K -linear endomorphism. The following statements are equivalent:*

1. *The element z generates R as a K -algebra;*
2. *The elements $1, z, z^2, \dots, z^{\dim_K(R)-1}$ are linearly independent over K ;*
3. *The characteristic polynomial of m_z equals its minimal polynomial;*
4. *The characteristic polynomial of m_z is squarefree.*

If z satisfies these four properties, then z is called a primitive element for R .

Proof. (2) \Rightarrow (1): By definition, the element z generates R as a K -algebra if and only if its powers generate R as a K -vector space.

(1) \Rightarrow (2): Let n_0 be the smallest positive integer such that $1, z, z^2, \dots, z^{n_0}$ are linearly dependent. The integer n_0 must be finite since $\dim_K(R)$ is finite. Write $z^{n_0} = \sum_{i=0}^{n_0-1} a_i z^i$ for some $a_0, \dots, a_{n_0} \in k$. By multiplying this relation by z^{n-n_0} and by induction on n , we obtain that for any $n \geq n_0$, z^n belongs to the vector space generated by $1, z, \dots, z^{n_0-1}$. This implies that the algebra generated by z has dimension n_0 as a K -vector space. By (1), we obtain that $n_0 = \dim_K(R)$.

(2) \Rightarrow (3): By (2), the minimal polynomial of m_z has degree at least $\dim_K(R)$, and hence it equals its characteristic polynomial.

(3) \Rightarrow (2): The degree of the characteristic polynomial is $\dim_K(R)$.

(3) \Rightarrow (4): Let ξ be the squarefree part of the characteristic polynomial of m_z . By (3), $\xi(z)$ must be nilpotent in R . But the only nilpotent element in an étale algebra is 0, so ξ must be a multiple of the minimal polynomial of m_z . Hence, by (3), ξ is the characteristic polynomial of m_z .

(4) \Rightarrow (3): This is a consequence of the facts that the characteristic polynomial and the minimal polynomial have the same set of roots, and the minimal polynomial divides the characteristic polynomial. \square

We are now ready to define the data structure that we will use to represent smooth effective divisors on the curve. A smooth effective divisor D on \mathcal{C} supported on the affine chart $Z \neq 0$ will be represented as:

- A scalar $\lambda \in K$;
- Three univariate polynomials $\chi, u, v \in K[S]$, such that χ is monic, χ has degree $\deg(D)$ and u, v have degree at most $\deg(D) - 1$.

such that

$$(\text{Div-H1}) \quad q(u(S), v(S)) \equiv 0 \pmod{\chi(S)};$$

$$(\text{Div-H2}) \quad \lambda u(S) + v(S) = S;$$

(Div-H3) $\text{GCD}(\frac{\partial q}{\partial X}(u(S), v(S)) - \lambda \frac{\partial q}{\partial Y}(u(S), v(S)), \chi(S)) = 1$.

We call the data structure above a *primitive element representation*. An important ingredient of the primitive element representation is that **(Div-H3)** enables us to use Hensel's lemma to encode the multiplicities. More precisely, **(Div-H3)** implies that at each of the closed points in the support of the divisor, the element $\lambda(X - \bar{x}) + (Y - \bar{y})$ is a uniformizing element for the associated discrete valuation ring, where \bar{x}, \bar{y} denote the classes of X, Y in the residue field.

Notice that this representation requires the existence of a primitive element of the form $\lambda X + Y$ which satisfies all the wanted properties. Fortunately, Proposition 5.3 below shows that such a primitive element exists as soon as K contains more than $\binom{\deg(D)+1}{2}$ elements.

Before introducing the data structure used for nodal divisors, which closely resembles the data structure for smooth divisors, we would like to give intuitions about primitive element representations and informally explain why they will indeed exist when the base field K is large enough. Finding a primitive element representation (λ, u, v, χ) for a smooth effective divisor D amounts to finding a primitive element of $K[\mathcal{C}^0]/(I)$ of the shape $\lambda X + Y$ for some $\lambda \in K$, which allows to build an isomorphism φ such that

$$\varphi : \begin{cases} K[\mathcal{C}^0]/I & \longrightarrow K[S]/\chi(S) \\ X & \longmapsto u(S) \\ Y & \longmapsto v(S) \\ \lambda X + Y & \longmapsto S \end{cases}$$

where I is an ideal such that $K[\mathcal{C}^0]/I$ describes the algebraic set corresponding to the support of D . In this representation, the polynomials u and v parametrize the abscissas and ordinates of the points involved in D with regard to a coordinate system parametrized by λ . Two obstacles may prevent the existence of a suitable λ : They are shown in the left part of Figure 5.2. Once the element λ is chosen, *i.e.*, once a line L is chosen, two points in the support of D might have the same projection on the line L and the tangent to \mathcal{C}^0 at some points in the support of D might be orthogonal to L . The first situation prevents us from defining the polynomial v and the second forbids the use of Hensel's lemma to encode the multiplicities of D . But choosing a different line L , *i.e.*, choosing a different λ will most likely solve both problems as emphasized in the right part of Figure 5.2.

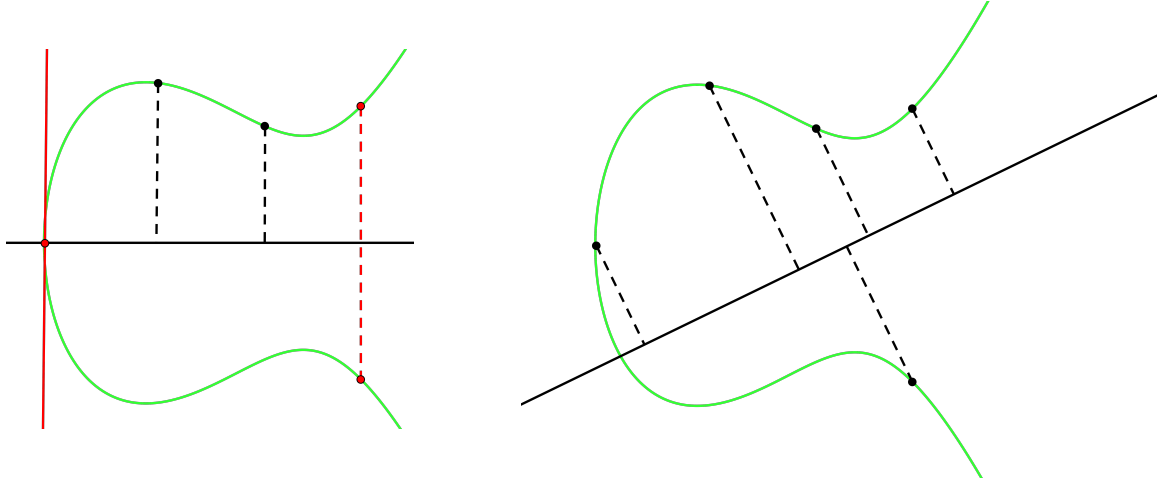


Figure 5.2: About the existence of a primitive element representation for a smooth effective divisor D . On the left: an unsuitable choice for λ . On the right: a suitable choice for λ .

Each point P in the support of D gives birth to one unsuitable line L (the line orthogonal to the tangent to \mathcal{C}^0 at P) and so do each couple of points (P_1, P_2) that both belong to the support

of D , namely the line that goes through P_1 and P_2 . Thus we can easily bound the number of unsuitable lines L , that is the number of unsuitable λ , in function of $\deg(D)$. If the cardinal of K is larger than the aforementioned bound, there is a $\lambda \in K$ such that the problematic situations pictured in the left part of Figure 5.2 never arise. Once such a λ is found, it is then possible to first build primitive element representations for the points in the support of D , lift those representations thanks to Hensel's lemma to encode their multiplicities and finally aggregate those partial representations via the CRT to yield a primitive element representation for D .

Data structure for the nodal divisor.

We shall represent the nodal divisor via an algebraic parametrization by the roots of a univariate polynomial. This algebraic structure is very similar to the representation of smooth divisors, except for a crucial difference: We shall not need to represent multiplicities, so this representation does not need to satisfy condition **(Div-H3)**. More precisely, the nodal divisor E will be represented as:

- A scalar $\lambda_E \in K$;
- Three univariate polynomials $\chi_E, u_E, v_E \in K[S]$, such that χ_E is monic and squarefree, χ_E has degree r and u, v have degree at most $r - 1$;
- A monic univariate polynomial $T_E \in k[S]$ of degree at most $2r$

such that

$$\textbf{(NodDiv-H1)} \quad \{(u_E(\zeta), v_E(\zeta)) \mid \zeta \in \overline{K}, \chi_E(\zeta) = 0\} \subset \overline{K}^2 \text{ is the set of nodes of } \mathcal{C}^0,$$

and such that the roots of T_E are the values $\lambda \in \overline{K}$ such that the vector $(1, -\lambda)$ is tangent to \mathcal{C}^0 at a node. Notice that the roots of T_E do not record vertical tangents at nodes, so the degree of T_E may be less than $2r$.

Such $(\lambda_E, \chi_E, u_E, v_E)$ satisfying **(NodDiv-H1)** exist as soon as K contains more than $\binom{r}{2}$ elements by Proposition 5.4. Computing T_E is also an easy task once $(\lambda_E, \chi_E, u_E, v_E)$ are known. This polynomial can be for instance obtained by considering the homogeneous form $Q_2(X, Y, S)$ of degree 2 of the shifted polynomial $q(X + u_E(S), Y + v_E(S))$. Then $T_E(\lambda) = \text{Resultant}_S(Q_2(1, -\lambda, S), \chi_E(S))$ satisfies the desired property. This polynomial T_E will be useful in Algorithm COMPPRINC DIV. Computing $\lambda_E, \chi_E, u_E, v_E, T_E$ can be thought of as a precomputation since this depends only on the curve \mathcal{C} and not on the input divisor D .

Before going any further, we summarize here the data structures for the input and the output of Algorithm 1 and the properties that they must satisfy.

Input data:

- A bivariate polynomial $q \in K[X, Y]$. This polynomial encodes the curve \mathcal{C} ;
- Data $(\lambda_E, \chi_E, u_E, v_E, T_E)$ encoding the nodal divisor E ;
- A smooth divisor $D = D_+ - D_-$ given by two tuples $(\lambda_+, \chi_+, u_+, v_+)$ and $(\lambda_-, \chi_-, u_-, v_-)$ with $\lambda_{\pm} \in K, \chi_{\pm}, u_{\pm}, v_{\pm} \in K[S]$.

The input data must satisfy the following constraints:

1. The bivariate polynomial $q \in K[X, Y]$ is absolutely irreducible, and its base field K is perfect;
2. The total degree of q equals its degree with respect to Y ;
3. The inequalities $\deg(u_{\pm}) < \deg(\chi_{\pm}), \deg(v_{\pm}) < \deg(\chi_{\pm}), \deg(u_E) < \deg(\chi_E), \deg(v_E) < \deg(\chi_E)$ hold;
4. The polynomials χ_{\pm}, χ_E and T_E are monic;

5. The polynomial χ_E is squarefree;
6. Both tuples $(\lambda_+, \chi_+, u_+, v_+)$ and $(\lambda_-, \chi_-, u_-, v_-)$ satisfy **(Div-H1)** to **(Div-H3)**;
7. The tuple $(\lambda_E, \chi_E, u_E, v_E, T_E)$ satisfies **(NodDiv-H1)**;
8. The degree of T_E is at most $2r$;
9. The roots of the univariate polynomial T_E are the values $\lambda \in \overline{K}$ such that the vector $(1, -\lambda)$ is tangent to \mathcal{C}^0 at a node.

Output data:

- A bivariate polynomial $h \in K[X, Y]$;
- A finite set of bivariate polynomials $B \subset K[X, Y]$.

The output data satisfies that the set $\{b/h \mid b \in B\}$ is a basis of the Riemann-Roch space associated to D on \mathcal{C} .

5.2.2 Existence of primitive element representations and nodal representations

This section is devoted to technical proofs about the primitive element representation and nodal representation. They make rigorous the intuitions that follow the description of primitive element representations in Section 5.2.1. The statements below will be used for proving the correctness of the subalgorithms, but they may be skipped without harming the general understanding of this chapter.

The following proposition asserts that primitive representations of smooth effective divisors do exist provided that the base field is large enough.

Proposition 5.3. *Let J be a nonzero ideal of $K[\mathcal{C}^0] = K[X, Y]/q(X, Y)$ such that $J + \langle \frac{\partial q}{\partial X}, \frac{\partial q}{\partial Y} \rangle = K[\mathcal{C}^0]$. Assume that the cardinality of K is larger than $\binom{\dim_K(K[\mathcal{C}^0]/J)+1}{2}$. Then there exist $\lambda \in K$ and polynomials $\chi, u, v \in K[S]$ satisfying **(Div-H1)** to **(Div-H3)** such that the map $K[\mathcal{C}^0]/J \rightarrow K[S]/\chi(S)$ sending X and Y to the classes of u and v is an isomorphism of K -algebras.*

The following proposition states a similar result for radical ideals which do not satisfy the smoothness assumption. It shows that the representation for nodal divisors described in Section 5.2.1 exists provided that the base field is large enough.

Proposition 5.4. *Let J be a nonzero radical ideal of $K[\mathcal{C}^0] = K[X, Y]/q(X, Y)$. Assume that the cardinality of K is larger than $\binom{\dim_K(K[\mathcal{C}^0]/J)}{2}$. Then there exist $\lambda_E \in K$ and polynomials $\chi_E, u_E, v_E \in K[S]$ satisfying **(NodDiv-H1)** such that the map $K[\mathcal{C}^0]/J \rightarrow K[S]/\chi(S)$ sending X and Y to the classes of u and v is an isomorphism of K -algebras.*

Before proving Propositions 5.3 and 5.4, we need some technical lemmas. First, the following lemma generalizes slightly the classical fact that ideals in the coordinate rings of smooth curves admit a unique factorization. Here, we do not assume that \mathcal{C}^0 is nonsingular, but the factorization property holds for ideals of regular functions which do not vanish at any singular point.

Lemma 5.5. *Let I be a nonzero ideal of $K[\mathcal{C}^0]$ such that $I + \langle \partial q/\partial X, \partial q/\partial Y \rangle = K[\mathcal{C}^0]$. Then there exists a unique factorization $I = \prod_{i=1}^{\ell} \mathfrak{m}_i^{\alpha_i}$ as a product of maximal ideals of $K[\mathcal{C}^0]$.*

Proof. First, we prove the existence of such a factorization. Let $\mathfrak{m} \subset K[\mathcal{C}^0]$ be an ideal containing I . If \mathfrak{m} is a nonsingular closed point of \mathcal{C}^0 , then the local ring $K[\mathcal{C}^0]_{\mathfrak{m}}$ is a discrete valuation ring by [Ful08, Sec. 3.2, Thm. 1]. Let $\text{val}_{\mathfrak{m}}(I) \in \mathbb{Z}_{\geq 0}$ denote the integer such that $I = \mathfrak{m}^{\text{val}_{\mathfrak{m}}(I)}$ in this local ring. Let J be the ideal $J = \prod_{\mathfrak{m} \supset I} \mathfrak{m}^{\text{val}_{\mathfrak{m}}(I)}$. By [AM69, Prop. 9.1], the equality $I = J$

holds if and only if it holds in all the local rings $K[\mathcal{C}^0]_{\mathfrak{m}}$ where $\mathfrak{m} \supset I$. Since the maximal ideals \mathfrak{m} are nonsingular closed points of \mathcal{C}^0 , this equality holds true because the corresponding local rings are discrete valuation rings, hence the equality of ideals is equivalent to the equality of their \mathfrak{m} -valuation.

We now prove the unicity of this factorization: By contradiction, assume that I has two distinct factorizations $\prod_{1 \leq i \leq \ell} \mathfrak{m}_i^{\alpha_i}$ and $\prod_{1 \leq i \leq \ell'} \mathfrak{m}_i^{\alpha'_i}$ of the ideal. Without loss of generality, assume that \mathfrak{m}_1 does not occur in the second factorization, or that it appears with a different multiplicity. This would lead to a contradiction since it would lead to distinct valuations of the same ideal in the local ring at \mathfrak{m}_1 . \square

An ideal $I \subset K[X, Y]$ in a polynomial ring is called 0-dimensional if the dimension of $K[X, Y]/I$ as a K -vector space is finite. The following lemma identifies values of λ for which $\lambda X + Y$ is not a primitive element for a 0-dimensional algebraic set.

Lemma 5.6. *Let $I \subset K[X, Y]$ be a radical 0-dimensional ideal, with associated variety $V = \{\alpha_i\}_{1 \leq i \leq \dim_K(K[X, Y]/I)} \subset \overline{K}^2$ and let $u, v \in K[X, Y]$ be elements such that for any distinct points $\alpha_i, \alpha_j \in V$, $u(\alpha_i) \neq u(\alpha_j)$ or $v(\alpha_i) \neq v(\alpha_j)$. Then the set of $\lambda \in K$ such that $\lambda u + v$ is not a primitive element for $K[X, Y]/I$ is contained in the set of roots of a nonzero univariate polynomial with coefficients in K of degree $\binom{\dim_K(K[X, Y]/I)}{2}$.*

Proof. Writing $\alpha_i = (\alpha_{i,x}, \alpha_{i,y})$, let $k \subset \overline{K}$ be a field extension of K where I factors as a product of degree-1 maximal ideals $\mathfrak{m}_i = \langle X - \alpha_{i,x}, Y - \alpha_{i,y} \rangle$. This provides an isomorphism of k -algebras between $k[X, Y]/I$ and $k^{\dim_K(K[X, Y]/I)}$ sending polynomials to their evaluations at $\alpha_1, \dots, \alpha_{\dim_K(K[X, Y]/I)}$. Using this isomorphism, and letting \mathbf{e}_i denote the i -th canonical vector in $k^{\dim_K(K[X, Y]/I)}$, we observe that \mathbf{e}_i is an eigenvector of the endomorphism of multiplication by $\lambda u + v$, with associated eigenvalue $\lambda u(\alpha_i) + v(\alpha_i)$.

Next, $\lambda u + v$ is a primitive element for $K[X, Y]/I$ if all these eigenvalues are distinct by Lemma 5.2. This is the case if and only if the discriminant of the characteristic polynomial is nonzero. Since the discriminant is the product of the squared differences of the roots, it equals

$$\prod_{\substack{1 \leq i \leq \dim_K(K[X, Y]/I) \\ 1 \leq j < i}} (\lambda(u(\alpha_i) - u(\alpha_j)) + v(\alpha_i) - v(\alpha_j))^2.$$

This discriminant is a polynomial in $K[\lambda]$ of degree $2 \binom{\dim_K(K[X, Y]/I)}{2}$. It is nonzero because of the assumption that for any distinct α_i, α_j , either $u(\alpha_i) \neq u(\alpha_j)$ or $v(\alpha_i) \neq v(\alpha_j)$. Let $\Delta(\lambda) = \prod_{1 \leq j < i} (\lambda(u(\alpha_i) - u(\alpha_j)) + v(\alpha_i) - v(\alpha_j)) \in \overline{K}[\lambda]$ be its square root.

It remains to prove that the polynomial Δ has coefficients in K . This is due to the fact that automorphisms in $\text{Gal}(k/K)$ permute the points α_i . Therefore the natural action of $\text{Gal}(k/K)$ acts on Δ by permuting its linear factors, and hence it leaves Δ invariant. \square

In the following lemma, the notation $\text{red}(R)$ stands for the quotient of a ring R by its Jacobson radical (i.e. the intersection of its maximal ideals). If I is an ideal of R , we use the notation \sqrt{I} to denote the radical of I . The ring $\text{red}(K[\mathcal{C}^0]/J)$ can be thought of as the coordinate ring of the 0-dimensional algebraic set corresponding to the points in the support of the effective divisor associated to J .

Lemma 5.7. *Let J be a nonzero ideal of $K[\mathcal{C}^0] = k[X, Y]/q(X, Y)$ such that $J + \langle \partial q / \partial X, \partial q / \partial Y \rangle = K[\mathcal{C}^0]$. Then there exists a nonzero univariate polynomial Δ with coefficients in K of degree at most $\binom{\dim_K(K[\mathcal{C}^0]/J) + 1}{2}$, such that for any λ which is not a root of Δ , the element $\lambda X + Y$ is primitive for $\text{red}(K[\mathcal{C}^0]/J)$ and $\partial q / \partial X - \lambda \partial q / \partial Y$ is invertible in $\text{red}(K[\mathcal{C}^0]/J)$.*

Proof. First, notice that $K[\mathcal{C}^0]/J$ is isomorphic to $K[X, Y]/(J + \langle q \rangle)$, by using the classical fact that ideals of a quotient ring R/I correspond to ideals of R containing I . Since q is irreducible, $J + \langle q \rangle \subset K[X, Y]$ is a zero-dimensional ideal, hence $\text{red}(K[X, Y]/(J + \langle q \rangle)) = K[X, Y]/\sqrt{J + \langle q \rangle}$.

Notice that $\dim_K(K[X, Y]/\sqrt{J + \langle q \rangle}) \leq \dim_K(K[\mathcal{C}^0]/J)$. Next, using the fact that two distinct points in the variety have distinct coordinates, Lemma 5.6 provides a nonzero polynomial Δ_0 of degree at most $\binom{\dim_K(K[\mathcal{C}^0]/J)}{2}$ such that $\lambda X + Y$ is not a primitive element for $\text{red}(K[\mathcal{C}^0]/J)$ only if λ is a root of Δ_0 .

Next, we notice that since $\sqrt{J + \langle q \rangle} \subset K[X, Y]$ is a radical 0-dimensional ideal, it can be decomposed as a product $\prod_{1 \leq i \leq \ell} \mathfrak{m}_i$ of maximal ideals. Consequently, $\partial q/\partial X - \lambda \partial q/\partial Y$ is invertible in $\text{red}(K[\mathcal{C}^0]/J)$ if and only if $\partial q/\partial X - \lambda \partial q/\partial Y$ does not belong to any of these maximal ideals. Equivalently, $\partial q/\partial X - \lambda \partial q/\partial Y$ must not vanish in any of the residue fields $\kappa_i = K[\mathcal{C}^0]/\mathfrak{m}_i$. Notice that the norm $N_{\kappa_i/K}(\partial q/\partial X - \lambda \partial q/\partial Y)$ is a polynomial Δ_i in λ with coefficients in K . It is nonzero since $J + \langle \partial q/\partial X, \partial q/\partial Y \rangle = K[\mathcal{C}^0]$ and hence either $\partial q/\partial X$ or $\partial q/\partial Y$ is nonzero in κ_i . Therefore Δ_i is either constant (if $\partial q/\partial Y$ vanishes in κ_i), or it has degree $[\kappa_i : K]$.

Finally, the proof is concluded by noticing that $\sum_{1 \leq i \leq \ell} [\kappa_i : K] = \dim_K(K[X, Y]/\sqrt{J + \langle q \rangle}) \leq \dim_K(K[\mathcal{C}^0]/J)$, so that the product $\Delta_0 \cdot \prod_{1 \leq i \leq \ell} \Delta_i$ has degree at most $\binom{\dim_K(K[\mathcal{C}^0]/J)+1}{2}$ and satisfies all the desired properties. \square

We now have all the tools that we need to prove Propositions 5.3 and 5.4.

Proof of Proposition 5.3. First, we assume that $J = \mathfrak{m}^\alpha$ is a power of a maximal ideal in $K[\mathcal{C}^0]$ such that $\mathfrak{m} + \langle \partial q/\partial X, \partial q/\partial Y \rangle = K[\mathcal{C}^0]$. Then $\text{red}(K[\mathcal{C}^0]/J) = K[\mathcal{C}^0]/\mathfrak{m}$. Let $\lambda \in K$ be an element which is not a root of the polynomial Δ provided by Lemma 5.7. Such an element exists since the cardinality of K is larger than the degree of Δ . Therefore, $\lambda X + Y$ is a primitive element for $K[\mathcal{C}^0]/\mathfrak{m}$ and hence there exist univariate polynomials $\tilde{u}, \tilde{v} \in K[S]$ such that $X = \tilde{u}(\lambda X + Y)$ and $Y = \tilde{v}(\lambda X + Y)$ in $K[\mathcal{C}^0]/\mathfrak{m}$. Let $\tilde{\chi}(S)$ be the minimal polynomial of $\lambda X + Y$ in $K[\mathcal{C}^0]/\mathfrak{m}$, which is irreducible since $K[\mathcal{C}^0]/\mathfrak{m}$ is a field. Notice that the map $K[\mathcal{C}^0]/\mathfrak{m} \rightarrow K[S]/\tilde{\chi}(S)$ sending the classes of X, Y to \tilde{u}, \tilde{v} is an isomorphism of K -algebras.

Next, set $\chi(S) = \tilde{\chi}(S)^\alpha$ and consider the bivariate system

$$\begin{cases} q(X, Y) = 0 \\ \lambda X + Y - S = 0. \end{cases} \quad (5.1)$$

By construction, this system has solution (\tilde{u}, \tilde{v}) over $K[S]/\tilde{\chi}(S)$. The Jacobian of this system is $\frac{\partial q}{\partial X}(X, Y) - \lambda \frac{\partial q}{\partial Y}(X, Y)$, which is invertible in $\text{red}(K[\mathcal{C}^0]/J)$ by Lemma 5.7, and therefore $\frac{\partial q}{\partial X}(\tilde{u}(S), \tilde{v}(S)) - \lambda \frac{\partial q}{\partial Y}(\tilde{u}(S), \tilde{v}(S))$ is invertible in $K[S]/\tilde{\chi}(S)$. By Hensel's lemma, there exist polynomials $u, v \in K[S]_{<\deg(\chi)}$ which are solutions of (5.1) over $K[S]/\chi(S)$: Indeed, for $i > 1$, if (\tilde{u}, \tilde{v}) is a solution of (5.1) over $K[S]/\tilde{\chi}(S)^i$, then a Taylor expansion of the system at order 1 shows that

$$\begin{bmatrix} \tilde{u} \\ \tilde{v} \end{bmatrix} - \begin{bmatrix} \partial q/\partial X & \partial q/\partial Y \\ \lambda & 1 \end{bmatrix}^{-1} \cdot \begin{bmatrix} q(\tilde{u}, \tilde{v}) \\ \lambda \tilde{u} + \tilde{v} - S \end{bmatrix}$$

is a solution of Eq. (5.1) over $K[S]/\tilde{\chi}(S)^{2i}$.

The map $K[\mathcal{C}^0]/J \rightarrow K[S]/\chi(S)$ is well-defined because \mathfrak{m} maps to 0 modulo $\tilde{\chi}$, hence $J = \mathfrak{m}^\alpha$ maps to 0 modulo $\chi = \tilde{\chi}^\alpha$. It is an isomorphism because $K[\mathcal{C}^0]/J$ and $K[S]/\chi(S)$ have the same dimension as vector spaces over K and the map $S \mapsto \lambda X + Y$ is the right inverse to the map $(X, Y) \mapsto (u(S), v(S))$. It remains to prove that **(Div-H3)** is satisfied by χ, u, v , which is a direct consequence of the fact that by Lemma 5.7, $\partial q/\partial X - \lambda \partial q/\partial Y$ does not belong to \mathfrak{m} and hence $\frac{\partial q}{\partial X}(u(S), v(S)) - \lambda \frac{\partial q}{\partial Y}(u(S), v(S))$ is invertible modulo $\chi(S)$.

Next, we consider the general case where J is a nonzero ideal in $K[\mathcal{C}^0]$ such that $J + \langle \partial q/\partial X, \partial q/\partial Y \rangle = K[\mathcal{C}^0]$. Again, let $\lambda \in K$ be an element which is not a root of the polynomial Δ provided by Lemma 5.7. Lemma 5.5 implies that J can be written as a product $J = \prod_{i=1}^\ell \mathfrak{m}_i^{\alpha_i}$ of powers of maximal ideals. Then for all i , the element $\lambda X + Y$ is primitive for $K[\mathcal{C}^0]/\mathfrak{m}$ and $\partial q/\partial X - \lambda \partial q/\partial Y$ is invertible in $K[\mathcal{C}^0]/\mathfrak{m}$. For each i , using the previous argument, we can construct univariate polynomials $\chi_i, u_i, v_i \in K[S]$ satisfying **(Div-H1)** to **(Div-H3)** with respect to λ such that the

maps $K[\mathcal{C}^0]/\mathfrak{m}_i^{\alpha_i} \rightarrow K[S]/\chi_i(S)$ sending X, Y to $u_i(S), v_i(S)$ are isomorphisms of K -algebras. Setting $\chi(S) = \prod_{i=1}^{\ell} \chi_i(S)$ and using the CRT, let $u, v \in K[S] < \deg(\chi)$ be such that for all i , we have $u(S) \equiv u_i(S) \pmod{\chi_i(S)}$ and $v(S) \equiv v_i(S) \pmod{\chi_i(S)}$. Then the fact that the CRT is a ring morphism allows us to conclude that the map $K[\mathcal{C}^0]/J \rightarrow K[S]/\chi(S)$ is an isomorphism and that χ, u, v satisfy **(Div-H1)** to **(Div-H3)**. \square

Proof of Proposition 5.4. The proof is similar to that of Proposition 5.3, by ignoring the argument about multiplicities. Since the Jacobian $\frac{\partial q}{\partial X}(X, Y) - \lambda \frac{\partial q}{\partial Y}(X, Y)$ need not be invertible in $\text{red}(k[\mathcal{C}^0]/J)$, it is sufficient to choose a value of λ which is not a root of the univariate polynomial constructed in Lemma 5.6 for J . \square

We have shown that the primitive element representation of an effective divisor and the nodal representation of the nodal divisor exist. The primitive element representation of an effective divisor is not unique: Two tuples $(\lambda_1, \chi_1, u_1, v_1)$ and $(\lambda_2, \chi_2, u_2, v_2)$ may encode the same effective divisor. The cases where this happens are detailed in the following proposition.

Proposition 5.8. *Let $(\lambda_1, \chi_1, u_1, v_1), (\lambda_2, \chi_2, u_2, v_2) \in K \times K[S]^3$ be data which satisfy **(Div-H2)**. Let $I_1, I_2 \subset K[X, Y, S]$ be the associated ideals $I_1 = \langle \chi_1(S), X - u_1(S), Y - v_1(S) \rangle, I_2 = \langle \chi_2(S), X - u_2(S), Y - v_2(S) \rangle$.*

Then $I_1 \cap K[X, Y] = I_2 \cap K[X, Y]$ if and only if χ_1 is the characteristic polynomial of $\lambda_1 u_2 + v_2$ in $K[S]/\chi_2(S)$, $u_1(\lambda_1 u_2(S) + v_2(S)) \equiv u_2(S) \pmod{\chi_2(S)}$ and $v_1(\lambda_1 u_2(S) + v_2(S)) \equiv v_2(S) \pmod{\chi_2(S)}$.

Proof. We first prove the “if” part of the statement. First, we notice that $K[X, Y]/(I_1 \cap K[X, Y])$ and $K[X, Y]/(I_2 \cap K[X, Y])$ are K -vector space of the same finite dimension, since $\deg(\chi_1)$ must equal $\deg(\chi_2)$. Therefore it is enough to show one inclusion to prove the equality. Let $f(X, Y) \in I_1 \cap K[X, Y]$. Using the equalities modulo χ_2 , we obtain that $f(X, Y) \equiv f(u_1(\lambda_1 u_2(S) + v_2(S)), v_1(\lambda_1 u_2(S) + v_2(S))) \pmod{I_2}$, which is divisible by $\chi_1(\lambda_1 u_2(S) + v_2(S))$ because f is in I_1 and by using Cayley-Hamilton theorem. Finally, we use the fact that $\chi_1(S)$ is the characteristic polynomial of $\lambda_1 u_2(S) + v_2(S)$ and hence χ_2 divides $\chi_1(\lambda_1 u_2(S) + v_2(S))$, which finishes to prove that $f \in I_2$.

Conversely, assume that $I_1 \cap K[X, Y] = I_2 \cap K[X, Y]$. By composing the isomorphisms

$$\begin{array}{ccccc} K[S]/\chi_1(S) & \rightarrow & K[X, Y]/(I_1 \cap K[X, Y]) & K[X, Y]/(I_2 \cap K[X, Y]) & \rightarrow & K[S]/\chi_2(S) \\ S & \mapsto & \lambda_1 X + Y & X & \mapsto & u_2(S) \\ & & & Y & \mapsto & v_2(S) \end{array}$$

we obtain that the map $K[S]/\chi_1(S) \rightarrow K[S]/\chi_2(S)$ which sends S to $\lambda_1 u_2(S) + v_2(S)$ is an isomorphism. This proves that χ_1 is the characteristic polynomial of $\lambda_1 u_2(S) + v_2(S)$ in $K[S]/\chi_2(S)$. To prove the two congruence relations, we observe that for all $f \in K[X, Y]$, $f(u_1, v_1) \equiv 0 \pmod{\chi_1}$ if and only if $f(u_2, v_2) \equiv 0 \pmod{\chi_2}$. In particular, the polynomial $P(X, Y) = u_1(\lambda_1 X + Y) - X$ satisfies $P(u_1, v_1) \equiv 0 \pmod{\chi_1}$, and hence $P(u_2, v_2) \equiv 0 \pmod{\chi_2}$. The proof of the last congruence relation is similar. \square

The proofs of the corectness of the subroutines involved in our main algorithm will be made easier by the two propositions below. They give a few more insight on how primitive element representations behave.

We have already shown that any effective smooth divisor has a primitive element representation. The next lemma shows a reciprocal: Any data satisfying **(Div-H1)** to **(Div-H3)** actually encodes a well-defined effective divisor with no singular point in its support.

Lemma 5.9. *Let (λ, χ, u, v) be such that **(Div-H1)** to **(Div-H3)** are satisfied, and let $I = \langle X - u(S), Y - v(S), \chi(S) \rangle \cap K[X, Y]$. Then $K[X, Y]/I$ is isomorphic as a K -algebra to $K[\mathcal{C}^0]/J$ where J is a nonzero ideal in $K[\mathcal{C}^0]$. Moreover, $J + \langle \partial q / \partial X, \partial q / \partial Y \rangle = K[\mathcal{C}^0]$, $\lambda X + Y$ is a primitive element for $\text{red}(K[X, Y]/I)$, and its minimal polynomial is the squarefree part of χ .*

Proof. Nonzero ideals of $K[\mathcal{C}^0]$ correspond to ideals of $K[X, Y]$ containing properly the principal ideal $\langle q(X, Y) \rangle$.

First, notice that **(Div-H1)** implies that $q(X, Y) \in I$. Also, by **(Div-H2)**, we get that $\chi(\lambda X + Y) \in I$. Notice that $\chi(\lambda X + Y)$ factors as a product of polynomials of degree 1 over the algebraic closure of K . Since q is supposed to be absolutely irreducible and to have degree at least 2, this implies that $\chi(\lambda X + Y)$ does not belong to the principal ideal $\langle q(X, Y) \rangle$. Consequently, I contains properly $\langle q(X, Y) \rangle$ and this proves the isomorphism between $K[X, Y]/I$ and $K[\mathcal{C}^0]/J$. In particular, we obtain that $\dim_K(K[\mathcal{C}^0]/J) = \dim_K(K[X, Y]/I) = \deg(\chi)$.

Next, **(Div-H3)** implies that $\partial q/\partial X - \lambda \partial q/\partial Y$ is invertible in $K[\mathcal{C}^0]/J$, hence $K[\mathcal{C}^0] = J + \langle \partial q/\partial X - \lambda \partial q/\partial Y \rangle \subset J + \langle \partial q/\partial X, \partial q/\partial Y \rangle$. Therefore, $J + \langle \partial q/\partial X, \partial q/\partial Y \rangle = K[\mathcal{C}^0]$. Using the isomorphism between $K[\mathcal{C}^0]/J$ and $K[S]/\chi(S)$ described in Proposition 5.3, we obtain that $\text{red}(K[X, Y]/I)$ is isomorphic to $\text{red}(K[S]/\chi(S))$, which is in turn isomorphic to $K[S]/\tilde{\chi}(S)$, where $\tilde{\chi}(S)$ is the squarefree part of $\chi(S)$. Finally, the proof is concluded by noticing that S is a primitive element for $K[S]/\tilde{\chi}(S)$ with minimal polynomial $\tilde{\chi}(S)$. \square

The following lemma explicits the link between the primitive element representation and the ideal vanishing on the 0-dimensional algebraic set that it represents.

Lemma 5.10. *Let (λ, χ, u, v) be data satisfying **(Div-H2)**. Set $I = \langle \chi(S), X - u(S), Y - v(S) \rangle \subset K[X, Y, S]$ and $J = \langle \chi(\lambda X + Y), X - u(\lambda X + Y), Y - v(\lambda X + Y) \rangle$. Then $I \cap K[X, Y] = J$.*

Proof. By **(Div-H2)** and by using the fact that $X - u(S), Y - v(S) \in I$, we deduce that $S - (\lambda X + Y) \in I$. This implies that $I \cap K[X, Y] = \{f(X, Y, \lambda X + Y) \mid f \in I\}$. \square

5.2.3 Operations on smooth divisors

From now on in this chapter, for $d > 0$ we let $K[S]_{<d}$ denote the vector space of univariate polynomials with coefficients in K of degree less than d . This section is devoted to presenting and proving algorithms allowing to perform sums and subtractions of smooth effective divisors given by primitive element representations. The take home message of this section is the following: If divisors are encoded by primitive element representations, then performing operations on divisors roughly boils down to performing simple operations on univariate polynomials.

The first step to perform arithmetic operations involving two smooth divisors given by primitive element representations is to agree on a common primitive element. In order to achieve this, the routine **CHANGEPRIME**LT (Algorithm 2) performs the necessary change of primitive element by using linear algebra. We will prove in Propositions 5.20 and 5.26 that the complexity of this step is the same as the complexity of the subroutine **NUMERATORBASIS** in the main algorithm. Therefore, decreasing the complexity of **CHANGEPRIME**LT would not change the global complexity and hence we make no effort to optimize it, although it might be possible to obtain a better complexity for this step by using a method similar to [GLS01, Algo. 5].

Proposition 5.11. *Algorithm 2 (**CHANGEPRIME**LT) is correct: If it does not fail, then $(\tilde{\lambda}, \tilde{\chi}, \tilde{u}, \tilde{v})$ satisfies properties **(Div-H1)** to **(Div-H3)** and it represents the same effective divisor as (λ, χ, u, v) .*

Proof. First, we prove that $(\tilde{\lambda}, \tilde{\chi}, \tilde{u}, \tilde{v})$ satisfies Properties **(Div-H1)** to **(Div-H3)**. We notice that the map ψ in Algorithm 2 can be extended to an isomorphism Ψ of K -algebras between $K[S]/\tilde{\chi}(S)$ and $K[S]/\chi(S)$. Property **(Div-H1)** follows from the fact that in $K[S]/\chi(S)$, we have $q(\tilde{u}, \tilde{v}) = q(\Psi^{-1}(u), \Psi^{-1}(v)) = \Psi^{-1}(q(u, v)) = 0$. Property **(Div-H2)** follows from the equalities $S = \Psi^{-1}(\Psi(S)) = \Psi^{-1}(\tilde{\lambda}u + v) = \tilde{\lambda}\psi^{-1}(u(S)) + \psi^{-1}(v(S)) = \tilde{\lambda}\tilde{u}(S) + \tilde{v}(S)$ in $K[S]/\tilde{\chi}(S)$. The fact that the equality $\tilde{\lambda}\tilde{u}(S) + \tilde{v}(S) = S$ also holds in $K[S]$ is a consequence of the degree bounds $\deg(\tilde{u}), \deg(\tilde{v}) < \deg(\tilde{\chi})$. If the first test does not fail, then $\frac{\partial q}{\partial X}(u(S), v(S)) - \tilde{\lambda} \frac{\partial q}{\partial Y}(u(S), v(S))$ is invertible modulo $\chi(S)$. Applying Ψ^{-1} shows **(Div-H3)**.

Function CHANGEPRIMELT;

Data: A scalar $\tilde{\lambda} \in K$ and a primitive element representation (λ, χ, u, v) of a smooth effective divisor D .

Result: Univariate polynomials $(\tilde{\chi}, \tilde{u}, \tilde{v})$ such that $(\tilde{\lambda}, \tilde{\chi}, \tilde{u}, \tilde{v})$ is a primitive element representation of D or “fail”.

if $\text{GCD}(\frac{\partial q}{\partial X}(u(S), v(S)) - \tilde{\lambda} \frac{\partial q}{\partial Y}(u(S), v(S)), \chi(S)) \neq 1$ **then**
 | Return “fail”.

end

$M \leftarrow \deg(\chi) \times \deg(\chi)$ matrix representing the linear map $\varphi : K[S]_{<\deg(\chi)} \rightarrow K[S]_{<\deg(\chi)}$
 such that $\varphi(f)(S) \equiv f(S) \cdot (\tilde{\lambda}u(S) + v(S)) \bmod \chi(S)$;

$\tilde{\chi} \leftarrow \text{CHARACTERISTICPOLYNOMIAL}(M)$;

$N \leftarrow \deg(\chi) \times \deg(\chi)$ matrix representing the linear map $\psi : K[S]_{<\deg(\tilde{\chi})} \rightarrow K[S]_{<\deg(\chi)}$
 such that $\psi(f)(S) \equiv f(\tilde{\lambda}u(S) + v(S)) \bmod \chi(S)$;

if N is not invertible **then**
 | Return “fail”.

end

$\tilde{u} \leftarrow \psi^{-1}(u)$;

$\tilde{v} \leftarrow \psi^{-1}(v)$;

Return $(\tilde{\chi}, \tilde{u}, \tilde{v})$.

Algorithm 2: Changing the primitive element in the representation of a smooth effective divisor.

Finally, we must prove that both representations encode the same divisor. By Proposition 5.8, this amounts to show that

$$\begin{cases} \tilde{\chi} \text{ is the characteristic polynomial of } \tilde{\lambda}u(S) + v(S) \\ \tilde{u}(\tilde{\lambda}u(S) + v(S)) \equiv u(S) \bmod \chi(S) \text{ and} \\ \tilde{v}(\tilde{\lambda}u(S) + v(S)) \equiv v(S) \bmod \chi(S). \end{cases}$$

The first item is immediate because of the definition of $\tilde{\chi}$ and the last two items are again proved directly by using the isomorphism Ψ^{-1} . \square

For the addition algorithm we also need another subalgorithm to take into account a potential increase in some multiplicities. This is done using Algorithm 3 which relies on a Hensel lift. Notice that we have already used a similar argument in the proof of Proposition 5.3 to build a primitive element representation for a divisor mP where P is a point and $m > 1$ from a primitive element representation for the divisor P .

Function HENSELLIFTINGSTEP;

Data: A squarefree bivariate polynomial $q \in K[X, Y]$, (λ, χ, u, v) which satisfies **(Div-H1)** to **(Div-H3)**, and a univariate polynomial $\hat{\chi}$ which divides χ^2 .

Result: Two polynomials $\hat{u}, \hat{v} \in K[S]_{<\deg(\hat{\chi})}$ such that $(\lambda, \hat{\chi}, \hat{u}, \hat{v})$ satisfies **(Div-H1)** to **(Div-H3)**.

$$\hat{u}(S) \leftarrow \left(u(S) - \frac{q(u(S), v(S)) - (\lambda u(S) + v(S) - S) \frac{\partial q}{\partial Y}(u(S), v(S))}{\frac{\partial q}{\partial X}(u(S), v(S)) - \lambda \frac{\partial q}{\partial Y}(u(S), v(S))} \right) \bmod \hat{\chi}(S);$$

$$\hat{v}(S) \leftarrow \left(v(S) - \frac{-\lambda q(u(S), v(S)) + (\lambda u(S) + v(S) - S) \frac{\partial q}{\partial X}(u(S), v(S))}{\frac{\partial q}{\partial X}(u(S), v(S)) - \lambda \frac{\partial q}{\partial Y}(u(S), v(S))} \right) \bmod \hat{\chi}(S);$$

Return (\hat{u}, \hat{v}) .

Algorithm 3: A step of Newton-Hensel’s lifting.

Proposition 5.12. *Algorithm 3 (HENSELLIFTINGSTEP) is correct: $(\lambda, \hat{\chi}, \hat{u}, \hat{v})$ satisfies (Div-H1) to (Div-H3).*

Proof. This is a special case of Newton-Hensel's lifting. Using Taylor expansion,

$$\begin{bmatrix} q(X, Y) \\ \lambda X + Y - S \end{bmatrix} = \begin{bmatrix} q(u(S), v(S)) \\ \lambda u(S) + v(S) - S \end{bmatrix} + \begin{bmatrix} \frac{\partial q}{\partial X}(u(S), v(S)) & \frac{\partial q}{\partial Y}(u(S), v(S)) \\ \lambda & 1 \end{bmatrix} \cdot \begin{bmatrix} X - u(S) \\ Y - v(S) \end{bmatrix} + \varepsilon(X, Y, S),$$

where ε is such that $\varepsilon(\tilde{u}(S), \tilde{v}(S), S) \equiv 0 \pmod{\chi(S)^2}$ for any polynomials $\tilde{u}, \tilde{v} \in K[S]$ such that $\tilde{u} \equiv u \pmod{\chi}$ and $\tilde{v} \equiv v \pmod{\chi}$.

Next, notice that the denominators in the definitions of \hat{u} and \hat{v} are invertible modulo $\chi(S)^2$ because they are invertible modulo $\chi(S)$. The proof of (Div-H1) and (Div-H2) follows from a direct computation by plugging the values of \hat{u} and \hat{v} in the Taylor expansion, and by noticing that $\hat{u} \equiv u \pmod{\chi}$ and $\hat{v} \equiv v \pmod{\chi}$, so that $\varepsilon(\hat{u}(S), \hat{v}(S), S) \equiv 0 \pmod{\chi(S)^2}$ and hence $\varepsilon(\hat{u}(S), \hat{v}(S), S) \equiv 0 \pmod{\hat{\chi}(S)}$. Finally, (Div-H3) is a direct consequence of the fact that $\frac{\partial q}{\partial X}(u(S), v(S)) - \lambda \frac{\partial q}{\partial Y}(u(S), v(S))$ is invertible modulo $\chi(S)$. \square

Now that we have a way to make representations agree on a common primitive element and a tool to modify the multiplicities in smooth divisors within the realm of primitive element representations, we can present algorithms to add and subtract effective divisors represented by one of their primitive element representation.

Function ADDDIVISORS;

Data: A polynomial $q \in K[X, Y]$ and two smooth effective divisors D_1, D_2 given by primitive element representations $(\lambda_1, \chi_1, u_1, v_1)$ and $(\lambda_2, \chi_2, u_2, v_2)$.

Result: A primitive element representation of the divisor $D_1 + D_2$ or “fail”.

$\hat{\lambda} \leftarrow \text{RANDOM}(K)$;

$(\hat{\chi}_1, \hat{u}_1, \hat{v}_1) \leftarrow \text{CHANGEPRIMELT}(\hat{\lambda}, \lambda_1, \chi_1, u_1, v_1)$;

$(\hat{\chi}_2, \hat{u}_2, \hat{v}_2) \leftarrow \text{CHANGEPRIMELT}(\hat{\lambda}, \lambda_2, \chi_2, u_2, v_2)$;

if $\hat{u}_1 \not\equiv \hat{u}_2 \pmod{\text{GCD}(\hat{\chi}_1, \hat{\chi}_2)}$ **then**

 Return “fail”

end

$\hat{\chi} \leftarrow \hat{\chi}_1 \cdot \hat{\chi}_2$;

$\tilde{\chi} \leftarrow \text{LCM}(\hat{\chi}_1, \hat{\chi}_2)$;

$\hat{u}_{12} \leftarrow \text{XCRT}((\hat{\chi}_1, \hat{\chi}_2), (\hat{u}_1, \hat{u}_2)) \in k[S]_{<\deg(\tilde{\chi})}$;

$\hat{v}_{12} \leftarrow \text{XCRT}((\hat{\chi}_1, \hat{\chi}_2), (\hat{v}_1, \hat{v}_2)) \in k[S]_{<\deg(\tilde{\chi})}$;

$(\hat{u}, \hat{v}) \leftarrow \text{HENSELLIFTINGSTEP}(q, \tilde{\chi}, \hat{\lambda}, \hat{u}_{12}, \hat{v}_{12}, \hat{\chi})$;

Return $(\hat{\lambda}, \hat{\chi}, \hat{u}, \hat{v})$.

Algorithm 4: Computing the sum of two smooth effective divisors.

Algorithm 4 uses a variant of the CRT, which we call the *Extended Chinese Remainder Theorem* and which we abbreviate as XCRT. Given four univariate polynomials $u_1, u_2, \chi_1, \chi_2 \in K[S]$ such that $u_1 \equiv u_2 \pmod{\text{GCD}(\chi_1, \chi_2)}$, it returns a polynomial $u \in K[S]$ of degree less than $\deg(\text{LCM}(\chi_1, \chi_2))$ such that $u \equiv u_1 \pmod{\chi_1}$ and $u \equiv u_2 \pmod{\chi_2}$. The main difference with the classical CRT is that we do not require χ_1 and χ_2 to be coprime. A minimal solution to the XCRT problem is given by

$$\text{XCRT}((\chi_1, \chi_2), (u_1, u_2)) = (u_2 a_1 (\chi_1/g) + u_1 a_2 (\chi_2/g)) \pmod{\text{LCM}(\chi_1, \chi_2)}, \quad (5.2)$$

where $g = \text{GCD}(\chi_1, \chi_2)$ and $a_1, a_2 \in K[S]$ are Bézout coefficients for χ_1, χ_2 , i.e., they satisfy $a_1 \chi_1 + a_2 \chi_2 = g$. Notice that XCRT is in fact a K -algebra isomorphism between $K[S]/\text{LCM}(\chi_1(S), \chi_2(S))$ and the subalgebra of $K[S]/\chi_1(S) \times K[S]/\chi_2(S)$ formed by pairs (u_1, u_2) such that $u_1 \equiv u_2 \pmod{\text{GCD}(\chi_1, \chi_2)}$.

Proposition 5.13. *Algorithm 4 (ADDDIVISORS) is correct: If it does not fail, then it returns a primitive element representation of the smooth effective divisor $D_1 + D_2$.*

Proof. Let I_1, I_2, J denote the three following ideals of $K[\mathcal{C}^0]$, using the notation in Algorithm 4:

$$\begin{aligned} I_1 &= \langle \chi_1(\lambda_1 X + Y), X - u_1(\lambda_1 X + Y), Y - v_1(\lambda_1 X + Y) \rangle, \\ I_2 &= \langle \chi_2(\lambda_2 X + Y), X - u_2(\lambda_2 X + Y), Y - v_2(\lambda_2 X + Y) \rangle, \\ J &= \langle \hat{\chi}(\hat{\lambda} X + Y), X - \hat{u}(\hat{\lambda} X + Y), Y - \hat{v}(\hat{\lambda} X + Y) \rangle. \end{aligned}$$

Proving that Algorithm 4 is correct amounts to showing that $I_1 \cdot I_2 = J$, and that **(Div-H1)** to **(Div-H3)** are satisfied by $\hat{\lambda}, \hat{\chi}, \hat{u}, \hat{v}$.

We start by proving that $\hat{\lambda}, \hat{\chi}, \hat{u}, \hat{v}$ satisfy **(Div-H1)** to **(Div-H3)**. For **(Div-H1)** and **(Div-H2)**, Proposition 5.11 ensures that $q(\hat{u}_i(S), \hat{v}_i(S)) \equiv 0 \pmod{\hat{\chi}_i(S)}$ for $i \in \{1, 2\}$. Using the fact that the XCRT is a morphism, we get that $q(\hat{u}_{12}(S), \hat{v}_{12}(S)) \equiv 0 \pmod{\text{LCM}(\hat{\chi}_1(S), \hat{\chi}_2(S))}$ and $\hat{\lambda}\hat{u}_{12}(S) + \hat{v}_{12}(S) \equiv S \pmod{\text{LCM}(\hat{\chi}_1(S), \hat{\chi}_2(S))}$. Next, Proposition 5.12 proves the equalities $q(\hat{u}(S), \hat{v}(S)) \equiv 0 \pmod{\text{LCM}(\hat{\chi}_1(S), \hat{\chi}_2(S))^2}$ and $\hat{\lambda}\hat{u}(S) + \hat{v}(S) \equiv S \pmod{\text{LCM}(\hat{\chi}_1(S), \hat{\chi}_2(S))^2}$. Since $\hat{\chi} = \hat{\chi}_1 \cdot \hat{\chi}_2$ divides $\text{LCM}(\hat{\chi}_1(S), \hat{\chi}_2(S))^2$, we get that $q(\hat{u}(S), \hat{v}(S)) \equiv 0 \pmod{\hat{\chi}}$ and $\hat{\lambda}\hat{u}(S) + \hat{v}(S) \equiv S \pmod{\hat{\chi}}$. For **(Div-H3)**, we observe that the fact that the XCRT is a ring morphism implies that $\frac{\partial q}{\partial X}(\hat{u}_{12}(S), \hat{v}_{12}(S)) - \lambda \frac{\partial q}{\partial Y}(\hat{u}_{12}(S), \hat{v}_{12}(S))$ is invertible in $K[S]/\text{LCM}(\hat{\chi}_1(S), \hat{\chi}_2(S))$. Consequently, $\frac{\partial q}{\partial X}(\hat{u}(S), \hat{v}(S)) - \lambda \frac{\partial q}{\partial Y}(\hat{u}(S), \hat{v}(S))$ is invertible in $K[S]/\text{LCM}(\hat{\chi}_1(S), \hat{\chi}_2(S))$, and hence it is also invertible in $K[S]/\hat{\chi}(S)$.

Next, let $I'_1, I'_2 \subset K[\mathcal{C}^0]$ be the ideals

$$\begin{aligned} I'_1 &= \langle \hat{\chi}_1(\hat{\lambda} X + Y), X - \hat{u}_1(\hat{\lambda} X + Y), Y - \hat{v}_1(\hat{\lambda} X + Y) \rangle, \\ I'_2 &= \langle \hat{\chi}_2(\hat{\lambda} X + Y), X - \hat{u}_2(\hat{\lambda} X + Y), Y - \hat{v}_2(\hat{\lambda} X + Y) \rangle. \end{aligned}$$

By Proposition 5.11 and Lemma 5.10, the equalities $I_1 = I'_1$ and $I_2 = I'_2$ hold.

Therefore, proving that $I_1 \cdot I_2 = J$ is achieved if we prove that $I'_1 \cdot I'_2 = J$ instead. Using the factorization as a product of maximal ideals given by Lemma 5.5, it is sufficient to prove that a power $\mathfrak{m}^\ell \subset K[\mathcal{C}^0]$ of a maximal ideal contains $I'_1 \cdot I'_2$ if and only if it contains J . Notice that the powers of maximal ideals which contain I'_1 are of the form $\langle \chi_{\mathfrak{m}}(\hat{\lambda} X + Y)^\ell, X - u_{1, \mathfrak{m}^\ell}(\hat{\lambda} X + Y), Y - v_{1, \mathfrak{m}^\ell}(\hat{\lambda} X + Y) \rangle$ where $\chi_{\mathfrak{m}}$ is a prime polynomial such that $\chi_{\mathfrak{m}}^\ell$ divides $\hat{\chi}_1$, $u_{1, \mathfrak{m}^\ell}(S) \equiv \hat{u}_1(S) \pmod{\chi_{\mathfrak{m}}(S)^\ell}$, and $v_{1, \mathfrak{m}^\ell}(S) \equiv \hat{v}_1(S) \pmod{\chi_{\mathfrak{m}}(S)^\ell}$. The same holds for I'_2 .

Let \mathfrak{m}^ℓ be a power of a maximal ideal which contains $I'_1 \cdot I'_2$. Using the unicity of the factorization in Lemma 5.5, the powers of maximal ideals which contain $I'_1 \cdot I'_2$ are those $\mathfrak{m}^{\ell_1 + \ell_2}$ where $I'_1 \subset \mathfrak{m}^{\ell_1}$ and $I'_2 \subset \mathfrak{m}^{\ell_2}$. This means that \mathfrak{m}^ℓ has the form

$$\mathfrak{m}^\ell = \langle \chi_{\mathfrak{m}}(\hat{\lambda} X + Y)^{\ell_1 + \ell_2}, X - u_{12}(\hat{\lambda} X + Y), Y - v_{12}(\hat{\lambda} X + Y) \rangle,$$

where u_{12} (resp. v_{12}) is any polynomial such that $u_{12}(S) \equiv u_{1, \mathfrak{m}^{\ell_1}}(S) \pmod{\chi_{\mathfrak{m}}(S)^{\ell_1}}$, $u_{12}(S) \equiv u_{2, \mathfrak{m}^{\ell_2}}(S) \pmod{\chi_{\mathfrak{m}}(S)^{\ell_2}}$ (resp. $v_{12}(S) \equiv v_{1, \mathfrak{m}^{\ell_1}}(S) \pmod{\chi_{\mathfrak{m}}(S)^{\ell_1}}$, $v_{12}(S) \equiv v_{2, \mathfrak{m}^{\ell_2}}(S) \pmod{\chi_{\mathfrak{m}}(S)^{\ell_2}}$). Then we notice that $\hat{\chi} = \hat{\chi}_1 \cdot \hat{\chi}_2$, and therefore $\chi_{\mathfrak{m}}(S)^{\ell_1 + \ell_2}$ divides $\hat{\chi}(S)$. By using the properties of XCRT and of Hensel's lifting, we get that

$$\begin{aligned} \hat{u}(S) &\equiv \hat{u}_1(S) \pmod{\hat{\chi}_1(S)}; \\ \hat{u}(S) &\equiv \hat{u}_2(S) \pmod{\hat{\chi}_2(S)}; \\ \hat{v}(S) &\equiv \hat{v}_1(S) \pmod{\hat{\chi}_1(S)}; \\ \hat{v}(S) &\equiv \hat{v}_2(S) \pmod{\hat{\chi}_2(S)}. \end{aligned}$$

This implies that $\mathfrak{m}^\ell = \langle \chi_{\mathfrak{m}}(\hat{\lambda} X + Y)^{\ell_1 + \ell_2}, X - \hat{u}(\hat{\lambda} X + Y), Y - \hat{v}(\hat{\lambda} X + Y) \rangle$, and hence \mathfrak{m}^ℓ contains J . The proof that any power of maximal ideal which contains J also contains $I'_1 \cdot I'_2$ is similar. \square

Function SUBTRACTDIVISORS;

Data: Two smooth effective divisors given by primitive element representations:

$$D_1 = (\lambda_1, \chi_1, u_1, v_1),$$

$$D_2 = (\lambda_2, \chi_2, u_2, v_2).$$

Result: A primitive element representation of the smooth effective divisor $[D_1 - D_2]_+$ or “fail”.

$\hat{\lambda} \leftarrow \text{RANDOM}(K);$

$(\hat{\chi}_1, \hat{u}_1, \hat{v}_1) \leftarrow \text{CHANGEPRIMELT}(\hat{\lambda}, \lambda_1, \chi_1, u_1, v_1);$

$(\hat{\chi}_2, \hat{u}_2, \hat{v}_2) \leftarrow \text{CHANGEPRIMELT}(\hat{\lambda}, \lambda_2, \chi_2, u_2, v_2);$

if $\hat{u}_1 \not\equiv \hat{u}_2 \pmod{\text{GCD}(\hat{\chi}_1, \hat{\chi}_2)}$ **then**

 Return “fail”

end

$\hat{\chi} \leftarrow \hat{\chi}_1 / \text{GCD}(\hat{\chi}_1, \hat{\chi}_2);$

$\hat{u}(S) \leftarrow \hat{u}_1(S) \pmod{\hat{\chi}(S)};$

$\hat{v}(S) \leftarrow \hat{v}_1(S) \pmod{\hat{\chi}(S)};$

Return $(\hat{\lambda}, \hat{\chi}, \hat{u}, \hat{v}).$

Algorithm 5: Computing the subtraction of smooth effective divisors.

Algorithm 5 (SUBTRACTDIVISORS) provides a method for subtracting effective divisors given by primitive element representations. We emphasize that the divisor returned is the subtraction $D_1 - D_2$ only if the result is also effective, *i.e.* if $D_1 \geq D_2$. If this is not the case, then it returns the positive part of the subtraction. This is not restrictive since our main algorithm will always use SUBTRACTDIVISORS on divisors D_1, D_2 satisfying $D_1 \geq D_2$.

Proposition 5.14. *Algorithm 5 (SUBTRACTDIVISORS) is correct: If it does not fail, then it returns a primitive element representation of the smooth effective divisor $[D_1 - D_2]_+$, where the notation $[D]_+$ denotes the positive part of the divisor D , *i.e.* the smallest effective divisor D' such that $D' \geq D$.*

Proof. Let I_1, I_2, J denote the three following ideals of $K[C^0]$, using the notation in Algorithm 5:

$$\begin{aligned} I_1 &= \langle \chi_1(\lambda_1 X + Y), \quad X - u_1(\lambda_1 X + Y), \quad Y - v_1(\lambda_1 X + Y) \rangle; \\ I_2 &= \langle \chi_2(\lambda_2 X + Y), \quad X - u_2(\lambda_2 X + Y), \quad Y - v_2(\lambda_2 X + Y) \rangle; \\ J &= \langle \hat{\chi}(\hat{\lambda} X + Y), \quad X - \hat{u}(\hat{\lambda} X + Y), \quad Y - \hat{v}(\hat{\lambda} X + Y) \rangle. \end{aligned}$$

The effective divisor $[D_1 - D_2]_+$ corresponds to the colon ideal $I_1 : I_2 = \{f \in K[C^0] \mid f \cdot I_2 \subset I_1\}$. Consequently, we must prove that $(\hat{\lambda}, \hat{\chi}, \hat{u}, \hat{v})$ satisfies **(Div-H1)** to **(Div-H3)** and that $J = I_1 : I_2$. The equalities **(Div-H1)** to **(Div-H3)** for $\hat{\lambda}, \hat{\chi}_1, \hat{u}_1, \hat{v}_1$ are satisfied by Proposition 5.11. Regarding them modulo $\hat{\chi}$ shows that $(\hat{\lambda}, \hat{\chi}, \hat{u}, \hat{v})$ satisfies **(Div-H1)** to **(Div-H3)**.

In order to prove that $J = I_1 : I_2$, we proceed as in the proof of Proposition 5.13, by noticing first that I_1 and I_2 can be rewritten as

$$\begin{aligned} I_1 &= \langle \hat{\chi}_1(\hat{\lambda} X + Y), \quad X - \hat{u}_1(\hat{\lambda} X + Y), \quad Y - \hat{v}_1(\hat{\lambda} X + Y) \rangle; \\ I_2 &= \langle \hat{\chi}_2(\hat{\lambda} X + Y), \quad X - \hat{u}_2(\hat{\lambda} X + Y), \quad Y - \hat{v}_2(\hat{\lambda} X + Y) \rangle. \end{aligned}$$

Using [AM69, Prop. 9.1] together with the fact that D_1 does not involve any singular point of the curve by **(Div-H3)**, the equality $I_1 : I_2 = J$ holds if and only if the powers of maximal ideals $\mathfrak{m}^\ell \subset K[C^0]$ which contain $I_1 : I_2$ are exactly those which contain J . Equivalently, this means that if \mathfrak{m}^{ℓ_1} is the largest power of \mathfrak{m} which contains I_1 and if \mathfrak{m}^{ℓ_2} is the largest power of \mathfrak{m} which contains I_2 , then $\mathfrak{m}^{\max(\ell_1 - \ell_2, 0)}$ is the largest power of \mathfrak{m} which contains J . As in the proof of Proposition 5.13, the maximal ideals $\mathfrak{m} \subset K[C^0]$ which contain I_1 have the form $\langle \chi_{\mathfrak{m}}(\hat{\lambda} X + Y), X - u_{\mathfrak{m}}(\hat{\lambda} X + Y), Y - v_{\mathfrak{m}}(\hat{\lambda} X + Y) \rangle$, where $u_{\mathfrak{m}} \equiv \hat{u}_1 \pmod{\chi_{\mathfrak{m}}}$, $v_{\mathfrak{m}} \equiv \hat{v}_1 \pmod{\chi_{\mathfrak{m}}}$. The

proof is concluded by noticing that for any prime factor Φ of $\hat{\chi}_1$, if Φ^{ℓ_1} is the largest power of Φ which divides $\hat{\chi}_1$ and Φ^{ℓ_2} is the largest power of Φ which divides $\hat{\chi}_2$, then the largest power Φ which divides $\hat{\chi} = \hat{\chi}_1 / \text{GCD}(\hat{\chi}_1, \hat{\chi}_2)$ is $\Phi^{\max(\ell_1 - \ell_2, 0)}$. \square

5.3 Subroutines of the main algorithm

Section 5.2.3 covers the presentation and proofs of correctness of two of the subroutines used in Algorithm 1 (RIEMANNROCHBASIS). This section addresses the same issues for the subroutines INTERPOLATE, COMPPRINC DIV and NUMERATORBASIS so that at the end of this section our main algorithm will be entirely described and proven.

5.3.1 Interpolation subroutine

This section focuses on the following interpolation problem: Given a smooth effective divisor D and the nodal divisor E , find a element $h \in K[\mathcal{C}^0]$ such that its associated principal divisor (h) satisfies $(h) \geq D + E$. Roughly speaking, the goal is to build a polynomial with prescribed zeros: It must vanish at all the points of the nodal divisor and all the points of D with correct multiplicities.

Function INTERPOLATE;

Data: The degree δ of the curve, a smooth effective divisor given by a primitive element representation (λ, χ, u, v) , and the nodal divisor given by $(\lambda_E, \chi_E, u_E, v_E, T_E)$.

Result: A polynomial $h \in K[X, Y]$ representing a form in $K[\mathcal{C}^0]$ such that $(h) \geq D + E$.

$t \leftarrow \deg(\chi) + \deg(\chi_E)$

if $\binom{\delta + 1}{2} \leq t$ **then**

$d \leftarrow \lfloor t/\delta + (\delta - 1)/2 \rfloor$

else

$d \leftarrow \lfloor (\sqrt{1 + 8t} - 1)/2 \rfloor$

end

Construct the matrix representing the linear map

$\varphi : \{f \in K[X, Y] \mid \deg(f) \leq d, \deg_Y(f) < \delta\} \rightarrow K[S]_{<\deg(\chi)} \times K[S]_{<\deg(\chi_E)}$ defined as
 $\varphi(f(X, Y)) = (f(u(S), v(S)) \bmod \chi(S), f(u_E(S), v_E(S)) \bmod \chi_E(S));$

Compute a basis $\mathbf{b}_1, \dots, \mathbf{b}_\ell$ of the kernel of φ ;

$(\mu_1, \dots, \mu_\ell) \leftarrow \text{RANDOM}(K^\ell \setminus \{\mathbf{0}\});$

Return $h = \sum_{i=1}^\ell \mu_i \mathbf{b}_i$.

Algorithm 6: Computing a function $h \in K[\mathcal{C}^0]$ of small degree such that $(h) \geq D + E$.

Proposition 5.15. *Algorithm 6 (INTERPOLATE) is correct: The kernel of φ has positive dimension, and its nonzero elements h satisfy $(h) \geq D + E$.*

Proof. The fact that the kernel φ has positive dimension follows from a dimension count, which is postponed to Lemma 5.16. We now prove the second part of the proposition. First, notice that $\deg_Y(h) < \deg_Y(q)$ for any nonzero h in the kernel of φ , hence h cannot be a multiple of q , which implies that $\langle 0 \rangle \subsetneq \langle h \rangle \subset K[\mathcal{C}^0]$. Next, by Lemmas 5.9 and 5.10, the ideal $I_{D+E} = \{f \in K[\mathcal{C}^0] \mid (f) \geq D + E\} = \{f \in K[\mathcal{C}^0] \mid (f) \geq D\} \cap \{f \in K[\mathcal{C}^0] \mid (f) \geq E\}$ equals

$$\langle \chi(\lambda X + Y), X - u(\lambda X + Y), Y - v(\lambda X + Y) \rangle \cap \langle \chi_E(\lambda_E X + Y), X - u_E(\lambda_E X + Y), Y - v_E(\lambda X + Y) \rangle.$$

By construction, $h(u(S), v(S)) \equiv 0 \bmod \chi(S)$ and $h(u_E(S), v_E(S)) \equiv 0 \bmod \chi_E(S)$ for any $h \in \ker \varphi$. The proof is concluded by noticing that the polynomials f in I_{D+E} are exactly those which satisfy $f(u(S), v(S)) \equiv 0 \bmod \chi(S)$ and $f(u_E(S), v_E(S)) \equiv 0 \bmod \chi_E(S)$, using the isomorphisms in Propositions 5.3 and 5.4. \square

The following lemma ensures that Algorithm 6 actually returns a nonzero element, *i.e.*, that the kernel of φ has positive dimension.

Lemma 5.16. *With the notations in Algorithm 6,*

$$t = \deg(\chi) + \deg(\chi_E) < \dim_K(\{f \in k[X, Y] \mid \deg(f) \leq d, \deg_Y(f) < \delta\}) \leq 3t.$$

In particular, the first inequality ensures that φ is not injective.

Proof. First, a direct dimension count gives

$$\dim_K(\{f \in K[X, Y] \mid \deg(f) \leq d, \deg_Y(f) < \delta\}) = \begin{cases} \delta(d - (\delta - 3)/2) & \text{if } d \geq \delta \\ \binom{d+2}{2} & \text{otherwise.} \end{cases}$$

On one hand, if $\binom{\delta+1}{2} \leq t$, then

$$\begin{aligned} d &= \lfloor t/\delta + (\delta - 1)/2 \rfloor \\ &\geq \left\lfloor \binom{\delta+1}{2}/\delta + (\delta - 1)/2 \right\rfloor \\ &\geq \delta, \end{aligned}$$

and hence

$$\begin{aligned} \delta(d - (\delta - 3)/2) &> \delta(t/\delta + (\delta - 1)/2 - 1 - (\delta - 3)/2) \\ &= t \\ \delta(d - (\delta - 3)/2) &\leq \delta(t/\delta + (\delta - 1)/2 - (\delta - 3)/2) \\ &\leq t + \delta \\ &\leq t + \binom{\delta+1}{2} \\ &\leq 2t. \end{aligned}$$

On the other hand, if $\binom{\delta+1}{2} > t$, then

$$\begin{aligned} d &= \lfloor (\sqrt{1+8t} - 1)/2 \rfloor \\ &< \lfloor (\sqrt{1+4\delta(\delta+1)} - 1)/2 \rfloor \\ &= \lfloor (\sqrt{(2\delta+1)^2 - 1})/2 \rfloor \\ &= \delta \end{aligned}$$

Since $\binom{x+2}{2} - t > 0$ for any $x > (\sqrt{1+8t} - 3)/2$, we get that $t < \binom{d+2}{2}$ as expected. Finally, the last inequality follows from

$$\binom{\lfloor (\sqrt{1+8t} - 1)/2 \rfloor + 2}{2} \leq t + (1 + \sqrt{1+8t})/2,$$

and direct computations show that $(1 + \sqrt{1+8t})/2 \leq 2t$ since $t \geq 1$. \square

5.3.2 Computing the smooth part of the principal divisor associated to a regular function on the curve

This section is devoted to the following problem: Given a polynomial $h \in k[\mathcal{C}^0]$ such that $(h) = D_h + E$ where D_h is a smooth divisor on the curve, compute a primitive element representation of D_h .

Let us mention that it may happen that h vanishes at infinity. Therefore, the support of D_h may contain points at infinity, but the primitive element representation only represents points in the affine chart $Z \neq 0$. Ignoring these zeros at infinity may lead to functions having unauthorized poles at infinity in the basis returned by Algorithm 1. As we already mentioned in Section 5.2.1, handling what happens at infinity is not a problem: This issue can be solved for instance by doing

the computations in three affine spaces which cover \mathbb{P}^2 , which would multiply the complexity by a constant factor. Notice also that it is easy to detect if h has zeroes at infinity: This happens if and only if the degree of the resultant of h and q is strictly less than $\deg(h) \deg(\mathcal{C})$, thanks to the fact that we assumed that \mathcal{C} is in projective Noether position. For simplicity, we will not discuss further this issue in the sequel of this chapter.

The central element of Algorithm COMPPRINC DIV is the computation of a resultant and of the associated first subresultant (as defined for instance in [EK03, Sec. 3]). However, a number of extra steps are required to ensure that this computation satisfies genericity assumptions and returns a correct result. First, a random direction of projection λ is selected for computing the resultant. This direction of projection must satisfy some conditions. In particular, distinct point in the support of h must project on distinct points. Also, in order to exploit the Poisson formula for the resultant, we ask that this direction is not a tangent at any node of the curve, see Lemma 5.18. This condition about the tangents at the node is tested via the evaluation of the univariate polynomial T_E . We also need a representation of the nodal divisor with respect to this λ . This is achieved by using a slightly modified version of Algorithm CHANGEPRIME LT where the first test — which is not relevant for the nodal divisor — is removed. Finally, Algorithm COMPPRINC DIV must clean out the singular points: this is done by noticing that the roots of the resultant which correspond to the singular points appear with multiplicity at least 2, see Lemma 5.18 below. Therefore these singular points are removed by dividing out by the square of the univariate polynomial $\hat{\chi}_E$ whose roots parametrize the coordinates of the singular points.

Proposition 5.17. *Algorithm 7 (COMPPRINC DIV) is correct: If it does not fail, then it returns a primitive element representation of the smooth part of the principal divisor (h).*

Before proving Proposition 5.17, we need the following technical lemma, which implies in particular that nodes appear as roots of the resultant with multiplicity at least two.

Lemma 5.18. *With the notations in Algorithm 7, let $s \in \bar{K}$ and $\lambda \in K \setminus \{0\}$. Let R_1, \dots, R_ℓ be the valuation rings in $\text{Frac}(\bar{K}[X, Y]/q)$ associated to the points of $\tilde{\mathcal{C}}$ above s , i.e., the points on \mathcal{C} which project to s via the projection $(X, Y) \mapsto \lambda X + Y$. Assume that the vector $(1, -\lambda)$ is not tangent to \mathcal{C}^0 at any of these points and that the coefficient of $Y^{\deg(\mathcal{C})}$ in $q((S - Y)/\lambda, Y)$ is nonzero. Let $m_1, \dots, m_{\deg(\mathcal{C})}$ denote the valuations of h in these valuation rings. Then s is a root of multiplicity $\sum_{i=1}^{\ell} m_i$ in $\text{Resultant}_Y(q((S - Y)/\lambda, Y), h((S - Y)/\lambda, Y))$.*

Proof. Since we assumed that the curve is nodal, that the coefficient of $Y^{\deg(\mathcal{C})}$ in $q((S - Y)/\lambda, Y)$ is nonzero, and that the vector $(1, -\lambda)$ is not tangent at any point above s , we get that the polynomial $q((S - Y)/\lambda, Y)$ splits over the ring $\bar{K}[[S - s]]$ of power series at s as a product of $\deg(\mathcal{C})$ factors, see e.g. [NRS17] and references therein. Notice that this factorization property holds even if some of the points above s are nodes. Let $\tilde{y}_1, \dots, \tilde{y}_{\deg(\mathcal{C})}$ denote its roots in $\bar{K}[[S - s]]$. Using the multiplicativity property of the resultant [Jou91, Sec. 5.7], we get

$$\text{Resultant}_Y(q((S - Y)/\lambda, Y), h((S - Y)/\lambda, Y)) = \alpha \prod_{i=1}^{\ell} h((S - \tilde{y}_i)/\lambda, \tilde{y}_i),$$

where $\alpha \in K$. The proof is concluded by noticing that $S - s$ is a uniformizing element for all the discrete valuation rings since the vector $(1, -\lambda)$ is not tangent to the curve at any of the points above s , so that m_i precisely corresponds to the largest integer γ such that $(S - s)^\gamma$ divides $h((s - \tilde{y}_i)/\lambda, \tilde{y}_i)$. \square

Proof of Proposition 5.17. In order to prove Proposition 5.17, we must prove that the output (λ, χ, u, v) satisfies **(Div-H1)** to **(Div-H3)** and that the two ideals $\langle h \rangle : I_E^\infty \subset K[\mathcal{C}^0]$ and $\langle \chi(\lambda X + Y), X - u(\lambda X + Y), Y - v(\lambda X + Y) \rangle \subset K[\mathcal{C}^0]$ are equal, where I_E is the radical ideal of $K[\mathcal{C}^0]$ which encodes the algebraic set of the nodes. Equality **(Div-H2)** follows directly from the definitions of $u(S)$ and $v(S)$ in Algorithm 7. To prove **(Div-H1)**, we shall prove that the equality holds modulo $(S - s)^\gamma$ for any root $s \in \bar{K}$ of χ of multiplicity γ . A classical property of the subresultants is

```

Function COMPPRINC DIV;
Data: A squarefree bivariate  $q \in K[X, Y]$  such that  $\deg(q) = \deg_Y(q)$ , a bivariate
        polynomial  $h \in K[X, Y]$ , and a representation  $(\lambda_E, \chi_E, u_E, v_E, T_E)$  of the nodal
        divisor.
Result: A primitive element representation  $(\lambda, \chi(S), u(S), v(S))$  of the smooth part of
        the principal effective divisor  $(h)$  or “fail”.
 $\lambda \leftarrow \text{RANDOM}(K)$ ;
if  $\lambda = 0$  or if the coefficient of  $Y^{\deg(q)}$  in  $q((S - Y)/\lambda, Y) \in K[S][Y]$  is 0 then
    | Return “fail”
end
if  $T_E(\lambda) = 0$  then
    | Return “fail”;
    | /* Ensures that  $(1, -\lambda)$  is not tangent to the curve at any node. */
end
 $(\hat{\chi}_E, \hat{u}_E, \hat{v}_E) \leftarrow \text{CHANGEPRIMELTNODAL}(\lambda, \lambda_E, \chi_E, u_E, v_E)$ ;
/* CHANGEPRIMELTNODAL is the same algorithm as CHANGEPRIMELT, but we skip
   the first test (which would fail on the nodal divisor). */
 $\tilde{\chi}(S) \leftarrow \text{Resultant}_Y(q((S - Y)/\lambda, Y), h((S - Y)/\lambda, Y))$ ;
 $a_0(S) + Y a_1(S) \leftarrow \text{FirstSubRes}_Y(q((S - Y)/\lambda, Y), h((S - Y)/\lambda, Y))$ ;
 $\chi \leftarrow \tilde{\chi} / \hat{\chi}_E^2$ ;
if  $\text{GCD}(\chi, \hat{\chi}_E) \neq 1$  then
    | Return “fail”
end
if  $\text{GCD}(a_1(S), \chi(S)) \neq 1$  then
    | Return “fail”
end
 $v(S) \leftarrow -a_0(S) \cdot a_1(S)^{-1} \bmod \chi(S)$ ;
 $u(S) \leftarrow (S - v(S)) / \lambda$ ;
if  $\text{GCD}(\frac{\partial q}{\partial X}(u(S), v(S)) - \lambda \frac{\partial q}{\partial Y}(u(S), v(S)), \chi(S)) \neq 1$  then
    | Return “fail”
end
Return  $(\lambda, \chi(S), u(S), v(S))$ .

```

Algorithm 7: Computing a primitive element representation of the smooth part of (h) .

that they belong to the ideal generated by the input polynomials. This implies that for any root $s \in \bar{K}$ of $\tilde{\chi}$ we have

$$a_0(S) + Y a_1(S) \in \langle q((S - Y)/\lambda, Y), h((S - Y)/\lambda, Y) \rangle \subset \bar{K}[[S - s]][Y].$$

If the algorithm does not fail, then $a_1(S)$ is invertible modulo $\chi(S)$. Consequently, it is also invertible in $\bar{K}[[S - s]]$ for any root $s \in \bar{K}$ of χ and hence

$$Y + a_0(S)a_1(S)^{-1} \in \langle q((S - Y)/\lambda, Y), h((S - Y)/\lambda, Y) \rangle \subset \bar{K}[[S - s]][Y].$$

Therefore, the GCD of $q((S - Y)/\lambda, Y)$ and $h((S - Y)/\lambda, Y)$ in $\text{Frac}(\bar{K}[[S - s]][Y])$ divides $Y + a_0(S)a_1(S)^{-1}$. But we also know that this GCD is nonconstant, since s is a root of the resultant $\tilde{\chi}$. By a degree argument, this GCD equals $Y + a_0(S)a_1(S)^{-1}$ and hence $q((S + a_0(S)a_1(S)^{-1})/\lambda, -a_0(S)a_1(S)^{-1}) = 0$ in $\bar{K}[[S - s]]$. Considering this equation modulo $(S - s)^\gamma$ and using the CRT over all the roots of χ finishes the proof of **(Div-H1)**. Finally, **(Div-H3)** is explicitly tested and hence it must be satisfied if the algorithm does not fail.

It remains to prove the equality of the ideals $\langle h \rangle : I_E^\infty \subset K[\mathcal{C}^0]$ and $\langle \chi(\lambda X + Y), X - u(\lambda X + Y), Y - v(\lambda X + Y) \rangle \subset K[\mathcal{C}^0]$. Using the isomorphism between $K[X, Y]/\langle \chi(\lambda X + Y), X - u(\lambda X + Y), Y - v(\lambda X + Y) \rangle$ and $K[S]/\chi(S)$ (see Proposition 5.3), the elements in $\langle \chi(\lambda X + Y), X - u(\lambda X + Y), Y - v(\lambda X + Y) \rangle$ are precisely the classes of the bivariate polynomials $\psi(X, Y) \in K[X, Y]$ such that $\psi(u(S), v(S)) \equiv 0 \pmod{\chi(S)}$. Using a proof identical to that of **(Div-H1)** we get that $h(u(S), v(S)) \equiv 0 \pmod{\chi(S)}$ which proves that $\langle h \rangle \subset \langle \chi(\lambda X + Y), X - u(\lambda X + Y), Y - v(\lambda X + Y) \rangle$. Saturating on both sides, we get that $\langle h \rangle : I_E^\infty \subset \langle \chi(\lambda X + Y), X - u(\lambda X + Y), Y - v(\lambda X + Y) \rangle : I_E^\infty = \langle \chi(\lambda X + Y), X - u(\lambda X + Y), Y - v(\lambda X + Y) \rangle$, where the last equality comes from the fact that $\text{GCD}(\chi, \hat{\chi}_E) = 1$.

For the other inclusion, we use [AM69, Prop. 9.1], which implies that $\langle \chi(\lambda X + Y), X - u(\lambda X + Y), Y - v(\lambda X + Y) \rangle \subset \langle h \rangle : I_E^\infty$ if this inclusion holds in the local ring associated to any maximal ideal $\mathfrak{m} \subset \bar{k}[\mathcal{C}^0]$ which contains $\langle \chi(\lambda X + Y), X - u(\lambda X + Y), Y - v(\lambda X + Y) \rangle$. Over \bar{K} , these maximal ideals have the form $\langle \lambda X + Y - s, X - u(s), Y - v(s) \rangle$, where $s \in \bar{K}$ is a root of χ . The assumption $\text{GCD}(\frac{\partial q}{\partial X}(u(S), v(S)) - \lambda \frac{\partial q}{\partial Y}(u(S), v(S)), \chi(S)) = 1$ ensures that all these maximal ideals correspond to nonsingular points, and hence the associated local rings are discrete valuation rings. For $s \in \bar{K}$ a root of χ , let $y_1, \dots, y_{\deg(\mathcal{C})}$ be the roots of the univariate polynomial $q((s - Y)/\lambda, Y) \in \bar{k}[Y]$. Let m_i denote the intersection multiplicity of h at the point $((s - y_i)/\lambda, y_i)$ of \mathcal{C}^0 . Since $\text{GCD}(\frac{\partial q}{\partial X}(u(S), v(S)) - \lambda \frac{\partial q}{\partial Y}(u(S), v(S)), \chi(S)) \neq 1$, we obtain that the vector $(1, -\lambda)$ is not tangent to \mathcal{C}^0 any of these points. Lemma 5.18 then gives that $m_1 + \dots + m_{\deg(\mathcal{C})} = \alpha$, where α is the multiplicity of the root s in χ . Let k be the integer such that $y_k = v(s)$. Then $m_k \leq \alpha$, which shows that we have $\langle \chi(\lambda X + Y), X - u(\lambda X + Y), Y - v(\lambda X + Y) \rangle \subset \langle h \rangle : I_E^\infty$ in the local ring at the point $(u(s), v(s))$. The statement [AM69, Prop. 9.1] concludes the proof of the inclusion $\langle \chi(\lambda X + Y), X - u(\lambda X + Y), Y - v(\lambda X + Y) \rangle \subset \langle h \rangle : I_E^\infty$. \square

5.3.3 Computing the linear space of regular functions of bounded degree having prescribed zeros

The task accomplished by Algorithm **NUMERATORBASIS** is similar to what Algorithm **INTERPOLATE** does: It computes a basis of the vector space of regular functions having prescribed zeros. The only difference with Algorithm **INTERPOLATE** is that Algorithm **NUMERATORBASIS** returns a basis of this linear space.

Proposition 5.19. *Algorithm 8 (**NUMERATORBASIS**) is correct: The nonzero elements g in the kernel of φ are not divisible by q and they satisfy $(g) \geq D + E$.*

Proof. The proof is similar to that of Proposition 5.15. \square

Function NUMERATORBASIS;

Data: A positive integer δ , a smooth effective divisor given by a primitive element representation $(\lambda, \chi(S), u(S), v(S))$, a positive integer d , and the nodal divisor given by $(\lambda_E, \chi_E, u_E, v_E, T_E)$.

Result: A basis of the space of polynomials $g \in K[X, Y]$ such that $\deg(g) \leq d, \deg_Y(g) < \delta$ and the associated divisor satisfies $(g) \geq D + E$.

Construct the matrix representing the linear map

$\varphi : \{f \in K[X, Y] \mid \deg(f) \leq d, \deg_Y(f) \leq \delta\} \rightarrow K[S]_{<\deg(\chi)} \times K[S]_{<\deg(\chi_E)}$ defined as $\varphi(f(X, Y)) = (f(u(S), v(S)) \bmod \chi(S), f(u_E(S), v_E(S)) \bmod \chi_E(S))$;

Compute and return a basis of the kernel of φ .

Algorithm 8: Computing a basis of the vector space of regular functions $g \in K[\mathcal{C}^0]$ of bounded degree such that $(g) \geq D + E$.

5.4 Analyses of the algorithm

This section explores the properties of the main algorithm besides correctness : complexity analysis, analysis of the failure probability. Experimental results obtained with our NTL/C++ implementation are also presented here.

5.4.1 Asymptotic complexity

This section is devoted to the complexity analysis of the main algorithm. It rests on the analysis of all the subalgorithms we have introduced so far in sections 5.2.3 and 5.3.

All complexity bounds count the number of arithmetic operations (additions, subtractions, multiplications, divisions) in K , all at unit cost. We do not include in our complexity bounds the cost of generating random elements, nor the cost of monomial manipulations, nor multiplications by fixed integer constants. In particular, we do not include in our complexity bounds the cost of computing the partial derivatives of a polynomial. We use the classical $O()$ and $\tilde{O}()$ notation, see e.g. [vzGG13, Sec. 25.7]. The notation $M(n)$ stands for the number of arithmetic operations required in K to compute the product of two univariate polynomials of degree n with coefficients in K . By [CK91], $M(n) = O(n \log n \log \log n)$. In the sequel, ω is a feasible exponent for matrix multiplication, *i.e.*, ω is such that there is an algorithm for multiplying two $n \times n$ matrices with entries in K within $O(n^\omega)$ arithmetic operations in K . The best known bound is $\omega < 2.3729$ [LG14]. In the following, we make the assumption that $\omega > 2$.

We begin by studying the subalgorithms used to perform arithmetic on divisors, then the algorithms for sum and subtraction of divisors and finally the algorithms described in Section 5.3.

Proposition 5.20. *Algorithm 2 (CHANGEPRIME) requires at most $O(\deg(\chi)^\omega)$ arithmetic operations in K .*

Proof. In order to construct the matrix M in Algorithm 2, we must compute the remainders $S^i \cdot (\tilde{\lambda}u(S) + v(S)) \bmod \chi(S)$ for $i \in \{0, \dots, \deg(\chi) - 1\}$. Each of these computations costs $O(M(\deg(\chi)))$ arithmetic operations, so the total cost of constructing the matrix M is bounded by $O(\deg(\chi) M(\deg(\chi)))$, which is bounded above by $O(\deg(\chi)^\omega)$. Computing the characteristic polynomial of M can be done within $O(\deg(\chi)^\omega)$ arithmetic operations [PS07].

We emphasize that in [PS07], it is assumed that the cardinality of K is at least $2 \deg(\chi)^2$, so that the probability of failure is bounded by $1/2$. In fact, using the same algorithm and the same proof as in [PS07], the assumption on the cardinality of K can be removed but the probability of failure will then only be bounded by $\deg(\chi)^2/|\mathcal{E}|$, where $\mathcal{E} \subset K$ is a finite subset in which we can draw elements uniformly at random. We will incorporate this probability of failure for the computation of the characteristic polynomial in our bound for the probability of failure of the main algorithm, see the proof of Theorem 5.35.

Constructing the matrix N is done by computing successively the remainders $(\tilde{\lambda}u(S)+v(S))^i \bmod \chi(S)$ for $i \in \{0, \dots, \deg(\chi) - 1\}$ at a total cost of $O(\deg(\chi) M(\deg(\chi)))$ which is again bounded by $O(\deg(\chi)^\omega)$. Finally, inverting N and applying the inverse linear map can be done using $O(\deg(\chi)^\omega)$ operations in K by using [BH74]. \square

Proposition 5.21. *Algorithm 3 (HENSELLIFTINGSTEP) requires at most $O(\deg(q)^2 M(\deg(\chi)))$ arithmetic operations in K .*

Proof. Algorithm 3 consists in evaluations of q and its partial derivatives at $(u(S), v(S))$, together with finitely many arithmetic operations in $K[S]/\chi(S)^2$. Each of the arithmetic operations modulo χ^2 costs $O(M(\deg(\chi)))$ arithmetic operations in K . Evaluating q at $(u(S), v(S))$ modulo $\chi(S)^2$ can be done by computing the remainders $u(S)^i v(S)^j \bmod \chi(S)^2$ for all $(i, j) \in \mathbb{Z}_{\geq 0}$ such that $i + j \leq \deg(q)$, then by multiplying these evaluations by the corresponding coefficients in q and by summing them. Computing all the modular products can be done in $O(\deg(q)^2 M(\deg(\chi)))$ operations in K , by considering the pairs (i, j) in increasing lexicographical ordering. Multiplying by the coefficients and summing then costs $O(\deg(q)^2 \deg(\chi))$ arithmetic operations in K . Computing the evaluations of the partial derivatives of q is done similarly and it has a similar cost. \square

Proposition 5.22. *Algorithm 4 (ADDDIVISORS) requires at most $O(\deg(q)^2 M(\nu) + \nu^\omega)$ arithmetic operations in K , where $\nu = \max(\deg(\chi_1), \deg(\chi_2))$.*

Proof. Algorithm 4 starts by two calls to the function CHANGEPRIMELT, with respective costs $O(\deg(\chi_1)^\omega)$ and $O(\deg(\chi_2)^\omega)$ by Proposition 5.20. The polynomial $\text{GCD}(\hat{\chi}_1, \hat{\chi}_2)$ can be computed at cost $O(M(\nu) \log(\nu))$ using the fast GCD algorithm [vzGG13, Coro. 11.9]. The product $\hat{\chi}$ in Algorithm 4 and the LCM are then also computed at costs $O(M(\nu))$ and $O(M(\nu) \log(\nu))$. The XCRT can be computed at cost $O(M(\nu) \log(\nu))$ by using Equation (5.2) together with the fact that Bézout coefficients can be computed within quasi-linear complexity [vzGG13, Coro. 11.9]. Finally, the Hensel lifting step can be achieved at cost $O(\deg(q)^2 M(\nu))$ by Proposition 5.21. \square

Proposition 5.23. *Algorithm 5 (SUBTRACTDIVISORS) requires at most $O(\nu^\omega)$ arithmetic operations in K , where $\nu = \max(\deg(\chi_1), \deg(\chi_2))$.*

Proof. Most of the steps of Algorithm 5 are similar to steps of Algorithm 4, except that no Hensel lifting is required here. The complexity analysis is similar and we refer to the proof of Proposition 5.22. The only step which does not appear in Algorithm 4 is the exact division of $\hat{\chi}_1$ by the GCD. The cost of this step does not hinder the global complexity since exact division of polynomials can be done in quasi-linear complexity [vzGG13, Thm. 9.1]. \square

In practice, if K is sufficiently large, then choosing a global value for λ and using the same value for all the representations of divisors would succeed with large probability. In this case, we do not need to call the function CHANGEPRIMELT within Algorithms ADDDIVISORS and SUBTRACTDIVISORS. This would decrease significantly the complexities of ADDDIVISORS and SUBTRACTDIVISORS. In any case, this would not change the global asymptotic complexity of Algorithm 1.

Proposition 5.24. *Algorithm 6 (INTERPOLATE) requires at most $O(t^\omega)$ arithmetic operations in K where $t = \deg(\chi) + r$. It returns a polynomial of degree less than $t/\delta + \delta$.*

Proof. First, we recall that $\deg(\chi_E) = r$. The computation of the degree d does not cost any arithmetic operations in K . The construction of the matrix representing the linear map φ can be done by computing all the modular products $u(S)^i v(S)^j$ modulo $\chi(S)$ and χ_E for pairs (i, j) such that $i + j \leq d$ and $j < \delta$. Lemma 5.16 states that the number of such pairs is bounded above by $3t$. By considering the pairs (i, j) in increasing lexicographical ordering, computing all these modular products can be done within $O(t M(t))$ operations in K . Then, since both dimensions of the matrix are in $O(t)$, computing a basis of the kernel can be done at cost $O(t^\omega)$ (for instance via a row echelon form computation, see [Sto00, Thm. 2.10]).

Next, we show the bound on the degree of the polynomial returned. By construction, the inequality $\deg(h) \leq d$ holds so it suffices to show that $d < t/\delta + \delta$. If $\binom{\delta+1}{2} \leq t$, we have $d = \lfloor t/\delta + (\delta - 1)/2 \rfloor < t/\delta + \delta$. Otherwise, $d = \lfloor (\sqrt{1+8t} - 1)/2 \rfloor < \delta < t/\delta + \delta$ by direct computations. In both cases, we have $\deg(h) < t/\delta + \delta$. \square

Proposition 5.25. *Algorithm 7 (COMPPRINC DIV) requires at most $\tilde{O}(\max(\deg(q), \deg(h))^2 \cdot \min(\deg(q), \deg(h)))$ arithmetic operations in K .*

Proof. The two costly steps in Algorithm 7 are the computations of the resultant and of the subresultant of two bivariate polynomials. This can be done within $\tilde{O}(\max(\deg(q), \deg(h))^2 \cdot \min(\deg(q), \deg(h)))$ operations using [vzGG13, Coro. 11.21]. The Bézout bound implies that the degree of the resultant $\tilde{\chi}(S)$ is at most $\deg(q) \deg(h)$, hence the complexities of all the other steps are quasi-linear in $\deg(q) \deg(h)$, which is negligible compared to the cost of the computation of the resultant and the subresultant. \square

We point out that the complexity of computing resultants and subresultants of bivariate polynomials have been recently improved in [Vil18, VDHL19] under some genericity assumptions. However, since the cost in Proposition 5.25 will be negligible in the global complexity estimate, we make no effort to optimize it further.

Proposition 5.26. *Algorithm 8 (NUMERATORBASIS) requires at most $O(t^\omega)$ arithmetic operations in K where $t = \deg(\chi) + r$.*

Proof. By Lemma 5.16, the domain of the map φ has dimension $O(t)$. Using the monomial basis, the matrix representing the map φ can be constructed within $\tilde{O}(t^2)$ operations by doing as in the proof of Proposition 5.24. Similarly to the proof of Proposition 5.24, a basis of the kernel of this matrix can be obtained by computing first a row echelon form of the matrix within $O(t^\omega)$ operations [Sto00, Thm. 2.10]. \square

Divisor	Degree	Subroutine	Complexity
D_h	$< \deg(\mathcal{C})^2 + \deg(D_+)$	INTERPOLATE	$O((\deg(D_+) + r)^\omega)$
D_{res}	$< \deg(\mathcal{C})^2$	COMPPRINC DIV	$\tilde{O}(\max(\deg(\mathcal{C})^3, (\deg(D_+) + r)^2 / \deg(\mathcal{C})))$
D_{num}	$< \deg(\mathcal{C})^2 + \deg(D_+)$	SUBTRACTDIVISORS	$O(\max(\deg(\mathcal{C})^{2\omega}, (\deg(D_+))^\omega))$
		ADDDIVISORS	$O(\max(\deg(\mathcal{C})^{2\omega}, (\deg(D_+))^\omega))$
		NUMERATORBASIS	$O(\max(\deg(\mathcal{C})^{2\omega}, \deg(D_+)^\omega))$

Table 5.1: Degrees of divisors and complexities of the subroutines in terms of the input size.

All the complexities and the degree estimates computed in this section are summed up in Table 5.1. For bounding the degree of $D_{\text{num}} = D_{\text{res}} + D_-$, we use the fact that we can assume without loss of generality that $\deg(D_-) \leq \deg(D_+)$, since otherwise $L(D_+ - D_-)$ is reduced to 0. Summing all the complexity bounds yields the global complexity bound:

Theorem 5.27. *Algorithm 1 (RIEMANNROCHBASIS) requires at most $O(\max(\deg(\mathcal{C})^{2\omega}, \deg(D_+)^\omega))$ arithmetic operations in K .*

Proof. A direct consequence of Propositions 5.22, 5.23, 5.24, 5.25 and 5.26 is that the complexity of Algorithm 1 is bounded by $O(\max(\deg(\mathcal{C})^{2\omega}, (\deg(D_+) + r)^\omega))$. The proof is concluded by noticing that $r = O(\deg(\mathcal{C})^2)$ since $g = \binom{\deg(\mathcal{C})-1}{2} - r$ is nonnegative. \square

5.4.2 Lower bounds on the probability of success

In this section, we examine all possible sources of failures for the main algorithm. In fact, if the assumptions detailed in Section 5.1.2 are satisfied, then failure can only come from a bad choice of an element picked at random. More precisely, we show that these bad choices can be characterized

algebraically and that they are included in the set of roots of polynomials. Bounding the degrees of these polynomials provides us with lower bounds on the probability of success if random elements in K are picked uniformly at random in a finite subset $\mathcal{E} \subset K$.

First, we investigate which values of λ make Algorithm 2 (CHANGEPRIMELT) fail: These are the values of λ such that there is a line of equation $\lambda X + Y + \gamma$ for some $\gamma \in \overline{K}$ which goes either through two distinct points in the support of the input divisor, or which is tangent to \mathcal{C}^0 at a point in the support of the divisor as previously emphasized in Section 5.2.1 and pictured in Figure 5.2.

Proposition 5.28. *Given an effective divisor $D = (\lambda, \chi, u, v)$, the set of $\tilde{\lambda} \in K$ such that Algorithm 2 (CHANGEPRIMELT) with input $D, \tilde{\lambda}$ fails is contained in the set of roots of a nonzero univariate polynomial with coefficients in K of degree at most $\binom{\deg(\chi)+1}{2}$.*

Proof. There are two possible sources of failures for Algorithm 2: if the vector $(1, -\tilde{\lambda})$ is tangent to the curve \mathcal{C}^0 at one of the points in the support of the effective divisor (first test) or if $\tilde{\lambda}X + Y$ is not a primitive element (second test).

Let $J = \langle \chi(\lambda X + Y), X - u(\lambda X + Y), Y - v(\lambda X + Y) \rangle \subset k[\mathcal{C}^0]$ be the ideal associated to the effective divisor D , and let Δ be the polynomial given by Lemma 5.7 for J . By construction, the polynomial Δ satisfies the wanted properties. \square

Before investigating Algorithms 4 and 5 (ADDDIVISORS and SUBTRACTDIVISORS), we need a technical lemma.

Lemma 5.29. *Let $\phi : R \rightarrow S$ be a surjective morphism of finite K -algebras, and let z be a primitive element for R . Then $\phi(z)$ is a primitive element for S .*

Proof. Since ϕ is surjective, any element $y \in S$ equals $\phi(x)$ for some $x \in R$. Since z is primitive, there exists a univariate polynomial $w(S) \in K[S]$ such that $x = w(z)$. Consequently, $y = \phi(w(z)) = w(\phi(z))$. Therefore, $\phi(z)$ is primitive for S . \square

Proposition 5.30. *For a given input (q, D_1, D_2) of Algorithm 4 (ADDDIVISORS), the set of $\hat{\lambda}$ which makes Algorithm 4 fail is contained in the set of roots of a nonzero univariate polynomial with coefficients in K and of degree bounded by $\binom{\deg(\chi_1)+\deg(\chi_2)+1}{2}$.*

Proof. For $i \in \{1, 2\}$, consider $I_i = \langle U - u_i(S), V - v_i(S), \chi_i(S) \rangle \cap K[U, V]$. Lemma 5.7 for the ideal $I_1 \cdot I_2 + \langle q(U, V) \rangle$ yields a nonzero polynomial Δ of degree at most $\binom{\deg(\chi_1)+\deg(\chi_2)+1}{2}$. We will prove that this polynomial satisfies the wanted properties.

Let $\hat{\lambda} \in K$ not be a root of Δ . By definition of Δ , $\hat{\lambda}U + V$ is a primitive element for $\text{red}(K[U, V]/(I_1 \cdot I_2))$ and

$$\begin{aligned} \text{GCD} \left(\frac{\partial q}{\partial X}(u_1(S), v_1(S)) - \hat{\lambda} \frac{\partial q}{\partial Y}(u_1(S), v_1(S)), \chi_1(S) \right) &= 1, \\ \text{GCD} \left(\frac{\partial q}{\partial X}(u_2(S), v_2(S)) - \hat{\lambda} \frac{\partial q}{\partial Y}(u_2(S), v_2(S)), \chi_2(S) \right) &= 1. \end{aligned} \tag{5.3}$$

There are three possible sources of failure for Algorithm 4: the two calls to CHANGEPRIMELT, and the conditional test. The fact that the calls to CHANGEPRIMELT succeed is a direct consequence of Lemma 5.29, using the canonical projections $\text{red}(K[U, V]/(I_1 \cdot I_2)) \rightarrow \text{red}(K[U, V]/I_i)$ for $i \in \{1, 2\}$, see the proof of Proposition 5.28.

Now we prove that the conditionnal test succeeds. By Lemma 5.10 and Proposition 5.11, we have that for $i \in \{1, 2\}$, $I_i = \langle U - \hat{u}_i(S), V - \hat{v}_i(S), \hat{\chi}_i(S) \rangle \cap K[U, V]$. Then $\hat{\lambda}U + V$ must be a primitive element for $\text{red}(K[U, V]/I_1)$ and for $\text{red}(K[U, V]/I_2)$ by Lemma 5.9. Let $\tilde{\chi}_1$ and $\tilde{\chi}_2$ denote the minimal polynomials of $\hat{\lambda}U + V$ in $\text{red}(K[U, V]/I_1)$ and $\text{red}(K[U, V]/I_2)$. Also, set $\xi = \text{LCM}(\tilde{\chi}_1, \tilde{\chi}_2)$. Consequently, $\tilde{\chi}_1(\hat{\lambda}U + V) \cdot \tilde{\chi}_2(\hat{\lambda}U + V) \in \sqrt{I_1 \cdot I_2}$ and ξ is the minimal

polynomial of $\widehat{\lambda}U + V$ in $\text{red}(K[U, V]/(I_1 \cdot I_2)) = K[U, V]/(\sqrt{I_1} \cap \sqrt{I_2})$. Since $\widehat{\lambda}U + V$ is a primitive element for $\text{red}(K[U, V]/(I_1 \cdot I_2))$, then the canonical map

$$\text{red}(K[U, V]/(I_1 \cdot I_2)) \rightarrow \text{red}(K[U, V]/I_2) \times \text{red}(K[U, V]/I_1)$$

becomes a map

$$K[S]/\xi(S) \rightarrow K[S]/\tilde{\chi}_1(S) \times K[S]/\tilde{\chi}_2(S).$$

This implies that there exists an element $\tilde{u}_{12} \in K[S]/\xi(S)$ (which is in fact the class of U in $\text{red}(K[U, V]/(I_1 \cdot I_2))$) such that $\tilde{u}_{12} \equiv \hat{u}_1 \pmod{\tilde{\chi}_1}$ and $\tilde{u}_{12} \equiv \hat{u}_2 \pmod{\tilde{\chi}_2}$. As a consequence, $\hat{u}_1 \equiv \hat{u}_2 \pmod{\text{GCD}(\tilde{\chi}_1, \tilde{\chi}_2)}$. Using Hensel's lemma, the property **(Div-H3)** and the CRT, we obtain that the equation $q(u(S), S - \lambda u(S)) = 0$ has a unique solution s in $K[S]/\text{GCD}(\tilde{\chi}_1(S), \tilde{\chi}_2(S))$ such that $s \equiv \hat{u}_1 \equiv \hat{u}_2 \pmod{\text{GCD}(\tilde{\chi}_1, \tilde{\chi}_2)}$. By **(Div-H1)**, both \hat{u}_1 and \hat{u}_2 are solutions, and therefore $\hat{u}_1 \equiv \hat{u}_2 \pmod{\text{GCD}(\tilde{\chi}_1, \tilde{\chi}_2)}$, which shows that the last conditional test succeeds. \square

Proposition 5.31. *For a given input (q, D_1, D_2) of Algorithm 5 (SUBTRACTDIVISORS), the set of $\widehat{\lambda}$ which makes Algorithm 5 fail is contained in the set of roots of a nonzero univariate polynomial with coefficients in k and of degree bounded by $\binom{\deg(\chi_1) + \deg(\chi_2) + 1}{2}$.*

Proof. The proof is similar to the first part of the proof of Proposition 5.30. With the same notation as in the proof of Proposition 5.30, Algorithm 5 fails only if $\widehat{\lambda}U + V$ is not a primitive element for $\text{red}(K[U, V]/(I_1 \cdot I_2))$ or if the vector $(1, -\widehat{\lambda})$ is tangent to the curve at one of the points in the support of one of the divisors. Using a proof similar to that of Proposition 5.30, this happens only when $\widehat{\lambda}$ is in the set of roots of the nonzero polynomial of degree at most $\binom{\deg(\chi_1) + \deg(\chi_2) + 1}{2}$ provided by Lemma 5.7 for the ideal $I_1 \cdot I_2 + \langle q(U, V) \rangle$. \square

Next, we wish to bound the probability that Algorithm 7 (COMPPRINC DIV) fails. Before stating the next proposition, we recall one assumption that we have made on the input divisor and which is described in Section 5.1.2. It ensures the existence of a form $h \in \overline{K}[\mathcal{C}]$ of given degree such that $(h) \geq D_+ + E$ and $(h) - E$ is smooth. With the notation in the following proposition, this assumption precisely means that $A \neq \ker(\varphi) \otimes_K \overline{K}$. Crudely said, Proposition 5.32 shows that if there is one form h such that $(h) \geq D_+ + E$ and $(h) - E$ is smooth, almost all the forms h verifying $(h) \geq D_+ + E$ are such that $(h) - E$ is smooth for the same reason that if an union of hyperplanes H is not equal to a whole vector space then a random point in this vector space is not in H .

Proposition 5.32. *Let $A \subset \ker(\varphi) \otimes_K \overline{K} \subset \overline{K}[X, Y]$ be the subset of all the regular functions h in the kernel of φ in Algorithm 6 which are such that $D_h = (h) - E$ is not a smooth divisor. If $A \neq \ker(\varphi)$, then A is contained in the join of at most $2r$ hyperplanes in $\ker(\varphi) \otimes_K \overline{K}$.*

Consequently, there is a nonzero polynomial in $\overline{K}[Z_1, \dots, Z_{\dim(\ker(\varphi))}]$ of degree at most $2r$ which vanishes at values $(\mu_1, \dots, \mu_{\dim(\ker(\varphi))})$ for which the third test in Algorithm 7 fails for all $\lambda \in \overline{K}$.

Proof. If D_h involves a point P of $\tilde{\mathcal{C}}$ which projects to a node, then $(h) \geq E + P$. The set of regular functions h of a given degree which satisfy $(h) \geq E + P$ is a linear space. The set of such h in $\ker(\varphi) \otimes_K \overline{K}$ forms a proper subspace since $A \neq \ker(\varphi) \otimes_K \overline{K}$. Consequently, it is contained in an hyperplane. Such a hyperplane H can be described by a linear form ψ in $\overline{K}[Z_1, \dots, Z_{\dim(\ker(\varphi))}]$ such that $\psi(\mu_1, \dots, \mu_{\dim(\ker(\varphi))}) = 0$ if and only if $\sum_{i=1}^{\dim(\ker(\varphi))} \mu_i \mathbf{b}_i \in H$, where $\mathbf{b}_1, \dots, \mathbf{b}_{\dim(\ker(\varphi))}$ is a basis of $\ker(\varphi)$.

Iterating this argument over all the $2r$ points of the nonsingular model $\tilde{\mathcal{C}}$ which project to nodes, we obtain that A is contained in the join of $2r$ hyperspaces. Multiplying the $2r$ corresponding linear forms in $\overline{K}[Z_1, \dots, Z_{\dim(\ker(\varphi))}]$ proves the last sentence of the proposition. \square

Proposition 5.33. *The set of values of λ which make the first test in Algorithm 7 fail is contained within the set of roots of a nonzero univariate polynomial with coefficients in K of degree $\deg(\mathcal{C}) + 1$.*

Proof. Writing $\tilde{q}(S, Y) = q((S - Y)/\lambda, Y)$, the first test fails if $\lambda = 0$ or if the coefficient of the monomial $Y^{\deg(\mathcal{C})}$ in \tilde{q} vanishes. Writing explicitly the change of variables, we obtain that this coefficient equals $\sum_{i=0}^{\deg(\mathcal{C})} (-1/\lambda)^i q_{i, \deg(\mathcal{C})-i}$, where $q_{i,j}$ stands for the coefficient of $X^i Y^j$ in q . Multiplying by $\lambda^{\deg(\mathcal{C})+1}$ clears the denominator and adds the root 0 to exclude the case $\lambda = 0$; this provides a polynomial satisfying the desired properties. \square

Proposition 5.34. *Let $h \in K[\mathcal{C}^0]$ be a regular function such that the support of $(h) - E$ does not contain any singular point. Then the set of λ which makes Algorithm 7 (COMPPRINC DIV) with input q, h fail is contained in the set of roots of a nonzero univariate polynomial with coefficients in K and of degree bounded by $2^{\binom{\deg(\mathcal{C})}{2} \deg(h)+1} + 2r + \deg(\mathcal{C}) + 1$.*

Proof. First, let Δ_1 be the univariate polynomial constructed in Proposition 5.33. The first test in Algorithm 7 does not fail only if λ is not a root of Δ_1 .

The second test in Algorithm 7 fails only if λ is a root of T_E .

By Bézout theorem, the effective divisor (h) has degree at most $\deg(\mathcal{C}) \deg(h)$. Therefore, Lemma 5.6 for the ideal $\sqrt{\langle q, h \rangle}$ yields a nonzero polynomial Δ_2 of degree at most $\binom{\deg(\mathcal{C}) \deg(h)}{2}$ such that the set of λ such that $\lambda X + Y$ is not a primitive element for $\text{red}(K[\mathcal{C}^0]/\langle h \rangle)$. Since $(h) \geq E$, the fact that $\Delta_2(\lambda) \neq 0$ implies that $\lambda X + Y$ is a primitive element for the K -algebra associated to the nodal divisor, and hence the call to the function CHANGEPRIMELTNODAL in Algorithm 7 does not fail. Since by assumption $(h) - E$ is smooth, this also implies that the roots of $\hat{\chi}_E$ are roots of χ with multiplicity exactly 2 by Lemma 5.18. Consequently, if λ is not a root of Δ_2 , then $\text{GCD}(\chi, \hat{\chi}_E) = 1$ and therefore the third test in Algorithm 7 must succeed.

Finally, Lemma 5.7 for the ideal $\sqrt{\langle q, h \rangle} : I_E^\infty \subset K[\mathcal{C}^0]$ yields a nonzero polynomial Δ_3 of degree at most $\binom{\deg(\mathcal{C}) \deg(h)+1}{2}$ such that the set of λ such that $\lambda X + Y$ is not a primitive element for $\text{red}(K[\mathcal{C}^0]/\langle h \rangle)$ or such that the last test in Algorithm 7 fails is contained within the set of roots of Δ_3 .

We claim that the product $\Delta_1 \cdot \Delta_2 \cdot \Delta_3 \cdot T_E$ satisfies the required properties. To prove this claim, it remains to show that if λ is not a root of $\Delta_1 \cdot \Delta_2 \cdot \Delta_3 \cdot T_E$, then the fourth test succeeds, i.e., $a_1(S)$ is invertible modulo $\chi(S)$.

To this end, we notice that $a_1(S)$ is invertible modulo $\chi(S)$ if and only if $a_1(s)$ is nonzero for any root $s \in \bar{K}$ of $\chi(S)$. By [EK03, Cor. 5.1], this is equivalent to the fact that the GCD of the polynomials $q((s - Y)/\lambda, Y)$, $h((s - Y)/\lambda, Y)$ has degree 1 for any root s of $\chi(S)$. Next, we note that if λ is not a root of Δ_3 , then any common root y of $q((s - Y)/\lambda, Y)$ and $h((s - Y)/\lambda, Y)$ has multiplicity 1 in $q((s - Y)/\lambda, Y)$: Indeed, the vanishing of the derivative $\partial/\partial Y$ of $q((s - Y)/\lambda, Y)$ at $Y = y$ would precisely mean that the vector $(1, -\lambda)$ is tangent to the curve at the intersection point, which is impossible by definition of Δ_3 . Consequently, the GCD of the polynomials $q((s - Y)/\lambda, Y)$, $h((s - Y)/\lambda, Y)$ must be squarefree. Finally, let $y_1, y_2 \in \bar{K}$ be two common roots of $q((s - Y)/\lambda, Y)$, $h((s - Y)/\lambda, Y)$. This means that $((s - y_1)/\lambda, y_1)$ and $((s - y_2)/\lambda, y_2)$ are two common zeros of $q(X, Y)$ and $h(X, Y)$. Since λ is not a root of Δ_2 , $\lambda X + Y$ is a primitive element for $\text{red}(K[\mathcal{C}^0]/\langle h \rangle) = K[X, Y]/\langle q, h \rangle$, which implies that $\lambda X + Y$ takes distinct values at all points (x, y) in the variety associated to the system $h(X, Y) = q(X, Y) = 0$ (the endomorphism of multiplication by $\lambda X + Y$ must have distinct eigenvalues, see e.g. the proof of Lemma 5.6 for more details). In particular, this means that $y_1 = y_2$, since $\lambda X + Y$ takes the same value s at $((s - y_1)/\lambda, y_1)$ and $((s - y_2)/\lambda, y_2)$. Consequently, the GCD of the polynomials $q((s - Y)/\lambda, Y)$, $h((s - Y)/\lambda, Y)$ is a squarefree polynomial with at most one root, hence it has degree at most 1. Since $\text{Resultant}(q((s - Y)/\lambda, Y), h((s - Y)/\lambda, Y))$ vanishes and the coefficient of $Y^{\deg(q)}$ in $q((s - Y)/\lambda, Y)$ is nonzero because λ is not a root of Δ_1 , this GCD must have degree at least 1. Therefore, this GCD has degree exactly 1, and hence $a_1(S)$ is invertible modulo $\chi(S)$. \square

Finally, we can derive our bound on the probability that the toplevel algorithm fails by summing the probabilities that the subroutines fail.

Theorem 5.35. *Let $\mathcal{E} \subset K$ be a finite set. Assume that each call to the function $\text{RANDOM}(k)$ is done by picking an element uniformly at random in \mathcal{E} . Then the probability that Algorithm 1 fails is bounded above by*

$$O(\max(\deg(\mathcal{C})^4, \deg(D_+)^2)/|\mathcal{E}|).$$

Proof. Propositions 5.30, 5.31, together with the fact that the number of roots in K of a univariate polynomial is bounded by its degree, directly imply that the probabilities of failure of Algorithms ADDDIVISORS and SUBTRACTDIVISORS are bounded by $O(\max(\deg(D_1), \deg(D_2))^2/|\mathcal{E}|)$, if the computation of the characteristic polynomial in Algorithm CHANGEPRIMELT succeeds. Following [PS07] (see also the remark in the proof of Proposition 5.20), the probability that the computation of the characteristic polynomial in CHANGEPRIME fails is bounded by $\deg(\chi)^2/|\mathcal{E}|$. Therefore, the probabilities that Algorithms ADDDIVISORS and SUBTRACTDIVISORS fail are still bounded by $O(\max(\deg(D_1, D_2))^2/|\mathcal{E}|)$ when we take into account the probability that the computations of the characteristic polynomials fail.

As previously stated, our technical assumption (described in Section 5.2.1) on the input divisor ensures that $A \neq \ker(\varphi)$ in Proposition 5.32. Using Proposition 5.32, Schwartz-Zippel lemma [Sch79, Coro. 1], Proposition 5.34, together with the fact that $r \leq \binom{\deg(\mathcal{C})-1}{2}$, we bound the probability that COMPPRINC DIV fails by $O(\deg(\mathcal{C})^2 \deg(h)^2/|\mathcal{E}|)$.

Algorithm	Failure probability	Statement
CHANGEPRIMELT	$O(\deg(D)^2/ \mathcal{E})$	Prop. 5.28
ADDDIVISORS	$O(\max(\deg(D_1), \deg(D_2))^2/ \mathcal{E})$	Prop. 5.30
SUBTRACTDIVISORS	$O(\max(\deg(D_1), \deg(D_2))^2/ \mathcal{E})$	Prop. 5.31
COMPPRINC DIV	$O(\deg(\mathcal{C})^2 \deg(h)^2/ \mathcal{E})$	Prop. 5.32 Prop. 5.34 Schwartz-Zippel lemma [Sch79, Coro. 1]

Table 5.2: Probabilities of failure.

The failure probabilities are summed up in Table 5.2. Next, notice that the probability of failure of Algorithm 1 is bounded by the sum of the probabilities of the subroutines. Finally, the proof is concluded by using the inequality $\deg(h) < (\deg(D_+) + r)/\deg(\mathcal{C}) + \deg(\mathcal{C})$ (Proposition 5.24) and the degree bounds in Table 5.1 for the divisors arising in Algorithm 1. \square

Deciding whether the assumptions on the input divisor are satisfied.

The result in Theorem 5.35 only holds true if the assumptions on the input divisor described in Section 5.2.1 are satisfied. The first assumption — namely, the smoothness of the input divisor — can be easily checked, so we focus here on deciding whether the second assumption is satisfied or not. Namely, this assumption requires the existence of a form $h \in \overline{K}[\mathcal{C}]$ of degree d — where d is the value computed during the execution of Algorithm INTERPOLATE — such that $(h) \geq D_+ + E$ and $(h) - E$ is smooth. In order to have a complete Las Vegas algorithm (in the sense of [Bab79, Sec. 0.1]), we need to be able to check whether this condition is satisfied.

To this end, instead of returning only one form during Algorithm INTERPOLATE, we can return a basis (h_1, \dots, h_ℓ) of the forms h such that $(h) \geq D_+ + E$. Then, we check if there exists a point above a node which is simultaneously in the support of all the divisors $\{(h_i) - E\}_{i \in \{1, \dots, \ell\}}$. This boils down to computing primitive element representations of the principal divisors $(h_1), \dots, (h_\ell)$, which is done by running on these ℓ forms a modified version of Algorithm COMPPRINC DIV where the last test is removed in order to allow singular points. Each execution of Algorithm COMPPRINC DIV costs $\tilde{O}(\max(\deg(\mathcal{C})^3, (\deg(D_+) + r)^2/\deg(\mathcal{C})))$ operations in K , and thus — using the fact that $\ell = O(\deg(D_+) + \deg(\mathcal{C})^2)$ — the total cost of the procedure is bounded above by $\tilde{O}(\max(\deg(\mathcal{C})^5, \deg(D_+)^{5/2}))$. Therefore, in theory, running this decision procedure increases the overall complexity stated in Theorem 5.27 since the best known value of ω is less than $5/2$. However, in practice this does not change the asymptotic complexity since practical algorithms

for linear algebra rely on Gauss or Strassen approaches; In this case, $\omega > 5/2$, hence the cost of this verification procedure is negligible compared to the global complexity of our algorithm. Multiplying the probability of failure of Algorithm COMPINCPRINC by the number of basis vectors yields the bound $O(\max(\deg(D_+)^3, \deg(\mathcal{C})^6)/|\mathcal{E}|)$ for the probability of failure of this verification procedure.

5.4.3 Implementation and experimental results

We have implemented Algorithm 1 in C++ for $K = \mathbb{Z}/p\mathbb{Z}$, relying on the NTL library for all operations on univariate polynomials and for linear algebra. We have also implemented the group law on the Jacobian of a curve via Riemann-Roch space computations. Our software `rrspace` is freely available at <https://gitlab.inria.fr/pspaenle/rrspace> and it is distributed under the LGPL-2.1+ license.

All the experiments presented below have been conducted on a Intel(R) Core(TM) i5-6500 CPU@3.20GHz with 16GB RAM. The comparisons with the computer algebra system Magma have been done with its version V2.23-8.

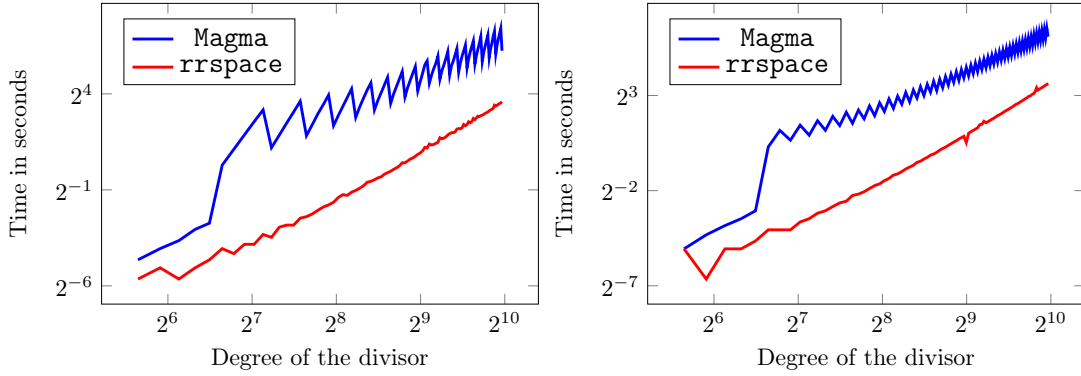


Figure 5.3: Comparison of the time required by `rrspace` and `Magma` to compute a basis of $L(D)$ on a fixed smooth curve of degree 10 over $\mathbb{Z}/65521\mathbb{Z}$. On the left, D is the sum of random irreducible effective divisors of degree 10. On the right, D is a multiple of an irreducible divisor of degree 10. Both axes are in logarithmic scale.

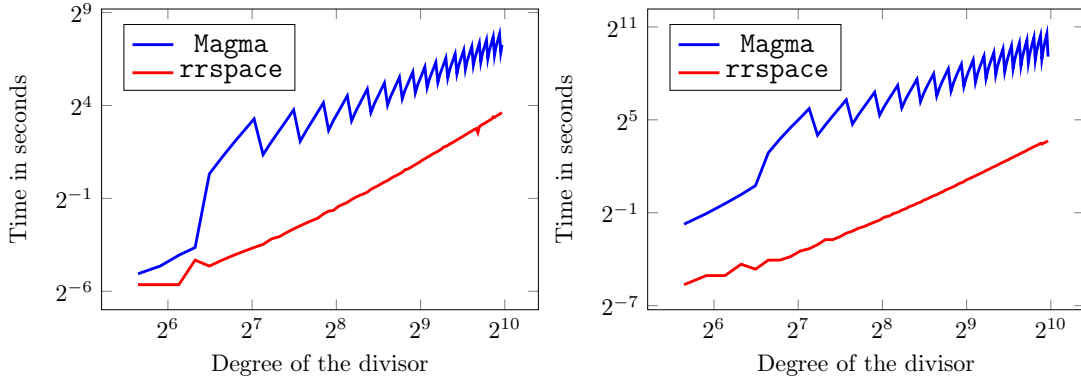


Figure 5.4: Comparison of the time required by `rrspace` and `Magma` to compute a basis of $L(D)$ on a fixed curve of degree 10, where D is the sum of random irreducible effective divisors of degree 10. On the left, the base field is $\mathbb{Z}/65521\mathbb{Z}$ and the curve is nodal. On the right, the base field is $\mathbb{Z}/(2^{32} - 5)\mathbb{Z}$ and the curve is smooth. Both axes are in logarithmic scale.

Our first experimental data is generated as follows. We set $K = \mathbb{Z}/65521\mathbb{Z}$. For i from 10 to 100, we consider a curve \mathcal{C} defined by a random bivariate polynomial of degree 10 over K , and we generate i random irreducible K -defined effective divisors D_1, \dots, D_i of degree 10 on \mathcal{C} by using the `RandomPlace()` function in `Magma`. Then we set $D = D_1 + \dots + D_i$ and we measure the time used for computing a basis of $L(D)$ by using either `Magma` via its function `RiemannRochSpace()` or the software `rrspace`. The experimental results are displayed in the left part of Figure 5.3. For these parameters, we observe that `rrspace` has a speed-up larger than 7 compared to `Magma`. Since we do not have access to the implementation of the function `RiemannRochSpace()` in `Magma`, we cannot explain the small variations which appear in the `Magma` timings.

Our second experimental data investigate the behavior of our algorithm when the input divisor contains multiplicities. To this end, we generate the input divisor as a multiple of a random place of degree 10 on the curve. The experimental results are displayed in the right part of Figure 5.3. For these parameters, we observe that `rrspace` has a speed-up larger than 6 compared to `Magma`.

Our third experimental data study the behavior of our algorithm in the presence of nodes. To this end, we fix the following nodal curve defined by the equation

$$Q(X, Y, Z) = -Y^2Z^8 + X^2Z^8 + Y^4Z^6 - X^3Z^7 + X^{10} - 5Y^{10} + 3X^3Y^7$$

which has a node at the origin and we generate input divisors as for the first experimental data. The experimental results are displayed in the left part of Figure 5.4. For these parameters, we observe that `rrspace` has a speed-up larger than 10 compared to `Magma`.

Finally, since the timings are very sensitive to the efficiency of the linear algebra routines, we study what happens for larger finite fields. The fourth experimental data are generated as for our first experimental data, but we replace the field $\mathbb{Z}/65521\mathbb{Z}$ by the field $\mathbb{Z}/(2^{32} - 5)\mathbb{Z}$. Here, the size of the field is out of the range of the highly optimized arithmetic in `Magma` for small finite fields, and consequently we observe speedups larger than 45 (the speedup goes up to more than 200 for some examples). The experimental results are displayed in the right part of Figure 5.4.

5.4.4 Conclusion

We propose a probabilistic geometric algorithm for computing Riemann-Roch spaces on a plane nodal projective curve \mathcal{C} defined over a sufficiently large perfect field K . Its efficiency relies on classical building blocks in modern computer algebra: fast arithmetic of univariate polynomials and fast linear algebra.

Our main result shows that this algorithm requires at most $O(\max(\deg(\mathcal{C})^{2\omega}, \deg(D_+)^{\omega}))$ arithmetic operations in K , where ω is a feasible exponent for matrix multiplication and D_+ is the smallest effective divisor such that $D_+ \geq D$. In the special case of the group law on the Jacobian of plane smooth curves where $\deg(D_+) = O(g)$ and $\deg(\mathcal{C}) = O(\sqrt{g})$ by the genus-degree formula, the complexity becomes $O(g^{\omega})$ which equals the complexity bound of Khuri-Makdisi's algorithm (see Section 2.2.1) which was the best known bound for this operation at the time this algorithm was designed.

Since then, our algorithm has been enhanced by Abelard, Lecerf and Couvreur [ACL20] in several ways. First, they manage to get rid of the assumption on the form h introduced in Section 5.1.1 by carefully bounding its degree. They also improve our complexity bounds for the arithmetic of smooth divisors and, using the latter results and structured linear algebra, sharpen the complexity bound for the computation of Riemann-Roch spaces : this operation requires $\tilde{O}\left((\deg \mathcal{C}^2 + \deg D_+)^{\frac{\omega+1}{2}}\right)$ operations in a field of sufficiently large characteristic under similar hypotheses as here, the one on h excluded.

Our algorithm may fail, but we show that provided that a few mild assumptions are satisfied, the failure probability is bounded by $O(\max(\deg(\mathcal{C})^4, \deg(D_+)^2)/|\mathcal{E}|)$, where \mathcal{E} is a finite

subset of K in which we pick elements uniformly at random. Roughly speaking, these assumptions on the input require that the impact of the singularities during the execution of the algorithm is minimal. In particular, they are always satisfied for smooth curves. If these mild assumptions are not satisfied, then the algorithm always fails. Therefore, we provide a Las Vegas procedure with complexity $O(\max(\deg(\mathcal{C})^5, \deg(D_+)^{5/2}))$ and probability of failure bounded by $O(\max(\deg(\mathcal{C})^6, \deg(D_+)^3)/|\mathcal{E}|)$ to decide whether these assumptions are satisfied. Combining this verification procedure with our main algorithm turns it into a complete Las Vegas method, at the cost of increasing slightly the complexity and the probability of failure. Anyway, we emphasize that our algorithm is geared towards curves defined over sufficiently large fields K , so that the probability of failure can be made small by choosing a large subset $\mathcal{E} \subset K$. A possible workaround to decrease the probability of failure for curves defined over small finite fields is to do the computations in a field extension, although doing so induces an extra arithmetic cost.

As previously stated, our algorithm rests on extensively studied topics, namely arithmetic on polynomials and linear algebra, for which efficient implementations exist. Consequently, it can easily be made practical. We have made a C++/NTL implementation of our algorithm which is freely distributed under LGPL-2.1+ license and which is available at <https://gitlab.inria.fr/pspaenle/rrspace>. Experimental data indicate that our prototype software is quite competitive with the reference implementation in the Magma computer algebra system, enjoying a speedup larger than 6 on many examples (and larger than 200 on some instances over large finite fields).

Bibliography

- [ABD⁺15] David Adrian, Karthikeyan Bhargavan, Zakir Durumeric, Pierrick Gaudry, Matthew Green, J. Alex Halderman, Nadia Heninger, Drew Springall, Emmanuel Thomé, Luke Valenta, et al. Imperfect forward secrecy: How Diffie-Hellman fails in practice. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 5–17, 2015.
- [Abe20] Simon Abelard. On the complexity of computing integral bases of function fields. In *International Workshop on Computer Algebra in Scientific Computing*, pages 42–62. Springer, 2020.
- [ACGH85] Enrico Arbarello, Maurizio Cornalba, Phillip Griffiths, and Joe Harris. *Geometry of algebraic curves: volume I*. Springer Science & Business Media, 1985.
- [ACL20] Simon Abelard, Alain Couvreur, and Grégoire Lecerf. Sub-quadratic time for riemann-roch spaces: case of smooth divisors over nodal plane projective curves. In *Proceedings of the 45th International Symposium on Symbolic and Algebraic Computation*, pages 14–21, 2020.
- [Adl91] Leonard M. Adleman. Factoring numbers using singular integers. In *Proceedings of the twenty-third annual ACM symposium on Theory of computing*, pages 64–71, 1991.
- [AM69] Michael F. Atiyah and Ian G. Macdonald. *Introduction to commutative algebra*. Addison-Wesley, 1969.
- [ANS14] Agence nationale de la sécurité des systèmes d’information. *Référentiel général de sécurité, v2.03, Annexe B1*, 2014. Publicly available at https://www.ssi.gouv.fr/uploads/2014/11/RGS_v-2-0_B1.pdf.
- [Bab79] László Babai. Monte-Carlo algorithms in graph isomorphism testing. *Université de Montréal, Technical Report*, 1979.
- [BBS04] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In *Annual international cryptology conference*, pages 41–55. Springer, 2004.
- [BCP97] Wieb Bosma, John Cannon, and Catherine Playoust. The Magma algebra system. I. The user language. *J. of Symbolic Computation*, 24(3-4):235–265, 1997.
- [Ber02] Daniel J. Bernstein. How to find small factors of integers, 2002.
- [BF01] Dan Boneh and Matt Franklin. Identity-based encryption from the weil pairing. In *Annual international cryptology conference*, pages 213–229. Springer, 2001.
- [BFHP20] Jean-François Biasse, Claus Fieker, Tommy Hofmann, and Aurel Page. Norm relations and computational problems in number fields. *arXiv preprint arXiv:2002.12332*, 2020.

- [BGG⁺20a] Fabrice Boudot, Pierrick Gaudry, Aurore Guillevic, Nadia Heninger, Emmanuel Thomé, and Paul Zimmermann. Comparing the difficulty of factorization and discrete logarithm: a 240-digit experiment. In *Annual International Cryptology Conference*, pages 62–91. Springer, 2020.
- [BGG⁺20b] Fabrice Boudot, Pierrick Gaudry, Aurore Guillevic, Nadia Heninger, Emmanuel Thomé, and Paul Zimmermann. Comparing the difficulty of factorization and discrete logarithm: A 240-digit experiment. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 62–91. Springer, Heidelberg, August 2020.
- [BGK15] Razvan Barbulescu, Pierrick Gaudry, and Thorsten Kleinjung. The tower number field sieve. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 31–55. Springer, 2015.
- [BH74] James R. Bunch and John E. Hopcroft. Triangular factorization and inversion by fast matrix multiplication. *Mathematics of Computation*, 28(125):231–236, 1974.
- [BL93] Daniel J. Bernstein and Arjen K. Lenstra. A general number field sieve implementation. In *The development of the number field sieve*, pages 103–126. Springer, 1993.
- [BL17] Razvan Barbulescu and Armand Lachand. Some mathematical remarks on the polynomial selection in NFS. *Math. Comp.*, 86(303):397–418, 2017.
- [BLP93] Joe P. Buhler, Hendrik K. Lenstra, and Carl Pomerance. Factoring integers with the number field sieve. In *The development of the number field sieve*, volume 1554 of *Lecture Notes in Math.*, page 50–94. Springer-Verlag, 1993.
- [BLS⁺02] John Brillhart, Derrick H. Lehmer, John L. Selfridge, Bryant Tuckerman, and Samuel S. Wagstaff Jr. *Factorizations of $b^n \pm 1$, $b = 2, 3, 5, 6, 7, 10, 11, 12$ Up to High Powers, Third edition*. American Mathematical Society, 2002.
- [Bou15] Cyril Bouvier. *Algorithmes pour la factorisation d’entiers et le calcul de logarithme discret*. PhD thesis, Université de Lorraine, 2015.
- [BP14] Razvan Barbulescu and Cécile Pierrot. The multiple number field sieve for medium- and high-characteristic finite fields. *LMS Journal of Computation and Mathematics*, 17(A):230–246, 2014.
- [CAD17] The CADO-NFS Development Team. *CADO-NFS, An Implementation of the Number Field Sieve Algorithm*, 2017. Release.
- [Can88] John Canny. Some algebraic and geometric computations in PSPACE. In *Proc. of the twentieth annual ACM Symposium on Theory of Computing (STOC)*, pages 460–467. ACM, 1988.
- [Cav02] Stefania Cavallar. *On the number field sieve integer factorisation algorithm*. PhD thesis, 2002.
- [CEP83] E. R. Canfield, Paul Erdős, and Carl Pomerance. On a problem of Oppenheim concerning “factorisatio numerorum”. *J. Number Theory*, 17(1):1–28, 1983.
- [CK91] David G. Cantor and Erich Kaltofen. On fast multiplication of polynomials over arbitrary algebras. *Acta Informatica*, 28(7):693–701, 1991.
- [Coa70] John Coates. Construction of rational functions on a curve. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 68, pages 105–123. Cambridge University Press, 1970.

- [Coh93] Henri Cohen. *A course in computational number theory*. Springer-Verlag, 1993.
- [Com70] Louis Comtet. Inversion de $y^\alpha e^y$ et $y \log^\alpha y$ au moyen des nombres de Stirling. *C. R. Hebdo. Acad. Sci. Paris Sér. A Math.*, 270:1085–1088, 1970.
- [Com74] Louis Comtet. *Advanced Combinatorics: The art of finite and infinite expansions*. 1974.
- [Cop93] Don Coppersmith. Modifications to the number field sieve. *Journal of Cryptology*, 6(3):169–180, 1993.
- [Cop94] Don Coppersmith. Solving homogeneous linear equations over $\text{gf}(2)$ via block Wiedemann algorithm. *Mathematics of Computation*, 62(205):333–350, 1994.
- [CW25] Allan J.C. Cunningham and Herbert J. Woodall. Factorisation of $y_n \pm 1$, $y = 2, 3, 5, 6, 7, 10, 11, 12$ up to high powers n . *Hodgson, London*, 1925.
- [Dav81] James H. Davenport. On the integration of algebraic functions. 1981.
- [DB51a] Nicolaas G. De Bruijn. The asymptotic behaviour of a function occurring in the theory of primes. *J. Indian Math. Soc. (N.S.)*, 15:25–32, 1951.
- [DB51b] Nicolaas G. De Bruijn. On the number of positive integers $\leq x$ and free of prime factors $> y$. *Proceedings of the Koninklijke Nederlandse Akademie van Wetenschappen. Series A: Mathematical Sciences*, 54(1):50–60, 1951.
- [dB90] Bert den Boer. Diffie-Hellman is as strong as discrete log for certain primes. In Shafi Goldwasser, editor, *Advances in Cryptology — CRYPTO’ 88*, pages 530–539. Springer New York, 1990.
- [DH76] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE transactions on Information Theory*, 22(6):644–654, 1976.
- [DH84] James A. Davis and Diane B. Holdridge. Factorization using the quadratic sieve algorithm. In *Advances in cryptology*, pages 103–113. Springer, 1984.
- [EGT11] Andreas Enge, Pierrick Gaudry, and Emmanuel Thomé. An $L(1/3)$ discrete logarithm algorithm for low degree curves. *Journal of Cryptology*, 24(1):24–41, 2011.
- [EK03] M’hammed El Kahoui. An elementary approach to subresultants theory. *J. of Symbolic Computation*, 35(3):281–292, 2003.
- [ElG85] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on information theory*, 31(4):469–472, 1985.
- [ENI14] European Union Agency for Network and Information Security. *Algorithms, key sizes and parameters report*, 2014. European Union Agency for Network and Information Security. Report available at <https://www.enisa.europa.eu/publications/algorithms-key-size-and-parameters-report-2014>.
- [FGHT17] Joshua Fried, Pierrick Gaudry, Nadia Heninger, and Emmanuel Thomé. A kilobit hidden SNFS discrete logarithm computation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 202–231. Springer, 2017.
- [FK05] Jens Franke and Thorsten Kleinjung. Continued fractions and lattice sieving. *Special-Purpose Hardware for Attacking Cryptographic Systems–SHARCS*, page 40, 2005.
- [FT14] Pierre-Alain Fouque and Mehdi Tibouchi. Close to uniform prime number generation with fewer random bits. In *International Colloquium on Automata, Languages, and Programming*, pages 991–1002. Springer, 2014.

- [Ful08] William Fulton. Algebraic curves: an introduction to algebraic geometry. Version of Jan. 28, 2008.
- [GLS01] Marc Giusti, Grégoire Lecerf, and Bruno Salvy. A Gröbner free alternative for polynomial system solving. *J. of Complexity*, 17(1):154–211, 2001.
- [GM19] Loïc Grenié and Giuseppe Molteni. An explicit Chebotarev density theorem under GRH. *Journal of Number Theory*, 200:441–485, 2019.
- [Gop77] Valerii Denisovich Goppa. Codes associated with divisors. *Problemy Peredachi Informatsii*, 13(1):33–39, 1977.
- [Gop83] Valery Denisovich Goppa. Algebraico-geometric codes. *Izvestiya: Mathematics*, 21(1):75–91, 1983.
- [Gor93] Daniel M. Gordon. Discrete logarithms in $GF(P)$ using the number field sieve. *SIAM Journal on Discrete Mathematics*, 6(1):124–138, 1993.
- [Gra08a] Aandrew Granville. Smooth numbers: Computational number theory and beyond. In *Algorithmic Number Theory: Lattices, Number Fields, Curves and Cryptography*, volume 44 of *Math. Sci. Res. Inst. Publ.*, pages 267–323. Cambridge University Press, December 2008.
- [Gra08b] Andrew Granville. Smooth numbers: computational number theory and beyond. *Algorithmic number theory: lattices, number fields, curves and cryptography*, 44:267–323, 2008.
- [Gro10] Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 321–340. Springer, 2010.
- [Hac95] Gaëtan Haché. Computation in algebraic function fields for effective construction of algebraic-geometric codes. In *Int. Symp. on Applied Algebra, Algebraic Algorithms, and Error-Correcting Codes*, pages 262–278. Springer, 1995.
- [Hes02] Florian Hess. Computing Riemann–Roch spaces in algebraic function fields and related topics. *J. of Symbolic Computation*, 33(4):425–445, 2002.
- [HI94] Ming-Deh Huang and Doug Ierardi. Efficient algorithms for the Riemann-Roch problem and for addition in the Jacobian of a curve. *J. of Symbolic Computation*, 18(6):519–539, 1994.
- [Hil86] Adolf Hildebrand. On the number of positive integers $\leq x$ and free of prime factors $> y$. *J. Number Theory*, 22(3):289–307, 1986.
- [HT93] Adolf Hildebrand and Gérard Tenenbaum. Integers without large prime factors. *Journal de théorie des nombres de Bordeaux*, 5(2):411–484, 1993.
- [JL03] Antoine Joux and Reynald Lercier. Improvements to the general number field sieve for discrete logarithms in prime fields. a comparison with the gaussian integer method. *Mathematics of computation*, 72(242):953–967, 2003.
- [JMV01] Don Johnson, Alfred Menezes, and Scott Vanstone. The elliptic curve digital signature algorithm (ecdsa). *International journal of information security*, 1(1):36–63, 2001.
- [Jou91] Jean-Pierre Jouanolou. Le formalisme du résultant. *Advances in Mathematics*, 90(2):117–263, 1991.
- [Kle06] Thorsten Kleinjung. On polynomial selection for the general number field sieve. *Mathematics of Computation*, 75(256):2037–2047, 2006.

- [KM07] Kamal Khuri-Makdisi. Asymptotically fast group operations on Jacobians of general curves. *Mathematics of Computation*, 76(260):2213–2239, 2007.
- [LBR88] Dominique Le Brigand and Jean-Jacques Risler. Algorithme de Brill-Noether et codes de Goppa. *Bulletin de la Société Mathématique de France*, 116(2):231–253, 1988.
- [Len87] Hendrik W. Lenstra, Jr. Factoring integers with elliptic curves. *Annals of mathematics*, pages 649–673, 1987.
- [LG14] François Le Gall. Powers of tensors and fast matrix multiplication. In *Proc. of the 39th Int. Symposium on Symbolic and Algebraic Computation*, pages 296–303. ACM, 2014.
- [LGS20] Aude Le Gluher and Pierre-Jean Spaenlehauer. A fast randomized geometric algorithm for computing riemann-roch spaces. *Mathematics of Computation*, 89(325):2399–2433, 2020.
- [LGST21] Aude Le Gluher, Pierre-Jean Spaenlehauer, and Emmanuel Thomé. Refined analysis of the asymptotic complexity of the number field sieve. *Mathematical Cryptology*, 1(1):71–88, 2021.
- [LLMP93a] Arjen K. Lenstra, Hendrik W. Lenstra, Mark S. Manasse, and John M. Pollard. The factorization of the ninth Fermat number. *Mathematics of Computation*, 61(203):319–349, 1993.
- [LLMP93b] Arjen K. Lenstra, Hendrik W. Lenstra, Mark S. Manasse, and John M. Pollard. The number field sieve. In *The development of the number field sieve*, pages 11–42. Springer, 1993.
- [LP31] Derrick H. Lehmer and Richard E. Powers. On factoring large numbers. *Bulletin of the American Mathematical Society*, 37(10):770–776, 1931.
- [LV01] Arjen K. Lenstra and Eric R. Verheul. Selecting cryptographic key sizes. *J. Cryptology*, 14(4):255–293, 2001.
- [LV18] Jonathan D. Lee and Ramarathnam Venkatesan. Rigorous analysis of a randomised number field sieve. *J. Number Theory*, 187:92–159, 2018.
- [M⁺99] Brian A. Murphy et al. *Polynomial selection for the number field sieve integer factorisation algorithm*. PhD thesis, 1999.
- [Mar57] Harry M. Markowitz. The elimination form of the inverse and its application to linear programming. *Management Science*, 3(3):255–269, 1957.
- [Mau94] Ueli M. Maurer. Towards the equivalence of breaking the Diffie-Hellman protocol and computing discrete logarithms. pages 271–281. Springer-Verlag, 1994.
- [MB75] Michael A. Morrison and John Brillhart. A method of factoring and the factorization of F_7 . *Mathematics of computation*, 29(129):183–205, 1975.
- [McE78] Robert J. McEliece. A public-key cryptosystem based on algebraic coding theory. *Coding Thv*, 4244:114–116, 1978.
- [Mer78] Ralph C. Merkle. Secure communications over insecure channels. *Communications of the ACM*, 21(4):294–299, 1978.
- [Min07] Lorenz Minder. *Cryptography based on error correcting codes*. PhD thesis, Verlag nicht ermittelbar, 2007.

- [Mir95] Rick Miranda. *Algebraic curves and Riemann surfaces*, volume 5. American Mathematical Soc., 1995.
- [Mon] Chris Monico. GGNFS.
- [Mur98] Brian A. Murphy. Modelling the yield of number field sieve polynomials. In *International Algorithmic Number Theory Symposium*, pages 137–150. Springer, 1998.
- [Mur99] Brian A. Murphy. *Polynomial Selection for the Number Field Sieve Integer Factorisation Algorithm*. PhD thesis, Australian National University, 1999.
- [MW83] Barry Mazur and Andrew Wiles. Analogies between function fields and number fields. *American Journal of Mathematics*, 105(2):507–521, 1983.
- [NIS19] National Institute of Standards and Technology and Canadian Centre for Cyber Security. *Implementation Guidance for FIPS 140-2 and the Cryptographic Module Validation Program*, 2019. Version of 2019, December 3. Available at <https://csrc.nist.gov/CSRC/media/Projects/Cryptographic-Module-Validation-Program/documents/fips140-2/FIPS1402IG.pdf>.
- [NRS17] Vincent Neiger, Johan Rosenkilde, and Éric Schost. Fast computation of the roots of polynomials over the ring of power series. In *Proc. of the 42nd Int. Symposium on Symbolic and Algebraic Computation*, pages 349–356. ACM, 2017.
- [NX09] Harald Niederreiter and Chaoping Xing. *Algebraic geometry in coding theory and cryptography*. Princeton University Press, 2009.
- [Pap] Jason Papadopoulos. Msieve.
- [PH78] Stephen Pohlig and Martin Hellman. An improved algorithm for computing logarithms over $GF(p)$ and its cryptographic significance (corresp.). *IEEE Transactions on information Theory*, 24(1):106–110, 1978.
- [Pol74] John M. Pollard. Theorems on factorization and primality testing. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 76, pages 521–528. Cambridge University Press, 1974.
- [Pol78] John M. Pollard. Monte carlo methods for index computation (mod p). *Mathematics of computation*, 32(143):918–924, 1978.
- [Pol93] John M. Pollard. Factoring with cubic integers. In *The development of the number field sieve*, pages 4–10. Springer, 1993.
- [Pom81] Carl Pomerance. On the distribution of pseudoprimes. *Mathematics of computation*, pages 587–593, 1981.
- [Pom84] Carl Pomerance. The quadratic sieve factoring algorithm. In *Workshop on the Theory and Application of Cryptographic Techniques*, pages 169–182. Springer, 1984.
- [Pom94] Carl Pomerance. The number field sieve. In *Proceedings of Symposia in Applied Mathematics*, volume 48, pages 465–480, 1994.
- [PS92] Carl Pomerance and Jeffrey W. Smith. Reduction of huge, sparse matrices over finite fields via created catastrophes. *Experimental Mathematics*, 1(2):89–94, 1992.
- [PS06] Kenneth G. Paterson and Jacob C.N. Schuldt. Efficient identity-based signatures secure in the standard model. In *Australasian conference on information security and privacy*, pages 207–222. Springer, 2006.

- [PS07] Clément Pernet and Arne Storjohann. Faster algorithms for the characteristic polynomial. In *Proc. of the 32th Int. Symposium on Symbolic and Algebraic Computation*, pages 307–314. ACM, 2007.
- [Ros02] Michael Rosen. *Number theory in function fields*, volume 210. Springer Science & Business Media, 2002.
- [RSA78] Ronald L. Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.
- [Sam97] Pierre Samuel. *Théorie algébrique des nombres*. Hermann, 1997.
- [Sch79] Jacob T. Schwartz. Probabilistic algorithms for verification of polynomial identities. In *EUROSAM: Symbolic and Algebraic Computation*, pages 200–215. Springer, 1979.
- [Sch93] Oliver Schirokauer. Discrete logarithms and local units. *Philosophical Transactions of the Royal Society of London. Series A: Physical and Engineering Sciences*, 345(1676):409–423, 1993.
- [Sch00] Oliver Schirokauer. Using number fields to compute logarithms in finite fields. *Mathematics of Computation*, 69(231):1267–1283, 2000.
- [Sch05] Oliver Schirokauer. Virtual logarithms. *Journal of Algorithms*, 57(2):140–147, 2005.
- [Sem02] Igor Semaev. Special prime numbers and discrete logs in finite prime fields. *Mathematics of computation*, 71(237):363–377, 2002.
- [Sev21] Francisco Severi. *Vorlesungen über algebraische Geometrie*. Springer, 1921.
- [Sha71] Daniel Shanks. Class number, a theory of factorization, and genera. In *Proc. of Symp. Math. Soc., 1971*, volume 20, pages 41–440, 1971.
- [Sho97a] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26:1484–1509, Oct 1997.
- [Sho97b] Victor Shoup. Lower bounds for discrete logarithms and related problems. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 256–266. Springer, 1997.
- [Sto00] Arne Storjohann. *Algorithms for matrix canonical forms*. PhD thesis, ETH Zurich, 2000.
- [VDHL19] Joris Van Der Hoeven and Grégoire Lecerf. Fast computation of generic bivariate resultants. Preprint, 2019.
- [Vil18] Gilles Villard. On computing the resultant of generic bivariate polynomials. In *Proc. of the 43rd Int. Symposium on Symbolic and Algebraic Computation*, pages 391–398. ACM, 2018.
- [Vol94] Emil J. Volcheck. Computing in the Jacobian of a plane algebraic curve. In *Int. Algorithmic Number Theory Symposium*, pages 221–233. Springer, 1994.
- [vzGG13] Joachim von zur Gathen and Jürgen Gerhard. *Modern computer algebra*. Cambridge university press, 2013. Third edition.
- [Wag13] Samuel S. Wagstaff. *The joy of factoring*, volume 68. American Mathematical Soc., 2013.
- [Wie86] Douglas Wiedemann. Solving sparse linear equations over finite fields. *IEEE transactions on information theory*, 32(1):54–62, 1986.

Résumé en français

Cryptographie à clé publique

L'un des buts de la cryptographie est de concevoir et analyser des protocoles permettant à deux entités de communiquer de manière sécurisée, même lorsque le canal de communication utilisé est compromis. De manière générale, un tel protocole se déroule ainsi : l'expéditeur, traditionnellement appelé Bob, chiffre son message à l'aide d'une clé K_0 . Puis, il envoie le message chiffré donc inintelligible au destinataire, classiquement appelé Alice. Cette dernière déchiffre le message à l'aide d'une clé K_1 . Dans le contexte de la cryptographie symétrique, $K_0 = K_1$ (ou du moins, K_1 peut facilement être déduite de K_0). Ainsi, tout individu ayant connaissance de la clé K_0 peut chiffrer des messages à la place de Bob et déchiffrer des messages à la place d'Alice. Par conséquent, si Alice et Bob souhaitent communiquer de manière secrète, ils doivent s'assurer d'être les seuls à connaître la clé K_0 . Ceci mène à une situation inconfortable : Bob et Alice doivent déjà partager un secret commun, la clé K_0 , avant même de s'échanger des secrets.

Dans les années 70, Merkle [Mer78] et Diffie et Hellman [DH76] décrivent une solution élégante à ce problème, posant ainsi les bases de la cryptographie asymétrique, aussi connue sous le nom de cryptographie à clé publique. Dans un protocole cryptographique asymétrique, les clés K_0 et K_1 sont mathématiquement liées mais il est extrêmement difficile de déduire K_1 uniquement à partir de K_0 . Ainsi, pour envoyer un message à Alice, Bob le chiffre avec la clé dite publique d'Alice, K_0 , qu'Alice aura pris soin de diffuser publiquement auparavant. Alice déchiffrera ledit message en utilisant sa clé privée, connue d'elle seule : K_1 . Un attaquant peut connaître la clé K_0 , mais contrairement à ce qui se produit dans un cadre symétrique, cette information ne lui permet pas de déduire la clé K_1 . Pour rendre cette idée pratique, il nous faut à présent construire des fonctions faciles à calculer, de sorte à obtenir un chiffrement efficace, mais très difficiles à inverser à moins de posséder une information spéciale (la clé K_1), de façon à ce que personne ne puisse déchiffrer un message à part le destinataire désiré. Ces fonctions, connues sous le nom de fonctions à sens unique avec trappe, sont au cœur de la cryptographie asymétrique.

Historiquement, deux fonctions ont été abondamment utilisées pour remplir ce rôle : la multiplication d'entiers et l'exponentiation modulaire. Cette dernière consiste à calculer $g^x \bmod N$ à partir de g , x et d'un premier N . Les inverses de ces fonctions sont respectivement la factorisation d'entiers et le calcul de logarithme discret. Cette dernière opération consiste à calculer x à partir de N , g et g^x selon les notations utilisées ci-dessus. La notion de logarithme discret peut même être étendue à un cadre plus général : si G est un groupe cyclique fini et g un générateur de G , calculer un logarithme discret de $h \in G$ consiste en trouver un élément x tel que $g^x = h$. Nous nous concentrons dans ce document sur le cas où G est un sous-groupe multiplicatif fini d'un corps fini.

La factorisation d'entiers et le calcul de logarithmes discrets dans des sous-groupes multiplicatifs de corps finis sont des problèmes mathématiques qui ont été longuement étudiés et ils sont considérés comme difficiles à résoudre. Cela a motivé leur utilisation dans de nombreux cryptosystèmes aujourd'hui largement déployés, comme RSA [RSA78], l'échange de clés de Diffie-Hellman [DH76] ou les schémas de chiffrement et signature de ElGamal [ElG85]. L'intérêt pour le calcul de logarithmes discrets a récemment été ravivé par l'étude grandissante des protocoles à base de couplages [BF01, PS06, BBS04]. Afin d'évaluer la robustesse de ces cryptosystèmes, il faut répondre à la question suivante : A quel point est-il ardu de factoriser un entier ou de calculer un logarithme discret dans un corps fini ? La réponse à cette question est cruciale pour les organismes internationaux chargés de conclure quant aux paramètres — et en particulier la taille des clés — à utiliser dans un cryptosystème donné afin d'assurer un niveau de sécurité voulu. En effet, la taille de la clé K_0 utilisée dans les cryptosystèmes susmentionnés est directement liée à la taille de l'entier à factoriser pour récupérer la clé K_1 . Pour garantir la sécurité de ces cryptosystèmes, ils doivent ainsi être utilisés avec des clés de taille suffisante pour garantir que le problème de factorisation ou de calcul de logarithme discret sous-jacent ne puisse pas être résolu facilement.

Pour répondre à la question posée dans le paragraphe précédent, on peut se tourner vers l'étude des algorithmes permettant de calculer la factorisation d'un entier ou un logarithme discret dans un corps fini. Dans cette thèse, nous nous concentrons sur la complexité temporelle de ces algorithmes : elle dépend bien sûr de la taille de l'entrée à traiter. Dans les années 90, Shor introduisit des algorithmes quantiques permettant de résoudre ces problèmes en temps polynomial [Sho97a], démontrant ainsi qu'il sont raisonnablement faciles à élucider dans ce contexte. Toutefois, à l'heure actuelle, il n'existe pas d'ordinateurs quantiques capables d'effectuer des calculs à grande échelle. C'est pourquoi nous ne quitterons pas dans cette thèse le contexte classique.

Dans ce contexte, le meilleur algorithme à ce jour permettant de calculer à la fois des factorisations d'entiers et des logarithmes discrets dans des corps finis est le crible algébrique (Number Field Sieve en anglais et abrégé NFS dans cette thèse). Plusieurs implémentations de cet algorithme existent. Plusieurs méthodes permettent d'évaluer les ressources, notamment temporelles, nécessaires à la résolution de ces deux problèmes par NFS. L'une est d'utiliser les implémentations existantes de cet algorithme afin de mesurer ses temps d'exécution sur une entrée donnée. Malheureusement, cette méthode consiste en définitive à utiliser une implémentation de NFS sur des entrées de plus en plus grandes : chaque entrée pour laquelle le calcul termine montre que ladite entrée était trop petite pour que sa factorisation (ou le calcul de son logarithme discret) soit difficile mais sur les entrées non testées ou pour lesquelles le calcul n'a pas encore abouti, on ne peut rien conclure. Ce problème peut être atténué en tentant d'estimer le temps d'exécution de l'algorithme sur de grandes entrées à partir des résultats obtenus pour des entrées plus petites mais ces extrapolations sont complexes. Une autre approche vise à *simuler* NFS. L'idée est de concevoir un algorithme efficace qui estimerait le temps requis par une implémentation de NFS pour achever un calcul sur une entrée donnée, sans réellement faire le calcul. Cette idée est attrayante mais de nombreux obstacles s'opposent à sa mise en pratique : la modélisation des opérations effectuées par NFS est malaisée et l'algorithme implique de nombreux paramètres interdépendants qui influencent grandement les temps de calcul. Ceci explique pourquoi la méthode standard pour évaluer les temps d'exécution de NFS, c'est-à-dire pour déterminer quelles sont les tailles de clés pour lesquelles la factorisation ou le calcul de logarithme discret est suffisamment difficile, repose sur la complexité asymptotique de cet algorithme.

La traduction anglaise de "crible algébrique" — à savoir, Number Field Sieve — nous éclaire sur les structures mathématiques utilisées dans cet algorithme : les corps de nombres. La structure algébrique de corps de nombres partage de nombreux points communs avec une autre structure : les corps de fonctions. Le parallèle est observable dès la définition de ces structures : un corps de nombres est une extension finie de \mathbb{Q} et peut toujours être obtenu en quotientant $\mathbb{Q}[X]$ par un polynôme irréductible tandis qu'un corps de fonctions est une extension finie de $\mathbb{F}_q(T)$ (le corps des fractions rationnelles à une variable sur un corps fini à q éléments) et peut toujours être construit à partir d'une courbe algébrique [Ros02, MW83]. Les similarités entre ces deux structures algébriques ont d'ailleurs motivé le fait de les unifier dans les années 30 sous une même dénomination : l'une comme l'autre sont des corps dits globaux. En dépit de cette forte analogie, le vocabulaire et les techniques utilisés dans le contexte des corps de nombres et dans celui des corps de fonctions sont généralement différents. Malgré tout, il est possible d'établir de nombreux parallèles entre les théorèmes et les structures développés dans chacun de ces contextes et, lorsqu'un résultat est établi pour l'un d'entre eux, il est souvent fructueux de le traduire dans l'autre et de développer des techniques analogues.

Une partie de cette thèse est consacrée à l'étude de certains espaces vectoriels dans des corps de fonctions : les espaces de Riemann-Roch. Ce sont des espaces de fonctions rationnelles dont les zéros et les pôles sont contraints par des points d'une courbe algébrique. Ces espaces permettent entre autres de rendre effectif le calcul de la loi de groupe dans la jacobienne d'une courbe, ce qui a de nombreuses applications en théorie des nombres et en géométrie algébrique. Les espaces de Riemann-Roch sont également au cœur de certaines constructions cryptographiques. Ils sont en effet utilisés pour construire des codes correcteurs d'erreur algébraico-géométriques dans la lignée des codes de Goppa [Gop83, Gop77]. Ce genre d'applications nécessite un moyen permettant de calculer efficacement une base d'un tel espace vectoriel.

Contributions

Analyse fine de la complexité asymptotique de NFS.

Sous plusieurs hypothèses heuristiques classiques, la complexité de NFS pour factoriser un entier N est donnée par la formule suivante (voir par exemple [BLP93]) :

$$\exp \left(\sqrt[3]{\frac{64}{9}} (\log N)^{1/3} (\log \log N)^{2/3} (1 + \xi(N)) \right)$$

où $\xi(N)$ est une fonction inconnue telle que $\xi(N) \in o(1)$. Une formule similaire existe dans le contexte du calcul de logarithme discret : N est alors remplacé par p , la taille du corps fini dans lequel ces calculs sont effectués. A ce jour, l'utilisation tronquée de cette formule est une méthode très largement utilisée pour évaluer les ressources nécessaires au calcul de la factorisation d'un entier ou d'un logarithme discret. Plus précisément, cette méthode consiste à remplacer $\xi(N)$ par zéro. La simplification $\xi(N) = 0$ est mathématiquement douteuse et a été initialement considérée comme telle, mais elle est progressivement entrée dans les mœurs et est aujourd'hui considérée comme étant une hypothèse tout à fait acceptable, principalement à cause du fait qu'il n'existe aucune autre méthode efficace pour évaluer le temps de calcul de NFS sur une entrée donnée. Soulignons que les recommandations sur les tailles de clés à utiliser dans les cryptosystèmes basés sur RSA reposent sur cette méthode [NIS19, Sec. 7.5], [ANS14, Sec. B.2.2] [ENI14, Table 3.1].

Dans cette thèse, nous calculons des développements asymptotiques précis pour la fonction ξ ce qui conduit à des formules plus précises pour la complexité asymptotique de NFS. Plus précisément :

1. Nous ré-établissons le problème de minimisation dont la complexité de NFS est solution et proposons une résolution de ce problème de sorte à obtenir la formule au premier ordre présentée ci-dessus pour la complexité asymptotique de NFS. Cette formule et sa preuve sont déjà connues, mais ces dernières utilisent souvent des hypothèses inutiles que nous évitons ici.
2. Nous calculons de nouveaux termes dans le développement asymptotique de la fonction ξ . Ainsi, il est désormais prouvé que

$$\xi(N) \times \log \log N = \frac{4}{3} \log \log \log N + \left(-2 \log 2 + \frac{\log 3}{6} - 2 \right) + o(1).$$

3. Nous montrons que, de manière générale, le développement asymptotique de ξ prend la forme d'une série bivariée évaluée en $\log \log \log N / \log \log N$ et $1 / \log \log N$. Nous proposons également un algorithme permettant de calculer les coefficients d'une série bivariée $\mathbf{A} \in \mathbb{Q}(\log 2, \log 3)[[X, Y]]$ avec coefficient constant égal à 1 et telle que la complexité asymptotique de NFS $C(N)$ vérifie :

$$\frac{\log C(N)}{\sqrt[3]{\frac{64}{9}} (\log N)^{\frac{1}{3}} (\log \log N)^{\frac{2}{3}}} = \mathbf{A}^{(n)} \left(\frac{\log \log \log N}{\log \log N}, \frac{1}{\log \log N} \right) + o \left(\frac{1}{(\log \log N)^n} \right)$$

où $\mathbf{A}^{(n)}$ représente la série \mathbf{A} tronqué au degré total n . Une implémentation de cet algorithme qui a permis de calculer plus d'une centaine de termes du développement asymptotique de la complexité de NFS est disponible à cette adresse :

https://gitlab.inria.fr/NFS_asymptotic_complexity/simulations

4. Nous étudions la convergence de la série associée à ξ et montrons que le domaine pour lequel il est légitime de remplacer $\xi(N)$ par son développement tronqué se situe au delà

de $N > \exp(\exp(22)) \approx 2^{5171935985}$ ce qui est très largement au delà des valeurs classiquement utilisées dans les applications en cryptographie (pour lesquelles on a tout au plus $N \leq 2^{20000}$). Ainsi, remplacer $\xi(N)$ par une de ses troncatures lorsque N est une valeur cryptographiquement pertinente revient à remplacer une série par ses premiers termes dans un domaine où cette série diverge. Se servir des formules tronquées pour la complexité de NFS dans l'espoir d'extrapoler des tailles de clés est donc une méthode à utiliser avec la plus grande prudence. Ceci met en exergue la nécessité de développer des outils de simulation fiables de NFS afin d'estimer sa complexité en pratique.

En résumé, ce travail montre que la complexité asymptotique de NFS peut être connue avec beaucoup de précision mais que ce fait ne se traduit malheureusement pas en outil prédictif fiable des temps de calcul de cet algorithme en pratique. Les résultats ici présentés étendent ceux établis dans l'article suivant, précisent quelques points techniques et modifient les arguments nécessaires à l'établissement du problème de minimisation dont la complexité de NFS est solution en utilisant l'hypothèse de Riemann généralisée.

[LGST21] Refined Analysis of the Asymptotic Complexity of the Number Field Sieve, with Pierre-Jean Spaenlehauer and Emmanuel Thomé, published in Mathematical Cryptology in 2021.

Etude d'un outil de simulation de l'étape du filtre dans NFS.

D'après ce qui précède, utiliser une formule tronquée de la complexité asymptotique de NFS pour déduire les temps de calcul de cet algorithme en pratique est une méthode rapide pour produire des estimations qui sont potentiellement très loin de la réalité. Ceci motive évidemment l'idée de trouver d'autres moyens qui permettraient de fournir des estimations précises du temps de calcul de NFS sur une entrée d'une taille donnée. Il en existe un : utiliser simplement une implémentation de NFS sur l'entrée désirée. Cette méthode produit effectivement des estimations très précises mais par définition elle est inapplicable pour des entrées de taille suffisamment grande. Nous aimerions donc déterminer une méthode permettant non seulement de faire des estimations précises, mais encore de les obtenir en un temps raisonnable. La simulation de NFS est une technique qui remplirait les deux critères : en modélisant les principales étapes de cet algorithme, il serait possible d'estimer leur temps d'exécution sans procéder à un calcul complet.

Il existe peu d'outils de simulation de NFS, mais l'un d'entre a récemment été conçu et utilisé avec succès lors des records RSA-240 et DLP-240 présentés dans [BGG⁺20a]. Il a été implémenté en utilisant l'implémentation CADO-NFS de NFS. Cet algorithme de simulation, qu'on appellera ESTIMATEMATSIZE dans cette thèse, prend en entrée un ensemble de paramètres, simule l'étape dite du filtre de NFS et produit en sortie une matrice dont la taille et la densité sont liées au temps d'exécution de l'étape d'algèbre linéaire de NFS. Il simule ainsi une partie de cet algorithme. Les auteurs de [BGG⁺20a] utilisent cet outil de simulation afin de tester des jeux de paramètres en amont du calcul réel et ainsi déterminer lesquels sont les plus adaptés à leur calcul record.

Dans cette thèse, nous étudions expérimentalement l'algorithme ESTIMATEMATSIZE. L'objectif à long terme de ce travail est de concevoir un outil de prédiction rapide et fiable pour une partie de l'algorithme NFS. Plus précisément :

1. Nous donnons une description précise de la modélisation de l'étape du filtre utilisée dans l'algorithme ESTIMATEMATSIZE.
2. Nous développons des outils efficaces et des méthodes permettant d'analyser rétrospectivement cet algorithme et ses performances. Ils visent en particulier à découpler autant que possible l'impact de chaque simplification effectuée dans la modélisation sur la qualité des prédictions.
3. Nous analysons les performances de cet algorithme de simulation dans des contextes variés. Cette étude permet de mieux évaluer la fiabilité de l'algorithme et a révélé des tendances dans son comportement qui restent inexplicables.

4. Nous proposons plusieurs modifications — dont certaines ont été implémentées — à la modélisation d'origine, visant en particulier à atténuer certaines approximations brutales.

Ces travaux ont mené à des contributions au logiciel CAD-NFS. Cette étude est pour l'heure principalement descriptive. Elle souligne à nouveau le fait que simuler NFS, même partiellement, est un défi complexe, en particulier à cause du nombre important de paramètres interdépendants impliqués dans cet algorithme. Lorsqu'on ajoute à ces derniers les paramètres liés à l'algorithme de simulation lui-même, l'explication des phénomènes inattendus observés dans cette thèse devient alors très compliquée. Toutefois, cette étude montre également que les milliers d'estimations faites avec l'implémentation de ESTIMATEMATSIZE disponible dans CADO-NFS se situent toujours dans une fourchette de 20% autour des valeurs réelles. Ainsi, l'outil de simulation existant semble déjà plutôt fiable ce qui est une bonne nouvelle pour le futur de la simulation de NFS.

Conception et implémentation d'un algorithme efficace pour le calcul d'espaces de Riemann-Roch.

Si \mathcal{C} est une courbe projective de genre g définie sur un corps K , un diviseur de \mathcal{C} et une somme formelle finie de points de \mathcal{C} à coefficients entiers. Tout diviseur D peut s'écrire sous la forme $D = D_+ - D_-$ où les coefficients de D_+ et D_- sont tous positifs. L'espace de Riemann-Roch $L(D)$ associé au diviseur $D = D_+ - D_- = \sum_{i=1}^n n_i P_i - \sum_{j=1}^k m_j Q_j$ est le K -espace vectoriel des fonctions f sur \mathcal{C} — c'est-à-dire, des fractions rationnelles dans lesquelles numérateurs et dénominateurs sont considérés modulo le polynôme décrivant \mathcal{C} — telle que f ne peut avoir de pôles qu'en les points P_i avec multiplicité au pire n_i et doit nécessairement avoir parmi ses zéros tous les points Q_j avec multiplicité au moins m_j .

Le calcul d'une base d'un tel espace vectoriel est une sous-routine utilisée dans plusieurs domaines informatiques et mathématiques, et en particulier dans la construction de codes correcteurs d'erreur algébriques-géométriques et dans le calcul effectif de la loi de groupe dans les jacobiniennes de courbes. Deux familles d'algorithmes permettant le calcul de ces bases existent.

Les approches arithmétiques reposent sur des algorithmes efficaces dans les corps de fonctions. L'algorithme de référence dans cette catégorie est du à Hess [Hes02] : il permet de calculer un espace de Riemann-Roch associé au diviseur $D = D_+ - D_-$ sans restriction sur les singularités de la courbe considérée. Il est polynomial en la taille de l'entrée, c'est-à-dire en $\deg \mathcal{C}$ et $\deg D_+$ où $\deg \mathcal{C}$ est le degré du polynôme qui définit \mathcal{C} et $\deg D_+$ est le nombre de points de D_+ comptés avec multiplicité ce qui est une mesure pertinente de la taille de D étant donné que $L(D) = \{0\}$ dès lors que $\deg D_- > \deg D_+$. Cet algorithme est implémenté dans le logiciel de calcul formel MAGMA.

Les approches géométriques ont été initiées par Goppa [Gop83]. Leur correction repose sur le théorème des résidus de Brill et Noether [Ful08, Section 8.1]. Dans les années 90, plusieurs algorithmes géométriques ont vu le jour, notamment l'algorithme de Huang et Ierardi [HI94] qui permet de calculer déterministiquement un espace de Riemann-Roch d'une courbe plane ordinaire en $O(\deg(\mathcal{C})^6 \deg(D_+)^6)$ opérations dans le corps de base, et celui de Haché [Hac95] une année plus tard. Le cas particulier crucial des algorithmes destinés à calculer la loi de groupe des jacobiniennes de courbes peut également être abordé par des méthodes géométriques. Volcheck [Vol94] décrit ainsi un algorithme permettant de résoudre ce problème dont la complexité est en $O(\max(\deg(\mathcal{C}), g)^7)$ et Khuri-Makdisi [KM07] obtient quelques années plus tard une complexité en $O(g^\omega)$ où ω est tel qu'il est possible de multiplier deux matrices de taille n en $O(n^\omega)$. Au début de cette thèse, cette complexité était la meilleure borne obtenue pour le calcul de la loi de groupe dans la jacobienne d'une courbe.

Dans cette thèse, nous concevons un algorithme probabiliste efficace pour le calcul d'espaces de Riemann-Roch sur une courbe nodale plane définie sur un corps parfait suffisamment grand. Les courbes nodales sont les plus simples des courbes singulières (leurs singularités sont toutes ordinaires d'ordre deux), mais elles sont aussi génériques puisque toute courbe algébrique admet

un modèle nodal lorsque le corps de base est suffisamment grand [ACGH85, Appendix A]. Plus précisément :

1. Nous proposons une façon de représenter un diviseur grâce à un ensemble de polynômes. Cette représentation est inspirée par les coordonnées de Mumford et pas la façon de représenter un ensemble algébrique par éléments primitifs.
2. Nous utilisons cette représentation et le schéma de Brill-Noether qui sous-tend les algorithmes géométriques pour construire et prouver un algorithme de calcul d'espaces de Riemann-Roch. Soulignons que les briques élémentaires utilisées dans cet algorithme, à savoir, arithmétique sur les polynômes univariés et algèbre linéaire, sont des domaines très étudiés pour lesquels des implémentations efficaces sont disponibles. Ainsi, l'analyse de notre algorithme et son implémentation sont aisées.
3. Nous prouvons que la complexité de cet algorithme est bornée par un $O(\max(\deg(\mathcal{C})^{2\omega}, \deg(D_+)^{\omega}))$ où ω est tel qu'il est possible de multiplier deux matrices de taille n en $O(n^{\omega})$. Dans le cas particulier du calcul de la loi de groupe dans la jacobienne d'une courbe lisse, notre algorithme fait donc en au plus $O(g^{\omega})$ opérations dans K (dans ce contexte en effet, $\deg(D_+) = O(g)$ et $\deg(\mathcal{C}) = O(\sqrt{g})$), ce qui correspond à la borne obtenue par Khuri-Makdisi.
4. A condition que quelques hypothèses légères soient satisfaites, nous prouvons que la probabilité que cet algorithme échoue est bornée par un $O(\max(\deg(\mathcal{C})^4, \deg(D_+)^2)/|\mathcal{E}|)$ où \mathcal{E} est un sous ensemble fini de K dans lequel on peut choisir un élément uniformément aléatoirement. Nous fournissons également un algorithme permettant de décider si les hypothèses susmentionnées sont satisfaites.
5. Une implémentation C++/NTL de cet algorithme est disponible à l'adresse suivante :

<https://gitlab.inria.fr/pspaenle/rrspace>

Plusieurs expériences montrent que l'efficacité de notre logiciel surpasse celle de l'implémentation de référence utilisée dans MAGMA.

Depuis, notre algorithme a été raffiné par Abelard, Couvreur et Lecerf [ACL20]. Ces derniers obtiennent une complexité sub-quadratique pour le calcul d'espaces de Riemann-Roch en utilisant notamment de l'algèbre linéaire structurée, mais ne fournissent pas d'implémentation pour cet algorithme théoriquement plus efficace. Les résultats décrits dans cette thèse figurent également dans l'article suivant.

[LGS20] A Fast Randomized Geometric Algorithm for Computing Riemann-Roch spaces, with Pierre-Jean Spaenlehauer, published in Mathematics of Computation in 2020.

Plan de la thèse

Dans un premier chapitre, nous détaillons quelques applications importantes à la factorisation d'entiers et au calcul de logarithmes discrets avant de présenter les principales étapes de l'algorithme du crible algébrique qui permet de résoudre ces deux problèmes. L'étape du filtre de NFS est particulièrement détaillée afin d'introduire les subtilités nécessaires à la bonne compréhension du fonctionnement de l'outil de simulation présenté au chapitre 4. Un deuxième chapitre pose les bases pour le chapitre 5 en introduisant les espaces de Riemann-Roch, leur applications, et la méthode géométrique due à Brill et Noether permettant de calculer ces espaces et qui sera au coeur de l'algorithme décrit dans ce dernier chapitre.

Les chapitres 3, 4 et 5 développent nos contributions : les deux premiers traitent de l'analyse des temps de calcul de NFS et le dernier du calcul d'espaces de Riemann-Roch. Le chapitre 3 se concentre sur les résultats au sujet de l'analyse de la complexité asymptotique de NFS. Le chapitre 4 délasse les analyses théoriques pour se plonger dans la description et l'évaluation de l'outil de simulation ESTIMATEMATSIZE. Enfin, le chapitre 5 présente les résultats relatifs au calcul d'espaces de Riemann-Roch.

Abstract

This thesis focuses on three problems that all relate to cryptography: the factorization of integers, the computation of discrete logarithms in multiplicative subgroups of finite fields and the computation of Riemann-Roch spaces on plane projective curves.

To this day, the Number Field Sieve (NFS for short) is the most efficient algorithm allowing to factor integers and compute discrete logarithms in finite fields, both in theory and in practice. First, we thoroughly study the asymptotic complexity of this algorithm. We prove very precise asymptotic formulas for the asymptotic complexity of NFS and show that, unfortunately, these formulas cannot be used to extrapolate NFS computing times for cryptographically-relevant input sizes. Indeed, such sizes are far smaller than the sizes needed for the use of the asymptotic formulas to even make sense. This study allows to question the standard method used to establish key sizes for RSA-based cryptography.

Since relying on its asymptotic complexity seems a questionable method to predict practical computing times for the Number Field Sieve, we turn to another approach: simulation. Thus, we study an algorithm that simulates a step of the NFS algorithm, namely, the filtering step. The end goal of such an algorithm is to partly predict the behaviour of a computation done with an NFS implementation without actually running it, which would be too costly. We describe this simulation tool in detail, propose a number of experiments aimed at assessing its reliability and accuracy, and present their results.

Finally, we present a probabilistic algorithm for the computation of Riemann-Roch spaces on projective plane nodal curves, whose efficiency rests on two extensively studied building blocks in modern computer algebra: fast arithmetic of univariate polynomials and fast linear algebra. As a by-product, our algorithm also yields a fast method for computing the group law on the Jacobian of a plane curve. We assess the efficiency of this algorithm both theoretically through a complexity analysis and experimentally using an implementation we made.

Résumé

Cette thèse étudie trois problèmes mathématiques liés à la cryptographie : la factorisation d'entiers, le calcul de logarithmes discrets dans des sous-groupes multiplicatifs de corps finis et le calcul d'espaces de Riemann-Roch sur des courbes projectives planes.

L'algorithme du crible algébrique (Number Field Sieve en anglais, classiquement abrégé en NFS) est à l'heure actuelle le plus performant, tant en théorie qu'en pratique, pour résoudre les deux premiers problèmes susmentionnés. Dans un premier temps, nous étudions en détail la complexité asymptotique de cet algorithme. Nous proposons pour cette dernière des formules asymptotiques très précises qui malheureusement ne se traduisent pas en outil fiable de prédiction des temps de calcul de NFS. En effet, en pratique, la taille des entrées pour lesquelles il serait pertinent d'estimer le temps de calcul de NFS est très largement inférieure aux tailles nécessaires pour que l'utilisation des formules asymptotiques ait un sens. Cette étude remet en question la procédure classique sur laquelle repose les recommandations pour les tailles de clés utilisées dans la cryptographie basée sur RSA.

Puisque utiliser la complexité asymptotique de NFS semble être une méthode critiquable pour évaluer les temps de calcul de cet algorithme en pratique, nous nous tournons vers une autre méthode de prédiction : la simulation. Ainsi, nous étudions dans un deuxième temps un algorithme simulant l'une des étapes de NFS : l'étape de filtrage. L'objectif d'un tel algorithme est de prédire en partie le comportement de NFS sur une entrée et un jeu de paramètres donnés, ce de manière plus efficace qu'en utilisant directement une implémentation de NFS sur les mêmes données. Après une description de cet outil de simulation, nous présentons les résultats d'études expérimentales destinées à évaluer sa fiabilité et sa précision.

Enfin, nous présentons un algorithme probabiliste permettant le calcul d'espaces de Riemann-Roch sur des courbes projectives planes nodales. L'analyse théorique de complexité de cet algorithme ainsi que les résultats expérimentaux obtenus avec l'implémentation qui en a été faite entérinent son efficacité.