Algorithmes pour les polynômes creux : interpolation, arithmétique, test d'identité

Armelle Perret Du Cray

THÈSE POUR OBTENIR LE GRADE DE DOCTEUR DE L'UNIVERSITÉ DE MONTPELLIER

En informatique

École doctorale I2S – Informations, Structures, Systèmes

Unité de recherche LIRMM – Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier

Algorithmes pour les polynômes creux : interpolation, arithmétique, test d'identité

Présentée par Armelle Perret du Cray

Le 29 mars 2023

Sous la direction de Pascal Giorgi

Devant le jury composé de

Grégoire LECERF, directeur de recherche CNRS, LIX, Palaiseau	Rapporteur
Éric SCHOST, professeur, University Waterloo (Canada)	Rapporteur
Stéphane BESSY, professeur, Université de Montpellier, LIRMM, Montpellier	Examinateur
Clément PERNET, professeur, Grenoble INP-UGA, LJK, Grenoble	Examinateur
Adeline ROUX-LANGLOIS, chargée de recherche CNRS, GREYC, Caen	Examinatrice
Pascal GIORGI, professeur, Université de Montpellier, LIRMM, Montpellier	Directeur
Bruno GRENET, maître de conférences, Université Grenoble Alpes, LJK, Grenoble	Co-encadrant

UNIVERSITÉ DE MONTPELLIER

Résumé

La manipulation de polynômes est une étape souvent incontournable, que ce soit pour la résolution de problèmes théoriques ou pour la modélisation du monde physique. Dans le cadre des polynômes denses, de nombreuses années de recherches ont permis de développer des algorithmes quasi-optimaux pour les opérations les plus classiques, comme la multiplication ou l'interpolation. Cependant, pour des raisons de compromis mémoire/ temps, il est souvent plus adéquat de représenter les polynômes sous une forme creuse. Dans cette représentation, l'optimalité (ou la quasi-optimalité) est bien plus difficile à atteindre. Cette thèse s'intéresse à cette problématique et présente de nouveaux algorithmes améliorant les complexités connues pour les polynômes creux.

La première opération traitée est l'interpolation d'un polynôme creux. Les solutions apportées précédemment dépendent fortement du modèle sous-jacent et de l'anneau de définition sans toutefois atteindre des complexités quasi-optimales. Nos travaux répondent favorablement à cette question d'optimalité dans le cas de polynômes ayant pour coefficients des nombres entiers.

Dans un second temps, la question difficile de la divisibilité entre deux polynômes creux est abordée. Tout d'abord, nous décrivons une famille non-triviale de polynômes pour laquelle nous proposons un test de divisibilité en temps polynomial. Nous proposons également des algorithmes très efficaces, optimaux dans certain cas, permettant de vérifier un produit de polynômes modulo un polynôme creux. Ces résultats permettent d'obtenir le premier algorithme quasi-linéaire de vérification de produits de polynômes creux.

Enfin, les opérations arithmétiques classiques : multiplication et division sont aussi étudiées. Cette thèse montre en particulier comment s'appuyer sur la vérification et l'interpolation pour obtenir des algorithmes de produit et division exacte efficaces. Dans le cas des polynômes à coefficients entiers, notre approche permet d'obtenir pour la première fois des algorithmes quasi-optimaux.

Abstract

Handling polynomials is often a necessary step, either for solving theoretical problems or for modeling the physical world. In the context of dense polynomials, after many years of research, quasi-optimal algorithms have been designed for the most classical operations, such as multiplication or interpolation. However, for reasons of memory/time trade-off, it is often more appropriate to use the sparse representation of the polynomials. In this representation, optimality (or quasi-optimality) is much harder to achieve. This thesis addresses this problem and presents new algorithms improving the known complexities for sparse polynomials.

The first discussed operation is the interpolation of a sparse polynomial. Previous solutions showed a strong dependency on the underlying model and the coefficient ring. Still, no quasi-optimal algorithms exist. Our work brings a first quasi-optimal algorithm for the case of integer polynomials.

In a second step, the hard question of divisibility between two sparse polynomials is addressed. First, we describe a non-trivial family of polynomials for which we propose a divisibility test in polynomial time. We also propose very efficient algorithms, optimal in some cases, to verify a product of polynomials modulo a sparse polynomial. These results lead to the first quasi-linear algorithm for the verification of sparse polynomials products.

Finally, the classical arithmetic operations: multiplication and division are also discussed. In particular, this thesis shows how to rely on verification and interpolation to obtain efficient algorithms. In the case of polynomials with integer coefficients, our approach reaches for the first time a quasi-optimal complexity.

Table des matières

Introduction **1**

1. Préliminaires **7**
1.1. Analyse d'algorithmes probabilistes 7
1.1.1. Calculs de probabilités 8
1.2. Polynômes denses ou creux 9
1.2.1. Représentation et taille des polynômes 9
1.2.2. Structure de données 11
1.2.3. Restriction aux polynômes univariés 12
1.3. Outils de base 13
1.3.1. Opérations élémentaires 14
1.3.1.1. Arithmétique d'entiers et de polynômes denses 14
1.3.1.2. Réduction modulaire et évaluation 15
1.3.2. Génération d'irréductibles dans un anneau 16
1.3.2.1. Générer des nombres premiers 16
1.3.2.2. Générer des polynômes irréductibles 17
1.4. Impact de l'arithmétique sur la taille des polynômes creux 18
1.4.1. Variation de tailles lors d'additions, produits et divisions 18
1.4.2. Croissance lors de réduction modulaire 20

2. Travailler avec des polynômes de taille toujours bornée **23**
2.1. Limiter la perte d'information lors des réductions 24
2.1.1. Empêcher l'annulation 24
2.1.1.1. Réduction modulo un nombre premier 25
2.1.1.2. Réduction d'un polynôme modulo un polynôme irréductible 26
2.1.1.3. Réduction d'un polynôme modulo un binôme 27
2.1.2. Cas des polynômes creux : éviter les collisions 28
2.2. Réduction d'un polynôme entier par l'évaluation dans un anneau modulaire 32
2.2.1. Générer un corps avec un sous groupe multiplicatif d'ordre p . . . 32
2.2.1.1. La primalité de p 33
2.2.1.2. La primalité de q 34
2.2.1.3. L'ordre multiplicatif de ω 36
2.2.1.4. La coprimalité de q et N 37
2.2.2. Approche heuristique 37
2.3. Évaluation d'un produit de polynôme dans un anneau quotient 40
2.3.1. Évaluation de la réduction cyclique d'un produit 41

2.3.2. Évaluation d'un produit modulaire dense 45
2.3.3. Évaluation d'un produit modulaire creux 49
2.3.3.1. Arbres de van Emde Boas creux 53

3. Interpolation creuse **59**
3.1. Historique des algorithmes d'interpolation creuse 61
3.1.1. Pour un polynôme sous forme de boîte noire 61
3.1.1.1. Extensions de la méthode de Prony et Ben-Or–Tiwari . . 64
3.1.1.2. Autres interpolations par boîte noire 65
3.1.2. Pour un polynôme sous forme de circuit arithmétique 66
3.1.2.1. Algorithme de Garg et Schost 67
3.1.3. Interpoler en utilisant aussi les dérivées du polynôme 70
3.2. Interpolation quasi-linéaire pour un polynôme sur les entiers 73
3.2.1. Idées générales 74
3.2.1.1. Une approche récursive 74
3.2.1.2. Calcul des exposants de $F \bmod x^p - 1$ dans un petit corps 75
3.2.1.3. Calcul des coefficients de $F \bmod x^p - 1$ dans un grand anneau 75
3.2.1.4. Écrire les exposants dans les coefficients 76
3.2.1.5. Répétitions 76
3.2.2. Calculer avec les différents modulo 76
3.2.3. Description et analyse de l'algorithme d'interpolation 81

4. Tester des identités polynomiales **85**
4.1. Divisibilité entre deux polynômes creux 87
4.1.1. Borner le nombre de monômes d'un quotient 88
4.1.2. Tests de divisibilité 90
4.1.3. Application aux polynômes à coefficients entiers 93
4.2. Vérification d'un produit modulaire 95
4.2.1. Cas général de polynôme dans $\mathbb{A}[x]$ 96
4.2.2. Cas des polynômes à coefficients entiers 99
4.2.3. Cas des polynômes à coefficients dans des corps finis 102
4.2.3.1. Vérification sans passer dans une extension 104
4.3. Vérification d'un produit de polynômes 108
4.3.1. Cas d'un produit de polynômes denses : algorithmes optimaux . . 110
4.3.1.1. L'algorithme de Kaminski 110
4.3.1.2. Analyse de la complexité binaire 112
4.3.2. Cas d'un produit de polynômes creux : algorithmes quasi-linéaires 115
4.3.2.1. Analyse de la complexité binaire sur les entiers 118
4.3.2.2. Analyse de la complexité binaire sur les corps finis 119

5. Arithmétique des polynômes creux : produit et division exacte **122**
5.1. État de l'art 124
5.1.1. Méthodes naïves 124

5.1.2. Vers une meilleure dépendance dans le nombre de monômes du produit . . . 126
5.1.2.1. Algorithmes quasi-linéaires dans le support structurel . . 127
5.1.2.2. Algorithmes quasi-linéaires dans le support arithmétique . 128
5.2. Produit de polynômes creux . . . 129
5.2.1. Algorithme quasi-linéaire sur les entiers . . . 129
5.2.2. Produit de polynômes creux sur un anneau $\mathbb{A}$ . . . 131
5.2.2.1. Réduction à l'interpolation . . . 131
5.2.2.2. Algorithme quasi-linéaire dans le support structurel et les autres paramètres sur un grand corps fini . . . 133
5.3. Division exacte de polynômes creux . . . 135
5.3.1. Adaptation de l'interpolation sur les entiers . . . 136
5.3.1.1. Les obstacles à la division . . . 136
5.3.1.2. Un algorithme quasi-linéaire . . . 138
5.3.2. Adapter des interpolations pour circuit arithmétique . . . 142
5.3.2.1. Permettre l'inversion modulo $x^p - 1$ . . . 142
5.3.2.2. Diviser grâce à des algorithmes d'interpolation . . . 143

6. Polynômes multivariés et substitution de Kronecker 148
6.1. Interpolation . . . 149
6.2. Arithmétique . . . 152

Conclusion 154

A. Code en SageMath pour mesurer la proportion de nombres premiers dans des suites arithmétiques 156

Liste des algorithmes

1. Triplet 33
2. ÉvaluationRéductionCyclique 42
3. CoefficientsDominants 47
4. ÉvaluationModulaire 48
5. CoefficientsDominantsCreux 51
6. ÉvaluationModulaireCreuse 57

7. Interpolation de Prony 62
8. Interpolation de Garg et Schost 68
9. Interpolation de Huang 72
10. ReconstructionMonômes 78
11. RemontéeDeRacine 80
12. Interpolation_BNM 81

13. TestDivisibilité 91
14. VérificationModulaire 97
15. VérificationModulaireEntière 99
16. VérificationSansExtension 106
17. VérificationdeKaminski 111
18. VérificationCreuse 116

19. ProduitNaïf 125
20. ProduitPolynômesEntiers 130
21. EssaisEtTests 132
22. ProduitStructurel 134
23. QuotientAvecBorneT 138
24. DivisionExacte 141
25. DivisionParInterpolation 144

26. RéductionÀ1Variable 148

Introduction

Aperçu historique des polynômes

L'apparition du polynôme comme objet symbolique tel que nous le connaissons aujourd'hui, peut être datée du XVI^{e} siècle. Si les problèmes algébriques et géométriques faisaient déjà intervenir des polynômes, ceux-ci n'étaient pas clairement formalisés. Jusqu'à cette époque, les mathématiciens n'utilisaient guère de variables symboliques et les résolutions de problèmes algébriques passaient plutôt par des preuves géométriques. En 1591, François Viète publie un traité [Vie91] proposant des écritures symboliques (des lettres) pour désigner à la fois les variables et les paramètres d'équations géométriques. Mais surtout il manipule ces expressions symboliques, ce qui amène à la résolution des problèmes géométriques qu'il considère dans un cadre général, sans avoir à reprendre la démarche si les valeurs numériques changent. L'écriture symbolique actuelle, avec des lettres de la fin de l'alphabet comme variables et des lettres du début de l'alphabet comme paramètres, vient plutôt de Descartes que de Viète. Cependant, en introduisant du formalisme, Viète a ouvert un nouveau champ à l'algèbre, basé sur la manipulation des polynômes comme objet à part entière.

Si les polynômes peuvent être étudiés pour eux-mêmes, ils restent fortement attachés à d'autres domaines. Dès l'origine ils sont liés à la géométrie, à tel point que Viète ne conçoit pas qu'il puisse exister des polynômes non homogènes, ceux-ci devant représenter une mesure géométrique en une dimension précise. Au delà des polynômes homogènes, les équations polynomiales permettent de définir la géométrie algébrique dont l'étude est riche et abondante [Har77 ; Per08 ; Dic+08]. Les polynômes permettent aussi d'étendre différentes structures algébriques, en particulier les extensions de corps finis [MP13].

Les possibilités théoriques qu'offrent les polynômes s'accompagnent d'applications pratiques. Par exemple, la géométrie algébrique permet l'étude des trajectoires de robots [Can88]. Les polynômes sont aussi très présents dans la théorie des codes : la famille des codes polynomiaux s'appuyant spécifiquement sur leurs propriétés [GRS22 ; Rot06 ; Dum+07]. De nombreux protocoles cryptographiques dépendent des polynômes et des structures algébriques qu'ils permettent de construire [KL07 ; Omo20]. Enfin, il ne faut pas oublier l'importance des polynômes dans la représentation du monde physique que ce soit pour décrire des lois, calculer des approximations de fonction ou interpoler des fonctions à partir de mesures. La première référence citée pour l'interpolation de polynômes creux [Pro95] est d'ailleurs un article portant sur la dynamique des fluides.

Dans le cadre du calcul formel

Dans tous ces domaines, il est nécessaire de représenter les polynômes et d'avoir des algorithmes de calcul effectifs. Il est intéressant de constater que l'introduction de la symbolique des polynômes a permis la mise en place de procédures plus efficaces que celles existantes pour résoudre des problèmes géométriques. Si aujourd'hui les procédures sont informatisées, la question de l'efficacité reste totalement d'actualité. Le degré des polynômes ou la taille des systèmes polynomiaux considérés peuvent être très vite importants. Les algorithmes restent-ils performants ? Sont-ils les meilleurs ? La recherche de solutions algorithmiques efficaces pour manipuler des polynômes et plus généralement tout objet mathématique (nombres, matrices, ...) constitue la base du calcul formel. Dans ce domaine, les produits sont des opérations centrales car de nombreuses opérations s'y réduisent [Bos+17 ; GG13]. Par exemple la division euclidienne, le pgcd, l'interpolation, l'évaluation multipoint de polynômes et la reconstruction rationnelle sont des problèmes qui se réduisent au produit de polynômes. Pour des matrices structurées (quasi-Toeplitz, quasi-Vandermonde, quasi-Cauchy) il est aussi possible de se ramener à des produits de polynômes lors de la résolution de problèmes d'algèbre linéaire [Pan01], par exemple le calcul du rang, du déterminant ou d'une solution d'un système linéaire.

Ainsi l'efficacité d'une seule opération, le produit de polynômes, impacte l'efficacité de nombreuses autres opérations. Pendant longtemps, l'algorithme naïf de multiplication de polynômes, de complexité quadratique, a semblé le seul à même d'effectuer cette opération. Kolmogorov émet même l'hypothèse que l'algorithme quadratique est le meilleur algorithme possible. Au début des années 60, Karatsuba dément cette hypothèse en décrivant un algorithme sous-quadratique pour le produit s'appuyant sur une méthode de type diviser pour régner [KO62]. À partir de cette date, de nombreux algorithmes sont très vite proposés pour améliorer encore plus la complexité jusqu'à ce que Nussbaumer obtienne en 1980 une complexité quasi-linéaire [Nus80]. Son algorithme adapte l'algorithme pour le produit d'entier de Schönhage et Strassen [SS71] qui s'appuie sur la transformée de Fourier rapide (FFT) [CT65]. Cette complexité dépend de l'anneau de définition des polynômes, mais Cantor et Kaltofen montrent en 1991 qu'elle peut être atteinte pour des polynômes définis sur toute algèbre [CK91]. La recherche d'optimisation se poursuit aujourd'hui encore en visant notamment à améliorer les facteurs logarithmiques [HH22].

En particulier pour les polynômes creux

Les améliorations évoquées jusqu'à présent concernent le temps de calcul d'un algorithme pour des polynômes représentés par le tableau de tous leurs coefficients. Est-il aussi possible d'obtenir une plus grande efficacité en changeant la représentation ? En particulier, la représentation par l'ensemble des coefficients peut être en elle-même inefficace. Si de nombreux coefficients du polynôme valent 0, le tableau stocke des informations inutiles qu'une personne écrivant le polynôme à la main se garderait bien de noter. En écrivant un polynôme, il est plus naturel de n'écrire que les *monômes significatifs*, ceux ayant un coefficient non nul. La représentation d'un polynôme par ces monômes signi-

ficatifs est ce qui est appelé la *représentation creuse* d'un polynôme. Sa taille dépend du nombre de monômes non nuls du polynôme. Plus ce nombre est faible, plus la représentation est efficace par rapport à la représentation *dense* par le tableau de tous les coefficients. Il est ainsi possible de manipuler des polynômes de plus grand degré puisque la représentation est plus compacte. Le degré du polynôme doit aussi être pris en compte dans la représentation creuse, cependant il est stocké en tant qu'entier pour chaque monôme non nul et donc seul son logarithme joue sur la taille du polynôme. Il est alors nécessaire de concevoir des algorithmes adaptés à cette représentation pour que cette réduction de taille permette aussi d'obtenir des gains en efficacité lors de calculs.

Pour le produit de polynôme creux, l'algorithme naïf est aussi quadratique. Mais quadratique par rapport au nombre de monômes non nuls, c'est-à-dire par rapport à la taille réduite. Il est donc plus efficace que le produit en représentation dense si les polynômes sont vraiment creux. Pour améliorer cet algorithme, l'attention s'est plutôt tournée vers des optimisations en pratique en s'appuyant sur la structure de tas ou la parallélisation [Joh74 ; MP09a ; MP11b]. Le produit peut aussi être effectué en s'appuyant à la fois sur les algorithmes pour des polynômes denses et sur ceux pour des polynômes creux pour essayer d'optimiser la complexité par rapport à la taille de polynômes [Roc11]. Les travaux de Cole et Hariharan [CH02] fournissent un algorithme quasi-linéaire par rapport au nombre de monômes non nuls pour le cas spécifique des polynômes à coefficients positifs. Cependant le recours à un nombre premier plus grand que le degré impacte fortement la complexité qui est cubique dans le logarithme du degré. Ce n'est qu'en 2015, qu'Arnold et Roche ont mis au point le premier algorithme de produit qui ne soit pas systématiquement quadratique [AR15]. Dans les cas favorables, cet algorithme est quasi-linéaire. Pour atteindre une telle complexité, ils se sont appuyés sur des algorithmes d'interpolation de polynômes creux. Ainsi le produit de polynômes creux se réduit à l'interpolation de polynômes creux. C'est similaire à ce qui se passe pour les polynômes denses où le produit utilise la FFT, une interpolation sur des racines de l'unité. Contrairement au cas dense, l'inverse cependant n'est pas vérifiée et aucune procédure ne réduit l'interpolation creuse au produit creux.

L'interpolation de polynômes creux a suscité également beaucoup d'intérêt et le développement de nombreux algorithmes. L'algorithme le plus ancien, celui de Prony [Pro95] redécouvert par Ben-Or et Tiwari [BT88] nécessite un nombre d'évaluations qui est seulement linéaire dans le nombre de monômes non nuls du polynôme à interpoler. Cependant, l'ensemble de l'algorithme est loin d'être linéaire surtout en considérant tous les paramètres qui définissent la taille d'un polynôme creux, en particulier par rapport au logarithme du degré. Différentes améliorations ont été proposées pour optimiser les calculs effectués à partir d'évaluations [KL88 ; KLW90 ; HL15] souvent en se restreignant à des cas particuliers [Kal10 ; Hua21] ou en modifiant légèrement le modèle d'évaluation [KN11 ; BJ14]. Une amélioration significative de la complexité totale de l'interpolation a été atteinte grâce à un changement complet de modèle proposé par Garg et Schost [GS09]. En considérant un polynôme creux donné par un circuit arithmétique et non par des évaluations, ils ont pu atteindre pour la première fois une complexité polynomiale dans le logarithme du degré de manière inconditionnelle pour effectuer l'interpolation de

polynôme creux. À la suite des travaux de Garg et Schost, d'autres algorithmes d'interpolation de circuit arithmétique ont été proposés. Ils permettent d'atteindre une complexité quasi-linéaire par rapport au nombre de monômes non nuls pour des polynômes creux sur les corps finis [AGR15] et aussi quasi-linéaire par rapport au logarithme du degré si la caractéristique du corps est plus grande que le degré [Hua19]. Nous avons évoqué précédemment les codes correcteurs comme usage des polynômes de manière générale. Il est justement possible d'en construire qui se basent spécifiquement sur l'interpolation creuse [KP14].

L'interpolation n'est pas la seule opération dont le calcul ne repose pas sur le produit de polynômes dans le cas d'une représentation creuse. C'est le cas de la division qui ne propose que quelques optimisations de structures pour améliorer sensiblement l'algorithme naïf [Joh74 ; MP11a]. Le calcul du pgcd est devenu pour cette représentation un problème bien plus complexe : Plaisted a montré que de simplement déterminer s'il est égal à 1 est NP-complet [Pla84]. Ses travaux démontrent également la difficulté d'autres problèmes liés à la factorisation. Ces trois opérations (division, pgcd et factorisation) peuvent entraîner d'importantes variations de taille durant les calculs et illustrent la difficulté majeure liée à la représentation creuse : obtenir des algorithmes polynomiaux dans la taille de la représentation [DC09]. La question de la factorisation de polynômes creux suscite également beaucoup d'intérêt, avec notamment les abondants travaux de Schinzel initiés dans les années 60, entre autres [Sch65 ; Sch69]. Plutôt que de chercher à factoriser complètement un polynôme creux, ce qui est aujourd'hui encore difficile, de nombreux algorithmes se restreignent à certains facteurs de petits degré [Len99a ; Len99b ; Gre16 ; Cha+21] ou de haut degré [GR11a] ou encore à la factorisation de polynômes d'une forme donnée [Fil+20].

Chacune des opérations évoquées est encore étudiée aujourd'hui et leur complexité n'est pas connue en toute généralité. Les questions ouvertes soulevées par ces opérations avant le début de mon doctorat (en 2019) sont résumées dans l'inventaire des algorithmes pour les polynômes creux dressé par Daniel S. Roche en 2018 [Roc18b]. Un point important et que, sauf pour l'addition, aucun algorithme quasi-linéaire n'est connu.

Les méthodes développées pour les polynômes creux peuvent ensuite être utilisées dans d'autres cadres, que ce soit pour considérer des fractions rationnelles creuses [GKS94 ; CL11 ; KY07], calculer des pgcd creux [HL21 ; HM16] ou résoudre des systèmes polynomiaux [CKY89].

Aperçu du contenu de cette thèse

Cette thèse se consacre à l'élaboration d'algorithmes efficaces pour travailler sur des polynômes en représentation creuse, plus communément appelés polynômes creux. Différentes opérations sont abordées ici : interpolation, vérification de produit, produit et division exacte. L'objectif de nos travaux est d'améliorer les complexités existantes pour chacune de ces opérations. En particulier, nous montrerons que dans le cas de polynômes à coefficients entiers nous obtenons des algorithmes quasi-linéaires.

Dans le chapitre 1 nous donnons les définitions utilisées pour l'ensemble du manuscrit ainsi que des résultats classiques en calcul formel nécessaires aux différents algorithmes. C'est aussi dans ce chapitre que nous introduisons une difficulté propre aux polynômes creux : si la représentation creuse permet d'optimiser la taille des polynômes, cette taille peut varier fortement dès lors qu'une opération, même relativement basique, est effectuée. Nous fournissons des exemples de telles croissances ainsi que des bornes supérieures.

Le chapitre 2 est un chapitre technique qui développe des méthodes algorithmiques générales pour contrôler la taille des polynômes. Puisque le point fort des polynômes creux est l'optimisation de leur taille, il ne faut pas que des calculs intermédiaires génèrent des polynômes de trop grande taille. Pour autant en limitant la taille de tous les calculs, il est possible de perdre de l'information. Dans ce chapitre, nous montrerons comment il est possible d'équilibrer ces deux contraintes. Une approche habituelle consiste à considérer les polynômes modulo un polynôme tiers, généralement $x^p - 1$. Plusieurs points sont abordés : comment choisir le polynôme pour réduire le degré sans perdre trop d'information, comment le choisir pour qu'il ait des propriétés arithmétiques précises qui facilitent la suite des calculs et aussi comment procéder efficacement à l'évaluation de produits modulo ce polynôme tiers.

Les chapitres suivants se consacrent véritablement à des opérations sur des polynômes creux et constituent nos contributions majeures dans cette thèse.

Dans le chapitre 3, l'opération que nous abordons est l'interpolation. Le cœur de ce chapitre est le premier algorithme quasi-linéaire pour l'interpolation d'un polynôme creux. Cet algorithme ne s'applique qu'à des polynômes définis sur des entiers et repose notamment sur l'existence d'un morphisme entre les entiers et les entiers modulaires pour tout modulo. Cette souplesse apportée par les anneaux des entiers permet de changer d'anneaux durant l'algorithme d'interpolation pour ajuster à tout instant la taille des données manipulées. En plus de contrôler la taille, notre algorithme combine subtilement les techniques des deux principales approches d'interpolation (l'interpolation à partir d'évaluations ou à partir d'un circuit arithmétique).

Le chapitre 4 aborde plutôt des tests pour déterminer la validité d'expression sur des polynômes creux ou denses. Nous y présentons une famille non triviale de polynômes creux pour laquelle nous avons mis au point un algorithme déterministe pour tester la divisibilité. L'existence d'un test de divisibilité entre deux polynômes creux en temps polynomial reste une question ouverte à l'heure actuelle. Le test que nous présentons repose sur la structure du potentiel diviseur et la présence d'écarts importants entre certains de ses monômes. En plus de la divisibilité, nous traitons aussi la vérification d'un produit de polynômes, creux ou denses, potentiellement modulo un polynôme creux. Les algorithmes proposés reposent sur l'évaluation d'un produit modulaire. Ils visent la plus grande efficacité, sont probabilistes et quasi-linéaires.

Enfin, le chapitre 5 est consacré à des opérations arithmétiques. Il s'agit du produit et de la division avec reste nul entre deux polynômes. Ce chapitre s'appuie sur les deux précédents. La difficulté de ces opérations pour des polynômes creux c'est que le nombre

de monômes non nuls du résultat est imprédictible. Il peut varier de constant à quadratique voire exponentielle pour la division sans qu'il soit possible de donner une borne fine a priori. Pour des polynômes à coefficients entiers, le problème se retrouve aussi pour la valeur des coefficients. Notre approche consiste à interpoler le résultat de l'opération avec une taille fixée. Si cette taille est trop faible, le résultat obtenu est erroné et nos algorithmes de vérification peuvent le détecter. Le processus est alors répété en fixant une taille plus grande. Pour pouvoir interpoler une opération (produit, division), il convient d'adapter soigneusement les algorithmes d'interpolation pour tenir compte des spécificités de l'opération en particulier dans le cas de la division. Pour des polynômes à coefficients entiers, notre approche permet d'obtenir des algorithmes quasi-linéaires.

Dans tous ces chapitres, les polynômes considérés n'ont qu'une seule variable, le chapitre 6 considère des polynômes en plusieurs variables. En particulier, nous verrons qu'il est possible d'adapter des algorithmes avec une seule variable pour des polynômes à plusieurs variables en utilisant la substitution de Kronecker. Cette adaptation et son effet sur la complexité sont discutés. Nous montrerons que les opérations arithmétiques abordées restent quasi-linéaires lorsque les polynômes sont à coefficients entiers.

1. Préliminaires

Ce premier chapitre est consacré à des définitions et résultats élémentaires concernant l'arithmétique et les polynômes. Il permet aussi de fixer les notations utilisées dans l'ensemble du document.

Il est d'abord question, en section 1.1, du cadre dans lequel les différents algorithmes présentés seront analysés. À savoir que l'analyse s'attachera à déterminer la complexité binaire des algorithmes. Parfois, les opérations comptées pourront ne pas être des opérations binaires, ce sera alors clairement explicité. Les algorithmes seront généralement probabilistes et décrits avec leur probabilité de succès. Si le temps de calcul est lui-même probabiliste, cette probabilité est aussi indiquée, sinon le temps de calcul est évalué dans le pire cas.

L'élément d'étude de cette thèse, le polynôme creux, est défini formellement en section 1.2. Il s'agit de la représentation d'un polynôme par ses seuls monômes non nuls. Cette représentation a une taille binaire spécifique. Le nombre des monômes non nuls doit notamment être pris en compte pour l'évaluer. La plus grande partie de cette thèse se restreint au cas des polynômes à une seule variable. Les opérations sur des polynômes à plusieurs variables se réduisent à des opérations sur des polynômes univariés.

En section 1.3 il est question d'arithmétique classique sur les entiers et les polynômes denses. Autant d'opérations qui sont nécessaires au développement des algorithmes de cette thèse.

Enfin, la section 1.4 est dédiée à des observations basiques sur des opérations entre polynômes creux. Il est question des changements de taille que ces opérations entraînent, notamment en faisant varier le nombre de monômes non nuls.

Un lecteur familier avec les polynômes creux pourra sans difficulté s'abstenir de lire ce chapitre. À noter que la section 1.4 prouve l'existence de certaines bornes sur les tailles qui n'avaient pas été aussi détaillées auparavant.

1.1. Analyse d'algorithmes probabilistes

Dans cette thèse, l'analyse des algorithmes sera généralement effectuée en comptant le nombre d'opérations binaires effectuées. Ainsi des opérations sur des éléments définis différemment pourront être comparées. Pour alléger les notations, les facteurs logarithmiques pourront être indiqués seulement par l'écriture $\tilde{\mathcal{O}}$. Ainsi pour une fonction ϕ, $\tilde{\mathcal{O}}(\phi)$ correspond à $\mathcal{O}(\phi(\log \phi)^k)$ pour une constante k.

Puisque l'analyse se base sur la complexité binaire et que de même les tailles considérées sont des tailles binaires, toutes les conventions iront dans ce sens. En particulier $\log n$ désignera toujours le logarithme de n en base 2. Toute autre base sera clairement indiquée : $\log_b$ pour un logarithme en base b et ln pour le logarithme népérien.

La plupart des algorithmes de cette thèse sont des algorithmes probabilistes. Ceux-ci se basent sur l'existence de générateurs d'entiers aléatoires. Ils permettent de tirer aléatoirement un entier dans un intervalle donné avec une distribution uniforme. Un tel générateur n'existe pas. En revanche il existe des générateurs pseudo-aléatoires dont la distribution est difficile à distinguer de la distribution uniforme. Ce qui donne une bonne approximation d'un générateur aléatoire et justifie que l'on puisse considérer des distributions uniformes pas seulement d'un point de vue théorique. Dans ce manuscrit, si un nombre est dit choisi ou tiré aléatoirement dans un ensemble $\mathcal{E}$, cela sous-entendra toujours que c'est effectué selon la loi uniforme sur $\mathcal{E}$.

Il existe différents types d'algorithmes probabilistes. La question est de savoir ce qui est probabiliste entre le résultat et le temps de calcul.

Définition 1. Un algorithme probabiliste est dit de type *Monte Carlo* si le résultat qu'il renvoie est probabiliste mais que son temps de calcul est déterministe.

Un algorithme probabiliste est dit de type *Las Vegas* si le résultat qu'il renvoie est toujours correct mais que son temps de calcul est probabiliste.

Un algorithme probabiliste est dit de type *Atlantic City* si le résultat qu'il renvoie est probabiliste et que son temps de calcul est probabiliste.

Dans les deux derniers cas, un temps de calcul probable avec la probabilité correspondante sera fournie lors de l'analyse de l'algorithme. Dans le cas d'un algorithme de type Monte Carlo en revanche, l'analyse du temps de calcul sera effectuée dans le pire cas.

1.1.1. Calculs de probabilités

Pour simplifier les calculs de probabilités, l'inégalité de Boole sera souvent utilisée.

Fait 1.1. *Pour deux évènements A et B,* $\Pr[A \text{ ou } B] \leq \Pr[A] + \Pr[B]$.

En particulier si un algorithme a deux sources d'erreurs avec probabilités ϵ_1 pour la première et ϵ_2 pour la seconde, la probabilité de succès de l'algorithme est supérieure à $1 - \epsilon_1 - \epsilon_2$.

Pour améliorer la probabilité de succès d'un algorithme de type Monte Carlo, il suffit de le répéter suffisamment souvent. Il y a alors deux possibilités.

La première est que l'algorithme soit biaisé d'un seul côté. Autrement dit le résultat de l'algorithme peut se comparer avec le résultat exact et cette comparaison est toujours dans le même sens. Le résultat de l'algorithme Monte Carlo est soit forcément plus petit, soit forcément plus grand que le résultat est exact. Si la sortie est un booléen, cela signifie

en particulier que l'algorithme ne renverra pas de faux positif (respectivement de faux négatif). Dans ce cas là, parmi les résultats de k répétitions il est possible de choisir le meilleur. En particulier, ce sera la réponse correcte si elle a été obtenue au moins une fois.

Fait 1.2. *Répéter k fois et de manière indépendante un algorithme Monte Carlo biaisé d'un seul côté avec une probabilité de succès de p permet de ramener la probabilité d'échec à $(1-p)^k$.*

Démonstration. Puisque l'algorithme n'est biaisé que d'un côté, pour qu'il y ait échec après k répétitions il faut qu'il y ait échec à chacune des répétitions. Le résultat vient de l'indépendance des répétitions. □

En particulier, pour atteindre une probabilité d'échec ϵ, le nombre de répétitions à effectuer est de $\log_{(1-p)} \epsilon = (\log \frac{1}{\epsilon})/\log \frac{1}{1-p}$. Pour une probabilité p constante, cela correspond à $\mathcal{O}(\log \frac{1}{\epsilon})$ répétitions.

Dans la seconde possibilité, l'algorithme peut se tromper dans les deux sens. Il n'est donc pas possible de déterminer la meilleure solution comme étant la plus petite (ou la plus grande). En revanche, après suffisamment de répétitions, si un élément est majoritaire parmi les résultats obtenus, alors c'est probablement le résultat exact.

Fait 1.3. *Répéter k fois et de manière indépendante un algorithme Monte Carlo avec une probabilité de succès de p et choisir l'élément majoritaire parmi les résultats obtenus permet de ramener la probabilité d'échec à $0 < \epsilon < 1$ en prenant $k = \frac{8p}{(2p-1)^2} \ln \frac{1}{\epsilon}$.*

Démonstration. Soit S, la variable aléatoire qui compte le nombre de fois où l'algorithme a été exécuté avec succès et a bien renvoyé le résultat exact. Puisque la probabilité de succès d'une exécution de l'algorithme est p et que les répétitions sont indépendantes, $\mathbb{E}[S] \geq pk$. Ainsi, en appliquant les bornes de Chernoff, la probabilité que le résultat soit produit par moins de la moitié des exécutions (et donc qu'il y ait échec malgré les répétitions) est

$$\Pr[S \leq \frac{k}{2}] = \Pr[S \leq (1-\frac{2p-1}{2p})\mathbb{E}[S]] \leq \exp(-\left(\frac{2p-1}{2p}\right)^2 \mathbb{E}[S]/2) \leq \exp(-\frac{(2p-1)^2}{8p}k).$$

Pour $k = \frac{8p}{(2p-1)^2} \ln \frac{1}{\epsilon}$, cette probabilité est bien inférieure à ϵ. □

Dans les deux cas, le nombre de répétitions est linéaire en $\log \frac{1}{\epsilon}$.

1.2. Polynômes denses ou creux

1.2.1. Représentation et taille des polynômes

Avant toute chose et notamment d'analyser des algorithmes ayant pour objet des polynômes creux, il convient de définir ce qu'est un polynôme creux. Et aussi de clarifier

la distinction avec les polynômes denses. Dans l'ensemble de ce manuscrit, un polynôme $F = \sum_{i=0}^{d} f_i x^i$ n'est pas en lui-même un polynôme creux. Cette dénomination indique une particularité de la représentation du polynôme et non une caractéristique propre au polynôme. La représentation creuse d'un polynôme s'attache uniquement aux monômes non nuls de F. Elle correspond à ce qu'une personne qui écrirait à la main noterait comme informations sur le polynôme. Tous les zéros étant des informations superflues.

Définition 2. La *représentation creuse* d'un polynôme $F = \sum_{i=0}^{d} f_i x^i$ est la liste des paires coefficients, exposants des monômes non nuls de F : $\{(f_i, i) | f_i \neq 0\}$.

Un *polynôme creux* est un polynôme décrit par une représentation creuse.

Cette définition n'est pas universelle. Il arrive aussi que l'appellation polynôme creux désigne un polynôme n'ayant que peu de monômes non nuls par rapport à son degré. Le terme *lacunaire* peut aussi être utilisé dans ce cas. Dans cette acception du terme, il s'agit en quelque sorte de considérer comme creux des polynômes dont la représentation creuse utilise moins d'espace que la représentation dense. La représentation dense quant à ellestocke tous les coefficients du polynôme.

Définition 3. La *représentation dense* d'un polynôme $F = \sum_{i=0}^{d} f_i x^i$ est le tableau de tous les coefficients de F : $[f_0, f_1, \ldots, f_d]$.

Un *polynôme dense* est un polynôme décrit par une représentation dense.

La représentation dense correspond plus généralement à la représentation d'un vecteur. La représentation creuse de manière similaire correspond à un *vecteur creux* dont les coordonnées non nulles sont stockées avec leur indice.

La taille de ces deux représentations diffère. Le nombre de monômes non nuls de F comparé à son degré permet de savoir laquelle est la plus efficace. La taille exacte de la représentation dépend aussi de la taille des coefficients. Les coefficients sont des éléments d'un anneau $\mathbb{A}$. De manière générale, $\mathbb{A}$ sera considéré comme un anneau commutatif intègre, en particulier l'anneau $\mathbb{Z}$ des entiers ou un corps fini $\mathbb{F}_q$.

Fait 1.4. *Soit F un polynôme dans $\mathbb{A}[x]$. Soit d le degré de F, t son nombre de monômes non nuls et B la taille maximale en bits des coefficients de F.*

- *La taille de la représentation creuse de F est $t(B + \lceil \log(\max(2, d+1)) \rceil)$ bits.*
- *La taille de la représentation dense de F est dB bits.*

Par *taille*, c'est toujours la taille binaire qui sera désignée. Ce niveau de précision est particulièrement nécessaire pour la représentation creuse. En effet celle-ci met au même niveau des éléments de $\mathbb{A}$, les coefficients et des éléments de $\mathbb{Z}$, les exposants. Ceux-ci ne peuvent être véritablement comparés qu'en allant au niveau de la représentation binaire.

Par rapport à la représentation dense, la représentation creuse fait intervenir un paramètre supplémentaire. Le nombre de monômes non nuls de F.

Définition 4. L'ensemble des exposants des monômes non nuls d'un polynôme $F = \sum_{i=0}^{d} f_i x^i$ forme le *support* de F : $\text{support}(F) = \{i | 0 \leq i \leq d \,\text{et}\, f_i \neq 0\}$.

La cardinalité du support d'un polynôme F sera noté $\#F$. S'il n'y a pas de risque de confusion, la lettre t sera aussi utilisée, en préférant la majuscule T pour une borne sur la cardinalité du support.

La taille des représentations dépend aussi de la taille des coefficients eux-mêmes. Cette taille dépend de $\mathbb{A}$, l'anneau de définition du polynôme. Celui-ci peut n'avoir qu'un nombre fini d'éléments (par exemple sur un corps fini ou des entiers modulaires). Il est aussi possible que ce soit un anneau infini. Il convient alors de considérer une borne sur les coefficients de F. C'est le cas notamment pour les polynômes dans $\mathbb{Z}[x]$.

Définition 5. La *hauteur* d'un polynôme $F = \sum_{i=0}^{d} f_i x^i$ dans $\mathbb{Z}[x]$ est le maximum des valeurs absolue de ses coefficients. Elle est notée $\|F\|$: $\|F\| = \max(\{|f_i|, 0 \leq i \leq d\})$.

Si un polynôme creux peut être formellement défini sur tout anneau $\mathbb{A}$, $\mathbb{A}$ doit respecter certaines conditions pour permettre l'utilisation informatique du polynôme F. Tout d'abord, les coefficients de F, éléments de $\mathbb{A}$ doivent disposer d'une représentation finie. La taille du polynôme F ne peut d'ailleurs être définie que dans ce cas. Il faut aussi que $\mathbb{A}$ soit un anneau effectif : qu'il existe un algorithme qui puisse calculer en temps fini toute opération sur des éléments de taille finie.

1.2.2. Structure de données

En plus d'une définition formelle, il faut ajouter plus de structure à la représentation creuse pour que son utilisation soit efficace en pratique. Cette structure doit répondre aux besoins des algorithmes sur les polynômes creux. Il y a principalement deux besoins distincts :

1. L'accès direct au coefficient du monôme de degré e, étant donné un exposant e quelconque.
2. Le parcours de tous les monômes par ordre croissant ou décroissant des exposants.

Dans les deux cas, il s'agit de rendre possible aussi bien la lecture que l'écriture.

Pour répondre au premier besoin, un dictionnaire avec pour clés les exposants fournit une structure adaptée. L'accès au coefficient du monôme de degré e ne nécessite que $\mathcal{O}(\log e)$ opérations, pour la lecture de la clé. Cependant les monômes ne sont pas ordonnés. Un dictionnaire ne permet pas de répondre efficacement au second besoin.

Une liste triée par exposants croissants (ou décroissants) permet de parcourir l'ensemble des monômes avec un coût constant pour passer d'un monôme à l'autre. Dans ce

cas l'accès au coefficient du monôme d'exposant e s'effectue en parcourant l'ensemble de la liste. Pour un polynôme de T monômes non nuls et de degré D, cela nécessite $\mathcal{O}(T)$ comparaisons d'entiers de $\log D$ bits.

Pour gérer simultanément les deux besoins de manière efficace, d'autres structures sont plus adaptées. Si les monômes sont stockés dans un tas, il est ainsi possible d'accéder (potentiellement supprimer ou insérer) au monôme d'exposant e en seulement $\mathcal{O}(\log T)$ comparaisons tout en préservant un certain ordre entre les monômes. Ceci ne permet cependant pas de garantir l'efficacité du parcours à chaque instant. En s'appuyant plutôt sur un arbre rouge/noir, les opération d'accès, d'insertion, de suppression d'un monôme aussi bien que de recherche du monôme suivant ou précédant nécessitent toutes $\mathcal{O}(\log T)$ comparaisons. Les arbres de Van Emde Boas creux permettent d'implémenter les mêmes opérations mais avec un dépendance dans le degré D, en $\mathcal{O}(\log \log D)$. Ces structures plus complexes sont présentées en détails dans le livre *Introudction to Algorithms* [Cor+09], mais ne sont pas nécessaires pour représenter les polynôme creux dans les algorithmes présentés dans cette thèse.

La plupart des algorithmes sur les polynômes creux n'ont pas simultanément besoin d'accéder à un monôme donné et de parcourir l'ensemble des monômes d'un polynôme. Il est ainsi possible de considérer qu'un polynôme creux F est donné comme un dictionnaire dont les clés sont les exposants et les valeurs sont les coefficients avec en plus la liste des exposants. Toutes les opérations par accès direct à un monôme passent directement par le dictionnaire en ajoutant des exposants à la liste d'exposants le cas échéant. Avant tout parcours des monômes de F, la liste d'exposants est triée, en temps linéaire puisqu'il s'agit d'une liste d'entiers.

Tout au long de la thèse, il sera considéré que la structure sous-jacente à la représentation creuse, permet de répondre aux deux besoins évoqués de manière efficace.

Fait 1.5. *Pour un polynôme creux F, l'accès à*

- *un monôme d'un degré spécifique,*
- *le monôme non nul qui le précède,*
- *ou le monôme non nul qui le suit*

nécessite asymptotiquement autant d'opérations que l'écriture du monôme concerné.

Le choix de la structure précise aura un impact sur l'efficacité des algorithmes. Ceci a été étudié notamment par Richard Fateman [Fat03] pour la multiplication de polynômes creux.

1.2.3. Restriction aux polynômes univariés

Les polynômes présentés jusqu'à présent sont tous univariés. Pourtant de très nombreuses applications font intervenir des polynômes à plusieurs variables. Ceux-ci sont même régulièrement représentés sous forme creuse pour limiter la taille des représentations. Comme dans le cas univarié, en fonction du nombre de monômes non nuls, la

représentation dense peut être exponentiellement plus grande que la creuse. En effet, un polynôme sur n variables de degré d en chacune des variables est stocké par un tableau n-dimensionnel de taille d^n sous forme dense. Il est fréquent que d^n soit bien plus grand que la cardinalité t de son support. Or un terme $\alpha \prod_{i=0}^{n-1} x_i^{e_i}$ peut être représenté par son coefficient α et le n-uplet $(e_0, \ldots, e_{n-1})$. Ainsi, sous forme creuse il suffit de stocker les t coefficients non nuls plus leur n-uplet d'exposants. Le stockage des exposants prenant une taille binaire de $tn \log d$.

Il est possible de transformer un polynôme multivarié en polynôme univarié sans augmenter la taille de sa représentation en utilisant une transformation de Kronecker.

Définition 6. La *substitution de Kronecker* est la bijection K_D entre les polynômes dans $\mathbb{A}[x_0, \ldots, x_{n-1}]$ de degré strictement inférieur à D en chaque variable et les polynômes dans $\mathbb{A}[x]$ de degré strictement inférieur à D^n, définie par $K_D(x_i) = x^{D^i}$ et étendue par linéarité.

Le polynôme obtenu par la substitution a le même nombre de monômes que le polynôme d'origine et un degré borné par D^n. Si D, n'est pas trop grand par rapport à d, $D = \mathcal{O}(d)$, la taille du polynôme univarié obtenu est essentiellement la même que celle du polynôme multivarié. En dense, il est stocké comme tableau de longueur $\mathcal{O}(d^n)$. En creux la liste des coefficients, exposants est toujours de taille t, avec des exposants borné par D^n. Le stockage des exposants nécessite donc $\mathcal{O}(n \log d)$ bits.

Par ailleurs, si une opération est interne au sous-ensemble de $\mathbb{A}[x_0, \ldots, x_{n-1}]$ considéré, autrement dit que le degré en chaque variable du résultat est strictement inférieur à D, K_D se comporte comme un morphisme pour cette opération. Ainsi en prenant pour D une borne appropriée, une suite d'opérations sur des polynômes multivariés peut se réduire à une suite d'opérations sur des polynômes univariés.

Ces constats justifient que la plus grande part de la présentation dans ce manuscrit se consacre exclusivement au cas des polynômes univariés. La substitution de Kronecker soulève cependant certaines difficultés qui seront présentées au chapitre 6.

1.3. Outils de base

Cette partie rappelle des résultats classiques concernant les entiers et polynômes denses. Ceux-ci sont fréquemment manipulés dans les algorithmes présentés dans ce document. Certaines opérations élémentaires sont donc nécessaires. Il s'agit d'opérations arithmétiques basiques en section 1.3.1 ou de la génération d'éléments irréductibles en section 1.3.2.

1.3.1. Opérations élémentaires

1.3.1.1. Arithmétique d'entiers et de polynômes denses

Avant de s'intéresser à l'arithmétique des polynômes creux, il convient de maîtriser les opérations élémentaires sur leurs coefficients. Les coefficients sont des éléments d'un anneau $\mathbb{A}$. La définition de cet anneau n'a pas besoin d'être spécialement contrainte dans un cadre théorique général. Il peut être simplement requis que ce soit un anneau effectif. Dans cette thèse il sera aussi considéré le plus souvent comme commutatif et intègre ce qui exclut en particulier les anneaux de matrices. Lors des analyses les plus précises, les opérations seront réalisées sur les entiers, les entiers modulaires et des anneaux de polynômes denses ce qui permet entre autre de considérer tous les corps finis. Dans tous ces anneaux, l'addition est une opération linéaire. L'opération essentielle est plutôt la multiplication. Puisque la complexité de nombreuses autres opérations se ramène à la complexité de la multiplication, il convient de désigner de manière unique ces complexités et de les donner précisément.

Fait 1.6. *Soit b, d, H des entiers strictement positifs et q une puissance d'un nombre premier.*

- $\mathsf{I}(b) = \mathcal{O}(b \log b)$ *est le coût binaire du produit d'entiers d'au plus b bits [HH21, Theorem 1.1].*
- $\mathsf{M}(d) = \mathcal{O}(d \log d \log\log d)$ *est le nombre d'opérations dans $\mathbb{A}$ pour un produit de polynômes de degré au plus d dans $\mathbb{A}[X]$ [CK91].*
- $\mathsf{M}_{<H}(d) = \mathsf{I}(d(\log d + \log H)) = \mathcal{O}(d(\log d + \log H)(\log d + \log\log H))$ *est le coût binaire du produit de polynômes de degré au plus d dans $\mathbb{Z}[X]$ de hauteur strictement inférieure à H. [GG13, Section 8.4].*
- $\mathsf{M}_q(d) = \mathcal{O}(d \log q \log(d \log q) 4^{\log^*(d)})$ *est le coût du produit de polynômes de degré au plus d dans $\mathbb{F}_q[x]$ [HH19].*
- $\mathsf{M}_q(d) = \mathcal{O}(d \log q \log(d \log q))$ *en admettant l'existence d'une constante de Linnik $L < 1 + 2^{-1162}$ [HH22, Theorem 1.2].*

On peut remarquer que le produit de polynômes à coefficients entiers se réduit à un produit d'entiers. La méthode utilisée pour calculer ce ce produit est l'évaluation des polynômes en une puissance de 2 suffisamment grande pour que les coefficients puissent être directement vu comme les chiffres des évalués exprimés dans cette base. Ainsi il faut que la puissance de 2 soit supérieure à dH^2, la hauteur du produit pour pouvoir lire les coefficients du produit de polynômes dans les chiffres du produit des évalués. Les entiers obtenus par l'évaluation ont alors environ $d(\log d + \log H)$ bits. En fonction de la valeur de H, il peut être plus efficace d'utiliser le théorème des restes chinois et de se ramener au calcul de produits de polynômes dans des petits corps finis. Dans les deux cas, la complexité est quasi-linéaire en $d \log H$, la taille des polynômes impliqués.

Dans tous ces anneaux, le coût de la division euclidienne est asymptotiquement le même que celui du produit. Ainsi le coût d'un produit sur les entiers modulaires (ou plus généralement dans une extension de type $\mathbb{A}[x]/P(x)$) n'a pas besoin d'être précisé

indépendamment. En effet, il s'agit simplement d'effectuer d'abord un produit puis une division d'entiers. Il est possible d'être plus précis pour la division euclidienne d'entiers de taille différentes. En effet, si la taille b du diviseur est plus petite que la taille c du dividende la division peut se ramener à $\frac{c}{b}$ divisions en taille b ce qui donne un coût binaire de $\mathcal{O}(\frac{c}{b}\mathsf{I}(b)) = \mathcal{O}(c \log b)$.

1.3.1.2. Réduction modulaire et évaluation

La division euclidienne mène à une autre opération souvent utilisée dans cette thèse : la réduction modulaire. Si le modulo est quelconque, passer par la division euclidienne est le plus efficace. En revanche pour certaines formes de modulo, il est possible d'être plus rapide. En particulier pour un polynôme de la forme $x^d - 1$.

Fait 1.7. *Soit F un polynôme de degré D dans $\mathbb{A}[x]$ ayant au plus T monômes non nuls et $P = x^d - 1$. Le calcul de $F \bmod P$ nécessite*

- *$\mathcal{O}(D)$ additions dans $\mathbb{A}$ si F est un polynôme dense,*
- *$\mathcal{O}(T)$ additions dans $\mathbb{A}$ plus $T\frac{\log D}{\log d}\mathsf{I}(\log d)$ opérations binaires si F est un polynôme creux.*

Démonstration. Réduire modulo $x^d - 1$ revient simplement à réduire les exposants des différents monômes modulo d et à additionner les coefficients des monômes qui se trouvent avoir des exposants égaux modulo d.

Pour un polynôme dense, la réduction s'effectue sans division en découpant simplement le tableau des coefficients de F en sous-tableaux de taille d. L'addition terme à terme de ces tableaux fournit alors les coefficients de $F \bmod x^d - 1$.

En représentation creuse, les exposants sont réduits modulo d par des divisions entières. □

De manière similaire, il est possible de calculer en $\mathcal{O}(b)$ opérations binaires, la réduction d'un entier de b-bits modulo $2^i - 1$.

Enfin, si la réduction est effectuée modulo $x - \alpha$, il s'agit simplement de l'évaluation du polynôme en α. Une opération pour laquelle il existe des algorithmes spécifiques.

Fait 1.8. *L'évaluation d'un polynôme $F \in \mathbb{A}[x]$ de degré d ayant T monômes non nuls sur un point α d'une extension $\mathbb{A}_{ext}$ de $\mathbb{A}$ nécessite*

- *$\mathcal{O}(d)$ opérations dans $\mathbb{A}_{ext}$ si F est un polynôme dense,*
- *$\mathcal{O}(\log d + T \log d / \log\log d)$ opérations dans $\mathbb{A}_{ext}$ si F est un polynôme creux, en utilisant l'exponentiation simultanée [Yao76].*

Pour l'évaluation d'un polynôme sous forme creuse, c'est généralement la complexité $\mathcal{O}(T \log d)$ qui sera utilisée. Elle simplifie les notations sans augmenter la complexité de manière significative. Cette complexité correspond à un algorithme d'évaluation qui effectue une exponentiation rapide pour chaque monôme de F.

1.3.2. Génération d'irréductibles dans un anneau

Il est souvent utile de pouvoir générer un élément irréductible, que ce soit un nombre premier ou un polynôme irréductible. De nombreuses procédures, notamment les algorithmes présentés dans cette thèse, utilise des éléments irréductibles. La question de la génération de tels éléments est un problème classique dont les solutions efficaces sont rappelées ici.

1.3.2.1. Générer des nombres premiers

La génération de nombres premiers peut avoir deux objectifs distincts : soit en générer un grand nombre de tailles plutôt petites, soit en trouver un seul de grande taille. Des algorithmes bien distincts sont utilisés dans les deux cas.

Pour générer un grand nombre de premiers, il convient plutôt de s'appuyer sur un crible. Le plus ancien est dû à Ératosthène durant l'antiquité, mais le plus efficace est celui de la roue.

Fait 1.9 ([Pri82])**.** *Pour $n \in \mathbb{N}$, la liste des nombres premiers inférieurs à n peut être calculée en $\mathcal{O}(n \log n / \log\log n) = \tilde{\mathcal{O}}(n)$ opérations binaires.*

Il est possible de passer à la construction d'une certaine quantité N de nombres premiers en se basant sur la densité des nombres premiers parmi les entiers.

Fait 1.10 ([RS62, Theorem 3])**.** *Pour un entier $N \geq 20$, le N-ième nombre premier est compris entre $N(\log N + \log\log N - 3/2)$ et $N(\log N + \log\log N - 1/2)$.*

Corollaire 1.11. *Pour $N \in \mathbb{N}$, il est possible de calculer la liste des N plus petits nombres premiers en $\mathcal{O}(N \log^2 N / \log\log(N \log N)) = \tilde{\mathcal{O}}(N)$ opérations binaires.*

La méthode est quasi-linéaire dans le nombre N de premiers désirés et dans la taille binaire de la liste générée qui est en $\mathcal{O}(N \log N)$.

Si l'objectif est de fournir un seul nombre premier de grande taille, utiliser un crible n'est plus une solution efficace. En effet le temps de calcul est quasi-linéaire par rapport à la valeur du plus grand nombre premier construit, donc exponentiel par rapport à sa taille. L'approche consiste alors à travailler directement dans la taille désirée en cherchant le nombre premier dans un intervalle $[\lambda, 2\lambda]$ adapté. Pour cela, il faut s'appuyer sur la densité des nombres premiers au sein d'un intervalle.

Fait 1.12 ([RS62, Corollary 3])**.** *Pour un entier $\lambda \geq 21$, il y a au moins $\frac{3}{5}\lambda / \ln\lambda$ nombre premiers dans l'intervalle $[\lambda, 2\lambda]$.*

Sachant cela, un nombre choisi aléatoirement parmi les nombres impairs de l'intervalle $[\lambda, 2\lambda]$ est premier avec probabilité au moins $\frac{3}{10 \ln\lambda}$. Il existe des tests de primalité déterministe [AKS04] en $\tilde{\mathcal{O}}(\log^{10.5}\lambda)$ ou probabiliste [Rab80] en $\tilde{\mathcal{O}}(\log^2\lambda)$ pour vérifier si un nombre impair ainsi obtenu. En répétant cette démarche de choix aléatoire suivi d'un test, il est possible d'obtenir un nombre premier avec une probabilité aussi grande

que souhaitée. De tels algorithmes sont classiques [Sho08, Chapter 9]. Pour avoir une meilleure complexité, il faut se baser sur le test probabiliste de Miller–Rabin et accepter que l'algorithme puisse renvoyer un nombre qui ne soit pas premier.

Fait 1.13. *Il existe un algorithme* PremierAléa(λ, ϵ) *qui étant donné un entier* λ *et une probabilité* $0 < \epsilon < 1$ *renvoie un entier* q *aléatoirement choisi dans l'intervalle* $[\lambda, 2\lambda]$ *et qui est premier avec une probabilité d'au moins* $1 - \epsilon$*. Cet algorithme s'effectue en* $\mathcal{O}(\log(\frac{1}{\epsilon}) \log^2(\lambda) \mathsf{I}(\log \lambda) \log\log \lambda)$ *opérations binaires.*

La version de l'algorithme considérée est une version Monte Carlo qui s'arrête au bout d'un certain nombre d'essais même si le test ne répond pas que le nombre est premier. Une version Las Vegas est aussi possible. Si elle se base sur le test de Miller–Rabin, il s'agit plutôt d'un algorithme de type Atlantic City qui renvoie une réponse probablement correcte dans un temps probablement polynomial en $\log \lambda$. Une approche de type Las Vegas s'accorde mieux avec un test déterministe. De tels tests sont cependant plus lents tout en restant polynomiaux en $\log \lambda$.

Remarque 1.14. En utilisant le test déterministe AKS [AKS04], il est possible d'obtenir un algorithme de type Las Vegas qui renvoie toujours un nombre premier et dont l'espérance du temps de calcul est polynomiale dans le logarithme de λ.

1.3.2.2. Générer des polynômes irréductibles

Un problème très proche est celui de la génération de polynômes irréductibles. Dans cette thèse cette question ne se posera que dans le cadre des polynômes univariés définis sur un corps fini. Les résultats qui suivent concernent donc exclusivement ce cadre. La démarche est la même que pour la génération d'un nombre premier dans un intervalle : générer un polynôme unitaire aléatoire d'un certain degré d, tester s'il est irréductible et recommencer jusqu'à avoir un polynôme irréductible ou jusqu'à avoir dépassé un certain nombre d'essais. Encore une fois, la validité de la méthode repose sur la densité des polynômes irréductibles parmi les polynômes unitaires de degré d.

Fait 1.15 ([Sho08, Chapter 19])**.** *Le nombre de polynômes irréductibles unitaires de degré* d *dans* $\mathbb{F}_q[x]$ *est compris entre* $\frac{q^d}{2d}$ *et* $\frac{q^d}{d}$*.*

La performance de l'algorithme de génération du polynôme irréductible est donnée pour sa version Monte Carlo même si Shoup [Sho08] en a présenté une version Las Vegas. De plus le coût est différencié en fonction de l'efficacité des algorithmes utilisés pour l'arithmétique des polynômes.

Fait 1.16 ([Sho08, Chapter 20])**.** *Étant donné un corps fini* $\mathbb{F}_q$*, un entier* d *et une probabilité* $0 < \epsilon < 1$*, il existe un algorithme qui génère un polynôme aléatoire de degré* d *dans* $\mathbb{F}_q[x]$*. Ce polynôme est irréductible avec une probabilité d'au moins* $1 - \epsilon$*. L'algorithme effectue* $\mathcal{O}(\log(\frac{1}{\epsilon}) d^2 \mathsf{M}(d)(\log q + \log\log d))$ *opérations dans* $\mathbb{F}_q$ *ou* $\mathcal{O}(\log(\frac{1}{\epsilon}) d^4 \log q)$ *opérations dans* $\mathbb{F}_q$ *si les produits de polynômes sont réalisés par l'algorithme naïf.*

Il existe des algorithmes plus efficaces pour générer des polynômes irréductibles [Sho93 ; Sho94], notamment en optimisant la complexité par rapport au degré [CL13] et non à $\log q$. Cependant les polynômes obtenus grâce à ces algorithmes ne sont pas choisis uniformément parmi l'ensemble des polynômes de degré d de $\mathbb{F}_q$ et nous aurons parfois besoin de cette propriété.

Il est intéressant de noter que la génération de polynômes irréductibles et celle de nombres premiers est en pratique plus rapide que la complexité annoncée. En effet les polynômes ou entiers composés ont généralement un facteur irréductible de petite taille et peuvent donc être écartés rapidement lors d'un test d'irréductibilité. Cependant, seules les complexités dans le pire cas sont présentées et seront utilisées dans le cadre de cette thèse.

1.4. Impact de l'arithmétique sur la taille des polynômes creux

Les opérations basiques vues sur les polynômes denses dans la section 1.3, à savoir le produit et la division, sont quasi-linéaires dans le degré des polynômes considérés. Pour des polynômes creux, il faut considérer comme paramètre le nombre t de monômes non nuls de ces polynômes. Or, il est impossible d'atteindre une complexité quasi-linéaire en t pour des polynômes creux. Et ce simplement car le résultat peut avoir beaucoup plus de monômes que cela comme le montrent les exemples suivants.

Exemple 1.17. Un produit de polynômes de 4 monômes chacun peut avoir 16 monômes non nuls : $(x^{73} + x^{56} + x^{22} + x^3)(x^{189} + x^{36} + x^{27} + x^{11}) = x^{262} + x^{245} + x^{211} + x^{192} + x^{109} + x^{100} + x^{92} + x^{84} + x^{83} + x^{67} + x^{58} + x^{49} + x^{39} + x^{33} + x^{30} + x^{14}$.

La croissance est encore plus importante pour une division euclidienne.

Exemple 1.18. Une division entre un binôme et un trinôme peut donner un quotient et un reste dont le nombre de monômes égale le degré :

$$x^{2d+1} + x^{2d} = (x^{d+1} - x^d + 1)(x^d + \sum_{i=0}^{d-1} 2x^i) + x^d - \sum_{i=0}^{s-1} 2x^i.$$

Cette section donne des bornes sur la croissance du nombre de monômes et, dans le cas de polynômes dans $\mathbb{Z}[x]$, des bornes sur la croissance des coefficients.

1.4.1. Variation de tailles lors d'additions, produits et divisions

Observons l'évolution du nombre de monômes et de la hauteur de polynômes creux lors des différentes opérations arithmétiques. L'addition et la multiplication sont des opérations faciles à analyser.

Fait 1.19. *Soient F et G deux polynômes dans $\mathbb{A}[x]$. Ils vérifient*

- $\#(F + G) \leq \#F + \#G$,

· $\#(FG) \leq \#F\#G$,

où pour tout polynôme P, $\#P$ désigne le nombre de ses monômes non nuls. Si de plus $\mathbb{A} = \mathbb{Z}$, avec $\|P\|$ la hauteur d'un polynôme P :

· $\|F + G\| \leq \|F\| + \|G\|$,

· $\|FG\| \leq \min(\#F, \#G)\, \|F\|\, \|G\|$.

Démonstration. Pour le nombre de monômes, c'est simplement qu'un élément du support de $F + G$ est forcément un élément du support de F ou un élément du support de G. Dans le cadre d'un produit, les exposants sont additionnés et donc un élément du support de FG est la somme d'un élément du support de F plus un du support de G.

Pour la hauteur de la somme, cela vient du fait qu'un monôme de $F + G$ est soit un monôme de l'un des deux polynômes soit la somme d'un monôme de F et d'un monôme de G. Un monôme du produit peut venir de plus de monômes de F et G. S'il a degré k, il s'écrit $\sum_{i+j=k} f_i x^i g_j x^j$ où les $f_i x^i$ et $g_j x^j$ sont des monômes de F et G respectivement. La somme compte au plus $\min(\#F, \#G)$ termes. Puisque $|f_i| \leq \|F\|$ pour tout i et $|g_j| \leq \|G\|$ pour tout j, le coefficient formé est borné en valeur absolue par $\leq \min(\#F, \#G)\, \|F\|\, \|G\|$, d'où le résultat. □

Comme vu dans l'exemple précédent, la division euclidienne peut provoquer des croissances beaucoup plus importantes. Et ce, même dans des cas où le reste est nul en prenant notamment la division de $x^p - 1$ par $x - 1$. Sans ajouter de contraintes sur les polynômes, la seule borne a priori sur le nombre de monômes du quotient et du reste est leur degré. En revanche, il est possible de donner des bornes en fonction du nombre de monômes non nuls du quotient.

Lemme 1.20. *Soient F, $P \in \mathbb{Z}[x]$ deux polynômes creux tels que la division euclidienne de F par P est bien définie. Posons $F = QP + R$ cette division euclidienne, alors*

· $\|Q\| \leq (\|P\| + 1)^{\#Q-1}\, \|F\|$,

· $\#R \leq \#F + \#P\#Q$,

· $\|R\| \leq (\|P\| + 1)^{\#Q}\, \|F\|$.

De plus, si $R = 0$:

· $\|Q\| \leq (\|P\| + 1)^{\lceil \frac{\#Q-1}{2} \rceil}\, \|F\|$.

Démonstration. Considérons $Q = \sum_{i=1}^{\#Q} q_i x^{e_i}$ avec $e_1 > e_2 > \cdots > e_T$. La preuve se fait par récurrence sur la suite des quotients et restes produits lors de l'algorithme de division euclidienne. Soit $Q_j = \sum_{i=1}^{j} q_i x^{e_i}$ et $R_j = F - Q_j P$ les éléments de cette suite qui débute avec $R_0 = F$ et $Q_0 = 0$. Les coefficients de Q sont définis par la relation $q_i = \mathsf{CD}(R_{i-1})/\mathsf{CD}(P)$ où CD désigne le coefficient dominant du polynôme considéré.

Ainsi $|q_i| \leq \|R_{i-1}\|$. L'algorithme construit les différents polynômes de la suite en utilisant la relation de récurrence $R_i = R_{i-1} - q_i x^{e_i} P$ et $Q_i = Q_{i-1} + q_i x^{e_i}$. Puisque $R_0 = F$ et

$$\|R_i\| \leq \|R_{i-1}\| + |q_i| \times \|P\| \leq \|R_{i-1}\| (1 + \|P\|),$$

on en déduit que $\|R_i\| \leq \|F\| (1 + \|P\|)^i$. Ainsi

$$\|Q\| = \max_i(|q_i|) \leq \max_i \|R_{i-1}\| \leq \|F\| (1 + \|P\|)^{\#Q-1}.$$

La borne sur la hauteur du reste s'obtient directement puisque $R = R_{\#Q}$. Pour le nombre de monômes, il suffit de constater qu'à chaque étape au plus $\#P$ monômes sont ajoutés à R_i.

Par ailleurs, s'il y a divisibilité, le polynôme réciproque $Q^\star$ de Q est défini par le quotient $F^\star/P^\star$. Ce quotient a la même hauteur que Q, et le calculer par le même algorithme revient à calculer une autre suite de restes R_i^* telle que $|q_{\#Q-i}| \leq \|R_{i-1}\|$. Cette suite verifie aussi $\|R_i\| \leq \|F\| (1 + \|P\|)^i$. Ceci permet de déduire l'inégalité $q_i \leq \min(\|F\| (1 + \|P\|)^{i-1}, \|F\| (1 + \|P\|)^{\#Q-i-1})$. Par conséquent

$$\|Q\| = \max_i(|q_i|) \leq \|F\| (1 + \|P\|)^{\lceil \frac{\#Q-1}{2} \rceil}. \qquad \square$$

1.4.2. Croissance lors de réduction modulaire

Lors d'une division, ne considérer que le reste ne permet pas d'obtenir de meilleures bornes. Même en ne considérant que la réduction d'un seul monôme par un polynôme de petit degré.

Exemple 1.21. Soit $P = x^{16} + 7x^{13} + 2x^{12} - 8x^{11} + x^{10} + 3$ un polynôme entier de degré 16 et de hauteur 8. Le polynôme $x^{131} \bmod P$ est de degré 15 avec 16 monômes non nuls mais sa hauteur est un entier de 120-bits.

Il reste possible de donner une borne sur la hauteur du reste mais celle-ci dépend du degré du quotient et non de son nombre de monômes. Pour avoir plus de finesse il est aussi possible de faire dépendre cette borne de la structure de P. L'élément de structure de P considéré est l'*écart* entre ses deux plus grands exposants.

Définition 7. Soit $P = ax^d + \sum_{i=0}^{k} p_i x^i$ pour $k < d$ avec $a, p_k \neq 0$. Le *paramètre d'écart* γ de P est $\gamma = \frac{1}{d}(d - k)$.

En particulier, le deuxième plus grand exposant de P est $(1 - \gamma)d$. Le paramètre γ est compris entre 0 et 1. Si γ est proche de 0, le polynôme n'a en fait aucun écart alors que si $\gamma = 1$ il s'agit d'un binôme $ax^d + p_0$. Notons qu'avec cette définition, $\frac{1}{\gamma}$ est toujours inférieur à d. Des polynômes avec un écart important sont connus sous le nom de polynômes sédimentaires [MP13]. Un polynôme est dit t-sédimentaire s'il est de la forme $x^d + P_0$ avec $\deg(P_0) = t$, il s'agit donc d'un polynôme avec un paramètre d'écart de $\frac{d-t}{d}$. Inversement un polynôme de paramètre d'écart γ est $(1 - \gamma)d$-sédimentaire. Le

lemme suivant montre comment ce paramètre d'écart influence la croissance du nombre de monômes et des coefficients lors de sa réduction modulo P. Le polynôme P y est considéré seulement dans le cas unitaire pour que la réduction ait du sens pour tout polynôme F.

Lemme 1.22. *Soit F, P des polynômes de degré respectivement $d + d_p - 1$ et d_p, ayant respectivement $\#F$ et $\#P$ monômes non nuls avec $\#P \geq 2$, P unitaire de paramètre d'écart γ. Le polynôme $F \bmod P$ a au plus $\#F(\#P-1)^{\lceil \frac{d}{\gamma d_p} \rceil}$ monômes non nuls. Si de plus F et P sont des polynômes à coefficients dans $\mathbb{Z}$, la hauteur de $F \bmod P$ vérifie $\|F \bmod P\| \leq \|F\| \left(\#P \, \|P\|\right)^{\lceil \frac{d}{\gamma d_p} \rceil}$.*

Démonstration. La croissance du nombre de monômes et de la valeur de leurs coefficients est analysée au cours de l'exécution d'une version modifiée de l'algorithme de division euclidienne. Plutôt que de réduire les monômes 1 par 1, cette version commence par réduire une fois tous les monômes de degré plus grand que d_p pour obtenir un nouveau dividende auquel cette réduction sera à nouveau appliquée jusqu'à obtenir un polynôme de degré au plus $d_p - 1$. Considérons la suite $(F_i)_i$ des dividendes ainsi calculés. Elle est définie par $F_0 = F$ et $F_{i+1} = (F_i \bmod x^{d_p}) + (F_i \operatorname{quo} x^{d_p})(x^{d_p} - P)$. Ainsi $F_i \bmod P = F \bmod P$ pour tout i. Puisque $\deg(F_i \operatorname{quo} x^{d_p}) = \deg(F_i) - d_p$ et $\deg(x^{d_p} - P) \leq (1-\gamma)d_p$, $\deg(F_{i+1}) \leq \max(d_p - 1, \deg(F_i) - \gamma d_p)$, d'où $\deg(F_i) \leq \max(d_p - 1, \deg(F) - i\gamma d_p)$. De plus, $\#(F_{i+1})$ est au plus $\#(F_i)(\#P - 1)$, d'où $\#(F_i) \leq \#F(\#P-1)^i$. Enfin,

$$\|F_{i+1}\| \leq \|F_i\| \left(1 + \min(\#(F_i), \#P - 1) \, \|P\|\right).$$

Par conséquent, $\|F_i\| \leq \left(\#P \, \|P\|\right)^i \|F\|$.

Puisque $\deg(F_i) \leq d + d_p - 1 - i\gamma d_p$, $\deg(F_i) < d_p$ pour $i = \lceil \frac{d}{\gamma d_p} \rceil$. À ce stade, $F_i = F \bmod P$ et l'algorithme s'arrête. □

Si $d = \mathcal{O}(d_p)$ et γ est une constante, les bornes ainsi obtenues sont des bornes polynomiales.

2. Travailler avec des polynômes de taille toujours bornée

L'intérêt de la représentation creuse des polynômes est de nécessiter moins de bits que la représentation dense dès que le polynôme n'a que peu de monômes non nuls. C'est d'ailleurs une représentation qui se veut la plus compacte possible en tendant à utiliser exactement le nombre de bits nécessaires à la représentation des polynômes. En partant d'une taille donnée, les calculs sur les polynômes creux peuvent cependant entraîner des croissances de taille très importante comme cela a été montré en section 1.4. S'il s'agit de la taille du résultat attendu, cela ne pose pas de problème. Cependant, ces croissances de taille peuvent survenir lors de calculs intermédiaires alors que le résultat lui-même est plus petit. Elles peuvent alors impacter de manière très négative le temps de calcul.

Un exemple de cela est l'évaluation d'un polynôme F de $\mathbb{Z}[x]$ sur un entier α. Cette opération peut être effectuée pour elle-même mais aussi pour tester si le polynôme est le polynôme nul. Si F est un polynôme de degré D, la taille de sa représentation creuse dépend de $\log D$. L'évaluation en α peut produire un entier supérieur en valeur absolue à $|\alpha|^D$ donc de taille au moins $D \log \alpha$. Entre les deux tailles, la croissance est exponentielle. Clairement, s'il s'agit simplement de tester la nullité du polynôme F et donc de renvoyer un booléen, l'utilisation de tant d'espace semble superflue. Il paraît naturel d'effectuer ce type d'opérations sur un corps fini pour limiter une telle croissance de taille quand c'est possible.

Un autre exemple que l'on voit au chapitre 3, est celui du circuit arithmétique. Un circuit arithmétique qui représente un polynôme creux F, avec peu de monômes non nuls, est une suite d'opérations qui permet de le calculer. En effectuant ces opérations les unes après les autres, il n'est absolument pas garanti que les polynômes intermédiaires restent creux, aient aussi peu de monômes non nuls que F. Si le but est d'obtenir la représentation creuse de F, c'est alors le calcul des polynômes intermédiaires qui domine la complexité sans égard pour la taille de F. Les algorithmes pour calculer efficacement F effectuent généralement les calculs modulo un polynôme de la forme $x^p - 1$ pour limiter la croissance des polynômes intermédiaires.

L'approche naturelle face à ces problèmes de croissance consiste à essayer de borner la taille des données, nombres ou polynômes, manipulées durant les algorithmes. Ceci doit se faire sans perdre toutes les informations originelles pour que les résultats obtenus restent significatifs. Nous voyons différentes techniques pour cela en section 2.1. Ces techniques sont essentielles à de nombreux algorithmes sur les polynômes creux et seront largement

utilisées dans la suite de cette thèse.

Pour prévenir des croissances indésirables et contrôler la taille des polynômes manipulés, ceux-ci sont considérés modulo un nombre premier ou un autre polynôme. Si un point ω est une racine d'un polynôme P, alors évaluer F en ω revient à évaluer le polynôme réduit $F \bmod P$ sans avoir besoin de connaître ou calculer $F \bmod P$. Pour des polynômes à coefficients entiers, effectuer cette évaluation dans un anneau modulaire permet de réduire du même coup les coefficients. Cela peut de plus être nécessaire pour trouver une racine de P. Nous verrons en section section 2.2 comment obtenir de tels points d'évaluations pour des réductions modulo $x^p - 1$.

De tels changements impactent les opérations effectuées sur les polynômes. Dans certains cas, comme le produit de polynômes cet impact n'est pas significatif. D'autres opérations peuvent être fortement complexifiées. C'est notamment le cas de l'évaluation d'un produit de polynômes. Des algorithmes pour effectuer une telle évaluation modulo un troisième polynôme sont décrits en section 2.3.

2.1. Limiter la perte d'information lors des réductions

Pour borner la taille des polynômes manipulés, il faut agir à la fois sur les coefficients et sur les degrés. Si les polynômes sont définis sur les entiers, les considérer modulo un nombre premier q permet de borner la taille de tous les coefficients par $\log q$. De plus considérer les polynômes modulo un polynôme P de degré d permet de borner le degré et le nombre de monômes non nuls par d. En considérant un polynôme F modulo q ou P, sa valeur exacte est perdue. Seuls sont connus les restes dans la division euclidienne de ses coefficients par q ou le reste de la division euclidienne de F par P. En particulier, le polynôme obtenu peut-être nul alors que F lui-même ne l'est pas.

Dans un premier temps nous regarderons, en section 2.1.1, le cas où l'on cherche seulement à prévenir l'annulation d'un polynôme lors d'une réduction modulaire.

Dans un second temps, en section 2.1.2, nous nous intéresserons spécifiquement au cas des polynômes creux. L'approche est alors plus fine. En plus de savoir si le polynôme entier a été annulé, il est possible de se demander combien de monômes ont été annulés ou quel est l'écart entre le nombre de monômes du polynôme d'origine et celui du polynôme réduit.

Les résultats présentés ici sont classiques dans l'algorithmique des polynômes creux [GS09 ; GR11b ; AGR13 ; HG20] et très utiles pour les algorithmes présentés dans cette thèse. Nous prendrons donc le temps de les présenter avec une preuve.

2.1.1. Empêcher l'annulation

Lors d'une réduction modulaire, envoyer un polynôme vers zéro donne certes l'information qu'il était divisible par le modulo mais pas plus. En particulier, il est impossible

de trancher si c'est le polynôme est égal au polynôme nul en général ou seulement à cause de la division. C'est une situation qu'il est généralement préférable d'éviter, notamment dans le cadre d'étude de la section 4.2 où l'objectif est justement de déterminer si un polynôme est nul ou pas. Nous nous intéressons à la manière de procéder dans trois situations : la division d'un entier ou d'un polynôme par un nombre premier, la division d'un polynôme par un polynôme irréductible et enfin la division d'un polynôme par un polynôme de la forme $x^i - 1$.

Dans chaque situation, le choix du modulo est libre, il convient donc de voir comment l'orienter pour s'assurer au moins avec une bonne probabilité qu'il n'y a pas divisibilité.

2.1.1.1. Réduction modulo un nombre premier

Le premier cas que nous regardons est celui de la réduction d'un entier modulo un nombre premier. Comme nous avons vu deux méthodes de génération d'un nombre premier en section 1.3.2, nous allons donner les propriétés des entiers selon les deux points de vue correspondants : pour un premier quelconque ou pour un premier dans un intervalle. Nous ferons de même pour les cas reposant sur ce cas de base.

Pour un entier N, l'objectif ici est de trouver un nombre premier p tel que $N \bmod p \neq 0$, sauf si N lui-même est égal à 0. Autrement dit p ne doit pas diviser N. L'approche consiste à dénombrer le nombre de diviseurs possibles de N puis à choisir p aléatoirement dans un ensemble de nombres premiers suffisament vaste pour que la probabilité que p divise N soit faible.

Fait 2.1. *Un nombre $N \in \mathbb{N}$ a au plus :*

- $\log N$ *diviseurs premiers*
- $\ln N / \ln \lambda$ *diviseurs premiers plus grand qu'un entier λ*

Démonstration. Il suffit de constater que le produit des diviseurs premiers de N supérieurs à λ ne peut excéder N et de prendre $\lambda = 2$ pour le premier cas. □

Corollaire 2.2. *Pour $N \in \mathbb{N}$ et $0 < \epsilon < 1$, la probabilité qu'un nombre p divise N est inférieure à ϵ dans les cas où p est choisi aléatoirement parmi :*

- *les $\frac{1}{\epsilon} \log N$ premiers nombres premiers, ou*
- *les nombres premiers de l'intervalle $[\lambda, 2\lambda]$ pour $\lambda \geq \max(21, \frac{5}{3\epsilon} \ln N)$.*

Démonstration. Ce corollaire s'obtient en comparant le nombre de premiers dans l'ensemble dans lequel est choisi p et le nombre d'entre eux qui peuvent diviser N. Le premier cas est évident. Pour le second, on se place dans l'intervalle $[\lambda, 2\lambda]$ qui contient au moins $\frac{3}{5}\lambda / \ln \lambda$ nombres premiers (fait 1.12) dont au plus $\ln N / \ln \lambda$ peuvent diviser N. Le choix fait pour λ dans le corollaire garantit qu'un premier choisi aléatoirement dans l'intervalle a une probabilité inférieure à ϵ de diviser N. □

Remarque 2.3. Le fait 2.1 et le corollaire 2.2, s'appliquent aussi à tout nombre entier n inférieur à N.

Puisque les entiers peuvent être annulés en passant modulo un nombre premier p, cela concerne aussi les coefficients d'un polynôme défini sur $\mathbb{Z}$. La probabilité que cela arrive si p est choisi aléatoirement peut être bornée de manière similaire. Nous ne considérerons ici que le cas d'un premier choisit dans un intervalle, l'autre cas étant similaire. En revanche, deux cas sont regardés : si au moins un des coefficients de F, et donc F, ne s'annule pas et si aucun de ses coefficients ne s'annule.

Corollaire 2.4. *Soit $F \in \mathbb{Z}[x]$ un polynôme ayant au plus T monômes non nuls de coefficients bornés par H et $0 < \epsilon < 1$. Avec probabilité au moins $1 - \epsilon$, un nombre premier p choisi aléatoirement dans l'intervalle $[\lambda, 2\lambda]$*

(i) ne divise pas un des coefficients de F (au moins) si $\lambda \geq \max(21, \frac{5}{3\epsilon} \ln H)$,

(ii) ne divise aucun des coefficients de F si $\lambda \geq \max(21, \frac{5}{3\epsilon} T \ln H)$.

Démonstration. C'est l'application du corollaire 2.2, avec d'une part $N = H$ pour borner celui des coefficients de F que p ne doit pas diviser et d'autre part $N = H^T$ pour borner le produit des coefficients de F, p ne devant en diviser aucun. □

Les nombres premiers obtenus en s'appuyant sur ce corollaire ont une valeur en $\mathcal{O}(T \log H)$ qui est linéaire dans la taille du polynôme F, que celui-ci soit sous forme creuse ou dense (de degré T). Le stockage des coefficients du polynôme réduit a alors une taille en $\mathcal{O}(T \log(T \log H))$ et même $\mathcal{O}(T \log \log H)$ si le but est seulement de préserver le caractère non nul d'un des coefficients. En comparaison, le stockage des coefficients de F nécessite $\mathcal{O}(T \log H)$ bits. La taille des éléments manipulés a donc été réduite. Et cela sans perdre trop d'information c'est-à-dire en restant capable de déterminer si le polynôme ou certains de ses coefficients sont nuls.

2.1.1.2. Réduction d'un polynôme modulo un polynôme irréductible

Pour restreindre la taille d'un polynôme, l'autre point à considérer est son degré. Pour cela le polynôme sera considéré modulo un autre polynôme. Une méthode absolument similaire à celle vue sur les entiers s'applique à ce cas. De manière générale, il s'agit de compter le nombre de diviseurs possibles (c'est à dire de modulos inappropriés) et évaluer la densité de ceux-ci dans un ensemble plus vaste au sein duquel ils peuvent être choisis aléatoirement. Nous présentons donc le résultat associé de manière plus succincte. Les diviseurs sont considérés dans l'ensemble des polynômes unitaires irréductibles d'un degré d fixé.

Fait 2.5. *Dans $\mathbb{F}_q[x]$, la probabilité qu'un polynôme choisi uniformément parmi les polynômes unitaires irréductibles de degré d divise un polynôme de degré D est d'au plus $\frac{2D}{q^d}$.*

Démonstration. Dans sa factorisation, un polynôme de degré D admet au plus $\frac{D}{d}$ facteurs de degré d. Or par le fait 1.15, il y a au moins $\frac{q^d}{2d}$ polynômes unitaires irréductibles de degré d. □

La probabilité dépend du degré du polynôme irréductible choisi. Si l'objectif est que la probabilité d'avoir la divisibilité soit inférieure à une borne ϵ, il faut fixer $d \geq \log_q(\frac{2D}{\epsilon})$.

2.1.1.3. Réduction d'un polynôme modulo un binôme

La réduction d'un polynôme modulo un binôme $x^i - 1$ présente l'avantage d'être plus efficace à calculer que modulo un polynôme irréductible qui peut avoir un nombre quelconque de monômes. En effet, la réduction ne nécessite pas d'autre opération dans l'anneau de base que des additions entre coefficients. Une telle réduction est introduite par Kaminski [Kam89] et se base aussi sur le dénombrement des diviseurs possibles de la forme $x^i - 1$. La preuve est plus technique mais nous donnons quelques éléments pour permettre un aperçu général de son fonctionnement.

Kaminski s'appuie sur le fait que tous les diviseurs des polynômes $x^i - 1$ sont des polynômes cyclotomiques, ce qui permet de prouver une propriété particulière sur leur ppcm, quelque soit l'anneau de définition $\mathbb{A}$.

Fait 2.6 ([Kam89, Proposition 3])**.** *Soit Φ_i le i-ième polynôme cyclotomique, alors pour tout ensemble d'entiers $I \subset \mathbb{N}$, $\prod_{i \in I} \Phi_i$ divise* ppcm$\{x^i - 1 : i \in I\}$ *dans $\mathbb{A}[x]$.*

Ce fait permet de fournir une borne inférieure au degré du ppcm d'un ensemble de polynômes $\{x^i - 1 | i \in I\}$. Cette borne, $D_I = \deg(\prod_{i \in I} \Phi_i) = \sum_{i \in I} \phi(i)$ pour ϕ l'indicatrice d'Euler, dépend de la valeur des éléments de I. Par ailleurs cette borne s'étend naturellement au degré de tout polynôme divisé par tous les $x^i - 1$ pour $i \in I$. En la prenant à rebours, elle fournit alors une borne sur le nombre de polynômes $x^i - 1$ qui peuvent diviser un polynôme de degré fixé. Pour F un polynôme de degré D, l'ensemble $I = \{i | F \bmod x^i - 1 = 0\}$ vérifie nécessairement que $D_I \leq D$. La borne ainsi obtenue par Kaminski repose entre autre sur un paramètre dépendant de la constante d'Euler. Pour simplifier la lecture, le paramètre est remplacé par 2, l'entier le plus proche qui conserve la validité du résultat.

Théorème 2.7 ([Kam89, Theorem 1, condition P1])**.** *Soit $F \in \mathbb{A}[x]$ un polynôme non nul de degré au plus $2D$, $0 < e < \frac{1}{2}$ et $k = \lceil 4D^e \ln\ln(D^{1-e}) \rceil$. Le polynôme F admet au plus $k - 1$ diviseurs de la forme $x^i - 1$ avec $D^{1-e} \leq i < 2D^{1-e}$.*

Une question naturelle est de savoir si de meilleures bornes, c'est-à-dire des valeurs de i potentiellement plus petites, peuvent être atteintes en se restreignant à des i premiers. Les polynômes Φ_p pour des p premiers étant premiers entre eux et de degré égal à $p - 1$, le degré du ppcm des polynômes $x^p - 1$ et simplement $1 + \sum_p (p - 1)$. En considérant seulement les k plus petits nombres premiers et en se basant sur le fait 1.10, le degré du ppcm est de l'ordre de k^2, à un facteur logarithmique près. Pour avoir un p tel que $x^p - 1$ ne divise pas notre polynôme F de degré $2D$, il faudrait donc choisir parmi un ensemble

de plus de $\sqrt{2D}$ nombres premiers. Ces nombres premiers auraient donc des valeurs assez proches de celles des i considérés au théorème 2.7. Sachant que la génération de nombres premiers est plus coûteuse que celle d'un nombre aléatoire, une telle approche n'est guère efficace dans le cas général. Cependant nous allons voir qu'il en va tout autrement dans le cas des polynômes creux.

2.1.2. Cas des polynômes creux : éviter les collisions

Dans le cas des polynômes creux les méthodes utilisées sont spécifiques et reposent sur le fait que l'on puisse travailler directement sur les monômes du polynôme. Par ailleurs les réductions regardées sont toujours modulo des polynômes de la forme $x^p - 1$. Ce type de réduction qui consiste à réduire les exposants envoie le polynôme $F = \sum_{i=0}^{T-1} f_i x^{e_i}$ sur $F_p = \sum_{i=0}^{T-1} f_i x^{e_i \bmod p}$ ce qui permet de conserver ou même de réduire le nombre de monômes. Au contraire une réduction modulo un polynôme dont la seule caractéristique serait un degré d pourrait produire un polynôme de d monômes et donc augmenter la taille de notre polynôme au lieu de la contenir.

Dans les réductions modulo un binôme il arrive que le nombre de monômes réduise à cause de ce qu'on appelle des *collisions*.

Définition 8. Pour un polynôme $F = \sum_{i=0}^{T-1} f_i x^{e_i}$ et un nombre $p \in \mathbb{N}$, on dit qu'il y a une *collision* modulo $x^p - 1$ s'il existe deux indices $i \neq j$ tels que $e_i \equiv e_j \bmod p$. On dit qu'un monôme $f_i x^{e_i}$ est *en collision* s'il existe $j \neq i$ tels que $e_i \equiv e_j \bmod p$. Dans le cas contraire ce monôme est *sans collision*, terminologie qui s'applique aussi à F si tous ses monômes sont sans collision.

L'objectif ne sera donc plus seulement de s'assurer que $x^p - 1$ ne divise pas F mais aussi de régler le nombre de collisions pour savoir à quel point le nombre de monômes de $F \bmod x^p - 1$ est proche de celui de F.

La première question reste quand même celle de la divisibilité. Pour s'assurer que $x^p - 1$ ne divise pas F, il suffit de s'assurer qu'un des monômes de F est sans collision. De manière arbitraire, c'est le premier monôme que nous choisirons pour cela.

Fait 2.8. *Soit* $F = \sum_{i=0}^{T-1} f_i x^{e_i} \in \mathbb{A}[x]$ *et* $p \in \mathbb{N}$.

Si p *ne divise pas* $\delta_0 = \prod_{i=1}^{T-1} (e_i - e_0)$ *alors* $x^p - 1$ *ne divise pas* F.

Démonstration. L'hypothèse revient seulement à dire que le monôme $f_0 x^{e_0}$ est sans collision modulo $x^p - 1$ puisque aucun autre exposant n'est congru à e_0 modulo p. Par conséquent $f_0 x^{e_0 \bmod p}$ est un monôme de $F \bmod x^p - 1$ qui est donc non nul. □

Ceci nous donne donc une méthode pour obtenir un polynôme $x^p - 1$ qui probablement ne divise pas F.

Corollaire 2.9. *Soit $F \in \mathbb{A}[x]$ un polynôme non nul de degré au plus D avec au plus T monômes, une probabilité $0 < \epsilon < 1$ et $\lambda \geq \max(21, \frac{10}{3\epsilon}(T-1)\ln D)$. Si p est un nombre premier choisi uniformément dans l'intervalle $[\lambda, 2\lambda]$, alors $F \bmod x^p - 1 \neq 0$ avec une probabilité d'au moins $1 - \frac{\epsilon}{2}$.*

En particulier, avec une probabilité d'au moins $1 - \epsilon$, $p = \text{PremierAléa}(\lambda, \frac{\epsilon}{2})$ est un nombre premier tel que $F \bmod x^p - 1 \neq 0$.

Démonstration. C'est l'application du corollaire 2.2 avec probabilité $\frac{\epsilon}{2}$ qu'un nombre premier de l'intervalle $[\lambda, 2\lambda]$ divise un entier N. L'entier N est pris égal à D^{T-1} pour avoir $\delta_0 \leq N$, δ_0 étant défini dans le fait 2.8. La probabilité $\frac{\epsilon}{2}$ s'applique donc au fait que $x^p - 1$ divise F. Le nombre premier est généré avec l'algorithme PremierAléa (voir fait 1.13, page 17) dans l'intervalle $[\lambda, 2\lambda]$ avec une probabilité d'échec d'au plus $\frac{\epsilon}{2}$, d'où la probabilité globale d'échec à ϵ. □

L'autre extrémité dans le choix de p est de s'assurer que le polynôme F est sans collision. Il peut-être traité de manière similaire en ne traitant pas un seul monôme mais tous les monômes.

Fait 2.10. *Soit $F = \sum_{i=0}^{T-1} f_i x^{e_i} \in \mathbb{A}[x]$ et $p \in \mathbb{N}$.*

Si p ne divise pas $\prod_{i=0}^{T-1} \delta_i$, où $\delta_i = \prod_{j \neq i} |e_j - e_i|$ alors F est sans collision modulo $x^p - 1$.

Démonstration. Si p ne divise pas leur produit, p ne divise aucun δ_i. Donc comme dans le cas du fait 2.8, les monômes correspondants sont sans collision, donc F aussi. □

Comme cette propriété sera utilisée avec les deux méthodes de génération de nombres premiers, le corollaire suivant présente les deux manières dont elle sera appliquée.

Corollaire 2.11. *Soit $F \in \mathbb{A}[x]$ un polynôme non nul de degré au plus D avec au plus T monômes. Il existe au plus $T(T-1)\log D$ nombres premiers p pour lesquels F a au moins une collision.*

Par ailleurs, pour une probabilité $0 < \epsilon < 1$ et $\lambda \geq \max(21, \frac{10}{3\epsilon}T(T-1)\ln D)$, avec une probabilité d'au moins $1 - \epsilon$, $\text{PremierAléa}(\lambda, \frac{\epsilon}{2})$ renvoie un nombre premier p tel que F est sans collision modulo $x^p - 1$.

Démonstration. C'est une application du fait 2.1 et du corollaire 2.2 qui bornent le nombre de diviseurs premiers d'un entier N et la probabilité qu'un tirage aléatoire parmi les nombres premiers de l'intervalle $[\lambda, 2\lambda]$ fournisse un diviseur de N. En prenant $N = D^{T(T-1)}$, on se place dans le cadre du fait 2.10 puisque chacun des δ_i est inférieur à

D^{T-1}. Comme pour le corollaire 2.9, la probabilité est ajustée pour prendre en compte la probabilité d'échec de l'algorithme PremierAléa. □

Le dernier point consiste à limiter le nombre de collisions à une fraction des monômes de F, cette fraction pouvant être adaptée en fonction des besoins spécifiques des algorithmes. Différentes versions du prochain lemme sont présentes dans la littérature, ne s'appuyant pas forcément sur la même preuve si les nombres premiers sont générés par crible [HG20 ; Hua19] ou dans un intervalle [AGR13 ; AGR14 ; AR15].

Fait 2.12. *Soit* $F = \sum_{i=0}^{T-1} f_i x^{e_i} \in \mathbb{A}[x]$, $p \in \mathbb{N}$ *et* $0 < \gamma < 1$.

Si $p^{\lceil (1-\gamma)T \rceil}$ *ne divise pas* $\prod_{i=0}^{T-1} \delta_i$, *où* $\delta_i = \prod_{j \neq i} |e_j - e_i|$ *alors une proportion* γ *des* T *monômes de* F *est sans collision modulo* $x^p - 1$.

Démonstration. Comme vu pour les faits précédents, 2.8 et 2.10, un monôme est en collision modulo $x^p - 1$ si p divise le δ_i correspondant. Or dans notre hypothèse, p divise moins de $(1-\gamma)T$ de ces δ_i, sinon $p^{\lceil (1-\gamma)T \rceil}$ diviserait leur produit. Cela donne donc une borne supérieure sur la proportion de monômes en collision. Il y en a alors au moins γT qui sont sans collision. □

Comme précédemment, il est possible d'en déduire un corollaire sur la génération d'un nombre premier laissant une certaine proportion de monômes d'un polynôme F sans collision. La formulation est ici légèrement différente pour mieux correspondre aux applications de ce corollaire.

Corollaire 2.13. *Soit* $F \in \mathbb{A}[x]$ *un polynôme de degré au plus* D *avec au plus* T *monômes, une probabilité* $0 < \epsilon < 1$ *et* $\lambda \geq \max(21, \frac{5}{3\epsilon} \min(T, \frac{1}{1-\gamma})(T-1) \ln D)$ *pour une proportion* $0 < \gamma < 1$.

Avec une probabilité d'au moins $1 - \epsilon$, *si* p *est un nombre premier choisi aléatoirement dans l'intervalle* $[\lambda, 2\lambda]$, *alors une proportion d'au moins* γ *monômes de* F *est sans collision modulo* $x^p - 1$.

Démonstration. C'est l'application du corollaire 2.2 en prenant $N = \left\lceil D^{\frac{(T-1)}{(1-\gamma)}} \right\rceil$. Ainsi pour un nombre entier inférieur à N, le nombre premier p choisi n'est pas un de ses diviseurs avec une probabilité d'au moins $1-\epsilon$. C'est vrai en particulier pour $\left(\prod_{i=0}^{T-1} \delta_i \right)^{1/\lceil (1-\gamma)T \rceil} \leq N$. Ainsi $p^{\lceil (1-\gamma)T \rceil}$ ne divise probablement pas $\prod_{i=0}^{T-1} \delta_i$ et le fait 2.12 permet de conclure sur la proportion de monômes de F sans collision. Le minimum entre T et $\frac{1}{1-\gamma}$, correspond à une optimisation pour les cas où γ se rapproche de 1 en prenant alors plutôt la borne qui évite toute collision du corollaire 2.11. □

La réduction modulo $x^p - 1$ entraîne directement une perte d'information sur la valeur des exposants de F. Si en plus il y a des collisions, des informations sont aussi perdues concernant les coefficients. Il est alors intéressant d'avoir une notion de *distance* entre F et $F \bmod x^p - 1$ pour mesurer la perte d'information sur les coefficients liée uniquement aux collisions. Lors d'une collision, les coefficients des monômes en collision sont additionnés, ainsi les valeurs d'origine sont perdues et une nouvelle valeur est ajoutée. Il convient de prendre en compte ces deux cas : les monômes *perdus* de F et les monômes *surnuméraires* de $F \bmod x^p - 1$.

Définition 9. Soit $p \in \mathbb{N}$, $F \in \mathbb{A}[x]$. La *distance* entre F et $F \bmod x^p - 1$ est le nombre de monômes de F en collision modulo $x^p - 1$ plus le nombre de monômes de $F \bmod x^p - 1$ qui sont la somme d'au moins 2 monômes de F après réduction modulo p de leur exposant.

Remarque 2.14. Il est possible de donner une définition plus formelle de cette distance : $\#\mathcal{C} + \sum_{E \in \mathcal{C}} \#E$ avec $E_j = \{e_i \in \text{support}(F) | e_i \equiv j \bmod p\}$ et $\mathcal{C} = \{E_i | \#E_i \geq 2\}$.

Cette distance évolue en fonction de la proportion γ des monômes sans collision modulo $x^p - 1$.

Fait 2.15. *Soit $F \in \mathbb{A}[x]$ un polynôme ayant au plus T monômes et p un entier tel qu'une proportion $0 < \gamma < 1$ des monômes de F est sans collision modulo $x^p - 1$.*

La distance entre F et $F \bmod x^p - 1$ est d'au plus $\frac{3}{2}(1-\gamma)T$.

Démonstration. Seuls les $(1-\gamma)T$ monômes potentiellement en collision contribuent à la distance entre F et $F \bmod x^p - 1$. Ils contribuent à hauteur de 1 par le simple fait d'être en collision. Par ailleurs un ensemble de monômes en collision ensemble s'additionne pour former un monôme de $F \bmod x^p - 1$, monôme qui lui-même contribue à hauteur de 1 à la distance.

Un seul monôme en collision contribue donc à la distance à hauteur d'au plus $1 + \frac{1}{k+1}$ où k est le nombre de monômes avec lequel il est en collision. Cette contribution est maximisée pour $k = 1$. D'où une distance inférieure à $\frac{3}{2}(1-\gamma)T$. □

Remarque 2.16. En prenant $\gamma = \frac{2}{3}$, la distance est au plus la moitié du nombre de monôme de F.

Dans les trois cas que nous avons vu pour contrôler le nombre de collisions, le polynôme F qui avait un degré exponentiel en sa taille peut être envoyé sur $F \bmod x^p - 1$ avec alors un degré polynomial, voire linéaire en la taille de F. En particulier cela signifie que l'utilisation de l'arithmétique des polynômes denses après la réduction pourra être efficace. Tout algorithme quasi-linéaire dans le degré, s'il est appliqué au polynôme réduit, est alors polynomial ou quasi-linéaire dans la taille de F.

2.2. Réduction d'un polynôme entier par l'évaluation dans un anneau modulaire

L'évaluation d'un polynôme F en ω une racine p-ième de l'unité revient à l'évaluation de $F \bmod x^p - 1$. Ainsi, les réductions de F que nous venons de voir en section 2.1.2 peuvent être partiellement connues par le biais d'évaluations en des points d'ordre p. De tels points n'existent pas dans tout anneau. En particulier sur les entiers, si p est un nombre premier supérieur à 2, la seule racine entière de $x^p - 1$ est 1. Pour trouver $\omega \neq 1$, il faudra donc chercher dans un anneau modulaire. Plus particulièrement, nous considérerons un corps fini $\mathbb{F}_q$ tel que p divise $q - 1$. Ceci permet d'effectuer à la fois les deux réductions possibles sur un polynôme dans $\mathbb{Z}[x]$: réduction du degré et réduction des coefficients. La possibilité de calculer de telles évaluations est un ingrédient central de notre algorithme d'interpolation au chapitre 3 qui reconstruit le polynôme $F \bmod x^p - 1$ à partir des $F(\omega^i)$ pour ω une racine primitive p-ième de l'unité.

Dans cette section, nous verrons comment générer un triplet (p, q, ω) tel que p et q soient premiers et ω soit une racine primitive p-ième de l'unité dans $\mathbb{F}_q$. Par ailleurs p sera choisi dans un intervalle $[\lambda, 2\lambda]$ pour pouvoir appliquer les résultats de la section 2.1. Nous préciserons aussi comment ajuster la valeur de λ pour que q ne divise probablement pas un entier N, ce qui pourra s'appliquer aux coefficients d'un polynôme F. Dans le reste de la section, nous ne ferons plus le rapport avec la réduction d'un polynôme mais parlerons simplement d'un triplet (p, q, ω)

La première partie, section 2.2.1 de cette section est consacrée à la preuve formelle de l'existence d'une telle méthode. Pour cela il faut considérer des nombres p et q très grands. Cette contrainte semble ne pas être nécessaire en pratique, ce qui sera explicité dans la section 2.2.2.

2.2.1. Générer un corps avec un sous groupe multiplicatif d'ordre p

La méthode utilisée pour calculer un triplet (p, q, ω) tel que ω soit une racine primitive p-ième de l'unité dans $\mathbb{F}_q$ avec p premier est très directe. Il s'agit de tirages successifs jusqu'à obtenir un triplet correspondant. Les tirages son effectués dans l'ordre pour d'abord obtenir un nombre premier p, ensuite obtenir un nombre premier $q \in \{ap + 1 : a \geq 1\}$ ce qui garantit l'existence d'une racine primitive p-ième de l'unité dans $\mathbb{F}_q$ (puisque $p|(q-1)$) et enfin obtenir $\omega \in \mathbb{F}_q$ d'ordre multiplicatif p.

Ces étapes sont décrites dans l'algorithme 1 Triplet qui correspond à l'algorithme "GetPrimeAP-5/6" de la thèse d'Arnold [Arn16] adapté pour avoir de meilleures bornes et pouvoir choisir la probabilité de succès. En plus de la probabilité de succès, l'algorithme n'a besoin que d'un paramètre, un entier λ qui détermine la taille de p et q. En particulier, p est choisi aléatoirement dans l'intervalle $[\lambda, 2\lambda]$, ce qui permet d'appliquer les différents résultats de la section 2.1 en ajustant la valeur de λ.

Théorème 2.17. *L'algorithme 1* Triplet *est un algorithme de type Monte Carlo. Étant*

Algorithme 1 TRIPLET

Entrée : Un entier $\lambda \geq \frac{2^{58}}{\epsilon^2}$, $0 < \epsilon < 1$
Sortie : Un triplet (p, q, ω) avec p, q premiers et $\omega \in \mathbb{F}_q$ d'ordre multiplicatif p.

1: **Répéter** au plus $\frac{5}{6} \ln \frac{4}{\epsilon} \ln \lambda$ fois
2: Tirer aléatoirement un entier impair p dans $[\lambda, 2\lambda]$
3: **Jusqu'à** obtenir un p premier
4: **Si** p n'est pas premier **alors**
5: **Renvoyer** ÉCHEC
6: **Répéter** au plus $12 \ln \frac{4}{\epsilon} \ln \lambda$ fois
7: Tirer aléatoirement un entier pair a dans $[1, \lambda^5]$
8: **Jusqu'à** avoir $q = ap + 1$ premier
9: **Si** q n'est pas premier **alors**
10: **Renvoyer** ÉCHEC
11: **Répéter** au plus $\log_p \frac{4}{\epsilon}$ fois
12: Tirer aléatoirement un élément $\alpha \in \mathbb{F}_q^*$
13: **Jusqu'à** ce que $\omega = \alpha^{(q-1)/p} \neq 1$
14: **Si** $\omega = \alpha^{(q-1)/p} = 1$ **alors**
15: **Renvoyer** ÉCHEC
16: **Renvoyer** (p, q, ω)

donné une probabilité $0 < \epsilon < 1$ *et une borne* $\lambda \geq \frac{2^{58}}{\epsilon^2}$, *il renvoie* ÉCHEC *avec une probabilité d'au plus* ϵ *et sinon génère un triplet* (p, q, ω) *ayant les propriétés suivantes :*

- p *est choisi aléatoirement dans l'intervalle* $[\lambda, 2\lambda]$;
- $q \leq 2\lambda^6$ *est un nombre premier tel que* $p \mid (q-1)$;
- ω *est une racine primitive* p*-ième de l'unité dans* $\mathbb{F}_q$;

Sa complexité est en $\tilde{\mathcal{O}}(\log \frac{1}{\epsilon} \log^{11.5}(\lambda))$.

De plus, si $\lambda \geq \sqrt[5]{\frac{48}{\epsilon} \ln N}$ *pour un entier* $N > 0$, *la probabilité que* q *divise* N *est d'au plus* ϵ.

Cet algorithme n'étant appelé ultérieurement que pour des algorithmes dans lesquels son coût est logarithmique, sa complexité élevée n'impacte pas négativement d'autres complexités. Chaque étape se justifie par l'abondance de nombres ayant la propriété désirée dans l'ensemble considéré. Ces étapes vont être détaillées et justifiées les unes après les autres dans les lemmes 2.18 et 2.21 à 2.23 pour apporter la preuve du théorème 2.17.

2.2.1.1. La primalité de p

La première partie de l'algorithme, lignes 1 à 5, correspond simplement à l'algorithme évoqué dans la remarque 1.14 pour générer un nombre premier en se basant sur l'algorithme de test déterministe d'Agrawal, Kayal et Saxena [AKS04] dit AKS. La version

utilisée est celle de type Monte Carlo, si aucun nombre premier n'a été trouvé au bout d'un certain nombre d'essais le résultat renvoyé est ÉCHEC, sinon p est un nombre premier avec certitude. Le test AKS nécessite $\tilde{\mathcal{O}}(\log^{10.5} \lambda)$ opérations binaires. Pour obtenir le temps de calcul de cette partie de l'algorithme, il faut multiplier cette complexité par le nombre de répétitions, $\mathcal{O}(\log \frac{1}{\epsilon} \log \lambda)$. Des détails supplémentaires sont donnés ici pour justifier de la probabilité de succès de l'étape.

Sachant que le nombre de premier dans l'intervalle $[\lambda, 2\lambda]$ est d'au moins $\frac{3}{5}\lambda/ln\lambda$ fait 1.12 et qu'ils sont tous impairs, la probabilité qu'un nombre impair choisit aléatoirement dans cette intervalle ne soit pas premier est d'au plus $1 - \frac{6}{5 \ln \lambda}$. La probabilité qu'après k tirages aléatoires, il n'y ait eu aucun nombre premier est donc d'au plus $(1 - \frac{6}{5 \ln \lambda})^k < e^{\frac{6k}{5 \ln \lambda}}$. D'où une probabilité d'échec d'au plus $\epsilon/4$ pour cette étape de l'algorithme puisque $k = \frac{5}{6} \ln \frac{4}{\epsilon} \ln \lambda$.

Ces quelques points sont résumés dans le lemme suivant.

Lemme 2.18. *La première partie de l'algorithme 1* TRIPLET*, lignes 1 à 5, s'effectue en* $\tilde{\mathcal{O}}(\log \frac{1}{\epsilon} \log^{11.5}(\lambda))$ *opérations binaires et fournit un nombre premier avec une probabilité d'au moins* $1 - \frac{\epsilon}{4}$.

2.2.1.2. La primalité de q

Une fois un nombre premier p obtenu, il faut trouver un autre nombre premier q tel que $p \,|\, (q-1)$ pour avoir un sous-groupe multiplicatif d'ordre p. Ainsi, le nombre premier q se trouve dans la suite arithmétique $p+1$, $2p+1$, $3p+1$, ... La question ne porte donc plus sur la densité des nombres premiers parmi un intervalle mais uniquement parmi cette suite arithmétique. D'après le théorème de la progression arithmétique de Dirichlet [Dir37], la distribution des nombres premiers dans cette suite arithmétique est *asymptotiquement* la même que leur distribution dans $\mathbb{Z}$. Le théorème de Bombieri-Vinogradov [Bom65] est plus précis en bornant les termes d'erreur. Ces résultats indiquent que théoriquement il suffit d'appliquer la même stratégie que pour un nombre premier choisi dans $\mathbb{Z}$. C'est-à-dire : prendre un entier a aléatoire, tester si $ap + 1$ est premier et répéter jusqu'à avoir effectivement un nombre premier. En revanche il manque des bornes pour savoir dans quel ensemble chercher a pour que cette démarche réussisse avec une bonne probabilité.

Des résultats plus récents de Akbary et Hambrook [AH15] et Sedunova [Sed18] donnent des bornes explicites et donc expoitables pour le théorème de Bombieri-Vinogradov.

Fait 2.19 ([Sed18, Corollary 1.5])**.** *Notons* $\pi(x)$ *le nombre d'entiers premiers* $\leq x$*,* $\pi(x; m, a)$ *le nombre d'entiers premiers* $\leq x$ *et égaux à* a *modulo* m*, et* $\ell(x)$ *le plus*

petit diviseur premier de x. Pour tout $\gamma \geq 4$ et $\lambda_1 \leq \lambda_2 \leq \gamma^{1/2}$,

$$\sum_{\substack{m \leq \lambda_2 \\ \ell(m) > \lambda_1}} \max_{2 \leq y \leq \gamma} \max_{a : \gcd(a,m)=1} \left| \pi(y; m, a) - \frac{\pi(y)}{\phi(m)} \right|$$
$$\leq 122.77 \left(14 \frac{\gamma}{\lambda_1} + 4\gamma^{1/2}\lambda_2 + 15\gamma^{2/3}\lambda_2^{1/2} + 4\gamma^{5/6} \ln(\frac{\lambda_2}{\lambda_1}) \right) (\ln \gamma)^{7/2}.$$

À partir de ce fait, il est possible de déduire des résultats probabilistes sur la quantité de nombres premiers dans une progression arithmétique.

Corollaire 2.20. *Soit $0 < \epsilon < \frac{1}{2}$, $\lambda \geq \frac{2^{54}}{\epsilon^2}$, et p un nombre premier choisi aléatoirement dans $[\lambda, 2\lambda]$. Le nombre d'entiers premiers $q \leq \lambda^6$ de la forme $q = ap + 1$ est supérieur ou égal à $\lambda^5/(24 \ln \lambda)$ avec une probabilité d'au moins $1 - \epsilon$.*

Démonstration. Il s'agit d'appliquer le fait 2.19 en prenant $\lambda_1 = \lambda$, $\lambda_2 = 2\lambda$ et $\gamma = \lambda^6$. La somme s'effectue seulement sur des nombres entiers puisqu'ils doivent vérifier $\ell(m) > \lambda_1$ alors que $\lambda_1 \geq m/2$. En prenant $y = \gamma$ et $a = 1$, la somme est simplifiée et surtout ne peut qu'être plus petite. La relation devient alors

$$\sum_{\substack{\lambda < p < 2\lambda \\ p \text{ premier}}} \left| \pi(\lambda^6; p, 1) - \frac{\pi(\lambda^6)}{p-1} \right| \leq 1.09 \cdot 10^6 (\lambda^5 + 1.27\lambda^{4.5} + 0.48\lambda^4)(\ln \lambda)^{7/2}.$$

Pour $\lambda \geq 2^8$, la somme est bornée par $1.2 \cdot 10^6 \lambda^5 (\ln \lambda)^{7/2}$.

Comptons maintenant le nombre de *mauvais* premiers dans l'intervalle $[\lambda, 2\lambda]$. Ce sont des premiers p tels que $\pi(\lambda^6; p, 1) \leq \lambda^5/(24 \ln \lambda)$. Puisque $\pi(\lambda^6) \geq \lambda^6/(6 \ln \lambda)$, si p est un mauvais premier dans $[\lambda, 2\lambda]$, alors $\pi(\lambda^6)/(p-1) \geq \pi(\lambda^6; p, 1)$ et puisque $p - 1 \leq 2\lambda$,

$$\left| \pi(\lambda^6; p, 1) - \frac{\pi(\lambda^6)}{p-1} \right| \geq \frac{\lambda^6/(6 \ln \lambda)}{p-1} - \frac{\lambda^5}{24 \ln \lambda} \geq \frac{\lambda^5}{24 \ln \lambda}.$$

S'il y a k mauvais premiers, alors la somme sur les nombres premiers vaut au moins $k\lambda^5/24 \ln \lambda$. Cette somme valant par ailleurs au plus $1.2 \cdot 10^6 \lambda^5 (\ln \lambda)^{7/2}$, il est possible d'en déduire une borne sur k :

$$k \leq \frac{1.2 \cdot 10^6 \lambda^5 (\ln \gamma)^{7/2}}{\lambda^5/(24 \ln \lambda)} = 2.88 \cdot 10^7 (\ln \lambda)^{9/2}.$$

Puisqu'il y a au moins $\frac{3}{5}\lambda/\ln \lambda$ nombres premiers distincts dans l'intervalle $[\lambda, 2\lambda]$ (fait 1.12), la probabilité qu'un nombre premier choisi aléatoirement dans cet intervalle soit un mauvais premier est d'au plus

$$\frac{2.88 \cdot 10^7 (\ln \lambda)^{9/2}}{\frac{3}{5}\lambda/\ln \lambda} = 4.8 \cdot 10^7 \lambda^{-1} (\ln \lambda)^{11/2}.$$

Cette probabilité tend vers zéro lorsque λ tend vers l'infini. En particulier pour $\lambda \geq 2^{55}$, elle est inférieure à $2^{27}\lambda^{-1/2}$. Ainsi, pour obtenir une probabilité d'au plus ϵ, il convient de prendre $\lambda \geq \frac{2^{54}}{\epsilon^2}$ (ce qui est bien supérieur à $> 2^{55}$ pour $\epsilon \leq \frac{1}{2}$). □

Le fait de considérer des nombres premiers q inférieurs à λ^6 et non une autre puissance de λ permet d'obtenir la meilleure borne possible à partir du fait 2.19. Une probabilité explicite ayant été donnée, il est maintenant possible d'analyser la seconde partie de l'algorithme 1 consistant à trouver le nombre premier q après avoir trouvé le nombre premier p.

Lemme 2.21. *Si la première partie, lignes 1 à 5, de l'algorithme 1* Triplet *a calculé un nombre premier $p \in [\lambda, 2\lambda]$, la seconde partie, lignes 6 à 10, s'effectue en temps $\tilde{\mathcal{O}}(\log \frac{1}{\epsilon} \log^{11.5} \lambda)$ et parvient à générer un nombre premier q avec une probabilité d'au moins $1 - \frac{\epsilon}{2}$*

Démonstration. Puisque $\lambda \geq \frac{2^{54}}{(\epsilon/4)^2}$ et que l'algorithme est bien parvenu à produire un nombre premier p choisi aléatoirement dans $[\lambda, 2\lambda]$, avec une probabilité d'au moins $1 - \frac{\epsilon}{4}$, il y a plus de $\lambda^5/(24 \ln \lambda)$ nombres premiers $q \leq \lambda^6$ de la forme $ap + 1$ d'après le corollaire 2.20.

Si p satisfait bien cette condition et que a est aléatoirement choisi parmi les entiers pairs ($(2a + 1)p + 1$ étant pair donc non premier et inutile à tester) entre 1 et λ^5, la probabilité que $ap + 1$ soit premier est d'au moins $\frac{1}{12 \ln \lambda}$. Et donc répéter ce tirage aléatoire $12 \ln \frac{4}{\epsilon} \ln \lambda$ fois permet d'augmenter cette probabilité à $1 - \frac{\epsilon}{4}$ comme dans le cas du lemme 2.18.

La probabilité totale de succès de cette étape est donc d'au moins $1 - \frac{\epsilon}{2}$ en prenant en compte la possibilité que p soit un mauvais premier et celle qu'un nouveau nombre premier q ne soit pas trouvé.

La complexité est donnée par la répétition des tests de primalité AKS, et est donc en $\tilde{\mathcal{O}}(\log \frac{1}{\epsilon} \log^{11.5} \lambda)$ puisque la taille des différents $ap + 1$ est essentiellement la taille de λ. □

La probabilité de succès est en réalité légèrement meilleure que la borne puisque la preuve ne regarde que les nombres premiers inférieurs à λ^6 alors que, vu le choix de a, q est compris entre $2p + 1$ et $p\lambda^5$, avec p pouvant aller jusqu'à $2\lambda - 1$.

2.2.1.3. L'ordre multiplicatif de ω

Pour la dernière partie de l'algorithme, la démarche est la même. En considérant que les deux premières parties ont réussi, il s'agit d'estimer la probabilité qu'un élément aléatoire de $\mathbb{F}_q^*$ ne permette pas de trouver une racine primitive p-ième de l'unité. Or, dans $\mathbb{F}_q$, le polynôme $x^{\frac{q-1}{p}} - 1$ a au plus $\frac{q-1}{p}$ racines. Par conséquent la probabilité que $\alpha^{(q-1)/p} = 1$ est d'au plus $\frac{1}{p}$ pour α choisi aléatoirement dans $\mathbb{F}_q^*$. Encore une fois la répétition du

tirage associée à un test déterministe permet de réduire la probabilité d'échec, à $\frac{\epsilon}{4}$ pour $\log_p \frac{4}{\epsilon}$ répétitions.

Par ailleurs, l'arithmétique s'effectuant dans $\mathbb{F}_q$ avec une exponentiation de puissance inférieure à q, son coût est de $\tilde{\mathcal{O}}(\log^2 q) = \tilde{\mathcal{O}}(\log^2 \lambda)$ opérations binaires.

Lemme 2.22. *Si les premières parties de l'algorithme 1* Triplet *sont parvenues à founir les premiers p et q, la dernière, lignes 11 à 15, calcule $\omega \in \mathbb{F}_q$ d'ordre p avec une probabilité d'au moins $1 - \frac{\epsilon}{4}$ et en $\tilde{\mathcal{O}}(\log \frac{1}{\epsilon} \log^2 \lambda)$.*

Ce troisième lemme permet de déduire que l'algorithme renvoie un triplet (p, q, ω) ayant les trois propriétés du théorème 2.17 en temps $\tilde{\mathcal{O}}(\log \frac{1}{\epsilon} \log^{11.5} \lambda)$ et avec une probabilité d'échec au plus $\epsilon = \frac{\epsilon}{4} + \frac{\epsilon}{2} + \frac{\epsilon}{4}$.

2.2.1.4. La coprimalité de q et N

Il reste à prouver la dernière propriété de q dans le théorème 2.17 à savoir qu'il ne divise probablement pas un nombre N si la valeur de λ est suffisamment grande.

Lemme 2.23. *Soit N un entier supérieur à 0 et q le second nombre premier généré par* Triplet(λ, ϵ) *avec $0 < \epsilon < 1$ et $\lambda \geq \max(\frac{2^{58}}{(\epsilon/2)^2}, \sqrt[5]{\frac{48}{\epsilon} \ln N})$. Avec une probabilité d'au moins $1 - \epsilon$, q ne divise pas N.*

Démonstration. Ce point vient encore de la densité des nombres premiers dans la suite arithmétique générée par p. Puisque $\lambda \geq \frac{2^5}{(\epsilon/2)^2}$, il y a au moins $\lambda^5/(24 \ln \lambda)$ premiers $q \leq \lambda^6$ de la forme $ap + 1$ avec une probabilité d'au moins $1 - \frac{\epsilon}{2}$ en considérant qu'un nombre premier p a bien été généré. Parmi eux, il y en a au plus $\log_\lambda N$ qui divisent N car ils sont tous plus grands que λ.

Ainsi, la probabilité que l'un d'entre eux choisit aléatoirement divise N est d'au plus $24 \log_\lambda N \ln \lambda / \lambda^5 \leq \frac{\epsilon}{2}$ car $\lambda^5 \geq \frac{48}{\epsilon} \ln N$. □

2.2.2. Approche heuristique

L'algorithme Triplet, nécessite dans sa version formelle un λ tellement grand ($> 2^{58}$) qu'il ne peut avoir d'application pratique. Cette borne vient de la mise en application de versions effectives du théorème de Dirichlet [Sed18]. Ainsi en théorie, il faut choisir un grand nombre premier p pour que celui-ci ait de bonnes chances d'avoir une quantité suffisante de nombres premiers dans sa suite arithmétique $(2ap + 1)_{a \in \mathbb{N}^*}$ qui soient inférieurs à p^6.

En pratique, des expérimentations montrent que cela peut être vérifié pour des valeurs de p bien plus faibles. Ces expérimentations on été effectuées de la manière suivante. La proportion de nombres premiers dans la portion de suite arithmétique $\{2ap+1 : 1 \leq a \leq p/2\}$ a été estimée pour tous les nombres premiers ayant de 10 à 20 bits (81 928 premiers). Pour ceux d'au plus 14 bits, le calcul a été fait de manière exacte en testant la primalité de

tous les nombres de la suite. Pour les plus grands premiers, la proportion est estimée par le tirage aléatoire d'au moins 1000 éléments dans la suite dont la primalité a été testée. Ceci a été fait dans le but de comparer cette proportion avec celle d'une version forte du théorème de Dirichlet due à de la Vallée Poussin qui indique qu'asymptotiquement cette proportion est $\mathcal{O}(\frac{1}{\ln p})$.

La table 2.1 donne, en classant les premiers par leur nombre de bits, les proportions les plus petites, les plus grandes, moyennes et théoriques pour chaque classe. La figure 2.1 montre la répartition de cette proportion pour tous les nombres premiers autour de la valeur théorique. Le code écrit en SageMath utilisé pour ces expérimentations est donné en annexe A.

Ces observations montrent des proportions de premiers parmi les nombres de la forme $2ap+1$ dont la moyenne est toujours proche de la théorie bien que légèrement inférieure et semblent indiquer que l'algorithme Triplet pourrait avoir un taux de réussite presque aussi élevé en partant avec $\lambda > 2^{10}$ plutôt que $\lambda > 2^{58}$.

Il existe encore une autre approche pour pouvoir travailler avec des λ plus petits. Il s'agirait simplement de prendre pour q le plus petit nombre premier dans la suite arithmétique $(2ap+1)_{a\in\mathbb{N}^*}$ et non un premier aléatoire. L'algorithme 1 Triplet serait modifié en utilisant les instructions suivantes pour générer q.

$q \leftarrow 2p+1$
Tant que q n'est pas premier **faire**
    $q \leftarrow q+2p$

La complexité n'en serait pas modifiée à condition que la boucle ne soit répétée que $\mathcal{O}(12\ln\frac{4}{\epsilon}\ln\lambda)$ fois. Par ailleurs cette partie est déterministe contrairement à l'approche d'origine qui peut renvoyer échec.

Pour déterminer le nombre de répétitions de la boucle et donc l'efficacité de cette modification, il faut déterminer la taille du plus petit nombre premier de la forme $2ap+1$. Une conjecture de Heath-Brown [Hea78] considère qu'il est atteint pour un $a \ll \ln^2 p$ et ce pour tout nombre premier p. La recherche de bornes sur ce a a donné lieu a beaucoup de conjectures tachant de coller toujours plus avec des observations numériques [Wag79 ; Pom80 ; GP90 ; LPS17]. Ces différentes hypothèses (qui concernent généralement le maximum des plus petits premiers dans chaque suite arithmétique $(ap+k)_a$ et pas seulement le cas $k=1$) tendent aussi à considérer comme justifié que a vaille au plus $\ln^2 p$. C'est donc l'hypothèse qui sera faite ici aussi.

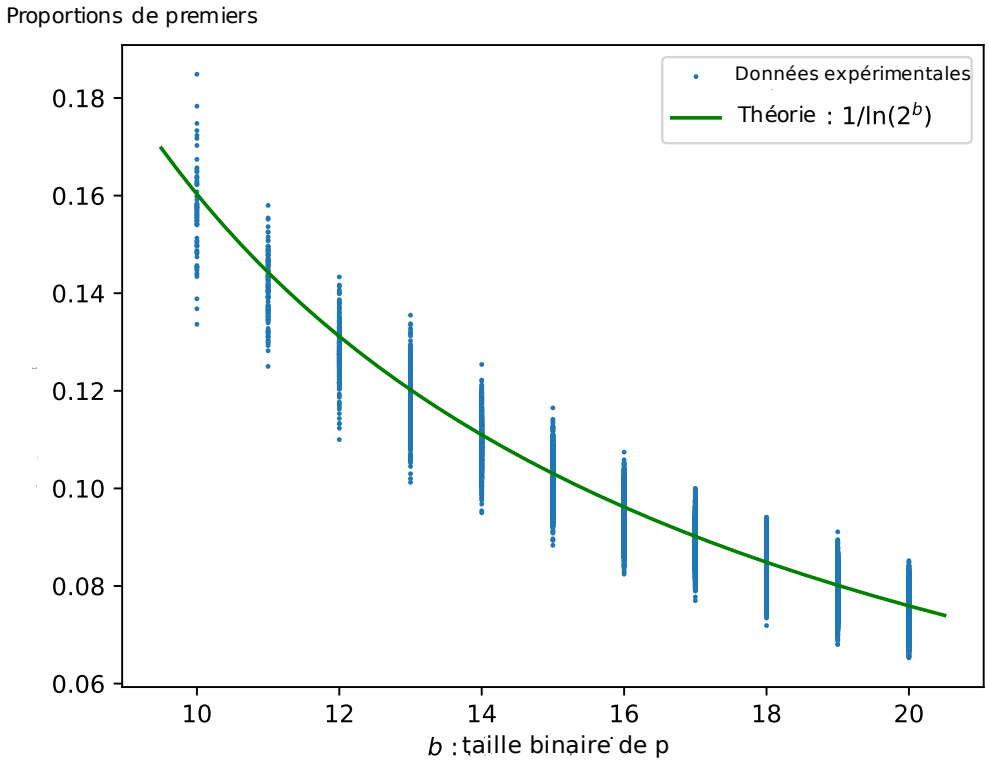
Hypothèse 2.1 (Sur le plus petit nombre premier dans une suite arithmétique)**.** Pour p un nombre premier, $\min\{a | 2ap+1 \text{ est premier}\} \leq \ln^2 p$.

Sous cette hypothèse, il est alors possible d'analyser l'apport que représente cette modification à l'algorithme 1 Triplet.

TABLE 2.1. – Proportion de nombres premiers dans $\{2ap+1 : 1 \leq a \leq p/2\}$.

Nombre de bits	Minimale	Moyenne	Maximale	Théorique ($\simeq \frac{1}{\ln p}$)
10	13.36%	15.63%	18.49%	16.03%
11	12.50%	14.12%	15.80%	14.43%
12	11.00%	12.79%	14.33%	13.12%
13	10.13%	11.79%	13.55%	12.02%
14	9.50%	10.93%	12.54%	11.10%
15	8.83%	10.14%	11.65%	10.30%
16	8.24%	9.46%	10.74%	9.62%
17	7.70%	8.89%	10.00%	9.02%
18	7.19%	8.36%	9.41%	8.49%
19	6.80%	7.91%	9.11%	8.01%
20	6.53%	7.49%	8.52%	7.59%

FIGURE 2.1. – Proportions de nombre premiers dans $\{2ap+1 : 1 \leq a \leq p/2\}$.



Théorème 2.24. *Sous l'hypothèse 2.1, en prenant $\lambda \geq 21$ et en remplaçant la génération aléatoire de q, lignes 6 à 10, par la recherche du plus petit premier de la forme $2ap+1$, l'algorithme* Triplet(λ, ϵ) *génère un triplet (p, q, ω) ayant les mêmes propriétés que dans le théorème 2.17 et ce avec une probabilité d'au moins $1 - \frac{\epsilon}{2}$ et sinon renvoie* échec.

Sa complexité est en $\tilde{\mathcal{O}}(\max(\log \frac{1}{\epsilon}, \log \lambda) \log^{11.5} \lambda)$ opérations binaires.

De plus, si $\lambda \geq \frac{5}{3\epsilon} \ln N$ pour un entier $N > 0$, la probabilité que q divise N est d'au plus ϵ.

Démonstration. La borne 21 sur λ est là pour assurer qu'il y ait suffisamment de nombres premiers dans l'intervalle $[\lambda, 2\lambda]$. La probabilité de succès a augmenté car la seconde partie qui a une probabilité d'échec de $\frac{\epsilon}{2}$ a été remplacée par une partie dont le résultat est déterministe. La complexité est justifiée par l'hypothèse qui dit que le nombre de répétitions de la nouvelle boucle sera au pire $\ln^2(2\lambda)$. Ainsi, il y a dans la première partie, lignes 1 à 5, $\mathcal{O}(\log \frac{1}{\epsilon} \log \lambda)$ répétitions du test AKS et dans la seconde il y en a $\mathcal{O}(\log^2 \lambda)$. Les tests sont toujours effectués sur des entiers de taille $\mathcal{O}(\log \lambda)$. C'est le maximum entre ces deux parties qui donne la complexité, la troisième partie restant dominée.

Enfin, pour la probabilité que q ne divise pas N, il suffit de constater que tirer aléatoirement p premier dans l'ensemble des premiers inclus dans $[\lambda, 2\lambda]$ revient exactement à tirer aléatoirement q parmi $\{2ap+1$ premiers $|p$ premiers $\in [\lambda, 2\lambda]\}$. Les deux ensembles ont exactement la même taille puisque $2a \leq \ln^2 p \leq \ln^2(2\lambda) < 2\lambda$ pour $\lambda > 21$ et donc $2ap+1$ ne peut pas être le plus petit nombre premier dans la suite arithmétique de deux p distincts de $[\lambda, 2\lambda]$. Le corollaire 2.2, sur l'intervalle $[\lambda, 2\lambda]$ pouvant être étendu à tout ensemble ayant le même nombre de premiers tous supérieurs à λ, le choix de la valeur de λ donne bien une probabilité d'au moins $1 - \epsilon$ que q ne divise pas N. □

La complexité n'augmente que peu, en remplaçant potentiellement $\log \frac{1}{\epsilon}$ par $\log \lambda$ et reste polynomiale en $\log \lambda$. Cette méthode est donc en pratique, et tant que l'hypothèse tient, la manière la plus efficace de générer un triplet (p, q, ω) avec de petites valeurs pour p.

2.3. Évaluation d'un produit de polynôme dans un anneau quotient

Les méthodes que nous venons de voir pour borner la taille des polynômes manipulés consistent principalement à passer dans un anneau quotient $\mathbb{A}[x]/\langle P(x)\rangle$ pour un polynôme défini dans $\mathbb{A}[x]$ avec P un polynôme unitaire. Passer dans l'anneau quotient n'impacte pas la complexité de l'addition de deux polynômes. Pour la multiplication de polynômes, il suffit de l'effectuer dans $\mathbb{A}[x]$ puis de réduire le résultat en le divisant par P, la division ayant asymptotiquement le même coût que la multiplication, cela n'impacte pas la complexité de l'opération. Cela ne se passe pas aussi bien pour l'évaluation de

polynômes. L'évaluation d'un polynôme dans $\mathbb{A}[x]$ sur un point α de $\mathbb{A}$ est une opération qui commute avec l'addition et la multiplication : $\left(\sum F_i G_i\right)(\alpha) = \sum F_i(\alpha) G_i(\alpha)$. En passant dans l'anneau quotient $\mathbb{A}[x]/\langle P\rangle$ c'est le produit qui pose des difficultés et il devient faux de dire qu'évaluer un produit revient à faire le produit des évalués, la commutativité est perdue sur la multiplication. Calculer $(F \bmod P)(\alpha) \times (G \bmod P)(\alpha)$ ne peut correspondre à l'évaluation de $(FG) \bmod P$ puisqu'il manque la réduction finale modulo P. C'est simplement l'évaluation de $(F \bmod P)(G \bmod P)$, un polynôme pouvant être de degré $2\deg(P) - 2$.

Bien sûr, il est possible d'effectuer l'évaluation du produit modulaire $(FG) \bmod P$ en calculant d'abord FG puis la réduction modulo P et enfin en effectuant l'évaluation. Cependant il est classique d'utiliser l'évaluation pour vérifier une opération et c'est notamment ce que nous ferons dans le chapitre 4 pour vérifier un produit modulaire. Il est donc nécessaire de savoir évaluer $(FG) \bmod P$ en un point α sans calculer $FG \bmod P$ et le plus efficacement possible pour que la vérification puisse être plus rapide que le calcul du produit modulaire. Pour assurer une bonne probabilité de succès aux algorithmes de vérifications basés sur l'évaluation, il est souvent nécessaire d'évaluer les polynômes sur un point α venant d'une extension $\mathbb{A}_{\text{ext}}$ de $\mathbb{A}$ et non de $\mathbb{A}$ ce qui peut changer le coût de certaines opérations. Cette section présente des algorithmes pour évaluer un produit modulaire $(FG) \bmod P$ dans $\mathbb{A}[x]/\langle P(x)\rangle$ sur un point α dans $\mathbb{A}$ ou une extension $\mathbb{A}_{\text{ext}}$ et ce sans calculer le produit $(FG) \bmod P$ lui-même. Le polynôme P est toujours considéré comme un polynôme creux unitaire et l'évaluation est effectuée pour des polynômes F et G aussi bien creux que denses.

La section 2.3.1 décrit la méthode dans le cas plus simple où le polynôme P est un binôme $P = x^d - 1$ aussi appelé réduction cyclique. Ce cas particulier permet d'illustrer et d'appréhender plus facilement la méthode générale et correspond aux diverses applications de la vérification d'un produit modulaire dans cette thèse (voir section 4.3 et chapitre 5). Les algorithmes obtenus effectuent un nombre linéaire d'opérations dans $\mathbb{A}$, que les polynômes F et G soient denses ou creux. Cette méthode est ensuite généralisée au cas d'un produit de polynômes denses modulo un polynôme creux en section 2.3.2 de forme plus générale et enfin au cas ou les polynômes F et G sont aussi creux en section 2.3.3. Cette section présente des algorithmes dans un cadre aussi général que possible, aussi les algorithmes seront analysés en nombre d'opérations arithmétiques et non en nombre d'opérations binaires qui dépendent du coût de l'arithmétique dans $\mathbb{A}$.

2.3.1. Évaluation de la réduction cyclique d'un produit

Le premier cas considéré est celui de l'évaluation de $FG \bmod P$ avec $P = x^d - 1$ pour F et G deux polynômes denses de degré au plus $d - 1$. L'approche utilisée ici, s'appuie sur celle proposée par Pascal Giorgi [Gio18] pour l'évaluation d'un produit médian $((FG) \operatorname{div} x^d) \bmod x^d$. L'observation essentielle à cette évaluation est qu'un produit de polynômes correspond au produit entre une matrice Toeplitz et un vecteur et qu'il en est donc de même pour le produit médian. L'évaluation est alors une projection par le

vecteur des puissances du point d'évaluation. En s'appuyant sur la structure Toeplitz de la matrice, il est alors possible de procéder efficacement à l'évaluation.

Un produit de polynômes modulo $x^d - 1$ peut aussi se décrire comme un produit entre une matrice Toeplitz et un vecteur. Une approche similaire s'applique donc. L'algorithme 2 ÉvaluationRéductionCyclique s'appuie sur cette méthode pour effectuer l'évaluation modulaire. La construction de l'évaluation en passant par un produit matrice-vecteur n'apparaît pas directement dans l'algorithme mais permet de justifier sa validité.

Algorithme 2 ÉvaluationRéductionCyclique

Entrée : $F = \sum f_i x^i$, $G = \sum g_i x^i \in \mathbb{A}[x]$ avec $\deg(F), \deg(G) < d$, et $\alpha \in \mathbb{A}$.
Sortie : $(FG \bmod x^d - 1)(\alpha)$

1: $c \leftarrow F(\alpha)$
2: $P_\alpha \leftarrow \alpha^d - 1$
3: $\beta \leftarrow cg_0$
4: **Pour** $j = 1$ à $d - 1$ **faire**
5: $\quad c \leftarrow \alpha c - P_\alpha f_{d-j}$
6: $\quad \beta \leftarrow \beta + cg_j$
7: **Renvoyer** β

Théorème 2.25. *Étant donné deux polynômes $F, G \in \mathbb{A}[x]$ de degré au plus $d-1$ et $\alpha \in \mathbb{A}$, l'algorithme 2* ÉvaluationRéductionCyclique *calcule l'évaluation de (FG) mod $x^d - 1$ en α en effectuant $\mathcal{O}(d)$ opérations dans $\mathbb{A}$.*

Démonstration. Soit $H = FG$ et $M = H \bmod x^d - 1$. Les coefficients des monômes de degré i sont notés f_i (respectivement g_i, h_i, m_i) pour le polynôme F (respectivement G, H, M). Soit $\vec{g} = (g_0, \ldots, g_{d-1})$, $\vec{h} = (h_0, \ldots, h_{2d-2})$ et $\vec{m} = (m_0, \ldots, m_{d-1})$. Pour simplifier les notations, les transformations d'un vecteur dans $\mathbb{A}^d$ en matrice $1 \times d$ (ou $d \times 1$) pour des produits vecteur-matrice (ou matrice-vecteur) resteront implicites.

La multiplication d'un polynôme de degré $d - 1$ par un polynôme F fixé est une application linéaire décrite par une matrice Toeplitz de taille $(2d - 1) \times d$ contenant les coefficients de F. Plus précisément, $\vec{h} = \mathcal{T}_F \vec{g}$ avec

$$\mathcal{T}_F = \begin{pmatrix} f_0 & & & \\ f_1 & f_0 & & \\ \vdots & & \ddots & \\ f_{d-1} & \cdots & \cdots & f_0 \\ & f_{d-1} & & f_1 \\ & & \ddots & \vdots \\ & & & f_{d-1} \end{pmatrix}.$$

Puisque $M = H \bmod x^d - 1$, $m_i = h_i + h_{i+d-1}$ pour $0 \leq i < d-1$ et $m_{d-1} = h_{d-1}$. Ainsi,

$\vec{m} = \mathcal{C}_F \vec{g}$ avec $\mathcal{C}_F$ la matrice circulante de taille $d \times d$ décrite par

$$\mathcal{C}_F = \begin{pmatrix} f_0 & f_{d-1} & \cdots & f_1 \\ f_1 & f_0 & \cdots & f_2 \\ \vdots & \vdots & & \vdots \\ f_{d-1} & f_{d-2} & \cdots & f_0 \end{pmatrix}.$$

Par ailleurs, l'évaluation de M en α correspond au produit scalaire $\vec{\alpha}_d \vec{m}$ avec $\vec{\alpha}_d = (1, \alpha, \ldots, \alpha^{d-1})$. L'objectif est donc de calculer $\vec{\alpha}_d \mathcal{C}_F \vec{g}$. L'approche standard consiste à d'abord calculer $\vec{m} = \mathcal{C}_F \vec{g}$ c'est à dire M, et ensuite $\vec{\alpha}_d \vec{m} = M(\alpha)$. Comme cela a été remarqué par Pascal Giorgi [Gio18], changer l'ordre des opérations en effectuant d'abord le produit vecteur matrice $\vec{\alpha}_d \mathcal{C}_F$ permet d'obtenir un algorithme plus efficace qui exploite pleinement la structure de la matrice circulante $\mathcal{C}_F$.

Soit $\vec{c} = \vec{\alpha}_d \mathcal{C}_F$. Les coordonnées de c vérifient : $c_{j+1} = \sum_{\ell=0}^{d-1} \alpha^\ell f_{(\ell-j-1) \bmod d} = f_{d-j-1} + \alpha \sum_{\ell=0}^{d-2} \alpha^\ell f_{(\ell-j) \bmod d}$. Puisque pour $j > 0$, $\sum_{\ell=0}^{d-2} \alpha^\ell f_{(\ell-j) \bmod d} = c_j - \alpha^{d-1} f_{d-j-1}$, ceci conduit à la relation de récurrence

$$\begin{cases} c_{j+1} = \alpha c_j - P(\alpha) f_{d-j-1} & \text{pour } j \geq 0 \\ c_0 = F(\alpha) \end{cases} \tag{2.1}$$

avec $P = x^d - 1$ et $c_0 = F(\alpha)$.

Il est aisé de voir que s'appuyer sur une telle récurrence pour évaluer $(FG) \bmod P$ en α donne une complexité de $\mathcal{O}(d)$ opérations dans $\mathbb{A}$. En effet, une fois que c_0 et $P(\alpha)$ ont été calculés, les autres c_j peuvent être obtenus récursivement pour $\mathcal{O}(1)$ opérations à chaque fois. Par ailleurs les évaluations de F et P nécessitent $\mathcal{O}(d)$ opérations dans $\mathbb{A}$ (fait 1.8). □

Le décompte du nombre d'opérations effectuées par l'algorithme 2 ÉvaluationRéductionCyclique peut être plus précis. Comme cela a été évoqué, il arrive que l'évaluation soit calculée sur un point α d'une extension de $\mathbb{A}_{\text{ext}}$ et non sur un point de $\mathbb{A}$. L'algorithme calcule l'évaluation sur α exactement de la même façon. Cependant, dans ce cas là, il est important de distinguer les différents type d'opérations : celles qui sont effectuées dans l'extension $\mathbb{A}_{\text{ext}}$ et celles qui sont effectuées entre des éléments de $\mathbb{A}$ et des éléments de $\mathbb{A}_{\text{ext}}$. En pratique, ces opérations n'ont pas le même coût, celles dans $\mathbb{A}_{\text{ext}}$ sont plus coûteuses. Dans le corollaire suivant, une multiplication entre un élément de $\mathbb{A}$ et un élément de $\mathbb{A}_{\text{ext}}$ est appelé une *multiplication scalaire*. L'objectif est ici de minimiser le nombre de multiplications qui ne sont pas scalaires.

Corollaire 2.26. *Étant donnés deux polynômes F, $G \in \mathbb{A}[x]$ de degré au plus $d-1$ et $\alpha \in \mathbb{A}_{ext}$ l'évaluation de $(FG) \bmod x^d - 1$ en α nécessite $2d-2$ multiplications et $3d-2$ additions dans $\mathbb{A}_{ext}$, plus $3d-2$ multiplications scalaires.*

Démonstration. Il faut d'abord calculer $\alpha^2, \alpha^3, \ldots, \alpha^d$ par $(d-1)$ multiplications. Ensuite, le calcul de $F(\alpha)$ ne nécessite plus que $(n-1)$ multiplications scalaires et additions, et celui de $P(\alpha) = \alpha^n - 1$ seulement une addition supplémentaire. Pour la valeur initiale cg_0 de β, il ne faut qu'une multiplication scalaire. À partir de là, chaque itération de la boucle nécessite une multiplication, deux multiplications scalaires et deux additions. Ainsi, l'évaluation totale requiert $3d-2$ additions, $2d-2$ multiplications et $3d-2$ multiplications scalaires. □

Ce corollaire a pour objectif de minimiser le nombre d'opérations dans $\mathbb{A}_{\text{ext}}$ car elles sont plus coûteuses. Si α est simplement un point de $\mathbb{A}$, l'optimisation de l'algorithme 2 ÉvaluationRéductionCyclique se fait plutôt sur le nombre total d'opérations qui sont toutes dans $\mathbb{A}$. L'approche est donc différente. L'évaluation de F doit plutôt se baser sur le schéma de Horner ce qui donne $(d-1)$ multiplications et $(d-1)$ additions. Le calcul d'α^d ensuite repose sur l'exponentiation rapide avec au plus $2\log d$ multiplications. Il ne faut qu'une addition supplémentaire pour obtenir $P(\alpha)$. La boucle de la fin de l'algorithme s'exécute de la même manière. Au total cela donne $3d-2$ additions et $4d-3+2\log d$ multiplications. Le nombre d'additions est le même mais le nombre total de multiplication est inférieur. Cette approche donnerait cependant plus de multiplications non scalaires si α était dans $\mathbb{A}_{\text{ext}}$.

Considérons maintenant le cas où F et G sont données en représentation creuse et les changements que cela apporte à l'algorithme.

Théorème 2.27. *Étant donné deux polynômes creux F, $G \in \mathbb{A}[x]$ de degré $< d$ avec respectivement $\#F$ et $\#G$ monômes non nuls et $\alpha \in \mathbb{A}$, il est possible de calculer l'évaluation de $(FG) \bmod x^n - 1$ en α en effectuant $\mathcal{O}((\#F + \#G)\log d)$ opérations dans $\mathbb{A}$.*

Démonstration. Les notations sont reprises de la preuve du théorème 2.25. Si le support de G est l'ensemble $\text{support}(G) = \{j_0, \ldots, j_{\#G-1}\}$ avec $j_0 < \cdots < j_{\#G-1}$, le produit scalaire $\vec{c} \cdot \vec{g}$ est égal à $\sum_{k=0}^{\#G-1} c_{j_k} g_{j_k}$. Par conséquent, il est inutile de calculer toutes les coordonnées de $\vec{c}$, seules les $\#G$ valeurs $c_{j_0}, \ldots, c_{j_{\#G-1}}$ qui correspondent au support de G sont utiles pour obtenir le résultat final.

En appliquant la relation de récurrence (2.1) aussi souvent que nécessaire pour passer d'une des coordonnées pertinentes à la suivante, on obtient une nouvelle relation de récurrence, indexée par le support de G :

$$\begin{cases} c_{j_{k+1}} = \alpha^{j_{k+1}-j_k} c_{j_k} - P(\alpha) \sum_{\ell=1+j_k}^{j_{k+1}} \alpha^\ell f_{d-\ell} & \text{pour } k \geq 0 \\ c_{j_0} = ((x^{j_0} F) \bmod x^d - 1)(\alpha). \end{cases} \tag{2.2}$$

La valeur initiale c_{j_0} peut être calculée en $\mathcal{O}(\#F \log d)$ opérations. En effet le polynôme $(x^{j_0}F) \bmod x^d - 1$ est un polynôme de degré au plus $d-1$ ayant $\#F$ monômes non nuls.

L'évaluation en $\alpha \in \mathbb{A}$ nécessite donc $\#F$ exponentiations de α avec des exposant valant au plus $d-1$ plus $\#F-1$ additions dans $\mathbb{A}$. Par ailleurs les coefficients de $(x^{j_0}F) \bmod x^d - 1$ sont ceux de F et ses exposants sont obtenus en ajoutant j_0 aux exposants de F et en retranchant d s'il est nécessaire de réduire modulo $x^f - 1$. Calculer le polynôme $(x^{j_0}F) \bmod x^d - 1$ nécessite donc $\mathcal{O}(\#F)$ additions et soustractions sur des entiers bornés par $2d$, c'est-à-dire $\mathcal{O}(\#F \log d)$ opérations binaires. Les éléments de $\mathbb{A}$ ayant au moins un bit, ce coût est négligeable par rapport aux $\mathcal{O}(\#F \log d)$ opérations dans $\mathbb{A}$.

Par ailleurs, puisque F aussi est creux, la valeur de $f_{d-\ell}$ est en fait zéro dans de nombreux cas. Un coefficient non nul f_t de F apparaît dans la définition de $c_{j_{k+1}}$ si et seulement si $d - j_{k+1} \leq t < d - j_k$. Ainsi, chaque f_t est utilisé exactement une fois lors du calcul de l'ensemble des c_{j_k} pour $k \neq 0$. Puisque pour chaque terme non-nul de la somme de la relation (2.2), il est aussi nécessaire de calculer α^ℓ pour une valeur $\ell < d$, le coût total du calcul de toutes ces sommes est de $\mathcal{O}(\#F \log d)$ opérations dans $\mathbb{A}$. De même, le calcul de $\alpha^{j_{k+1}-j_k} c_{j_k}$ pour tout $k \in [0, \#G - 2]$ requiert $\mathcal{O}(\#G \log d)$ opérations dans $\mathbb{A}$ plus $\#G - 1$ additions d'entiers inférieurs à d pour obtenir les exposants. Encore une fois, le coût des opérations sur les entiers est négligeable par rapport à celui des opérations dans $\mathbb{A}$. La dernière étape est celle du produit scalaire $\vec{c} \cdot \vec{g}$ qui nécessite $\mathcal{O}(\#G)$ opérations dans $\mathbb{A}$ ce qui donne bien la complexité annoncée. □

Comme dans le cas dense, la complexité pourrait être détaillée entre additions, multiplications et multiplications scalaires si α se trouve en fait dans une extension $\mathbb{A}_{\text{ext}}$. Mais dans le cas creux, le coût est totalement dominé par les opérations sur α pour les différentes exponentiations qui sont toutes dans $\mathbb{A}_{\text{ext}}$. Il est cependant possible d'améliorer la complexité en n'effectuant pas les exponentiations les unes après les autres mais en utilisant la technique d'exponentiation simultanée de Yao [Yao76] pour les calculer toutes d'un coût. Ainsi seules $\log d + \mathcal{O}((\#F + \#G) \log d / \log\log d)$ multiplications dans $\mathbb{A}_{\text{ext}}$ sont nécessaires. Une fois les puissances de α calculées, même de manière rapide, les autres étapes de l'évaluation effectuent toujours $\mathcal{O}(\#F + \#G)$ opérations que ce soit des additions, des multiplications scalaires ou des additions d'entiers. Seules les opérations sur les entiers ne sont plus négligeables quand on les compare au calcul efficace des puissances de α.

Corollaire 2.28. *Étant donnés deux polynômes creux F, $G \in \mathbb{A}[x]$ de degré $< d$ et $\alpha \in \mathbb{A}_{ext}$, il est possible de calculer l'évaluation de $(FG) \bmod x^n - 1$ en α en effectuant $\log d + \mathcal{O}((\#F + \#G) \log d / \log\log d)$ opérations dans $\mathbb{A}_{ext}$ plus $\#F + \#G - 1$ additions d'entiers de $\mathcal{O}(\log d)$ bits.*

2.3.2. Évaluation d'un produit modulaire dense

Cette section présente la généralisation de l'algorithme 2 ÉvaluationRéduction-Cyclique à l'évaluation d'un polynôme $FG \bmod P$ où P est un polynôme creux unitaire quelconque et F et G sont deux polynômes denses de degré strictement inférieur à $d = \deg(P)$.

Nous présentons d'abord la méthode d'évaluation d'un produit modulaire adaptée de l'évaluation modulo un binôme. En particulier nous décrivons de nouvelles relations de récurrence propres à ce cadre plus général. Elles sont nécessaires pour justifier les algorithmes 3 CoefficientsDominants et 4 ÉvaluationModulaire qui permettent l'évaluation du produit modulaire. Ces algorithmes seront donnés dans un second temps.

Les étapes suivies pour l'évaluation en un point α de $FG \bmod P$ avec P quelconque sont les mêmes que celles vues pour l'évaluation avec $P = x^d - 1$.

1. Considérer la matrice $\mathcal{C}_{F \bmod P}$ de l'application linéaire correspondant à la multiplication par F dans $\mathbb{A}[x]/\langle P\rangle$;
2. Calculer d'abord $\vec{c} = \vec{\alpha}\mathcal{C}_{F \bmod P}$ en s'appuyant sur une relation de récurrence ;
3. Calculer le produit scalaire $\vec{c} \cdot \vec{g}$.

En reprenant les notations vectorielles de la section 2.3.1.

Lorsque P n'est pas un binôme $x^d - 1$, la matrice $\mathcal{C}_{F \bmod P}$, n'est plus une matrice circulante et la relation de récurrence n'est donc pas aussi directe. Les colonnes de la matrice correspondent aux polynômes $F^{[i]}$ avec $F^{[i]} = (x^i F) \bmod P$. On peut en effet vérifier que $FG \bmod P = \sum_{i=0}^{d-1} g_i F^{[i]}$ pour $G = \sum_{i=0}^{d-1} g_i x^i$. L'évaluation de $FG \bmod P$ sur α en effectuant d'abord le produit $\vec{\alpha}\mathcal{C}_{F \bmod P}$ revient au calcul de l'égalité

$$(FG \bmod P)(\alpha) = \sum_{i=0}^{d-1} g_i F^{[i]}(\alpha). \tag{2.3}$$

L'objectif est donc d'avoir un moyen efficace de calculer tous les $F^{[i]}(\alpha)$. Pour cela, il faut remarquer que la définition des $F^{[i]}$ permet d'obtenir la relation de récurrence $F^{[i+1]} = (xF^{[i]}) \bmod P$. Puisque $\deg(F^{[i]}) = d-1$ et que P est unitaire, $(xF^{[i]}) \bmod P = xF^{[i]} - f^{[i]}_{d-1}P$ avec $f^{[i]}_{d-1}$ qui désigne le coefficient de degré $d-1$ de $F^{[i]}$. Par conséquent, les $F^{[i]}(\alpha)$ sont décrits par la relation de récurrence suivante

$$\begin{cases} F^{[i+1]}(\alpha) = \alpha F^{[i]}(\alpha) - f^{[i]}_{d-1}P(\alpha) & \text{pour } i \geq 0 \\ F^{[0]}(\alpha) = F(\alpha) \end{cases} \tag{2.4}$$

Pour pouvoir appliquer cette relation il convient de connaître $F(\alpha)$ et tous les coefficients $f^{[i]}_{d-1}$ des $F^{[i]}$ pour $0 < i < n-1$. L'évaluation en α se calcule directement à partir des coefficients de F. Pour la liste des coefficients des monômes de degré $d-1$ des $F^{[i]}$, un algorithme spécifique est nécessaire. Ce n'était pas le cas lorsque P était pris égal à $x^d - 1$ puisque ces coefficients étaient directement donnés par la relation $f^{[i]}_{d-1} = f_{d-1-i}$. Dans le cas général, leur calcul repose sur la relation de récurrence qui lie les $F^{[i]}$, $F^{[i+1]} = xF^{[i]} - f^{[i]}_{d-1}P$, et implique une relation entre leurs coefficients :

$$f^{[i+1]}_k = f^{[i]}_{k-1} - f^{[i]}_{d-1}p_k \tag{2.5}$$

pour $0 < k \leq d-1$ avec $P = \sum_{i=0}^{d} p_i x^i$.

En partant des $f_k^{[0]}$ (les coefficients de F), il est alors possible de calculer tous les $f_{d-1}^{[i]}$. Puisque P est un polynôme creux, l'équation (2.5) correspond en fait à une égalité $f_k^{[i+1]} = f_{k-1}^{[i]}$ dans de nombreux cas, si k n'est pas dans support(P). L'algorithme 3 CoefficientsDominants qui calcule les coefficients de degré $d-1$ de tous les $F^{[i]}$ en tient compte pour n'effectuer que les opérations nécessaires.

Algorithme 3 CoefficientsDominants

Entrée : $P = \sum p_i x^i$ et $F = \sum f_i x^i$ dans $\mathbb{A}[x]$, avec $\deg(F) < \deg(P) = d$ et P unitaire.

Sortie : Le vecteur $[f_{d-1}^{[0]}, f_{d-1}^{[1]}, \ldots, f_{d-1}^{[d-2]}]$, avec $f_{d-1}^{[i]}$ qui désigne le coefficient de degré $d-1$ de $F^{[i]} = (x^i F) \bmod P$.

1: $V \leftarrow [f_{d-1}, f_{d-2}, \ldots, f_1]$
2: **Pour** $i = 0$ à $d-2$ **faire**
3: **Pour** $k \in$ support(P) tel que $i < k < d$ **faire**
4: $V[i+d-k] \leftarrow V[i+d-k] - p_k V[i]$
5: **Renvoyer** V

Lemme 2.29. *L'algorithme 3* CoefficientsDominants *calcule bien le vecteur des coefficients de degré $d-1$. Il exécute $\mathcal{O}(d\#P)$ opérations dans $\mathbb{A}$.*

Démonstration. Le nombre d'opérations est clair : elles sont toutes effectuées lors de l'étape 4 qui est elle-même effectuée $\mathcal{O}(d\#P)$ fois. Il est d'ailleurs possible d'arrêter la boucle externe dès qu'il est certain qu'il n'existe plus de $k \in$ support(P) tel que $i < k < d$. Autrement dit pour i strictement supérieur à $\deg(x^d - P) - 1$. Ce changement n'affecterait cependant pas le nombre d'opérations nécessaires.

La validité de l'algorithme 3 CoefficientsDominants peut être prouvée en vérifiant par récurrence qu'après l'itération i de la boucle externe, la propriété $\mathcal{P}(i)$ est valide :

$$\mathcal{P}(i) = \text{"}V[j] = f_{d-1}^{[j]} \text{ pour tout } j \leq i+1 \text{ et } V[j] = f_{d-(j-i)}^{[i+1]} \text{ pour } j > i+1.\text{"}$$

Avant la première itération, $\mathcal{P}(-1)$ est valide puisqu'elle correspond à $V[j] = f_{d-j-1}^{[0]}$ pour tout j, et que $F = F^{[0]}$ par définition.

Supposons maintenant que $\mathcal{P}(i-1)$ soit valide. En particulier, $V[j] = f_{d-1}^{[j]}$ pour $j \leq i$. Au cours de l'itération i, seuls les coordonnées $V[i+1]$ à $V[d-2]$ du vecteur V peuvent être modifiées, les égalités sur les premières coordonnées restent donc valides. Pour $j > i$, $V[j] = f_{d-(j-i+1)}^{[i]}$ avant l'itération par hypothèse. Après, on obtient

$$V[j] = f_{d-(j-i+1)}^{[i]} - p_{d-j+i} V[i] = f_{d-j+i-1}^{[i]} - p_{d-j+i} f_{d-1}^{[i]}.$$

D'après l'équation (2.5) cela nous donne donc $V[j] = f_{d-j+i}^{[i+1]}$ ce qui implique que $\mathcal{P}(i)$ est valide.

Ainsi, après la dernière itération, $V[j] = f_{d-1}^{[j]}$ pour tout $j \leq d-2$, l'algorithme est donc correct. □

Avec les coefficients de F et les $d-1$ coefficients de degré $d-1$ des premiers polynômes $F^{[i]}$, il est alors possible de calculer les d évaluations $F^{[i]}(\alpha)$ et donc $(FG \bmod P)(\alpha)$. L'algorithme d'évaluation proposé considère le cas où α est un élément d'une extension $\mathbb{A}_{\text{ext}}$ de $\mathbb{A}$. Son analyse fait aussi la distinction entre les opérations dans $\mathbb{A}$ et celles dans $\mathbb{A}_{\text{ext}}$.

Algorithme 4 ÉvaluationModulaire

Entrée : $P = \sum p_i x^i$, $F = \sum f_i x^i$, $G = \sum g_i x^i \in \mathbb{A}[x]$ avec $\deg(F), \deg(G) < \deg(P) = d$, P creux et unitaire, et $\alpha \in \mathbb{A}_{\text{ext}}$.

Sortie : $(FG \bmod P)(\alpha)$

1: $V \leftarrow [f_{d-1}^{[0]}, \ldots, f_{d-1}^{[d-2]}]$ en utilisant CoefficientsDominants(P, F)
2: $P_\alpha \leftarrow P(\alpha)$
3: $F_\alpha \leftarrow F(\alpha)$
4: $\beta \leftarrow F_\alpha g_0$
5: **Pour** $i = 1$ à $d-1$ **faire**
6: $\quad F_\alpha \leftarrow \alpha F_\alpha - V[i-1] P_\alpha$
7: $\quad \beta \leftarrow \beta + F_\alpha g_i$
8: **Renvoyer** β

Théorème 2.30. *Étant donnés trois polynômes F, G et P dans $\mathbb{A}[x]$ avec $\deg(F), \deg(G) < \deg(P) = d$, P creux et unitaire, et $\alpha \in \mathbb{A}_{ext}$ une extenion de $\mathbb{A}$, l'algorithme 4* ÉvaluationModulaire *calcule bien $(FG \bmod P)(\alpha)$. Pour cela il effectue $\mathcal{O}(d\#P)$ opérations dans $\mathbb{A}$ et $\mathcal{O}(d)$ opérations dans $\mathbb{A}_{ext}$.*

Démonstration. En utilisant l'algorithme 3 CoefficientsDominants(P, F), le vecteur V contient bien les valeurs indiquées. L'étape 6 s'appuie sur l'équation (2.4) pour calculer $F_\alpha = F^{[i]}(\alpha)$. L'étape 7 utilise cette évaluation et l'équation (2.3) pour calculer, en effectuant la somme terme après terme, $(FG \bmod P)(\alpha)$. Ces considérations assurent que l'algorithme est correct.

La première étape nécessite $\mathcal{O}(d\#P)$ opérations dans $\mathbb{A}$ par le lemme 2.29. C'est la seule étape qui ne dépend pas de α. Les autres étapes effectuent $\mathcal{O}(d)$ opérations dans $\mathbb{A}_{\text{ext}}$ puisque dépendant de α. □

Comme précédemment, les opérations dans $\mathbb{A}_{\text{ext}}$ sont tantôt des multiplications scalaires par un élément de $\mathbb{A}$ tantôt des opérations (additions ou multiplications) entre deux éléments de $\mathbb{A}_{\text{ext}}$. Voici maintenant une analyse précise de la complexité qui minimise le nombre de multiplications non scalaires. Ceci permet de minimiser le nombre d'opérations dans $\mathbb{A}$ les multiplications non scalaires dans $\mathbb{A}$ correspondant généralement à des produits de polynômes dans $\mathbb{A}[x]$.

Corollaire 2.31. *Soient P, F, G et α comme dans l'algorithme 4* ÉvaluationModulaire. *Il est possible de calculer $(FG \bmod P)(\alpha)$ en effectuant au plus $2d-2$ multiplications et $(3d-5+\#P)$ additions dans $\mathbb{A}_{ext}$, $(3d-3+\#P)$ multiplications scalaires*

d'éléments de $\mathbb{A}_{ext}$ par des éléments de $\mathbb{A}$, et $(d-1)(\#P-1)$ multiplications et additions dans $\mathbb{A}$.

Démonstration. Le premier point est de remarquer que le nombre d'opérations effectuées par l'algorithme 3 CoefficientsDominants est d'au plus $(d-1)(\#P-1)$ multiplications et additions dans $\mathbb{A}$. Dans l'algorithme 4 ÉvaluationModulaire, il faut évaluer les polynômes P et F en α. Pour minimiser le nombre de multiplications non scalaires, il faut calculer toutes les puissance de α ($\alpha^2, \ldots, \alpha^d$) en $d-1$ multiplications dans $\mathbb{A}_{\text{ext}}$. Les évaluations sont ensuite calculées en procédant à $\#P-1$ multiplications scalaire et additions dans $\mathbb{A}_{\text{ext}}$ pour P, et $d-1$ multiplications scalaires plus $d-2$ additions dans $\mathbb{A}_{\text{ext}}$ pour F. Ensuite, l'initialisation de β et la boucle pour nécessitent $d-1$ multiplications et $2d-2$ additions dans $\mathbb{A}_{\text{ext}}$ plus $2d-1$ multiplications scalaires. Ceci donne bien $2d-2$ multiplications et $(3d-5+\#P)$ additions dans $\mathbb{A}_{\text{ext}}$ plus $(3d-3+\#P)$ multiplications scalaires à ajouter aux opérations dans $\mathbb{A}$ effectuées à la première étape. □

Il est aussi possible de calculer l'évaluation en calculant le produit modulaire FG mod P d'abord en s'appuyant sur l'arithmétique des polynômes denses, puis en réalisant l'évaluation. Une telle approche nécessite $\mathsf{M}(d) = \mathcal{O}(d \log d \log\log d)$ opérations dans $\mathbb{A}$ pour le produit de polynômes $F \times G$ et la division par P et $\mathcal{O}(d)$ opérations dans $\mathbb{A}_{\text{ext}}$ pour l'évaluation. Ainsi si P vérifie $\#P < \log d \log\log d$, la technique présentée ici pour l'évaluation est plus efficace.

2.3.3. Évaluation d'un produit modulaire creux

Dans cette section, les algorithmes précédents sont adaptés pour s'appliquer à des polynômes F et G donnés en représentation creuse. Comme dans la section précédente, nous commençons par décrire notre méthode et les relations de récurrence sur lesquelles elle s'appuie. Les résultats obtenus dépendent de la différence entre les deux plus grands exposants de P. Dans la définition 7 (page 20), nous avons défini le *paramètre d'écart* γ pour mesurer cette différence. Il vérifie $1-\gamma = \frac{1}{d}\deg(P - x^d)$ pour un polynôme P unitaire de degré d. En particulier, le deuxième plus grand exposant de P vaut $(1-\gamma)d$. Le lemme 1.22 (page 21) montre qu'il y a une relation entre le paramètre d'écart et le nombre de monômes non nuls de (FG) mod P. Ce nombre peut être bien plus grand que celui des monômes non nuls de F, G et P. Cependant cette croissance est en quelque sorte contrôlée par γ, ce qui explique que les prochains résultats dépendent de γ.

Pour adapter la méthode de la section 2.3.2, il faut d'abord adapter les relations de récurrence qui la justifient au cas où tous les polynômes sont creux. Puisque G est creux, l'équation (2.3) devient

$$(FG \bmod P)(\alpha) = \sum_{i \in \text{support}(G)} g_i F^{[i]}(\alpha), \tag{2.6}$$

en conservant les notations de la section précédente. La relation de récurrence $F^{[i+1]} = xF^{[i]} - f^{[i]}_{d-1}P$ est toujours valide est donc aussi l'égalité $F^{[i+1]}(\alpha) = \alpha F^{[i]}(\alpha) - f^{[i]}_{d-1}P(\alpha)$.

L'objectif est à nouveau de calculer efficacement les $F^{[i]}(\alpha)$ mais cette fois seulement pour les $i \in \text{support}(G)$ et non pour tous les indices. Lorsque γ n'est pas trop proche de zéro, il n'y a en fait que peu d'indices i tels que $f_{d-1}^{[i]} \neq 0$ et pour lesquels une addition est nécessaire. Le nombre exact dépend de $\#F, \#P$ et γ. Soit $I = \{i_1, \ldots, i_t\}$ l'ensemble de ces indices. Dans le lemme 2.32 nous apporterons la preuve que cet ensemble est de taille au plus $\mathcal{O}(\#F\#P^{\lceil 1/\gamma - 1 \rceil})$. Avant son énoncé, nous continuons à décrire les relations qui permettent de mettre au point un algorithme d'évaluation efficace pour un produit modulaire creux.

Une remarque importante est que pour tout $0 \leq j < d-1$ (en particulier pour $j \in \text{support}(G)$), si i est le plus grand indice de I qui n'excède pas j, alors l'équation (2.5) implique

$$F^{[j]}(\alpha) = \alpha^{j-i} F^{[i]}(\alpha). \tag{2.7}$$

Par conséquent, la relation de récurrence donnée en (2.4) devient

$$\begin{cases} F^{[i_{k+1}]}(\alpha) = \alpha^{i_{k+1}-i_k-1}(\alpha F^{[i_k]}(\alpha) - f_{d-1}^{[i_k]} P(\alpha)) & \text{pour } k \geq 0 \\ F^{[i_1]}(\alpha) = \alpha^{i_1} F^{[0]}(\alpha) \end{cases} \tag{2.8}$$

Pour pouvoir utiliser efficacement les deux équations (2.7) et (2.8) afin d'obtenir l'évaluation, il faut donc une version creuse de l'algorithme 3 CoefficientsDominants. Cet algorithme doit calculer une représentation creuse du vecteur $V = [f_{d-1}^{[0]}, \ldots, f_{d-1}^{[d-2]}]$, c'est à dire le vecteur creux $\{(i, f_{d-1}^{[i]}) : f_{d-1}^{[i]} \neq 0\}$.

L'idée de l'algorithme 5 CoefficientsDominantsCreux est d'imiter l'algorithme 3 CoefficientsDominants mais en manipulant des objets creux. Pour simplifier la présentation, V sera juste considéré comme un vecteur creux sans structure particulière, ce qui est suffisant pour prouver le résultat sur la taille de I. La complexité liée à la manipulation de V peut être minimisée en choisissant une structure particulière comme nous en discuterons dans le lemme 2.33.

Les valeurs initialement non nulles de V sont les coefficients non nuls de $F = F^{[0]}$, avec $V[i] = f_{d-1-i}$ pour $d-i-1 \in \text{support}(F)$. Considérons maintenant la boucle externe dans l'algorithme 3 CoefficientsDominants. L'itération i n'effectue aucune opération si $f_{d-1}^{[i]} = 0$ puisque l'équation (2.5) se simplifie en $f_k^{[i+1]} = f_{k-1}^{[i]}$ dans ce cas. Ainsi, la boucle ne doit considérer que les indices pour lesquels $f_{d-1}^{[i]} \neq 0$. Pour de tels indices, il faut exécuter les mêmes mises à jour de valeurs que dans l'étape 4 de l'algorithme 3 CoefficientsDominants. Pour $k \in \text{support}(P)$, $i < k < d$, il faut donc réaliser l'opération $V[i+d-k] \leftarrow V[i+d-k] - p_k f_{d-1}^{[i]}$ qui soit crée une nouvelle valeur dans V soit met à jour une valeur existante.

Il reste à s'assurer qu'il est possible d'effectuer la boucle seulement sur les indices i tels que $f_{d-1}^{[i]} \neq 0$. Supposons que l'itération i a été exécutée puisque $f_{d-1}^{[i]} \neq 0$. La preuve du lemme 2.29 montre que $V[i+1]$ contient $f_{d-1}^{[i+1]}$. Par conséquent, dans un cadre creux, l'itération $i+1$ doit être effectuée si et seulement si $V[i+1] \neq 0$. De manière générale, l'indice j de l'itération de la boucle à effectuer après l'itération i est celui de la plus petite

coordonnée non nulle de V après $V[i]$. L'algorithme 5 CoefficientsDominantsCreux qui suit, utilise une telle approche pour calculer tous les coefficients $f_{d-1}^{[i]}$ non nuls ainsi que leurs indices i.

Algorithme 5 CoefficientsDominantsCreux

Entrée : P, $F \in \mathbb{A}[x]$ avec $\deg(F) < \deg(P) = d$, et P unitaire

Sortie : La liste $\{(i, f_{d-1}^{[i]}) : 0 \leq i < d-1, f_{d-1}^{[i]} \neq 0\}$, classée par valeurs croissantes de i.

1: $L \leftarrow$ une liste vide
2: $V \leftarrow \{(i, f_{d-1-i}) : d-1-i \in \text{support}(F)\}$ (vecteur creux)
3: **Tant que** V est non vide **faire**
4: Extraire (i, v) de V avec i l'indice minimum
5: **Si** $v \neq 0$ **alors**
6: Ajouter (i, v) à la liste L
7: **Pour** $k \in \text{support}(P)$ tels que $i < k < d$ **faire**
8: $V[i+d-k] \leftarrow V[i+d-k] - p_k v$
9: **Renvoyer** L

Lemme 2.32. *L'algorithme 5* CoefficientsDominantsCreux *calcule bien la liste des* $\{(i, f_{d-1}^{[i]}) : 0 \leq i < d-1, f_{d-1}^{[i]} \neq 0\}$. *Si le polynôme P a pour paramètre d'écart γ, la liste contient au plus $\#F\#P^{\lceil 1/\gamma - 1 \rceil}$ paires et l'algorithme effectue $\mathcal{O}(\#F\#P^{\lceil 1/\gamma-1 \rceil})$ opérations dans $\mathbb{A}$ et additions d'entiers de $\mathcal{O}(\log d)$ bits.*

Démonstration. Comme expliqué au préalable, l'algorithme 5 CoefficientsDominants-Creux est une adaptation de l'algorithme 3 CoefficientsDominants au cas creux qui ne calcule que les $f_{d-1}^{[i]}$ non nuls en utilisant les équations (2.7) et (2.8) à la place de l'équation (2.4). Nommons "itération i", l'itération de la boucle *tant que* qui extrait de V une paire (i, v). La validité de l'algorithme est prouvée par récurrence en montrant qu'à la fin de l'itération i :

$$V = \{(j, f_{d-j+i}^{[i+1]}) : j > i, f_{d-j+i}^{[i+1]} \neq 0\} \text{ et } L = \{(j, f_{d-1}^{[j]}) : j \leq i, f_{d-1}^{[j]} \neq 0\}.$$

Avant la boucle tant que ("itération -1") cette propriété est vraie puisque L est vide et V contient exactement les paires $(j, f_{d-j-1}^{[0]})$ telles que $f_{d-j-1}^{[0]} \neq 0$.

Supposons que la propriété est vraie après l'itération ℓ, et considérons (i, v) la paire extraite à l'itération suivante. Le premier point à prouver est que $f_{d-1}^{[i]} = v$ et $f_{d-1}^{[j]} = 0$ pour $\ell < j < i$. Par minimalité de i et hypothèse de récurrence, $f_{d-j+\ell}^{[\ell+1]} = 0$ pour $\ell < j < i$ à la fin de l'itération ℓ. En particulier, $f_{d-1}^{[\ell+1]} = 0$. D'après l'équation (2.5), $f_{d-j+\ell+1}^{[\ell+2]} = f_{d-j+\ell}^{[\ell+1]} - f_{d-1}^{[\ell+1]} p_{d-j+\ell+1} = 0$. Cette égalité peut facilement s'étendre par récurrence à tous les $f_{d-1}^{[j]}$ pour $\ell < j < i$ qui sont donc tous nuls et ne seront pas stockés dans L à raison. Le fait que tous ces termes soient nuls permet en outre de déduire de l'équation (2.5) que $f_{d-1}^{[i]} = f_{d-2}^{[i-1]} = \cdots = f_{d-i+\ell}^{[\ell+1]}$. Or, par hypothèse de récurrence, à la fin de l'itération ℓ, V contient $(j, f_{d-j+i}^{[\ell+1]})$ si $f_{d-j+i}^{[\ell+1]} \neq 0$. Ainsi, si $f_{d-1}^{[i]}$ n'est pas nul alors

$f_{d-i+\ell}^{[\ell+1]}$ ne l'est pas non plus et V contient la paire $(i, f_{d-i+\ell}^{[\ell+1]}) = (i, f_{d-1}^{[i]})$. Autrement dit la valeur v extraite de V est bel et bien égale à $f_{d-1}^{[i]}$. Après l'itération i, la propriété concernant L est donc vérifiée.

Avec le même argument, il est possible de montrer que $f_{d-1-j+i}^{[i]} = f_{d-j+\ell}^{[\ell+1]}$ pour $\ell < j < i$. Ainsi, juste avant l'itération i, V contient les paires $(j, f_{d-1-j+i}^{[i]})$ telles que $f_{d-1-j+i}^{[i]} \neq 0$. Après l'itération i, ces paires sont remplacées par $(j, f_{d-1-j+i}^{[i]} - p_{d-j+i} f_{d-1}^{[i]})$, qui est exactement $(j, f_{d-j+i}^{[i+1]})$ d'après l'équation (2.5). Et si $f_{d-1-j+i}^{[i]} = 0$ mais $p_{d-j+i} \neq 0$, une nouvelle paire $(j, -p_{d-j+i} f_{d-1}^{[i]}) = (j, f_{d-j+i}^{[i]})$ est ajoutée à V. Par conséquent, la propriété concernant V est aussi vérifiée et à la fin de l'algorithme c'est bien la liste $\{(i, f_{d-1}^{[i]}) : 0 \leq i < d-1, f_{d-1}^{[i]} \neq 0\}$ qui a été calculée.

La seconde partie de la preuve concerne la complexité de l'algorithme. Puisque la boucle tant que s'arrête dès que V est vide, le nombre d'opérations dans $\mathbb{A}$ est au plus deux fois le nombre d'éléments insérés dans V durant l'algorithme. Ce nombre correspond aussi au nombre d'additions d'entiers effectuées pour calculer les indices des paires, ces entiers étant bornés par d. Pour dénombrer les paires, nous allons partitionner leur ensemble en plusieurs *générations*. Initialement, V contient $\#F$ paires qui forment la génération 0. De nouvelles paires peuvent être ajoutées à V quand une paire (i, v) est extraite de V. Si (i, v) est une paire de la génération t, les nouvelles paires ajoutées à l'itération i appartiennent à la génération $t+1$. À chaque itération c'est au plus $\#P-1$ nouvelles paires qui peuvent être ajoutées. Ainsi, il y a au plus $\#F(\#P-1)$ paires dans la génération 1, $\#F(\#P-1)^2$ paires dans la génération 2, et plus généralement $\#F(\#P-1)^t$ paires dans la génération t. Il reste donc à déterminer le nombre maximal de générations. À la génération 0, les paires ont un indice i compris entre 0 et $d-1$. À la génération 1 en revanche, les nouvelles paires ont pour indice $(i + d - k)$ pour un $k \in \text{support}(P)$, $k < d$. C'est là qu'intervient le paramètre d'écart γ. En effet le plus grand exposant de P différent de d est au plus $(1-\gamma)d$ par définition de γ. Par conséquent à la génération 1, les indices des paires valent au moins $i + d - (1-\gamma)d \geq \gamma d$. À la génération 2, les paires ont donc des indices valant au moins $2\gamma d$ et à la génération t, ils valent au moins $t\gamma d$. Par ailleurs, les indices valent au plus $p-1$, il ne peut donc pas y avoir de génération t si $t\gamma d \geq d$. Autrement dit, l'indice de la dernière génération possible est $t = \lceil 1/\gamma - 1 \rceil$. Ceci donne donc un nombre total de paires potentiellement ajoutées à V d'au plus

$$\sum_{t=0}^{\lceil 1/\gamma - 1 \rceil} \#F(\#P-1)^t = \#F \, \frac{(\#P-1)^{\lceil 1/\gamma \rceil} - 1}{\#P - 2}$$

si $\#P > 2$, et d'au plus $\lceil 1/\gamma \rceil \#F$ si $\#P = 2$. Pour simplifier les notations, les deux nombres sont bornés par $\#F\#P^{\lceil 1/\gamma - 1 \rceil}$ dans la suite. Bien évidemment, ce nombre est aussi une borne sur le nombre de paires qui sont extraites de V durant l'algorithme.

Cette valeur donne deux informations. D'une part elle borne la taille de la liste L, formée à partir de V, qui est renvoyée par l'algorithme et d'autre part elle borne aussi le nombre d'exécutions de l'étape 8, c'est à dire le nombre d'opérations par $\mathcal{O}(\#F\#P^{\lceil 1/\gamma - 1 \rceil})$. □

Pour analyser la complexité, nous avons considéré comme négligeable le coût des accès aux éléments de V, ainsi que des modifications, ajouts, suppressions, extractions de minimum sur la structure V. Ceci n'est pas forcément automatique.

Si V est représenté par un tas ordonné par les indices i, chacune de ces opérations nécessite un nombre de comparaisons entre indices logarithmique dans la longueur du tas. Nous avons vu que V peut contenir jusqu'à $\#F\#P^{\lceil 1/\gamma-1\rceil}$ éléments. Ainsi, les opérations sur V effectuent $\mathcal{O}(\log(\#F\#P^{\lceil 1/\gamma-1\rceil}))$ comparaisons entre les indices i qui sont bornés par le degré d. Chacune de ces comparaisons nécessite la lecture des indices, autrement dit $\mathcal{O}(\log d)$ opérations binaires. Ainsi, la gestion de V comme un tas ajoute à chaque opération de l'algorithme 5 CoefficientsDominantsCreux un cout de $\mathcal{O}(\log(\#F\#P^{\lceil 1/\gamma-1\rceil}))$ opérations binaires ce qui domine le coût d'une addition d'entiers de $\mathcal{O}(\log d)$ bits, opération comptabilisée dans le lemme 2.32.

Pour représenter V sans perdre d'efficacité, il est donc nécessaire de passer par une structure plus performante. Pour cela, nous considérons V comme un arbre de van Emde Boas creux ([Cor+09, Chapter 20]). Nous donnons d'abord le coût total des opérations sur V au cours de l'algorithme avant de donner plus d'explications sur les arbres de van Emde Boas creux. Le point essentiel pour le lemme 2.33 est que chaque opération sur V requiert alors $\mathcal{O}(\log d)$ opérations, comme nous le verrons dans le lemme 2.35.

Lemme 2.33. *L'ensemble de toutes les opérations d'*Insertion, Suppression, Minimum *et* Recherche *sur les paires (i, ν) dans l'arbre de van Emde Boas creux V qui sont effectuées au cours de l'algorithme 5* CoefficientsDominantsCreux *ne nécessite que $\mathcal{O}(\#F\#P^{\lceil 1/\gamma-1\rceil}\log d)$ opérations binaires.*

Le coût lié à la structure est donc asymptotiquement équivalent au coût de l'ensemble des opérations sur les entiers donné dans le lemme 2.32.

2.3.3.1. Arbres de van Emde Boas creux

Un arbre de van Emde Boas est une structure de données visant à stocker efficacement un ensemble $\mathcal{E}$ d'entiers tous distincts. Par efficacement, il faut entendre que la taille du stockage est limitée et aussi que le coût des opérations sur cet ensemble est faible. Il existe pour ce faire différentes structures de données, nous présentons ici les arbres de van Emde Boas pour lesquels plus de détails sont donnés dans le livre *Introduction to Algorithms* [Cor+09, Chapter 20].

Si toutes ses valeurs sont strictement inférieures à un entier u, l'ensemble $\mathcal{E}$ peut être représenté par un tableau T de taille u donnant les valeurs de sa fonction indicatrice. C'est-à-dire que si $i \in \mathcal{E}$ alors $T[i] = 1$, sinon $T[i] = 0$. L'espace utilisé pour cette représentation est de seulement u bits, l'accès, l'insertion ou la suppression d'éléments nécessitent seulement de lire la valeur de l'élément. En revanche, la recherche du minimum ou du maximum demande $\mathcal{O}(u)$ opérations.

L'idée dans un arbre de van Emde Boas est de couper ce tableau en $\sqrt{u}$ blocs de taille

$\sqrt{u}$. Un étage supérieur est ajouté : un tableau de taille $\sqrt{u}$ qui stocke l'ensemble des indices des blocs contenant un élément. Pour former complètement la structure d'arbre, chacun des blocs et le tableau des indices sont récursivement stockés de la même manière jusqu'à atteindre des tableaux de taille 2. Pour accélérer, les recherches et parcours dans la structure de données, le minimum et le maximum des valeurs de chacun des tableaux sont aussi stockés de manière indépendante. Les valeurs de u considérées sont des puissances de 2, et pour simplifier la présentation nous nous restreignons au cas $u = 2^{2^k}$ pour qu'à chaque étape de la construction récursive, $\sqrt{u}$ soit aussi une puissance de 2. Si $u = 2^{2n+1}$ était une puissance impaire de 2, il faudrait plutôt séparer le tableau en 2^{n+1} blocs de taille 2^n ce qui assure aussi de garder comme paramètres des puissances de 2.

Définition 10. Soit $u = 2^{2^k}$, un *arbre de van Emde Boas* sur un univers de taille u, $vEB(u)$ est défini de la manière suivante :

- si $u = 2$, $vEB(u) = \{2, min, max\}$,
- sinon, $vEB(u) = \{u, min, max, Sommaire, Tab\}$ avec $Sommaire$ un arbre de van Emde Boas d'univers $\sqrt{u}$ et Tab un tableau de $\sqrt{u}$ arbres de van Emde Boas d'univers $\sqrt{u}$.

Un nombre $a < u$ est stocké dans $vEB(u)$ si :

- $a = min$,
- ou $a = max$,
- ou $a \bmod \sqrt{u}$ est stocké dans $Tab[a \operatorname{div} \sqrt{u}]$. Dans ce cas, $a \operatorname{div} \sqrt{u}$ est aussi stocké dans $Sommaire$.

Fait 2.34. *Un arbre de van Emde Boas sur un univers de taille u a une taille binaire en $\mathcal{O}(u)$.*

Démonstration. Si $u = 2$, alors min et max demandent chacun 1 bit de stockage auxquels il faut ajouter le stockage de u sur 2 bits. La taille binaire est donc bien en $\mathcal{O}(2)$. Sinon, le stockage de u, min et max nécessitent $\mathcal{O}(\log u)$ bits auxquels il faut ajouter $\sqrt{u} + 1$ arbres de van Emde Boas de taille $\sqrt{u}$ pour Tab et $Sommaire$. Par récurrence, cela donne bien une taille totale de $\mathcal{O}(u)$ bits. □

En plus de la taille, il convient de s'intéresser aux différentes opérations pouvant s'appliquer à un arbre de van Emde Boas. Les opérations de minimum et maximum ont un coût constant par définition. La recherche d'un élément se fait de manière récursive en suivant la définition. Il en est de même pour l'insertion d'un élément a qui passe par l'insertion de $a \bmod \sqrt{u}$ dans $Tab[a \operatorname{div} \sqrt{u}]$ et de $a \operatorname{div} \sqrt{u}$ dans $Sommaire$ en mettant à jour les minimums et maximums à chaque fois que cela est nécessaire.

La recherche du successeur de a est un peu plus technique. Il y a quatre cas à considérer :

1. $a = max$: il n'y a pas de successeur de a dans l'arbre de van Emde Boas.
2. $u = 2$: si $a = 0$ et $max = 1$ alors 1 est le successeur de a .

3. $u > 2$ et $a \bmod \sqrt{u}$ est le max de $Tab[a \operatorname{div} \sqrt{u}]$: soit v le successeur de $a \operatorname{div} \sqrt{u}$ dans $Sommaire$, le successeur de a est v plus le min de $Tab[v]$
4. $u > 2$ et $a \bmod \sqrt{u}$ n'est pas le max de $Tab[a \operatorname{div} \sqrt{u}]$: soit s le successeur de a dans $Tab[a \operatorname{div} \sqrt{u}]$ le successeur de a dans l'arbre de van Emde Boas est $a \operatorname{div} \sqrt{u} + s$.

Pour le prédécesseur de a, il faut considérer les mêmes cas en remplaçant successeur par prédécesseur et en inversant min et max et 0 et 1.

Ces recherches se font de manière récursive. Dans l'esprit, cette récurrence part d'un arbre d'univers de taille u, jusqu'à un arbre d'univers 2 en passant par un seul arbre de chacun des univers intermédiaires. En effet, soit il y a toujours un successeur en regardant dans les Tab successifs qui ont une taille de plus en plus petite. Soit il faut chercher un successeur des $Sommaire$ et à partir de ce moment la récurrence se poursuit uniquement dans l'arbre de van Emde Boas $Sommaire$, avec la même décroissance pour les tailles considérées.

La suppression d'un élément se fait en suivant la définition, si nécessaire min et max sont mis à jour en recherchant le prédécesseur ou successeur de l'élément considéré. Si le dernier élément d'un des arbres de van Emde Boas de Tab est supprimé, il faut alors supprimer son indice dans $Sommaire$.

Toutes ces opérations permettent de calculer avec un arbre de van Emde Boas d'univers $u = 2^{2^k}$. Mis à part le calcul du minimum ou du maximum en temps constant, elles nécessitent essentiellement de passer par un seul arbre de van Emde Boas intermédiaire pour chacun de 2^{2^i} avec $0 \leq i \leq k$. Pour évaluer le coût de ces opérations, nous considérons à chaque fois le coût d'obtention et de lecture ou écriture des nouveaux indices et éléments calculés pour la poursuite de la récurrence.

Lemme 2.35. *L'insertion, la suppression, la recherche d'un élément, du minimum ou du maximum, le calcul d'un prédécesseur ou d'un successeur sur un arbre de van Emde Boas d'univers u ne nécessitent que $\mathcal{O}(\log u)$ opérations binaires.*

Démonstration. Le minimum et le maximum s'obtiennent en temps constant. Pour les autres opérations, en partant d'un élément $a < u$, elles demandent essentiellement de calculer $a \bmod \sqrt{u}$ et $a \operatorname{div} \sqrt{u}$ et de poursuivre soit avec $a \operatorname{div} \sqrt{u}$ dans $Sommaire$ soit avec $a \bmod \sqrt{u}$ dans $Tab[a \operatorname{div} \sqrt{u}$. Autrement dit, la suite du calcul s'effectue avec un nombre inférieur à $\sqrt{u}$ dans un arbre de van Emde Boas de taille $\sqrt{u}$. Et ce jusqu'à atteindre la taille 2. Pour $u = 2^{2^k}$, les divisions ne coûtent que la lecture du résultat, c'est à dire $\log u = 2^k$ pour la première étape puis 2^i avec $i < k$ pour les étapes suivantes jusqu'à avoir $i = 0$. Le coût total est donc de $\mathcal{O}\left(\sum_{i=0}^{k} 2^i\right) = \mathcal{O}(2^k) = \mathcal{O}(\log u)$. □

La structure présentée jusqu'ici est dense et a donc une taille trop importante. En remplaçant Tab par un dictionnaire ne contenant que les arbres de van Emde Boas non vide, il est cependant aisé de définir un arbre de van Emde Boas creux. Pour un tel arbre, les opérations ont le même coût que dans le cas général et la taille est donnée par

le nombre d'éléments qui sont stockés. Il n'est en effet pas nécessaire de construire toutes la structure récursive dès lors qu'un arbre ne contient pas d'autres éléments que son *min* et son *max*.

Définition 11. Un *arbre de van Emde Boas* creux est un arbre de van Emde Boas où le tableau *Tab* est remplacé par *Dico* un dictionnaire ne contenant que des arbres de van Emde Boas non vides ayant pour clé l'indice qu'ils auraient eu dans *Tab*. De plus, récursivement chaque arbre de van Emde Boas de *Dico* et aussi *Sommaire* sont aussi des arbres de van Emde Boas creux.

Corollaire 2.36. *Soit un arbre de van Emde Boas creux d'univers u. S'il y a T éléments dans l'arbre, alors la taille de la structure est de $\mathcal{O}(T \log u)$ bits. Par ailleurs toutes les opérations sur l'arbre nécessitent $\mathcal{O}(\log u)$ opérations binaires.*

Enfin, le dernier point à remarquer est qu'un arbre de van Emde Boas peut contenir non seulement des entiers mais aussi des éléments plus complexes ayant un entier pour clé. En particulier, ils peuvent contenir des paires (ind, val) pour représenter un tableau contenant la valeur *val* à l'indice *ind*. Toutes les opérations de comparaisons et divisions par $\sqrt{u}$ sont réalisées uniquement sur l'indice. On peut aussi considérer que l'arbre de van Emde Boas stocke seulement les clés d'un dictionnaire qui permet lui d'accéder aux valeurs. En prenant l'un ou l'autre des modèles, il est bien possible de représenter un vecteur creux grâce à un arbre de van Emde Boas.

Nous avons maintenant décrit une structure de données permettant d'effectuer efficacement l'algorithme 5 CoefficientsDominantsCreux. Revenons maintenant à l'évaluation d'un produit modulaire de polynômes creux $FG \bmod P$. L'algorithme 5 CoefficientsDominantsCreux permet de calculer les coefficients intervenant dans l'équation (2.8) (page 50). Cette équation définit les relations entre les évaluations des polynômes $F^{[i]} = (x^i F) \bmod P$ faisant intervenir des coefficients non nuls de F. Cette équation décrit avec l'équation (2.7) toutes les récurrences entre les polynômes $F^{[i]}$ nécessaires pour le calcul de $(FG \bmod P)(\alpha)$ à partir de l'équation 2.6 (49). Plus précisément, les $F^{[i]}(\alpha)$ pour les i tels que $f_{d-1}^{[i]} \neq 0$ sont calculés grâce à l'équation (2.8). À partir de ces valeurs et de l'équation (2.7), il est possible d'obtenir les $F^{[j]}(\alpha)$ pour $j \in \text{support}(G)$. Ils forment, après multiplication par g_j, les termes de la somme décrivant $(FG \bmod P)(\alpha)$ dans l'équation (2.6). Ainsi, il est alors possible d'en déduire l'évaluation de $FG \bmod P$.

Dans l'algorithme 6 ÉvaluationModulaireCreuse, les calculs ne sont pas effectués les uns après les autres mais de manière entremêlée. L'idée est d'effectuer une boucle sur les j tels que soit $f_{d-1}^{[j]} \neq 0$ soit $j \in \text{support}(G)$. Dans le premier cas, il faut mettre à jour la valeur $F^{[j]}(\alpha)$ par l'équation (2.8). Dans le second cas, un nouveau terme est ajoutée à la somme de l'équation (2.6) en utilisant l'équation (2.7) pour construire progressivement l'évaluation.

Théorème 2.37. *Étant donnés P, F et $G \in \mathbb{A}[x]$, avec $\deg(F), \deg(G) < \deg(P) = d$, P unitaire et $\alpha \in \mathbb{A}_{ext}$, l'algorithme 6* ÉvaluationModulaireCreuse *calcule bien*

Algorithme 6 ÉvaluationModulaireCreuse

Entrée : P, F et $G \in \mathbb{A}[x]$, avec $\deg(F), \deg(G) < \deg(P) = d$, P unitaire et $\alpha \in \mathbb{A}_{\text{ext}}$.
Sortie : $(FG \bmod P)(\alpha)$

1: $V \leftarrow \{(i, f_{d-1}^{[i]}) : 1 \leq i < n - 1, f_{d-1}^{[i]} \neq 0\}$, par CoefficientsDominantsCreux(P, F)
2: **Si** $f_{d-1}^{[0]} = 0$ **alors** ajouter $(0, 0)$ dans V
3: $P_\alpha \leftarrow P(\alpha)$
4: $F_\alpha \leftarrow F(\alpha)$
5: $\beta \leftarrow F_\alpha g_0$ ▷ $\beta \leftarrow 0$ si $0 \notin \text{support}(G)$
6: $i \leftarrow 0$
7: **Pour** $j \in \text{support}(V) \cup \text{support}(G) \setminus \{0\}$, en ordre croissant **faire**
8: **Si** $j \in \text{support}(V)$ **alors**
9: $F_\alpha \leftarrow \alpha^{j-i-1}(\alpha F_\alpha - V[i]P_\alpha)$ ▷ équation (2.8)
10: $i \leftarrow j$
11: **Si** $j \in \text{support}(G)$ **alors**
12: $\beta \leftarrow \beta + \alpha^{j-i} F_\alpha g_j$ ▷ équations (2.6) et (2.7)
13: **Renvoyer** β

$(FG \bmod P)(\alpha)$. *Il effectue* $\mathcal{O}(\#F\#P^{\lceil \frac{1}{\gamma}-1 \rceil})$ *opérations dans* $\mathbb{A}$, *et* $\mathcal{O}((\#F\#P^{\lceil \frac{1}{\gamma}-1 \rceil} + \#G)\log d)$ *opérations dans* $\mathbb{A}_{ext}$. *Le coût des opérations dans* $\mathbb{Z}$ *est négligeable.*

Démonstration. La validité de l'algorithme se déduit du fait qu'à la fin de l'itération j, $F_\alpha = F^{[j]}(\alpha)$ si $j \in \text{support}(F)$ et $\beta = \sum_i g_i F^{[i]}(\alpha)$ avec une somme seulement sur les indices i tels que $i \in \text{support}(G) \cap \{0, \ldots, j\}$. Cette propriété peut se montrer par récurrence. Elle est satisfaite après l'itération 0 (avant l'entrée dans la boucle) puisque $F_\alpha = F^{[0]}(\alpha)$ et $\beta = F_\alpha g_0 = g_0 F^{[0]}(\alpha)$. Supposons maintenant qu'elle soit aussi vraie juste avant de commencer l'itération j. L'indice i représente le plus grand élément du support de F strictement inférieur à j. Ainsi, si $j \in \text{support}(F)$, l'équation (2.8) garantit que F_α a la bonne valeur après l'itération j puisque $V[i] = f_{d-1}^{[i]}$. Par ailleurs, les équations (2.6) et (2.7) montrent que β a aussi la bonne valeur si $j \in \text{support}(G)$. La propriété est donc vérifiée après l'itération j, d'où le calcul par l'algorithme de $(FG \bmod P)(\alpha)$.

Les évaluations $P(\alpha)$ et $F(\alpha)$ nécessitent $\mathcal{O}(\#P \log d)$ et $\mathcal{O}(\#F \log d)$ opérations dans $\mathbb{A}_{\text{ext}}$ respectivement. Les étapes 9 et 12 effectuent chacune $\mathcal{O}(\log d)$ opérations dans $\mathbb{A}_{\text{ext}}$ pour le calcul des puissances de α et $\mathcal{O}(1)$ additions avec des entiers de taille $\mathcal{O}(\log d)$ pour calculer l'exposant approprié. Ces étapes sont répétées pour tous les éléments de $V \cup \text{support}(G)$ c'est à dire $\mathcal{O}(\#F\#P^{\lceil 1/\gamma - 1 \rceil} + \#G)$ fois en appliquant le lemme 2.32 pour le nombre d'éléments de V. Ceci donne donc un total de $\mathcal{O}((\#F\#P^{\lceil \frac{1}{\gamma}-1 \rceil} + \#G)\log d)$ opérations dans $\mathbb{A}_{\text{ext}}$ plus $\mathcal{O}((\#F\#P^{\lceil \frac{1}{\gamma}-1 \rceil} + \#G)\log d)$ opérations binaires. Comme il est naturel de considérer qu'une opération dans $\mathbb{A}_{\text{ext}}$ nécessite au moins une opération binaire, le coût des opérations sur les entiers est négligeable. La complexité de l'étape 1

est donnée par le lemme 2.32. Il est toujours possible d'utiliser un arbre de van Emde Boas creux pour les opérations sur l'union des supports de V et de G à l'étape 7 avec un total de $\mathcal{O}((\#F\#P^{\lceil \frac{1}{\gamma}-1 \rceil} + \#G)\log d)$ opérations binaires d'après le lemme 2.35 pour le coût de la structure de données. Il s'agit du même coût que celui des opérations dans $\mathbb{Z}$. □

Naturellement, comme le produit de polynôme est commutatif sur des anneaux commutatifs, les rôles de F et de G peuvent être intervertis dans l'algorithme 6. En particulier si $\#G < \#F$, cela permet de diminuer la complexité dans le théorème 2.37. Autrement dit, il est possible dans cette complexité de remplacer $\#F$ par $\min(\#F, \#G)$ et $\#G$ par $\max(\#F, \#G)$. La même remarque s'applique aux résultats qui suivent.

Remarque 2.38. Comme dans le corollaire 2.28, il est possible de diminuer le nombre d'opérations dans $\mathbb{A}_{\text{ext}}$ en utilisant l'exponentiation simultanée pour calculer toutes les puissances de α requises. Ainsi, le nombre d'opérations dans $\mathbb{A}_{\text{ext}}$ est ramené à $\log d + \mathcal{O}((\#F\#P^{\lceil 1/\gamma-1 \rceil} + \#G)\log d/\log\log d)$.

Si le paramètre d'écart γ est proche de $\frac{1}{d}$, le polynôme $FG \bmod P$ a en général $\Omega(d)$ monômes non nuls et ce même si FG lui même n'en a que très peu (voir exemple 1.21). Dans ce cas là, il est plus approprié d'utiliser l'algorithme 4 ÉvaluationModulaire pour les polynômes denses. Au contraire, si γ est proche de 1, le nombre de monômes non nuls de $FG \bmod P$ ne croît pas spécialement, en particulier si $\gamma \geq \frac{1}{2}$.

Remarque 2.39. Si $\gamma \geq \frac{1}{2}$, l'évaluation du produit modulaire nécessite $\mathcal{O}((\#F\#P + \#G)\log d)$ opérations dans $\mathbb{A}_{\text{ext}}$.

Remarque 2.40. Le facteur $\#F\#P^{\lceil \frac{1}{\gamma}-1 \rceil} + \#G$ dans la complexité peut être bien plus élevé que le véritable nombre de monômes de $FG \bmod P$, en particulier si P divise FG. Cependant, il est plus petit que la borne générale $\#F\#G(\#P-1)^{\lceil \frac{1}{\gamma} \rceil}$ donnée par le lemme 1.22. Ainsi en général, il est plus efficace de calculer l'évaluation par la méthode présentée ici qu'en calculant $FG \bmod P$ pour l'évaluer directement.

Dans ce chapitre nous avons présenté des techniques pour réduire ou contrôler la taille des polynômes. Il s'agit principalement de considérer ces polynômes modulo un autre polynôme P avec le plus souvent $P = x^p - 1$. Nous avons également vu comment manipuler des polynômes réduits, en particulier comment obtenir la réduction par l'évaluation pour des polynômes à coefficients entiers et comment évaluer efficacement des produits de polynômes en tenant compte de la réducion modulaire.

Ces points forment la base des résultats qui sont présentés dans la suite de cette thèse.

3. Interpolation creuse

L'interpolation d'un polynôme consiste à reconstruire le polynôme dans la représentation désirée à partir de connaissances indirectes à son sujet. Le type de ces connaissances peut fortement varié et impacte les algorithmes utilisés pour résoudre ce problème ainsi que leur efficacité. Le problème d'interpolation le plus classique est la recherche de la représentation d'un polynôme de degré au plus D à partir de $D+1$ évaluations quelconques du polynôme. Le nombre d'évaluations permet de s'assurer de l'unicité de la solution.

Problème (Interpolation)**.** À partir d'une liste de paires $(x_0, y_0), (x_1, y_1), \ldots, (x_D, y_D)$ avec des x_i tous distincts, l'*interpolation* consiste à trouver le polynôme F de degré au plus D vérifiant $F(x_i) = y_i$ pour tout $0 \leq i \leq D$.

Si la liste de points est quelconque, la méthode de Lagrange est la plus efficace pour calculer le polynôme F. Cependant, l'interpolation à base de transformée de Fourier utilisant l'évaluation du polynôme sur des racines de l'unité est plus efficace d'un facteur $\log D$. En ajoutant des contraintes sur les données du problème, il est possible d'obtenir des solutions plus efficaces.

Dans le cas de l'interpolation d'un polynôme creux F avec $\deg(F) \leq D$ et $\#F \leq T$, une liste de D paires de points est exponentiellement plus grande que la taille de F. Pour que la taille de l'entrée corresponde à la taille de la sortie, il faudrait moins de paires (point, évaluation). Si n, le nombre de paires est strictement inférieur à T, il est possible d'utiliser l'interpolation de Lagrange pour obtenir un polynôme respectant les bornes données. Mais contrairement au cas dense, l'unicité n'est pas assurée. Si n est plus grand que T, la question est plutôt de savoir si il existe une solution voire simplement si cela est décidable. Cette question a été étudiée par Borodin et Tiwari [BT90], pour des cas particuliers sans pour autant apporter de réponses dans le cas général.

Si les points d'évaluation ne sont pas quelconques, il est possible d'en déduire plus d'informations, tout comme il est possible d'être plus efficace dans le cas dense. Des informations supplémentaires comme des évaluations dans une extension ou des évaluations des dérivées peuvent aider à trouver le polynôme F plus facilement. Le problème de l'interpolation creuse se pose donc différemment : les données prises en entrées ne seront pas une liste d'évaluations mais une *méthode* pour obtenir des évaluations et parfois plus d'informations sur le polynôme creux recherché.

Problème (Interpolation creuse)**.** À partir d'une méthode pour évaluer un polynôme F et de bornes D et T sur le degré et le nombre de monômes non nuls de F, l'*interpolation creuse* consiste à trouver la représentation creuse du polynôme F.

Il y a principalement deux modèles pour décrire cette *méthode d'évaluation* : la boîte noire et le circuit arithmétique. La boîte noire correspond à la fonction d'évaluation du polynôme et permet donc de l'évaluer sur n'importe quel point. En particulier les points d'évaluations peuvent être choisis au cours de l'algorithme. Il est possible d'étendre la boîte noire en autorisant des points d'évaluation hors de l'anneau de définition du polynôme. Le circuit arithmétique correspond à une suite d'opérations élémentaires, additions et multiplications, dont le résultat est le polynôme recherché. Des bornes sont aussi fournies sur les paramètres du polynôme comme son degré ou le nombre de ses monômes. Les algorithmes d'interpolation les utilisent pour déterminer quelles évaluations seront demandées à la boîte noire ou au circuit arithmétique. Elles permettent, en particulier pour l'interpolation de boîte noire, de s'assurer de la validité du résultat notamment de son unicité. L'évaluation de la complexité des algorithmes d'interpolation creuse repose sur ces bornes.

Ces deux modèles, boîte noire et circuit arithmétique, permettent le développement d'applications de l'interpolation creuse à l'arithmétique. Que ce soit dans le cadre d'opérations de base, multiplication et division que nous verrons au chapitre 5, ou pour des opérations plus complexes, calcul de pgcd multivarié ou de remontée de Hensel [Zip79 ; Zip81 ; HM16 ; MT16]. Ces opérations peuvent souvent être réalisées pour des polynômes connus eux-mêmes implicitement grâce à une boîte noire [KT90 ; HL21] ou un circuit arithmétique [Kal88].

L'interpolation creuse fournit aussi un outil pour corriger un faible nombre d'erreurs après un calcul potentiellement coûteux sans avoir à le refaire intégralement. Une telle démarche a ainsi été proposé pour des opérations matricielles [Roc18a ; Dum+19]. L'idée est de transformer une matrice de taille $n \times n$ en polynôme en la projetant à gauche et à droite par les vecteurs $(1, x, \ldots, x^{n-1})$ et $(1, y, \ldots, y^{n-1})$.

Comer, Kaltofen et Pernet ont montré [CKP12] que la résolution du problème d'interpolation creuse peut, elle-même, être robuste face à des erreurs dans les évaluations. Ceci a conduit à la définition de codes correcteurs basés sur l'interpolation creuse [KP14 ; KY21].

Ce chapitre consacré à l'interpolation de polynômes creux présentera d'abord, en section 3.1, les principaux algorithmes connus pour résoudre ce problème dans différents anneaux. Ensuite, en section 3.2 nous nous concentrerons le cas des polynômes à coefficients entiers et présenterons notre algorithme quasi-linéaire publié à ISSAC en 2022. Seul le cas d'un polynôme à une variable sera traité dans ce chapitre, l'extension de notre algorithme aux polynômes multivariés étant détaillée au chapitre 6.

Ce chapitre se concentre donc sur l'interpolation d'un polynôme creux $F \in \mathbb{A}[x]$ de degré au plus D ayant au plus T monômes non nuls, pour un anneau de coefficients $\mathbb{A}$ qui sera principalement $\mathbb{Z}$ ou un corps fini. La taille de F est d'au plus $T(\log D + B)$ bits, avec B une borne sur la taille des coefficients de F. Les performances des algorithmes sont donc analysées par rapport à cette taille et par rapport à la taille du circuit arithmétique le cas échéant. L'analyse porte non seulement sur le coût des opérations arithmétiques

effectuées pour obtenir le polynôme mais aussi sur le coût de l'obtention d'information par le système considéré (boîte noire ou circuit).

3.1. Historique des algorithmes d'interpolation creuse

Le premier algorithme d'interpolation creuse connu est dû à Prony en 1795 [Pro95] et s'applique à l'interpolation d'un polynôme creux donné par une boîte noire.. Cette date peut sembler très ancienne mais il a fallu attendre près de deux siècles pour que l'interpolation creuse devienne réellement un sujet d'intérêt pour les scientifiques avec en particulier la redécouverte par Ben-Or et Tiwari en 1988 [BT88] de l'algorithme de Prony. De nombreuses améliorations ont depuis été apportées aux techniques d'interpolation. En particulier en 2009 Garg et Schost [GS09] ont proposé le premier algorithme de complexité polynomiale dans le logarithme du degré plutôt que polynomial dans le degré lui-même. Pour cela ils se sont placés dans un modèle différent : le circuit arithmétique plutôt que la boîte noire ou une version étendue de boîte noire.

Les sections 3.1.1 et 3.1.2 présentent les algorithmes liés à ces deux modèles. La section 3.1.3 traite de l'usage du polynôme dérivé, idée introduite par Huang en 2019 [Hua19], pour améliorer les algorithmes d'interpolation.

3.1.1. Pour un polynôme sous forme de boîte noire

Le premier modèle étudié est celui de la boîte noire où le seul moyen d'obtenir des informations sur le polynôme F que l'on cherche à calculer est par l'évaluation en des points. Ces points peuvent être librement choisis par la personne réalisant l'interpolation.

FIGURE 3.1. – Boîte noire

L'algorithme de Prony ou Ben-Or–Tiwari repose justement sur la possibilité de choisir les points d'évaluations et utilise des points en suite géométrique. Ceux-ci donnent en effet à leurs évalués des propriétés intéressantes.

Fait 3.1. *Soit* $F = \sum_{i=0}^{T-1} f_i x^{e_i}$ *un polynôme dans* $\mathbb{A}[x]$ *et* ω *un point de* $\mathbb{A}$. *La suite* $(F(\omega^j))_{j\in\mathbb{N}}$ *est une suite récurrente linéaire de polynôme caractéristique* $\Lambda(z) = \prod_{i=0}^{T-1}(z - \omega^{e_i})$.

Ce premier point, montre comment les exposants du polynôme sont liés aux évaluations sur les puissances d'un seul et même point.

Après avoir fourni les exposants du polynôme F, la suite $(F(\omega^j))_j$ permet aussi de retrouver les coefficients de F.

Fait 3.2. *Pour tout* $F = \sum_{i=0}^{T-1} f_i x^{e_i} \in \mathbb{A}[x]$ *et* $\omega \in \mathbb{A}$ *:*

$$\begin{pmatrix} 1 & \cdots & 1 \\ \omega^{e_0} & \cdots & \omega^{e_{T-1}} \\ \omega^{2e_0} & \cdots & \omega^{2e_{T-1}} \\ \vdots & & \vdots \\ \omega^{(T-1)e_0} & \cdots & \omega^{(T-1)e_{T-1}} \end{pmatrix} \begin{pmatrix} f_0 \\ f_1 \\ f_2 \\ \vdots \\ f_{T-1} \end{pmatrix} = \begin{pmatrix} F(1) \\ F(\omega) \\ F(\omega^2) \\ \vdots \\ F(\omega^{T-1}) \end{pmatrix}.$$

Ainsi les coefficients de F s'obtiennent par la résolution d'un système linéaire de type Vandermonde de taille $T \times T$.

Ces deux points permettent de justifier l'algorithme d'interpolation de Prony, décrit dans l'algorithme 7 Interpolation de Prony.

Algorithme 7 Interpolation de Prony

Entrée : Une boîte noire pour un polynôme creux $F \in \mathbb{A}[x]$, D et T des bornes sur le degré et le nombre de monômes de F.

Sortie : Le polynôme $F = \sum_{i=0}^{T-1} f_i x^{e_i}$ sous forme creuse.

1: Prendre un point $\omega \in \mathbb{A}$ d'ordre supérieur à D
2: Évaluer par la boîte noire $F(1), F(\omega), \cdots F(\omega^{2T-1})$
3: Calculer $\Lambda(z) = \prod_{i=0}^{T-1} (z - \omega^{e_i})$, le générateur de cette séquence
4: Calculer les racines de Λ
5: Calculer leur logarithme en base ω pour obtenir les exposants
6: Calculer les coefficients en résolvant le système du fait 3.2
7: **Renvoyer** les paires (f_i, e_i)

Un premier point à remarquer sur cet algorithme est le faible nombre d'évaluations requises, $2T$, et le fait qu'elles puissent toutes être calculées en une seule fois. Autrement dit, il s'agit d'un algorithme non adaptatif.

Le fait que l'ordre du point soit supérieur au degré de F est essentiel. De manière générale, si o est l'ordre du point ω le polynôme caractéristique de la suite est $\prod_{i=0}^{T-1}(z - \omega^{e_i \bmod o})$. À partir de telles informations le polynôme ne pourrait être reconstruit que modulo $x^o - 1$ et non à son vrai degré.

L'efficacité de l'algorithme dépend des algorithmes utilisés pour résoudre chacun des problèmes algébriques des étapes 3 à 6. Les étapes 3 et 6 en particulier ont des solutions efficaces qui seront utiles par la suite et pour cela formalisées dès maintenant en prenant en compte les contraintes de la section 3.2. Ainsi, pour l'étape 3, nous regardons spécifiquement le cas des corps finis. Et pour l'interpolation des coefficients par la matrice

de Vandermonde, étape 6, nous donnons aussi le coût de l'opération inverse, l'évaluation multipoint.

Tout d'abord le polynôme générateur peut être construit par l'algorithme de Berlekamp-Massey.

Fait 3.3 ([Mas69 ; Sch71 ; Dor87])**.** *Le polynôme générateur d'une suite récurrente linéaire de degré T dans $\mathbb{F}_q$ peut être calculé en $\mathcal{O}(\mathsf{M}(T)\log T) = \tilde{\mathcal{O}}(T)$ opérations dans $\mathbb{F}_q$.*

L'autre point efficace est l'obtention des coefficients en résolvant le système défini dans le fait 3.2. La matrice définissant ce système est la transposée d'une matrice de Vandermonde, ce qui permet de relier la résolution du système à de l'arithmétique de polynômes denses de degré T [KL88 ; BLS03]. Ceci ne peut être réalisé dans n'importe quel anneau $\mathbb{A}$, il faut garantir que la matrice soit inversible, c'est-à-dire que $\omega^{e_i} - \omega^{e_j}$ soit inversible pour $i \neq j$. Cette propriété vraie sur les corps, peut aussi être vérifiée sur des anneaux. Notamment dans le cas où ω est une racine principale de l'unité : ω a un ordre multiplicatif fini et $\omega^i - 1$ n'est pas un diviseur de zéro pour les i tels que $\omega^i \neq 1$. Dans ces conditions l'interpolation des coefficients à partir des exposants est efficace ainsi que l'opération inverse, l'évaluation multipoint qui correspond à un produit matrice-vecteur.

Fait 3.4. *Soit $\mathbb{A}$ un anneau, $F \in \mathbb{A}[x]$ un polynôme creux de degré au plus D, avec pour exposants $\{e_0, \cdots e_{T-1}\}$ et $\omega \in \mathbb{A}$ une racine principale de l'unité d'ordre supérieur à D.*

- *L'interpolation des coefficients de F à partir des évaluations de F sur les T premières puissances d'ω*
- *et l'évaluation de F sur ces mêmes T points à partir des coefficients*

peuvent être calculées en $\mathcal{O}(\mathsf{M}(T)\log T + T\log D) = \tilde{\mathcal{O}}(T\log D)$ opérations dans $\mathbb{A}$.

Les étapes 4 et 5 qui recherchent les racines d'un polynôme puis en calculent les logarithmes discrets, peuvent être plus coûteuses. La recherche de racines, dont l'efficacité dépend de $\mathbb{A}$, peut s'effectuer avec des algorithmes polynomiaux mais pas nécessairement quasi-linéaires [KU11 ; Bha+22 ; HHN11]. Par ailleurs, sans ajouter de contraintes sur la base ω du logarithme discret et sur l'anneau $\mathbb{A}$, aucun algorithme connu n'est polynomial en la taille des exposants, $\log D$. En particulier si ω est un entier la taille binaire des différents ω^{e_i}, qui est de $D\log\omega$ bits empêche d'atteindre une complexité polynomiale en $\log D$. En considérant la taille des nombres manipulés, il peut être cohérent d'essayer déjà d'optimiser la complexité par rapport à D. Ces deux étapes de recherche de racines et de calcul de logarithmes discrets peuvent être effectuées de manière combinée en $\tilde{\mathcal{O}}(D)$ opérations dans $\mathbb{A}$. Pour cela, il suffit de s'appuyer sur la transformée de Bluestein [Blu70] pour évaluer Λ sur les points $1, \omega, \ldots, \omega^D$. Les zéros obtenus donnent à la fois les racines de Λ et les exposants de F.

Fait 3.5. *À partir du polynôme Λ de l'étape 3, les exposants de F peuvent être calculés en $\mathsf{M}(D)$ opérations dans $\mathbb{A}$.*

Une dernière difficulté de l'algorithme est l'existence même d'un point ω d'ordre suffisant dans l'anneau $\mathbb{A}$ et son obtention. Ce problème n'apparaît pas systématiquement

et dans la version de cet algorithme sur $\mathbb{Z}$ telle qu'initialement proposée par Ben-Or et Tiwari, il ne pose aucune difficulté. Ils choisissent pour ω un nombre premier et dans le cas multivarié un nombre premier différent pour chaque variable et effectuent alors une factorisation pour obtenir les exposants.

Cette discussion permet de justifier une analyse générale du coût de l'interpolation creuse à la Prony.

Théorème 3.6. *Soit $F \in \mathbb{A}[x]$ de degré au plus D avec au plus T monômes. Étant donné un point ω d'ordre supérieur à D, l'*INTERPOLATION DE PRONY *:*

- *peut être effectuée en $\tilde{\mathcal{O}}(D)$ opérations dans $\mathbb{A}$*
- *nécessite exactement $2T$ évaluations par la boîte noire.*

Le fait que l'algorithme ne soit pas polynomial dans la taille binaire de F concerne à la fois le coût de la recherche de racines et du logarithme discret par rapport à $\log D$, mais aussi le coût lié à la taille des nombres manipulés. Comme nous l'avons évoqué utiliser l'interpolation de Prony sur des entiers conduit à manipuler des grands nombres. Pour un polynôme F dans $\mathbb{Z}[x]$ de degré D, avec des coefficients bornés par H, $F(\omega)$ a une taille binaire de $D \log \omega + \log H$ qui est exponentielle en $\log D$. La manipulation des nombres et même simplement leur lecture après l'évaluation par la boîte noire représentent donc une véritable difficulté.

3.1.1.1. Extensions de la méthode de Prony et Ben-Or–Tiwari

De nombreuses extensions ou améliorations ont été apportées à l'algorithme initial de Prony ou Ben-Or–Tiwari depuis la fin des années 80 [Zip90 ; KL88 ; HG19].

Puisque la première version se concentrait sur les polynômes dans $\mathbb{Z}[x]$, une extension naturelle a été l'étude de ses techniques dans les corps finis [GKS90 ; HR99 ; JM10 ; GR11b ; Hua21]. L'étape la plus coûteuse dans ce cas est celle où sont calculés les logarithmes discrets. Pour contourner cette difficulté, il est classique de modifier le modèle pour autoriser des évaluations non seulement sur des points du corps $\mathbb{F}_q$ mais aussi dans des anneaux de polynômes $\mathbb{F}_q[y]/\langle P(y)\rangle$. D'un autre côté, l'algorithme sur les entiers a aussi été perfectionné, notamment en basculant dans les corps finis après l'évaluation pour manipuler des nombres de taille limitée [KLW90 ; MF96 ; HL15].

Une autre question a été soulevée par Ben-Or et Tiwari en 1988 [BT88] : peut-on se passer des bornes D et T pour interpoler un polynôme F ? Ils apportent une réponse positive mais seulement dans le cas de polynômes dans $\mathbb{N}[x]$. En 2003, Kaltofen et Lee proposent une méthode générale s'appuyant sur la terminaison anticipée de l'algorithme de Berlekamp-Massey pour calculer F en ayant seulement la borne sur le degré D [KL03]. L'algorithme nécessite donc moins d'informations initiales pour fonctionner et aura une complexité qui dépendra du véritable nombre de monômes de F plutôt que d'une borne. Le nombre d'évaluations par la boîte noire n'augmente pas, en revanche elles ne peuvent plus être effectuées en une seule fois.

Pour l'amélioration de l'interpolation sur les entiers, la taille même des évaluations représente une limite infranchissable et montre que dans le modèle de la simple boîte noire tout algorithme sera forcément exponentiel en $\log D$. Pour dépasser cette difficulté, un nouveau modèle légèrement modifié a été proposé, celui de la boîte noire modulaire. Dans ce contexte la boîte noire peut évaluer le polynôme modulo n'importe quel entier fourni en même temps que le point d'évaluation.

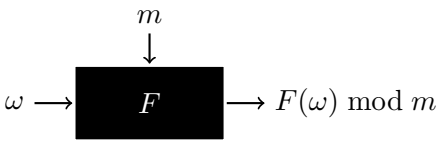


FIGURE 3.2. – Boîte noire modulaire

Ainsi les nombres obtenus grâce à l'évaluation de la boîte noire ont toujours une taille bornée. Si cette borne correspond à celle sur les coefficients du polynôme F, l'arithmétique modulo m est alors quasi-linéaire, ce qui permet d'envisager des algorithmes d'interpolation plus efficaces. Ce modèle a été largement utilisé [GR10 ; KN11 ; GR11b ; BJ14] et a permis à Kaltofen [Kal10] de proposer un algorithme qui soit polynomial en $\log D$. Pour cela, les évaluations sont effectuées dans un corps $\mathbb{F}_q$ pour un premier $q > D$ tel que $q - 1$ est divisible par une grande puissance de 2 pour permettre des logarithmes discrets efficaces. La difficulté que rencontre cette approche est l'obtention du nombre premier q de la bonne forme. D'une part, pour s'assurer de l'existence d'un tel premier q supérieur à D mais pas exponentiellement plus grand, il faut s'appuyer sur les conjectures évoquées en section 2.2.2, (en particulier une version de l'hypothèse 2.1, page 38 pour tout entier n et pas seulement un premier p). Même sous cette hypothèse, l'obtention de q est plutôt coûteuse à cause des tests de primalité à effectuer. Ainsi, l'approche de Kaltofen a une complexité polynomiale en $\log D$ mais avec un exposant important.

Bien que l'algorithme de Ben-Or–Tiwari soit déterministe, les améliorations les plus efficaces sont elles probabilisites.

3.1.1.2. Autres interpolations par boîte noire

Dans la partie précédente j'ai choisi de me concentrer sur l'algorithme de Prony, Ben-Or–Tiwari pour son ancienneté, son caractère déterministe et son impact sur la recherche. L'algorithme présenté en section 3.2 repose en partie sur celui-ci. Cependant d'autres techniques d'interpolation existent. Je voudrais notamment mentionner l'algorithme de Zippel [Zip90] qui a comme particularité dans le cadre multivarié de traiter les variables les unes après les autres. Mansour [Man95] a proposé un algorithme pour interpoler un polynôme à coefficients entiers en utilisant le calcul numérique et en considérant une boîte noire qui évalue sur $\mathbb{C}$ à n'importe quelle précision. Les exposants sont reconstruits bit par bit avant d'obtenir les coefficients, le tout en temps polynomial.

L'utilisation de l'algorithme de Prony dans le cadre du calcul numérique avec une boîte

noire sur $\mathbb{C}$ qui n'est pas traitée ici a cependant un intérêt propre. Elle est reliée à d'autres problèmes algébriques [JLM19] et a des applications pratiques notamment en traitement du signal [RK89 ; SK92].

Enfin, l'interpolation de boîte noire ne concerne pas seulement des polynômes mais aussi le cas des fonctions rationnelles ayant une représentation creuse. Ce problème connexe a aussi suscité l'intérêt des chercheurs et les algorithmes proposés pour le résoudre [KY07 ; KN11 ; CL11 ; HL21] peuvent utiliser des techniques proches de celles de l'interpolation de polynômes creux. Il est intéressant de noter que dans ces cas, la fraction rationnelle est donnée sous la forme d'une fraction de polynômes creux qui n'est pas forcément une fraction réduite.

3.1.2. Pour un polynôme sous forme de circuit arithmétique

Pour améliorer les algorithmes d'interpolation par boîte noire, nous avons vu qu'il peut être nécessaire d'autoriser l'évaluation dans d'autres anneaux que l'anneau de définition du polynôme. Le modèle du circuit arithmétique permet justement d'évaluer le polynôme dans tout anneau dans lequel les constantes de l'anneau d'origine peuvent être envoyées. Il s'agit en effet de la description du polynôme par une suite d'additions et de multiplications qui à partir de constantes permet d'évaluer le polynôme en tout point ω.

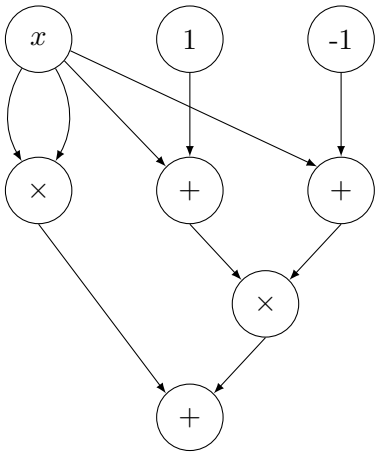


FIGURE 3.3. – Circuit arithmétique pour $F(x) = 2x^2 - 1$

Contrairement à la boîte noire le circuit arithmétique a une taille propre qui doit être prise en compte dans l'analyse de l'algorithme. Celle-ci est donnée par sa longueur L, c'est-à-dire le nombre d'opérations, et la taille des constantes utilisées. Pour simplifier, on ne considérera pas le nombre exact de constantes du circuit, seulement qu'il en a au plus L, et donc a pour taille LB où B est une borne sur la taille des constantes. À partir de la taille du circuit, on peut essayer de donner des bornes à priori sur les paramètres du polynôme qu'il décrit. Le degré comme le nombre de monômes sont bornés par 2^L et la taille binaire de ses coefficients par $2^L B$ dans des anneaux à coefficients non bornés.

Dans le cadre des polynômes creux, c'est le logarithme du degré qui importe vraiment. La borne sur le degré qui donne une taille binaire linéaire en L pourrait donc être utilisée pour des applications pratiques. Ce n'est pas le cas des bornes sur le nombre de monômes non nuls ou la taille binaire des coefficients qui sont exponentielles. Elles sont de plus inadaptées dans la plupart des cas, ces bornes n'étant généralement pas atteintes.

Dans le cadre de l'interpolation d'un polynôme creux F donné par un un circuit arithmétique, il convient donc de considérer que des bornes sont fournies comme des entrées du problème en plus du circuit lui-même. Les bornes considérées sont le degré D et le nombre T de monômes non nuls avec si nécessaire une borne sur les coefficients.

Une autre différence entre le circuit arithmétique et la boîte noire vient du coût de l'évaluation. Si dans le second cas, il ne s'agissait que du coût d'écriture du point et de lecture de son évaluation, pour le circuit arithmétique toutes les opérations doivent être effectuées. Une évaluation requiert donc L opérations dans l'anneau d'évaluation. Ceci limite le choix des anneaux d'évaluation. En particulier, s'il est possible d'obtenir directement le polynôme F en évaluant le circuit arithmétique en x dans l'anneau de polynômes $\mathbb{A}[x]$ où F est défini, une telle évaluation aurait un coût exponentiel. En effet, chaque opération du circuit s'effectue entre deux polynômes dont le degré et le nombre de monômes non nuls sont bornés uniquement par le degré maximal possible à savoir 2^L. Ces opérations intermédiaires peuvent alors s'effectuer dans des tailles exponentiellement plus grandes que celle du polynôme F et du circuit.

Ce modèle a cependant permis à Garg et Schost en 2009 [GS09] de proposer le premier algorithme d'interpolation de polynôme creux qui soit déterministe et de complexité polynomial en T et $\log D$. De plus, l'algorithme fonctionne sur n'importe quel anneau.

3.1.2.1. Algorithme de Garg et Schost

L'idée principale est que si l'évaluation dans $\mathbb{A}[x]$ est trop coûteuse c'est qu'elle fait intervenir des polynômes de degré non borné. On peut alors passer dans un anneau quotient $\mathbb{A}[x]/(x^p-1)$ dans lequel le degré (et donc le nombre de monômes non nuls) de tous les polynômes impliqués dans les calculs reste inférieur à p.

L'évaluation du circuit en x dans $\mathbb{A}[x]/(x^p-1)$ fournit seulement le polynôme $F \bmod x^p-1$ et non le polynôme F lui-même. Il y donc eu une perte d'information pour deux raisons possibles. La première est qu'un monôme $f_i x^{e_i}$ de F est envoyé sur $f_i x^{e_i \bmod p}$, les exposants de F ne sont donc connus que modulo p. L'autre source de perte d'information vient de possibles collisions modulo x^p-1 : si $e_i \equiv e_j \bmod p$, les coefficients des monômes $f_i x^{e_i}$ et $f_j x^{e_j}$ seront perdus puisque seule leur addition f_i+f_j apparaîtra dans le polynôme $F \bmod x^p-1$.

Le problème des collisions se résout en s'appuyant sur le corollaire 2.11 (page 29), qui indique le nombre maximal de nombres premiers pour lesquels F a au moins une collision modulo x^p-1. Il suffit alors de prendre plus de premiers p_i distincts et de calculer tous

les $F \bmod x^{p_i} - 1$ pour être sûr que le polynôme $F \bmod x^p - 1$ qui a le plus de monômes non nuls est sans collision.

Pour retrouver les exposants d'origine, l'idée est de s'appuyer sur le théorème des restes chinois en prenant suffisamment de nombres premiers p_k distincts, avec F sans collision. Ce théorème n'est pas appliqué directement au exposants car il est impossible de savoir directement l'ensemble des réductions modulo les différents p_k, d'un seul exposant. En effet si deux coefficients f_i et f_j sont égaux, il est impossible à partir du monôme $f_i x^{e_i \bmod p_k}$ de déterminer s'il correspond à $f_i x^{e_i}$ ou à $f_j x^{e_j}$ sans connaître e_i ou e_j. En revanche, l'ensemble des exposants est connu modulo différents p_k, le théorème des restes chinois peut donc s'appliquer pour reconstruire le polynôme $\prod_{i=0}^{T-1}(z - e_i) \in \mathbb{Z}[x]$ à partir de ses projections dans $\mathbb{F}_{p_k}[x]$. Les exposants de F sont les racines de ce polynôme et peuvent être retrouvées efficacement.

Une fois les exposants connus, les coefficients peuvent être retrouvés à partir de l'un des $F \bmod x^p - 1$ sans collision puisque le coefficient du monôme de degré e_i de F correspond alors exactement à celui du monôme de degré $e_i \bmod p$ de $F \bmod x^p - 1$.

Ces points permettent de comprendre l'algorithme de Garg et Schost dont voici une version explicite ainsi que sa complexité.

Algorithme 8 Interpolation de Garg et Schost

Entrée : Un circuit arithmétique de longueur L pour un polynôme creux $F \in \mathbb{A}[x]$, D et T des bornes sur le degré et le nombre de monômes non nuls de F.

Sortie : Le polynôme $F = \sum_{i=0}^{T-1} f_i x^{e_i}$ sous forme creuse

1: Générer l'ensemble $\mathcal{P}$ des $T^2 \log D + T \log D$ plus petits nombres premiers
2: Évaluer le circuit en x, dans chaque anneau $\mathbb{A}[x]/(x^p - 1)$ pour $p \in \mathcal{P}$ et ne garder que ceux avec le support maximal
3: Calculer le polynôme $\prod_{i=0}^{T-1}(z - e_i)$ dans les différents $\mathbb{F}_p[x]$ et le reconstruire dans $\mathbb{Z}[x]$.
4: Calculer les exposants qui sont les racines de $\prod_{i=0}^{T-1}(z - e_i)$ dans $\mathbb{Z}[x]$
5: Retrouver les coefficients à partir d'un $F \bmod x^p - 1$ sans collision.
6: **Renvoyer** les paires (f_i, e_i) pour i entre 0 et T

Théorème 3.7 ([GS09, Theorem 1])**.** *À partir d'un circuit arithmétique de longueur L décrivant un polynôme $F \in \mathbb{A}[x]$ et de bornes D et T, sur le degré et le nombre de monômes non nuls de F, il est possible de calculer la représentation creuse de F en effectuant $\tilde{\mathcal{O}}(LT^4(\log D)^2)$ opérations dans $\mathbb{A}$ plus un nombre similaire d'opérations binaires.*

S'il est polynomial dans les différents paramètres, l'algorithme de Garg et Schost n'est pas encore optimal, ayant d'importantes puissances dans sa complexité.

De nombreuses propositions ont été faites pour l'améliorer, particulièrement dans le cadre de circuits arithmétiques définis sur des corps finis. Il y a deux outils principaux : la diversification [GR11b] et la progressivité de la reconstruction de F [AGR13 ; AGR14 ; Arn16 ; HG20].

La diversification consiste à passer du polynôme $F(x)$ au polynôme $F(\alpha x)$ pour un α aléatoire. Si α est choisi dans un ensemble assez vaste, tous les coefficients de $F(\alpha x)$ sont distincts. Ainsi, en l'absence de collision il est possible de déterminer quels monômes des polynômes $F \bmod x^{p_i} - 1$, pour des p_i distincts, correspondent au même monôme d'origine dans F.

Pour la progressivité, il s'agit de ne pas reconstruire les monômes de F en une seule fois, mais en plusieurs étapes. Pour cela, il suffit de préserver seulement une partie, et non la totalité, des monômes de F des collisions modulo $x^p - 1$. Cette fraction de monômes sans collision peut-être reconstruite à partir de la réduction modulo $x^p - 1$. Les monômes restant forment un polynôme de plus petite taille qui est ensuite interpoler de manière récursive. Pour cela, les algorithmes s'appuient sur différentes versions du corollaire 2.13 (page 30) qui explique comment obtenir un nombre premier p pour qu'une proportion de monômes soit probablement épargnée par les collisions. Cette proportion peut être adaptée en fonction des spécificités de chaque algorithme. À cause des collisions des monômes erronés peuvent être construits mais leur nombre, comme celui de collisions, est limité. Ils seront alors supprimés lors d'étapes ultérieures de l'algorithme. Cette approche a l'avantage de réduire la valeur des nombres premiers p utilisés en passant de quadratique en T à linéaire en T.

Dans la plupart des cas, les algorithmes obtenus sont des algorithmes probabilistes, et seuls ceux-là atteignent une complexité quasi-linéaire en T.

Sur un corps fini quelconque, la complexité reste cubique en $\log D$ même pour l'algorithme le plus rapide qui s'appuie sur la diversification .

Théorème 3.8 ([AGR14])**.** *À partir d'un circuit arithmétique décrivant un polynôme $F \in \mathbb{F}_q[x]$ et de bornes D et T, sur le degré et le nombre de monômes de F, il est possible de calculer, avec une probabilité de succès d'au moins $1 - \epsilon$ pour $0 < \epsilon < 1$, la représentation creuse de F en effectuant $\tilde{\mathcal{O}}(LT(\log D)^2(\log D + \log q)\log \frac{1}{\epsilon})$ opérations binaires.*

L'algorithme de Garg et Schost étant défini pour tout anneau, il fonctionne directement sur $\mathbb{Z}$. Cependant la taille des coefficients n'est pas bornée lors des opérations propres au circuit arithmétique. Elle pourrait devenir exponentielle alors qu'il s'agit de calculs intermédiaires. Pour éviter cela, une approche naturelle est de travailler modulo différents petits nombres premiers et de reconstruire F en utilisant le théorème des restes chinois. Cette approche nécessite l'interpolation dans des corps $\mathbb{F}_{q^i}$ tels que $\prod_i q_i > 2H$, avec H une borne sur la hauteur de F. En prenant le nombre minimum de premiers q_i distincts, on a $\sum_i \log q_i = \mathcal{O}(\log H)$. Si en plus la probabilité d'échec pour l'interpolation dans chacun des corps finis correspondant est inférieure à $\frac{\epsilon}{\log H}$, il est possible d'obtenir le corollaire suivant pour l'interpolation de polynômes à coefficients entiers.

Corollaire 3.9. *À partir d'un circuit arithmétique décrivant un polynôme $F \in \mathbb{Z}[x]$ et de bornes D, T et H, sur le degré, le nombre de monômes non nuls et la hauteur de F, il est possible de calculer, avec une probabilité de succès d'au moins $1-\epsilon$ pour $0 < \epsilon < 1$, la représentation creuse de F en effectuant $\tilde{\mathcal{O}}(LT \log H (\log D)^3 \log \frac{1}{\epsilon})$ opérations binaires.*

Si tous ces algorithmes ne parviennent pas à baisser le facteur $(\log D)^3$, c'est à cause de la reconstruction des exposants par le théorème des restes chinois qui nécessite systématiquement de répéter les calculs pour environ $\log D$ nombres premiers.

3.1.3. Interpoler en utilisant aussi les dérivées du polynôme

En 2019, Huang propose une nouvelle méthode pour l'interpolation d'un polynôme donné par un circuit arithmétique qui ne nécessite plus de répéter $\log D$ fois les mêmes opérations [Hua19]. L'idée principale est d'écrire les exposants dans les coefficients pour pouvoir y accéder à partir de tout monôme sans collision d'une réduction modulo $x^p - 1$. Une telle approche a aussi été utilisée pour le calcul de la somme de deux ensembles d'entiers [AR15] permettant ainsi de construire un nouvel algorithme pour le produit de polynômes creux. Dans le cadre des polynômes, il existe un moyen naturel de faire passer les exposants dans les coefficients : la dérivation.

Pour un polynôme donné par un circuit arithmétique de longueur L, la différentiation automatique de Baur et Strassen [BS83] permet de construire en temps $\mathcal{O}(L)$ un circuit de longueur au plus $3L$ qui correspond à sa dérivée. Leur méthode permet en fait de produire toutes les dérivées partielles d'un polynôme multivarié.

Ainsi dans le cadre des circuits arithmétiques, l'évaluation du polynôme F ou de sa dérivée F' s'effectue asymptotiquement dans le même temps. Comme dans les méthodes vues précédemment pour l'interpolation d'un polynôme creux donné par un circuit arithmétique, Huang évalue F et F' en x modulo $x^p - 1$ pour des p tels que F a probablement peu de collisions. L'évaluation des deux polynômes permet de retrouver les exposants des monômes sans collision car ceux-ci sont sans collision à la fois dans F et F'.

Fait 3.10. *Pour p un entier, si un monôme $f_i x^{e_i}$ d'un polynôme $F \in \mathbb{A}[x]$ est sans collision modulo $x^p - 1$ alors $f_i x^{e_i \bmod p}$ est un monôme du polynôme $F \bmod x^p - 1$ et $e_i f_i x^{(e_i - 1) \bmod p}$ est un monôme du polynôme $F' \bmod x^p - 1$.*

Les deux monômes des polynômes modulo $x^p - 1$ sont obtenus par les circuits arithmétiques sous la forme $f_i x^{d_i}$ et $g_i x^{d_i - 1}$. L'écart de 1 entre les degrés marque la correspondance entre les monômes, le premier a pour coefficient le coefficient du monôme d'origine et l'exposant du monôme d'origine est simplement $\frac{g_i}{f_i}$.

Les exposants sont des entiers et non des éléments de $\mathbb{A}$ contrairement aux coefficients f_i et g_i. Le morphisme canonique de $\mathbb{Z}$ vers $\mathbb{A}$ permet bien d'écrire l'exposant dans $\mathbb{A}$ mais pas forcément de retrouver de manière unique l'exposant à partir de $\frac{g_i}{f_i}$. Le morphisme entre $\mathbb{Z}$ et $\mathbb{A}$ n'est injectif que si $\mathbb{A}$ est de caractéristique zéro. Puisqu'une borne D est donnée sur les exposants, il suffit que le morphisme soit injectif pour l'ensemble des

entiers entre 0 et D. Autrement dit cette méthode ne fonctionne qu'en caractéristique zéro ou supérieure strictement à D.

La dérivation, plutôt que la diversification, est la clef qui permet d'obtenir un algorithme plus efficace pour les circuits arithmétiques. La diversification utilise les coefficients des polynômes $F(\alpha x) \bmod x^{p_i} - 1$ avec de nombreux p_i distincts pour identifier quels monômes sont des réductions du même monôme de F. L'exposant de ce monôme est reconstruit grâce au théorème des restes chinois. Avec la dérivée, le calcul de seulement deux polynômes, F et F', modulo $x^p - 1$ pour un seul p est suffisant à la reconstruction de monômes de F.

Le nombre premier p est choisi pour permettre une reconstruction récursive de F. Comme cela a été évoqué pour les améliorations de l'algorithme de Garg et Schost, il est possible en s'appuyant sur le corollaire 2.13 (page 30) de contrôler la proportion de monômes de F en collision modulo $x^p - 1$. Les monômes sans collision peuvent directement être reconstruits à partir de $F \bmod x^p - 1$ et $F' \bmod x^p - 1$. Les monômes en collisions ne peuvent être directement calculés, leurs collisions peuvent même entraîner la reconstruction de monômes erronés. Le calcul des monômes manquants et la correction des erreurs sont effectuées de manière récursive. Pour que la récursion se fasse en taille réduite, il faut contrôler le nombre d'ajouts ou de corrections à effectuer après une étape. La remarque 2.16 (page 31) indique quelle proportion des monômes de F doit être sans collision pour que la distance entre F et $F \bmod x^p - 1$ soit inférieure à $\frac{1}{2}T$. Cette distance correspond exactement aux nombres de monômes qu'il reste à retrouver ou à supprimer après en avoir reconstruit à partir de $F \bmod x^p - 1$ et $F' \bmod x^p - 1$. Fixer une borne de $\frac{1}{2}T$ sur la distance entre F et $F \bmod x^p - 1$ permet de s'assurer que l'ensemble des véritables coefficients de F pourra être reconstruit en $\log T$ étapes.

L'ensemble de ces éléments permet de décrire l'algorithme 9 Interpolation de Huang.

Théorème 3.11 ([Hua19, Theorem 3.10])**.** *À partir d'un circuit arithmétique de longueur L décrivant un polynôme $F \in \mathbb{A}[x]$ et de bornes D et T, sur le degré et le nombre de monômes non nuls de F, avec la caractéristique de $\mathbb{A}$ qui est nulle ou strictement supérieure à D, il est possible de calculer la représentation creuse de F avec une probabilité d'échec inférieure à ϵ pour $0 < \epsilon < 1$. Ce calcul effectue $\tilde{\mathcal{O}}(LT \log D \log \frac{1}{\epsilon})$ opérations dans $\mathbb{A}$ plus un nombre similaire d'opérations binaires.*

L'algorithme ne gère pas les croissances potentielles de coefficients au cours de son exécution. Pour limiter la croissance si $\mathbb{A} = \mathbb{Z}$, il faudrait passer sur des corps finis. Pour les algorithmes n'utilisant pas la dérivée, cela peut se faire en utilisant des petits corps finis et donc sans impacter le coût global de l'algorithme (corollaire 3.9). Dans le cas de l'algorithme de Huang en revanche, pour pouvoir passer sur $\mathbb{F}_q$, il faut avoir $q > D$. Ainsi la génération d'un tel nombre premier ajoute un terme cubique en $\log D$ à la complexité (fait 1.13). C'est pourquoi nous ne détaillons le coût binaire de l'algorithme que sur des corps finis.

Corollaire 3.12. *Si $\mathbb{A} = \mathbb{F}_{q^s}$ pour un nombre premier $q > D$, la complexité binaire de*

Algorithme 9 Interpolation de Huang

Entrée : Une circuit arithmétique de longueur L pour un polynôme creux $F \in \mathbb{A}[x]$, D et T des bornes sur le degré et le nombre de monôme de F. $\mathbb{A}$ étant de caractéristique 0 ou $> D$.

Sortie : Le polynôme $F = \sum_{i=0}^{T-1} f_i x e_i$ sous forme creuse

1: $F^* \leftarrow 0$
2: Générer l'ensemble $\mathcal{P}$ des $\lceil 12(T-1)\log D \rceil$ plus petits nombres premiers
3: **Tant que** $T \geq 1$ **faire**
4: Trouver un premier p dans $\mathcal{P}$ tel que la distance entre $F - F^*$ et $(F - F^*) \bmod x^p - 1$ soit probablement inférieure à $\frac{1}{2}\#(F - F^*)$
5: Calculer $F_p = (F - F^*) \bmod x^p - 1$ et $F'_p = (F' - F^{*\prime}) \bmod x^p - 1$
6: **Pour** tous les monômes fx^d de F_p **faire**
7: $g \leftarrow$ le coefficient du monôme de degré $d-1$ de F'_p
8: $e \leftarrow \frac{f}{g}$
9: **Si** e est un entier **alors** Ajouter le monôme fx^e à F^*
10: $T \leftarrow \lfloor \frac{T}{2} \rfloor$
11: **Renvoyer** F^*

l'algorithme de Huang est $\tilde{\mathcal{O}}(LT \log D \log q^s \log \frac{1}{\epsilon})$.

Cette complexité est certes linéaire dans chacun des paramètres mais pas dans la taille d'entrée ni de sortie de l'interpolation d'un polynôme creux donné par un circuit arithmétique. Sur $\mathbb{F}_{q^s}$, les entrées sont de taille $L \log q^s$. Cette taille correspond à celle du circuit arithmétique. Les bornes D et T ont une taille inférieure à L puisque $T \leq D \leq 2^L$, et donc la donnée de ces bornes en entrée n'augmente pas la taille des entrées. Le polynôme F quant à lui a une taille en $T(\log D + \log q^s)$. Le coût de l'algorithme peut donc être décrit comme le produit entre la taille des entrées et celle des exposants de la sortie.

Huang a par la suite proposé un nouvel algorithme [Hua21] qui permet de séparer L et $\log D$ dans la complexité. Celle-ci devient alors $\tilde{\mathcal{O}}(LT \log q^s + T \log D + T(\log q^s)^2)$. Cette séparation, s'accompagne en revanche d'une augmentation de la complexité en $\log q^s$. L'intérêt est aussi, dans le cadre multivarié de n'avoir qu'un seul terme $(T \log D)$ qui soit impacté par le nombre de variables.

Ce nouvel algorithme utilise le circuit pour générer des dérivées et permettre l'évaluation du polynôme et de sa dérivée sur les puissances d'un point ω tel que les ω^e pour e dans le support de F soient tous distincts. Les techniques de Prony permettent de retrouver les différents ω^e. Ceci correspond aux étapes 1 à 4 de l'algorithme 7 Interpolation de Prony. À partir de ceux-ci, et sans calculer de logarithmes discrets, il est possible de considérer les systèmes de type Vandermonde (fait 3.2) correspondants à F et F' pour trouver leur coefficients et de là les exposants. Ceci permet d'éviter l'étape 5 de l'algorithme de Prony, étape qui est la plus coûteuse sur les corps finis. En plus de

l'idée originale de passer par la dérivée, cet algorithme s'appuie exclusivement sur les techniques de Prony et n'utilise le circuit arithmétique que pour permettre la dérivation.

Kaltofen [Kal22] propose une autre approche de l'usage de la dérivée pour des méthodes d'interpolation à la Prony. Son but est de diminuer le nombre de points sur lesquels une évaluation doit être effectuée. En considérant une boîte noire pour le polynôme F et une boîte noire pour le polynôme dérivé F', il montre qu'il est possible d'interpoler le polynôme F en utilisant $3T$ évaluations par l'une ou l'autre des boîtes noires sur seulement $\frac{3}{2}T$ points d'évaluations. L'algorithme de Prony nécessite moins d'évaluations, seulement $2T$ mais sur plus de points : $2T$. De son côté, le dernier algorithme de Huang calcule autant d'évaluations que l'algorithme de Kaltofen ($3T$) sur plus de points ($2T$). La méthode se généralise même à l'utilisation de dérivées d'ordres supérieurs : avec l'utilisation de ses dérivées jusqu'à l'ordre l, le polynôme F peut être interpolé à partir de $(l+2)T$ évaluations différentes sur $\frac{l+1}{l}T$ points distincts.

L'idée d'utiliser la dérivée est en réalité antérieure aux travaux de Huang en 2019. Avendaño, Krick et Pacetti ont proposé en 2006 [AKP06], une interpolation d'un polynôme complexe donné par un circuit arithmétique en passant par l'évaluation de ses dérivées jusqu'à l'ordre T (le nombre de monômes). Cet algorithme qui repose sur du calcul numérique a cependant un facteur T^4 dans sa complexité, comme l'algorithme de Garg et Schost.

De manière générale, la section 3.1 s'est concentrée principalement sur la présentation des algorithmes les plus marquants pour l'avancée des connaissances dans ce domaine et dont les techniques servent à l'élaboration de notre algorithme quasi-linéaire dans $\mathbb{Z}[x]$. Pour plus de détails sur d'autres méthodes, il est recommandé de consulter l'article de van der Hoeven et Lecerf [HL19] qui offre un panorama des différentes techniques.

3.2. Interpolation quasi-linéaire pour un polynôme sur les entiers

Dans les algorithmes d'interpolation décrits précédemment, l'approche la plus efficace pour interpoler un polynôme creux dans $\mathbb{Z}[x]$ était généralement de s'appuyer sur les algorithmes pour les polynômes à coefficients dans un corp fini. Le passage dans un corps fini permet en effet d'empêcher la croissance des nombres manipulés. Dans cette section, nous présentons un algorithme qui est conçu spécifiquement pour les polynômes à coefficients entiers.

Le polynôme F à interpoler dans cette section est donné par une boîte noire modulaire (BNM), qui permet de l'évaluer sur n'importe quel nombre entier dans tout anneau $\mathbb{Z}/m\mathbb{Z}$ pour $m \in \mathbb{N}^*$ (voir section 3.1.1.1). Ce modèle est choisi car il représente les conditions minimales pour permettre le fonctionnement de l'algorithme. Comme déjà mentionné, la boîte noire non-modulaire ne permet pas de contrôler la taille des évaluations et donc d'atteindre l'optimalité. Le circuit arithmétique permet de contrôler la

taille des évaluations mais contient beaucoup plus d'informations que nécessaire pour l'algorithme, notamment la capacité de dériver le polynôme. Un algorithme pour une boîte noire modulaire s'adapte cependant au cas du circuit arithmétique qui permet de réaliser des évaluations modulaires.

Le problème d'interpolation étudié ici peut se présenter comme suit.

Problème. Étant donné une boîte noire modulaire décrivant un polynôme dans $\mathbb{Z}[x]$, ainsi que trois bornes D, H, T sur son degré, ses coefficients et le nombre de ses monômes non nuls, comment déterminer efficacement sa représentation creuse ?

L'efficacité est évaluée en prenant en compte d'une part le coût binaire des opérations effectuées et d'autre part le nombre d'appels à la boîte noire pour évaluation. Dans le cas que nous considérons, celui d'un polynôme univarié, le coût des appels à la boîte noire, écriture du point d'évaluation et lecture du résultat, est compris dans le coût de des opérations qui permettent de déterminer le modulo m puisque ce sont des nombres inférieurs à m. Nous nous concentrerons donc sur le coût des opérations qui sera comparé à la taille a priori du polynôme F : $T(\log D + \log H)$.

L'algorithme décrit dans cette section est le premier algorithme quasi-linéaire pour l'interpolation d'un polynôme creux dans $\mathbb{Z}[x]$, répondant à une question ouverte [Roc18b], dans le cas particulier où la boîte noire est modulaire.

Les idées principales permettant de faire fonctionner l'algorithme sont présentées dans la section 3.2.1. Des points plus techniques pour justifier le choix des différents modulo ainsi que la possibilité de travailler avec eux sont développés ensuite (section 3.2.2). Ces éléments permettent de donner une description détaillée ainsi que la preuve de l'algorithme (section 3.2.3).

Les travaux présentés ici viennent d'un article publié en 2022 avec Pascal Giorgi, Bruno Grenet et Daniel S. Roche [Gio+22].

3.2.1. Idées générales

Ce nouvel algorithme d'interpolation creuse s'appuie sur des techniques existantes pour l'interpolation de polynômes creux donnés tant par des boîtes que par des circuits arithmétiques. Ces méthodes ont été décrites en section 3.1. Mêmes si elles s'appliquent à différents modèles, ces méthodes peuvent être articulées ensemble pour résoudre l'interpolation d'un polynôme creux $F \in \mathbb{Z}[x]$ donné par une boîte noire modulaire.

3.2.1.1. Une approche récursive

Comme dans les travaux d'Arnold, Giesbrecht et Roche [AGR13 ; AGR14] ou de Huang et Gao [HG20 ; Hua19 ; Hua21], l'approche générale consiste à construire récursivement de *potentiels* monômes de F. Ces monômes sont calculés à partir du polynôme $F \bmod x^p - 1$. Le nombre p est un premier choisi pour que la distance entre F et $F \bmod x^p - 1$ soit de

$\frac{1}{2}T$ (voir remarque 2.16), assurant ainsi la reconstruction intégrale de F en $\log T$ étapes. En effet, chaque étape vise à interpoler un polynôme ayant moitié moins de monômes que celui de l'étape précédente.

Le premier p ainsi choisi est suffisamment petit, $p \in \mathcal{O}(T \log D)$ pour que des opérations de complexité binaire quasi-linéaire en p soient quasi-linéaires dans la taille de F. Dans les algorithmes d'interpolation de circuit arithmétique cités, le choix de p est suivi de l'évaluation du circuit arithmétique modulo $x^p - 1$. Pour cela les opérations du circuit sont réalisés par des algorithmes d'arithmétique de polynômes denses de degré p. La complexité de ces opérations est quasi-linéaire en p fois la taille des coefficients manipulés, et donc non quasi-linéaire dans la taille de F. Calculer $F \bmod x^p - 1$ par des opérations arithmétiques de polynômes semble donc inapproprié.

L'idée est donc d'éviter de calculer directement $F \bmod x^p - 1$, mais de l'interpoler à la place.

Cette interpolation est même découpée en deux étapes : d'abord les exposants et ensuite les coefficients. Pour chacune de ces étapes un modulo différent est choisi.

3.2.1.2. Calcul des exposants de $F \bmod x^p - 1$ dans un petit corps

La méthode utilisée pour trouver les exposants de $F \bmod x^p - 1$ se base sur l'interpolation creuse à la Prony (voir section 3.1.1). Cette interpolation s'appuie sur l'évaluation dans les puissances d'un point ω d'ordre supérieur au degré. Si ω est d'ordre p, le polynôme interpolé est $F \bmod x^p - 1$ ce qui est exactement le but recherché.

Pour avoir un tel ω, l'évaluation ne doit pas s'effectuer dans $\mathbb{Z}$, mais modulo un nombre premier q tel que $\mathbb{F}_q$ a un sous-groupe multiplicatif d'ordre p. Un tel premier existe avec $q \in \text{poly}(p)$ (voir section 2.2) ce qui fournit un petit corps. Par petit, il faut entendre que vu la valeur de p, la taille de q et de tous les éléments du corps est logarithmique dans la taille des entrées.

En ayant ainsi baissé la valeur du degré et la taille des coefficients il est possible d'effectuer un algorithme basé sur la méthode de Prony en temps quasi-linéaire.

Le fait que q soit petit empêche en revanche de retrouver les vrais coefficients de $F \bmod x^p - 1$. Ils sont seulement connus modulo q.

3.2.1.3. Calcul des coefficients de $F \bmod x^p - 1$ dans un grand anneau

Pour retrouver, les vraies valeurs des coefficients il faut donc travailler dans $\mathbb{Z}/m\mathbb{Z}$ avec un m plus grand : $m > H$. La taille de m est alors trop importante pour qu'un algorithme à la Prony reste quasi-linéaire modulo m. Et même trop importante pour que la génération d'un nombre premier de cette taille ne soit pas trop coûteuse. Cependant, le polynôme $F \bmod x^p - 1$ a déjà été calculé modulo q, ses exposants sont donc connus. Il ne reste à réaliser que la dernière part de l'algorithme de Prony : l'étape 6 qui inverse

le système de type Vandermonde du fait 3.2. Cette étape effectue justement un nombre quasi-linéaire en $T \log p$ d'opérations dans l'anneau (fait 3.4).

Pour retrouver les coefficients, cette étape est effectuée dans un anneau $\mathbb{Z}/q^k\mathbb{Z}$ avec $q^k > 2H$. Prendre une puissance de q permet de trouver ce nouveau modulo sans coût additionnel et de trouver une nouvelle racine p-ième de l'unité ω_k modulo q^k efficacement. En effet, une racine p-ième de l'unité ω est déjà connue modulo q et par itération de Newton il est possible d'obtenir ω_k.

L'algorithme repose fortement sur cette capacité à changer de modulo en conservant de bonnes propriétés pour que chaque opération puisse être quasi-linéaire.

3.2.1.4. Écrire les exposants dans les coefficients

Le passage modulo q^k permet d'obtenir les coefficients de $F \bmod x^p - 1$. Pour les monômes sans collision ce sont les coefficients exacts de F. En revanche, les exposants de F ne sont connus que modulo p. Puisque nous savons comment obtenir les véritables coefficients, l'algorithme utilise l'idée d'écrire les exposants dans les coefficients [HL15 ; AR15 ; Hua19]. Dans son algorithme, Huang utilise la dérivée mais ceci n'est possible que dans le cadre des circuits arithmétiques.

Dans le cadre d'une boîte noire modulaire, l'idée inspirée d'un schéma de Paillier [Pai99], et proposée par Arnold et Roche [AR15] est plus adaptée. Il s'agit, étant donné un modulo m, de considérer à la fois le polynôme $F(x) \bmod x^p - 1$ et le polynôme $F((1+m)x) \bmod x^p - 1$ dans l'anneau $\mathbb{Z}/m^2\mathbb{Z}$. L'identité $(1+m)^{e_i} \bmod m^2 = 1 + e_i m$ garantit que le ratio entre les coefficients correspondants des deux polynômes permet de retrouver $e_i \bmod m$ dès qu'il n'y a pas de collisions.

Ainsi pour retrouver les exposants d'origine, en plus des coefficients, la dernière partie de l'algorithme de Prony doit être effectuée modulo q^{2k} avec $q^k > D$. Et ce à la fois pour le polynôme $F(x)$ et pour le polynôme $F((1+q^k)x)$ qui ont le même support déjà connu.

3.2.1.5. Répétitions

Les choix des nombres premiers reposant sur des algorithmes probabilistes, les différentes étapes décrites précédemment peuvent échouer. Pour pouvoir assurer une probabilité arbitraire de succès, la méthode consiste pour cet algorithme à effectuer des répétitions et choisir le polynôme majoritaire parmi tous ceux ainsi calculés.

3.2.2. Calculer avec les différents modulo

Notre approche repose fortement sur le fait de trouver des nombres premiers p, q avec de bonnes propriétés et des racines primitives p-ièmes de l'unité ω et ω_k se trouvant respectivement dans le corps $\mathbb{F}_q$ et l'anneau $\mathbb{Z}/q^{2k}\mathbb{Z}$.

Il faut à la fois s'assurer que les nombres premiers p et q ont de bonnes propriétés par rapport à F, que ω existe et puisse être trouvé efficacement et enfin que ω_k, qui est dans un anneau et non un corps, soit principale pour pouvoir appliquer le fait 3.4 et effectuer la fin de l'algorithme de Prony.

Le premier point est de s'assurer que F n'a pas trop de collisions modulo $x^p - 1$ pour pouvoir reconstruire suffisamment de monômes non nuls de F à chaque étape. Les monômes de F sont reconstruits à partir des coefficients des polynômes $F \bmod x^p - 1$ et $F((1 + q^k)x) \bmod x^p - 1$. La correspondance entre leurs coefficients et ceux des polynômes non réduits permet de reconstruire des monômes corrects. La distance entre F et $F \bmod x^p - 1$ (c'est la même pour $F((1+q^k)x)$, les supports étant identiques) détermine donc le nombre de monômes qu'il reste à reconstruire. Les conditions à poser sur p sont donc données par la remarque 2.16 (page 31) associée au corollaire 2.13 (page 30). Pour plus de clarté elles sont rappelées dans le fait suivant.

Fait 3.13. *Soit $F \in \mathbb{Z}[x]$ un polynôme de degré au plus D ayant au plus T monômes et $0 < \epsilon < 1$. Pour tout $\lambda \geq \max(21, \frac{5}{\epsilon}(T-1)\ln D)$, si p est un nombre premier aléatoirement choisi dans l'intervalle $[\lambda, 2\lambda]$ alors la distance entre F et $F \bmod x^p - 1$ est inférieure à $\frac{1}{2}T$ avec probabilité au moins $1 - \epsilon$.*

Pour le nombre premier q, la condition à respecter par rapport à F est simplement qu'il ne divise aucun des coefficients de $F \bmod x^p - 1$. Ainsi les monômes sans collision modulo $x^p - 1$ n'ont pas leur coefficients annulés modulo q et même ils sont inversibles modulo q^{2k}. De plus, que ce soit modulo q ou modulo q^{2k}, le polynôme $F \bmod x^p - 1$ est assuré de garder le même support. Une telle propriété signifie que q ne doit pas diviser le produit des coefficients de $F \bmod x^p - 1$, un nombre borné par H^T.

Avec ces deux points il est possible de s'appuyer sur les méthodes vues en section 2.2, en particulier l'algorithme 1 TRIPLET (page 33) pour générer un triplet (p, q, ω) ayant les bonnes propriétés.

Lemme 3.14. *Soit $F \in \mathbb{Z}[x]$ un polynôme de degré au plus D, ayant au plus T monômes de coefficients bornés par H, $\lambda \geq \max(\frac{2^{58}}{\epsilon^2}, \frac{5}{\epsilon}(T-1)\ln D, \sqrt[5]{\frac{48}{\epsilon}T\ln H})$ et $0 < \epsilon < 1$. Avec une probabilité d'au moins $1 - 3\epsilon$, TRIPLET(λ, ϵ) renvoie un triplet (p, q, ω) tel que :*

- *p et q sont premiers et ω une racine p-ième de l'unité dans $\mathbb{F}_q$,*
- *la distance entre F et $F \bmod x^p - 1$ est inférieure à $\frac{1}{2}T$,*
- *q ne divise aucun des coefficients de $F \bmod x^p - 1$.*

De plus, $\lambda \leq p \leq 2\lambda$ et $\lambda \leq q \leq 2\lambda^6$.

Démonstration. L'algorithme TRIPLET a une probabilité inférieure à ϵ de renvoyer ÉCHEC. Dans le cas contraire, les nombres générés vérifient la propriété du premier point (théorème 2.17 page 32). Par ailleurs, en appliquant le fait 3.13 et le fait que q ne doit pas diviser un nombre inférieur à H^T, il est clair que chacun des deux derniers points a une probabilité inférieure à ϵ d'échouer. D'où une probabilité totale de succès d'au moins

$1 - 3\epsilon$. Enfin l'algorithme TRIPLET garantit les bornes données sur les valeurs de p et q. □

Le lemme 3.15 explicite clairement dans quelle mesure un tel choix de p et de q permet de diminuer le nombre de monômes de F à retrouver. La méthode utilisée pour reconstruire des monômes repose, comme expliqué en section 3.2.1, sur le calcul de $F(x)$ et $F((1+q^k)x)$ modulo $x^p - 1$ et q^{2k} pour un k assez grand. Elle est décrite dans l'algorithme 10 RECONSTRUCTIONMONÔMES.

Algorithme 10 RECONSTRUCTIONMONÔMES

Entrée : $F(x) \bmod \langle x^p - 1, q^{2k}\rangle = \sum f_i x^{d_i}$ et $F((1+q^k)x) \bmod \langle x^p - 1, q^{2k}\rangle = \sum g_i x^{d_i}$ avec $q^k > \deg(F)$ et $q^{2k} > 2\,\|F\|$.
Sortie : G un polynôme contenant de potentiels monômes de F.

1: **Pour** $d_i \in \text{support}(F)$ **faire**
2: **Si** f_i est inversible modulo q^{2k} **alors**
3: $r_i \leftarrow g_i f_i^{-1} - 1 \bmod q^{2k}$ en prenant le représentant positif entre 0 et $q^{2k}-1$ pour r_i
4: $e_i \leftarrow \frac{r_i}{q^k}$ dans $\mathbb{Q}$
5: **Si** e_i est dans $\mathbb{N}$ **alors**
6: Ajouter le monôme $f_i x^{e_i}$ à G en prenant le représentant de f_i entre $-(q^{2k}-1)/2$ et $(q^{2k}-1)/2$.
7: **Renvoyer** G

Lemme 3.15. *Soit $F \in \mathbb{Z}[x]$ de degré au plus D, de degré au plus H et ayant au plus T monômes non nuls. Soit $p \in \mathbb{N}$ tel que $F \bmod x^p - 1$ est à une distance au plus $\frac{1}{2}T$ de F, q un nombre premier qui ne divise aucun coefficient de $F \bmod x^p - 1$ et k un entier tel que $q^k > D$ et $q^{2k} > 2H$.*

L'algorithme 10 renvoie un polynôme G, tel que $F - G$ a au plus $\frac{1}{2}T$ monômes.

Le calcul de G nécessite $\mathcal{O}(T\mathsf{I}(\log q^{2k}))$ opérations binaires.

Démonstration. Les supports de $F(x)$ et de $F((1+q^k)x$ étant identiques, les collisions modulo $x^p - 1$ sont entièrement définies par les collisions des monômes de F.

Considérons d'abord le cas d'un monôme $f_i x^{e_i}$ de F sans collision modulo $x^p - 1$. Puisqu'il n'y a pas de collision, le polynôme $F \bmod x^p - 1$ contient le monôme $f_i x^{e_i \bmod p}$ et le polynôme $F((1+q^k)x) \bmod x^p - 1$ contient le monôme de même degré $f_i(1+q^k)^{e_i} x^{e_i \bmod p}$, qui modulo q^{2k} a comme coefficient $f_i(1+e_i q^k)$. Puisque f_i est un coefficient de F et de $F \bmod x^p - 1$, il n'est pas divisible par le nombre premier q, donc inversible modulo q^{2k}. Ainsi il est possible de calculer r_i qui vaut $e_i q^k$ modulo q^{2k}. Comme $q^k > D$, en considérant la représentation de $\mathbb{Z}/q^{2k}\mathbb{Z}$ par $\{0, 1, \ldots, q^{2k}-1\}$, l'égalité $r_i = e_i q^k$ est également vraie sur $\mathbb{N}$. La division par q^k peut être effectuée sur les entiers pour obtenir e_i. Pour retrouver la valeur d'origine de f_i, il faut au contraire utiliser la représentation de $\mathbb{Z}/q^{2k}\mathbb{Z}$ par $\{-(q^{2k}-1)/2, \ldots, -1, 0, 1, \ldots, (q^{2k}-1)/2\}$. Puisque $q^{2k} > 2H$ toutes

les valeurs possibles de coefficients de F sont bien présentes dans cet ensemble. Par conséquent, c'est bien le monôme $f_i x^{e_i}$ qui est présent dans G et donc ôté à F lors de la différence. Le polynôme $F - G$ ne contient donc aucun des monômes sans collision de F.

En revanche si un monôme $f_i x^{e_i}$ est en collision, il n'est pas retrouvé par les calculs décrits puisque son coefficient n'est pas le seul à contribuer au monôme de degré $e_i \bmod p$ dans $F \bmod x^p - 1$. Ce monôme en revanche peut générer (si la division par q^k est possible) un monôme non présent dans F. Le nombre de ces monômes "perdus" ou construits "à tort" correspond exactement à la définition de la distance. De plus ce sont les seuls monômes de $F - G$, qui a donc bien au plus $\frac{1}{2}T$ monômes.

Le calcul de G, ne nécessite qu'un nombre constant d'opérations par monôme de $F \bmod x^p - 1$. Ces opérations sont effectuées sur des entiers bornés par q^{2k} et $F \bmod x^p - 1$ a au plus T monômes. D'où la complexité en $\mathcal{O}(T\mathsf{I}(\log q^{2k}))$ opérations binaires. □

Au cours de cette étape de l'algorithme qui permet de retrouver les véritables valeurs des coefficients et exposant, il faut connaître $F(x) \bmod \langle x^p - 1, q^{2k}\rangle$ et $F((1+q^k)x) \bmod \langle x^p - 1, q^{2k}\rangle$. Pour cela, nous utilisons une interpolation à la Prony ce qui nécessite d'évaluer le polynôme F sur ω_k une racine primitive p-ième de l'unité dans $\mathbb{Z}/q^{2k}\mathbb{Z}$.

Un moyen naturel d'obtenir un tel point serait d'effectuer un tirage aléatoire dans $\mathbb{Z}/q^{2k}\mathbb{Z}$ et de l'élever à la puissance $\varphi(q^k)/p = (q-1)q^{k-1}/p$. Cette méthode fournit bien un élément d'ordre divisant p, c'est-à-dire une racine primitive p-ième de l'unité ou 1 puisque p est premier. Cependant le coût de cette approche est trop élevé pour qu'elle soit utilisée dans un algorithme quasi-linéaire. En effet les nombres manipulés ainsi que l'exposant ont une taille de l'ordre de $k \log q = \log D + \log H$ bits. L'exponentiation dans ce cas est donc quadratique.

Il y a cependant une autre solution pour construire cette racine p-ième de l'unité ω_k qui consiste à exploiter le fait qu'une racine primitive p-ième de l'unité ω soit déjà connue modulo q et peut être remontée modulo q^{2k} par itération de Newton.

Cette approche fonctionne grâce au lemme suivant. Pour simplifier la lecture, les résultats sur ce point de génération d'une racine primitive p-ième de l'unité seront donnés en notant ω_i la racine obtenue modulo q^i. Ce n'est qu'après la partie technique, pour l'algorithme en section 3.2.3, que ω_k désignera à nouveau la racine dont nous avons vraiment besoin, une racine ω_{2k} modulo q^{2k}.

Lemme 3.16. *Soient p et q deux nombres premiers tels que $p \mid (q-1)$ et $k \geq 1$. Si ω_k est une racine primive p-ième de l'unité modulo q^k, alors $\omega_k \bmod q$ l'est aussi modulo q. De plus, ω_k est principale : $\omega_k^i - 1$ n'est pas un diviseur de zéro pour $0 < i < p$.*

Démonstration. Soit g un générateur du groupe multiplicatif $(\mathbb{Z}/q^k\mathbb{Z})^*$, qui est cyclique puisque q^k est la puissance d'un nombre premier. Alors $g \bmod q$ est nécessairement un générateur du plus petit groupe $(\mathbb{Z}/q\mathbb{Z})^*$. En effet, si ce n'était pas le cas l'ensemble $\{g^i \bmod q^k\}_{i\geq 0}$ ne contiendrait pas tous les éléments de $(\mathbb{Z}/q^k\mathbb{Z})^*$ (en particulier les éléments inférieurs à q qui ne sont pas des puissances de g modulo q). Comme g est un

générateur et ω_k une racine primitive p-ième de l'unité modulo q^k, ω_k peut s'écrire sous la forme $g^{i\varphi(q^k)/p}$ pour un entier $i \in \{1, 2, \ldots, p-1\}$. Cette égalité reste vraie modulo q, d'où

$$\omega_k \bmod q = g^{i\varphi(q^k)/p} \bmod q = (g \bmod q)^{i(q-1)/p} \bmod q,$$

puisque $\varphi(q^k) = (q-1)q^{k-1}$ et $a^q \bmod q = a$ pour tout entier a. Comme $g \bmod q$ est un générateur du groupe multiplicatif de $\mathbb{F}_q$, et $1 \leq i \leq p-1$, cela signifie que $\omega_k \bmod q$ est une racine primitive p-ième de l'unité modulo q.

Pour le second point, puisque $\omega_k \bmod q$ est une racine primitive p-ième de l'unité modulo q, $(\omega_k^i - 1) \bmod q$ est différent de zéro pour $0 < i < p$. Or les diviseurs de zéro modulo q^k sont des multiples de q, q étant premier, donc égaux à zéro modulo q. Ce qui permet de conclure que ω_k est bien principale. □

Ce que dit le lemme 3.16, c'est qu'il y a une bijection entre les racines primitives p-ièmes de l'unité dans $\mathbb{F}_q$ et celles dans $\mathbb{Z}/q^k\mathbb{Z}$, et que cette bijection correspond à une égalité si toutes les racines sont regardées modulo q.

À partir d'une racine primitive p-ième de l'unité ω modulo q, son équivalent ω_k modulo q^k peut se contruire par itération de Newton en résolvant l'équation $\omega_k^p - 1 = 0$ modulo des puissances de plus en plus grandes de q. Pour passer de $\omega_i = \omega_k \bmod q^i$ à $\omega_{2i} = \omega_k \bmod q^{2i}$, considérons l'écriture de ω_{2i} en base q^i : $\omega_{2i} = \omega_i + aq^i$, avec $a < q^i$. La résolution de l'équation modulaire $\omega_{2i}^p \bmod q^{2i} = 1$ donne la valeur de a :

$$a = \left(\frac{1 - \omega_i^p \bmod q^{2i}}{q^i}\right) \omega_i p^{-1} \bmod q^i.$$

Dans cette écriture, la fraction correspond en fait à une division entre deux entiers et l'inversion de p est effectuée modulo q^i. Ces considérations assurent la validité de l'algorithme 11 qui suit exactement cette démarche pour trouver ω_k.

Algorithme 11 RemontéeDeRacine

Entrée : p, q des entiers tels que $p \,|\, (q-1)$, $\omega \in \mathbb{F}_q$ une racine primitive p-ième de l'unité et un entier $k \geq 1$.

Sortie : ω_k, une racine primitive p-ième de l'unité modulo q^k.

1: $i \leftarrow 1$; $\omega_1 \leftarrow \omega$
2: **Tant que** $i < k$ **faire**
3: $\quad a \leftarrow \omega_i^p \bmod q^{2i}$
4: $\quad a' \leftarrow (1-a)/q^i$ par une division sur $\mathbb{Z}$
5: $\quad a'' \leftarrow a'\omega_i p^{-1} \bmod q^i$
6: $\quad \omega_{2i} \leftarrow \omega_i + a''q^i$
7: $\quad i \leftarrow 2i$
8: **Renvoyer** $\omega_i \bmod q^k$

Théorème 3.17. *Pour p, q des entiers tels que $p \,|\, (q-1)$, $\omega \in \mathbb{F}_q$ une racine primitive p-ième de l'unité et un entier $k \geq 1$, l'algorithme 11 renvoie bien une racine primitive p-ième de l'unité modulo q^k. Sa complexité binaire est $\mathcal{O}(\log p \mathsf{I}(\log q^k) \log k)$.*

Démonstration. La preuve de la validité de l'algorithme est donnée dans la discussion préalable. Il reste à évaluer sa complexité. La boucle est répétée $O(\log k)$ fois. Son étape la plus coûteuse est l'étape 3 qui effectue une exponentiation modulo q^{2i} en $\mathcal{O}(\log p|(\log q^{2i})\log p)$ opérations binaires. C'est lors de la dernière répétition avec $k \geq 2i$ que cette étape a son coût maximal. □

Ainsi, nous avons tous les éléments pour passer d'un anneau à l'autre et travailler avec des nombres d'une taille correspondant juste à celle permettant de rester quasi-linéaire.

3.2.3. Description et analyse de l'algorithme d'interpolation

L'algorithme INTERPOLATION_BNM (le sigle BNM signifiant Boîte Noire Modulaire) décrit dans l'algorithme 12 suit les idées exposées en section 3.2.1 pour interpoler un polynôme creux F à coefficients entiers donné par une boîte noire modulaire dans un temps quasi-linéaire dans la taille $T(\log D + \log H)$ du polynôme.

Algorithme 12 INTERPOLATION_BNM

Entrée : Une boîte noire modulaire représentant un polynôme $F \in \mathbb{Z}[x]$ et des bornes D, T et H sur son degré, le nombre de ses monômes et la taille de ses coefficients respectivement.

Sortie : La représentation creuse de $F \in \mathbb{Z}[x]$ avec une probabilité d'au moins $\frac{2}{3}$, sinon un autre polynôme respectant les mêmes bornes ou ÉCHEC

1: $F^* \leftarrow 0$; $\epsilon \leftarrow 1/(9\lceil \log T \rceil)$
2: $\lambda \leftarrow \max\left(\frac{2^{58}}{\epsilon^2}, \frac{5}{\epsilon}(T-1)\ln D, \sqrt[5]{\frac{48}{\epsilon}T\ln 2H}\right)$
3: **Tant que** $T \geq 1$ **faire**
4: $(p, q, \omega) \leftarrow$ TRIPLET(λ, ϵ) pour avoir une racine primitive p-ième de l'unité $\omega \in \mathbb{F}_q$, p, q premiers, $\lambda < p < 2\lambda$ et $q = \text{poly}(p)$ ▷ lemme 3.14
5: **Si** L'exécution de TRIPLET échoue **alors Renvoyer échec**
6: Évaluer $(F - F^*)$ sur 1, ω, …, ω^{2T-1} et calculer les exposants de $(F - F^*) \bmod \langle x^p - 1, q\rangle$ ▷ faits 3.3 et 3.5
7: Calculer une racine primitive p-ième de l'unité modulo q^{2k} ω_k avec $k = \lceil \max(\frac{1}{2}\log_q 4H, \log_q D)\rceil$ ▷ théorème 3.17
8: Évaluer $(F - F^*)$ sur 1, ω_k, …, ω_k^{T-1} et calculer la représentation creuse de $(F - F^*) \bmod \langle x^p - 1, q^{2k}\rangle$ ▷ fait 3.4
9: Procéder de même en multipliant les points d'évaluation par $(1 + q^k)$ pour $(F - F^*)((1 + q^k)x) \bmod \langle x^p - 1, q^{2k}\rangle$
10: Calculer de potentiels monômes de $(F - F^*)$ ▷ algorithme 10
11: Ajouter ces monômes à F^* ; $T \leftarrow \lfloor T/2 \rfloor$
12: **Renvoyer** F^*

Théorème 3.18. *Soit $F \in \mathbb{Z}[x]$ polynôme décrit par une boîte noire modulaire et des bornes D, T et H sur son degré, le nombre de ses monômes et la hauteur de ses coefficients, l'algorithme* INTERPOLATION_BNM *permet de reconstruire la représentation*

creuse de F avec une probabilité de succès supérieure à $\frac{2}{3}$. Il utilise $\mathcal{O}(T)$ évaluations de la boîte noire, et réalise $\tilde{\mathcal{O}}(T(\log D + \log H))$ opérations binaires.

L'algorithme effectue plus précisément $\mathcal{O}(n \log^3 n \log^2 T (\log\log n)^2)$ opérations binaires, en posant $n = T(\log D + \log H)$.

Pour tout $\rho \geq 1$, répéter l'algorithme $\mathcal{O}(\rho)$ fois et sélectionner le résultat majoritaire permet d'atteindre une probabilité de succès d'au moins $1 - \frac{1}{2^\rho}$.

Démonstration : Validité. La seule étape aléatoire de l'algorithme est celle de la génération du triplet (p, q, ω). D'après le lemme 3.14, elle réussit avec une probabilité d'au moins $1 - 3\epsilon$ à fournir un triplet ayant les propriétés requises. La probabilité d'échec à chaque itération et donc au plus $3\epsilon = \frac{1}{3\lceil \log T \rceil}$. Cette probabilité reste valide pour chaque passage dans la boucle puisque tout au long de l'algorithme, les bornes T, D et H, sur F s'appliquent aussi à F^* et fournissent des bornes T, D et $2H$ sur $F - F^*$. En effet d'une part, sauf échec, le nombre de monômes de $F - F^*$ ne peut que diminuer (la justification de ce point est apportée dans la suite de la preuve) et d'autre part tout monôme ayant un degré ou un coefficient trop grand peut être rejeté comme non valide lors de la reconstruction (étape 11). Comme la boucle est répétée $\log T$ fois (T étant divisé par 2 à chaque fois), la probabilité totale d'échec est au plus $\frac{1}{3}$.

Supposons maintenant qu'il n'y a eu aucun échec lors de la génération des différents triplets (p, q, ω) pour prouver que dans cette situation l'algorithme calcule bien F. Puisque p et q sont premiers et que ω est une racine primitive p-ième de l'unité, les évaluations de $F - F^*$ dans les puissances de ω correspondent à des évaluations de $(F - F^*) \bmod x^p - 1$. Ainsi, les étapes 6, 8 et 9 permettent bien de calculer les exposants de $(F - F^*) \bmod x^p - 1$ d'après les faits 3.3 et 3.5, puis les coefficients de $(F - F^*)(x) \bmod x^p - 1$ et $(F - F^*)((1 + q^k)x) \bmod x^p - 1$ et donc leur représentation creuse d'après le fait 3.4. Enfin, les propriétés des nombres premiers p et q garantissent que $(F - F^*) \bmod x^p - 1$ n'a aucun coefficient divisible par q et est à une distance au plus $\frac{1}{2}t$ de $F - F^*$ avec $t \leq T$ le nombre de monômes de $(F - F^*)$. Ainsi, en appliquant le lemme 3.15 à $F - F^*$, les monômes reconstruits à l'étape 10 forment un polynôme G tel que $F - F^* - G = F - (F^* + G)$ a au plus $\frac{1}{2}t$ monômes. À chaque passage dans la boucle, le nombre de monômes à retrouver est donc divisé par 2 ce qui permet bien de les retrouver tous en $\lceil \log T \rceil$ étapes.

Pour améliorer la probabilité de succès, il suffit de répéter l'algorithme $48\rho / \log e$ fois et de renvoyer le polynôme majoritaire parmi ceux renvoyés lors des différentes exécutions. D'après le fait 1.3, ceci conduit à une probabilité de succès de $1 - \frac{1}{2^\rho}$.

□

Démonstration : Complexité. Chaque boucle fait appel $3T$ fois à la boîte noire modulaire, avec T qui correspond à la valeur de T pour ce passage précis. La valeur de T étant divisée par 2 à la fin de chaque boucle, le nombre total d'appels à la boîte noire est inférieur à $6T$.

La génération du triplet (p, q, ω) est en $poly(\log p \log \frac{1}{\epsilon}) = poly(\log(T \log DH))$ ce qui négligeable par rapport au coût des autres étapes de l'algorithme. Pour pouvoir évaluer $F - F^*$, il convient aussi d'évaluer F^* qui n'est pas donné par la boîte noire. Puisque les points d'évaluations sont systématiquement des racines primitives p-ièmes de l'unité, donc principales, les 3 évaluations multi-points nécessitent $\mathcal{O}(\mathsf{M}(T) \log T + T \log p)$ opérations dans $\mathbb{F}_q$ ou $\mathbb{Z}/q^{2k}\mathbb{Z}$ d'après le fait 3.4. En plus de l'évaluation, l'étape 6 calcule les exposants de $F \bmod x^p - 1$ en passant par le polynôme générateur de la suite des évaluations et la recherche de ses racines par évaluation multi-point. Cette étape nécessite $\mathcal{O}(\mathsf{M}(T) \log T + \mathsf{M}(p))$ opérations dans $\mathbb{F}_q$ (faits 3.3 et 3.5). L'étape 7 génère ω_k en $\mathcal{O}(\log p \mathsf{I}(\log q^{2k}) \log k)$ opérations binaires (théorème 3.17). Les étapes 8 et 9 calculent les coefficients modulo q^{2k} en $\mathcal{O}(\mathsf{M}(T) \log T + T \log p)$ opérations dans $\mathbb{Z}/q^{2k}\mathbb{Z}$ (fait 3.4). Enfin, l'étape 10 calcule les coefficients et exposants d'origine en $\mathcal{O}(T)$ opérations sur des entiers bornés par q^{2k} (lemme 3.15). Toutes ces étapes sont répétées $\log T$ fois.

Le choix de λ donne $p = \mathcal{O}(T(\log D + \log H) \log^2 T) = \mathcal{O}(n \log^2 T)$ en posant $n = T(\log D + \log H)$. On a donc $\log q = \mathcal{O}(\log p) = \mathcal{O}(\log n)$ puisque $q = \text{poly}(p)$. L'ensemble des opérations dans $\mathbb{F}_q$ ont donc un coût binaire total de

$$\mathcal{O}((\mathsf{M}(T) \log T + T \log p + \mathsf{M}(p))\mathsf{I}(\log q) \log T) = \mathcal{O}(n \log^2 n \log^3 T (\log \log n)^2).$$

Par ailleurs $k = \mathcal{O}(\log \max(D, H)) = \mathcal{O}(n)$ et $q^{2k} = \mathcal{O}(\max(D, H))$ d'où $\log q^{2k} = \mathcal{O}(\log(DH)) = \mathcal{O}(n)$, les générations de ω_k à chaque passage dans la boucle s'effectuent donc en

$$\mathcal{O}(\log^2 n \mathsf{I}(n) \log T) = \mathcal{O}(n \log^3 n \log T)$$

opérations binaires. Par ailleurs, l'ensemble des opérations dans $\mathbb{Z}/q^{2k}\mathbb{Z}$ ont un coût binaire total en

$$\mathcal{O}((\mathsf{M}(T) \log T + T \log p)\mathsf{I}(\log q^{2k}) \log T) = \mathcal{O}(T \log(DH) \log^2 n \log^2 T \log \log n).$$

Puisque $T < n = T(\log D + \log H)$, en majorant chacune de ces expressions on obtient la complexité générale qui est de

$$\mathcal{O}(n \log^3 n \log^2 T (\log \log n)^2) = \tilde{\mathcal{O}}(T(\log D + \log H))$$

opérations binaires. □

L'usage du terme $\frac{2^{58}}{\epsilon^2}$ semble nuire à l'efficacité pratique de l'algorithme. Il est cependant nécessaire dans le cadre d'une approche formelle reposant sur des preuves solides. En revanche, les discussions de la section 2.2.2 (page 37) disent qu'en pratique ce terme peut être remplacé par 1. Ceci soit en suivant une approche heuristique par l'observation de la répartition des nombres premiers connus, soit en faisant une hypothèse, plus forte que ce qui est connu mais raisonnable, sur les nombres premiers en progression arithmétique.

L'algorithme est quasi-linéaire dans la taille de F. Pour cela il repose notamment sur l'existence de morphismes naturels permettant de passer d'un polynôme à coefficients dans $\mathbb{Z}$ à un polynôme à coefficient dans n'importe quel anneau. Ce point permet

d'adapter la taille des nombres manipulés en fonction du nombre d'opérations à réaliser pour conserver une complexité quasi-linéaire. Ce point est bien illustré par la répartition des opérations entre les petits corps $\mathbb{F}_q$ et les grands anneaux $\mathbb{Z}/q^{2k}\mathbb{Z}$.

Remarque 3.19. L'algorithme INTERPOLATION_BNM effectue $\tilde{\mathcal{O}}(T\log(DH))$ opérations sur des entiers de taille $\mathcal{O}(\log(T\log(DH)))$ et $\tilde{\mathcal{O}}(T\log\log(DH))$ sur des entiers de taille $\mathcal{O}(\log(DH))$.

L'algorithme est décrit pour une boîte noire modulaire avec des bornes car cela correspond au minimum d'information nécessaire pour qu'il puisse fonctionner. Il est aussi adapté à des modèles donnant plus d'information, notamment si la boîte noire est remplacée par un circuit arithmétique. C'est alors le coût des évaluations du circuit arithmétique sur des points dans le grand anneau qui domine la complexité.

Corollaire 3.20. *Avec un circuit arithmétique de longueur L plutôt qu'une boîte noire, l'algorithme* INTERPOLATION_BNM *a les mêmes garanties de succès. Si H est aussi une borne sur les constantes du circuit arithmétique, il effectue $\tilde{\mathcal{O}}(LT(\log D + \log H))$ opérations binaires.*

La taille du circuit arithmétique donné en entrée est $\mathcal{O}(L\log H)$ et celle de la sortie $\mathcal{O}(T(\log D + \log H))$. L'algorithme n'est donc pas pas quasi-linéaire dans la taille de l'entrée *plus* celle de la sortie. Il s'agit cependant de l'algorithme connu le plus efficace pour un polynôme dans $\mathbb{Z}[x]$ donné par circuit arithmétique.

Dans ce chapitre, nous avons présenté un algorithme d'interpolation de polynômes creux dans $\mathbb{Z}[x]$ de complexité quasi-linéaire. Pour le mettre au point nous nous sommes appuyés sur des techniques d'interpolation creuse dans différents modèles. Nous avons aussi fortement utilisé la souplesse qu'offre le morphisme canonique de $\mathbb{Z}$ vers tout anneau. Ainsi, à chaque instant nous avons pu contrôler la taille des nombres et polynômes manipulés pour que le coût des opérations n'excèdent jamais la taille du résultat.

4. Tester des identités polynomiales

La manipulation de polynômes donnés sous une forme implicite comme une boîte noire, un circuit arithmétique ou le résultat d'un calcul sur des polynômes ne se résume pas simplement à tenter d'obtenir explicitement le polynôme concerné. Le problème peut aussi consister à déterminer si ce polynôme a ou n'a pas une certaine propriété et quelle est la complexité de cette question.

La première question qui vienne à l'esprit est sans doute de savoir si le polynôme est identiquement nul ou non. Lorsque le polynôme est donné par un circuit arithmétique, ce problème connu sous le nom de test d'identité polynomiale (PIT) a suscité beaucoup d'intérêt. Il existe un algorithme probabiliste en temps polynomial et même quasi-linéaire. En revanche la question de savoir s'il existe un algorithme déterministe en temps polynomial reste une question ouverte alors même que la propriété à tester semble plutôt basique. En 2009, Saxena a proposé un inventaire des différents travaux relatifs à ce problème [Sax09 ; Sax14]. Différents cas particuliers y sont détaillés notamment en fonction de la profondeur du circuit ou du caractère creux du polynôme concerné. Ce problème peut être transformé en posant plutôt la question : étant donné n, le polynôme s'annule-t-il sur une racine primitive n-ième de l'unité ? La classification exacte de ce problème plus restreint est une question toujours actuelle [Bal+21]. Ce problème modifié consiste en fait à savoir si le n-ième polynôme cyclotomique divise le polynôme représenté par le circuit. Ce problème concentre donc deux aspects délicats de la manipulation de polynômes creux : la division et les polynômes cyclotomiques.

La division de polynômes creux est problématique car elle risque fortement d'avoir un résultat qui n'est absolument pas creux. Ainsi pour répondre à une question sur une division de polynôme creux il est en général prohibé de passer par le calcul du quotient et du reste. Il est possible de définir une variante du PIT liée à la division par un polynôme quelconque. Un tel problème de PIT modulaire peut se représenter comme le test d'une égalité de la forme $\sum_i \prod_j \sum_k \cdots \prod_\ell F_{i,j,k,\ldots,\ell} \bmod P = 0$. Si tous les polynômes impliqués sont denses, une solution polynomiale est le calcul direct du polynôme représenté à gauche. L'objectif face à un tel problème est donc la recherche de la solution la plus efficace, si possible en temps quasi-linéaire. En revanche, si les polynômes impliqués sont creux la seule borne sur le nombre de monôme du polynôme est le degré de P. Son calcul ne peut s'effectuer qu'en temps exponentiel. Aucune solution en temps polynomial, même probabiliste n'est connue si l'on ajoute cette contrainte de la division par un polynôme creux.

Cette complexité liée à la division par des polynômes creux est théoriquement appuyée

par les travaux de Plaisted [Pla77 ; Pla84] qui listent des problèmes NP-difficiles ou NP-complets pour des polynômes creux. Un grand nombre de ces problèmes concernent des questions de divisibilité, de pgcd, des questions sur les racines, le quotient ou le reste. Une forte particularité de ses travaux réside aussi dans l'utilisation récurrente de familles de polynômes cyclotomiques dans les preuves de complexité. En particulier les problèmes liés aux racines concernent spécifiquement le rapport entre les racines du polynôme et les racines de l'unité. Les polynômes cyclotomiques semblent tellement porteurs des principales difficultés rencontrées lors de la manipulation de polynômes creux que Carette et Davenport ont proposé une représentation des polynômes qui stockent tous les facteurs cyclotomiques [CD10].

Ce chapitre est dédié à des tests d'identités polynomiales principalement pour des cas particuliers de PIT modulaire.

Tout d'abord, en section 4.1, le problème est considéré dans sa version la plus simple, $F \bmod P = 0$. Autrement dit il s'agit simplement de déterminer si P divise F ou non. L'existence ou l'absence d'un algorithme en temps polynomial pour résoudre ce problème reste une question ouverte. En revanche dans cette section une solution polynomiale est apportée pour une large famille de polynômes. Dans un second temps, en section 4.2, le problème est étudiée pour des identités un peu plus générales, de la forme $\sum_i F_i G_i \bmod P = 0$ avec P creux et unitaire, les F_i et G_i soit denses soit creux mais de degré toujours borné. Cette borne sur le degré et la structure considérée pour P simplifie la résolution générale de ce problème. Cette section se consacre donc plutôt sur la mise en place d'algorithmes aussi efficaces que possible pour cette résolution. Pour obtenir les meilleures complexités, nous mettons au point des algorithmes probabilistes et non déterministes. Enfin, en section 4.3, notre intérêt se porte sur des identités non modulaires, de la forme $\sum_i F_i G_i = 0$. Il s'agit alors d'être à nouveau aussi efficace que possible en s'appuyant notamment sur la vérification probabiliste modulo un polynôme P de la forme $x^d - 1$.

Dans ces deux dernières parties, il s'agit en fait de vérifier des produits, modulaires ou non. La vérification en pratique peut correspondre à deux situations. Soit un manque de confiance dans un calcul effectué par un tiers, soit le besoin de tester des implémentations complexes d'algorithmes. Dans ces deux situations les besoins ne sont pas exactement les mêmes. Pour la première, un algorithme plus performant est nécessaire, sinon il est tout aussi simple d'effectuer directement le calcul soi-même sans le déléguer à un tiers. Dans le second cas, l'algorithme doit être plus fiable, c'est à dire qu'on doit avoir une plus grande confiance dans l'implémentation que l'on peut réaliser de l'algortihme de test que dans l'implémentation de l'algorithme effectuant le calcul. Pour cela, les opérations effectuées doivent être aussi simples que possible. Les algorithmes de ces sections tendront à répondre à au moins l'une de ces exigences et seront analysés dans ce sens.

4.1. Divisibilité entre deux polynômes creux

Cette section se concentre sur l'identité polynomiale $F \bmod P = 0$ pour F et P deux polynômes creux et reprend des résultats publiés en 2021 [GGP21]. Il s'agit donc de déterminer si P divise F ou non. C'est un cas très restreint des problèmes P1 (sur le pgcd) et P2 (sur le facteur d'un produit) qui ont été prouvés NP-difficiles par David Plaisted dans son article de 1977 [Pla77].

Une approche naturelle serait de simplement calculer la division euclidienne de F par P pour savoir si le reste est nul ou pas [Joh74], ou plus directement de calculer le reste sans même calculer le quotient. Le problème dans le cas de polynômes creux c'est que de tels calculs peuvent être intrinsèquement exponentiels à cause de l'augmentation du nombre de monômes des polynômes quotient et reste, comme cela est illustré par l'exemple 4.1.

Exemple 4.1. Si $F = x^{2d+1} + x^{2d}$ et $P = x^{d+1} - x^d + 1$, les deux polynômes ont un nombre constant de monômes. Leur division euclidienne $F = PQ + R$ produit deux polynômes $Q = x^d + 2\sum_{i=0}^{d-1} x^i$ et $R = x^d - 2\sum_{i=0}^{d-1} x^i$ ayant chacun $d+1$ monômes.

Le calcul de la division euclidienne s'avère dans certains cas exponentiel. Étant donné deux polynômes creux F et $P \in \mathbb{A}[x]$, l'objectif, plus restreint, est donc de vérifier si P divise F en temps polynomial. En prenant T le nombre maximal de monômes de F ou de P, $\deg(P) = d_p$, $\deg(F) = d_p + d - 1$, la taille des polynômes considérés est $\mathcal{O}(T \log(d + d_p))$ plus $\mathcal{O}(T)$ fois la taille des éléments de $\mathbb{A}$. Les opérations de base entre éléments d'un anneau effectif étant généralement déjà polynomiales dans leur taille binaire, leur coût n'aura pas d'impact négatif sur l'objectif de complexité visé et sera par conséquent négligé. Ainsi, le but est d'avoir un test de divisibilité en $(T \log(d + d_p))^{\mathcal{O}(1)}$.

L'existence ou l'absence d'un tel test pour toute paire de polynômes F et P reste une question ouverte, formulée explicitement par Plaisted en 1984 [Pla84]. Il est en revanche connu que, sous l'hypothèse de Riemann étendue, ce problème est dans co-NP [GKO92]. La preuve repose sur l'existence, en cas de non-divisibilité, d'un élément qui est racine de P avec un ordre supérieur que celui qu'il a comme racine de F.

Des solutions existent cependant déjà pour des cas particuliers.

Fait 4.2. *Soient F, $P \in \mathbb{A}[x]$ de degrés $d_p + d - 1$ et d_p respectivement, et ayant au plus T monômes. Si d ou d_p est polynomial en $T(\log d_p + d)$, il est possible de tester si P divise F en temps polynomial.*

Démonstration. Soit $F = PQ + R$ la division euclidienne de F par P. Si d admet une borne polynomiale, Q et R peuvent être calculés en temps polynomial par l'algorithme de division euclidienne, la vérification étant alors triviale. Si d_p admet une borne polynomiale, le degré de R étant inférieur à d_p, il peut être calculé en temps polynomial sans calculer Q. En effet, il suffit de calculer $x^e \bmod P$ par exponentiation rapide pour chaque exposant $e \in \text{support}(F)$. Les polynômes manipulés sont denses mais de degré inférieur à d_p, le calcul s'effectue bien en temps polynomial. □

Le reste de cette section peut être vu comme une généralisation de ce fait : s'il existe une borne polynomiale sur la taille du quotient alors il est possible de tester la divisibilité en temps polynomial soit en calculant $F - PQ$ soit en garantissant que $F = PQ$.

Des bornes sur la taille du quotient sont données en section 4.1.1. Elles sont utilisées en section 4.1.2 pour fournir un test de divisibilité en temps polynomial dans des cas plus généraux que ceux présentés dans le fait 4.2.

Pour parvenir à donner des bornes et un test le plus général possible, la méthode présentée s'appuie beaucoup sur les polynômes réciproques et les liens entre la divisibilité des polynômes et celle de leur réciproques. La remarque suivante sera donc particulièrement utile.

Remarque 4.3. Soit F, P deux polynômes dans $\mathbb{A}[x]$, et $F^\star = x^{\deg F} F(1/x)$, $P^\star = x^{\deg P} P(1/x)$ les polynômes réciproques. Le polynôme P divise le polynôme F si et seulement si $P^\star$ divise $F^\star$. Dans ce cas, le quotient $Q^\star = F^\star \operatorname{quo} P^\star$ est le polynôme réciproque du quotient $Q = F \operatorname{quo} P$.

Démonstration. En appliquant la définition, $(PQ)^\star = P^\star Q^\star$ pour tout polynôme P, $Q \in \mathbb{A}[x]$. Ainsi, s'il existe un polynôme Q tel que $F = PQ$, alors $F^\star = P^\star Q^\star$. La réciproque vient du caractère involutif de l'application qui a un polynôme associe son polynôme réciproque. □

Il faut cependant bien noter que l'égalité entre les quotients $(F \operatorname{quo} P)^\star = F^\star \operatorname{quo} P^\star$ n'est pas vraie en général si P ne divise pas F. Si $F = PQ + R$ avec $\deg(R) < \deg(P)$ alors $F^\star = x^{\deg(F)} P(1/x) Q(1/x) + x^{\deg(F)} R(1/x) = P^\star Q^\star + x^{\deg(F)} R(1/x)$. Ainsi $Q^\star = F^\star \operatorname{quo} P^\star$ si et seulement cette expression est une division euclidienne, c'est à dire que $\deg(x^{\deg(F)} R(1/x)) < \deg(P^\star)$.

4.1.1. Borner le nombre de monômes d'un quotient

Les bornes sur le nombre de monômes du quotient $Q = F \operatorname{quo} P$ données dans cette section dépendent de la structure de P. C'est à dire de l'*écart* qu'il y a soit entre les deux plus petits des exposants de P, soit entre les deux plus grands exposants de P.

Une première étape est de borner le nombre de monômes de petit degré de l'inverse de P en tant que série formelle. Si nous nous intéressons à cette série formelle, c'est qu'elle intervient dans la division euclidienne par $P^\star$.

Lemme 4.4. *Soit $P \in \mathbb{A}[x]$ de degré d_p avec $\#P$ monômes, et tel que $P = 1 - x^k P_1$ avec $P_1 \in \mathbb{A}[x]$ de degré $d_p - k$. Pour tout $t \geq 0$, $1/P \bmod x^{kt}$ a au plus $\frac{1}{(t-1)!}(\#P + t - 2)^{t-1}$ monômes non nul.*

Démonstration. Puisque $P(0) = 1$, P est inversible dans l'anneau des séries formelles. Notons cette inverse $\phi = \sum_{i \geq 0} x^{ki} P_1^i \in \mathbb{A}[[x]]$. Dès que $i \geq t$, $x^{ki} \bmod x^{kt} = 0$. Par

conséquent

$$\phi \bmod x^{kt} = \sum_{i=0}^{t-1} x^{ki} P_1^i \bmod x^{kt}.$$

Or le support de P^{t-1} est un sous-ensemble de l'ensemble $S = \{\sum_{j=1}^{t-1} e_j : e_j \in \text{support}(P)\}$ qui a au plus $\binom{\#P+t-2}{t-1} \leq \frac{1}{(t-1)!}(\#P + t - 2)^{t-1}$ éléments. En s'appuyant sur le développement de $P^{t-1} = (1 - x^k P_1)^{t-1} = \sum_{i=0}^{t-1} \binom{t-1}{i}(-1)^i x^{ki} P_1^i$, il est aisé de constater que le support de $\phi \bmod x^{kt}$ est lui aussi inclus dans S qui est le support maximal possible de P^{t-1}, si la somme n'entraîne pas d'annulations. Ceci nous donne une borne sur le nombre de monômes de $1/P \bmod x^{kt}$. □

À partir de ce lemme, il est possible de déduire une borne sur le nombre de monômes du quotient, borne qui dépend de la structure de P.

Corollaire 4.5. *Soit F et $P \in \mathbb{A}[x]$ de degré respectivement $d + d_p - 1$ et d_p et ayant respectivement $\#F$ et $\#P$ monômes. Si $P = x^{d_p} - P_0$ avec $P_0 \in \mathbb{A}[x]$ de degré $d_p - k$, alors le quotient $Q = F \operatorname{quo} P$ a au plus $\frac{1}{(\lceil d/k \rceil - 1)!}\#F(\#P + \lceil d/k \rceil - 2)^{\lceil d/k \rceil - 1}$ monômes non nuls.*

Démonstration. Soit $F = PQ + R$ avec $\deg(R) < d_p$, la division euclidienne de F par P. Il est bien connu que le polynôme réciproque $Q^\star$ de Q est égal à $F^\star/P^\star \bmod x^d$ [GG13]. Le lemme 4.4 peut être appliqué à $P^\star$ puisque $P^\star = 1 - x^k P_0^\star$. Ainsi, $1/P^\star \bmod x^d$ a au plus $\frac{1}{(\lceil d/k \rceil - 1)!}(\#P - \lceil d/k \rceil - 2)^{\lceil d/k \rceil - 1}$ monômes non nuls, en prenant $t = \lceil d/k \rceil$ ce qui donne $d \leq kt$. Pour retrouver $Q^\star$, il ne reste plus qu'à multiplier par $F^\star$ et supprimer les monômes de trop haut degré. En considérant seulement la multiplication, ceci implique que le nombre de monômes non nuls de $Q^\star$, qui est égal à celui de Q, est d'au plus $\frac{1}{(\lceil d/k \rceil - 1)!}\#F(\#P + \lceil d/k \rceil - 2)^{\lceil d/k \rceil - 1}$. □

Si au contraire, l'écart considéré est celui entre les deux plus grands exposants de P, pour atteindre la même borne il faut considérer le cas où il y a bien divisibilité.

Corollaire 4.6. *Soit F, $P \in \mathbb{A}[x]$ de degré respectivement $d + d_p - 1$ et d_p et ayant respectivement $\#F$ et $\#P$ monômes. Si $P = 1 - x^k P_1$ et P divise F, alors le quotient $Q = F \operatorname{quo} P$ a au plus $\frac{1}{(\lceil d/k \rceil - 1)!}\#F(\#P + \lceil d/k \rceil - 2)^{\lceil d/k \rceil - 1}$ monômes non nuls.*

Démonstration. Il s'agit d'une application du corollaire 4.5 à $F^\star$ et $P^\star$. En effet, on peut écrire $P^\star = x^{d_p} - P_0$ pour un polynôme P_0 de degré $d_p - k$ et, puisque P divise F, $F \operatorname{quo} P = (F^\star \operatorname{quo} P^\star)^\star$. □

La divisibilité de F par P est bien nécessaire pour obtenir cette borne. En son absence il n'y a pas de borne polynomiale sur le nombre de monômes non nuls du quotient comme le montre l'exemple suivant.

Exemple 4.7. Soit $F = x^{d+d_p-1} - 1$ et $P = x^{d_p} - x^{d_p-1} + 1$. Alors $F \operatorname{quo} P = \sum_{i=0}^{n-1} x^i$ a autant de monômes que possible, à savoir d.

En l'absence de divisibilité, il est possible de trouver une borne qui dépendra alors aussi du nombre de monômes non nuls du reste R de la division euclidienne $F = PQ + R$. En effet, la divisibilité se retrouve alors mais avec le polynôme $F - R = PQ$. Lors d'une division par un polynôme de la forme $P = 1 + x^k P_1$, le nombre de monôme non nuls du quotient Q est donc relié au nombre de monômes des trois autres polynômes par une borne polynomiale. Ceci n'est pas vrai pour un polynôme P quelconque. Cependant, cette borne sur $\#Q$ par rapport à $\#(F - R)$ ne donne pas d'information supplémentaire puisqu'il n'y a pas de bornes sur $\#R$.

4.1.2. Tests de divisibilité

Pour des polynômes F et $P \in \mathbb{A}[x]$ de degré respectivement $d_p + d - 1$ et d_p, si une condition supplémentaire $d = \mathcal{O}(d_p)$ est ajoutée, les résultats de la section précédente donnent déjà un moyen d'effectuer un test de divisibilité en temps polynomial si P a la structure appropriée. Si $P = x^{d_p} - P_0$ avec $\deg(P_0) \leq d_p - k$ pour un $k = \mathcal{O}(d_p)$, le nombre de monômes du quotient $F \operatorname{quo} P$ admet une borne polynomiale dans la taille des entrées. Si $P = 1 + x^k P_1$, c'est le quotient dans la division des réciproques $F^\star \operatorname{quo} P^\star$ qui admet alors une telle borne. Dans les deux cas, les divisions euclidiennes correspondantes peuvent être effectuées en temps polynomial ce qui permet de tester si les restes correspondants sont nuls et donc si P divise F. Cette approche peut-être étendue à une famille plus large de diviseurs P en s'appuyant sur une généralisation du lemme 4.4 à un polynôme P de la forme $P_0 - x^k P_1$ donnée dans le lemme suivant.

Lemme 4.8. *Soit $P \in \mathbb{A}[x]$ un polynôme de degré d_p ayant $\#P$ monômes non nuls et tel que $P = P_0 - x^k P_1$ avec $\deg(P_0) < k$ et $P(0) \neq 0$. Pour tout entier t, $P_0^t / P \bmod x^{tk}$ a au plus $\frac{1}{(t-1)!}(\#P + t - 2)^{t-1}$ monômes non nuls.*

Démonstration. Puisque $P(0) \neq 0$, P_0 et P sont inversibles dans l'anneau des séries formelles. La division dans $\mathbb{A}[[x]]$ de P_0 par P donne $P_0/P = 1/(1 - x^k P_1 P_0^{-1}) = \sum_{i \geq 0} x^{ki} P_1^i P_0^{-i}$. Ainsi, pour tout entier t, $P_0^t/P = \sum_{i \geq 0} x^{ki} P_1^i P_0^{t-i-1}$. Puisque $x^{ki} \bmod x^{kt} = 0$ pour $i \geq t$,

$$P_0^t / P \bmod x^{kt} = \sum_{i=0}^{t-1} x^{ki} P_1^i P_0^{t-1-i} \bmod x^{kt}.$$

Le support du polynôme $P_0^t / P \bmod x^{kt}$ est alors lui-aussi un sous-ensemble de l'ensemble S défini dans la preuve du lemme 4.4 puisque $P^{t-1} = \sum_{i=0}^{t-1} \binom{t-1}{i} (-1)^i x^{ki} P_1^i P_0^{t-1-i}$. Par conséquent, son nombre de monômes non nuls est d'au plus $\frac{1}{(t-1)!}(\#P + t - 2)^{t-1}$. □

Ce lemme ne regarde plus uniquement une division par P, mais une division par P associée à la multiplication par un autre polynôme P_0^t. Pour pouvoir l'utiliser, il faut donc s'assurer du comportement de la propriété de divisibilité lorsqu'un produit est effectué.

Fait 4.9. *Soit F, P et $C \in \mathbb{A}[x]$, $C \neq 0$. Le polynôme P divise F si et seulement si P divise FC et C divise FC/P.*

Démonstration. Si P divise F, alors clairement P divise FC. Par ailleurs, on peut écrire $F = PQ_1$ et il est tout aussi clair que C divise $Q_1C = FC/P$.

Si P divise FC et C divise FC/P, on peut écrire $FC/P = CQ_2$. Ainsi $F = PQ_2$ et P divise F. □

En s'appuyant sur ce fait, il est possible de proposer un algorithme pour tester la divisibilité par un polynôme P de la forme $P_0 - x^k P_1$. La démarche consiste à regarder d'une part si P divise FP_0^t pour un certain t et d'autre part si P_0^t divise FP_0^t/P en calculant explicitement pour la première le quotient et pour la seconde seulement le reste. L'efficacité de l'algorithme dépend des bornes qui peuvent être apportées sur le nombre de monômes du quotient et du reste en question.

Algorithme 13 TestDivisibilité

Entrée : F, P des polynômes dans $\mathbb{A}[x]$ avec $P = P_0 - x^k P_1$ et $P_0(0) \neq 0$
Sortie : Vrai si P divise F, faux sinon

1: $t \leftarrow \left\lceil \frac{\deg(P)}{k - \deg(P_0)} \right\rceil$
2: Calculer la division euclidienne $(FP_0^t)^\star = P^\star Q_0^\star + R_0^\star$
3: **Si** $R_0^\star \neq 0$ **alors**
4: **Renvoyer faux**
5: Calculer R_1, le reste de la division euclidienne de $Q_0 = (Q_0^\star)^\star$ par P_0^t en calculant $qx^e \bmod P_0^t$ pour tout monôme qx^e de Q_0.
6: **Si** $R_1 \neq 0$ **alors**
7: **Renvoyer faux**
8: **Sinon**
9: **Renvoyer vrai**

Théorème 4.10. *Soit F et $P \in \mathbb{A}[x]$ deux polynômes creux de degré respectivement $d_p + d - 1$ et d_p et ayant chacun au plus T monômes non nuls. Il est possible de tester si P divise F en temps polynomial en $T(\log d_p + \log d)$ si $P = P_0 - x^k P_1$ avec $k - \deg(P_0) = \Omega(d)$ et si soit $\deg(P_0)$ soit $\deg(P_1)$ admet une borne polynomiale en $T(\log d_p + \log d)$.*

Démonstration. Seul le cas où $\deg(P_0) = (T(\log d_p + \log d))^{\mathcal{O}(1)}$ est traité, le second cas se ramenant à celui-ci en considérant la division entre les polynômes réciproques $F^\star$ et $P^\star$. Si $P_0(0) = 0$ cela signifie qu'une certaine puissance de x divise P. Or si x^a divise P, P ne peut diviser F que dans le cas où x^a divise aussi F, c'est à dire si a est inférieur au plus petit exposant de F. Il faut alors en plus que P/x^a divise F/x^a. En comparant le plus petit exposant de P et de F, il est possible d'écarter un cas trivial de non-divisibilité. Si la divisibilité est encore possible, il suffit d'ôter a à tous les exposants de F et de P pour se ramener au cas où $P(0) \neq 0$.

Le test est alors effectué en suivant l'algorithme 13 TestDivisibilité. La validité de l'algorithme est donnée par le fait 4.9 puisque qu'il teste qu'à la fois P divise $P_0^t F$ et

P_0^t divise $Q_0 = FP_0^t/P$. Le premier test est effectué sur les polynômes réciproques en application de la remarque 4.3.

Il reste à vérifier qu'il s'effectue bien en temps polynomial en $T(\log d_p + \log d)$. Par le lemme 4.8, $P_0^t/P \bmod x^{kt}$ a au plus $\frac{1}{(t-1)!}(T+t-2)^{t-1}$ monômes non nuls, d'où le fait que $FP_0^t/P \bmod x^{kt}$ en a au plus $\frac{1}{(t-1)!}(T+t-2)^{t-1}T$. Il faut noter que le choix de t entraîne que $t = \mathcal{O}(1)$ puisque $k - \deg(P_0) = \Omega(d)$ par hypothèse, et aussi que $kt \geq d + \deg(P_0)t$. Ainsi le nombre de monômes non nuls de $Q_0^\star = ((FP_0^t)^\star \operatorname{quo} P^\star)^\star = FP_0^t/P \bmod x^{d+\deg(P_0)t}$ est borné par celui de $FP_0^t/P \bmod x^{kt}$. Il est donc polynomial en T. Par conséquent, le calcul de $Q_0^\star$ et de $R_0^\star$ peut bien s'effectuer en temps polynomial. Si $R_0^\star = 0$, l'algorithme calcul alors R_1, le reste dans la division euclidienne de $Q_0 = (Q_0^\star)^\star$ par P_0^t. Le nombre de monômes de Q_0 est $T^{\mathcal{O}(1)}$ et $\deg(P_0^t) = t\deg(P_0)$ ce qui est polynomial en $T(\log d_p + \log d)$ d'après l'hypothèse sur $\deg(P_0)$ et le fait que $t = \mathcal{O}(1)$. Ainsi le test de divisibilité correspond au cas de base vu dans le fait 4.2 et peut s'effectuer en temps polynomial en réduisant tous les monômes de Q_0 modulo P_0^t. □

La preuve qui vient d'être apportée pour le théorème 4.10 s'étend à des tests pour une famille plus large de diviseurs. Tout ce qui est requis c'est une borne sur le nombre de monômes non nuls de Q_0 et un algorithme pour tester si P_0^t divise Q_0. Ce test pourrait s'effectuer de manière récursive et en temps polynomial si P_0^t est lui-même un polynôme respectant les conditions du théorème. Cette approche conduit à la généralisation suivante du théorème.

Corollaire 4.11. *Soit F et $P \in \mathbb{A}[x]$ deux polynômes creux de degré respectivement $d_p + d - 1$ et d_p et ayant chacun au plus T monômes non nuls. Il est possible de tester si P divise F en temps polynomial en $T(\log d_p + \log d)$ si $P = P_0 + x^k P_1 - x^\ell P_2$ avec P_0, P_1, $P_2 \in \mathbb{A}[x]$ tels que $k - \deg(P_0) = \Omega(d)$, $\ell - k - \deg(P_1) = \Omega(d)$ et $\deg(P_1) = (T(\log d_p + d))^{\mathcal{O}(1)}$.*

Démonstration. Comme dans la preuve du théorème 4.10, il est possible de se restreindre au cas $P(0) \neq 0$, en divisant par la plus grande puissance de x qui divise P si cette puissance divise aussi F, sinon il n'y a pas divisibilité.

Il est aussi possible de se restreindre au cas où $T(\log d_p + \log d) = d^{o(1)}$ puisque sinon il suffit de s'appuyer sur le fait 4.2 pour avoir un algorithme polynomial.

D'après le lemme 4.8, $F(P_0 + x^k P_1)^t/P \bmod x^{kt}$ a au plus $T^{\mathcal{O}(1)}$ monômes non nuls pour un entier $t = \mathcal{O}(1)$. Ainsi, comme vu précédemment, il est possible de calculer le quotient et le reste de la division euclidienne de $(F(P_0 + x^k P_1)^t)^\star$ par $P^\star$ en temps polynomial pour $t = \lceil d/(\ell - k - \deg(P_1)) \rceil = \mathcal{O}(1)$. Si le reste est non nul, P ne divise pas F. Sinon, en considérant les réciproques le calcul a fourni un polynôme Q_{01} tel que $F(P_0 + x^k P_1)^t = Q_{01}P$. Il reste à tester si $\tilde{P} = (P_0 + x^k P_1)^t$ divise Q_{01}. Or le polynôme $\tilde{P}$ satisfait aux conditions du théorème 4.10. En effet,

$$\tilde{P} = \sum_{i=0}^{t} \binom{t}{i} x^{ki} P_1^i P_0^{t-i} = x^{kt} P_1^t + \sum_{i=0}^{t-1} \binom{t}{i} x^{ki} P_1^i P_0^{t-i} = x^{kt} P_1^t + \tilde{P}_0$$

avec $\tilde{P}_0$ de degré au plus $k(t-1) + \deg(P_1)(t-1) + \deg(P_0)$. Ainsi $kt - \deg(\tilde{P}_0) \geq k - \deg(P_0) - (t-1)\deg(P_1) = \Omega(d)$ puisque $k - \deg(P_0) = \Omega(d)$ et $\deg(P_1) = (T(\log d_p + d))^{\mathcal{O}(1)} = d^{o(1)}$. Il est donc possible de tester si $\tilde{P}$ divise Q_{01} en temps polynomial d'après le théorème 4.10. □

En calculant le quotient d'une division différente, le théorème 4.10 et le corollaire 4.11 permettent de répondre à la question de la divisibilité de F par P dans des cas où le quotient de la division de F par P et celui de la division de $F^\star$ par $P^\star$ ont un nombre de monômes exponentiels comme c'est le cas dans l'exemple suivant.

Exemple 4.12. Soit $F = x^{2d-1} - x^d - x^{d-1} + x^2 - x + 1$ et $P = P_0 + x^{d-1}P_1$ avec $P_0 = P_1 = 1 - x$. Le polynôme P vérifie bien les hypothèses du théorème 4.10, $F \operatorname{quo} P = \sum_{i=0}^{d-1} x^i$ et $F^\star \operatorname{quo} P^\star = x^{d-1} + \sum_{i=0}^{d-3} x^i$.

4.1.3. Application aux polynômes à coefficients entiers

Jusqu'à présent les opérations dans l'anneau ont été négligées car pour celles qui sont utilisées (addition, multiplication et division) elles sont généralement polynomiales dans la taille des éléments de l'anneau. Ceci est vrai pour les opérations dans $\mathbb{Z}$, en revanche la taille des éléments considérés peut croître et alors ne plus être polynomiale dans la taille des polynômes donnés en entrée.

Dans l'algorithme 13 TestDivisibilité qui effectue le test de divisibilité il convient donc de regarder la valeur maximale que pourrait atteindre les coefficients de Q_0 et R_1, celle-ci fournissant une borne sur la valeur des entiers à manipuler. La seule opération qui peut entraîner une croissance non polynomiale des coefficients est la division euclidienne comme cela a été vu en section 1.4.

Une borne spécifique à la division exacte, avec reste nul, a même été fournie.

Fait 4.13 (lemme 1.20, page 19)**.** *Soit F, P, $Q \in \mathbb{Z}[x]$ trois polynômes creux. Si $F = QP$, alors*

$$\|Q\| \leq (\|P\| + 1)^{\left\lceil \frac{\#Q-1}{2} \right\rceil} \|F\| .$$

La preuve du lemme 1.20 passe par l'analyse de la hauteur des restes successifs dans l'algorithme de division euclidienne. En particulier, elle permet de déduire que la borne s'applique aussi à tous ces restes. Si lors de l'algorithme de division euclidienne cette borne n'est pas respectée, cela indique donc directement qu'il n'y a pas divisibilité. Cette borne sur la hauteur du quotient dépend du nombre de monômes non nuls de Q de manière exponentielle. Cependant elle donne une borne sur la taille binaire des coefficients de Q qui est elle polynomiale, le stockage de tous ceux-ci ne requérant qu'au plus $\#Q(\left\lceil \frac{\#Q-1}{2} \right\rceil \log \|P\| + \log \|F\|)$ bits. Dans les conditions du théorème 4.10, le quotient $Q_0^\star$ calculé en première partie de l'algorithme 13 TestDivisibilité a justement un nombre de monômes polynomial dans la taille des entrées.

Corollaire 4.14. *Pour des polynômes F et P dans $\mathbb{Z}[x]$ tels que décrits dans le théorème 4.10, l'algorithme 13* TestDivisibilité *peut soit calculer Q_0 soit déduire qu'il n'y a pas divisibilité en temps polynomial dans la taille binaire des entrées F et P.*

Démonstration. Par le fait 1.19, FP_0^t a une hauteur d'au plus $(T\,\|P\|)^t\,\|F\|$. La hauteur demeurant inchangée lors du passage au polynôme réciproque, le lemme 1.20 donne $(\|P\|+1)^{\lceil\frac{\#Q-1}{2}\rceil}(T\,\|P\|)^t\,\|F\|$ comme borne sur la hauteur de $Q_0^\star = (FP_0^t)^\star \operatorname{quo} P^\star$ s'il y a divisibilité. Puisque les hypothèses du théorème 4.10 garantissent que $\#Q_0$ est polynomial dans la taille de F et P, les nombres manipulés auront tous une taille binaire polynomiale. S'il n'y a pas divisibilité et qu'un nombre dépasse la borne lors du calcul de la division euclidienne, l'algorithme peut conclure à la non divisibilité sans aller au bout du calcul de $Q_0^\star$. □

Ainsi la première partie de l'algorithme est tout aussi efficace dans le cas de polynômes à coefficients entiers.

En revanche dans la seconde partie de l'algorithme seul le reste est calculé car son degré est polynomial alors que le quotient pourrait avoir un nombre exponentiel de monômes non nuls. Mais dans un tel calcul, la réduction modulo un polynôme P sans borne (autre que le degré) sur le nombre de monômes non nuls du quotient, la croissance des coefficients peut être extrêmement importante.

Des bornes plus fines, dépendant de la structure du diviseur, ont été données en section 1.4

Fait 4.15 (lemme 1.22, page 21)**.** *Soit F, P des polynômes dans $\mathbb{Z}[x]$ de degré $d+d_p-1$ et d_p, ayant $\#F$ et $\#P$ monômes non nuls avec $\#P \geq 2$, P unitaire de paramètre d'écart γ. Le polynôme $F \bmod P$ a au plus $\#F(\#P-1)^{\lceil\frac{d}{\gamma d_p}\rceil}$ monômes non nuls. Sa hauteur est d'au plus $\|F \bmod P\| \leq \|F\|\,(\#P\,\|P\|)^{\lceil\frac{d}{\gamma d_p}\rceil}$.*

La preuve du lemme 1.22 donne en fait une méthode pour calculer le reste en manipulant uniquement des polynômes respectant les bornes données pour $F \bmod P$. Il s'agit d'un cas particulier du théorème 4.10 avec $P_1 = 1$ ce qui donne un paramètre d'écart γ valant $\frac{d}{d_p}$ à une constante près. Ainsi il permet d'affirmer que dans ce cas précis la divisibilité est encore plus facile à tester puisque le reste peut être calculé directement en s'appuyant sur la preuve du lemme. Si dans ce cas le reste a un nombre polynomial de monômes non nuls, il n'en va pas forcément de même pour le quotient. Un exemple extrême peut être donné par les polynômes cyclotomiques en prenant $F = x^d - 1$ et $P = x - 1$.

Cependant, dans le cas général une telle borne n'est pas réjouissante. Elle indique que pour s'assurer que le reste R_1 a des coefficients bornés, il faut avoir un écart important, proportionnel au degré de F, entre les deux premiers coefficients de P. Or le reste qu'il faut calculer dans la seconde partie de l'algorithme 13 TestDivisibilité est celui d'une division par P_0^t dont le degré est polynomial dans $T(\log d + \log d_p)$. Vu la forme du

polynôme (et sachant que $P(0) \neq 0$), l'écart le plus important est de $\deg(P_0)$ et le plus faible de 1. Ce qui donne pour la borne sur la hauteur une valeur au mieux exponentielle en $\frac{d}{\deg(P_0)}$, au pire exponentielle en d. Dans tous les cas, vu la borne sur le degré de P_0, la taille binaire des coefficients peut être linéaire en d donc exponentielle en la taille de F qui dépend de $\log d$.

Cette croissance est trop importante pour pouvoir effectuer le test en temps polynomial. La solution est alors d'effectuer les calculs modulo un nombre premier qui ne divise pas ce reste. L'approche repose sur le corollaire 2.4 mais est alors probabiliste. Si le nombre premier est mal choisi et divise tous les coefficients d'un reste non nul, l'algorithme répondra quand même qu'il y a divisibilité. En revanche, il n'y a pas d'erreur si P divise bel et bien F.

Corollaire 4.16. *Soit F et P dans $\mathbb{Z}[x]$ des polynômes tels que décrits dans le théorème 4.10. En passant dans $\mathbb{F}_q$ pour le calcul de R_1 avec $q = \text{PremierAléa}(\lambda, \epsilon)$ pour $\lambda = \max(21, \frac{5}{3\epsilon} \ln \|F\| (\#P \|P\|)^d)$, l'algorithme 13 TestDivisibilité peut décider si P divise F en temps polynomial en la taille binaire de F et P et avec une probabilité d'erreur d'au plus 2ϵ dans le cas où P ne divise pas F.*

Il n'est pas connu si décider qu'un polynôme creux en divise un autre est un problème polynomial ou non. Cette section a cependant permis de prouver l'existence de familles non triviales de polynômes pour lesquelles il existe un test en temps polynomial pour décider de la divisibilité. Ces polynômes sont caractérisés par d'importants écarts entre des regroupement de monômes formant des polynômes de plus petit degré. Si le test est déterministe en caractéristique positive, il est seulement probabiliste pour les polynômes dans $\mathbb{Z}[x]$.

4.2. Vérification d'un produit modulaire

Cette section est consacrée au test d'identités polynomiales de la forme $FG \bmod P = 0$ avec P un polynôme creux. Le polynôme $FG \bmod P$ peut être calculé en temps polynomial dans certains cas. En particulier si les polynômes F et G sont des polynômes denses, ce calcul ne nécessite qu'un nombre quasi-linéaire d'opérations dans l'anneau de définition. Par ailleurs ce calcul est déterministe. Un algorithme pour tester l'identité $FG \bmod P = 0$ doit donc être plus efficace que le calcul de $FG \bmod P$. Pour pouvoir être plus efficace que le calcul du produit puis de la division, nous passons par une méthode probabiliste. L'idée est classique, il s'agit d'évaluer le polynôme $FG \bmod P$ en un point aléatoire. Cette évaluation s'appuie sur les algorithmes présentés dans la section 2.3 ce qui permet de ne pas avoir à calculer le produit lui-même. Puisque la réduction modulo P ainsi que l'évaluation en un point commutent avec l'addition, cette méthode s'étend à des sommes de produits modulaires $\sum_i F_i G_i \bmod P$. En particulier, elle permet de tester une égalité de la forme $FG \bmod P = R$ correspondant à $FG + (-1)R \bmod P = 0$ en prenant essentiellement le même temps que pour $FG \bmod P = 0$. Il est ainsi possible de vérifier la validité d'un produit modulaire. Ce type de produit est une opération très fréquente,

notamment dans l'arithmétique sur les corps finis puisque tout corps non premier $\mathbb{F}_{q^s}$ est une extension $\mathbb{F}_q/P$ d'un corps premier $\mathbb{F}_q$. De plus, le polynôme irréductible P de degré s est souvent creux et même n'a souvent qu'un nombre constant de monômes que ce soit sur $\mathbb{F}_2$ [Gol82] ou sur un corps $\mathbb{F}_q$ plus général [MP13].

Si le polynôme P est toujours considéré comme creux, ce n'est pas le cas des polynômes F et G. Les algorithmes présentés dans cette sections sont analysés dans les deux cas s'il s'agit de deux polynômes denses ou s'il s'agit de deux polynômes creux. Dans ce second cas, l'analyse prendra en compte le paramètre d'écart γ de P qui permet de contrôler la densification de la réduction.

La question traitée dans cette partie est très proche de la question de la divisibilité vue en section 4.1, puisqu'il s'agit de tester si P divise le polynôme FG. C'est un cas particulier plus simple puisque le degré de FG est borné par $2d = 2\deg(P)$. De plus, si le paramètre γ est une constante fixée, P est toujours dans les conditions du théorème 4.10 (avec $P = P_0 + x^d$ et $\deg(P_0) = (1-\gamma)d$). L'objectif ici n'est donc pas de montrer l'existence d'un test déterministe ou probabiliste en temps polynomial mais de fournir un test aussi efficace que possible. Ce test doit notamment être plus rapide que le calcul explicite de $FG \bmod P$ qui fournirait déjà un test déterministe. Les algorithmes présentés dans cette section sont en général plus rapides mais probabilistes.

La vérification du produit modulaire dense ou creux est d'abord présentée en section 4.2.1 de manière abstraite pour des polynômes définis dans un anneau intègre quelconque. L'analyse est alors effectuée en comptant les opérations dans l'anneau. Dans les sections 4.2.2 et 4.2.3 l'analyse est affinée pour compter le nombre d'opérations binaires pour des polynômes définis respectivement sur les entiers et sur des corps finis. Quelques adaptations sont aussi apportées pour permettre le bon fonctionnement et l'efficacité des algorithmes. Les résultats présentés dans cette section et la suivante proviennent d'un article publié en 2023 [GGP23].

4.2.1. Cas général de polynôme dans $\mathbb{A}[x]$

L'algorithme 14 VérificationModulaire décrit ci-après est une application directe des théorèmes 2.30 et 2.37 (pages 48 et 56). Il couvre à la fois le cas des polynômes denses et des polynômes creux. Sa présentation sert principalement de point de départ pour pouvoir discuter de ses adaptations dans des anneaux de coefficients spécifiques. À ce stade, les seules contraintes sur l'anneau des coefficients est qu'il soit intègre et suffisamment grand, sans cela la validité de l'algorithme ne serait plus assurée.

Théorème 4.17. *Si $\mathbb{A}$ est un anneau intègre ayant au moins $\frac{1}{\epsilon}(d-1)$ éléments, l'algorithme 14* VérificationModulaire *décide si $FG \bmod P$ est égal à 0. Il peut se tromper, avec une probabilité d'échec inférieure à ϵ, lorsqu'il n'y a pas égalité.*

- *Si F et G sont des polynômes denses, l'algorithme effectue $\mathcal{O}(\#Pd)$ opérations dans $\mathbb{A}$.*

Algorithme 14 VérificationModulaire

Entrée : F, G et $P \in \mathbb{A}[x]$, $\mathbb{A}$ un anneau intègre, P unitaire et $\deg(F), \deg(G) < d = \deg(P)$; $0 < \epsilon < 1$.

Sortie : Vrai si $FG \bmod P = 0$, faux avec une probabilité d'au moins $1 - \epsilon$ dans le cas contraire.

1: $\alpha \leftarrow$ un élément aléatoire d'un ensemble $\mathcal{E} \subset \mathbb{A}$, ayant au moins $\frac{1}{\epsilon}(d-1)$ éléments
2: $\beta \leftarrow (FG \bmod P)(\alpha)$ ▷ théorèmes 2.30 et 2.37
3: **Renvoyer vrai** if $\beta = 0$, **faux** sinon

· *Si F et G sont des polynômes creux et γ est le paramètre d'écart de P, l'algorithme effectue $\mathcal{O}((\#F\#P^{\lceil \frac{1}{\gamma}-1 \rceil} + \#G)\log d)$ opérations dans $\mathbb{A}$.*

Démonstration. Si $FG \bmod P = 0$, $(FG \bmod P)(\alpha) = 0$ pour tout α et l'algorithme renvoie toujours vrai à raison. Sinon, $FG \bmod P$ est un polynôme non nul et a donc au plus $d-1$ racines puisque $\mathbb{A}$ est intègre. Par conséquent, la probabilité qu'un point α, choisi aléatoirement dans $\mathcal{E}$, soit une racine de $FG \bmod P$ est d'au plus $(d-1)/\frac{1}{\epsilon}(d-1) = \epsilon$. Dans ce cas, l'algorithme se trompe et renvoie vrai avec une probabilité d'au plus ϵ.

La complexité est déterminée par le coût d'une seule évaluation modulaire. Celui-ci est donné dans les théorèmes 2.30 et 2.37 (pages 48 et 56) pour des polynômes F et G respectivement denses ou creux. □

La méthode d'évaluation d'un produit modulaire présentée en section 2.3.1 ne s'applique pas à des produits de plus de deux polynômes. Cependant, puisqu'elle s'applique à tout produit modulaire de deux polynômes, l'algorithme s'étend naturellement à une somme de m produits modulo P. Il convient simplement d'évaluer les m produits modulaires et d'ajouter les résultats obtenus. En effet, la réduction modulaire commute avec l'addition et le degré restant strictement inférieur à d, l'argument sur le nombre de racines s'applique toujours.

Corollaire 4.18. *Il est possible de décider si $\sum_{i=0}^{m-1} F_i G_i \bmod P$ est nulle pour des polynômes F_i, G_i de degré strictement inférieur à $d = \deg P$ et ayant au plus T monômes chacun en $\mathcal{O}(m\#Pd)$ opérations dans $\mathbb{A}$ pour des polynômes denses, ou $\mathcal{O}(mT\#P^{\lceil \frac{1}{\gamma}-1 \rceil}\log d)$ pour des polynômes creux. La probabilité de succès est d'au moins $1-\epsilon$.*

Si $\mathbb{A}$ ne contient pas suffisamment de points, l'algorithme n'apporte pas les mêmes garanties de succès. La probabilité $1-\epsilon$ peut être retrouvée en répétant l'algorithme tout en choisissant α dans un ensemble plus petit. Ainsi, si $\mathcal{E}$ contient nd éléments avec $n > 1$, $\mathcal{O}(\log \frac{1}{\epsilon})$ répétitions de l'algorithme sont suffisantes. Une telle approche nécessite encore une fois que $\mathbb{A}$ soit suffisamment large, même s'il peut être plus petit. De plus toute la complexité de l'algorithme est multipliée par un facteur $\log \frac{1}{\epsilon}$ ce qui représente une perte d'efficacité sauf si les évaluations sont effectuées en parallèle. L'approche qui sera préférée dans toute cette section est le passage dans une extension $\mathbb{A}_{\text{ext}}$ de l'anneau

$\mathbb{A}$ qui elle contiendra un sous ensemble assez large. En section 2.3, les complexités de l'évaluation ont aussi été décrites pour un point $\alpha \in \mathbb{A}_{\text{ext}}$.

Corollaire 4.19. *Soit $\mathbb{A}$ un anneau intègre ayant moins de $\frac{1}{\epsilon}(d-1)$ points et $\mathbb{A}_{ext}$ une extension de $\mathbb{A}$ qui a elle plus de $\frac{1}{\epsilon}(d-1)$ points. L'algorithme 14* VérificationModulaire *peut être adapté en choisissant aléatoirement un point dans $\mathbb{A}_{ext}$. La probabilité de succès reste $1-\epsilon$. Il effectue alors $\mathcal{O}(d\#P)$ opérations dans $\mathbb{A}$ et $\mathcal{O}(d)$ opérations dans $\mathbb{A}_{ext}$ si les polynômes sont denses et $\mathcal{O}((\#F\#P^{\lceil \frac{1}{\gamma}-1 \rceil} + \#G)\log d)$ opérations dans $\mathbb{A}_{ext}$ s'il sont creux.*

Dans le cas dense, l'algorithme 14 VérificationModulaire effectue un nombre asymptotiquement optimal d'opérations dans $\mathbb{A}$ à la condition que $\#P$ soit une constante. Pour tout polynôme F et G, il est plus rapide que le calcul du produit modulaire si $\#Pd < \mathsf{M}(d)$ c'est à dire si $\#P < \log(d)\log\log(d)$ pour un anneau intègre $\mathbb{A}$ quelconque. Dans le cas creux, la complexité donnée au théorème 4.17 n'est pas linéaire dans la taille des entrées. En effet, le nombre $\#P$ est élevé à une puissance potentiellement importante $\lceil \frac{1}{\gamma} - 1 \rceil$ mais surtout il y a un facteur $\log d$ sur le nombre d'opérations dans $\mathbb{A}$ alors que les polynômes donnés en entrée ne contiennent que $(\#F + \#G)$ éléments de $\mathbb{A}$. Cependant l'efficacité de cet algorithme de vérification ne doit pas seulement être comparée à la taille des entrées, mais aussi à celle de l'autre alternative qui est le calcul déterministe de $FG \bmod P$. Puisqu'il n'existe pas d'algorithme optimisant une telle opération, il est raisonnable de considérer qu'elle est effectuée en deux étapes d'abord le produit ensuite la division. Ainsi le coût du calcul de $FG \bmod P$ comprend au moins une opération par monôme non nul de FG, et ce quelque soit l'algorithme utilisé pour le produit. Ainsi au moins $\#FG$ opérations dans $\mathbb{A}$ sont effectuées. L'algorithme de vérification en effectue lui $\mathcal{O}((\#F + \#G)\log d)$ en supposant que $\#P$ est une constante. Puisque $\#FG$ peut valoir jusqu'à $\#F\#G$, la vérification est plus rapide si en plus $d = 2^{\mathcal{O}(\#F+\#G)}$. Et ceci sans prendre en compte le coût de la division.

Ces conditions sur P ne sont pas trop restrictives, en particulier dans la section 4.3, pour vérifier un produit de polynômes, le produit modulaire sera utilisé avec P un binôme dont le degré dépendra du degré et du nombre de monômes des entrées.

L'efficacité en pratique de l'algorithme 14 VérificationModulaire dépend bien évidemment de l'anneau intègre $\mathbb{A}$ sur lequel il est exécuté. En effet, la multiplication de polynômes elle-même est plus ou moins efficace en fonction de $\mathbb{A}$ et effectue notamment moins de $\mathcal{O}(d\log d\log\log d)$ opérations dans $\mathbb{A}$ pour $\mathbb{A} = \mathbb{F}_q$ [HH19 ; HH22]. De plus si $\mathbb{A}$ est trop petit, le passage dans une extension apporte un coût supplémentaire puisqu'une opération dans $\mathbb{A}_{\text{ext}}$ correspond généralement à un nombre non constant d'opérations dans $\mathbb{A}$. Dans les prochaines sections, sont développées des adaptations de l'algorithme de vérification pour les cas de polynômes à coefficients entiers ou dans un corps fini.

4.2.2. Cas des polynômes à coefficients entiers

Si les polynômes sont définis sur $\mathbb{Z}$, il n'y a naturellement aucune difficulté à obtenir un sous-ensemble $\mathcal{E}$ de $\mathbb{Z}$ ayant suffisamment de points pour exécuter l'algorithme 14. Cependant, il faut prévenir la croissance des entiers lors de l'évaluation. En effet élever un entier à la puissance $d-1$ en fait un entier d'au moins d bits. La manipulation de tels entiers conduirait à une complexité polynomiale dans le cas dense et exponentielle dans le cas creux. Pour éviter cela il est classique de choisir un nombre premier aléatoire q et d'effectuer l'ensemble des opérations dans $\mathbb{F}_q$. Pour que l'algorithme de vérification reste valide, il faut s'assurer de deux propriétés relatives au nombre premier q. D'une part il doit être assez grand pour avoir une probabilité de succès supérieure à $1-\epsilon$, c'est à dire valant au moins $\frac{1}{\epsilon}(d-1)$. D'autre part, si $FG \bmod P \neq 0$, cela doit rester vrai modulo q. Le corollaire 2.4 donne le moyen de choisir q pour que cela ait une forte probabilité d'être vérifié.

En veillant à respecter ces deux contraintes, il est possible d'adapter l'algorithme 14 au cas des entiers, ce qui est fait dans l'algorithme VérificationModulaireEntière qui suit. Pour rappel, pour tout polynôme $R \in \mathbb{Z}[x]$, $\|R\|$ désigne la hauteur de R, le maximum des valeurs absolues de ses coefficients.

Algorithme 15 VérificationModulaireEntière

Entrée : F, G et $P \in \mathbb{Z}[x]$, avec P unitaire, de paramètre d'écart γ et $\deg(F), \deg(G) < d = \deg(P)$; $0 < \epsilon < 1$.

Sortie : Vrai si $FG \bmod P = 0$, faux avec une probabilité d'au moins $1-\epsilon$ dans le cas contraire.

1: $H \leftarrow \min(\#F, \#G)\, \|F\|\, \|G\|\, (\#P\, \|P\|)^{\lceil \frac{1}{\gamma} \rceil}$
2: $q \leftarrow$ PremierAléa$(\lambda, \frac{\epsilon}{4})$ avec $\lambda = \max(21, \frac{2}{\epsilon}n, \frac{20}{3\epsilon}\ln H)$
3: $(F_q, G_q, P_q) \leftarrow (F \bmod q, G \bmod q, P \bmod q)$
4: **Renvoyer** VérificationModulaire$(F_q, G_q, P_q, \frac{\epsilon}{2})$

Théorème 4.20. *Pour F, G et P trois polynômes dans $\mathbb{Z}[x]$, l'algorithme 15 décide si $FG \bmod P$ est égal à 0. Il peut se tromper, avec une probabilité d'échec inférieure à ϵ, lorsqu'il n'y a pas égalité. Si $H = \max(\|P\|, \|F\|, \|G\|)$ et $T = \max(\#F, \#G)$, l'algorithme effectue $\mathcal{O}(\log^2(\frac{d}{\epsilon}\log H)\log\log(\frac{d}{\epsilon}\log H)\log\frac{1}{\epsilon}\mathsf{I}(\log(\frac{d}{\epsilon}\log H)))$ opérations binaires pour obtenir un nombre premier q, plus*

- $\mathcal{O}((\#Pd + \frac{\log H}{\log(\frac{d}{\epsilon}\log H)})\mathsf{I}(\log(\frac{d}{\epsilon}\log H)))$ *opérations binaires si F et G sont denses, ou*
- $\mathcal{O}((T\#P^{\lceil \frac{1}{\gamma}-1 \rceil}\log d + (T+\#P)\frac{\log H}{\log(\frac{d}{\epsilon}\log H)})\mathsf{I}(\log(\frac{d}{\epsilon}\log H)))$ *opérations binaires si F et G sont creux.*

Démonstration. Si $FG \bmod P = 0$, l'algorithme renvoie toujours vrai. Sinon, il peut renvoyer vrai à tort pour deux raisons : soit $(F_qG_q) \bmod P_q = 0$ alors qu'il n'y avait pas égalité sur $\mathbb{Z}$, soit l'algorithme VérificationModulaire lui-même renvoie vrai à tort. Chacune de ces deux situations n'est possible qu'avec une probabilité d'au plus $\frac{\epsilon}{2}$. En

effet, en appliquant le fait 1.19 et le lemme 1.22 à $FG \bmod P$, il est possible de voir que H est une borne sur la hauteur du polynôme. Un nombre premier choisi aléatoirement dans l'intervalle $[\lambda, 2\lambda]$ avec $\lambda \geq \max(21, \frac{20}{3\epsilon} \ln H)$ a donc une probabilité au plus $\frac{\epsilon}{4}$, d'annuler tous les coefficients de ce polynôme d'après le corollaire 2.4. Le nombre q choisi à l'étape 2 est justement un tel nombre premier avec une probabilité d'au moins $1 - \frac{\epsilon}{4}$. Par ailleurs, pour que l'appel à l'algorithme 14 VérificationModulaire (page 97) puisse fonctionner correctement, c'est à dire avec une probabilité d'échec d'au plus $\frac{\epsilon}{2}$, il faut que q vaille au moins $\frac{2}{\epsilon}(d-1)$ ce qui est le cas puisqu'il est choisi dans l'intervalle $[\lambda, 2\lambda]$ avec $\lambda \geq \frac{2}{\epsilon}d$. En additionnant toutes ces probabilités d'erreurs, cela donne une probabilité d'échec pour l'algorithme 15 d'au plus ϵ.

Pour analyser la complexité de l'algorithme, il faut exprimer q en fonction des paramètres d'entrées. Comme $\#P$, $\#F$, $\#G \leq d$, $H = \mathcal{O}(d^{1+\lceil \frac{1}{\gamma} \rceil} C^{2+\lceil \frac{1}{\gamma} \rceil})$. D'où, $\ln H = \mathcal{O}(\frac{1}{\gamma}(\log d + \log H)) = \mathcal{O}(d(\log d + \log H))$ puisque $\frac{1}{\gamma} \leq d$. Ceci implique que $q = \mathcal{O}(\frac{d}{\epsilon}(\log d + \log H))$ et $\log q = \mathcal{O}(\log(\frac{d}{\epsilon} \log H))$.

D'après le fait 1.13, l'étape 2 effectue $\mathcal{O}(\log \frac{1}{\epsilon} \log^2 q \log\log q \mathsf{I}(\log q))$ opérations binaires, autrement dit

$$\mathcal{O}\left(\log \tfrac{1}{\epsilon} \log^2(\tfrac{d}{\epsilon} \log H) \log\log(\tfrac{d}{\epsilon} \log H) \mathsf{I}(\log(\tfrac{d}{\epsilon} \log H))\right).$$

L'étape 3 effectue la division de tous les coefficients de F, G et P par q ce qui nécessite $\mathcal{O}(d \frac{\log H}{\log q} \mathsf{I}(\log q))$ opérations binaires dans le cas dense et $\mathcal{O}((T + \#P) \frac{\log H}{\log q} \mathsf{I}(\log q))$ opérations binaires dans le cas creux. Enfin, l'étape 4 effectue $\mathcal{O}(\#P d \times \mathsf{I}(\log q))$ opérations binaires pour des polynômes denses et $\mathcal{O}(T \#P^{\lceil \frac{1}{\gamma} - 1 \rceil} (\log d) \times \mathsf{I}(\log q))$ opérations binaires pour des polynômes creux d'après le théorème 4.17. En remplaçant $\log q$ par $\log(\frac{d}{\epsilon} \log H)$ et en additionnant toutes les complexités, on retrouve bien la complexité attendue dans chacun des deux cas. □

Si obtenir q s'avère trop coûteux, notamment à cause de la contrainte $q > d$, il est possible d'en choisir un qui ne la respecte pas tout en s'assurant qu'il n'annule pas $FG \bmod P$ si le polynôme n'est pas nul sur $\mathbb{Z}$. Pour cela, il suffit de prendre $\lambda = \max(21, \frac{2}{\epsilon}n, \frac{20}{3\epsilon} \ln H)$, puis dans le corps $\mathbb{F}_q$ correspondant générer un polynôme irréductible de la forme $x^{r^k} - a$ [MP13, Corollary 3.4.7], r étant relié à l'ordre multiplicatif de $a \in \mathbb{F}_q$ et k pouvant être aussi grand que nécessaire pour pouvoir appliquer l'algorithme 14 VérificationModulaire dans $\mathbb{F}_q[x]/(x^{r^k} - a)$ avec la probabilité de succès désirée. Cette méthode en revanche teste un produit modulaire de polynômes en effectuant de l'arithmétique modulaire de polynômes ce qui n'est pas complètement satisfaisant. De plus, la génération du nombre premier q est rarement la partie la plus coûteuse de l'algorithme. En particulier pour des polynômes denses, puisqu'elle a une dépendance en $\log d$ quand leur taille dépend de d.

Remarque 4.21. Dans la plupart des cas, le coût de l'obtention du nombre premier est négligeable en comparaison du coût du reste de l'algorithme. Pour des polynômes creux, c'est le cas si le degré d n'est pas trop large comparé aux autres paramètres de taille, plus précisément si $\frac{d}{\epsilon} = ((T + \#P) \log H)^{\mathcal{O}(1)}$.

Lors d'une réduction modulaire de polynômes à coefficients entiers, les coefficients peuvent croître de manière significative, comme le montre la borne donnée par le lemme 1.22 (page 21). De son côté, l'algorithme 15 VérificationModulaireEntière travaille avec des entiers de $\mathcal{O}(\log(\frac{d}{\epsilon}\log H))$ bits. Cela correspond à une taille logarithmique en la taille des entrées dans le cas dense et linéaire dans le cas creux. Ainsi la vérification évite de payer des surcoûts liés au grossissement des coefficients, ce qui n'est pas le cas lors du calcul direct du polynôme $FG \bmod P$. Prendre cette potentielle croissance en compte pour comparer la vérification au calcul modulaire est cependant délicat. La prochaine remarque liste donc des cas où la vérification du produit modulaire est plus rapide que le calcul du produit modulaire sans tenir compte de la croissance des coefficients.

Remarque 4.22. Pour $\epsilon = 1/d^{\mathcal{O}(1)}$, L'algorithme 15 VérificationModulaireEntière est en général plus rapide que le produit modulaire de polynômes quand :

(i) les polynômes sont denses et $\#P < \min(\frac{\log d}{\log\log d}, \frac{\log H}{\log\log\log H})$;

(ii) les polynômes sont creux et $\frac{d}{\epsilon} = ((T + \#P)\log H)^{\mathcal{O}(1)}$.

Démonstration. Considérons $\epsilon = 1/d^{\mathcal{O}(1)}$. En particulier, $\log\frac{d}{\epsilon} = \mathcal{O}(\log d)$, le coût de la génération de q est donc négligeable dans le cas dense. Il l'est aussi dans le cas creux d'après la remarque 4.21 puisque en plus $\frac{d}{\epsilon} = ((T + \#P)\log H)^{\mathcal{O}(1)}$. Cette inégalité implique aussi que $\log d = \mathcal{O}(\min(T, \#P))$ et $\mathsf{I}(\log(d\log H)) = \mathsf{I}(\log\log H)$. Ce qui rend la vérification plus rapide que le produit puisque celui-ci nécessite en général $\#(FG \bmod P) = \mathcal{O}(T^2\#P^{\lceil\frac{1}{\gamma}\rceil})$ opérations sur des entiers de $\log H$ bits.

Pour le cas dense, la comparaison se fait avec le coût de la multiplication de deux polynômes de degré d et avec des coefficients bornés par H. Cette multiplication se ramène à la multiplication d'entiers par une substitution de Kronecker et nécessite $\mathsf{I}(d(\log d + \log H)) = \mathcal{O}(d(\log^2 d + \log d\log H + \log H\log\log H))$ opérations binaires (fait 1.6). D'après le théorème 4.20 la vérification effectue $\mathcal{O}(\#Pd\mathsf{I}(\log d + \log\log H) + d\log H\log(\log d + \log\log H))$ opérations binaires puisque $\epsilon = 1/d^{\mathcal{O}(1)}$. Le second terme est clairement dominé par le coût de la multiplication de deux polynômes. Le premier terme se développe en

$$\mathcal{O}(\#Pd((\log d + \log\log H)\log\log d + (\log d + \log\log H)\log\log\log H)).$$

Quand $\#P < \min(\frac{\log d}{\log\log d}, \frac{\log H}{\log\log\log H})$, ce terme est dominé par $\mathcal{O}(d(\log^2 n + \log d\log H + \log H\log\log H))$, ce qui correspond à la complexité du produit de polynômes. □

Encore une fois, l'algorithme s'étend naturellement au cas d'une somme de m produits en effectuant les $2m$ réductions modulo q, les m évaluations des produits modulaires et la somme des résultats. Il faut juste veiller à borner convenablement la hauteur du polynôme à tester. Elle est simplement m fois plus grande que celle pour un seul produit modulaire. En effet elle s'obtient comme la hauteur de la réduction modulo P de la somme des produits et la hauteur du polynôme réduit n'influence que d'un facteur linéaire la hauteur du reste.

Corollaire 4.23. *Il est possible de décider si* $\sum_{i=0}^{m-1} F_i G_i \bmod P$ *est nulle pour des polynômes* $F_i, G_i \in \mathbb{Z}[x]$ *de degré strictement inférieur à* $d = \deg P$ *et ayant au plus* T *monômes chacun avec des coefficients bornés par* H*. La complexité est celle du théorème 4.20 en remplaçant* H *par* mH *et en multipliant par* m *les coûts non liés à la génération du nombre premier. Ainsi cela nécessite* $\mathcal{O}(\log^2(\frac{d}{\epsilon}\log mH)\log\log(\frac{d}{\epsilon}\log mH)\log\frac{1}{\epsilon}\mathsf{I}(\log(\frac{d}{\epsilon}\log mH)))$ *opérations binaires pour obtenir un nombre premier* q*, plus*

- $\mathcal{O}(m(\#Pd + \frac{\log H}{\log(\frac{d}{\epsilon}\log mH)})\mathsf{I}(\log(\frac{d}{\epsilon}\log mH)))$ *opérations binaires si* F *et* G *sont denses, ou*
- $\mathcal{O}(m(T\#P^{\lceil\frac{1}{\gamma}-1\rceil}\log d + (T+\#P)\frac{\log mH}{\log(\frac{d}{\epsilon}\log mH)})\mathsf{I}(\log(\frac{d}{\epsilon}\log mH)))$ *opérations binaires si* F *et* G *sont creux.*

4.2.3. Cas des polynômes à coefficients dans des corps finis

La situation pour des polynômes sur un corps fini $\mathbb{F}_q$ est différente. D'une part il n'y a pas de croissance de coefficients à prévenir ce qui simplifie les calculs. D'autre part, il peut ne pas y avoir suffisamment de points dans le corps fini ce qui impose de changer l'anneau d'évaluation. Si q est assez grand, le théorème 4.17 s'applique directement. Autrement, les calculs peuvent être effectués dans une extension $\mathbb{F}_{q^s}$ ayant suffisamment de points pour appliquer le corollaire 4.19.

Corollaire 4.24. *Soient* F*,* G *et* $P \in \mathbb{F}_q[x]$ *avec* P *unitaire et* $\deg(F), \deg(G) < d = \deg(P)$*. Il est possible de décider si* $FG \bmod P = 0$ *avec une probabilité de succès d'au moins* $1-\epsilon$ *en utilisant l'algorithme 14* VérificationModulaire *et en effectuant,* $\mathcal{O}(\log\frac{1}{\epsilon}\log_q\frac{d}{\epsilon}\log(\log_q\frac{d}{\epsilon})\mathsf{M}(\log_q\frac{d}{\epsilon})(\log q + \log\log_q\frac{d}{\epsilon}))$ *opérations dans* $\mathbb{F}_q$ *pour obtenir un polynôme irréductible de degré* $\mathcal{O}(\log_q\frac{d}{\epsilon})$ *si nécessaire, plus*

- $\mathcal{O}(d\#P + d\mathsf{M}(\log_q\frac{d}{\epsilon}))$ *opérations dans* $\mathbb{F}_q$ *si* F *et* G *sont des polynômes denses, ou*
- $\mathcal{O}(T\#P^{\lceil\frac{1}{\gamma}-1\rceil}\log d\mathsf{M}(\log_q\frac{d}{\epsilon}))$ *opérations dans* $\mathbb{F}_q$ *si* F *et* G *sont des polynômes creux ayant au plus* T *monômes.*

Démonstration. Si $q \geq \frac{1}{\epsilon}(d-1)$, le théorème 4.17 s'applique directement et il n'est pas nécessaire de calculer un polynôme irréductible. Dans le cas contraire, le corollaire 4.19 passe par une extension de $\mathbb{F}_q$ ayant au moins $\frac{1}{\epsilon}(d-1)$ points. Plus précisément, en utilisant le fait 1.16 il est possible de calculer avec une probabilité d'au moins $1-\frac{\epsilon}{2}$ un polynôme irréductible de degré s dans $\mathbb{F}_q[x]$, avec s le plus petit entier tel que $q^s \geq \frac{2}{\epsilon}(d-1)$. L'algorithme ainsi obtenu peut échouer soit s'il ne parvient pas à trouver un polynôme irréductible pour définir $\mathbb{F}_{q^s}$, soit si l'algorithme 14 appelé sur $\mathbb{F}_{q^s}$ échoue. Vu le choix de s, cette seconde cause d'échec n'advient qu'avec une probabilité d'au plus $\frac{\epsilon}{2}$. Ainsi la probabilité d'échec totale est bien d'au plus ϵ.

La génération du polynôme irréductible de degré s nécessite $\mathcal{O}(\log\frac{1}{\epsilon}s\log s\mathsf{M}(s)(\log q + \log s))$ opérations dans $\mathbb{F}_q$ d'après le fait 1.16. Puisqu'une opération dans $\mathbb{F}_{q^s}$ correspond

à $\mathsf{M}(s)$ opérations dans $\mathbb{F}_q$ et que, par définition, le degré de l'extension est $s = \mathcal{O}(\log_q \frac{d}{\epsilon})$ on obtient bien la complexité annoncée en appliquant le corollaire 4.19. □

Comme dans le cas général (corollaire 4.18), cette approche s'étend au cas de la somme de m produits modulaires en multipliant par m la complexité. La partie concernant la génération du polynôme irréductible n'a en fait pas besoin d'être multipliée par m. Cette partie s'avère bien souvent négligeable.

Remarque 4.25. Si $\epsilon = 1/d^{\mathcal{O}(1)}$, le coût de l'obtention du polynôme irréductible est négligeable dans le cas dense. Ainsi l'algorithme 14 effectue $\mathcal{O}(\#Pd + d \times \mathsf{M}(\log_q d))$ opérations dans $\mathbb{F}_q$. Si de plus le degré vérifie $\log d = o(T)$, le coût de l'obtention du polynôme irréductible est aussi négligeable dans le cas creux et l'algorithme effectue $\mathcal{O}(T\#P^{\lceil \frac{1}{\gamma} - 1 \rceil}(\log d) \times \mathsf{M}(\log_q d))$ opérations dans $\mathbb{F}_q$.

Comme le nombre de monômes non nuls de $FG \bmod P$ est borné par $T^2\#P^{\lceil \frac{1}{\gamma} \rceil}$ et que calculer $FG \bmod P$ nécessite au moins une opération dans $\mathbb{F}_q$ par monôme, cette remarque donne directement un cas où la vérification est plus rapide que le produit modulaire en général. De plus, l'arithmétique dans $\mathbb{F}_q$ peut même être effectuée avec des algorithmes naïfs dans ce cas.

Remarque 4.26. Si $\epsilon = 1/d^{\mathcal{O}(1)}$ et $d = T^{\mathcal{O}(1)}$, l'algorithme 14 pour des polynômes creux est en général plus rapide que le produit modulaire et effectue $\tilde{\mathcal{O}}(T\#P^{\lceil \frac{1}{\gamma} - 1 \rceil})$ opérations binaires si $q < \frac{d}{\epsilon}$.

Dans le cas de polynômes denses, la remarque 4.25 indique plutôt que la vérification peut être moins efficace que le calcul direct du produit modulaire $FG \bmod P$. Dans la remarque 4.27, nous précisons des cas dans lequel le résultat est positif, la vérification étant plus rapide que le calcul du produit. Pour simplifier les notations, il est supposé l'existence d'une constante de Linnik qui permet d'avoir $\mathsf{M}_q(d) = \mathcal{O}(d \log q(\log d + \log\log q))$ opérations binaires [HH22]. Par conséquent $\mathsf{M}_q(\log_q \frac{d}{\epsilon}) = \mathcal{O}(\log \frac{d}{\epsilon}(\log\log \frac{d}{\epsilon} + \log\log q))$. Alors que le calcul de $FG \bmod P$ effectue $\mathsf{M}_q(d)$ opérations binaires, la vérification en effectue $\mathcal{O}(\#Pn \log q \log\log q + n \log \frac{d}{\epsilon}(\log\log \frac{d}{\epsilon} + \log\log q))$. Si $\#P$ est une constante, la vérification est toujours plus efficace.

Remarque 4.27. En supposant que $\#P$ est une constante et $\epsilon = 1/d^{\mathcal{O}(1)}$, l'algorithme 14 VérificationModulaire dans le cas de polynômes denses avec $\mathbb{A} = \mathbb{F}_q$ est asymptotiquement plus rapide que le produit modulaire si

(i) $\log \frac{d}{\epsilon} < q < \frac{d}{\epsilon}$, puisque le produit modulaire coûte $\mathcal{O}(d \log q \log d)$ opérations binaires alors que la vérification effectue $\mathcal{O}(d \log \frac{d}{\epsilon} \log\log \frac{d}{\epsilon})$ opérations binaires en passant dans une extension.

(ii) $\frac{d}{\epsilon} < q < 2^{\frac{d}{\epsilon}}$, puisque le produit modulaire coûte $\mathcal{O}(d \log q \log d)$ opérations binaires alors que la vérification effectue $\mathcal{O}(d \log q \log\log q)$ opérations binaires en restant dans $\mathbb{F}_q$.

Si le corps est très grand, $q > 2^{\frac{d}{\epsilon}}$, l'algorithme 14 VérificationModulaire est asymptotiquement aussi rapide que le produit modulaire. Dans les deux cas, le facteur dominant est $\mathcal{O}(d \log q \log\log q)$.

Il est important de remarquer que le coût de la vérification du point (i) de la remarque 4.27 est obtenu en ayant recours à des algorithmes rapides de produits (modulaires) de polynômes pour vérifier un produit modulaire de polynômes de degré $\mathcal{O}(d)$. En effet pour construire cette vérification du produit modulaire passe par une extension de $\mathbb{F}_q$ ayant suffisamment de points pour appliquer l'analyse de probabilité du théorème 4.17. Or une extension de corps est obtenue comme un anneau quotient $\mathbb{F}_q[x]/\langle R\rangle$ pour un polynôme irréductible R. Ainsi tout produit dans l'extension est en fait un produit modulaire de polynômes avec R comme modulo. Pour annoncer la complexité de la remarque 4.27, c'est l'arithmétique la plus efficace qui a été considérée, en particulier pour ces produits modulaires. La vérification d'un produit modulaire dépend donc du calcul efficace de produits modulaires. Même si une telle dépendance n'est pas problématique en théorie, ce n'est pas forcément satisfaisant en pratique. Une solution serait alors de n'utiliser que des algorithmes naïfs pour les opérations arithmétiques dans l'extension de $\mathbb{F}_q$. Cependant une telle approche réduit le nombre de cas où l'algorithme de vérification est plus efficace.

Remarque 4.28. En considérant que toute l'arithmétique dans l'extension est effectuée par des algorithmes naïfs, l'algorithme 14 VérificationModulaire reste plus efficace que le produit modulaire seulement si $d^{1/3} < q$ dans le cas de polynômes denses.

4.2.3.1. Vérification sans passer dans une extension

Une autre approche est possible. Elle permet d'améliorer la complexité de la vérification dans le cas de polynômes denses sur des petits corps $\mathbb{F}_q$. La vérification peut alors être plus rapide que le produit modulaire même pour des $q < \log d$, ce qui est particulièrement intéressant pour le corps $\mathbb{F}_2$. Par ailleurs, la dépendance dans l'arithmétique de polynômes peut être totalement supprimée. Cette approche repose sur l'évaluation des polynômes non pas sur des éléments du corps mais sur des matrices. Ainsi il n'est pas nécessaire de passer dans une extension pour avoir plus de choix de points d'évaluation, il suffit d'augmenter les dimensions de la matrice. L'arithmétique de polynômes nécessaire aux calculs dans des extensions de corps est alors remplacée par l'arithmétique des matrices. Celle-ci peut cependant s'avérer plus coûteuse. Pour ne pas perdre en efficacité, les calculs ne seront pas effectués de manière déterministe mais vérifier de manière probabiliste en s'appuyant aussi sur l'algorithme de Freivalds pour vérifier des produits de matrices [Fre79].

En effet, c'est le choix de α dans une extension qui mène automatiquement à effectuer des produits de polynômes. Plutôt que de choisir un point aléatoire qui ne soit probablement pas une racine de $\Delta := (FG) \bmod P$, il est possible de choisir un polynôme $R \in \mathbb{F}_q[x]$ de degré $k < d$ qui ne soit probablement pas un diviseur de Δ. Pour tester si

R divise Δ, il suffit d'évaluer Δ sur la matrice compagnon C_R de R, définie par

$$C_R = \begin{pmatrix} 0 & 0 & \cdots & 0 & -r_0 \\ 1 & 0 & \cdots & 0 & -r_1 \\ 0 & 1 & \cdots & 0 & -r_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & -r_k \end{pmatrix}$$

avec $R = \sum_{i=0}^{k} r_i x^i$. Cette stratégie repose sur le fait que R est le polynôme minimal de sa matrice compagnon. Par conséquent, $R(C_R) = 0$ et tout polynôme Δ tel que $\Delta(C_R) = 0$ est forcément un multiple de R. Autrement dit R divise Δ si et seulement si $\Delta(C_R) = 0$. Dans la suite de cette section, il est prouvé que choisir R parmi les polynômes irréductibles de $\mathbb{F}_q[x]$ de degré $k = \mathcal{O}(\log d)$ permet de rendre cette approche plus rapide que celle passant par une extension, quand ϵ est constant.

Pour vérifier si $\Delta(C_R) = 0$, il faudrait procéder à l'évaluation et vérifier qu'elle donne bien la matrice nulle. Cette évaluation ne peut évidemment pas se faire directement puisque cette évaluation modulaire sur un anneau de matrices nécessiterait $\mathcal{O}(dk^\omega)$ opérations dans $\mathbb{F}_q$ avec $2 \leq \omega < 2.37286$ le meilleur exposant pour un produit de matrices [AW21]. Puisque $k = \mathcal{O}(\log d)$, cela n'apporterait aucune amélioration à la remarque 4.27.

Au lieu de calculer l'évaluation, il est possible de simplement la vérifier en s'appuyant sur la technique de Freivalds pour vérifier un produit de matrices [Fre79]. L'idée est que le produit de matrices $C = A \times B \in \mathbb{A}^{k \times k}$ peut être vérifié en s'assurant que $uC = (uA) \times B$ pour un vecteur aléatoire $u \in \{0,1\}^k$ avec une probabilité d'erreur de $1/2$. Pour s'assurer que l'évaluation d'un polynôme sur la matrice C_R vaut bien 0, il suffit de vérifier que c'est le cas pour sa projection par un vecteur u. La projection peut-être utilisée tout au long de l'évaluation en suivant un schéma de Horner pour éviter des produits matriciels.

$$uF(C_R) = u \sum_{i=0}^{d-1} f_i {C_R}^i = \left(\sum_{i=1}^{d-1} f_i u {C_R}^{i-1} \right) C_R + u f_0. \tag{4.1}$$

Ainsi, le calcul de $uF(C_R)$ pour un polynôme $F = \sum_{i=0}^{d-1} f_i x^i \in \mathbb{F}_q[x]$ peut être réalisé en effectuant $d-1$ produits de vecteurs par la matrice C_R et aucun produit matriciel. La projection d'une matrice compagnon ne nécessitant que $\mathcal{O}(k)$ opérations dans $\mathbb{F}_q$, $uF(C_R)$ peut être calculé en $\mathcal{O}(dk)$ opérations dans $\mathbb{F}_q$. L'évaluation de $\Delta = FG \bmod P$ est l'évaluation d'un produit modulaire et non d'un simple polynôme. Le schéma de Horner ne permet donc pas de calculer $u\Delta(C_R)$. L'algorithme permettant l'évaluation d'un produit modulaire est l'algorithme 4 ÉvaluationModulaire (page 48). Il s'applique à tout point α potentiellement dans une extension de l'anneau de définition des polynômes. Ici, l'extension est un anneau de matrices. C'est un anneau non commutatif, il convient de s'assurer que toutes les opérations décrites peuvent encore être effectuées. L'algorithme commence par évaluer F et P sur le point d'évaluation α. Ici, ces évaluations peuvent

être effectuées sur C_R avec la projection à gauche par u en suivant le schéma de Horner. La seconde partie construit progressivement le résultat en effectuant des produits par α l'un après l'autre. Si les évaluations initiales ont été projetées par u, multiplier par C_R revient simplement à faire un produit vecteur matrice et toutes les autres opérations ne concernent que des vecteurs. Il faut cependant faire attention que les produits concernés ne sont plus commutatifs. Alors que dans l'algorithme 4 ÉvaluationModulaire l'opération indiquée est une multiplication à gauche par α, il convient dans ce cas précis de multiplier à droite par C_R. Ainsi la projection par u qui s'effectue par la gauche est préservée.

Remarque 4.29. Il suffit de remplacer l'évaluation de $F(\alpha)$ et $P(\alpha)$ par $uF(C_R)$ et $uP(C_R)$ dans l'algorithme 4 ÉvaluationModulaire pour calculer $u(FG \bmod P)(C_R)$ dans le cas dense en $\mathcal{O}(d(\#P + \deg(R)))$ opérations dans $\mathbb{F}_q$. Moins formellement, il suffit de dire qu'une opération dans $\mathbb{A}_{\text{ext}}$ dans le théorème 2.30 (page 48) correspond maintenant au produit d'un vecteur par C_R.

Ceci nous permet de décrire un algorithme de vérification de produit modulaire ne passant pas par une extension de corps finis.

Algorithme 16 VérificationSansExtension

Entrée : F, G et $P \in \mathbb{F}_q[x]$, avec P unitaire $\deg(F), \deg(G) < d = \deg(P)$; $0 < \epsilon < 1$.
Sortie : Vrai si $FG \bmod P = 0$, faux avec une probabilité d'au moins $1 - \epsilon$ dans le cas contraire.

1: $\epsilon_1 \leftarrow \frac{1}{5}$, $k \leftarrow \lceil \log_q \frac{2d}{\epsilon_1} \rceil$
2: **Répéter** $(\log \frac{1}{\epsilon}) / \log(\frac{1}{2} + 2\epsilon_1)$ fois
3: $R \leftarrow$ un polynôme irréductible de degré k, avec une probabilité d'au moins $1 - \epsilon_1$ ▷ fait 1.16
4: $u \leftarrow$ vecteur aléatoire dans $\{0,1\}^k$.
5: Calculer $u((FG \bmod P)(C_R))$ ▷ théorème 2.30 et remarque 4.29
6: **Si** ce n'est pas le vecteur nul
7: **Renvoyer faux**
8: **Renvoyer vrai**

Théorème 4.30. *Soit F, G et P des polynômes dans $\mathbb{F}_q[x]$, tels que P est unitaire et $\deg(F), \deg(G) < d = \deg(P)$. Il est possible de vérifier si $FG \bmod P = 0$ en effectuant $\mathcal{O}(\#Pd + d \log_q d \log \frac{1}{\epsilon})$ opérations dans $\mathbb{F}_q$ et ce avec une probabilité d'erreur d'au plus ϵ si $FG \bmod P \neq 0$.*

Démonstration. L'algorithme est décrit dans l'algorithme 16 VérificationSansExtension et s'effectue en deux étapes. Tout d'abord, il calcule avec une probabilité d'au moins $1-\epsilon_1$ un polynôme R irréductible et de degré $k = \lceil \log_q \frac{2d}{\epsilon_1} \rceil$ en s'appuyant sur le fait 1.16. Ensuite il calcule $u(FG \bmod P)(C_R)$ pour un vecteur aléatoire $u \in \{0,1\}^k$. Ces étapes sont répétées $(\log \frac{1}{\epsilon}) / \log(\frac{1}{2} + 2\epsilon_1)$ fois.

Si $FG \bmod P = 0$, toutes les évaluations sont nulles et l'algorithme répond toujours

vrai. Supposons que $\Delta := FG \bmod P \neq 0$. Pour que l'algorithme se trompe et renvoie vrai, il faut qu'à chaque répétition $u\Delta(C_R) = 0$. Ceci peut arriver soit si R divise Δ, et alors $H\Delta(C_R) = 0$, soit si $u\Delta(C_R) = 0$ alors que R ne divise pas Δ. La probabilité que R, polynôme irréductible de degré k divise Δ est d'au plus $\frac{2d}{q^k} \leq \epsilon_1$ d'après le fait 2.5. En prenant en compte la probabilité que R ne soit pas irréductible, la probabilité que R divise Δ est d'au plus $2\epsilon_1$. En appliquant l'argument de Freivalds, si R ne divise pas Δ, la probabilité que $u\Delta(C_R) = 0$ est d'au plus $\frac{1}{2}$. Ainsi une répétition renvoie vrai à tort avec probabilité d'au plus $\frac{1}{2} + 2\epsilon_1 < 1$. Par conséquent, la probabilité que $\log \frac{1}{\epsilon} / \log(\frac{1}{2} + 2\epsilon_1)$ répétitions indépendantes renvoient toutes vraies est d'au plus ϵ.

Analysons maintenant la complexité de l'algorithme. Comme ϵ_1 est une constante, l'étape 5 effectue $\mathcal{O}(\#Pd + d \log_q d)$ opérations dans $\mathbb{F}_q$ d'après la remarque 4.29. L'étape 3 a elle un coût négligeable même si les opérations sur les polynômes sont effectuées avec un algorithme naïf. Dans cette complexité, le terme $\#Pd$ vient du coût de l'algorithme 3 CoefficientsDominants (page 47) appelé par l'algorithme 4 ÉvaluationModulaire. Puisque cet algorithme est déterministe et ne dépend que de P et de F, il peut n'être effectué qu'une seule fois plutôt qu'à chaque répétition. Ainsi la complexité totale est de $\mathcal{O}(\#Pd + d \log_q d \log \frac{1}{\epsilon})$. □

Remarque 4.31. Contrairement à l'algorithme évaluant en un point α d'une extension $\mathbb{F}_{q^s}$, aucune opération ne dépend d'ϵ. Si ϵ est une constante, cette méthode remplace un facteur $\mathsf{M}(\log_q n)$ dans la complexité par un facteur $\log_q n$. De plus les calculs effectués sont plus simples : additions de vecteurs, multiplication d'un vecteur par un scalaire et produit matrice vecteur avec une matrice compagnon. Enfin, si $\#P$ et ϵ sont des constantes, la vérification est toujours plus rapide que le coût du produit modulaire et ce pour tout q.

Cette nouvelle approche effectue encore quelques produits de polynômes, mais ceux-ci peuvent être effectués par des algorithmes naïfs sans augmenter la complexité asymptotique. L'origine de ces produits de polynômes se trouve dans les calculs de pgcd utilisés pour s'assurer que le polynôme R a une bonne probabilité d'être un polynôme irréductible. Pour qu'il n'y ait plus aucun produit de polynômes, il suffirait de ne pas s'assurer que R est bien irréductible et de juste évaluer en C_R pour un R aléatoire. Le choix d'un certain nombre de polynômes aléatoires permet alors d'atteindre une probabilité désirée qu'au moins l'un d'entre soit irréductible.

Corollaire 4.32. *Soit F, G et P des polynômes dans $\mathbb{F}_q[x]$, tels que P est unitaire et $\deg(F), \deg(G) < d = \deg(P)$. Il est possible de vérifier si $FG \bmod P = 0$ sans effectuer aucun produit de polynômes mais en effectuant $\mathcal{O}(\#Pd + d(\log_q d)^2 \log \frac{1}{\epsilon})$ opérations dans $\mathbb{F}_q$ et avec une probabilité d'erreur d'au plus ϵ lorsque $FG \bmod P \neq 0$.*

Démonstration. Dans la preuve du théorème 4.30, il faut remplacer une évaluation en C_R avec R qui est irréductible avec une probabilité d'au moins $1 - \epsilon_1$ par plusieurs évaluations sur différents C_R pour des polynômes aléatoires R unitaires et de degré k. Comme la probabilité qu'un tel polynôme soit irréductible est d'au moins $\frac{1}{2k}$ d'après le

fait 1.15, il faut générer $\mathcal{O}(k \log \frac{1}{\epsilon_1})$ polynômes aléatoires R pour qu'avec une probabilité d'au moins $1 - \epsilon_1$, l'un d'entre eux au moins soit irréductible. Ainsi la partie évaluation de l'algorithme est répétée $\mathcal{O}(k) = \mathcal{O}(\log_q d)$ fois puisque ϵ_1 est une constante. □

Ce dernier corollaire n'améliore pas la complexité de la vérification par l'évaluation en un point d'une extension $\mathbb{F}_{q^s}$ en utilisant seulement des produits de polynômes naïfs. Cependant il montre qu'il est possible de se passer totalement des algorithmes de produit de polynômes pour tester une égalité sur un produit modulaire. Ce résultat peut sembler sans intérêt au premier abord mais il sera utilisé en section 4.3.1 pour fournir une vérification efficace du produit de polynômes. Notamment car dans cette application le degré d du modulo P sera plus petit que celui des entrées.

Cette méthode s'appuyant sur une matrice compagnon peut aussi être utilisée dans le cas creux, mais pas avec la même efficacité. Les puissances α^t dans l'algorithme 6 ÉvaluationModulaireCreuse (page 57) sont remplacées par C_R^t. En revanche, il n'est pas nécessaire de calculer toutes les puissance C_R^t pour $1 < t < d$ mais seulement certaines d'entre elles. Malheureusement, ne bénéficiant plus du schéma de Horner pour la propager, la projection par un vecteur u n'entraîne plus de gain d'efficacité. En effet, en se reposant à la fois sur l'exponentiation rapide et la structure des puissances des matrices compagnons [GL80] permet déjà d'atteindre une complexité de $\mathcal{O}((\#F\#P^{\lceil \frac{1}{\gamma} - 1 \rceil} + \#G) \log d \log_q^2 d)$ opérations dans $\mathbb{F}_q$ pour évaluer $FG \bmod P$ en C_R. La projection par un vecteur aléatoire ferait perdre les avantages apportés par l'exponentiation rapide. Ainsi, puisque la technique de Freivalds n'est plus utile, il est possible d'atteindre une meilleure probabilité de succès en une seule évaluation. En générant $\mathcal{O}(\log \frac{d}{\epsilon} \log \frac{1}{\epsilon})$ polynômes R aléatoirement, au moins l'un d'entre eux est irréductible et ne divise pas $FG \bmod P$ avec une probabilité d'au moins $1 - \epsilon$. Ceci permet, dans le cas où F et G sont creux, de construire un algorithme qui n'effectue aucun produit de polynômes. Cette version n'est cependant pas aussi rapide que celui reposant sur l'évaluation dans une extension. Nous la mentionnons seulement par soucis de complétude.

Corollaire 4.33. *Soientt $P \in \mathbb{F}_q[x]$ un polynôme unitaire de degré d, F et $G \in \mathbb{F}_q[x]$ des polynômes de degré au plus $d - 1$ ayant au plus T monômes non nuls, et $0 < \epsilon < 1$. En utilisant directement l'évaluation sur une matrice compagnon, il est possible de vérifier si $FG \bmod P = 0$ avec une probabilité d'erreur d'au plus ϵ lorsque $FG \bmod P \neq 0$ en effectuant $\mathcal{O}(T\#P^{\lceil \frac{1}{\gamma} - 1 \rceil} \log d (\log_q \frac{d}{\epsilon})^3 \log \frac{1}{\epsilon})$ opérations dans $\mathbb{F}_q$ et sans effectuer de produit de polynômes.*

4.3. Vérification d'un produit de polynômes

Cette section est consacrée au problème plus d'un simple de la vérification d'un produit de polynômes. La question est donc étant donnés trois polynômes F, G et R dans $\mathbb{A}[x]$ de degré au plus d, a-t-on $FG = R$?

L'approche classique pour vérifier cette égalité consiste simplement à vérifier si $R(\alpha) =$

$F(\alpha)G(\alpha)$ pour un point α choisi aléatoirement dans un ensemble $\mathcal{E}$ suffisamment grand. Une telle stratégie ne fournit cependant pas une complexité binaire optimale et ce que les polynômes soient denses ou creux.

Si les polynômes sont denses, la vérification par l'évaluation en un point de $\mathcal{E}$ nécessite un nombre linéaire en d d'opérations dans $\mathcal{E}$. Si $\mathbb{A}$ a plus de d éléments, considérer un ensemble $\mathcal{E} \subset \mathbb{A}$ est suffisant pour que l'évaluation ait une probabilité non nulle de fournir un résultat pertinent. La vérification effectue donc un nombre linéaire en d d'opérations dans $\mathbb{A}$. Ce coût est à comparer avec celui du produit dans $\mathbb{A}[x]$ qui n'est pas linéaire (voir fait 1.6) mais quasi-linéaire en terme de complexité arithmétique. Pour la complexité binaire, le produit de deux polynômes de degré d est quasi-linéaire, en $\mathcal{O}(d \log q \log(d \log q))$, sur un corps fini $\mathbb{F}_q$.

L'évaluation fournit quant à elle une approche un peu plus rapide, en $\mathcal{O}(d \log q \log\log q)$ opérations binaires, mais pas encore optimale. Cette complexité ne tient que pour les corps finis suffisamment grands. Comme nous l'avons vu dans la section précédente, si q est trop petit, il faut se placer dans une extension. Cependant, les opérations qui sont alors effectuées dans cette extension ne correspondent plus à des opérations dans $\mathbb{F}_q$ mais à des opérations sur des polynômes. Le degré de ces polynômes n'est pas négligeable, et le nombre d'opérations effectuées dans $\mathbb{F}_q$ n'est même plus linéaire en d dans ce cas. Par exemple, dans $\mathbb{F}_2$, il est nécessaire de considérer une extension de degré supérieure à $\log d$ pour avoir plus de d points. Le coût binaire d'une opération dans cette extension est alors $\mathcal{O}(\log d \log\log d)$. L'évaluation nécessitant d opérations de ce type, l'utiliser pour la vérification entraînerait un coût binaire en $\mathcal{O}(d \log d \log\log d)$, supérieur au coût du produit lui-même.

Si la vérification par l'évaluation peut sembler optimale en complexité arithmétique, elle ne l'est absolument pas en complexité binaire. Pour surmonter cette difficulté, il est possible de s'appuyer sur une approche différente proposée par Kaminski [Kam89] qui revient à évaluer simultanément sur toutes les racines i-ème de l'unité pour un i choisi aléatoirement dans un intervalle approprié d'entiers inférieurs à d. Sa méthode consiste à effectuer le produit des polynômes mais dans $\mathbb{A}[x]/(x^i - 1)$ plutôt que dans $\mathbb{A}[x]$. Ainsi il est possible d'atteindre une complexité arithmétique optimale avec un nombre linéaire en d d'opérations dans $\mathbb{A}$ quelle que soit la taille de l'anneau. Ces opérations en revanche ne sont pas toujours linéaires quant à leur complexité binaire. L'algorithme ainsi obtenu n'est donc pas encore optimal. Dans la section 4.3.1, cette approche est présentée avec une analyse précise de la complexité binaire absente de l'article originel. Cette analyse permet de montrer qu'une telle approche peut être optimale même en terme de complexité binaire pour tous les polynômes dans $\mathbb{Z}[x]$ et des polynômes dans $\mathbb{F}_q[x]$ s'ils satisfont une contrainte sur q et d.

Si les polynômes F, G et R sont donnés sous forme creuse, il est plus aisé de voir que l'évaluation en un point aléatoire ne peut pas fournir un algorithme de vérification optimal. En effet, si les polynômes ont au plus T monômes non nuls, les évaluer en un point de $\mathcal{E} \subset \mathbb{A}$ nécessitera $\mathcal{O}(T \log d)$ opérations dans $\mathbb{A}$. Or la taille des polynômes est donnée par la taille de leurs exposants *plus* la taille de leurs coefficients, c'est à

dire $\mathcal{O}(T(\log d + B))$ bits où B désigne le nombre de bits nécessaires pour représenter un élément de $\mathbb{A}$. La complexité binaire de l'évaluation contient au moins un facteur $(T \log d)B$. Une telle approche pour la vérification n'est donc même pas quasi-linéaire, transformant une somme en un produit. Pour qu'il y ait une bonne probabilité de succès, il faut que $\mathcal{E}$ ait plus de d éléments et donc que ceux-ci soit représentés sur au moins $\log d$ bits ce qui montre que ce facteur $(\log d)B$ est en particulier quadratique en $\log d$. Et seul le cas favorable le plus simple a été considéré. L'anneau $\mathbb{A}$ pourrait être trop petit et l'évaluation devrait être effectuée dans une extension. L'anneau $\mathbb{A}$ pourrait aussi être trop grand avec des éléments non bornés et la taille binaire d'une évaluation pourrait alors être de dB bits si les coefficients des polynômes ont B bits. La section 4.3.2 est consacrée à une méthode publiée dans [GGP20], pour effectuer la vérification d'un produit de polynôme creux avec une complexité binaire quasi-linéaire.

4.3.1. Cas d'un produit de polynômes denses : algorithmes optimaux

Dans [Kam89], Kaminski propose un algorithme pour vérifier un produit de polynômes denses $R = FG \in \mathbb{A}[x]$ en utilisant un nombre linéaire d'opérations dans $\mathbb{A}$, et sans que cela dépende de la taille de $\mathbb{A}$. Son approche consiste à choisir aléatoirement un polynôme P de la forme $x^i - 1$ qui probablement ne divise pas le polynôme $\Delta = R - FG$ dans le cas où $\Delta \neq 0$. Ensuite, l'égalité $R = FG \in \mathbb{A}[x]/\langle P \rangle$ est vérifiée en utilisant des algorithmes rapides de produits de polynômes. Pour atteindre un nombre quasi-linéaire d'opérations dans $\mathbb{A}$, il fait le choix dans son algorithme d'un $i = o(d)$, ce qui est suffisant.

4.3.1.1. L'algorithme de Kaminski

La première étape est le choix aléatoire du polynôme $P = x^i - 1$. En prenant une telle forme de polynôme, Kaminski s'assure de deux propriétés importantes. Tout d'abord, la réduction d'un polynôme de degré d modulo un tel P a un coût linéaire : $\mathcal{O}(d)$ additions dans $\mathbb{A}$. Par ailleurs, en s'appuyant sur le caractère cyclotomiques des diviseurs de P, il est possible de prouver que le polynôme $\Delta = R - FG$ n'a qu'un nombre limité de diviseurs de cette forme. D'après le théorème 2.7, il est possible de choisir i aléatoirement dans l'intervalle $[d^{1-e}, 2d^{1-e}[$ pour une constante $0 < e < \frac{1}{2}$.

L'algorithme 17 VérificationdeKaminski fournit la description précise de cette approche.

Théorème 4.34 ([Kam89])**.** *Soit F, G et $R \in \mathbb{A}[x]$ des polynômes de degré au plus d, d et $2d$, et $0 < e < \frac{1}{2}$. L'algorithme 17* VérificationdeKaminski *effectue $\mathcal{O}(d)$ opérations dans $\mathbb{A}$, et sa probabilité d'échec dans le cas où $R \neq FG$ est d'au plus $(k-1)/d^{1-e}$ avec $k = \lceil 4d^e \ln \ln(d^{1-e}) \rceil$.*

Remarque 4.35. Pour être plus précis, l'algorithme 17 VérificationdeKaminski nécessite $\mathcal{O}(d)$ additions dans $\mathbb{A}$ à l'étape 2 pour calculer les trois premières réductions modulo $x^i - 1$, $\mathsf{M}(d^{1-e})$ opérations dans $\mathbb{A}$ pour effectuer le produit de polynômes à l'étape 3 et $\mathcal{O}(d^{1-e})$ additions dans $\mathbb{A}$ pour la dernière réduction.

Algorithme 17 VérificationdeKaminski

Entrée : F, G, $R \in \mathbb{A}[x]$ de degré d, d et $2d$ et $0 < e < \frac{1}{2}$.
Sortie : Vrai si $R = FG$, dans le cas contraire faux avec une probabilité d'au moins $1 - (\lceil 4d^e \ln\ln(d^{1-e})\rceil - 1)/d^{1-e}$.

1: $i \leftarrow$ un entier choisi aléatoirement dans $[d^{1-e}, 2d^{1-e}[$
2: $F_i, G_i, R_i \leftarrow F \bmod x^i - 1, G \bmod x^i - 1, R \bmod x^i - 1$
3: $M \leftarrow F_i G_i$ ▷ Par un algorithme de produit rapide
4: $M_i \leftarrow M \bmod x^i - 1$
5: **Renvoyer vrai** si $M_i = R_i$, **faux** sinon

Dans l'algorithme 17 VérificationdeKaminski, il est précisé que l'algorithme pour effectuer le produit de polynômes à l'étape 3 doit être rapide. La contrainte plus exacte est qu'il doit assurer que $\mathsf{M}(d^{1-e}) = \mathcal{O}(d)$ pour que le coût reste linéaire en d. Puisque $e < \frac{1}{2}$, l'algorithme de produit doit être au moins sous-quadratique. L'efficacité requise pour cet algorithme dépend du choix du paramètre e. Plus e est grand moins l'algorithme a besoin d'être efficace. En particulier, si e est proche de $\frac{1}{2}$, l'algorithme de Karatsuba est suffisant pour atteindre un nombre linéaire d'opérations dans $\mathbb{A}$.

La probabilité d'échec est en $\mathcal{O}((\log\log d)/(d^{1-2e}))$, d'où la nécessité d'avoir $e < 1/2$ pour qu'elle soit inférieure à 1. Cette probabilité peut être bornée par $\mathcal{O}(\frac{1}{d^{e'}})$ pour tout nombre positif $e' < 1 - 2e$. Pour atteindre une probabilité d'erreur ϵ qui puisse être arbitrairement faible, il convient de répéter l'algorithme $\mathcal{O}(\log_d \frac{1}{\epsilon})$ fois. Ce nombre de répétitions reste constant si c'est une probabilité d'échec inversement polynomiale $\epsilon = 1/d^{\mathcal{O}(1)}$ qui est requise.

Un défaut de cette approche est de s'appuyer sur un algorithme de produit rapide pour des polynômes de degré supérieur à $\sqrt{d}$. Ainsi, la vérification optimale d'un produit de polynômes de degré d s'appuie sur un produit de polynômes de degré relativement proche de d. Dans certain cas, comme la vérification de l'implémentation d'un algorithme, s'appuyer sur le problème même que l'on cherche à vérifier est problématique.

À partir de l'étape 3 qui effectue un produit de polynômes, le but de l'algorithme consiste simplement à vérifier si $R_i = F_i G_i \bmod x^i - 1$ de manière déterministe. Il est aisé de voir que ces étapes peuvent être remplacées par la vérification probabiliste d'un produit de polynômes présentée dans la section 4.2. Pour des polynômes sur les entiers ou un corps fini, il est alors possible de se passer totalement de produits de polynômes lors de la vérification.

Corollaire 4.36. *Pour $\mathbb{A} = \mathbb{Z}$ ou $\mathbb{F}_q$, et F, G et $R \in \mathbb{A}[x]$ de degrés d, d et $2d$. Il est possible de vérifier si $R = FG$ avec une probabilité d'échec dans le cas où $R \neq FG$ d'au plus ϵ. Cette vérification nécessite $\mathcal{O}(d \log_d \frac{1}{\epsilon})$ additions dans $\mathbb{A}$ plus $o(d \log_d \frac{1}{\epsilon})$ produits dans $\mathbb{A}$, et ne repose sur aucun produit de polynômes dans $\mathbb{A}[x]$. En particulier, le nombre d'opérations dans $\mathbb{A}$ est optimal si $\epsilon = 1/d^{\mathcal{O}(1)}$.*

Démonstration. Les trois dernières étapes de l'algorithme VérificationdeKaminski

sont remplacées par la vérification d'un produit modulaire avec une probabilité d'échec d'au plus $1/d$. Sur $\mathbb{Z}$ ou un corps finis $\mathbb{F}_q$ suffisamment grand, la complexité de cette seconde partie est donnée par la version dense du corollaire 4.18 avec $\#P = 2$, $m = 2$ et degré $i < 2d^{1-e}$. Pour des petits corps finis, il convient de s'appuyer sur le corollaire 4.32 et sa généralisation naturelle à la somme de produits d'au plus deux polynômes. Dans les deux cas, la probabilité d'échec d'au plus $1/d$ peut-être atteinte et la valeur de $i < 2d^{1-e}$ permet d'assurer que le nombre d'opérations dans $\mathbb{A}$ reste bien en $o(d)$. Ce nombre est en effet $\mathcal{O}(i)$ dans le premier cas et $\mathcal{O}(i(\log d)^3)$ dans le second.

La probabilité d'échec de l'algorithme modifié est ainsi bornée par $1/d + \mathcal{O}(1/d^{e'}) = \mathcal{O}(1/d^{e'})$ pour un $0 < e' < 1 - 2e$. En répétant cet algorithme modifié $\mathcal{O}(\log_d \frac{1}{\epsilon})$ fois, il est alors possible d'atteindre une probabilité d'échec inférieure à ϵ. □

4.3.1.2. Analyse de la complexité binaire

Dans [Kam89], Kaminski ne fournit que la complexité arithmétique de son algorithme sans développer la complexité binaire. Or en analysant précisément cette complexité sur des corps finis ou $\mathbb{Z}$, il est possible de se rendre compte que dans de nombreux cas, l'algorithme est aussi linéaire en nombre d'opérations binaires. Dans le cas de polynômes dans $\mathbb{F}_q[x]$ pour que l'algorithme ne soit pas quasi-linéaire il faut que q soit doublement exponentiel dans le degré d. Pour les polynômes dans $\mathbb{Z}[x]$, une borne similaire s'applique. Cependant, il est possible de décrire une variante de l'algorithme qui a une complexité binaire linéaire même pour des polynômes ayant de très grands coefficients. Pour cela, il convient de s'appuyer sur l'algorithme linéaire de vérification d'un produit d'entiers présenté par Kaminsky dans le même article [Kam89]. Ainsi, la vérification d'un produit de polynômes à coefficients entiers est optimale dans tous les cas.

Tout d'abord, analysons la complexité binaire de l'algorithme de Kaminski sur un corps fini. Ici, comme dans le reste de la thèse, les analyses sont effectuées en supposant l'existence d'une constante de Linnik qui permette d'avoir $\mathsf{M}_q(d) = \mathcal{O}(d \log q(\log d + \log\log q))$ opérations binaires [HH22]. Si cette constante n'existait pas, il faudrait simplement ajouter un facteur $4^{\log^*(d)}$ à la complexité. Dans l'analyse de l'optimalité il faudrait alors considérer $d^e 4^{\log^*(d)}$ plutôt que d^e. Ceci alourdirait les notations sans changer significativement les bornes obtenues.

Théorème 4.37. *Soit F, G et $R \in \mathbb{F}_q[x]$ de degrés d, d et $2d$, et $0 < e < \frac{1}{2}$. L'algorithme* VérificationdeKaminski *effectue $\mathcal{O}(d \log q + d^{1-e} \log q \log\log q)$ opérations binaires. Quand $\log\log q = \mathcal{O}(d^e)$, il est possible de vérifier si $R = FG$ avec une probabilité d'échec dans le cas où $R \neq FG$ d'au plus ϵ, en $\mathcal{O}(d \log q \log_d \frac{1}{\epsilon})$ opérations binaires. La complexité est optimale si $\epsilon = 1/d^{\mathcal{O}(1)}$.*

Démonstration. Il s'agit d'appliquer le compte détaillé des opérations de l'algorithme de Kaminski donné dans la remarque 4.35. Les additions donnent le terme $\mathcal{O}(d \log q)$. Pour le produit de polynômes de degré $\mathcal{O}(d^{1-e})$ sur $\mathbb{F}_q$ la complexité binaire est $\mathsf{M}_q(d^{1-e}) = \mathcal{O}(d^{1-e} \log q \log(d \log q))$. Cette complexité se décompose en deux termes : d'une part

$\mathcal{O}(d^{1-e}\log d\log q)$ dominé par le coût des additions et d'autre part $\mathcal{O}(d^{1-e}\log q\log\log q)$ qui est en $\mathcal{O}(d\log q)$ dès que q vérifie $\log\log q = \mathcal{O}(d^e)$. Ce qui permet d'obtenir les deux complexités annoncées, en tenant compte des $\mathcal{O}(\log_d \frac{1}{\epsilon})$ répétitions nécessaires pour atteindre une probabilité d'échec inférieure à ϵ. □

La borne $\log\log q = \mathcal{O}(d^e)$ pour obtenir un nombre linéaire d'opérations binaires n'est valide que si l'algorithme utilisé pour le produit de polynôme est l'algorithme le plus rapide qui soit connu. Si c'est un algorithme sous-quadratique mais moins efficace qui est utilisé, cette borne devient plus petite. Par exemple, s'il s'agit de l'algorithme de Karatsuba, le produit de polynômes de degré $\mathcal{O}(d^{1-e})$ nécessite $\mathcal{O}(d^{(1-e)\log 3}\log q\log\log q)$ opérations binaires. Pour que l'algorithme VérificationdeKaminski soit optimal en utilisant ce produit, il faut donc que $d^{(1-e)\log 3}\log\log q = \mathcal{O}(d)$. Ce qui implique donc d'une part que $e \geq 1 - 1/\log 3 \simeq 0.367$ et d'autre part que la borne sur q devienne $\log\log q = \mathcal{O}(d^{1-(1-e)\log 3})$. Pour un paramètre $e = 0.45$ proche de $1/2$, cette borne devient $\log\log q = \mathcal{O}(d^{0.13})$ alors que c'est $\log\log q = \mathcal{O}(d^{0.45})$ avec l'algorithme de produit le plus efficace.

Cependant, il a déjà été mentionné qu'utiliser un algorithme de produit de polynômes rapide pour vérifier un produit de polynômes n'est pas complètement satisfaisant. Il convient donc d'analyser aussi la complexité binaire de la variante de l'algorithme 17 VérificationdeKaminski sans produit de polynômes du corollaire 4.36. Pour cet algorithme modifié, la même complexité et la même borne sur q sont obtenues sans effectuer de produit de polynômes.

Remarque 4.38. Soit $F, G, R \in \mathbb{F}_q[x]$ de degrés d, d et $2d$, et $0 < e < \frac{1}{2}$. L'algorithme 17 VérificationdeKaminski peut être implémenté en utilisant la vérification d'un produit modulaire et sans effectuer de produits de polynômes. Une telle variante a une complexité binaire de $\mathcal{O}(d\log q + d^{1-e}\log q\log\log q)$. Quand $\log\log q = \mathcal{O}(d^e)$, il est possible de vérifier si $R = FG$ avec une probabilité d'échec dans le cas où $R \neq FG$ d'au plus ϵ, en $\mathcal{O}(d\log q\log_d \frac{1}{\epsilon})$ opérations binaires ce qui est optimal si $\epsilon = 1/d^{\mathcal{O}(1)}$ et ce sans aucun produit de polynômes sur $\mathbb{F}_q$.

Démonstration. La preuve est similaire à celle du théorème 4.37 mais en s'appuyant sur le corollaire 4.32 plutôt que sur la remarque 4.35. □

Il reste à considérer le cas où F, G et R sont des polynômes définis dans $\mathbb{Z}[x]$ avec une borne H sur la valeur absolue de leurs coefficients. L'analyse de la complexité binaire de l'algorithme 17 VérificationdeKaminski donne des conditions pour que celle-ci soit optimale.

Théorème 4.39. *Soit F, G et $R \in \mathbb{Z}[x]$ de degrés d, d et $2d$, et hauteurs au plus H, et une probabilité $0 < e < \frac{1}{2}$. L'algorithme 17* VérificationdeKaminski *effectue $\mathcal{O}(d\log H + d^{1-e}\log H\log\log H)$ opérations binaires. Quand $\log\log H = \mathcal{O}(d^e)$, il est possible de vérifier si $R = FG$ avec une probabilité d'échec dans le cas où $R \neq FG$*

d'au plus ϵ, en $\mathcal{O}(d \log H \log_d \frac{1}{\epsilon})$ opérations binaires. Cette complexité est optimale si $\epsilon = 1/d^{\mathcal{O}(1)}$.

Démonstration. Les trois premières réductions effectuent $\mathcal{O}(d)$ additions dans $\mathbb{Z}$ pour calculer F_i, G_i et R_i, dont les hauteurs sont bornées par $d^e H$. Si ces additions sont effectuées soigneusement en utilisant un arbre binaire pour contrôler la croissance des hauteurs, une réduction nécessite en fait $\mathcal{O}(\sum_{i=1}^{\log d} \frac{d}{2^i} \log(iH) = \mathcal{O}(d \log H)$ opérations binaires. Le produit de polynômes est alors effectué avec des polynômes de degré $< 2d^{1-e}$ et hauteur $d^e H$. Il est donc calculé en $\mathsf{I}(d^{1-e}(\log d^{1-e} + \log(d^e H)))$ opérations binaires (fait 1.6), c'est à dire $\mathcal{O}(d^{1-e}(\log d + \log H)(\log d + \log\log H)) = \mathcal{O}(d \log H + d^{1-e} \log H \log\log H)$. La dernière réduction quant à elle s'applique à un polynôme de degré $< 4d^{1-e}$ et hauteur bornée par $d(d^e H)^2$. Elle nécessite donc $\mathcal{O}(d^{1-e}(\log d + \log H) = \mathcal{O}(d \log H)$ opérations binaires.

La répétition $\mathcal{O}(\log_d \frac{1}{\epsilon})$ fois de l'algorithme fournit la seconde partie du théorème avec une probabilité d'échec inférieure à ϵ. □

Comme dans le cas des polynômes sur les corps finis, la fin de l'algorithme peut être remplacée par la vérification d'un produit modulaire de polynômes à coefficients entiers. Il se trouve que dans ce cas la complexité obtenue est légèrement meilleure. La borne sur H est donc encore moins contraignante pour que l'algorithme soit optimal.

Remarque 4.40. Soit F, G et $R \in \mathbb{Z}[x]$ de degrés d, d et $2d$, et hauteurs au plus H, et $0 < e < \frac{1}{2}$. L'algorithme 17 VérificationdeKaminski peut être implémenté en utilisant la vérification d'un produit modulaire et sans effectuer de produits de polynômes. Une telle variante a une complexité binaire de $\mathcal{O}(d \log H + d^{1-e}(\log H)(\log\log\log H))$. Quand $\log\log\log H = \mathcal{O}(d^e)$, il est possible de vérifier si $R = FG$ avec une probabilité d'échec dans le cas où $R \neq FG$ d'au plus ϵ, en $\mathcal{O}(d \log H \log_d \frac{1}{\epsilon})$ opérations binaires ce qui est optimal si $\epsilon = 1/d^{\mathcal{O}(1)}$, et ce sans aucun produit de polynômes.

Démonstration. La preuve est similaire à celle de l'algorithme utilisant le produit de polynôme. Il s'agit juste de remplacer le coût du produit par celui de sa vérification avec des polynômes de degré au plus $2d^{1-e}$ et hauteur au plus $d^e H$. La partie dense du théorème 4.20 donne le coût binaire d'une telle vérification : $\mathcal{O}(d^{1-e}(\mathsf{I}(\log(d \log H) + (\log H) \log\log(d \log H))))$ ce qui est $\mathcal{O}(d \log H + d^{1-e}(\log H)(\log\log\log H))$. □

Tant que les coefficients ne prennent pas des valeurs excessivement élevées par rapport au degré, la remarque précédente indique que la vérification d'un produit de polynômes entiers est optimale. Plus précisément, la remarque s'applique à des polynômes dont la hauteur H peut aller d'une valeur constante $\mathcal{O}(1)$ jusqu'à $2^{2^{2^{\mathcal{O}(d)}}}$. Pour gérer les coefficients qui dépassent cette borne, il est possible de passer par une autre approche qui se ramène à la vérification du produit d'entiers. Cette approche ne concerne pas que les polynômes ayant des coefficients gigantesques. Elle est aussi valide dès que $\log d = \mathcal{O}(\log H)$. Ainsi la vérification d'un produit de polynômes à coefficients entiers peut toujours être

effectuée avec une complexité binaire optimale. Pour des hauteurs H comprises entre $d^{\mathcal{O}(1)}$ et $2^{2^{2^{\mathcal{O}(d)}}}$, il existe deux méthodes optimales, ce qu'il est intéressant de remarquer pour pouvoir chercher l'implémentation la plus efficace.

Pour traiter le cas des grands coefficients, la méthode proposée ici s'appuie sur l'algorithme de vérification d'un produit d'entiers de Kaminski. La technique pour cette vérification est similaire à celle utilisée pour un produit de polynômes. Il s'agit de commencer par réduire des entiers de s bits modulo $2^i - 1$ pour un i choisi aléatoirement entre s^{1-e} et $2s^{1-e}$ et ensuit d'effectuer le produit entre les entiers réduits pour vérifier s'il y a égalité modulo $2^i - 1$.

Théorème 4.41 ([Kam89])**.** *Soit a, b, c des entiers d'au plus s, s et $2s$ bits, $0 < e < \frac{1}{2}$ et $k = \lceil 4s^e \ln\ln(s^{1-e}) \rceil$. Il est possible de vérifier si $ab = c$ en $\mathcal{O}(s)$ opérations binaires et avec une probabilité d'erreur d'au plus $(k-1)/s^{1-e}$ dans le cas où $ab \neq c$.*

La vérification du produit de polynômes $R = FG$ dans $\mathbb{Z}[x]$ peut se ramener à la vérification d'un produit d'entiers en utilisant une transformation de Kronecker. Si les polynômes sont évalués sur une puissance de 2 suffisamment grande β, les coefficients de FG correspondent directement aux chiffres de l'entier $F(\beta)G(\beta)$ dans la base β. L'évaluation en β, une puissance de 2, d'un polynôme dans $\mathbb{Z}[x]$ ne nécessite aucune opération. La vérification de $R = FG$ dans $\mathbb{Z}[x]$ se réduit donc facilement à la vérification de $R(\beta) = F(\beta)G(\beta)$ dans $\mathbb{Z}$.

Théorème 4.42. *Soit F, G, $R \in \mathbb{Z}[x]$ de degrés d, d et $2d$, et hauteur au plus H. Si $\log d = \mathcal{O}(\log H)$, il est possible de vérifier si $R = FG$ avec une probabilité d'échec dans le cas où $R \neq FG$ d'au plus ϵ, en $\mathcal{O}(d \log H \log_{d \log H} \frac{1}{\epsilon})$ opérations binaires ; Cette complexité est optimale si $\epsilon = 1/d^{\mathcal{O}(1)}$.*

Démonstration. Puisque F et G ont une hauteur d'au plus H et pour degré d, la hauteur de FG est d'au plus dH^2. Soit β la première puissance de 2 supérieure à $2dH^2$. Alors $R = FG$ si et seulement si $R(\beta) = F(\beta)G(\beta)$. Les entiers $F(\beta)$, $G(\beta)$ et $R(\beta)$ ont pour taille binaire $\mathcal{O}(d \log \beta) = \mathcal{O}(d \log(dH)) = \mathcal{O}(d \log H)$ puisque $\log d = \mathcal{O}(\log H)$. Comme β est une puissance de 2 qui borne tous les coefficients, les évaluations en β ne nécessitent aucune opération. Ainsi le coût vient seulement de la vérification du produit d'entiers $F(\beta)G(\beta) = R(\beta)$. Celle-ci est linéaire dans la taille de $F(\beta)$, $G(\beta)$ et $R(\beta)$ d'après le théorème 4.41 et donc linéaire en $d \log H$.

Pour obtenir la borne ϵ sur la probabilité d'échec, il suffit de répéter cette procédure $\mathcal{O}(\log_{d \log H} \frac{1}{\epsilon})$ fois. En effet, la probabilité d'erreur d'une seule exécution est bornée par $1/s^{\mathcal{O}(1)}$ dans le théorème 4.41 où s est la taille des entiers considérés. □

4.3.2. Cas d'un produit de polynômes creux : algorithmes quasi-linéaires

Le cas de la vérification de l'égalité $FG = R$ pour F, G et R des polynômes creux dans $\mathbb{A}[x]$ n'avait quant à lui pas été traité avant notre article de 2020 [GGP20]. Comme déjà

mentionnée, l'évaluation en un point ne peut pas fournir une approche quasi-linéaire. De manière semblable à celle de Kaminski, l'approche proposée ici consiste à considérer l'égalité modulo un polynôme P. Ainsi pour vérifier l'égalité $FG = R$, il est possible de se ramener à la vérification de l'égalite $(F \bmod x^p - 1)(G \bmod x^p - 1) \mod x^p - 1 = R \bmod x^p - 1$. Cette seconde égalité peut elle-même être vérifiée grâce aux méthodes de vérification modulaire de la section 4.2 avec $P = x^p - 1$. Dans la section 4.2 il était plus simplement question d'une égalite du type $FG \bmod P = 0$. Il a cependant été vu que cela s'étendait naturellement aux égalités de la forme $\sum_{i=0}^{m-1} F_i G_i \bmod P = 0$. Le cas considéré ici est celui d'un $m = 2$, constant, avec un second produit plus simple puisque de la forme $R \times 1$. La complexité de cette vérification revient donc à la complexité d'une vérification du type $FG \bmod P = 0$. Pour ne pas alourdir les discussions, il ne sera pas systématiquement rappelé qu'il s'agit de faire appel à une légère extension des résultats principaux concernant la vérification modulaire.

Contrairement au cas dense, la vérification du produit modulaire n'est pas utilisée seulement pour éviter de vérifier un produit de polynômes en calculant des produits de polynômes. Dans le cas particulier des polynômes creux, c'est une nécessité pour prévenir des croissances de taille propres à l'arithmétique des polynômes creux. En effet, si F a $\#F$ monômes non nuls et G en a $\#G$, le polynôme $(F \bmod x^p - 1)(G \bmod x^p - 1)$ peut en avoir jusqu'à $\#F\#G$ ce qui est alors quadratique dans la taille des polynômes donnés en entrée. Le calculer directement ne permettrait donc pas d'obtenir une complexité meilleure que quadratique.

La méthode est explicitement décrite dans l'algorithme 18 VérificationCreuse qui s'applique à tout anneau intègre $\mathbb{A}$ suffisamment grand. Dans un second temps nous présentons des analyses détaillées pour les cas où $\mathbb{A} = \mathbb{Z}$ et $\mathbb{A} = \mathbb{F}_q$ même si q est inférieur à d.

Algorithme 18 VérificationCreuse

Entrée : R, F, $G \in \mathbb{A}[x]$; $0 < \epsilon < 1$.
Sortie : Vrai si $R = FG$, faux avec une probabilité au moins $1 - \epsilon$ dans le cas contraire.

1: Définir $0 < \epsilon_1 < \frac{3}{10}$ et $0 < \epsilon_2 < 1$ tels que $\frac{10\epsilon_1}{3} + (1 - \frac{10\epsilon_1}{3})\epsilon_2 \leq \epsilon$
2: $d \leftarrow \deg(R)$
3: **Si** $\#R > \#F\#G$ ou $d \neq \deg(F) + \deg(G)$ **alors Renvoyer faux**
4: $\lambda \leftarrow \max(21, \frac{1}{\epsilon_1}(\#F\#G + \#R)\ln d)$
5: $p \leftarrow$ PremierAléa$(\lambda, \frac{5\epsilon_1}{3})$
6: $(F_p, G_p, R_p) \leftarrow (F \bmod x^p - 1,\ G \bmod x^p - 1,\ R \bmod x^p - 1)$
7: **Renvoyer vrai** si $R_p = (F_p G_p) \bmod x^p - 1$, **faux** sinon ▷ théorème 4.17 avec probabilité ϵ_2

Théorème 4.43. *Si $\mathbb{A}$ est un anneau intègre de taille $\geq \frac{2}{\epsilon_1\epsilon_2}(\#F\#G + \#R)\ln(d)$, l'algorithme 18* VérificationCreuse *permet de vérifier l'égalité $FG = R$ pour des polynômes creux dans $\mathbb{A}[x]$. Dans le cas où $FG \neq R$ la probabilité d'erreur est infé-*

rieure à $1-\epsilon$. *En posant* $d = \deg(R)$ *et* $T = \max(\#F, \#G, \#R)$, *l'algorithme effectue* $\mathcal{O}(T\log(\frac{1}{\epsilon}T\log d))$ *opérations dans* $\mathbb{A}$, *et* $\mathcal{O}(T\log d\log\log(\frac{1}{\epsilon}T\log d))$ *opérations binaires plus* $\mathcal{O}(\log\frac{1}{\epsilon}\log^3(\frac{1}{\epsilon}T\log d)\log^2\log(\frac{1}{\epsilon}T\log d))$ *opérations binaires pour obtenir le nombre premier* p.

Démonstration. L'étape 3 permet d'écarter deux cas où il n'y a trivialement pas égalité tout en s'assurant que d corresponde bien au degré de FG.

Si $R = FG$, l'algorithme répond toujours vrai. Dans le cas contraire, il y a deux sources d'erreurs. L'une vient de la possibilité d'avoir $x^p - 1$ qui divise $R - FG$. Puisque ce polynôme a au plus $\#R + \#F\#G$ monômes non nuls, la probabilité que ceci arrive est d'au plus $\frac{10\epsilon_1}{3}$ d'après le corollaire 2.9. L'autre source d'erreur est la possibilité que la vérification modulaire échoue alors que $x^p - 1$ ne divise pas $R - FG$. La probabilité d'un tel échec est d'au plus ϵ_2. Ainsi la probabilité d'échec de l'algorithme est d'au plus $\frac{10\epsilon_1}{3} + (1 - \frac{10\epsilon_1}{3})\epsilon_2 \leq \epsilon$.

Pour analyser la complexité, considérons $\epsilon_1, \epsilon_2 \sim \epsilon$ (par exemple $\epsilon_1 = \frac{3\epsilon}{20}$ et $\epsilon_2 = \frac{\epsilon}{2}$). Remarquons alors que $p = \mathcal{O}(\frac{1}{\epsilon}T^2\log d)$. Pour obtenir ce nombre premier p, l'étape 5 effectue $\mathcal{O}(\log\frac{1}{\epsilon}\log^3 p\log^2\log p)$ opérations binaires d'après le fait 1.13. Ce qui donne bien le dernier terme annoncé pour la complexité en remplaçant $\log p$ par $\mathcal{O}(\log(\frac{1}{\epsilon}T\log d))$.

L'étape 6 effectue T divisions d'entiers inférieurs à d par le même entier p plus T additions dans $\mathbb{A}$. Les divisions d'entiers nécessitent $\mathcal{O}(T\frac{\log d}{\log p}\mathsf{I}(\log p)) = \mathcal{O}(T\log d\log\log p)$ opérations binaires, autrement dit $\mathcal{O}(T\log d\log\log(\frac{1}{\epsilon}T\log d)))$ opérations binaires.

Enfin, à l'étape 7, F_p, G_p et R_p ont pour degré $p - 1 = \mathcal{O}(\frac{1}{\epsilon}T^2\log d)$ et au plus T monômes non nuls. Ce sont encore des polynômes creux. Il est donc possible d'appliquer la version creuse du théorème 4.17 avec $P = x^p - 1$. Il s'agit plus précisément d'appliquer son corollaire 4.18 avec $m = 2$. La vérification du produit $R_p = F_pG_p \bmod x^p - 1$ nécessite donc $\mathcal{O}(T\log p) = \mathcal{O}(T\log(\frac{1}{\epsilon}T\log d))$ opérations dans $\mathbb{A}$.

Le coût des autres étapes est négligeable en comparaison du coût de ces 3 étapes. □

Pour rendre les complexité plus lisibles, la notation $\mathcal{O}_\epsilon(f(d))$ sera utilisée pour désigner une complexité en $\mathcal{O}(f(d)\log^k\frac{1}{\epsilon})$ pour un certain k. Avec cette notation, la complexité de l'algorithme 18 VérificationCreuse s'écrit $\mathcal{O}_\epsilon(T\log(T\log d))$ opérations dans $\mathbb{A}$ plus $\mathcal{O}_\epsilon(T\log d\log\log(T\log d)))$ opérations binaires puisque l'obtention du nombre premier p est logarithmique en T et $\log d$ et est polynomial seulement en $\log\frac{1}{\epsilon}$.

Le reste de cette section est consacrée à l'analyse de cet algorithme sur les entiers ou les corps finis. L'objectif est d'avoir une complexité aussi proche que possible d'une complexité linéaire. Pour simplifier la comparaison entre les deux, la taille binaire des entrées est donnée avec un seul paramètre s. Habituellement, un polynôme creux est plutôt donné avec trois paramètres un pour son nombre de monômes non nuls, un pour leurs exposants et un pour leurs coefficients. Ainsi, pour un polynôme de degré d avec T

monômes non nuls, s remplace $T(\log d + \log q)$ dans $\mathbb{F}_q[x]$ ou $T(\log d + \log H)$ dans $\mathbb{Z}[x]$ si le polynôme a pour hauteur H.

Un premier point qu'il est alors facile de noter est que la réduction des polynômes modulo $x^p - 1$ à l'étape 6 n'est pas linéaire. En effet, il a été prouvé dans la preuve du théorème 4.43 (pour l'étape 6) que le calcul des nouveaux exposants nécessite $\mathcal{O}_\epsilon(T \log d \log\log(T \log d))$ opérations binaire. Ce qui donne, avec s la taille binaire des trois polynômes en entrée, $\mathcal{O}_\epsilon(s \log\log s)$. Cette étape, qui ne varie pas en fonction de l'anneau $\mathbb{A}$, peut dans certains cas être l'étape dominante en terme de complexité.

4.3.2.1. Analyse de la complexité binaire sur les entiers

Le premier anneau spécifiquement considéré est $\mathbb{A} = \mathbb{Z}$.

Corollaire 4.44. *Soit F, G et $R \in \mathbb{Z}[x]$ de degré au plus d, de hauteur au plus H et ayant au plus T monômes non nuls. Alors l'algorithme 18* VérificationCreuse *effectue $\mathcal{O}_\epsilon(s \log s \log\log s)$ opérations binaires, avec $s = T(\log d + \log H)$ la taille des entrées.*

Démonstration. La seule modification se trouve à l'étape 7, où la vérification du produit modulaire est effectuée en s'appuyant sur le théorème 4.20 avec $P = x^p - 1$ et F_p, G_p, R_p qui ont au plus T monômes dont les coefficients sont bornés par TH. Cette étape coûte donc

$$\mathcal{O}_\epsilon(T \log p \mathsf{I}(\log(p \log H)) + T \log(TH) \log\log(p \log TH)).$$

Puisque $T \leq d$ et $p = \mathcal{O}(\frac{1}{\epsilon} T^2 \log d)$, $T \log p = \mathcal{O}_\epsilon(T \log d) = \mathcal{O}_\epsilon(s)$. Par ailleurs $\log(p \log H) = \mathcal{O}_\epsilon(\log(T \log d \log H)) = \mathcal{O}_\epsilon(\log(T \log d) + \log\log H) = \mathcal{O}_\epsilon(\log s)$. Ainsi le premier terme dans la complexité de cette étape est en fait $\mathcal{O}_\epsilon(s \log s \log\log s)$. De plus, $T \log(TH) = \mathcal{O}(T \log dH) = \mathcal{O}(s)$. Comme $\log(p \log TH) = \mathcal{O}_\epsilon(\log(T \log d) + \log\log H) = \mathcal{O}_\epsilon(\log s)$, le second terme est $\mathcal{O}_\epsilon(s \log\log s)$.

Comme l'étape 6 ne change pas et a une complexité binaire en $\mathcal{O}_\epsilon(s \log\log s)$ et que l'étape 5 a comme facteurs dominants des facteurs en $\log s$ vue la valeur de p, la complexité totale est bien en $\mathcal{O}_\epsilon(s \log s \log\log s)$. □

Si les polynômes ont vraiment peu de monômes non nuls, la complexité est encore meilleure.

Remarque 4.45. Si F, G, $R \in \mathbb{Z}[x]$ de taille binaire s ont au plus $T = \Theta(\log^k d)$ monômes non nuls, pour une certaine constante k, alors l'algorithme 18 VérificationCreuse effectue $\mathcal{O}_\epsilon(s \log\log s)$ opérations binaires.

Démonstration. La taille des entrées est $s = \Theta(\log^{k+1} d + \log^k d \log H)$. Dans ce cas, $\log p = \mathcal{O}_\epsilon(\log\log d)$.

Dans la preuve précédente, il n'y avait qu'un terme en $\mathcal{O}_\epsilon(s \log s \log\log s)$ qui domine tous les autres termes qui sont au plus en $\mathcal{O}_\epsilon(s \log\log s)$. Il est donc suffisant de mon-

trer qu'avec la condition donnée sur T et d ce terme dominant est en fait lui-aussi en $\mathcal{O}_\epsilon(s \log\log s)$.

Or le terme dominant $\mathcal{O}_\epsilon(s \log s \log\log s)$ dans la preuve précédente vient d'un terme de la forme $\mathcal{O}_\epsilon(T \log p \mathsf{I}(\log(p \log H)))$. Puisqu'en tenant compte des nouvelles conditions $\log(p \log H) = \mathcal{O}_\epsilon(\log\log d + \log\log H)$, ce terme dominant devient

$$\mathcal{O}_\epsilon(\log^k d \log\log d(\log\log d + \log\log H) \log(\log\log d + \log\log H)).$$

Or $\log\log d$ et $\log\log H$ sont tous les deux en $\mathcal{O}(\log s)$. Il est donc possible de réécrire ce terme sous la forme $\mathcal{O}_\epsilon(\log^k d \log^2 s \log\log s)$. Puisque $\log^k d = \mathcal{O}(s^{k/(k+1)})$, ceci donne bien $\mathcal{O}_\epsilon(s \log\log s)$. □

4.3.2.2. Analyse de la complexité binaire sur les corps finis

Passons maintenant au cas $\mathbb{A} = \mathbb{F}_q$. Il s'agit en réalité de plusieurs cas puisque le résultat varie en fonction de la taille de q par rapport au degré et au nombre de monômes non nuls des polynômes creux. Le premier cas est celui des grands corps finis qui permet d'appliquer directement l'algorithme général avec les mêmes garanties de succès et une complexité quasi-linéaire. Pour les corps finis, il faut utiliser un algorithme de vérification modulaire spécifique. La complexité est meilleure dans ces petits corps, et encore plus si les polynômes sont très creux.

Les valeurs atteintes dans les corollaires 4.46 et 4.47 supposent l'existence d'une constante de Linnik qui permet d'avoir $\mathsf{M}_q(d) = \mathcal{O}(d \log q(\log d + \log\log q))$ [HH22]. Au contraire, la remarque 4.48 est inconditionnelle.

Corollaire 4.46. *Soit F, G et $R \in \mathbb{F}_q[x]$ de degré au plus d avec au plus T monômes non nuls et $q > \frac{2}{\epsilon_1\epsilon_2}(\#F\#G + \#R) \ln d$. L'algorithme 18* VérificationCreuse *effectue $\mathcal{O}_\epsilon(s \log^2(s))$ opérations binaires avec $s = T(\log d + \log q)$ la taille des entrées.*

Démonstration. Encore une fois, il est suffisant d'analyser l'étape 7. Chaque opération dans $\mathbb{F}_q$ recquiert $\mathcal{O}(\log(q)\log\log(q))$ opérations binaires. Ainsi le coût binaire de l'étape 7 est $\mathcal{O}_\epsilon(T \log(T \log d) \log(q) \log\log(q))$. Puisque à la fois $T \log q$ et $T \log d$ sont en $\mathcal{O}(s)$ et $\log\log q = \mathcal{O}(\log s)$, la complexité binaire est bien en $\mathcal{O}_\epsilon(s \log^2(s))$. □

Si le corps n'est pas assez grand, il est alors nécessaire de passer dans une extension. Cette contrainte va en fait améliorer la complexité. En effet l'extension choisie est alors exactement de la bonne taille pour garantir la même probabilité de succès. En comparaison un corps qui est assez grand peut s'avérer beaucoup plus grand que nécessaire.

Corollaire 4.47. *Soit F, G et $R \in \mathbb{F}_q[x]$ de degré au plus d et ayant au plus T monômes non nuls avec $q < \frac{2}{\epsilon_1\epsilon_2}(\#F\#G + \#R) \ln d$. L'algorithme 18* VérificationCreuse *effectue $\mathcal{O}_\epsilon(s \log s \log\log s)$ opérations binaires avec $s = T(\log d + \log q)$ la taille des entrées.*

Démonstration. En appliquant le corollaire 4.24, l'étape 7 effectue $\mathcal{O}_\epsilon(T \log p \mathsf{M}_q(\log_q p) + (\log_q p)^2 \mathsf{M}_q(\log_q p)(\log q + \log\log_q p))$ opérations dans $\mathbb{F}_q$. Puisque $\log p = \mathcal{O}_\epsilon(\log(T \log d)) = \mathcal{O}_\epsilon(\log s)$ et $\log q = \mathcal{O}_\epsilon(\log s)$ aussi, le second terme de la complexité de l'étape 7 dépend uniquement de $\log s$ et non de s. Comme $\log p = \mathcal{O}_\epsilon(\log(T \log d))$ le premier terme est en fait $\mathcal{O}_\epsilon(T \log(T \log d) \mathsf{M}_q(\log_q(T \log d)))$. Or $\log(T \log d) = \mathcal{O}(\log d)$ d'où $T \log(T \log d) = \mathcal{O}(s)$. De plus, $\log(T \log d) = \mathcal{O}(\log s)$ et le premier terme se simplifie donc en $\mathcal{O}_\epsilon(s \mathsf{M}_q(\log_q s))$. Or $\mathsf{M}_q(\log_q s) = \mathcal{O}(\log s \log\log s)$. Ainsi $T \log p \mathsf{M}_q(\log_q p) = \mathcal{O}_\epsilon(s \log s \log\log s)$ et c'est le terme dominant dans la complexité. □

Dans ce cas des petits corps finis, le recours à une extension et donc à des produits de polynômes est nécessaire pour obtenir la meilleure complexité possible. Il faut noter cependant que les produits de polynômes effectués pour calculer dans l'extension ne sont pas du même type que le produit que l'on cherche à vérifier. En effet, l'arithmétique de l'extension est assurée par des produits de polynôme denses, alors que le produit à vérifier est un produit de polynômes creux. Il y a par ailleurs une différence significative entre les degrés puisqu'on passe de d à essentiellement $\log_q(T \log d)$.

Encore une fois, si le polynôme est très creux, l'algorithme est plus efficace.

Remarque 4.48. Soit F, G et $R \in \mathbb{F}_q[x]$ de degré au plus d avec au plus $T = \Theta(\log^k d)$ monômes non nuls pour une certaine constante k, avec $q < \frac{2}{\epsilon_1 \epsilon_2}(\#F \#G + \#R) \ln d$. L'algorithme VérificationCreuse effectue $\mathcal{O}_\epsilon(s \log\log s)$ opérations binaires.

Démonstration. Comme vu dans la preuve précédente, le terme dominant dans la complexité de l'étape 7 pour des petits corps finis est $\mathcal{O}_\epsilon(T \log(T \log d) \mathsf{M}_q(\log_q(T \log d)))$. Il peut être borné par $\mathcal{O}_\epsilon(T \log^3(T \log d))$. Or $T = \mathcal{O}(s^{k/(k+1)})$, le terme dominant devient donc simplement $\mathcal{O}_\epsilon(s)$. La complexité de l'algorithme est donc dominée par celle de l'étape 6 à savoir $\mathcal{O}_\epsilon(s \log\log s)$. □

De manière générale, la complexité binaire de l'algorithme VérificationCreuse sur les entiers ou les corps finis est comprise entre $\mathcal{O}_\epsilon(s \log\log s)$ dans les cas les plus favorables et $\mathcal{O}_\epsilon(s \log^2 s)$ quand c'est moins favorable. Dans les meilleures cas, c'est une opération qui semble pourtant simple, la réduction modulo $x^p - 1$ des polynômes F, G et R qui domine la complexité.

Dans ce chapitre nous nous sommes concentrés sur des identités modulaires. Nous avons proposé un test déterministe pour la divisibilité dans certaines familles de polynômes et des tests probabilistes s'il s'agit simplement d'un produit modulaire. Ces résultats peuvent être vus comme une étape vers la mise au point d'algorithmes pour tester des identités polynomiales modulaires en toute généralité.

Nous les avons déjà appliqués au problème de la vérification de produit de polynômes pour obtenir des algorithmes linéaires ou quasi-linéaires. Les résultats obtenus pour les

polynômes creux nous ont permis de mettre au point les algorithmes que nous allons présenter dans le chapitre 5 pour l'arithmétique de polynômes creux.

5. Arithmétique des polynômes creux : produit et division exacte

Pour pouvoir manipuler des polynômes creux, il convient de maîtriser les opérations arithmétiques basiques : somme, produit ou division euclidienne. Si la somme correspond simplement à une fusion de listes, le produit ou la division se révèlent plus complexes. Ces deux opérations sont traitées dans ce chapitre avec une restriction pour la division au cas où le reste est nul, c'est-à-dire où la division est exacte. Des algorithmes existent déjà pour de tels calculs. L'objectif consiste donc à obtenir de meilleures complexités en tentant de prendre en compte tous les paramètres des polynômes creux : le nombre de monômes non nuls, le degré (et la hauteur sur $\mathbb{Z}$). Et ce aussi bien pour les polynômes donnés en entrée que pour le résultat.

En effet, si ces opérations, produit et division exacte, se révèlent délicates à effectuer de manière optimale, c'est notamment car la taille du résultat n'est pas connue a priori. Les tailles peuvent varier, des bornes et exemples de ces variations ont été fournies en section 1.4. Des variations de tailles sont fréquentes lors d'opérations entre polynômes, même dans le cas dense. Ainsi le produit de deux polynômes de degré d et de hauteur H dans $\mathbb{Z}[x]$ a pour degré $2d$ et une hauteur d'au plus dH^2. La taille du résultat est bornée par $2d(\log d + 2\log H)$. Même si cette borne peut être surestimée pour la taille des coefficients, elle est quasi-linéaire dans la taille des entrées ($2d \log H$). Pour le cas d'un polynôme sur un corps fini $\mathbb{F}_q$, la taille $2d \log q$ correspond à la fois à la taille du polynôme produit et des entrées.

Les croissances qui apparaissent lors d'opérations entre polynômes creux sont plus problématiques. Les bornes a priori ne sont pas quasi-linéaires mais surtout, elles peuvent être largement surestimées. La croissance exacte est elle imprévisible. Les exemples suivants illustrent ce caractère imprévisible en prenant comme entrées deux polynômes qui ont le même support alors que le nombre de monômes du résultat varie fortement.

Exemple 5.1. À partir des mêmes supports de cardinalité $\mathcal{O}(T)$ le produit a un nombre de monômes pouvant aller de 2 à $\Omega(T^2)$:

$$\begin{aligned}
(\sum_{i=0}^{T-1} x^i) \times (\sum_{i=0}^{T-1} (x^{Ti+1} - x^{Ti})) &= x^{T^2} - 1, \\
(\sum_{i=0}^{T-1} x^i) \times (\sum_{i=0}^{T-1} (x^{Ti+1} + x^{Ti})) &= x^{T^2} + \sum_{i=0}^{T^2-1} 2x^i + 1.
\end{aligned}$$

Exemple 5.2. À partir des mêmes supports de cardinalité constante la division peut être exacte ou non et le nombre de monômes des quotients et restes être constants ou correspondre au degré. Pour simplifier la comparaison, la division euclidienne de F par G est notée $F/G = Q\ ,\ R$ si $F = GQ + R$.

$$\begin{aligned}(x^{2d+1} - x^{2d} + x^d)/(x^{d+1} - x^d + 1) &= x^d\ ,\ 0,\\ (x^{2d+1} + x^{2d} + x^d)/(x^{d+1} - x^d + 1) &= x^d + \sum_{i=0}^{d-1} 2x^i\ ,\ 2x^d - \sum_{i=0}^{d-1} 2x^i.\end{aligned}$$

Ces observations sur la taille des résultats montrent que pour mesurer la performance d'un algorithme effectuant un produit ou une division, il convient de prendre en compte la taille du résultat. Si la taille de la sortie est quadratique dans celle des entrées, il ne peut exister d'algorithme quasi-linéaire dans la taille des entrées. En revanche un algorithme quadratique dans la taille des entrées sera alors linéaire dans la taille de la sortie et donc optimal. Une telle approche n'est envisagée que depuis récemment pour le produit [AR15]. En revanche, l'algorithme de division euclidienne naïf qui calcule le quotient monôme par monôme en mettant à jour le dividende dépend déjà du nombre de monôme du quotient et donc d'une partie de la sortie.

Les variations du nombre de monômes montrent que les deux opérations (produit et division de polynômes) ne sont pas aussi fortement liées qu'elle sne le sont pour des polynômes denses. Dans le cas dense, la division euclidienne entre deux polynômes se réduit au calcul d'un nombre fini de produits de polynômes de même taille que les polynômes initiaux. Dans le cas creux, le nombre de monômes d'un produit est au plus quadratique dans le nombre de monômes des polynômes d'entrée. Pour une division, le nombre de monômes de la sortie n'est pas borné par ceux de l'entrée, mais par le degré. Un nombre fini de produits de polynômes de la taille des polynômes d'entrées ne peut donc pas correspondre au calcul d'une division euclidienne. Les méthodes permettant de réduire la division au produit dans le cas dense (approche de type Newton ou diviser pour régner) peuvent être utilisées pour des polynômes creux. Cependant, les polynômes intermédiaires calculés par ces algorithmes (l'inverse de $G^\star$ ou un produit de polynômes de degré deux fois plus petit $G_1 \times Q_1$) n'ont aucune raison d'être aussi creux que le quotient et le reste calculés.

Pour pouvoir être quasi-linéaire par rapport au support des polynômes d'entrée et de sortie, l'approche considérée dans cette thèse est différente. Pour le produit, comme pour la division exacte, les algorithmes s'appuient sur l'interpolation de polynôme creux et la vérification de produits. Le recours à l'interpolation est classique pour l'arithmétique de polynômes. Il a permis d'atteindre des complexités sous-quadratiques pour le produit dense.

Ce chapitre présente d'abord en section 5.1 les algorithmes existants pour le produit et la division. Il s'agit aussi bien d'optimisation des algorithmes naïfs que d'algorithmes visant à calculer le produit avec une complexité dépendant de la taille du résultat. Nos nouveaux algorithmes sont décrits en sections 5.2 et 5.3 pour le produit puis la division

exacte avec notamment les premières complexités quasi-linéaires pour ces opérations atteintes pour des polynômes à coefficients entiers.

5.1. État de l'art

5.1.1. Méthodes naïves

Pendant longtemps, les seuls algorithmes pour effectuer un produit ou une division euclidienne étaient des algorithmes naïfs quadratiques. Même pour une méthode naïve, il faut cependant manipuler prudemment les polynômes creux. Sans cela il est possible de générer un important surcoût pour la gestion des données. Au contraire, une gestion méticuleuse des structures de données permet d'optimiser le coût en espace des algorithmes. C'est ce qu'ont fait Johnson [Joh74] et Monagan et Pearce [MP07 ; MP09b ; MP11a] en s'appuyant sur des tas. Dans ces algorithmes, les tas servent à la fois de structures pour les polynômes creux et pour les calculs intermédiaires. L'algorithme naïf présenté par van der Hoeven et Lecerf [HL13] s'appuie sur une structure de vecteur creux et propose une approche différente, de type diviser pour régner. La complexité a la même dépendance dans la taille des supports dans les deux cas, aussi seule l'approche directe sera présentée.

Pour le produit de deux polynômes $F = \sum_{i=0}^{\#F-1} f_i x^{d_i}$ et $G = \sum_{j=0}^{\#G-1} g_j x^{e_j}$, l'algorithme naïf consiste simplement à calculer tous les produits de monômes $f_i g_j x^{d_i+e_j}$ et à les ajouter au polynôme produit FG. Pour être efficace dans la gestion de l'espace additionnel et ne pas ajouter trop de surcoût, il convient de ne pas calculer tous ces produits d'un coup et de le faire dans un ordre approprié. Dans cette présentation, les polynômes sont considérés comme des tas triés par l'ordre décroissant des exposants. C'est ce même ordre qui est utilisé pour le calcul des monômes de FG. L'algorithme s'adapte naturellement si le tri suit l'ordre croissant des monômes.

L'idée est de calculer d'abord le monôme dominant de FG : $f_{\#F-1}g_{\#G-1}x^{d_{\#F-1}+e_{\#G-1}}$. Le monôme suivant est celui d'exposant maximal entre $f_{\#F-2}g_{\#G-1}x^{d_{\#F-2}+e_{\#G-1}}$ et $f_{\#F-1}g_{\#G-2}x^{d_{\#F-1}+e_{\#G-2}}$. Ces deux monômes sont alors insérés dans un tas trié par exposant décroissant dont le maximum sera toujours le prochain monôme à ajouter à FG. Lorsqu'un monôme $f_i g_j x^{d_i+e_j}$ est extrait du tas pour être ajouté à FG, il y a deux monômes de degré inférieur qui pourraient être insérés dans le tas : $f_{i-1}g_j x^{d_{i-1}+e_j}$ et $f_i g_{j-1} x^{d_i+e_{j-1}}$. Il n'est pas forcément nécessaire d'insérer ces deux monômes. En particulier, tant que $f_{i-1}g_{j+1}x^{d_{i-1}+e_{j+1}}$ n'a pas encore été ajouté à FG, $f_{i-1}g_j x^{d_{i-1}+e_j}$ ne peut pas y être ajouté. Il est possible de ne décrémenter que sur les monômes de G (en ajoutant juste $f_i g_{j-1} x^{d_i+e_{j-1}}$ au tas) sauf si $j = \#G$ où les deux monômes sont ajoutés dans le tas. Ainsi on s'assure de ne jamais avoir $f_{i-1}g_j x^{d_{i-1}+e_j}$ dans le tas avant que $f_{i-1}g_{j+1}x^{d_{i-1}+e_{j+1}}$ ne soit dans FG. De plus ceci permet de limiter la taille du tas à $\#F$. Les rôles de F et G peuvent être inversés pour avoir le tas le plus petit possible. Toute cette procédure est décrite dans l'algorithme 19 ProduitNaïf.

Algorithme 19 ProduitNaïf

Entrée : $F = \sum_{i=0}^{\#F-1} f_i x^{d_i}$, $G = \sum_{j=0}^{\#G-1} g_j x^{e_j}$ polynômes dans $\mathbb{A}[x]$ avec $\#F \leq \#G$

Sortie : FG

1: $P \leftarrow$ le polynôme nul
2: $L \leftarrow$ tas trié par degrés décroissants
3: Ajouter le triplet $(f_{\#F-1}g_{\#G-1}, d_{\#F-1} + e_{\#G-1}, (\#F-1, \#G-1))$ à L
4: **Tant que** L n'est pas vide **faire**
5: Extraire $(coef, deg, (i,j))$ de L
6: Ajouter $coef \times x^{deg}$ à P, soit en additionnant au dernier monôme s'ils ont le même degré soit comme monôme de poids faible de P
7: **Si** $j \neq 0$ **alors**
8: Ajouter $(f_i g_{j-1}, d_i + e_{j-1}, (i, j-1))$ à L
9: **Si** $j = \#G$ **alors**
10: Ajouter $(f_{i-1}g_j, d_{i-1} + e_j, (i-1, j))$ à L
11: **Renvoyer** P

L'algorithme tel que présenté regarde si deux monômes ont le même degré au moment d'un ajout à P, il est aussi possible de le faire au moment de l'ajout à L. Lors de l'ajout de $(coef, deg, (i,j))$ au tas L, s'il contient déjà un triplet de la forme $(coef', deg, (i', j'))$ il y a deux possibilités. Soit considérer arbitrairement que l'un des triplets est plus grand que l'autre comme c'est sous-entendu dans l'algorithme ProduitNaïf. Soit procéder à l'addition sans perdre les indices concernés ce qui donne un triplet du type $(coef + coef', deg, [(i,j),(i',j')])$. Lorsque ce monôme est ajouté à R, il faut ensuite insérer dans L les nouveaux monômes correspondant aux deux paires d'indices. Le nombre de paires d'indices donnant le même degré peut être supérieur à deux, le troisième membre du triplet sera alors une liste contenant toutes les paires d'indices. Pour optimiser encore plus, il est possible de ne pas stocker les produits $f_i g_j$ et de ne les effectuer qu'au moment de l'ajout à R, de passer par des pointeurs et potentiellement d'autres tas. De telles améliorations sont décrites dans [MP11a].

Théorème 5.3. *Pour deux polynômes creux $F, G \in \mathbb{A}[x]$ avec $\#F \leq \#G$, l'algorithme* ProduitNaïf *calcule le produit FG en effectuant $\#F\#G$ produits dans $\mathbb{A}$ et autant d'additions dans $\mathbb{Z}$ ainsi que $\#F\#G \log \#F$ comparaisons dans $\mathbb{Z}$ pour gérer le tas L. L'espace additionnel utilisé est un tas de longueur $\#F$.*

Cet algorithme est particulièrement efficace par rapport aux degrés (linéaire dans le logarithme du degré) et aux coefficients. Seuls les produits ont été comptés dans $\mathbb{A}$. Les additions sont au plus aussi nombreuses et ont en général un coût plus faible. La dépendance dans le nombre de monômes est la seule à ne pas être quasi-optimale. Elle est systématiquement quadratique. Cela correspond au pire cas mais pas à tous les cas comme montrés par les exemples avec des produits pouvant avoir un nombre constant de monômes.

Passons maintenant à la division d'un polynôme F par un polynôme G. L'algorithme naïf consiste à d'abord diviser le monôme de poids fort de F par le monôme de poids fort de G pour obtenir le monôme m de poids fort de Q. Ensuite, le dividende est mis à jour en lui ajoutant $-Gm$. Le processus est répété pour calculer tous les monômes de Q, jusqu'à ce que le dividende soit de degré inférieur à celui de G et vaille le reste R. Ce schéma correspond à calculer $R = F - GQ$ en calculant progressivement le produit GQ au fur et à mesure que de nouveaux monômes de Q sont calculés. Le calcul progressif de ce produit de polynômes peut être effectué en s'appuyant sur un tas comme pour un produit où les polynômes sont intégralement connus dès le départ. Ainsi, il n'est pas nécessaire de mettre à jour tout le dividende, juste son monôme de poids fort. Les informations pour la mise à jour des autres monômes de poids fort ou pour le calcul du reste sont conservées dans le tas du produit GQ (dont les monômes de poids fort sont extraits au fur et à mesure). Le schéma de calcul correspond bien à celui du produit en décrémentant soit le long des monômes de F soit dans le tas représentant GQ. La longueur de ce tas peut être optimisée pour être le minimum entre $\#G$ et $\#Q$, même si ce n'est qu'au cours des calculs que l'on se rend compte si Q a plus de monômes que G [MP11a].

Théorème 5.4. *Pour deux polynômes creux $F, G \in \mathbb{A}[x]$, la division euclidienne $F = GQ+R$ peut être calculée en effectuant $\#Q$ divisions et $\#G\#Q$ produits dans $\mathbb{A}$, $\#Q(\#G+1)$ additions dans $\mathbb{Z}$ ainsi que $\#F+\#G\#Q \log \min(\#G, \#Q)$ comparaisons dans $\mathbb{Z}$. L'espace additionnel utilisé est un tas de longueur $\min(\#G, \#Q)$.*

La complexité prend déjà en compte la taille de la sortie et dépend de $\#Q$ mais n'est pas quasi-linéaire à cause des facteurs $\#G\#Q$. Dans certains cas ceci peut être quasi-linéaire, le reste R pouvant avoir jusqu'à $\#F + \#G\#Q$ monômes non nuls, mais ce n'est pas toujours vrai. Comme dans le cas du produit, c'est cette partie de la complexité liée au support des polynômes qui appelle une amélioration, les autres parties étant déjà très efficaces.

5.1.2. Vers une meilleure dépendance dans le nombre de monômes du produit

Les améliorations présentées ici ne concernent que le produit. À notre connaissance aucun algorithme n'a été proposé pour calculer une division euclidienne sans se reposer sur la méthode naïve.

Pour obtenir de meilleurs algorithmes de produit, il faut améliorer les cas où le polynôme FG a moins de $\#F\#G$ monômes non nuls. Pour que FG ait si peu de monômes il faut qu'il y ait des collisions lors du produit. C'est à dire qu'il y ait des paires (d, d') et (e, e') dans leurs supports respectifs tels que $d + e = d' + e'$. Ils sont alors additionnés en un seul monôme de FG, voire en aucun si en plus les coefficients s'annulent. Ainsi il est possible de considérer un niveau de support intermédiaire : en prenant en compte les collisions mais pas les annulations liées aux coefficients.

Définition 12. Soit F, G deux polynômes dans $\mathbb{A}[x]$. Le *support structurel* de FG est la

somme ensembliste de support(F) et support(G). Il est à distinguer du *support arithmétique* qui est exactement support(FG).

Si le support arithmétique (ou le support structurel) de FG est connu, il est possible de retrouver FG en résolvant le système de Vandermonde utilisé dans l'algorithme de Prony en section 3.1.1 (faits 3.2 et 3.4 pages 62 et 63). Le calcul de FG est alors linéaire dans la cardinalité du support arithmétique (ou structurel) de FG. Ceci est d'ailleurs vrai pour tout ensemble contenant le support arithmétique et a été decrit précisémment dans [HL13].

5.1.2.1. Algorithmes quasi-linéaires dans le support structurel

La question importante est donc l'obtention du support structurel c'est-à-dire de la somme entre le support de F et celui de G. Plus généralement il s'agit de la somme entre deux ensembles d'entiers positifs. Arnold et Roche [AR15] ont proposé en 2015 une méthode pour calculer cette somme, permettant ainsi d'avoir un algorithme de produit quasi-linéaire dans la cardinalité du support structurel. Leur idée est de revenir des ensembles aux polynômes en considérant pour un ensemble d'entiers $\mathcal{A}$, le polynôme $A = \sum_{a\in\mathcal{A}} x^a$. En définissant de manière similaire le polynôme B pour un ensemble $\mathcal{B}$, le support de AB est la somme ensembliste $\mathcal{A}+\mathcal{B}$. Ce support est calculé grâce à des algorithmes d'interpolation et en utilisant le fait que $A((l+1)x) \bmod l^2 = \sum_{a\in\mathcal{A}} (al+1)x^a$ pour l suffisamment grand.

Les grandes étapes du calcul sont les suivantes :

1. Choisir p tels que AB soit sans collision modulo x^p-1 et calculer $A_p = A \bmod x^p-1$ et $B_p = B \bmod x^p-1$.
2. Trouver une borne fine sur la taille du support de A_pB_p en calculant $A_pB_p \bmod x^q-1$ pour différents premiers q tels que A_pB_p n'ait qu'au plus une moitié de ses monômes en collision.
3. En utilisant la borne trouvée, interpoler A_pB_p et $(A((l+1)x) \bmod x^p-1)(B((l+1)x) \bmod x^p-1)$ en considérant les coefficients modulo l^2 avec $l = 4\deg(AB)$.
4. Réduire les produits modulo x^p-1.
5. À partir des monômes cx^e et $c'x^e$ des deux polynômes reconstruire les éléments $(\frac{c'}{c}-1)/l$ de $\mathcal{A}+\mathcal{B}$.

Les choix de p et des q sont probabilistes et reposent sur les corollaires 2.11 et 2.13 (pages 29 et 30) pour limiter le nombre de collisions. Il faut noter que si p est bien choisi le support de A_pB_p contient au plus 2 fois plus d'élements que celui de AB. En effet, les supports de AB et $AB \bmod x^p-1 = A_pB_p \bmod x^p-1$ ont la même cardinalité et A_pB_p a degré au plus $2p-2$. S'il y a des collisions modulo x^p-1 dans A_pB_p elles ne concernent que deux monômes à la fois et il ne peut y avoir d'annulation puisque les coefficients sont

positifs. C'est pour cela que regarder la taille du support après une deuxième réduction, modulo $x^q - 1$ est pertinent pour borner le support de AB.

Cette procédure a permis à Arnold et Roche de décrire un algorithme de produit de type Monte-Carlo qui est quasi-linéaire dans la cardinalité du support structurel. La première étape correspond au calcul du support structurel. Ensuite un système de type Vandermonde est construit pour obtenir d'abord le support arithmétique en manipulant des coefficients de petite taille puis le polynôme lui-même.

Théorème 5.5 ([AR15]). *Soit $F, G \in \mathbb{Z}[x]$ deux polynômes de degré inférieur à D et hauteur H. Soit $T = \max(\#F, \#G, \#(FG))$ et S la cardinalité du support structurel de FG. Il existe un algorithme de type Monte-Carlo pour calculer FG en temps $\tilde{\mathcal{O}}(T \log H + S \log D)$.*

Le lien entre la somme ensembliste et le produit de polynômes est très fort, particulièrement dans le cas où les polynômes sont à coefficients positifs. Cela a donné lieu à des études récentes [BN20 ; BFN21 ; BFN22] dédiées à la somme ensembliste et aux produits de polynômes à coefficients positifs. Il est à noter différents types d'algorithmes, notamment déterministe et Las Vegas et le recours à deux réductions consécutives. Si les réductions sont données par des fonctions de hachages un peu plus générales que le modulo $x^p - 1$ puis $x^q - 1$, l'enchaînement est à nouveau rendu possible par le caractère positif des coefficients. Ces algorithmes sont quasi-linéaires dans le support arithmétique mais seulement puisque celui-ci est égal au support structurel en l'absence d'annulation, les coefficients étant tous positifs.

5.1.2.2. Algorithmes quasi-linéaires dans le support arithmétique

Des algorithmes ont aussi été proposés pour être quasi-linéaires dans le support arithmétique.

Nakos [Nak20] en a ainsi décrit un pour des polynômes à coefficients entiers. Il utilise aussi des réductions modulo $x^p - 1$ mais les effectue en passant par l'évaluation sur une racine complexe de l'unité via du calcul numérique. Les différentes étapes de l'algorithme sont effectuées avec une cardinalité supposée T pour le support puis le résultat est vérifié. Si le résultat n'est pas correct, la valeur de T est augmentée. En procédant ainsi, la complexité atteinte est bien quasi-linéaire en T. Cependant, elle compte aussi un facteur cubique dans le logarithme du degré. En comparaison, si l'algorithme naïf est quadratique dans le nombre de monômes des entrées et potentiellement de la sortie, il est toujours linéaire dans le logarithme du degré.

Il convient aussi de mentionner l'approche de Joris van der Hoeven [Hoe20]. Elle repose sur une heuristique qui vient de ce que les probabilités données en section 2.1.1 (page 24) semblent en pratique assez pessimistes. En diversifiant les coefficients et en dilatant les exposants à partir de nombres aléatoires pour générer six polynômes différents, cela suffirait pour que des calculs modulo $x^r - 1$ permettent de construire le produit. Son algorithme se concentre spécifiquement sur le cas des polynômes multivariés. De plus il

est supposé que le degré en chaque variable, d est relativement petit par rapport aux autres paramètres et des facteurs de la forme $(\log d)^{\mathcal{O}(1)}$ sont négligés.

5.2. Produit de polynômes creux

Cette partie se concentre sur nos algorithmes pour calculer un produit de polynômes creux FG. L'objectif est d'être quasi-optimal et ce en prenant en compte tous les paramètres. Une telle complexité n'est cependant pas atteinte dans tous les cas mais seulement pour les polynômes à coefficients entiers.

La méthode générale utilisée pour effectuer le produit s'appuie sur l'interpolation et la vérification. Elle est décrite par les étapes suivantes :

1. Fixer une borne T sur le nombre de monômes non nuls du produit.
2. Tenter d'interpoler FG en prenant T comme borne.
3. Si un algorithme de vérification indique que le résultat obtenu ne correspond pas au produit FG, reprendre à l'étape précédente en doublant T.

La présence d'une borne T est nécessaire pour de nombreux algorithmes d'interpolation dont la complexité dépend de T au moins linéairement pour les plus efficaces. Puisque la seule borne connue a priori sur T est quadratique dans les entrées (voir fait 1.19 page 18) elle ne peut être utilisée directement. C'est pour contourner cette difficulté qu'est utilisée une borne croissante associée à une vérification.

Une telle approche implique une dépendance dans l'efficacité des algorithmes d'interpolation. Puisque le plus efficace est celui que nous avons décrit pour des polynômes entiers, c'est logiquement dans $\mathbb{Z}[x]$ qu'une complexité quasi-optimale sera atteinte.

5.2.1. Algorithme quasi-linéaire sur les entiers

Si F et G sont des polynômes creux dans $\mathbb{Z}[x]$ de degré d, il est possible de s'appuyer sur l'algorithme 12 INTERPOLATION_BNM d'interpolation creuse décrit en section 3.2 (page 81). Cet algorithme effectue des évaluations d'une boîte noire modulaire sur $\mathcal{O}(T)$ racines p-ièmes de l'unité ω. Pour l'adapter au cas du produit, il suffit d'être capable d'effectuer ces évaluations pour le produit FG.

Pour cela, à chaque passage dans la boucle F et G doivent d'abord être réduits modulo $x^p - 1$ ce qui nécessite $\mathcal{O}(T\mathsf{I}(\log d))$ opérations binaires. De même, pour les évaluations en $(1 + q^{2k})\omega$ dans $\mathbb{F}_{q^{2k}}$, le polynôme $F((1 + q^{2k})x) \bmod \langle x^p - 1, q^{2k}\rangle$ est calculé au préalable en utilisant le fait qu'il vaut $\sum(f_i + d_i q^{2k} \bmod q^{2k})x^{d_i \bmod p}$ si $F = \sum f_i x^{d_i}$. Le même procédé s'applique aussi à G. Ceci n'ajoute que $\mathcal{O}(T\mathsf{I}(\log q^{2k}))$ opérations binaires à la seule réduction modulo $x^p - 1$. Ensuite les évaluations des polynômes de degré p sur l'ensemble des points d'évaluations ne nécessitent plus que $\tilde{\mathcal{O}}(T \log p)$ opérations dans chacun des anneaux d'évaluation d'après le fait 3.4 (page 63). Il faut noter que si $T < \#F, \#G$, ces coûts dépendent en fait de $\#F, \#G$ et non de T.

Enfin, après avoir évalué les polynômes F et G, il faut multiplier deux à deux les évalués obtenus ce qui n'ajoute qu'une opération. En reprenant la preuve de complexité du théorème 3.18 (page 81), on voit que le coût de cette évaluation correspond au coût d'un passage dans la boucle. Le coût de l'interpolation à partir d'une boîte noire modulaire ou de deux polynômes dont on cherche le produit est donc asymptotiquement le même, à condition de bien réduire les polynômes modulo $x^p - 1$ avant d'effectuer une évaluation.

Ceci permet donc de décrire le premier algorithme quasi-linéaire de produit de polynôme creux.

Algorithme 20 ProduitPolynômesEntiers

Entrée : F, $G \in \mathbb{Z}[x]$ de degré inférieur à D et hauteur inférieure à H, $\rho \geq 1$
Sortie : FG avec une probabilité d'au moins $1 - \frac{1}{2^{\rho+1}}$

1: $T \leftarrow 1$
2: **Tant que** $T < \#F\#G$ **faire**
3: $T \leftarrow \min(2T, \#F\#G)$
4: Calculer $\mathcal{O}(\rho)$ candidats R pour FG en utilisant l'algorithme 12 Interpolation_BNM avec pour bornes : T sur le nombre de monômes, $2D$ sur le degré et $\min(\#F, \#G)H^2$ sur la hauteur.
▷ L'évaluation de la boîte noire modulaire est remplacée par la réduction des polynômes modulo $x^p - 1$ suivie des évaluations
5: Garder le candidat le plus fréquent
6: Tester si $FG = R$ avec une probabilité d'échec d'au plus $\frac{1}{2^{\rho+1}T}$ ▷ corollaire 4.44
7: **Si** le test renvoie **vrai alors**
8: **Renvoyer** R

Théorème 5.6. *Soit F, G deux polynômes creux dans $\mathbb{Z}[x]$, de degré au plus D et hauteur H et $\rho \geq 1$. Avec une probabilité d'au moins $1 - \frac{1}{2^\rho}$, l'algorithme* ProduitPolynômesEntiers *renvoie FG en effectuant $\tilde{\mathcal{O}}(t(\log D + \log H)\rho + \rho^4)$ opérations binaires avec $t = \max(\#(FG) + \#F + \#G)$.*

Démonstration. La probabilité $1 - \frac{1}{2^\rho}$ concerne à la fois la validité et la complexité de l'algorithme qui est de type Atlantic-City. Nous allons prouver que chacune est vérifiée indépendamment avec une probabilité d'au moins $\geq 1 - \frac{1}{2^{\rho+1}}$.

La réponse de l'algorithme est incorrecte si $FG \neq R$. Ceci ne peut arriver que si le candidat produit est incorrect mais que l'algorithme de vérification ne le détecte pas. Autrement dit, pour que l'algorithme renvoie un polynôme qui est bien le produit, il faut que chacune des vérifications soit correcte. Puisqu'une vérification a une probabilité d'erreur d'au plus $\frac{1}{2^{\rho+1}T}$ avec T qui prend comme valeur des puissances de 2, l'algorithme est correct avec une probabilité d'au moins $1 - \sum_T \frac{1}{2^{\rho+1}T} \geq 1 - \frac{1}{2^{\rho+1}}$.

Pour la complexité, il faut borner le nombre de répétitions de la boucle. Puisque les valeurs de T sont des puissances de 2, la première valeur $\geq \#(FG)$ vaut au plus $\min(2\#(FG), \#F\#G)$ car $\#F\#G$ est toujours une borne sur $\#(FG)$. Dès que T atteint

cette valeur, le candidat est bien FG avec une probabilité d'au moins $1 - \frac{1}{2^{\rho+1}}$ d'après le théorème 3.18 (page 81) pour un nombre de candidats supérieur à $48(\rho+1)/\log e$. Dans ce cas, le test, qui n'est biaisé que d'un côté, répond toujours vrai et l'algorithme renvoie $R = FG$. Ainsi, avec une probabilité d'au moins $1 - \frac{1}{2^{\rho+1}}$, le nombre de répétitions est $\mathcal{O}(\log \#(FG))$. La complexité de la phase d'interpolation est $\tilde{\mathcal{O}}(\rho(t(\log D + \log H))$. Pour la vérification, le corollaire 4.44 (page 118) donne une complexité en $\mathcal{O}_{\frac{1}{2^\rho}}(s \log s \log\log s)$ avec $s = t(\log D + \log H)$. Pour être plus précis sur les probabilités, le théorème 4.43 (page 116) montre que ce terme principal est multiplié par ρ et qu'un terme logarithmique en s est multiplié par ρ^4. Ce qui permet bien d'atteindre la complexité indiquée avec une probabilité d'au moins $1 - \frac{1}{2^{\rho+1}}$. □

Remarque 5.7. En s'assurant que la valeur maximale de T ne dépasse pas $\#F\#G \geq \#(FG)$, on fixe la complexité dans le pire cas à $\tilde{\mathcal{O}}(\#F\#G(\log D + \log H)\rho + \rho^4)$ opérations binaires. Si les calculs sont toujours erronés à ce stade, l'algorithme ne renvoie aucun polynôme.

5.2.2. Produit de polynômes creux sur un anneau $\mathbb{A}$

Si les polynômes sont définis avec des coefficients dans un anneau $\mathbb{A}$ différent de $\mathbb{Z}$, les algorithmes d'interpolation ne sont plus aussi efficaces et pour les plus rapides sont décrits pour des circuits arithmétiques et non des boîtes noires. Il est alors nécessaire de s'assurer qu'une démarche similaire fonctionne toujours ou s'il est possible d'être plus rapide que l'interpolation.

5.2.2.1. Réduction à l'interpolation

L'approche utilisée pour le produit sur les entiers est en fait beaucoup plus générale que le cas restreint dans lequel elle a été utilisée. L'analyse de l'algorithme 20 ProduitPolynômesEntiers et la preuve du théorème 5.6 peuvent s'étendre à une version plus générale d'une méthode reposant sur la tentative et la vérification avec une borne croissante sur le nombre de monômes. C'est-à-dire à toute procédure de la forme suivante.

Théorème 5.8. *Avec un probabilité d'au moins $1 - \epsilon$, la procédure* EssaisEtTests *renvoie bien $R = \star(F_1, \ldots, F_k)$ pour un coût qui est $\mathcal{O}(\log(t))$ fois le coût de l'obtention et du test d'un candidat ayant $\mathcal{O}(t)$ monômes non nuls. En considérant que t est le nombre de monômes de $\star(F_1, \ldots, F_k)$ et que le coût des différents calculs augmente quand T augmente.*

Cet algorithme reprend la démarche suivie dans l'algorithme du produit dans $\mathbb{Z}[x]$, tout en montrant à quel point elle se généralise.

Dans le cas du produit de polynômes sur un anneau quelconque, la généralisation consiste simplement à prendre un algorithme d'interpolation adapté tout en conservant comme test la vérification de produit creux décrite dans l'algorithme 18.

Algorithme 21 EssaisEtTests

Entrée : Une liste de polynômes $(F_1, \ldots, F_k)$ dans $\mathbb{A}[x]$, $0 < \epsilon < 1$ et une opération $\star$ à appliquer à ces polynômes

Sortie : Un polynôme creux, résultat correct de l'opération $\star(F_1, \ldots, F_k)$ avec une probabilité d'au moins $1 - \frac{\epsilon}{2}$

1: $T \leftarrow 1$
2: **Tant que vrai faire**
3: $\quad T \leftarrow 2T$
4: $\quad$ Calculer un candidat R qui, si T est une borne correcte sur le nombre de monôme, vaut $\star(F_1, \ldots, F_k)$ avec une probabilité d'au moins $1 - \frac{\epsilon}{2}$
5: $\quad$ Tester si $R = \star(F_1, \ldots, F_k)$ avec une probabilité d'échec d'au plus $\frac{\epsilon}{2T}$
6: $\quad$ **Si** Le test renvoie **vrai alors**
7: $\quad\quad$ **Renvoyer** R

Corollaire 5.9. *Soit F, $G \in \mathbb{A}[x]$, le calcul de FG se réduit à l'interpolation d'un polynôme de degré $d = \deg(F) + \deg(G)$ et ayant $t(= \#(FG))$ monômes. Avec probabilité $1-\epsilon$, le résultat est correct et obtenu en répétant $\mathcal{O}(\log t)$ interpolation avec une borne $T \leq 2t$ sur le nombre de monômes plus pour les vérifications $\tilde{\mathcal{O}}((t+\#F+\#G)(\log \frac{1}{\epsilon} + \log\log d))$ opérations dans $\mathbb{A}$ et $\tilde{\mathcal{O}}((t + \#F + \#G)\log d + \log^4 \frac{1}{\epsilon})$ opérations binaires.*

Sur un corps fini $\mathbb{F}_q$, les algorithmes d'interpolation les plus rapides [Hua19 ; AGR14] sont décrits sur des circuits arithmétiques et évaluent le polynôme en x ou αx modulo $x^p - 1$ pour certains premiers p et éléments α aléatoires. Dans ces algorithmes et sur un circuit de taille L, de telles évaluations nécessitent $\tilde{\mathcal{O}}(Lp \log q)$ opérations binaires en se basant sur de l'arithmétique dense de degré p. Le premier p est choisi d'une valeur à peu près $T \log d$ pour avoir de bonnes propriétés de divisibilité. Ainsi, le coût binaire de l'évaluation est $\tilde{\mathcal{O}}(LT \log D \log q)$. Si le polynôme est donné comme un produit, une telle évaluation ne coûte que $\tilde{\mathcal{O}}(T \log D \log q)$ opérations binaires en calculant d'abord pour F et G les réductions modulo $x^p - 1$ séparément. Le produit entre les deux réductions est effectué par un algorithme d'arithmétique dense en degré p. En l'absence du facteur L, cette évaluation est toujours plus rapide. Puisque la vérification est quasi-linéaire sur tous les corps finis (voir section 4.3.2), le produit s'effectue essentiellement au même coût que l'interpolation sans le facteur L en prenant une probabilité de succès constante pour simplifier.

Corollaire 5.10. *Soit F, $G \in \mathbb{F}_{q^s}[x]$ avec q premier, $d = \deg(F) + \deg(G)$ et $t = \#(FG)$. Avec probabilité de succès constante, il est possible de calculer correctement FG en effectuant*

- $\tilde{\mathcal{O}}((t + \#F + \#G)\log d \log q^s)$ *opérations binaires si $q > d$,*
- $\tilde{\mathcal{O}}((t + \#F + \#G)(\log d)^2(\log d + \log q^s))$ *opérations binaires si $q \leq d$.*

Démonstration. Il s'agit simplement de s'appuyer sur l'algorithme de Huang (corollaire 3.12, page 71) dans le premier cas et celui d'Arnold, Giesbrecht et Roche dans

le second (théorème 3.8, page 69). □

Les algorithmes ainsi obtenus sont quasi-linéaires dans la cardinalité du support grâce au recours au schéma d'essai et de vérification avec une borne croissante sur la cardinalité. Le facteur quadratique des algorithmes naïfs est donc amélioré. En revanche, ils ne sont pas globalement quasi-linéaires car ils s'appuient sur des algorithmes d'interpolation qui eux-mêmes ne le sont pas. C'est clair si $q \leq d$ puisque l'algorithme est cubique en $\log d$. Si $q > d$, le facteur $\log d \log q^s$ est quadratique puisque la taille d'un monôme est en $\log d + \log q^s$.

5.2.2.2. Algorithme quasi-linéaire dans le support structurel et les autres paramètres sur un grand corps fini

Dans un article de 2020 [GGP20], nous pensions avoir mis au point le premier algorithme de produit quasi-linéaire sur les entiers et les grands corps finis. Hélas, il y avait une erreur d'analyse dans l'article et l'algorithme proposé a une complexité qui est quasi-linéaire en considérant le support structurel et non le support arithmétique.

Puisque dans la section précédente, un algorithme quasi-linéaire par rapport au support arithmétique a été décrit pour les entiers, seul le cas des grands corps sera décrit ici. Je préciserai le point d'analyse erroné dans l'article et pourquoi cela donne une complexité qui dépend du support structurel.

L'idée est de se baser sur l'algorithme d'interpolation de Huang décrit en section 3.1.3. Celui-ci s'appuie sur le fait qu'à partir du polynôme $R \bmod x^p - 1$ et du polynôme $R' \bmod x^p - 1$, il est possible de reconstruire le polynôme R s'il est sans collision modulo $x^p - 1$ (et donc R' est aussi sans collision). Cette méthode est très efficace mais pas quasi-linéaire. Pour l'améliorer, l'idée était non pas d'utiliser l'algorithme d'interpolation pour calculer FG, mais pour calculer $(F \bmod x^p - 1)(G \bmod x^p - 1)$ un polynôme de plus petit degré. Dans le même temps, l'interpolation pouvait être utilisée pour calculer $(F \bmod x^p - 1)(G' \bmod x^p - 1)$ et $(F' \bmod x^p - 1)(G \bmod x^p - 1)$. En réduisant encore une modulo $x^p - 1$ ces trois polynômes et en sommant les deux derniers, on obtient $(FG) \bmod x^p - 1$ et $(FG)' \bmod x^p - 1$. En appliquant la règle de reconstruction par la dérivée, il est alors possible de calculer FG.

La procédure fonctionne bien, mais son analyse est erronée. Si p est choisi pour assurer que le support de FG et celui de $(FG) \bmod x^p - 1$ aient exactement le même nombre d'éléments, il ne dit rien du support de $(F \bmod x^p - 1)(G \bmod x^p - 1)$. Notre argument était de dire que puisque p est choisi en suivant le corollaire 2.11 (page 29) pour le polynôme FG avec la borne $\#F\#G$ sur le nombre de monômes, alors pour un polynôme $(F \bmod x^p - 1)(G \bmod x^p - 1)$ de degré plus petit avec une même borne sur le nombre de monômes il aurait aussi la bonne propriété. C'est-à-dire que $(F \bmod x^p - 1)(G \bmod x^p - 1)$ serait sans collision modulo $x^p - 1$ et aurait donc autant de monômes que $(FG) \bmod x^p - 1$ et donc que FG. L'erreur est que si p est choisi aléatoirement pour F et G, son choix n'est plus aléatoire pour $(F \bmod x^p - 1)$ et $(G \bmod x^p - 1)$ qui dépendent de lui. La propriété

probabiliste ne s'applique donc pas. Le polynôme $(F \bmod x^p - 1)(G \bmod x^p - 1)$ peut avoir plus de monômes que FG.

Exemple 5.11. Pour $F = x^7 - x^4$, $G = x^4 + x$ et $p = 5$. Le polynôme $FG = x^{11} - x^5$ a 2 monômes comme $FG \bmod x^5 - 1 = x - 1$. En revanche $(F \bmod x^5 - 1)(G \bmod x^5 - 1) = -x^8 + x^6 + x^3 - 1$ a 4 monômes non nuls. 8 est dans le support structurel mais pas arithmétique de FG et correspond à deux éléments, 3 et 8 dans le support de $(F \bmod x^5 - 1)(G \bmod x^5 - 1)$.

Dans $(F \bmod x^p - 1)(G \bmod x^p - 1)$, il peut y avoir des monômes distincts dont les exposants sont égaux modulo p. Comme le degré de ce polynôme est inférieur à $2p - 2$, ces monômes vont par paire, pas plus. Dans une paire, l'exposant inférieur à $p - 1$ correspond à un élément du support structurel de FG réduit modulo p. L'élément des paires d'exposants supérieur à p lui est surnuméraire. Enfin, chaque monôme qui n'est pas en paire correspond à exactement un monôme de $(FG) \bmod x^p - 1$ donc de FG. Ainsi il reste possible de borner le nombre de monômes de $(F \bmod x^p - 1)(G \bmod x^p - 1)$ par $2S$ où S désigne la cardinalité du support structurel.

Voici néanmoins cet algorithme d'interpolation qui sera suivi de l'analyse corrigée.

Algorithme 22 ProduitStructurel

Entrée : $F, G \in \mathbb{F}_{q^s}[x]$ avec $q > \deg(F) + \deg(G)$, $0 < \epsilon < 1$.
Sortie : $R \in \mathbb{F}_{q^s}[x]$ tel que $R = FG$ avec une probabilité d'au moins $1 - \epsilon$.

1: $p \leftarrow \mathcal{O}((\#F\#G)^2 \ln(\deg(F) + \deg(G)))$ nombre premier aléatoire pour que FG soit sans collision modulo $x^p - 1$ avec probabilité au moins $1 - \frac{\epsilon}{2}$ ▷ corollaire 2.11
2: $F_p \leftarrow F \bmod X^p - 1$, $G_p \leftarrow G \bmod X^p - 1$
3: $F'_p \leftarrow F' \bmod X^p - 1$, $G'_p \leftarrow G' \bmod X^p - 1$
4: Avec un paramètre de probabilité $\frac{\epsilon}{2}$, appliquer la procédure 21 EssaisEtTests pour calculer le couple $(F_pG_p, F_pG'_p + F'_pG_p)$. Pour cela utiliser l'interpolation de Huang et la vérification de produit sur chacun des éléments. ▷ algorithmes 9, 18 et 21
5: $(R_p, R'_p) \leftarrow$ les polynômes du couple réduits modulo $x^p - 1$.
6: $R \leftarrow$ le polynôme nul
7: **Pour** $(cx^e, c'x^{e-1})$ monômes de degrés correspondant de R_p et R'_p **faire**
8: Ajouter $cx^{c'/c}$ à R
9: **Renvoyer** R

Théorème 5.12. *Soit F, $G \in \mathbb{F}_{q^s}[x]$ avec $q > D$ premier, $d = \deg(F) + \deg(G)$, S la cardinalité du support structurel de FG et $0 < \epsilon < 1$ une constante. Avec une probabilité de succès d'au moins $1 - \epsilon$, l'algorithme 22* ProduitStructurel *calcule correctement FG en effectuant $\tilde{\mathcal{O}}(S \log q^s)$ opérations binaires.*

Démonstration. Avec une probabilité d'au moins $1 - \frac{\epsilon}{2}$, EssaisEtTests renvoie les bons polynômes $(F_pG_p, F_pG'_p + F'_pG_p)$ en $\tilde{\mathcal{O}}(S(\log p \log q^s))$ d'après le corollaire 5.10. En effet le nombre de monômes non nuls dans ces polynômes est au plus linéaire en S puisque la discussion qui précède le théorème s'applique aussi à $F_pG'_p + F'_pG_p$. Le choix de p

garantit d'une part que cette complexité soit en fait en $\tilde{\mathcal{O}}(S \log q^s)$, d'autre part qu'avec une probabilité d'au moins $1 - \frac{\epsilon}{2}$ la boucle pour permette bien de reconstruire FG. En effet, il s'agit de la méthode de l'algorithme de Huang qui permet de reconstruire des monômes du polynôme recherché s'ils sont sans collision modulo $x^p - 1$. Comme il n'y a pas de collision, tous les monômes sont bien retrouvés. Ainsi avec une probabilité d'au moins $1 - \epsilon$ l'algorithme calcule FG en réalisant $\tilde{\mathcal{O}}(S \log q^s)$ opérations binaires plus le coût des réductions modulo $x^p - 1$ au début et de la reconstruction finale en $\tilde{\mathcal{O}}(S \log q^s)$. Il y a bien des opérations sur les exposants, mais comme $q > D$, leur coût est négligable par rapport à celui des opérations sur les coefficients. De même, $\#F, \#G$ et $\#(FG)$ n'apparaissent pas dans la complexité car ils sont plus petits que S. □

Par rapport au corollaire 5.10, cet algorithme présente une amélioration puisqu'il n'y a pas de facteur $\log D \log q^s$, mais aussi un recul puisque S est là pour le support structurel au lieu du vrai support.

5.3. Division exacte de polynômes creux

Après le produit, nous passons à la division exacte de polynômes creux. Si l'opération est restreinte à la division exacte, c'est pour pouvoir s'appuyer comme dans la section précédente sur l'interpolation. Or les algorithmes d'interpolation efficaces passent par des réductions modulo $x^p - 1$. Cette réduction rend difficile l'interpolation de deux polynômes simultanément en empêchant de distinguer ce qui est dû au reste et ce qui est dû au quotient. L'égalité de la division euclidienne $F = GQ + R$ avec une contrainte sur le degré du reste n'a en effet plus autant de sens modulo $x^p - 1$, tous les degrés ayant la même borne p. En particulier elle ne permet plus de définir Q et R de manière unique. C'est assez radical puisque pour tout polynôme Q de degré $\deg(F) - \deg(G)$, il existe un polynôme R de degré inférieur à p (et donc en général à $\deg(G)$) qui permet de vérifier l'égalité $F \bmod x^p - 1 = (GQ + R) \bmod x^p - 1$. En revanche, si le reste est fixé à 0, l'égalité $F \bmod x^p - 1 = GQ \bmod x^p - 1$ donne toujours des contraintes sur Q. En particulier, si G est inversible modulo $x^p - 1$, le polynôme $Q \bmod x^p - 1$ est unique. Nous considérons donc ce cas précis où il est possible d'obtenir des informations sur le quotient modulo $x^p - 1$.

Cette remarque montre aussi qu'il ne suffira pas d'utiliser la procédure EssaisEtTests comme pour le produit. Il faudra en plus adapter les algorithmes d'interpolation pour qu'ils puissent calculer le polynôme Q défini par une division. L'utilisation d'algorithmes de vérification de produit ne pose quant à elle aucune difficulté.

En adaptant les algorithmes d'interpolation, il est possible d'obtenir des algorithmes qui sont quasi-linéaires dans la cardinalité du support du quotient. Pour des polynômes à coefficients entiers, l'algorithme obtenu est quasi-linéaire. Les complexités dépendent de l'algorithme d'interpolation tout comme l'algorithme de division les adaptations dépendant des techniques d'interpolation utilisées.

5.3.1. Adaptation de l'interpolation sur les entiers

La division exacte de F par G pour deux polynômes dans $\mathbb{Z}[x]$ peut être vue comme un circuit arithmétique avec une seule division. Ceci soulève de nouvelles difficultés par rapport à l'adaptation de l'interpolation pour l'algorithme de produit, en particulier le risque d'une division par 0.

5.3.1.1. Les obstacles à la division

La première difficulté rencontrée pour adapter un algorithme d'interpolation est en fait commune au produit et à la division : l'absence de borne fine sur le nombre de monômes du résultat. Pour le quotient, la seule borne a priori est même excessivement large, il s'agit du degré. Autrement dit, en se fiant à cette borne il serait plus efficace de traiter les polynômes comme des polynômes denses pour calculer le quotient. Pour considérer le quotient comme un polynôme creux, l'approche par la procédure EssaisEtTests vue pour le produit fonctionne encore : tenter l'interpolation avec des bornes T sur la cardinalité du quotient croissante et s'arrêter dès que l'algorithme de vérification de produit assure que l'on a trouvé un polynôme Q tel que $F = GQ$.

La seconde difficulté est propre à la division de polynômes à coefficients entiers. La borne a priori sur la hauteur du quotient dépend de son nombre de monômes de manière exponentielle (lemme 1.20, page 19). Cette borne peut elle aussi s'avérer très pessimiste. Une approche semblable réalisant des essais avec une borne de hauteur croissante suivis de vérifications permet de surmonter cette difficulté. Ceci présente une nouvelle difficulté, il y a maintenant deux bornes à tester et deux bornes à mettre à jour mais pas simultanément. Pour savoir quelle borne entre la hauteur et le nombre de monômes est sous-estimée et doit être augmentée, il faut effectuer les tests séparément. Il se trouve que la vérification liée à la hauteur du polynôme peut être effectuée au cours de l'algorithme d'interpolation plutôt qu'à son terme. En effet, l'étape 7 de l'algorithme 12 Interpolation_BNM (page 81) est déterministe. Elle calcule le polynôme modulo x^p-1 dans $\mathbb{Z}/q^{2k}\mathbb{Z}$ à partir du support modulo x^p-1. Si q^{2k} est plus petit que la hauteur du polynôme, certains coefficients sont réduits. En comparant le polynôme modulo x^p-1 et q^{2k} avec le polynôme juste modulo x^p-1, ces différences sont détectables ce qui conduit alors à augmenter la borne sur la hauteur. Le test qui permet cette comparaison est celui d'une égalité modulaire tel que décrit dans la section 4.2 (page 95).

Enfin, la dernière difficulté vient de l'évaluation elle-même. Le polynôme à interpoler est F/G. Pour cela, l'algorithme 12 Interpolation_BNM effectue trois séries d'évaluations. Le polynôme F/G doit d'abord être évalué sur des racines p-ièmes de l'unité ω dans $\mathbb{F}_q$, puis sur des racines p-ièmes de l'unité ω_k dans $\mathbb{Z}/q^{2k}\mathbb{Z}$. Enfin le polynôme transformé $F((1+q^k)x)/G((1+q^k)x)$ doit être évalué sur les ω_k. Pour pouvoir effectuer les divisions, il faut à chaque fois que les points concernés ne soient pas des racines de G ou de $G((1+q^k)x)$ dans l'anneau d'évaluation. Si ω_k est une racine de G modulo q^{2k}, alors $G(\omega_k)$ est un multiple de q^{2k} donc de q et de plus $G(\omega_k \bmod q)$ est aussi un multiple de q. Or d'après le lemme 3.16 (page 79), $\omega_k \bmod q$ est une racine primitive p-ième de l'unité

dans $\mathbb{F}_q$ qui est donc aussi racine de G. Par ailleurs si ω_k est racine de $G((1+q^k)x)$ modulo q^{2k}, on obtient le même résultat à savoir que $\omega_k \bmod p$ est une racine de G modulo q. Ainsi, il suffit de s'assurer qu'aucune des racines primitives p-ièmes de l'unité n'est racine de G dans $\mathbb{F}_q$. Autrement dit de s'assurer que les polynômes G et $\Phi_p = \sum_{j=0}^{p-1} x^j$ sont premiers entre eux dans $\mathbb{F}_q[x]$. Cela peut se faire en réglant convenablement le choix de p et q durant l'algorithme : d'une part choisir p pour qu'ils soient premiers entre eux dans $\mathbb{Z}[x]$, d'autre part choisir q pour qu'ils le restent modulo q.

Lemme 5.13. *Soit G un polynôme de degré d, $0 < \epsilon < \frac{1}{2}$. Si p et q sont deux nombres générés par l'algorithme 1* TRIPLET *(page 33) à partir d'un paramètre λ supérieur à $\frac{5}{\epsilon}(\#G-1)\ln d$ et $\sqrt[4]{\frac{96}{\epsilon}\ln(\#G\,\|G\|)}$, alors la probabilité que G et $\Phi_p = \sum_{j=0}^{p-1} x^j$ soient premiers entre eux dans $\mathbb{F}_q[x]$ est d'au moins $1-2\epsilon$.*

Démonstration. Si l'algorithme TRIPLET parvient à générer des nombres premiers p et q, alors p est choisi uniformément parmi les premiers de l'intervalle $[\lambda, 2\lambda]$ (théorème 2.17, page 32). Puisque $\lambda \geq \frac{5}{\epsilon}(\#G-1)\ln d$, le corollaire 2.9 (page 29) indique que la probabilité d'avoir $G \bmod x^p - 1 \neq 0$ est d'au moins $1-\epsilon$. Tout facteur commun à G et Φ_p est aussi commun à $G \bmod x^p - 1 \neq 0$ et Φ_p. Puisque Φ_p est irréductible dans $\mathbb{Z}[x]$, le seul facteur propre à considérer est Φ_p lui-même. Or le degré de $G \bmod x^p - 1$ est au plus égal au degré de Φ_p. Pour qu'il y ait un facteur commun, il faudrait donc avoir $G \bmod x^p - 1 = n\Phi_p$ pour un entier $n \in \mathbb{Z}$, $n \neq 0$. Les deux polynômes doivent donc avoir le même nombre de monômes. Le polynôme $G \bmod x^p - 1$ en a au plus $\#G$ et Φ_p en a p. Il est donc impossible d'avoir $G \bmod x^p - 1 = n\Phi_p$ avec $n \neq 0$ puisque p étant premier et supérieur à λ, il est toujours strictement plus grand que $\#G$. L'inégalité $p > \#G$ se vérifie en considérant d'abord les cas avec $\#G$ ou d petit, et ensuite les cas sans bornes supérieures. Si $\#G = 1$, alors tout nombre premier p vérifie $p > \#G$. Si $\#G = 2$ et $d = 1$ alors $\lambda \geq \sqrt[4]{\frac{96}{\epsilon}\ln(2\,\|G\|)} \geq \sqrt[4]{192\ln 2} \geq 2,58$, ainsi tout nombre supérieur à λ est supérieur à $\#G$. Si $\#G \geq 1$ et $d \geq 2$ alors on a $\lambda \geq \frac{5}{\epsilon}(\#G-1)\ln d \geq 10\ln 2(\#G-1) > 2(\#G-1) \geq \#G$. Tout ceci permet de conclure qu'avec une probabilité d'au moins $1-\epsilon$, G et Φ_p sont premiers entre eux dans $\mathbb{Z}[x]$.

Pour qu'ils soient aussi premiers entre eux dans $\mathbb{F}_q[x]$, il suffit de s'assurer que q ne divise pas le résultant de G et Φ_p que l'on sait être non nul. Or le résultant de G et Φ_p est donné par les évaluations de G sur les racines de Φ_p : $Res(G, \Phi_p) = \prod_{j=1}^{p-1} G(e^{\frac{ij\pi}{p}})$. Puisque les évaluations sont sur des éléments de norme 1, $Res(G, \Phi_p)$ est borné en valeur absolue par $(\#G\,\|G\|)^{p-1}$. Si $\lambda \geq \sqrt[5]{\frac{48}{\epsilon}\ln((\#G\,\|G\|)^{p-1})}$, il est possible d'appliquer le théorème 2.17 pour affirmer qu'avec une probabilité d'au moins $1-\epsilon$, q ne divise pas $Res(G, \Phi_p)$. Or $p \leq 2\lambda$ donc $\sqrt[5]{\frac{48}{\epsilon}\ln((\#G\,\|G\|)^{p-1})} \leq \sqrt[5]{\frac{96}{\epsilon}\lambda\ln(\#G\,\|G\|)}$ et il faudrait que ce soit inférieur à λ. En posant $a = \frac{96}{\epsilon}\ln(\#G\,\|G\|)$, l'inégalité qu'il faut obtenir devient $a^{1/5}\lambda^{1/5} \leq \lambda$. Or par hypothèse, $\lambda \geq a^{1/4}$ d'où $a^{1/5}\lambda^{1/5} \leq \lambda^{4/5}\lambda^{1/5} \leq \lambda$. Ainsi λ est assez grand pour pouvoir affirmer qu'avec une probabilité d'au moins $1-\epsilon$, q ne

divise pas $Res(G, \Phi_p)$ s'il est non nul.

Pour résumer, avec une probabilité d'au moins $1 - \epsilon$, G et Φ_p sont premiers entre eux dans $\mathbb{Z}[x]$. Avec une probabilité d'au moins $1 - \epsilon$, ils le restent dans $\mathbb{F}_q[x]$. Ceci donne bien une probabilité d'au moins $1 - 2\epsilon$ qu'ils soient premier entre eux dans $\mathbb{F}_q[x]$. □

Le cas où 1 serait une racine de G peut aussi sembler problématique puisque les évaluations ont lieu sur des puissance de ω à partir de ω^0. Cependant, décaler la suite des puissance en commençant à ω^1 ne change pas la validité des algorithmes appelés sur ces différentes suites. Il suffit donc de veiller à ce que les racines primitives de l'unité d'ordre p ne soit pas des racines de G, comme le fait le lemme 5.13

5.3.1.2. Un algorithme quasi-linéaire

En s'appuyant sur les remarques précédentes, il est possible d'adapter l'algorithme 12, INTERPOLATION_BNM pour calculer le quotient d'une division exacte de polynômes creux. Le premier algorithme décrit correspond à une application directe avec une borne connue sur le nombre de monômes du quotient et une probabilité de succès constante.

Algorithme 23 QUOTIENTAVECBORNET

Entrée : Deux polynômes creux F, $G \in \mathbb{Z}[x]$ tel que F est de degré D et G divise F ; un entier T

Sortie : F/G avec une probabilité d'au moins $\frac{2}{3}$ dans le cas où $T \geq \#(F/G)$

1: $H_{max} \leftarrow (1 + \|G\|)^{\lceil \frac{1}{2}(T-1)\rceil} \cdot \|F\|$
2: $\epsilon \leftarrow \frac{1}{18}(\lceil \log T \rceil + \lceil \log \log H_{max} \rceil)$; $C \leftarrow \max(2H_{max}, \#G \, \|G\|)$
3: $\lambda \leftarrow \max\left(\frac{2^{58}}{\epsilon^2}, \frac{5}{\epsilon}(\max(T, \#G) - 1) \ln D, \sqrt[4]{\frac{96}{\epsilon} \ln C}\right)$
4: $Q \leftarrow 0$; $H_0 \leftarrow \|G\| + 1$
5: **Tant que** $T \geq 1$ **faire**
6: $(p, q, \omega) \leftarrow$ TRIPLET(λ, ϵ) : p, q premiers, $\lambda < p < 2\lambda$, $q \leq \lambda^6$, $\omega \in \mathbb{F}_q$ racine primitive p-ième de l'unité
7: **Si** l'exécution de TRIPLET échoue **alors Renvoyer échec**
8: Calculer $Q_p = (F/G - Q) \bmod \langle x^p - 1, q^{2k} \rangle$ comme dans l'algorithme 12, avec $k = \lceil \max(\frac{1}{2} \log_q(4TH_0 \, \|F\|), \log_q D) \rceil$
9: **Si** une division par 0 survient **alors Renvoyer échec**
10: Vérifier si $F \bmod x^p - 1 = G \times (Q_p + Q) \bmod x^p - 1$ dans $\mathbb{Z}$ avec une probabilité d'erreur $\leq \epsilon$ ▷ théorème 4.20
11: **Si** il y a bien égalité **alors**
12: Calculer de possibles monômes de $F/G - Q$ ▷ algorithme 12, étapes 9 et 10
13: Mettre à jour Q, avec des monômes de hauteur au plus H_{max}
14: $T \leftarrow \lfloor T/2 \rfloor$
15: **Sinon** $H_0 \leftarrow H_0^2$
16: **Renvoyer** H

Si l'algorithme ne renvoie pas F/G, ce peut être soit parce qu'il renvoie un message d'erreur si TRIPLET a échoué, soit parce qu'il renvoie un polynôme erroné. Un tel polynôme peut contenir des monômes qui n'appartiennent pas au vrai quotient. Cependant sa taille ne peut pas être beaucoup plus importante que celle du polynôme correct.

Lemme 5.14. *Si l'algorithme 23* QUOTIENTAVECBORNET *renvoie un polynôme, c'est toujours un polynôme ayant au plus* $2T$ *monômes de coefficients bornés par* $T \cdot tH$ *avec* t *et* H *le véritable nombre de monômes et la véritable hauteur du quotient* F/G *que l'algorithme essaie de calculer.*

Démonstration. Pour le nombre de monômes, il faut remarquer que les méthodes de l'algorithme 12 INTERPOLATION_BNM utilisées lors de l'étape 8 reposent sur un système de type Vandermonde pour interpoler un polynôme creux ayant au plus T monômes. C'est à dire que le système est de taille T et ne permet pas de calculer plus que T monômes. Comme à chaque fois que des termes sont ajoutés à Q, la borne T est divisée par 2, même si tous les monômes sont erronés il ne peut y en avoir au maximum que $2T$.

Concernant la hauteur du polynôme, seuls les coefficients de monômes erronés peuvent dépasser en valeur absolue H, la hauteur de F/G. De tels monômes viennent nécessairement de collisions modulo $x^p - 1$. Ainsi à chaque itération, la somme des coefficients erronés en valeur absolue est au plus égale à la somme des coefficients en valeur absolue de $F/G - Q$. Par conséquent l'ajout de monômes erronés peut au plus doubler la somme des coefficients de $F/G - Q$ pour l'itération suivante. Au départ, $Q = 0$ et la somme des coefficients de $F/G - Q$ est bornée par tH puisque tous les monômes viennent de F/G. Après $\lceil \log T \rceil$ itérations, la somme est bornée par $T \cdot tH$. Il en est donc de même des coefficients de Q qui contribuent soit à cette somme soit à celle d'une itération précédente qui est plus petite. □

Théorème 5.15. *Pour des polynômes* $F, G \in \mathbb{Z}[x]$ *tels que* G *divise* F *et* $T \geq \#(F/G)$*, l'algorithme 23* QUOTIENTAVECBORNET *calcule le quotient* F/G *avec une probabilité d'au moins* $\frac{2}{3}$*. Il effectue* $\tilde{\mathcal{O}}((T + \#F + \#G)(\log D + \log H))$ *opérations binaires avec* $D = \deg(f)$ *et* $H = \max(\|F\|, \|G\|, \|F/G\|)$*.*

Pour tout $\rho \geq 1$*,* $\mathcal{O}(\rho)$ *répétitions de l'algorithme permettent d'augmenter la probabilité de succès à* $1 - \frac{1}{2^\rho}$ *en sélectionnant le résultat majoritaire.*

Démonstration : Validité. L'algorithme peut ne pas renvoyer le polynôme F/G pour cinq raisons différentes. Les trois premières sont communes avec l'algorithme 12 INTERPOLATION_BNM : soit le calcul du triplet (p, q, ω) (étape 6) est un échec, soit le nombre premier p entraîne trop de collisions dans $F/G - Q$ modulo $x^p - 1$, soit certains coefficients non nuls de $F/G - Q \bmod (x^p - 1)$ sont annulés par la réduction modulo q. Les deux autres sources d'échec sont propres à l'algorithme de division : l'une des puissances de ω peut être une racine de G entraînant une division par 0 ou le test de l'étape 10 peut échouer à détecter une erreur.

Tel que défini, λ vérifie $\lambda \geq \max\left(\frac{2^{58}}{\epsilon^2}, \frac{5}{\epsilon}(T-1)\ln D, \sqrt[5]{\frac{48}{\epsilon}T\ln H_{max}}\right)$. Le dernier terme est obtenu en utilisant le fait que $T < 2\lambda$. Or d'après le lemme 1.20 (page 19), H_{max} est une borne sur la hauteur de F/G. Ainsi, les conditions du lemme 3.14 (page 77) sont vérifiées pour le polynôme F/G. Il permet d'affirmer qu'avec une probabilité d'au moins $1-3\epsilon$, aucune des trois premières source d'échec n'arrive : l'étape 6 renvoie un triplet (p, q, ω) tel que p entraîne suffisamment peu de collisions pour que le nombre de monômes à retrouver soit divisé par deux, et que q n'annule aucun des coefficients de $F/G - Q \bmod x^p - 1$

Par ailleurs, le choix de λ par rapport à $\#G$ et $\|G\|$ permet d'appliquer le lemme 5.13 au triplet (p, q, ω) pour affirmer qu'avec une probabilité d'au moins $1-2\epsilon$, G n'admet pas comme racine une racine primitive p-ième de l'unité modulo q et donc pas non non plus modulo q^{2k}.

Si tous ces points sont vérifiés, il est possible d'appliquer les faits 3.3 à 3.5 (page 63) pour calculer Q_p à l'étape 8 de manière déterministe comme dans l'algorithme d'interpolation. Le choix de k garantit que q^{2k} est au moins deux fois plus grand que la hauteur de $(F/G-Q) \bmod x^p-1$ dès que $H_0 \|F\|$ est supérieur à $\|F/G\|$ (dont la valeur exacte est inconnue). C'est pour que la comparaison s'applique bien à la hauteur $(F/G-Q) \bmod x^p-1$ et pas seulement à celle de $F/G - Q$, que la dépendance en T a été ajoutéé par rapport à l'algorithme d'interpolation. Dans ce cas, l'égalité $Q_p = (F/G - Q) \bmod x^p - 1$, valide modulo q^{2k}, est aussi vérifiée dans $\mathbb{Z}[x]$ et le test de l'étape 10 renvoie **vrai**. Le calcul de potentiels monômes et la mise à jour de Q sont alors effectués exactement comme dans l'algorithme 12 Interpolation_BNM.

Si $H_0 \|F\| < \|F/G\|$, il y a deux possibilités. La première est que $Q_p \neq (F/G - Q) \bmod x^p - 1$ dans $\mathbb{Z}[x]$. Alors, avec probabilité d'au moins $1-\epsilon$, le test de l'étape 10 détecte cette différence et la borne H_0 est augmentée en H_0^2. La seconde possibilité est que l'égalité $Q_p = (F/G - Q) \bmod x^p - 1$ soit quand même vraie. Ceci implique que les monômes de hauteur supérieure à H_0 sont en collision modulo $x^p - 1$. Les termes sans collision sont eux correctement calculés.

Ainsi, avec une probabilité d'au moins $1-6\epsilon$, soit le nombre de monômes à reconstruire a été divisé par deux, soit la borne sur la hauteur a été élevée au carré si elle était trop petite. Il faut au plus $\lceil \log\log \|F/G\| \rceil \leq \lceil \log\log H_{max} \rceil$ itérations où le test de l'étape 10 renvoie **faux** pour atteindre une borne correcte sur la hauteur. À celles-là, s'ajoutent les $\lceil \log T \rceil$ itérations où le test renvoie **vrai** nécessaires pour calculer tous les coefficients de F/G.

L'algorithme effectue donc au plus $(\lceil \log T \rceil + \lceil \log\log H_{max} \rceil)$ itérations. La probabilité de succès totale de l'algorithme est donc d'au moins $1-5\epsilon(\lceil \log T \rceil + \lceil \log\log H_{max} \rceil) \geq \frac{2}{3}$. Pour améliorer cette probabilité à $1 - \frac{1}{2^\rho}$, il suffit de répéter l'algorithme $48\rho/\log e$ fois et de renvoyer le résultat majoritaire (fait 1.3). □

Démonstration : Complexité. Puisque le nombre d'itérations n'ajoute à la complexité qu'un facteur logarithmique dans la taille des entrées et de la sortie, concentrons nous

donc sur le coût d'une itération.

Les évaluations de F/G (données par celles de F et de G) et de Q nécessitent $\tilde{\mathcal{O}}((T + \#F + \#G)\log p)$ opérations dans chacun des anneaux d'évaluation par le fait 3.4 plus $\mathcal{O}(T(\mathsf{I}(\log d) + \mathsf{I}(\log q^{2k})))$ opérations binaires pour les réductions préalables des polynômes (discutées au début de la section 5.2.1).

À partir des évaluations, l'algorithme, comme celui d'interpolation, effectue $\tilde{\mathcal{O}}(T + p)$ opérations dans $\mathbb{F}_q$ et $\tilde{\mathcal{O}}(T \log p)$ opérations dans $\mathbb{Z}/q^{2k}\mathbb{Z}$ pour calculer les exposants modulo p et retrouver les coefficients et exposants sans borne.

Puisque la hauteur d'un résultat erroné est d'au plus T^2H d'après le lemme 5.14, H_0 n'est augmenté que jusqu'à dépasser cette borne. La valeur maximale de q^{2k} est donc $\mathcal{O}(T^3H + D)$. Ainsi une opération arithmétique dans $\mathbb{Z}/q^{2k}\mathbb{Z}$ a un coût binaire de $\tilde{\mathcal{O}}(\log H + \log D)$. De plus, le choix de λ garantit que $p = \tilde{\mathcal{O}}((T + \#G)(\log D + \log H))$. Par conséquent, la complexité totale est de $\tilde{\mathcal{O}}((T + \#F + \#G)(\log D + \log H))$ opérations binaires, q étant polynomial en p. □

L'algorithme 23 QuotientAvecBorneT correspond à un algorithme d'interpolation d'un quotient exact de deux polynômes creux si une borne sur son nombre de monômes est connu. En travaillant avec une borne croissante et en testant la validité des résultats obtenus, il permet d'obtenir un algorithme pour calculer le quotient même dans les cas où la borne n'est pas connue. Il s'agit d'une procédure de type EssaisEtTests, comme celle utilisée pour le produit en section 5.2.

Algorithme 24 DivisionExacte

Entrée : Deux polynômes creux F, $G \in \mathbb{Z}[x]$ tel que F est de degré D et G divise F ; $\rho \geq 1$
Sortie : F/G avec une probabilité d'au moins $1 - \frac{1}{2^{\rho+1}}$

1: $T \leftarrow 1$
2: **Tant que vrai faire**
3: $\quad T \leftarrow 2T$
4: $\quad$ Calculer $\mathcal{O}(\rho)$ candidats Q pour F/G par QuotientAvecBorneT(F, G, T).
5: $\quad$ Garder le candidat le plus fréquent
6: $\quad$ Tester si $F = GQ$ avec une probabilité d'échec d'au plus $\frac{1}{2^{\rho+1}T}$ ▷ corollaire 4.44
7: $\quad$ **Si** le test renvoie **vrai alors**
8: $\quad\quad$ **Renvoyer** R

Théorème 5.16. *Soit F, G deux polynômes creux dans $\mathbb{Z}[x]$ tels que G divise F et $\rho \geq 1$. Avec une probabilité d'au moins $1 - \frac{1}{2^\rho}$, l'algorithme* DivisionExacte *renvoie F/G en effectuant $\tilde{\mathcal{O}}(t(\log D + \log H)\rho + \rho^4)$ opérations binaires avec $D = \deg(f)$, $t = \max(\#F, \#G, \#(F/G))$ et $H = \max(\|F\|, \|G\|, \|F/G\|)$.*

Démonstration. La preuve est la même que celle du théorème 5.6, en remplaçant l'interpolation par l'algorithme QuotientAvecBorneT. Le lemme 5.14 assure que la taille

des polynômes erronés fournis lors de tentatives infructueuses reste quasi-linéaire et n'impacte pas la complexité. □

Cet algorithme est le premier algorithme quasi-linéaire de division de polynôme creux. Il a été obtenu en modifiant le fonctionnement de l'algorithme d'interpolation sur lequel il est basé pour permettre de gérer l'absence de bornes (hormis le degré) et empêcher les divisions par 0.

5.3.2. Adapter des interpolations pour circuit arithmétique

Comme dans le cas du produit, il serait intéressant d'avoir une méthode générale pour définir une procédure de type EssaisEtTests qui permette d'adapter directement un algorithme d'interpolation en un algorithme de division exacte. Cependant l'algorithme précédent illustre bien qu'il faut vraiment prendre en compte le fonctionnement de l'algorithme d'interpolation et y apporter potentiellement des changements. Les transformations vers des algorithmes de division semblent plutôt devoir être conçues de manière ad hoc.

Dans la section précédente, nous avons présenté un algorithme quasi-linéaire pour des polynômes à coefficients entiers. Si les polynômes sont définis sur d'autres anneaux, cet algorithme ne fonctionne plus puisque l'interpolation n'est plus adaptée. De manière générale ce sont les algorithmes d'interpolation de circuit arithmétique qui ont les meilleures complexités sur un anneau quelconque. Cette section se consacre donc exclusivement à l'adaptation d'algorithmes d'interpolation de circuit arithmétique. De plus, nous nous restreignons à ceux qui, comme celui de Garg et Schost, passent par l'évaluation en x modulo $x^p - 1$ pour différents nombre premiers p. C'est pour cette évaluation que nous avons développé des adaptations.

5.3.2.1. Permettre l'inversion modulo $x^p - 1$

Le "circuit arithmétique" considéré est le polynôme F/G, un circuit arithmétique avec une division à la fin. Pour pouvoir l'évaluer en x modulo $x^p - 1$, il est donc nécessaire de pouvoir inverser G modulo $x^p - 1$. Ici les polynômes sont définis sur un anneau intègre $\mathbb{A}$. Contrairement au cas des polynômes dans $\mathbb{Z}[x]$, l'anneau d'évaluation ne peut donc pas varier pour s'adapter à p. L'idée est alors de transformer le polynôme G pour qu'il soit inversible, c'est-à-dire premier avec $x^p - 1$. Pour cela, il est possible de s'appuyer sur la technique de diversification [GR11b], comme le décrit le lemme suivant.

Lemme 5.17. *Soit A et $B \in \mathbb{A}[x]$ deux polynômes non nuls avec $B(0) \neq 0$. Dans toute extension $\mathbb{A}_{ext}$ de $\mathbb{A}$, il y a au plus* $\deg(A)\deg(B)$ *points α distincts tels que $A(\alpha x)$ et $B(x)$ ont un facteur commun.*

Démonstration. Soit β une racine de B dans une clôture algébrique $\overline{\mathbb{A}}$ de $\mathbb{A}$. Le point β est une racine de $A(\alpha x)$ si et seulement si $A(\alpha\beta) = 0$, autrement dit si α est une racine

de $A(\beta x)$. Puisque $A(\beta x) \neq 0$ et $\deg(A(\beta x)) = \deg(A)$, il y a au plus $\deg(A)$ racines de $A(\beta x)$ dans $\overline{\mathbb{A}}$. Puisque B a au plus $\deg(B)$ racines dans $\overline{\mathbb{A}}$, il y a au plus $\deg(A)\deg(B)$ points α tels qu'il existe une racine β commune à $A(\alpha X)$ et $B(X)$. □

En interpolant $F(\alpha x)/G(\alpha x)$, plutôt que F/G, il est donc possible d'effectuer les divisions nécessaires. Ou, au moins, probablement possible si α est choisi aléatoirement dans une extension de $\mathbb{A}$ suffisamment grande. Les algorithmes d'interpolation de circuits arithmétiques effectuent plusieurs évaluations en x modulo $x^p - 1$ pour différents nombres premiers p. Puisqu'il y a plusieurs nombres premiers, l'objectif est d'obtenir un polynôme $G(\alpha x)$ qui soit premier avec tous les $x^p - 1$.

Corollaire 5.18. *Soit $G \in \mathbb{A}[x]$ un polynôme de degré D et $\mathcal{P}$ un ensemble de k nombres premiers plus petits que N et $0 < \epsilon < 1$. Si α est un élément choisi aléatoirement dans une extension $\mathbb{A}_{ext}$ de $\mathbb{A}$ de taille supérieure à $\frac{DkN}{\epsilon}$, alors la probabilité que $G(\alpha x)$ et $\prod_{p\in\mathcal{P}} (x^p - 1)$ soient premiers entre eux est d'au moins $1 - \epsilon$.*

Démonstration. Il s'agit de l'application du lemme 5.17 aux polynômes G et $\prod_{p\in\mathcal{P}} (x^p - 1)$. Le degré du second étant borné par kN. □

Les nombres premiers considérés sont choisis soit dans un intervalle $[\lambda, 2\lambda]$ soit parmi la liste des k plus petits nombres premiers. Les valeurs de λ et k sont généralement fixées en s'appuyant sur des résultats semblables aux corollaires 2.9, 2.11 et 2.13 (page 29, et 30). Le corollaire suivant spécifie ces deux cas.

Corollaire 5.19. *Soit $G \in \mathbb{A}[x]$ un polynôme de degré D. La probabilité que $G(\alpha x)$ et $\prod_{p\in\mathcal{P}} (x^p - 1)$ soient premiers entre eux est d'au moins $1 - \epsilon$ si α est choisi aléatoirement dans une extension $\mathbb{A}_{ext}$ de $\mathbb{A}$ de taille supérieure à*

- *$\frac{Dk^2 \log k}{\epsilon}$ si $\mathcal{P}$ est l'ensemble des k plus petits nombres premiers et $k > 3$.*
- *ou $\frac{14D\lambda^2}{5\epsilon \log \lambda}$ si $\mathcal{P}$ est l'ensemble des nombres premiers de l'intervalle $[\lambda, 2\lambda]$, $\lambda > 1$.*

Démonstration. Il s'agit simplement de borner la somme des k plus petits nombres premiers par $k^2 \log k$, et le nombre de premiers dans $[\lambda, 2\lambda]$ par $\frac{7\lambda}{5 \log \lambda}$ [RS62, Corollary 3]. □

5.3.2.2. Diviser grâce à des algorithmes d'interpolation

Si les évaluations du circuit arithmétique sont toutes des évaluations du polynôme modulo $x^p - 1$, l'algorithme d'interpolation avec une borne T peut être transformé en un algorithme de division exacte si une borne T sur le nombre de monômes du quotient est connue. Il suffit de choisir aléatoirement un point α dans une extension assez grande, de calculer $F(\alpha x)$, $G(\alpha x)$ puis de remplacer l'évaluation du circuit par la division. Le polynôme obtenu est alors $(F/G)(\alpha x)$ et F/G est obtenu grâce à une évaluation en $\alpha^{-1}x$.

Ceci ajoute une contrainte supplémentaire : il faut que α soit inversible. C'est donc au nombre d'éléments inversibles dans $\mathbb{A}_{\text{ext}}$ que les bornes précédentes s'appliquent. Dans le cas d'extensions de corps finis, ceci n'apporte pas de contrainte supplémentaire.

Algorithme 25 DivisionParInterpolation

Entrée : $F, G \in \mathbb{A}[x]$ tels que G divise F, $T > \#(F/G)$ et $0 < \epsilon < 1-\mu$; un algorithme $\mathcal{I}$ d'interpolation de polynômes creux par l'évaluation d'un circuit arithmétique modulo $x^p - 1$ pour différents nombres premiers p ayant une probabilité de succès d'au moins $1 - \mu$

Sortie : Le polynôme F/G avec une probabilité d'au moins $1 - (\epsilon + \mu)$

1: $k, N \leftarrow$ le nombre et la valeur maximale des nombres premiers utilisés par $\mathcal{I}$ pour un polynôme de degré $\deg(F) - \deg(G)$ ayant T monômes non nuls.
2: $\alpha \leftarrow$ un élément aléatoirement choisi parmi au moins $\frac{DkN}{\epsilon}$ inversibles d'une extension $\mathbb{A}_{\text{ext}}$ de $\mathbb{A}$.
3: $F_\alpha \leftarrow F(\alpha x)$; $G_\alpha \leftarrow F(\alpha x)$
4: Utiliser $\mathcal{I}$ pour calculer $Q_\alpha \leftarrow F_\alpha / G_\alpha$ en remplaçant l'évaluation du circuit en x modulo $x^p - 1$ par la réduction de F_α, G_α modulo $x^p - 1$ puis la division.
5: **Renvoyer** $Q_\alpha(\alpha^{-1}x)$

Théorème 5.20. *En appliquant la procédure* DivisionParInterpolation*, il est possible de calculer F/G, dont le nombre de monômes est connu, pour deux polynômes F et $G \in \mathbb{A}[x]$ en passant par l'interpolation d'un polynôme dans $\mathbb{A}_{ext}[x]$, si $\mathbb{A}_{ext}$ est telle que décrite dans la procédure. La probabilité d'échec est de $\epsilon + \mu$ où μ est la probabilité d'échec de l'algorithme d'interpolation utilisé.*

Pour la complexité, il faut modifier la complexité de l'algorithme d'interpolation en ajoutant $\mathcal{O}(t \log D)$ opérations dans $\mathbb{A}_{ext}$ et en remplaçant toute occurrence de $L \times \mathsf{M}(p)$ opérations dans $\mathbb{A}_{ext}[x]$ par $\mathcal{O}(T\mathsf{I}(\log D))$ opérations binaires et $\mathcal{O}(T)$ additions dans $\mathbb{A}_{ext}$ plus une division dans $\mathbb{A}_{ext}[x]/(x^p - 1)$ avec $t = \max(T, \#F, \#G)$, $D = \deg(F)$ et L la longueur du circuit.

Démonstration. La validité et la probabilité de succès se déduisent de la discussion qui précède la description de la procédure. Pour la complexité, il y a d'une part les trois évaluations en αx ou α^{-1} qui nécessitent chacune $\mathcal{O}(t \log D)$ opérations dans $\mathbb{A}_{\text{ext}}$ (fait 1.8). D'autre part, les évaluations en x modulo $x^p - 1$ qui demandent L opérations dans $\mathbb{A}_{\text{ext}}[x]/(x^p - 1)$, c'est à dire $L\mathsf{M}(p)$ opérations dans $\mathbb{A}_{\text{ext}}$ sont remplacés par deux réductions en $\mathcal{O}(T\mathsf{I}(\log D))$ opérations binaires et $\mathcal{O}(T)$ additions dans $\mathbb{A}_{\text{ext}}$ (fait 1.7) plus la division de $F_\alpha \bmod x^p - 1$ par $G_\alpha \bmod x^p - 1$. □

Il peut sembler qu'une telle approche fasse passer dans une large extension, mais la représentation de $\frac{DkN}{\epsilon}$ ne demande que $\log \frac{DkN}{\epsilon}$ bits. Cette taille reste quasi-linéaire dans la taille des polynômes considérés si $\frac{kN}{\epsilon}$ est polynomial dans cette même taille, ce qui est généralement le cas. En revanche effectuer $\log D$ opérations dans cette extension n'est pas quasi-linéaire. Il faut noter que c'est souvent plus rapide que l'évaluation du

circuit en x dans $\mathbb{A}_{\text{ext}}[x]/(x^p-1)$ car généralement p est au moins linéaire en $T\log D$. Si l'extension n'est pas trop complexe à produire, la division aura essentiellement le même coût que l'interpolation, en l'améliorant même du facteur L.

Cette procédure peut s'appliquer à l'algorithme 9 Interpolation de Huang (page 72) pour des polynômes sur des corps finis de grande caractéristique. Celui-ci évalue aussi le circuit de la dérivée. Il est donc aussi nécessaire de calculer F' et G' pour obtenir $(F/G)' = (F'G - FG')/G^2$ en ne divisant que par le polynôme G dont on s'assure déjà qu'il sera inversible.

Corollaire 5.21. *Soit $F, G \in \mathbb{F}_q[x]$ avec $\mathbb{F}_q$ de caractéristique supérieure à $\deg(F) - \deg(G)$, G qui divise F, T une borne sur $\#(F/G)$ et $0 < \epsilon < 1$ une probabilité constante. Il est possible de calculer F/G avec une probabilité d'échec d'au plus ϵ et en $\tilde{\mathcal{O}}(t\log D\log q)$ opérations binaires avec $t = \max(T, \#F, \#G)$ et $D = \deg(F)$.*

Démonstration. La probabilité d'échec peut être divisée en deux entre la probabilité que le choix de α ne garantisse pas l'inversibilité de G et la probabilité que l'interpolation échoue. L'algorithme 9 Interpolation de Huang choisit tous ses nombres premiers parmi les $k = \lceil 12(T-1)\log(\deg(F) - \deg(G))\rceil$ plus petits nombres premiers. Par conséquent α doit être choisi dans une extension de corps $\mathbb{F}_{q^s}$ avec $s = \left\lceil \log_q \frac{(\deg(F)-\deg(G))k^2\log k}{\epsilon} \right\rceil$. Soit $s = 1$, soit une opération dans $\mathbb{F}_{q^s}$ nécessite $\tilde{\mathcal{O}}(\log D/\log q)$ opérations dans $\mathbb{F}_q$, c'est à dire $\mathcal{O}(1)$ puisque $q > D$. L'ensemble des opérations propres à la division d'après le théorème 5.20 nécessite donc $\tilde{\mathcal{O}}(T\log D\log q)$ opérations binaires. Le coût de l'algorithme de Huang vient essentiellement de l'évaluation du circuit (voir théorème 3.11 page 71). Pour la division, il vient donc principalement des parties qui la remplacent, ce qui donne donc $\tilde{\mathcal{O}}(T\log D\log q)$. □

Pour les corps finis de petite caractéristique, le passage par une extension peut s'avérer plus coûteux. Cependant ce passage est déjà effectué par les algorithmes d'interpolation efficaces. En particulier le plus rapide ([AGR14], mentionné au théorème 3.8) utilise déjà la diversification et calcule $F(\alpha_i x) \bmod x^{p_j} - 1$ pour différents α_i et p_j. Puisqu'il y a diversification dans l'algorithme d'interpolation, en ajouter une au début peut certes garantir l'inversibilité de $G(\alpha x)$ modulo $x^p - 1$ mais pas celle de tous les $G(\alpha_i \alpha x)$. La bonne approche est plutôt pour chaque choix de α_i, de s'assurer de l'inversibilité de $G(\alpha_i x)$ modulo les différents $x^{p_j} - 1$. Les nombres premiers p_j sont tirés aléatoirement dans des intervalles $[\lambda, 2\lambda]$ avec $\lambda = \mathcal{O}(T\log D)$. Les points α_i, eux aussi choisis aléatoirement, se trouvent si nécessaire dans une extension de degré $s = \mathcal{O}(\log_q D)$ [AGR14, Lemma 3.1, Lemma 4.1]. Ces points ont une certaine probabilité de conduire à l'échec de l'algorithme d'interpolation. De plus, en se ramenant au cas de la division, il y a une certaine probabilité pour qu'un α_i ne permette pas à $G(\alpha_i x)$ d'être premier avec un $x^p - 1$ pour p premier dans $[\lambda, 2\lambda]$. Pour régler cette probabilité, il est possible d'augmenter s et de prendre plutôt $s = \mathcal{O}(\log_q \frac{D\lambda^2}{\epsilon})$ en appliquant le corollaire 5.19. Vu la valeur de λ, cela donne $s = \mathcal{O}(\log_q \frac{D}{\epsilon})$. Ainsi sans augmenter significativement la taille des extensions considérées, il est possible de garantir une bonne probabilité de pouvoir effectuer toutes

les divisions correspondant aux évaluations de l'algorithme d'interpolation. Puisque la taille de l'extension n'évolue que peu, adapter l'interpolation à la division donnera un algorithme ayant essentiellement la même complexité, au facteur L de la taille du circuit près.

Corollaire 5.22. *Soit $F, G \in \mathbb{F}_q[x]$ avec $\mathbb{F}_q$ de caractéristique inférieure à $\deg(F) - \deg(G)$, G qui divise F, T une borne sur $\#(F/G)$ et $0 < \epsilon < 1$ une probabilité constante. Il est possible de calculer F/G avec une probabilité d'échec d'au plus ϵ et en $\tilde{\mathcal{O}}(t(\log D)^2(\log D + \log q))$ opérations binaires avec $t = \max(T, \#F, \#G)$ et $D = \deg(F)$.*

Ces deux cas ont permis, sans entrer dans tous les détails, de voir qu'effectuer une division en s'appuyant sur l'interpolation nécessite de se pencher sur le fonctionnement de l'algorithme d'interpolation pour déterminer quelle partie précisément doit être modifiée. Une fois ces modifications réalisées, elles permettent d'obtenir des algorithmes quasi-linéaires dans une borne sur le nombre de monômes du résultat. Il est alors possible d'utiliser ces algorithmes $\mathcal{O}(\log \#(F/G))$ fois avec une borne croissante sur T en les associant à la vérification de produits de polynômes creux (théorème 4.43, page 116) pour obtenir un algorithme de division sans borne initiale sur le nombre de monômes du résultat. Il s'agit d'une nouvelle application de la procédure EssaisEtTests.

Corollaire 5.23. *Soit $F, G \in \mathbb{F}_q[x]$ tels que G divise F et $0 < \epsilon < 1$ une probabilité constante. Il est possible de calculer F/G avec une probabilité d'échec d'au plus ϵ et en*

- *$\tilde{\mathcal{O}}(T \log D \log q)$ opérations binaires si $\mathbb{F}_q$ est de caractéristique supérieure au degré du quotient $\deg(F) - \deg(G)$*
- *ou $\tilde{\mathcal{O}}(T(\log D)^2(\log D + \log q))$ opérations binaires sinon,*

avec $T = \max(\#(F/G), \#F, \#G)$ et $D = \deg(F)$.

Comme dans le cas du produit, les algorithmes obtenus sont quasi-linéaires en T le nombre de monômes des polynômes ce qui est un progrès par rapport aux algorithmes basés sur la division naïve. En revanche, ils ne sont pas quasi-linéaire dans la taille des polynômes, il faudrait avoir $\tilde{\mathcal{O}}(T(\log D + \log q))$. La division exacte est quasi-linéaire uniquement sur les entiers notamment car elle repose sur un algorithme d'interpolation quasi-linéaire.

De manière générale, les algorithmes que nous avons présentés dans ce chapitre pour le produit ou la division exacte sont quasi-linéaires par rapport au nombre de monômes non nuls des polynômes en entrée et du résultat. Dans le cas des polynômes à coefficients entiers, ils sont même quasi-linéaires dans la taille binaire des polynômes considérés. Si ce n'est pas également le cas sur les corps finis c'est que nous réduisons l'arithmétique à l'interpolation qui n'est pas aussi efficace sur les corps finis que sur les entiers.

6. Polynômes multivariés et substitution de Kronecker

Les résultats nouveaux présentés jusqu'ici ne concernent que des polynômes à une seule variable. Or de nombreux usages des polynômes creux concernent des polynômes multivariés. Face à des polynômes multivariés, il y a plusieurs approches possibles : soit conserver le caractère multivarié et gérer astucieusement les monômes en les regroupant par blocs pertinents [Yan98 ; HL12] ; soit passer sur des polynômes univariés et utiliser les algorithmes efficaces dans ce cas. Ce court chapitre traite précisément de la réduction d'un problème à plusieurs variables à un problème à une seule variable. Comme indiqué dans la section 1.2.3, le passage s'effectue généralement par une transformation de Kronecker dont la définition est rappelée ici.

Définition 13. La *substitution de Kronecker* est la bijection K_D entre les polynômes dans $\mathbb{A}[x_0, \ldots, x_{n-1}]$ de degré strictement inférieur à D en chaque variable et les polynômes dans $\mathbb{A}[x]$ de degré strictement inférieur à D^n, définie par $K_D(x_i) = x^{D^i}$ et étendue par linéarité.

Cette transformation a l'avantage d'être compatible avec les opérations arithmétiques élémentaires pourvu que la règle sur le degré soit toujours respectée. Le terme *compatible* pour une opération $\star$ entre polynômes et K_D signifie ici que $\star$ et K_D commutent. En particulier, c'est le cas des opérations étudiées durant cette thèse, puisque pour trois polynômes F, G, H de degré en chaque variable borné par D, $K_D(F) = K_D(G)K_D(H)$ si et seulement si $F = GH$.

L'approche générale est explicitée dans la procédure suivante.

Algorithme 26 RéductionÀ1Variable

Entrée : Une liste de polynômes $(F_1, \ldots F_k) \in \mathbb{A}[x_0, \ldots, x_{n-1}]$, $\star$ une opération compatible avec la substitution de Kronecker disposant d'un algorithme pour l'effectuer dans $\mathbb{A}[x]$.

Sortie : $\star(F_1, \ldots, F_k)$

1: $D \leftarrow$ une borne sur le maximum entre les degrés en chaque variable des F_i et les degrés en chaque variable du polynôme résultat $\star(F_1, \ldots, F_k)$
2: Calculer $R = \star(K_D(F_1), \ldots K_D(F_k))$
3: **Renvoyer** $K_D^{-1}(R)$

Cette procédure s'applique à des opérations arithmétiques, mais il est aussi possible

de s'appuyer sur la substitution de Kronecker pour l'interpolation. En effet, la substitution de Kronecker est partiellement compatible avec l'évaluation, opération courante des algorithmes d'interpolation. Si $F_u = K_D(F)$ pour $F \in \mathbb{A}[x_0, \ldots, x_{n-1}]$, et ω est un point de $\mathbb{A}$, alors $F_u(\omega) = F(\omega, \omega^D, \ldots, \omega^{D^{n-1}})$. Ainsi le polynôme univarié peut être évalué en tout point de $\mathbb{A}$ grâce à une évaluation du polynôme multivarié. La réciproque n'est pas vrai : l'évaluation de F en tout point $(\omega_0, \ldots, \omega_{n-1})$ de $\mathbb{A}^n$ ne pouvant être obtenue à partir d'une seule évaluation de F_u.

L'intérêt de la substitution de Kronecker est son coût limité et le peu d'impact qu'elle a sur la taille des polynômes manipulés.

Fait 6.1. *Si F est un polynôme creux multivarié de T monômes avec des coefficients de B bits et une borne D sur les degrés en chaque variable alors $F_u = K_D(F)$ est un polynôme creux univarié de T monômes avec les mêmes coefficients, de degré inférieur à D^n et donc de taille $T(n \log D + B)$. Le calcul de $K_D(F)$ ou de $K_D^{-1}(F_u)$ nécessite $\mathcal{O}(Tn \log D \log^2(n \log D)) = \tilde{\mathcal{O}}(Tn \log D)$ opérations binaires.*

Démonstration. La taille découle de la définition de Kronecker. Pour le coût, il suffit de voir que le passage d'un monôme $a \prod_i x_i^{e_i}$ au monôme $a x^{\sum_i e_i D^i}$ correspond à un changement de base entre l'entier $\sum_i e_i D^i$ et sa représentation en base D : $(e_0, \ldots, e_{n-1})$. Pour un entier de b bits, ce changement s'effectue en $\mathsf{I}(b) \log b$ opérations binaires [BZ10]. □

Le polynôme multivarié F est lui-même de taille $T(n \log D + B)$. La taille reste donc inchangée et le coût de la transformation est quasi-linéaire dans la taille de F. Si D est une puissance de 2, ce coût est même linéaire.

Les points discutés dans ce chapitre sont l'utilisation de la substitution de Kronecker dans les cadres évoqués dans cette thèse : interpolation (section 6.1) et opérations arithmétiques (section 6.2). Il serait aussi possible d'utiliser des algorithmes d'interpolation de polynômes multivariés pour effectuer les opérations arithmétiques. Cependant cette solution est moins efficace au vu du coût de la substitution de Kronecker et des algorithmes d'interpolation de polynômes univariés.

6.1. Interpolation

Les algorithmes d'interpolation pour un polynôme univarié passent généralement par son évaluation en plusieurs points soit dans l'anneau d'origine, soit dans une extension, soit dans une réduction modulaire (pour les polynômes dans $\mathbb{Z}[x]$). La relation $F_u(\omega) = F(\omega, \omega^D, \ldots, \omega^{D^{n-1}})$ pour $F \in \mathbb{A}[x_0, \ldots, x_{n-1}]$ de degré strictement inférieur à D en chaque variable, donne un moyen d'évaluer F_u qui est univarié en passant par l'évaluation de F. Ainsi, l'interpolation de F peut être effectuée en s'appuyant sur des algorithmes pour polynôme univarié et en reconstruisant F.

Théorème 6.2. *L'interpolation d'un polynôme $F \in \mathbb{A}[x_0, \ldots, x_{n-1}]$, donné de manière implicite (une boîte noire ou un circuit arithmétique) avec un degré strictement inférieur à D en chacune des variables peut être réalisée grâce à un algorithme d'interpolation univarié. Cela nécessite le même nombre d'évaluations et d'opérations supplémentaires que pour un polynôme univarié de degré D^n plus $\tilde{\mathcal{O}}(Tn \log D)$ opérations binaires.*

Démonstration. Il s'agit simplement d'interpoler le polynôme $F_u = K_D(F)$ en évaluant la boîte noire ou le circuit en $(\omega, \omega^D, \ldots, \omega^{D^{n-1}})$ à chaque fois qu'une évaluation en ω est nécessaire, puis de calculer $F = K_D^{-1}(F_u)$. Le coût vient du fait que la borne sur le degré de F_u utilisée pour l'interpolation est D^n. □

Globalement, il suffit de remplacer les facteurs $\log D$ dans la complexité de l'interpolation univariée par $n \log D$ pour obtenir la complexité d'un algorithme pour des polynômes multivariés. En particulier, pour des polynômes à coefficients entiers, l'algorithme présenté dans cette thèse (section 3.2) associé à la substitution de Kronecker fournit un algorithme efficace d'interpolation pour des polynômes multivariés.

Corollaire 6.3. *Soit $F \in \mathbb{Z}[x_0, \ldots, x_{n-1}]$, avec des bornes T, D, H sur son nombre de monômes, son degré en chaque variable et sa hauteur et $0 < \epsilon < 1$. Il est possible de calculer F avec une probabilité d'échec d'au plus ϵ en*

- *$\mathcal{O}(T \log \frac{1}{\epsilon})$ évaluations de la boîte noire, et $\tilde{\mathcal{O}}(T(n \log D + \log H) \log \frac{1}{\epsilon})$ opérations binaires si F est décrit par une boîte noire modulaire*
- *$\tilde{\mathcal{O}}(LT(n \log D + \log H) \log \frac{1}{\epsilon})$ si F est décrit par un circuit arithmétique dont les constantes sont bornées par H.*

Démonstration. La complexité vient du théorème 3.18 (page 81) et du corollaire 3.20 (page 84) en prenant D^n pour le degré et en comptant les $\mathcal{O}(\log \frac{1}{\epsilon})$ répétitions nécessaires pour atteindre la complexité escomptée. □

Il faut cependant faire attention que la taille des évaluations change. Pour un polynôme univarié de degré D^n et de hauteur H, les plus grandes évaluations sont effectuées sur des nombres de plus de $\max(D^n, H)$ bits. L'écriture de tous ces nombres pour $\mathcal{O}(T)$ évaluations se fait donc sur $\mathcal{O}(T(n \log D + H))$ bits. Or avec la substitution de Kronecker, pour évaluer F_u sur un point ω, il faut écrire $(\omega, \omega^D, \ldots, \omega^{D^{n-1}})$ et évaluer F sur ce n-uplet. L'écriture du n-uplet demande n fois plus de place que celle d'un seul point. L'évaluation d'une boîte noire pour un polynôme univarié n'a pas le même coût que l'évaluation de ce même polynôme univarié par une boîte noire d'un polynôme multivarié.

Fait 6.4. *Soit $F_u = K_D(F)$. La requête d'une évaluation de F_u par une boîte noire représentant F est n fois plus chère que la requête d'une évaluation de F_u par une boîte noire représentant directement F_u.*

Dans le cadre des circuits arithmétiques, ce surcoût est caché par le fait que la longueur du circuit L est nécessairement plus grande que n pour que chacune des variables intervienne dans la construction du circuit.

Ainsi, la substitution de Kronecker ne permet pas d'être aussi efficace qu'on pourrait l'espérer. Ceci est encore plus vrai dans le cas de l'interpolation de polynômes sur des corps finis $\mathbb{F}_{q^s}$. En effet, l'algorithme le plus efficace (corollaire 3.12, page 71) a besoin que la caractéristique q soit plus grande que le degré. Passer le degré à D^n réduit fortement son champ d'application. Pour des corps de plus petite caractéristique, l'algorithme le plus efficace (théorème 3.8, page 69) est cubique dans le logarithme du degré, ce qui donnerait donc un algorithme cubique en n. Pour pallier ces problèmes, les concepteurs de ces algorithmes ont travaillé pour mettre au point des substitutions de Kronecker aléatoire [HG19 ; AR14]. Dans les deux cas, l'idée est globalement d'envoyer un n-uplet d'exposants $(e_0, \dots, e_n)$ non pas sur $\sum_{i=0}^{n-1} e_i D^i$ mais sur $\sum_{i=0}^{n-1} e_i s_i$ pour des coefficients s_i aléatoires avec des valeurs en $\tilde{\mathcal{O}}(T \log(nD))$. Le degré du polynôme obtenu, en $\tilde{\mathcal{O}}(TnD)$, est bien plus petit que celui que donne la substitution de Kronecker, D^n. Cependant, cette substitution n'est pas inversible et peut envoyer deux n-uplets distincts vers le même nombre. Pour pallier ces limites, cette transformation est répétée avec différentes valeurs pour les s_i. En particulier, pour la substitution de Huang et Gao, un premier $p = \tilde{\mathcal{O}}(T \log(nD))$ est fixé et le vecteur de coefficients $s = (s_0 \dots, s_{n-1})$ avec $s_j < p$ est considéré en même temps que les n vecteurs $\sigma_i = (\sigma_{i,0}, \dots, \sigma_{i,n-1})$ qui vérifient $\sigma_{i,i} = s_i + p$ et $\sigma_{i,j} = s_j$. Ainsi, la différence entre la substitution donnée par s et celle donnée par un σ_i permet de retrouver les exposants de x_i. C'est cette approche qui permet d'obtenir les algorithmes les plus efficaces en se ramenant à l'interpolation univariée. La méthode d'Arnold et Roche passe par des calculs matriciels pour l'inversion de la substitution.

Fait 6.5 ([Hua19 ; HG20]). *Soit $F \in \mathbb{F}_{q^s}[x_0, \dots, x_{n-1}]$ un polynôme décrit par un circuit arithmétique de taille L, avec des bornes T, D sur son nombre de monômes et son degré en chaque variable. Il est possible de calculer F avec une probabilité d'échec constante en*

- $\mathcal{O}(LnT \log D \log q^s)$ *opérations binaires si* $q > \Omega(nDT \log(nD) \log(T \log(nD)))$.
- $\mathcal{O}(LnT \log^2 D(\log D + \log q^s))$ *opérations binaires sinon.*

Toutes les approches ne reposent pas sur la substitution de Kronecker qui a ses limites. Huang a proposé récemment un algorithme utilisant l'évaluation simultanée d'un polynôme F et de ses dérivées partielles à partir du circuit arithmétique décrivant F.

Fait 6.6 ([Hua21]). *Soit $F \in \mathbb{F}_{q^s}[x_0, \dots, x_{n-1}]$ un polynôme décrit par un circuit arithmétique de taille L, avec des bornes T, D sur son nombre de monômes et son degré en chaque variable. Si $q > D$, il est possible de calculer F avec une probabilité d'échec constante en $\tilde{\mathcal{O}}(LT \log q^s + nT \log q^s + T(\log q^s)^2)$ opérations binaires.*

Les complexités reposent sur cinq paramètres distincts, dont les valeurs déterminent quel algorithme est le plus efficace. Le dernier algorithme est le seul à ne pas avoir de facteur n^2 (caché dans le facteur Ln des complexités des algorithmes précédents) qui vient notamment de la répétition d'évaluations sur des n-uplets.

6.2. Arithmétique

Le cas des opérations arithmétiques est grandement différent. En effet, les polynômes d'entrée sont explicitement connus. Il est donc possible de les transformer effectivement en polynômes univariés par la substitution de Kronecker. L'arithmétique est alors effectuée sur des polynômes univariés dont tous les coefficients et exposants sont connus. En particulier, il est possible de les évaluer en un point ω directement, sans passer par l'évaluation des polynômes d'origine sur un n-uplet de puissances de ω, $(\omega, \omega^D, \ldots, \omega^{D^{n-1}})$. Ainsi, le coût d'une opération effectuée par le biais d'une substitution de Kronecker correspond au coût de l'opération entre polynômes univariés avec les paramètres obtenus par la transformation, à savoir le même nombre de monômes, les mêmes coefficients et le degré D^n, plus le coût de la substitution qui est quasi-linéaire (fait 6.1). En particulier, si l'opération sur un polynôme univarié est quasi-linéaire, cela conduit à une opération qui reste quasi-linéaire pour des polynômes à plusieurs variables. Les algorithmes quasi-linéaires décrits dans cette thèse s'étendent donc aux polynômes multivariés avec une complexité toujours quasi-linéaire.

Corollaire 6.7. *Soit F, G, Q, R des polynômes dans $\mathbb{Z}[x_0, \ldots, x_{n-1}]$ tels que Q divise F et $\rho \geq 0$.*

- *Avec une probabilité d'au moins $1 - \frac{1}{2^\rho}$ il est possible de calculer FG ou F/Q en $\tilde{\mathcal{O}}(T(n \log D + \log H)\rho + \rho^4)$ opérations binaires ;*
- *de plus il est possible de vérifier si $FG = R$ avec une probabilité d'erreur d'au plus $\frac{1}{2^\rho}$ en cas d'inégalité en $\tilde{\mathcal{O}}(T(n \log D + \log H)\rho + \rho^4)$ opérations binaires,*

avec D, T et H les maximum des degrés en chaque variable, du nombre de monômes et de la hauteur de chacun des polynômes, y compris les résultats FG et F/Q.

Démonstration. Il s'agit d'appliquer la procédure RéductionÀ1Variable à chacune des opérations. La transformation de Kronecker K_D et son inverse K_D^{-1} permettent de passer à des polynômes à une seule variable ou d'obtenir à nouveau des polynômes à n variables en $\tilde{\mathcal{O}}(Tn \log D)$ opérations binaires (fait 6.1). Le coût vient de celui des différentes opérations en degré D^n. Ils sont donnés dans le théorème 5.6 (page 130) pour le produit, le théorème 5.16 (page 141) pour la division exacte et le corollaire 4.44 (page 118) pour la vérification (la dépendance en ρ se déduisant du théorème 4.43, page 116). □

De manière similaire, la vérification d'un produit de polynômes univariés sur des corps finis est quasi-linéaire quelle que soit la taille du corps fini (corollaires 4.46 et 4.47, page 119) et peut donc s'étendre aux polynômes multivariés.

Corollaire 6.8. *Soit F, G, R des polynômes dans $\mathbb{F}_q[x_0, \ldots, x_{n-1}]$ de degré maximum en chaque variable inférieur à D et ayant au plus T monômes, $\rho \geq 0$. Il est possible de vérifier si $FG = R$ avec une probabilité d'erreur d'au plus $\frac{1}{2^\rho}$ en cas d'inégalité en $\tilde{\mathcal{O}}(T(n \log D + \log q)\rho + \rho^4)$ opérations binaires.*

Pour le produit et la division exacte de polynômes creux, c'est plus délicat car les algorithmes ne sont pas quasi-linéaires et dépendent du rapport entre la caractéristique et le degré.

Corollaire 6.9. *Soit $F, G, Q \in \mathbb{F}_{q^s}[x_0, \ldots, x_{n-1}]$ avec q premier, de degré maximum en chaque variable inférieur à D et ayant au plus T monômes chacun et tels que Q divise F. Avec une probabilité de succès constante, il est possible de calculer correctement FG ou F/Q en effectuant*

- $\tilde{\mathcal{O}}(Tn \log D \log q^s)$ *opérations binaires si* $q > D^n$,
- $\tilde{\mathcal{O}}(Tn^2 (\log D)^2 (n \log D + \log q^s))$ *opérations binaires sinon.*

Pour obtenir de meilleures complexités, il faudrait s'appuyer sur les algorithmes d'interpolation à base de substitution de Kronecker aléatoire dont la complexité est donnée dans le fait 6.5. Cela pose un problème avec la procédure 25 DivisionParInterpolation (voir section 5.3.2, page 144). Celle-ci permet de transformer un algorithme d'interpolation de circuit arithmétique en un algorithme de division exacte avec une borne sur le nombre de monômes du quotient. Et c'est cet algorithme qui est ensuite utilisé pour calculer le quotient en ne s'appuyant sur aucune borne préalable. Or cette procédure s'appuie sur le fait que les algorithmes d'interpolation calculent des évaluations en x modulo $x^p - 1$ mais ce n'est plus le cas après une transformation de Kronecker aléatoire.

Conclusion

Cette thèse traite de différents sujets liés aux polynômes creux : l'interpolation, l'arithmétique et des vérifications d'identités polynomiales. Avant sa réalisation, la meilleure complexité connue pour l'interpolation ne concernait que des polynômes dans des corps finis de grande caractéristique. Cette complexité n'est pas quasi-linéaire dans la taille du polynôme mais quasi-linéaire dans chaque paramètre de cette taille. Par ailleurs les opérations arithmétiques ne dépendaient pas du véritable nombre de monômes du résultat ou alors pas de manière linéaire. Les résultats présentés dans cette thèse apportent des améliorations sur ces deux points en fournissant les premiers algorithmes quasi-linéaires pour interpoler, multiplier ou diviser avec reste nul des polynômes à coefficients entiers. De plus alors que la classe de complexité du problème de la divisibilité reste inconnue, nous avons fourni un algorithme déterministe pour la tester en temps polynomial sur une famille non triviale de polynômes creux.

Il reste aussi des questions à résoudre. Si l'arithmétique des polynômes creux est moins efficace sur les corps finis, c'est que les algorithmes d'interpolation ne sont pas quasi-linéaires. Des algorithmes d'interpolation quasi-linéaires existent-t-ils ? Pour les corps de petite caractéristique, en particulier $\mathbb{F}_2$ et ses extensions, la question est déjà de savoir s'il existe des algorithmes d'interpolation qui ne soient pas cubiques dans le logarithme du degré ou s'il s'agit aussi d'une borne inférieure sur la complexité. Les techniques les plus efficaces à l'heure actuelle passent par des évaluations sur des points d'ordre $p = \tilde{\mathcal{O}}(T \log D)$ dans un anneau contenant plus de D points. Sur les entiers nous avons montré qu'il est possible de choisir un modulo q^k qui fournisse des points d'ordre p. Sur un corps fini $\mathbb{F}_q$ en revanche, le modulo est fixé. Pour avoir suffisamment de points il faut passer dans une extension d'ordre $s > \log_q D$ et un point d'ordre p sera donné par x dans $\mathbb{F}_{q^s}[x]/(x^p - 1)$. De telles techniques semblent mener à une complexité au moins en $\log^2(D)$. Cette complexité est déjà atteinte en grande caractéristique, est-elle aussi atteignable en petite caractéristique ?

Un point me semble plus important, la représentation creuse s'oppose à la représentation dense. Son objectif est d'être juste de la bonne taille pour le polynôme sans stocker d'information inutile. Cependant pour tous les algorithmes, nous avons considéré qu'un polynôme $F = \sum f_i x^{e_i}$ est de taille $\#F(\log(\deg(F)) + \log \|F\|)$. Ceci n'est qu'une borne mais pas la véritable taille. En effet, $\deg(F)$ et $\|F\|$ ne sont que des bornes sur le degré et la hauteur de chaque monôme de F. Pour être plus précis, on note qu'un monôme $f_i x^{e_i}$ contribue de $\log e_i + \log |f_i|$ bits à la taille totale qui est alors $\sum(\log e_i + \log |f_i|)$. Pour vraiment répondre au soucis d'optimisation auquel correspond la représentation creuse il serait judicieux d'avoir des algorithmes dont la complexité dépende du degré moyen ou

de la hauteur moyenne. De telles considérations permettent aussi de voir une limite de la substitution de Kronecker. Si elle conserve bien la borne sur la taille du polynôme, elle modifie sa véritable taille en envoyant le monôme x_{n-1} de taille n (représenté par le n-uplet $(0,\ldots,0,1)$) sur le monôme $x^{D^{n-1}}$ de taille $(n-1)\log D$. Il convient pour respecter au mieux la taille des polynômes de concevoir des algorithmes propres aux polynômes multivariés. Les méthodes de substitutions de Kronecker aléatoires visent justement à limiter une croissance excessive. L'adaptation de ces méthodes à nos algorithmes d'arithmétiques représenterait déjà une progression significative.

Enfin, cette thèse n'a traité que de polynômes creux par rapport à la base monomiale. Ce n'est qu'une étape dans le développement d'une algorithmique pour les polynômes structurés. La base monomiale est parfois la plus efficace pour obtenir une représentation compacte d'un polynôme creux ayant peu de monômes non nuls. Un polynôme ayant une structure différente sera plus efficacement représenté comme un polynôme creux dans une autre base (Tchebychev, Pochhammer, etc). L'interpolation creuse a d'ailleurs été largement étudiée dans différentes bases [LS95 ; GKL03 ; GLL04 ; PT14 ; AK15 ; IKY18] et même en recherchant la base de représentation la plus efficace [GR10]. Il serait intéressant d'étudier si les résultats obtenus ou les techniques employées dans cette thèse peuvent s'étendre à des polynômes creux par rapport à d'autres bases. Une autre question est l'extension des résultats à des polynômes creux non commutatifs qu'il s'agisse de polynômes tordus ou d'opérateurs différentiels linéaires. La multiplication de tels polynômes ayant fait l'objet d'études récentes [GHS20 ; GHS21].

A. Code en SageMath pour mesurer la proportion de nombres premiers dans des suites arithmétiques

```
def primes_in_arith_prog(p, bound, samples = 1000, random=True):
    """
    Estimate the number of primes <= bound in the arithmetic
    progression A = { 2pk+1 : k >= 1 }, by sampling samples
    random values k. The primality test is randomized if
    random = True.

    Note. If |A| <= bound, the number is computed exactly by
    iterating over all elements of A.
    """
    count = 0
    size = bound // (2*p)
    if size > samples:
        for _ in range(samples):
            k = ZZ(randint(1, size))
            if (2*k*p+1).is_prime(proof=random):
                count += 1
        return RR(count/samples*size)

    for k in range(size):
        if ZZ(2*k*p+1).is_prime(proof=random):
            count += 1
    return count

def distrib_primes(bitsize, bound, samples = 1000, random=True):
    """
    Estimate the proportion of primes <= bound in each arithmetic
    progression A = {2kp+1 : k >= 1} for primes p of the given
    bitsize.
    """
    L = []
```

```
    for p in prime_range(2**(bitsize-1),2**bitsize):
        n = primes_in_arith_prog(p,bound,samples,random)
        L.append(float(n/(bound//(2*p))))
    return L

def test(bmin, bmax):
    """
    This function generates the whole data set as a dictionary
    and prints the table that summarizes the result.
    """
    D = {}
    for b in range(bmin,bmax):
        L = distrib_primes(b,2**(2*b),samples=1000*(b-10+1))
        D[b] = L
    print("bitsize\tmin\taverage\tmax")
    for b in D:
        print(f"{b}\t{min(D[b]):.3f}\t{mean(D[b]):.3f}\t{max(D[b]):.3f}")
    return D

test(10, 21)
```

Bibliographie

[AGR13] Andrew Arnold, Mark Giesbrecht et Daniel S. Roche. "Faster Sparse Interpolation of Straight-Line Programs". In : *Computer Algebra in Scientific Computing*. T. 8136. Series Title : Lecture Notes in Computer Science. Springer International Publishing, 2013, p. 61-74. doi : 10.1007/978-3-319-02297-0_5 (cf. p. 24, 30, 69, 74).

[AGR14] Andrew Arnold, Mark Giesbrecht et Daniel S. Roche. "Sparse interpolation over finite fields via low-order roots of unity". In : *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation*. ISSAC'14. Association for Computing Machinery, 2014, p. 27-34. doi : 10.1145/2608628.2608671 (cf. p. 30, 69, 74, 132, 145).

[AGR15] Andrew Arnold, Mark Giesbrecht et Daniel S. Roche. "Faster sparse multivariate polynomial interpolation of straight-line programs". In : *Journal of Symbolic Computation* (2015). doi : 10.1016/j.jsc.2015.11.005 (cf. p. 4).

[AH15] Amir Akbary et Kyle Hambrook. "A variant of the Bombieri-Vinogradov theorem with explicit constants and applications". In : *Mathematics of Computation* 84.294 (2015), p. 1901-1932. doi : 10.1090/S0025-5718-2014-02919-0 (cf. p. 34).

[AK15] Andrew Arnold et Erich Kaltofen. "Error-Correcting Sparse Interpolation in the Chebyshev Basis". In : *Proceedings of the 2015 ACM on International Symposium on Symbolic and Algebraic Computation*. ISSAC '15. Bath, United Kingdom : Association for Computing Machinery, 2015, p. 21-28. doi : 10.1145/2755996.2756652 (cf. p. 155).

[AKP06] Martin Avendaño, Teresa Krick et Ariel Pacetti. "Newton–Hensel Interpolation Lifting". In : *Foundations of Computational Mathematics* 6.1 (2006), p. 82-120. doi : 10.1007/s10208-005-0172-3 (cf. p. 73).

[AKS04] Manindra Agrawal, Neeraj Kayal et Nitin Saxena. "PRIMES is in P". In : *Annals of Mathematics* 160.2 (2004), p. 781-793. doi : 10.4007/annals.2004.160.781 (cf. p. 16, 17, 33).

[AR14] Andrew Arnold et Daniel S. Roche. "Multivariate Sparse Interpolation Using Randomized Kronecker Substitutions". In : *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation*. ISSAC '14. Kobe, Japan : Association for Computing Machinery, 2014, p. 35-42. doi : 10.1145/2608628.2608674 (cf. p. 151).

[AR15] Andrew Arnold et Daniel S. Roche. "Output-Sensitive Algorithms for Sumset and Sparse Polynomial Multiplication". In : *Proceedings of the 2015*

ACM on International Symposium on Symbolic and Algebraic Computation. ISSAC '15. Bath, United Kingdom : ACM, 2015, p. 29-36. DOI : 10.1145/2755996.2756653 (cf. p. 3, 30, 70, 76, 123, 127, 128).

[Arn16] Andrew ARNOLD. "Sparse Polynomial Interpolation and Testing". Thèse de doct. University of Waterloo, 2016. URL : http://hdl.handle.net/10012/10307 (cf. p. 32, 69).

[AW21] Josh ALMAN et Virginia Vassilevska WILLIAMS. "A Refined Laser Method and Faster Matrix Multiplication". In : *Proceedings of the Thirty-Second Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA '21. USA : Society for Industrial et Applied Mathematics, 2021, p. 522-539. DOI : 10.5555/3458064.3458096 (cf. p. 105).

[Bal+21] Nikhil BALAJI, Sylvain PERIFEL, Mahsa SHIRMOHAMMADI et James WORRELL. "Cyclotomic Identity Testing and Applications". In : *Proceedings of the 2021 on International Symposium on Symbolic and Algebraic Computation*. ISSAC '21. Association for Computing Machinery, 2021, p. 35-42. DOI : 10.1145/3452143.3465530 (cf. p. 85).

[BFN21] Karl BRINGMANN, Nick FISCHER et Vasileios NAKOS. "Sparse nonnegative convolution is equivalent to dense nonnegative convolution". In : *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*. STOC 2021. New York, NY, USA : Association for Computing Machinery, juin 2021, p. 1711-1724. DOI : 10.1145/3406325.3451090 (cf. p. 128).

[BFN22] Karl BRINGMANN, Nick FISCHER et Vasileios NAKOS. "Deterministic and Las Vegas Algorithms for Sparse Nonnegative Convolution". In : *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. Proceedings. Society for Industrial et Applied Mathematics, jan. 2022, p. 3069-3090. DOI : 10.1137/1.9781611977073.119 (cf. p. 128).

[Bha+22] Vishwas BHARGAVA, Sumanta GHOSH, Zeyu GUO, Mrinal KUMAR et Chris UMANS. "Fast Multivariate Multipoint Evaluation Over All Finite Fields". In : *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*. 2022, p. 221-232. DOI : 10.1109/FOCS54457.2022.00028 (cf. p. 63).

[BJ14] Markus BLÄSER et Gorav JINDAL. "A new deterministic algorithm for sparse multivariate polynomial interpolation". In : *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation*. New York, NY, USA : Association for Computing Machinery, 2014. DOI : 10.1145/2608628.2608648 (cf. p. 3, 65).

[BLS03] Alin BOSTAN, Grégoire LECERF et Éric SCHOST. "Tellegen's Principle into Practice". In : *Proceedings of the 2003 International Symposium on Symbolic and Algebraic Computation*. ISSAC '03. Philadelphia, PA, USA : ACM, 2003, p. 37-44. DOI : 10.1145/860854.860870 (cf. p. 63).

[Blu70] Leo I. BLUESTEIN. "A Linear Filtering Approach to the Computation of Discrete Fourier Transform". In : *IEEE Transactions on Audio and Electroacoustics* 18.4 (1970), p. 451-455. DOI : 10.1109/TAU.1970.1162132 (cf. p. 63).

[BN20] Karl Bringmann et Vasileios Nakos. “Top-k-convolution and the quest for near-linear output-sensitive subset sum”. In : *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*. STOC 2020. New York, NY, USA : Association for Computing Machinery, juin 2020, p. 982-995. doi : 10.1145/3357713.3384308 (cf. p. 128).

[Bom65] Enrico Bombieri. “On the large sieve”. In : *Mathematika* 12 (1965), p. 201-225 (cf. p. 34).

[Bos+17] Alin Bostan et al. *Algorithmes Efficaces en Calcul Formel*. Palaiseau : Frédéric Chyzak (auto-édit.), sept. 2017. isbn : 979-10-699-0947-2. url : https://hal.archives-ouvertes.fr/AECF/ (cf. p. 2).

[BS83] Walter Baur et Volker Strassen. “The complexity of partial derivatives”. In : *Theoretical Computer Science* 22.3 (1983), p. 317-330. doi : 10.1016/0304-3975(83)90110-X (cf. p. 70).

[BT88] Michael Ben-Or et Prasoon Tiwari. “A Deterministic Algorithm for Sparse Multivariate Polynomial Interpolation”. In : *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*. STOC ’88. Chicago, Illinois, USA : Association for Computing Machinery, 1988, p. 301-309. doi : 10.1145/62212.62241 (cf. p. 3, 61, 64).

[BT90] Allan Borodin et Prasoon Tiwari. “On the Decidability of Sparse Univariate Polynomial Interpolation”. In : *Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing*. STOC ’90. Baltimore, Maryland, USA : Association for Computing Machinery, 1990, p. 535-545. doi : 10.1145/100216.100292 (cf. p. 59).

[BZ10] Richard P. Brent et Paul Zimmermann. *Modern Computer Arithmetic*. Cambridge Monographs on Applied and Computational Mathematics. Cambridge University Press, 2010. doi : 10.1017/CBO9780511921698 (cf. p. 149).

[Can88] John F. Canny. *Complexity of Robot Motion Planning*. Sous la dir. de MIT Press. 1988. isbn : 0262031361, 9780262031363 (cf. p. 1).

[CD10] Jacques Carette et James H. Davenport. “The Power of Vocabulary : The Case of Cyclotomic Polynomials”. 2010. url : https://arxiv.org/abs/1002.0012 (cf. p. 86).

[CH02] Richard Cole et Ramesh Hariharan. “Verifying candidate matches in sparse and wildcard matching”. In : *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*. ACM, 2002, p. 592-601. doi : 10.1145/509907.509992 (cf. p. 3).

[Cha+21] Arkadev Chattopadhyay, Bruno Grenet, Pascal Koiran, Natacha Portier et Yann Strozecki. “Computing the multilinear factors of lacunary polynomials without heights”. In : *Journal of Symbolic Computation* 104 (2021), p. 183-206. doi : 10.1016/j.jsc.2020.04.013 (cf. p. 4).

[CK91] David G. Cantor et Erich Kaltofen. “On fast multiplication of polynomials over arbitrary algebras”. In : *Acta Informatica* 28 (1991), p. 693-701. doi : 10.1007/BF01178683 (cf. p. 2, 14).

[CKP12] Matthew T. Comer, Erich Kaltofen et Clément Pernet. “Sparse Polynomial Interpolation and Berlekamp/Massey Algorithms That Correct Outlier

Errors in Input Values". In : *Proceedings of the 37th International Symposium on Symbolic and Algebraic Computation*. ISSAC '12. Association for Computing Machinery, 2012, p. 138-145. DOI : 10.1145/2442829.2442852 (cf. p. 60).

[CKY89] J. F. CANNY, E. KALTOFEN et L. YAGATI. "Solving Systems of Nonlinear Polynomial Equations Faster". In : *Proceedings of the ACM-SIGSAM 1989 International Symposium on Symbolic and Algebraic Computation*. ISSAC '89. Portland, Oregon, USA : Association for Computing Machinery, 1989, p. 121-128. DOI : 10.1145/74540.74556 (cf. p. 4).

[CL11] Annie CUYT et Wen-shin LEE. "Sparse interpolation of multivariate rational functions". In : *Theoretical Computer Science*. Symbolic and Numerical Algorithms 412.16 (2011), p. 1445-1456. DOI : 10.1016/j.tcs.2010.11.050 (cf. p. 4, 66).

[CL13] Jean-Marc COUVEIGNES et Reynald LERCIER. "Fast Construction of Irreducible Polynomials over Finite Fields". In : *Israel Journal of Mathematics* 194.1 (2013), p. 77-105 (cf. p. 18).

[Cor+09] Thomas H. CORMEN, Charles E. LEISERSON, Ronald L. RIVEST et Clifford STEIN. *Introduction to Algorithms*. 3rd. The MIT Press, 2009 (cf. p. 12, 53).

[CT65] James W. COOLEY et John W. TUKEY. "An Algorithm for the Machine Calculation of Complex Fourier Series". In : *Mathematics of Computation* 19 (1965), p. 297-301. DOI : 10.1090/S0025-5718-1965-0178586-1 (cf. p. 2).

[DC09] James Harold DAVENPORT et Jacques CARETTE. "The Sparsity Challenges". In : *11th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*. 2009, p. 3-7. DOI : 10.1109/SYNASC.2009.62 (cf. p. 4).

[Dic+08] Alicia DICKENSTEIN, Frank-Olaf SCHREYER, Andrew J. SOMMESE, Douglas N. ARNOLD et Arnd SCHEEL, éd. *Algorithms in Algebraic Geometry*. T. 146. The IMA Volumes in Mathematics and its Applications. New York, NY : Springer, 2008. DOI : 10.1007/978-0-387-75155-9 (cf. p. 1).

[Dir37] Gustav Lejeune DIRICHLET. "Beweis eines Satzes über die arithmetische Progression". In : *Bericht über die Verhandlungen der königlich preussischen Akademie der Wissenschaften* (1837). URL : https://gallica.bnf.fr/ark:/12148/bpt6k99435r/f320# (cf. p. 34).

[Dor87] Jean-Louis DORNSTETTER. "On the Equivalence Between Berlekamp's and Euclid's Algorithms". In : *IEEE Transactions on Information Theory* 33.3 (1987), p. 428-431. DOI : 10.1109/TIT.1987.1057299 (cf. p. 63).

[Dum+07] Jean-Guillaume DUMAS, Jean-Louis ROCH, Eric TANNIER et Sébastien VARRETTE. *Théorie des Codes : compression, cryptage, correction*. Dunod, jan. 2007 (cf. p. 1).

[Dum+19] Jean-Guillaume DUMAS, Joris van der HOEVEN, Clément PERNET et Daniel S. ROCHE. "LU Factorization with Errors". In : *Proceedings of the 2019 on International Symposium on Symbolic and Algebraic Computation, ISSAC*

2019, Beijing, China, July 15-18, 2019. ACM, 2019, p. 131-138. DOI : 10.1145/3326229.3326244 (cf. p. 60).

[Fat03] Richard FATEMAN. "Comparing the Speed of Programs for Sparse Polynomial Multiplication". In : *SIGSAM Bull.* 37.1 (mars 2003), p. 4-15. DOI : 10.1145/844076.844080 (cf. p. 12).

[Fil+20] Michael FILASETA, Huixi LI, Frank PATANE et Dane SKABELUND. "On the irreducibility of the non-reciprocal part of polynomials of the form $f(x)x^n + g(x)$". In : *Acta Arithmetica* 196 (2020). DOI : 10.4064/aa190907-11-2 (cf. p. 4).

[Fre79] Rusins FREIVALDS. "Fast probabilistic algorithms". In : *Mathematical Foundations of Computer Science*. T. 74. Springer Berlin Heidelberg, 1979, p. 57-69. DOI : 10.1007/3-540-09526-8_5 (cf. p. 104, 105).

[GG13] Joachim von zur GATHEN et Jürgen GERHARD. *Modern Computer Algebra (third edition)*. Cambridge University Press, 2013 (cf. p. 2, 14, 89).

[GGP20] Pascal GIORGI, Bruno GRENET et Armelle PERRET DU CRAY. "Essentially optimal sparse polynomial multiplication". In : *Proceedings of the 45th International Symposium on Symbolic and Algebraic Computation*. ISSAC'20. Kalamata, Greece, 2020, p. 202-209. DOI : 10.1145/3373207.3404026 (cf. p. 110, 115, 133).

[GGP21] Pascal GIORGI, Bruno GRENET et Armelle PERRET DU CRAY. "On exact division and divisibility testing for sparse polynomials". In : *Proceedings of the 2021 on International Symposium on Symbolic and Algebraic Computation*. ISSAC'21. 2021, p. 163-170. DOI : 10.1145/3452143.3465539 (cf. p. 87).

[GGP23] Pascal GIORGI, Bruno GRENET et Armelle PERRET DU CRAY. "Polynomial modular product verification and its implications". In : *Journal of Symbolic Computation* 116 (2023), p. 98-129. DOI : https://doi.org/10.1016/j.jsc.2022.08.011 (cf. p. 96).

[GHS20] Mark GIESBRECHT, Qiao-Long HUANG et Éric SCHOST. "Sparse Multiplication for Skew Polynomials". In : ISSAC '20. Kalamata, Greece : Association for Computing Machinery, 2020, p. 194-201. DOI : 10.1145/3373207.3404023 (cf. p. 155).

[GHS21] Mark GIESBRECHT, Qiao-Long HUANG et Éric SCHOST. "Sparse Multiplication of Multivariate Linear Differential Operators". In : *Proceedings of the 2021 on International Symposium on Symbolic and Algebraic Computation*. ISSAC '21. Virtual Event, Russian Federation : Association for Computing Machinery, 2021, p. 155-162. DOI : 10.1145/3452143.3465527 (cf. p. 155).

[Gio+22] Pascal GIORGI, Bruno GRENET, Armelle PERRET DU CRAY et Daniel S. ROCHE. "Sparse Polynomial Interpolation and Division in Soft-Linear Time". In : *Proceedings of the 2022 International Symposium on Symbolic and Algebraic Computation*. ISSAC '22. Association for Computing Machinery, 2022, p. 459-468. DOI : 10.1145/3476446.3536173 (cf. p. 74).

[Gio18] Pascal GIORGI. "A probabilistic algorithm for verifying polynomial middle product in linear time". In : *Information Processing Letters* 139 (2018), p. 30-34. DOI : 10.1016/j.ipl.2018.06.014 (cf. p. 41, 43).

[GKL03] Mark GIESBRECHT, Erich KALTOFEN et Wen-shin LEE. "Algorithms for Computing Sparsest Shifts of Polynomials in Power, Chebyshev, and Pochhammer Bases". In : *Journal of Symbolic Computation* 36.3–4 (sept. 2003), p. 401-424. DOI : 10.1016/S0747-7171(03)00087-7 (cf. p. 155).

[GKO92] Dima GRIGORIEV, Marek KARPINSKI et Andrew M. ODLYZKO. "Existence of Short Proofs for Nondivisibility of Sparse Polynomials under the Extended Riemann Hypothesis". In : *Papers from the International Symposium on Symbolic and Algebraic Computation*. ISSAC '92. Berkeley, California, USA : Association for Computing Machinery, 1992, p. 117-122. DOI : 10.1145/143242.143287 (cf. p. 87).

[GKS90] Dima GRIGORIEV, Marek KARPINSKI et Michael F. SINGER. "Fast parallel algorithms for sparse multivariate polynomial interpolation over finite fields". In : *SIAM Journal on Computing* 19.6 (1990), p. 1059-1063. DOI : 10.1137/0219073 (cf. p. 64).

[GKS94] Dima GRIGORIEV, Marek KARPINSKI et Michael F. SINGER. "Computational Complexity of Sparse Rational Interpolation". In : *SIAM Journal on Computing* 23.1 (1994), p. 1-11. DOI : 10.1137/S0097539791194069 (cf. p. 4).

[GL80] David GRIES et Gary LEVIN. "Computing Fibonacci Numbers (and Similarly Defined Functions) in Log Time". In : *Information Processing Letters* 11 (1980), p. 68-69 (cf. p. 108).

[GLL04] Mark GIESBRECHT, George LABAHN et Wen-shin LEE. "Symbolic-Numeric Sparse Polynomial Interpolation in Chebyshev Basis and Trigonometric Interpolation". In : *Workshop on Computer Algebra in Scientific Computation (CASC)*. 2004, p. 195-205 (cf. p. 155).

[Gol82] Solomon W. GOLOMB. *Shift register sequences*. Aegean Park Press, 1982 (cf. p. 96).

[GP90] Andrew GRANVILLE et Carl POMERANCE. "On the Least Prime in Certain Arithmetic Progressions". In : *Journal of the London Mathematical Society* s2-41.2 (1990), p. 193-200. DOI : 10.1112/jlms/s2-41.2.193 (cf. p. 38).

[GR10] Mark GIESBRECHT et Daniel S. ROCHE. "Interpolation of Shifted-Lacunary Polynomials". In : *computational complexity* 19 (2010), p. 333-354 (cf. p. 65, 155).

[GR11a] Mark GIESBRECHT et Daniel S. ROCHE. "Detecting Lacunary Perfect Powers and Computing Their Roots". In : *Journal of Symbolic Computation* 46.11 (nov. 2011), p. 1242-1259. DOI : 10.1016/j.jsc.2011.08.006 (cf. p. 4).

[GR11b] Mark GIESBRECHT et Daniel S. ROCHE. "Diversification improves interpolation". In : *Proceedings of the 36th international symposium on Symbolic and algebraic computation - ISSAC '11*. Association for Computing Machinery, 2011, p. 123. DOI : 10.1145/1993886.1993909 (cf. p. 24, 64, 65, 69, 142).

[Gre16] Bruno GRENET. "Bounded-Degree Factors of Lacunary Multivariate Polynomials". In : *Journal of Symbolic Computation* 75.C (juill. 2016), p. 171-192. DOI : 10.1016/j.jsc.2015.11.013 (cf. p. 4).

[GRS22] Venkatesan GURUSWAMI, Atri RUDRA et Madhu SUDAN. *Essential Coding Theory*. 2022. URL : https://cse.buffalo.edu/faculty/atri/courses/coding-theory/book/ (cf. p. 1).

[GS09] Sanchit GARG et Éric SCHOST. "Interpolation of polynomials given by straight-line programs". In : *Theoretical Computer Science* 410.27-29 (2009), p. 2659-2662. DOI : 10.1016/j.tcs.2009.03.030 (cf. p. 3, 24, 61, 67, 68).

[Har77] Robin HARTSHORNE. *Algebraic Geometry*. T. 52. Graduate Texts in Mathematics. New York, NY : Springer, 1977. DOI : 10.1007/978-1-4757-3849-0 (cf. p. 1).

[Hea78] Roger HEATH-BROWN. "Almost-primes in arithmetic progressions and short intervals". In : *Mathematical Proceedings of the Cambridge Philosophical Society* 83.03 (1978), p. 357-375. DOI : 10.1017/S0305004100054657 (cf. p. 38).

[HG19] Qiao-Long HUANG et Xiao-Shan GAO. "Revisit Sparse Polynomial Interpolation Based on Randomized Kronecker Substitution". In : *Computer Algebra in Scientific Computing*. Springer International Publishing, 2019, p. 215-235. DOI : 10.1007/978-3-030-26831-2_15 (cf. p. 64, 151).

[HG20] Qiao-Long HUANG et Xiao-Shan GAO. "Faster interpolation algorithms for sparse multivariate polynomials given by straight-line programs". In : *Journal of Symbolic Computation* 101 (2020), p. 367-386. DOI : 10.1016/j.jsc.2019.10.005 (cf. p. 24, 30, 69, 74, 151).

[HH19] David HARVEY et Joris van der HOEVEN. "Faster Polynomial Multiplication over Finite Fields Using Cyclotomic Coefficient Rings". In : *Journal of Complexity* 54.C (oct. 2019). DOI : 10.1016/j.jco.2019.03.004 (cf. p. 14, 98).

[HH21] David HARVEY et Joris van der HOEVEN. "Integer multiplication in time $\mathcal{O}(n \log n)$". In : *Annals of Mathematics* 193.2 (2021), p. 563-617. DOI : 10.4007/annals.2021.193.2.4 (cf. p. 14).

[HH22] David HARVEY et Joris van der HOEVEN. "Polynomial Multiplication over Finite Fields in Time $O(n \log n$". In : *Journal of the ACM* 69.2 (mars 2022). DOI : 10.1145/3505584 (cf. p. 2, 14, 98, 103, 112, 119).

[HHN11] William HART, Mark van HOEIJ et Andrew NOVOCIN. "Practical polynomial factoring in polynomial time". In : *Proceedings of the 36th international symposium on Symbolic and algebraic computation - ISSAC '11*. Association for Computing Machinery, 2011, p. 163-170. DOI : 10.1145/1993886.1993914 (cf. p. 63).

[HL12] Joris van der HOEVEN et Grégoire LECERF. "On the Complexity of Multivariate Blockwise Polynomial Multiplication". In : *Proceedings of the 37th International Symposium on Symbolic and Algebraic Computation*. ISSAC '12. Association for Computing Machinery, 2012, p. 211-218. DOI : 10.1145/2442829.2442861 (cf. p. 148).

[HL13] Joris van der Hoeven et Grégoire Lecerf. "On the bit-complexity of sparse polynomial and series multiplication". In : *Journal of Symbolic Computation* 50 (2013), p. 227-0254. doi : 10.1016/j.jsc.2012.06.004 (cf. p. 124, 127).

[HL15] Joris van der Hoeven et Grégoire Lecerf. "Sparse Polynomial Interpolation in Practice". In : *ACM Communications in Computer Algebra* 48.3/4 (2015), p. 187-191. doi : 10.1145/2733693.2733721 (cf. p. 3, 64, 76).

[HL19] Joris van der Hoeven et Grégoire Lecerf. "Sparse polynomial interpolation. Exploring fast heuristic algorithms over finite fields". technical report. 2019. url : https://hal.archives-ouvertes.fr/hal-02382117 (cf. p. 73).

[HL21] Joris van der Hoeven et Grégoire Lecerf. "On sparse interpolation of rational functions and gcds". In : *ACM Communications in Computer Algebra* 55.1 (2021), p. 1-12. doi : 10.1145/3466895.3466896 (cf. p. 4, 60, 66).

[HM16] Jiaxiong Hu et Michael Monagan. "A Fast Parallel Sparse Polynomial GCD Algorithm". In : *Proceedings of the ACM on International Symposium on Symbolic and Algebraic Computation*. ISSAC '16. Waterloo, ON, Canada : Association for Computing Machinery, 2016, p. 271-278. doi : 10.1145/2930889.2930903 (cf. p. 4, 60).

[Hoe20] Joris van der Hoeven. "Probably faster multiplication of sparse polynomials". technical report. 2020. url : https://hal.archives-ouvertes.fr/hal-02473830 (cf. p. 128).

[HR99] Ming-Deh A. Huang et Ashwin J. Rao. "Interpolation of Sparse Multivariate Polynomials over Large Finite Fields with Applications". In : *Journal of Algorithms* 33.2 (1999), p. 204-228. doi : 10.1006/jagm.1999.1045 (cf. p. 64).

[Hua19] Qiao-Long Huang. "Sparse Polynomial Interpolation over Fields with Large or Zero Characteristic". In : *Proceedings of the 2019 on International Symposium on Symbolic and Algebraic Computation*. ISSAC '19. Beijing, China : ACM Press, 2019, p. 219-226. doi : 10.1145/3326229.3326250 (cf. p. 4, 30, 61, 70, 71, 74, 76, 132, 151).

[Hua21] Qiao-Long Huang. "Sparse polynomial interpolation based on diversification". In : *Science China Mathematics* (2021). doi : 10.1007/s11425-020-1791-5 (cf. p. 3, 64, 72, 74, 151).

[IKY18] Erdal Imamoglu, Erich Kaltofen et Zhengfeng Yang. "Sparse Polynomial Interpolation With Arbitrary Orthogonal Polynomial Bases". In : *Proceedings of the 2018 ACM International Symposium on Symbolic and Algebraic Computation*. ISSAC '18. New York, NY, USA : Association for Computing Machinery, 2018, p. 223-230. doi : 10.1145/3208976.3208999 (cf. p. 155).

[JLM19] Cédric Josz, Jean Bernard Lasserre et Bernard Mourrain. "Sparse Polynomial Interpolation : Sparse Recovery, Super-Resolution, or Prony ?" In : *Advances in Computational Mathematics* 45.3 (juin 2019), p. 1401-1437. doi : 10.1007/s10444-019-09672-2 (cf. p. 66).

[JM10] Seyed Mohammad Mahdi Javadi et Michael Monagan. “Parallel sparse polynomial interpolation over finite fields”. In : *Proceedings of the 4th International Workshop on Parallel and Symbolic Computation*. PASCO ’10. New York, NY, USA : Association for Computing Machinery, 2010, p. 160-168. doi : 10.1145/1837210.1837233 (cf. p. 64).

[Joh74] Stephen C. Johnson. “Sparse polynomial arithmetic”. In : *SIGSAM Bulletin* 8.3 (1974), p. 63-71. doi : 10.1145/1086837.1086847 (cf. p. 3, 4, 87, 124).

[Kal10] Erich Kaltofen. “Fifteen years after DSC and WLSS2 : What parallel computations I do today [invited lecture at PASCO 2010]”. In : *Proceedings of the 4th International Workshop on Parallel and Symbolic Computation*. PASCO ’10. Grenoble, France : ACM, 2010, p. 10-17. doi : 10.1145/1837210.1837213 (cf. p. 3, 65).

[Kal22] Erich Kaltofen. “Sparse Polynomial Hermite Interpolation”. In : *Proceedings of the 2022 International Symposium on Symbolic and Algebraic Computation*. ISSAC ’22. Association for Computing Machinery, 2022, p. 469-478. doi : 10.1145/3476446.3535501 (cf. p. 73).

[Kal88] Erich Kaltofen. “Greatest common divisors of polynomials given by straight-line programs”. In : *Journal of the ACM* 35.1 (jan. 1988), p. 231-264. doi : 10.1145/42267.45069 (cf. p. 60).

[Kam89] Michael Kaminski. “A Note on Probabilistically Verifying Integer and Polynomial Products”. In : *Journal of the ACM* 36.1 (1989), p. 142-149. doi : 10.1145/58562.214082 (cf. p. 27, 109, 110, 112, 115).

[KL03] Erich Kaltofen et Wen-shin Lee. “Early termination in sparse interpolation algorithms”. In : *Journal of Symbolic Computation* 36.3-4 (2003), p. 365-400. doi : 10.1016/S0747-7171(03)00088-9 (cf. p. 64).

[KL07] J. Katz et Y. Lindell. *Introduction to Modern Cryptography : Principles and Protocols*. Chapman et Hall, 2007. doi : 10.1201/9781420010756 (cf. p. 1).

[KL88] Erich Kaltofen et Yagati Lakshman. “Improved Sparse Multivariate Polynomial Interpolation Algorithms”. In : *Symbolic and Algebraic Computation*. Springer Berlin Heidelberg, 1988, p. 467-474. doi : 10.1007/3-540-51084-2_44 (cf. p. 3, 63, 64).

[KLW90] Erich Kaltofen, Yagati Lakshman et John-Michael Wiley. “Modular rational sparse multivariate polynomial interpolation”. In : *Proceedings of the international symposium on Symbolic and algebraic computation*. ISSAC ’90. Tokyo, Japan : ACM Press, 1990, p. 135-139. doi : 10.1145/96877.96912 (cf. p. 3, 64).

[KN11] Erich Kaltofen et Michael Nehring. “Supersparse black box rational function interpolation”. In : *Proceedings of the 36th international symposium on Symbolic and algebraic computation - ISSAC ’11*. Association for Computing Machinery, 2011, p. 177-186. doi : 10.1145/1993886.1993916 (cf. p. 3, 65, 66).

[KO62] Anatolii Karatsuba et Yu Ofman. “Multiplication of Multidigit Numbers on Automata”. In : *Soviet Physics Doklady* 7 (déc. 1962), p. 595 (cf. p. 2).

[KP14] Erich Kaltofen et Clément Pernet. “Sparse Polynomial Interpolation Codes and Their Decoding beyond Half the Minimum Distance”. In : *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation*. ISSAC ’14. Association for Computing Machinery, 2014, p. 272-279. doi : 10.1145/2608628.2608660 (cf. p. 4, 60).

[KT90] Erich Kaltofen et Barry M. Trager. “Computing with Polynomials Given By Black Boxes for Their Evaluations : Greatest Common Divisors, Factorization, Separation of Numerators and Denominators”. In : *Journal of Symbolic Computation* 9.3 (1990), p. 301-320. doi : 10.1016/S0747-7171(08)80015-6 (cf. p. 60).

[KU11] Kiran S. Kedlaya et Christopher Umans. “Fast Polynomial Factorization and Modular Composition”. In : *SIAM Journal on Computing* 40.6 (2011), p. 1767-1802. doi : 10.1137/08073408X (cf. p. 63).

[KY07] Erich Kaltofen et Zhengfeng Yang. “On exact and approximate interpolation of sparse rational functions”. In : *Proceedings of the 2007 international symposium on Symbolic and algebraic computation*. ISSAC ’07. Waterloo, Ontario, Canada : ACM Press, 2007, p. 203. doi : 10.1145/1277548.1277577 (cf. p. 4, 66).

[KY21] Erich Kaltofen et Zhi-Hong Yang. “Sparse Interpolation With Errors in Chebyshev Basis Beyond Redundant-Block Decoding”. In : *IEEE Transactions on Information Theory* 67.1 (jan. 2021), p. 232-243. doi : 10.1109/TIT.2020.3027036 (cf. p. 60).

[Len99a] Hendrik W. Lenstra Jr. “Finding small degree factors of lacunary polynomials”. In : *Number theory in progress*. De Gruyter, 1999, p. 267-276 (cf. p. 4).

[Len99b] Hendrik W. Lenstra Jr. “On the factorization of lacunary polynomials”. In : *Number theory in progress*. De Gruyter, 1999, p. 277-291 (cf. p. 4).

[LPS17] Junxian Li, Kyle Pratt et George Shakan. “A lower bound for the least prime in an Arithmetic progression”. In : *The Quarterly Journal of Mathematics* 68.3 (sept. 2017), p. 729-758. doi : 10.1093/qmath/hax001 (cf. p. 38).

[LS95] Yagati Lakshman et B. David Saunders. “Sparse Polynomial Interpolation in Nonstandard Bases”. In : *SIAM Journal on Computing* 24.2 (1995), p. 387-397. doi : 10.1137/S0097539792237784 (cf. p. 155).

[Man95] Yishay Mansour. “Randomized Interpolation and Approximation of Sparse Polynomials”. In : *SIAM Journal on Computing* 24.2 (1995), p. 357-368. doi : 10.1137/S0097539792239291 (cf. p. 65).

[Mas69] James L. Massey. “Shift-register synthesis and BCH decoding”. In : *IEEE Transactions on Information Theory* 15.1 (1969), p. 122-127. doi : 10.1109/TIT.1969.1054260 (cf. p. 63).

[MF96] Hirokazu Murao et Tetsuro Fujise. “Modular Algorithm for Sparse Multivariate Polynomial Interpolationand its Parallel Implementation”. In : *Journal of Symbolic Computation* 21.4-6 (1996), p. 377-396. doi : 10.1006/jsco.1996.0020 (cf. p. 64).

[MP07] Michael Monagan et Roman Pearce. “Polynomial Division Using Dynamic Arrays, Heaps, and Packed Exponent Vectors”. In : *Computer Algebra in Scientific Computing*. CASC ’07. 2007, p. 295-315. DOI : 10.1007/978-3-540-75187-8_23 (cf. p. 124).

[MP09a] Michael Monagan et Roman Pearce. “Parallel Sparse Polynomial Multiplication Using Heaps”. In : *Proceedings of the 2009 International Symposium on Symbolic and Algebraic Computation*. ISSAC ’09. Seoul, Republic of Korea : Association for Computing Machinery, 2009, p. 263-270. DOI : 10.1145/1576702.1576739 (cf. p. 3).

[MP09b] Michael Monagan et Roman Pearce. “Parallel sparse polynomial multiplication using heaps”. In : *Proceedings of the 2009 International Symposium on Symbolic and Algebraic Computation*. ISSAC’09. 2009, p. 263-270. DOI : 10.1145/1576702.1576739 (cf. p. 124).

[MP11a] Michael Monagan et Roman Pearce. “Sparse polynomial division using a heap”. In : *Journal of Symbolic Computation*. Special Issue in Honour of Keith Geddes on his 60th Birthday 46.7 (2011). DOI : 10.1016/j.jsc.2010.08.014 (cf. p. 4, 124-126).

[MP11b] Michael Monagan et Roman Pearce. “Sparse Polynomial Multiplication and Division in Maple 14”. In : *ACM Communications Computer Algebra* 44.3/4 (jan. 2011), p. 205-209. DOI : 10.1145/1940475.1940521 (cf. p. 3).

[MP13] Gary L. Mullen et Daniel Panario. *Handbook of Finite Fields*. 1st. Chapman & Hall/CRC, 2013 (cf. p. 1, 20, 96, 100).

[MT16] Michael Monagan et Baris Tuncer. “Using Sparse Interpolation in Hensel Lifting”. In : *Computer Algebra in Scientific Computing*. Springer International Publishing, 2016, p. 381-400. DOI : 10.1007/978-3-319-45641-6_25 (cf. p. 60).

[Nak20] Vasileios Nakos. “Nearly Optimal Sparse Polynomial Multiplication”. In : *IEEE Transactions on Information Theory* 66.11 (2020), p. 7231-7236. DOI : 10.1109/TIT.2020.2989385 (cf. p. 128).

[Nus80] Henri Nussbaumer. “Fast polynomial transform algorithms for digital convolution”. In : *IEEE Transactions on Acoustics, Speech, and Signal Processing* 28.2 (1980), p. 205-215. DOI : 10.1109/TASSP.1980.1163372 (cf. p. 2).

[Omo20] Amos Omondi. *Cryptography Arithmetic : Algorithms and Hardware Architectures*. Springer, jan. 2020. ISBN : 978-3-030-34141-1. DOI : 10.1007/978-3-030-34142-8 (cf. p. 1).

[Pai99] Pascal Paillier. “Public-Key Cryptosystems Based on Composite Degree Residuosity Classes”. In : *Advances in Cryptology – EUROCRYPT ’99*. Lecture Notes in Computer Science. Berlin, Heidelberg : Springer, 1999, p. 223-238. DOI : 10.1007/3-540-48910-X_16 (cf. p. 76).

[Pan01] Victor Y. Pan. *Structured Matrices and Polynomials*. Boston, MA : Birkhäuser Boston, 2001. DOI : 10.1007/978-1-4612-0129-8 (cf. p. 2).

[Per08] Daniel Perrin. *Algebraic Geometry*. London : Springer, 2008. DOI : 10.1007/978-1-84800-056-8 (cf. p. 1).

[Pla77] David Alan Plaisted. “Sparse complex polynomials and polynomial reducibility”. In : *Journal of Computer and System Sciences* 14.2 (1977), p. 210-221. doi : 10.1016/S0022-0000(77)80013-5 (cf. p. 86, 87).

[Pla84] David A. Plaisted. “New NP-hard and NP-complete polynomial and integer divisibility problems”. In : *Theoretical Computer Science* 31.1 (jan. 1984), p. 125-138. doi : 10.1016/0304-3975(84)90130-0 (cf. p. 4, 86, 87).

[Pom80] Carl Pomerance. “A note on the least prime in an arithmetic progression”. In : *Journal of Number Theory* 12.2 (1980), p. 218-223. doi : 10.1016/0022-314X(80)90056-6 (cf. p. 38).

[Pri82] Paul Pritchard. “Explaining the Wheel Sieve”. In : *Acta Informatica* 17.4 (1982), p. 477-485. doi : 10.1007/BF00264164 (cf. p. 16).

[Pro95] Gaspar de Prony. “Essai expérimental et analytique sur les lois de la Dilatabilité de fluides élastique et sur celles de la Force expansive de la vapeur de l’eau et de la vapeur de l’alkool, à différentes températures”. In : *Journal de l’École Polytechnique* 1.Floréal et Prairial, an IV (1795), p. 24-76. url : https://gallica.bnf.fr/ark:/12148/bpt6k433661n/f32.item (cf. p. 1, 3, 61).

[PT14] Daniel Potts et Manfred Tasche. “Sparse polynomial interpolation in Chebyshev bases”. In : *Linear Algebra and its Applications* 441 (2014). Special Issue on Sparse Approximate Solution of Linear Systems, p. 61-87. doi : 10.1016/j.laa.2013.02.006 (cf. p. 155).

[Rab80] Michael O. Rabin. “Probabilistic algorithm for testing primality”. In : *Journal of Number Theory* 12.1 (1980), p. 128-138. doi : 10.1016/0022-314X(80)90084-0 (cf. p. 16).

[RK89] Richard Roy et Thomas Kailath. “ESPRIT-estimation of signal parameters via rotational invariance techniques”. In : *IEEE Transactions on Acoustics, Speech, and Signal Processing* 37.7 (1989), p. 984-995. doi : 10.1109/29.32276 (cf. p. 66).

[Roc11] Daniel S. Roche. “Chunky and equal-spaced polynomial multiplication”. In : *Journal of Symbolic Computation* 46.7 (2011). Special Issue in Honour of Keith Geddes on his 60th Birthday, p. 791-806. doi : 10.1016/j.jsc.2010.08.013 (cf. p. 3).

[Roc18a] Daniel S. Roche. “Error Correction in Fast Matrix Multiplication and Inverse”. In : *Proceedings of the 2018 ACM on International Symposium on Symbolic and Algebraic Computation, ISSAC 2018, New York, NY, USA, July 16-19, 2018*. ACM, 2018, p. 343-350. doi : 10.1145/3208976.3209001 (cf. p. 60).

[Roc18b] Daniel S. Roche. “What Can (and Can’t) we Do with Sparse Polynomials ?” In : *Proceedings of the 2018 ACM International Symposium on Symbolic and Algebraic Computation*. ISSAC’18. ACM, 2018, p. 25-30. doi : 10.1145/3208976.3209027 (cf. p. 4, 74).

[Rot06] Ron Roth. *Introduction to Coding Theory*. Cambridge University Press, 2006. doi : 10.1017/CBO9780511808968 (cf. p. 1).

[RS62] John Barkley Rosser et Lowell Schoenfeld. “Approximate formulas for some Functions of Prime Numbers”. In : *Illinois Journal of Mathematics* 6 (1962), p. 64-94. DOI : 10.1215/ijm/1255631807 (cf. p. 16, 143).

[Sax09] Nitin Saxena. “Progress on Polynomial Identity Testing”. In : *Electron. Colloquium Comput. Complex.* 16 (2009), p. 101. URL : https://eccc.weizmann.ac.il/report/2009/101/ (cf. p. 85).

[Sax14] Nitin Saxena. “Progress on Polynomial Identity Testing-II”. In : *Perspectives in Computational Complexity : The Somenath Biswas Anniversary Volume*. Springer International Publishing, 2014, p. 131-146. DOI : 10.1007/978-3-319-05446-9_7 (cf. p. 85).

[Sch65] Andrzej Schinzel. “On the reducibility of polynomials and in particular of trinomials”. eng. In : *Acta Arithmetica* 11.1 (1965), p. 1-34. DOI : 10.4064/aa-11-1-1-34 (cf. p. 4).

[Sch69] Andrzej Schinzel. “Reducibility of lacunary polynomials I”. eng. In : *Acta Arithmetica* 16.2 (1969), p. 123-160. DOI : 10.4064/aa-16-2-123-160 (cf. p. 4).

[Sch71] Arnold Schönhage. “Schnelle Berechnung von Kettenbruchentwicklungen”. In : *Acta Informatica* 1 (juin 1971), p. 139-144. DOI : 10.1007/BF00289520 (cf. p. 63).

[Sed18] Alisa Sedunova. “A partial Bombieri–Vinogradov theorem with explicit constants”. In : *Publications mathématiques de Besançon. Algèbre et théorie des nombres* (2018), p. 101-110. DOI : 10.5802/pmb.24 (cf. p. 34, 37).

[Sho08] Victor Shoup. *A Computational Introduction to Number Theory and Algebra*. Second. Cambridge University Press, 2008 (cf. p. 17).

[Sho93] Victor Shoup. “Fast Construction of Irreducible Polynomials over Finite Fields”. In : *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA ’93. Society for Industrial et Applied Mathematics, 1993, p. 484-492. DOI : 10.5555/313559.313865 (cf. p. 18).

[Sho94] Victor Shoup. “Fast Construction of Irreducible Polynomials over Finite Fields”. In : *Journal of Symbolic Computation* 17.5 (1994), p. 371-391. ISSN : 0747-7171. DOI : 10.1006/jsco.1994.1025 (cf. p. 18).

[SK92] A. Lee Swindlehurst et Thomas Kailath. “A performance analysis of subspace-based methods in the presence of model errors. I. The MUSIC algorithm”. In : *IEEE Transactions on Signal Processing* 40.7 (1992), p. 1758-1774. DOI : 10.1109/78.143447 (cf. p. 66).

[SS71] Arnold Schönhage et Volker Strassen. “Schnelle Multiplikation großer Zahlen”. In : *Computing* 7.3 (sept. 1971), p. 281-292. DOI : 10.1007/BF02242355 (cf. p. 2).

[Vie91] Francisci Vietae. *In artem analyticem isagoge*. 1591. URL : https://gallica.bnf.fr/ark:/12148/bpt6k108865t (cf. p. 1).

[Wag79] Samuel S. Wagstaff. “Greatest of the least primes in arithmetic progressions having a given modulus”. In : *Mathematics of Computation* 33 (1979), p. 1073-1080. DOI : 10.1090/S0025-5718-1979-0528061-7 (cf. p. 38).

[Yan98] Thomas YAN. "The Geobucket Data Structure for Polynomials". In : *Journal of Symbolic Computation* 25.3 (1998), p. 285-293. DOI : 10.1006/jsco.1997.0176 (cf. p. 148).

[Yao76] Aandrew Chi-Chih YAO. "On the Evaluation of Powers". In : *SIAM Journal on Computing* 5.1 (1976), p. 100-103. DOI : 10.1137/0205008 (cf. p. 15, 45).

[Zip79] Richard ZIPPEL. "Probabilistic Algorithms for Sparse Polynomials". In : *Symbolic and Algebraic Computation. In : Lecture Notes in Comput. Sci., vol. 72*. Springer-Verlag, 1979, p. 216-226. DOI : 10.1007/3-540-09519-5_73 (cf. p. 60).

[Zip81] Richard ZIPPEL. "Newton's Iteration and the Sparse Hensel Algorithm (Extended Abstract)". In : *Proceedings of the Fourth ACM Symposium on Symbolic and Algebraic Computation*. SYMSAC '81. Snowbird, Utah, USA : Association for Computing Machinery, 1981, p. 68-72. ISBN : 0897910478. DOI : 10.1145/800206.806372 (cf. p. 60).

[Zip90] Richard ZIPPEL. "Interpolating polynomials from their values". In : *Journal of Symbolic Computation*. Computational algebraic complexity editorial 9.3 (1990), p. 375-403. DOI : 10.1016/S0747-7171(08)80018-1 (cf. p. 64, 65).