Multiplication algorithms : bilinear complexity and fast asymptotic methods

Svyatoslav Covanov

UNIVERSITÉ DE LORRAINE

École doctorale IAEM Lorraine

Algorithmes de multiplication : complexité bilinéaire et méthodes asymptotiquement rapides

THÈSE

présentée et soutenue publiquement le 5 juin 2018

pour l'obtention du

Doctorat de l'Université de Lorraine

(mention informatique)

par

Svyatoslav Covanov

Composition du jury

Rapporteurs :	Daniel Augot	Directeur de recherche, Inria Saclay
	Clément Pernet	Maître de Conférences (HDR), Université Grenoble-Alpes
Encadrants :	Emmanuel Thomé	Directeur de recherche, Inria Nancy
	Jérémie Detrey	Chargé de recherche, Inria Nancy
Examinatrices :	Sylvie Boldo	Directrice de recherche, Inria Saclay
	Natacha Portier	Maître de Conférences, ÉNS Lyon

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Mis en page avec la classe thesul.

Remerciements

«Avant, je ne savais pas que la beauté avait un sens. Je l'acceptais comme telle, comme une réalité sans rime ni raison. J'étais dans l'ignorance. A présent, je sais, ou plus exactement, je commence à savoir. Cette herbe me paraît beaucoup plus belle maintenant que je sais pourquoi elle est herbe, par quelle alchimie du soleil, de la pluie et de la terre elle est devenue ce qu'elle est.»

Martin Eden, Jack London.

En premier lieu, je remercie mes directeurs de thèse Jérémie Detrey et Emmanuel Thomé. Ils m'ont proposé un sujet de thèse passionnant, qui m'a été très enrichissant, que cela soit d'un point de vue théorique ou méthodologique. J'ai compris beaucoup de choses sur la science en général grâce à ce sujet. Par ailleurs, mes directeurs de thèse m'ont beaucoup aidé à clarifier, formaliser, rendre intelligible, les contributions que j'ai pu apporter à mon sujet de thèse. Ils ont su me laisser la liberté nécessaire pour être créatif et avoir de bonnes idées, et ils ont su m'aider à aller jusqu'au bout de la rédaction.

En second lieu, je remercie tous les membres de mon jury de thèse de s'être intéressé à mon travail et de l'avoir évalué. Je remercie Daniel Augot et Clément Pernet pour avoir accepté d'être rapporteurs, et d'avoir relu mon manuscrit. Je remercie Sylvie Boldo et Natacha Portier pour avoir accepté d'être membre de mon jury de thèse. Je remercie également Hugues Randriambololona d'avoir accepté de venir à ma soutenance.

Je remercie tous mes collègues de l'équipe CARAMBA. En particulier, je remercie tous mes cobureaux: Laurent, Hamza, Simon, Sandra, Aude, Kevin. Ils ont participé à de longs débats dont le niveau de trollitude sera inscrit dans les annales. Les matchs de ping-pong avec Laurent et Hamza ont été marquants. Je remercie spécialement Laurent pour sa disponibilité et sa connaissance très fine des subtilités administratives. Je remercie les doctorants, stagiaires et post-doctorants du pique-nique des doctorants pour ces longues discussions du mercredi midi. La partie de Diplomacy qu'on a organisée ensemble sur plusieurs semaines est un souvenir mémorable, qui aura brisé toute confiance entre nous et fait du LORIA un lieu de complot permanent. Je remercie les membres de l'équipe PESTO, en particulier Ludovic, Eric, Joseph, Alicia, Catalin et Ivan, avec lesquels j'ai eu de longues discussions, des parties de coinche, de go, d'échecs,... Enfin, je remercie Aurélien, Hugo, Jordi et Paul pour ces séances d'escalade à Casabloc ou au mur du SUAPS.

Je remercie les amis qui m'ont soutenu et qui ont été disponible pendant ma thèse. Je remercie Razvan et Dorian en particulier pour les discussions mathématiques, les échanges d'énigmes (à base de schtroumpfs entre autres) et leur passion des sciences. Je remercie spécialement Claire pour sa patience et pour avoir partagé ces quatre années avec moi. Elle m'a permis de sortir de mon quotidien et je n'aurais sans doute pas pleinement profité de ma thèse sans elle. Je remercie toute ma famille pour avoir été un soutien moral indéfectible. Je remercie mes parents qui ont toujours été là pour moi. C'est grâce à eux que j'ai acquis très tôt la passion des sciences et des mathématiques en particulier. Cette thèse n'aurait jamais existé sans leurs efforts permanents. Enfin, je remercie toutes les personnes avec lesquelles j'ai noué des liens d'amitié forts et qui m'ont permis de m'ouvrir pendant ma thèse à de nombreux domaines, en philosophie, en littérature, en économie, en politique... Ces quatre années auront marqué ma vie et changé ce que j'étais sur le plan intellectuel et humain, et je remercie donc tous ceux que j'ai rencontrés pendant cette période et qui m'ont fait progresser.

Résumé en français

De nombreux problèmes de calcul formel utilisent la multiplication comme brique de base dans les algorithmes qui permettent leur résolution. Par exemple, des problèmes de théorie algorithmique des nombres nécessitent souvent de savoir multiplier rapidement de grands entiers. La recherche de grands nombres premiers, le pgcd de polynômes, requièrent des algorithmes efficaces pour multiplier des polynômes ou des grands entiers. Des problèmes tels que le logarithme discret modulo un nombre premier p utilisent souvent la multiplication modulo p comme une brique de base. Cette thèse présente des algorithmes permettant de calculer le produit de différents objets mathématiques, tels que les grands entiers, les polynômes, les matrices... Les algorithmes permettant la résolution de ces problèmes peuvent être rangés en deux catégories: les algorithmes asymptotiquement rapides et les algorithmes efficaces pour de petites tailles (polynômes de petits degrés, matrices de petites dimensions). Par exemple, si on considère des polynômes de degré d à coefficients dans un anneau $\mathcal{R}$, trouver un algorithme efficace quand d croît est un problème de la première catégorie, tandis que trouver le meilleur algorithme quand d est fixé est un problème de la seconde catégorie. Ces problèmes sont tous les deux reliés, comme on peut l'illustrer avec l'exemple de la multiplication d'entiers ou de matrices.

La première amélioration de l'algorithme naïf qui multiplie des entiers de n bits avec une complexité quadratique, donc en $O(n^2)$, a été établie en 1962, bien que Kolmogorov avait conjecturé en 1952 que la complexité quadratique était optimale. Cette première amélioration est due à Karatsuba et Ofman [37]. Leur méthode transforme deux entiers de n bits a et b en polynômes A et B de longueur 2, obtenus en coupant a et b en deux moitiés. Ainsi, $A = a_0 + a_1X$ et $B = b_0 + b_1X$, avec $A(2^{\lceil n/2 \rceil}) = a$, $B(2^{\lceil n/2 \rceil}) = b$ et la taille en bits de a_0, a_1, b_0, b_1 est approximativement $n/2$. Karatsuba et Ofman ont réduit le problème de la multiplication d'entiers de n-bits à la multiplication de polynômes linéaires avec des coefficients de $n/2$ bits. Le produit $A \cdot B$ nécessite, pour son calcul, quatre multiplications utilisant l'algorithme naïf : a_0b_0, a_1b_0, a_0b_1, a_1b_1. Avec l'amélioration de Karatsuba et Ofman, les coefficients du produit $A \cdot B$ peuvent être calculés avec trois multiplications non scalaires (on ne compte pas les multiplications par un élément de l'anneau de base) : a_0b_0, $(a_0 + a_1)(b_0 + b_1)$, a_1b_1. On obtient les formules suivantes :

$$\begin{aligned} a_0b_0 &= a_0b_0, \\ a_0b_1 + a_1b_0 &= (a_0 + a_1)(b_0 + b_1) - a_0b_0 - a_1b_1, \\ a_1b_1 &= a_1b_1. \end{aligned}$$

Ces formules sont optimales si le nombre de multiplications non scalaires impliquées dans leur expression est minimal. L'algorithme de Karatsuba et Ofman peut être utilisé pour améliorer la complexité binaire de la multiplication de deux entiers de n bits en l'appliquant de façon récursive : au lieu d'une complexité en $O(n^2)$ avec l'algorithme naïf, ils obtiennent $O(n^{\log_2 3})$. Il est

courant de se référer à l'algorithme de Karatsuba et Ofman comme "l'algorithme de Karatsuba". En conséquence, c'est ce qu'on fait dans la suite.

Il est clair que, étant donné un degré $d > 1$, trouver des formules similaires à Karatsuba pour calculer le produit de polynômes de degré d améliore la complexité du produit d'entiers. Par exemple, dans [57], les entiers sont découpés en trois parties, et la multiplication d'entiers se réduit au problème de multiplier des polynômes de degré deux, ce qu'on peut réaliser en cinq multiplications. La complexité est alors $O(n^{\log_3 5})$. De manière générale, on peut déduire d'un ensemble de formules pour un produit de polynômes un algorithme pour le produit d'entiers. En réalité, les algorithmes de Karatsuba et de Toom-Cook peuvent être compris comme des cas particuliers de deux problèmes.

La première manière de les voir est comme une application de la méthode d'interpolation de Lagrange. En effet, étant donné un polynôme de degré d avec des coefficients dans un corps $\mathcal{R}$ contenant $d+1$ points distincts, on peut retrouver les coefficients du polynôme à partir de ses évaluations en $d+1$ points distincts du corps $\mathcal{R}$, en utilisant l'interpolation de Lagrange. Par exemple, l'algorithme de Karatsuba peut être compris comme l'évaluation de $A \cdot B$ en 0, 1 et le point particulier ∞ (l'évaluation en ∞ donne le terme dominant d'un polynôme), étant donné que

$$A(0) \cdot B(0) = a_0 b_0,\ A(1) \cdot B(1) = (a_0 + a_1)(b_0 + b_1) \text{ et } A(\infty) \cdot B(\infty) = a_1 b_1.$$

L'algorithme de Toom-Cook peut être compris comme l'évaluation de $A \cdot B$ en 0, -1, 1, 2 et ∞.

La seconde manière de les voir est, étant donné des polynômes de degré d avec leurs coefficients dans un corps $\mathcal{R}$, comme des formules minimisant le nombre de multiplications non scalaires requises pour calculer leur produit. Ces multiplications non scalaires ne sont pas nécessairement des évaluations du produit $A \cdot B$ en certains points. De plus, on peut aussi considérer le problème spécifique sur un corps fini $\mathbb{F}_q$ avec $q < d+1$, ce qui implique qu'on ne peut pas utiliser l'interpolation de Lagrange comme ci-dessus (car on n'a pas $d+1$ points distincts dans $\mathbb{F}_q$). On a une approximation du nombre minimal de multiplications non scalaires requises pour le calcul d'un produit de polynômes de degré d via la notion de "rang bilinéaire" d'une "application bilinéaire" représentant le produit de polynômes. On définit plus tard ces notions. Cependant, on peut décrire comment on obtient le rang bilinéaire pour l'exemple du produit de polynômes de degré d à coefficients dans $\mathbb{F}_2$. On formule le problème en termes de matrices. Les coefficients du produit de polynômes de degré deux sont vus comme cinq formes bilinéaires :

$$\left(\begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix}, \begin{pmatrix} b_0 \\ b_1 \\ b_2 \end{pmatrix}\right) \mapsto a_0 b_0,\ \left(\begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix}, \begin{pmatrix} b_0 \\ b_1 \\ b_2 \end{pmatrix}\right) \mapsto a_1 b_0 + a_0 b_1,\ \left(\begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix}, \begin{pmatrix} b_0 \\ b_1 \\ b_2 \end{pmatrix}\right) \mapsto a_2 b_0 + a_1 b_1 + a_0 b_2,$$

$$\left(\begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix}, \begin{pmatrix} b_0 \\ b_1 \\ b_2 \end{pmatrix}\right) \mapsto a_2 b_1 + a_1 b_2 \text{ et } \left(\begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix}, \begin{pmatrix} b_0 \\ b_1 \\ b_2 \end{pmatrix}\right) \mapsto a_2 b_2.$$

Ces formes bilinéaires sont représentées, dans la base canonique, par les matrices

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

Le cardinal de l'ensemble minimal de matrices M_i de rang un telles que chacune des matrices précédentes est une combinaison linéaire des M_i définit le rang bilinéaire du produit. Par exemple, à partir des six matrices

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

$$M_3 = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}, M_4 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}, M_5 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix},$$

représentant les multiplications non scalaires a_0b_0, $(a_0 + a_1)(b_0 + b_1)$, a_1b_1, $(a_0 + a_1 + a_2)(b_0 + b_1 + b_2)$, a_2b_2 et $(a_1 + a_2)(b_1 + b_2)$, on déduit des formules permettant le calcul des coefficients du produits de polynômes de degré deux utilisant six multiplications. On peut prouver qu'il n'existe pas d'ensemble de matrices de rang un de cardinal cinq permettant de calculer les coefficients de ce produit sur $\mathbb{F}_2$. Ainsi, le rang bilinéaire du produit de polynômes de degré deux sur $\mathbb{F}_2$ est six. En ce sens, les formules obtenues à partir des M_i sont optimales au sens du rang bilinéaire.

Le produit de n'importe quel objet mathématique (matrices, polynômes) peut être compris comme une application bilinéaire. Trouver des formules optimales (au sens du rang bilinéaire) pour évaluer des applications bilinéaires est un problème appartenant à un domaine appelé "théorie de la complexité algébrique" [12, 11, 56, 36]. Trouver des algorithmes optimaux pour le calcul du produit de matrices ou de polynômes sont des problèmes importants de complexité algébrique : la complexité du produit de matrices a été étudiée dans [56, 18, 43] et la complexité du produit de polynômes dans [37, 57, 52, 32].

Pour le produit de matrices, le premier résultat est apparu en 1969 : Strassen [56] a proposé des formules améliorant le coût du produit de deux matrices 2×2, ce qui, appliqué récursivement sur des matrices de dimensions plus larges, donne une complexité binaire en $O(n^{\log_2 7})$ au lieu de $O(n^{\log_2 8}) = O(n^3)$ (algorithme naïf). Pour un produit de matrices

$$\begin{pmatrix} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \end{pmatrix} \text{ et } \begin{pmatrix} b_{0,0} & b_{0,1} \\ b_{1,0} & b_{1,1} \end{pmatrix},$$

Strassen a proposé d'utiliser les multiplications non scalaires suivantes :

$$\pi_0 = (a_{0,0} + a_{1,1})(b_{0,0} + b_{1,1}), \pi_1 = (a_{1,0} + a_{1,1})b_{0,0}, \pi_2 = a_{0,0}(b_{0,1} - b_{1,1}), \pi_3 = a_{1,1}(b_{1,0} - b_{0,0}),$$

$$\pi_4 = (a_{0,0} + a_{0,1})b_{1,1}, \pi_5 = (a_{1,0} - a_{0,0})(b_{0,0} + b_{0,1}), \pi_6 = (a_{0,1} - a_{1,1})(b_{1,0} + b_{1,1}).$$

Les formules correspondantes sont :

$$\begin{aligned} a_{0,0}b_{0,0} + a_{0,1}b_{1,0} &= \pi_0 + \pi_3 - \pi_4 + \pi_6, \\ a_{0,0}b_{0,1} + a_{0,1}b_{1,1} &= \pi_2 + \pi_4, \\ a_{1,0}b_{0,0} + a_{1,1}b_{1,0} &= \pi_1 + \pi_3, \\ a_{1,0}b_{0,1} + a_{1,1}b_{1,1} &= \pi_0 - \pi_1 + \pi_2 + \pi_5. \end{aligned}$$

Si on note par ω l'exposant matriciel, c'est-à-dire la quantité telle que $O(n^\omega)$ est la meilleure borne supérieure asymptotique de la complexité arithmétique du produit de matrices $n \times n$ sur $\mathbb{C}$,

de nombreuses contributions [18, 15, 16, 43, 55, 61] ont permis d'améliorer les bornes supérieures sur ω. La meilleure borne à ce jour est due à Williams et Le Gall [61, 43] et est égale à 2.3729, bien que la borne proposée par Coppersmith et Winograd [18], égale à 2.376, est souvent prise comme référence. Dans [15], il est conjecturé que $\omega = 2$. Cependant, aucun des algorithmes proposés ci-dessus ne sont utilisés en pratique et on utilise encore largement l'algorithme naïf, sauf pour des matrices de grandes tailles, auquel cas on applique l'algorithme de Strassen. De manière générale, on peut utiliser une formule permettant de multiplier des matrices de tailles fixes pour multiplier des matrices de tailles quelconques, en appliquant cette formule de façon récursive. Ceci explique l'intérêt de chercher des formules optimales pour des tailles légèrement plus grandes que celles du produit de matrices 2×2 par 2×2 (idéalement le produit de matrices 3×3 par 3×3).

Smirnov décrit dans [54] des algorithmes qui peuvent être utilisés en pratique pour des matrices de plus grande taille. On peut remarquer que, la plupart du temps, des algorithmes meilleurs que celui de Strassen sont inconnus. Par exemple, on ne sait toujours pas si le nombre minimal de multiplications nécessaires pour le calcul du produit de deux matrices 3×3 à coefficients dans $\mathbb{C}$ est égal à 23 (Laderman [40] a été le premier à proposer un algorithme en 23 multiplications), bien que Bläser a prouvé dans [8] que 19 est une borne inférieure sur le rang de ce produit. La borne proposée par Bläser est un cas spécial d'une borne plus générale pour n'importe quel produit de matrices $m \times n$ par $n \times m$. Cette borne a été améliorée par Landsberg [41] en utilisant de la géométrie algébrique [42].

Le principal obstacle à la recherche de formules optimales est le fait que le calcul du rang bilinéaire d'une application bilinéaire est NP-difficile [35]. La méthode des moindres carrés semble être l'une des plus populaires [54] pour approcher ce problème. Une autre manière de faire consiste à utiliser des éléments de la géométrie algébrique [5] et à trouver une généralisation de la décomposition des matrices en valeurs propres pour des tenseurs d'ordre trois. Cependant, ces méthodes ne fonctionnent par pour des applications bilinéaires sur des corps finis : elles supposent qu'on travaille sur des corps de caractéristique zéro comme $\mathbb{C}$ pour la méthode des moindres carrés ou un corps algébriquement clos pour les méthodes à base de géométrie algébrique. Dans notre contexte, on se concentre sur la recherche de formules optimales pour une application bilinéaire sur un corps fini. Sur un corps fini, le nombre de formules possibles impliquant r multiplications non scalaires est fini. Par conséquent, on peut utiliser des méthodes de recherche exhaustive pour trouver le rang bilinéaire d'une application bilinéaire, ainsi que toutes les formules optimales.

Pour le produit de polynômes sur des corps finis, il existe une méthode originale utilisant des courbes algébriques [14], qui a permis d'améliorer les bornes supérieures sur le rang bilinéaire du produit de polynômes ou le produit d'éléments d'un corps fini de petite caractéristique. En particulier, la complexité bilinéaire de cette stratégie a été décrite dans [50, 1, 48] et dépend du rang bilinéaire du produit d'éléments dans $\mathbb{F}_{q^d}[y]/(y^\ell)$, où d et ℓ sont des entiers positifs. Ainsi, il devrait être possible d'améliorer ces bornes pour des petites valeurs de d et ℓ avec des méthodes à base de recherche exhaustive, puisque le produit d'éléments de $\mathbb{F}_{q^d}[y]/(y^\ell)$ est une application bilinéaire.

Montgomery propose dans [45] un algorithme pour calculer de telles décompositions pour le cas particulier de polynômes de petit degré sur des corps finis. L'auteur énumère toutes les formules possibles d'une certaine forme, en utilisant le fait que, sur un corps fini, le nombre de formules optimales possibles est fini. Il obtient alors de nouvelles formules pour le produits de polynômes de degrés cinq, six et sept à coefficients dans $\mathbb{F}_2$. Dans [46], Oseledets propose une approche heuristique pour résoudre le problème du rang bilinéaire pour le produit de polynômes à coefficients dans $\mathbb{F}_2$. L'auteur utilise une formalisation de ce problème en termes d'espaces vectoriels et se restreint à des formules présentant certaines symétries. Plus tard, Barbulescu et

al. développent dans [2] ce formalisme et calculent le rang bilinéaire pour différentes applications bilinéaires, telles que le produit court ou le produit médian sur un corps fini. Leur algorithme permet de générer toutes les formules optimales pour n'importe quelle application bilinéaire sur un corps fini. Ce travail est la source principale d'inspiration de la première partie de cette thèse. Le but de la première partie est d'améliorer le temps requis pour trouver toutes les formules optimales pour une application bilinéaire donnée, et de prouver des bornes inférieures sur le rang bilinéaire, en particulier pour le produit de matrices et le produit court sur $\mathbb{F}_2$.

L'amélioration qu'on propose à l'algorithme décrit dans [2] utilise le groupe constitué des automorphismes stabilisant une application bilinéaire, ainsi que la notion de "tige" d'un espace vectoriel de formes bilinéaires. Plus spécifiquement, on calcule toutes les formules optimales pour le produit court de polynômes P et Q modulo X^5 et le produit de matrices 3×2 par 2×3. Ce dernier produit était hors d'atteinte en pratique avec la méthode proposé dans [2]. On prouve, en particulier, que l'ensemble des formules optimales pour ce produit matriciel est unique à l'action d'un élément du groupe d'automorphismes près. Bien que fournir une analyse de complexité suffisamment fine de notre approche est difficile (elle implique des propriétés intrinsèques aux applications bilinéaires qu'on considère, dont l'impact sur la complexité finale est difficile à quantifier), on a des arguments théoriques pour expliquer pourquoi cette approche est plus rapide que [2].

Lorsque d est grand, les multiplications scalaires ne sont plus négligeables. Alors, même dans des corps tels que le corps complexe $\mathbb{C}$, dans lequel on a toujours $d+1$ points distincts, la complexité de l'évaluation en $d+1$ points distincts est un problème qui doit être pris en compte lorsque d croît. Par conséquent, pour le produit de polynômes de degré d avec d grand, il faut regarder comment choisir au mieux les $d+1$ points tels qu'on obtient la meilleure complexité. Ce problème est relié à la complexité asymptotique de la multiplication d'entiers, par le biais de la substitution de Kronecker. Plusieurs améliorations ont été réalisées pour la complexité binaire de la multiplication d'entiers, en particulier par Schönhage, Strassen, Fürer, Harvey, van der Hoeven et Lecerf [52, 26, 32]. Ces améliorations reposent sur la complexité de la multiplication de polynômes de grand degré et sur le schéma d'évaluation-interpolation. Cependant, leur optimalité est toujours une question ouverte.

L'algorithme naïf ou l'amélioration de Karatsuba et Ofman permettent d'avoir un algorithme avec une complexité polynomiale pour multiplier des entiers. L'algorithme de Toom-Cook a la même propriété. Le premier algorithme réalisant une complexité dite *quasi-linéaire* est celui de Schönhage et Strassen [52, 51] : leur algorithme multiplie des entiers de n bits avec une complexité binaire égale à $O(n \cdot \log n \cdot \log^{(2)} n)$. L'algorithme de Schönhage-Strassen utilise la transformée de Fourier rapide (FFT pour Fast Fourier Transform) pour évaluer rapidement un polynôme en les puissances d'une racine primitive de l'unité [59, §8]. De plus, la complexité est obtenue via un choix adéquat d'un anneau $\mathcal{R}$ dans lequel cette évaluation doit être réalisée. Concrètement, le choix $\mathcal{R} = \mathbb{Z}/(2^e+1)$, avec e une puissance de deux, amène à une complexité en $O(n \cdot \log n \cdot \log\log n)$, tandis que d'autres choix naturels pour $\mathcal{R}$, tels que $\mathcal{R} = \mathbb{C}$, amènent à une complexité moins bonne.

En 2007, M. Fürer observe que l'anneau $\mathcal{R} = \mathbb{C}[x]/(x^P+1)$, avec P une puissance de deux bien choisie, est particulièrement intéressant [26]. En utilisant cet anneau $\mathcal{R}$, il est possible d'exploiter une FFT avec une grande base pour obtenir une complexité en $O(n \cdot \log n \cdot 2^{O(\log^* n)})$ (ce qui s'oppose à la FFT en base deux qui était utilisée dans l'algorithme de Schönhage-Strassen). La notation $\log^*$ représente le logarithme itéré (voir §6.6). Le résultat de Fürer a été une amélioration surprenante de la complexité de Schönhage-Strassen, qui était considérée comme la meilleure pendant 35 ans. Une extension récente du travail de Fürer, proposée dans [21], remplace le corps $\mathbb{C}$ dans la définition de $\mathcal{R}$ par un anneau p-adique et atteint la même complexité

asymptotique. Cette variante p-adique permet de mettre de côté les problèmes de précision qu'on a dans $\mathbb{C}$. Harvey, van der Hoeven et Lecerf dans [32], et plus tard Harvey et van der Hoeven dans [30] ont proposé de nouveaux algorithmes et une analyse de complexité plus fine permettant d'obtenir une expression plus explicite de la complexité binaire, c'est-à-dire $O(n \cdot \log n \cdot 8^{\log^* n})$ et $O(n \cdot \log n \cdot 4^{\log^* n})$ conjecturalement. Ils montrent aussi qu'une analyse de l'algorithme original de Fürer permet d'arriver à $O(n \cdot \log n \cdot 16^{\log^* n})$. Très récemment, dans [31], Harvey et Van der Hoeven semblent avoir montré qu'on peut arriver à une complexité en $O(n \cdot \log n \cdot 4^{\log^* n})$ sans faire appel à aucune conjecture de théorie des nombres. Cependant, ces algorithmes, qui réalisent une borne asymptotique similaire à celle obtenue par Fürer, sont surtout perçus, pour l'instant, comme des résultat n'ayant qu'une valeur théorique.

On décrit dans la deuxième partie de cette thèse un algorithme atteignant une complexité en $O(n \cdot \log n \cdot 4^{\log^* n})$ et reposant sur une conjecture qui peut être vue comme une version explicite de la conjecture de Bateman-Horn [4], appuyée par des expériences numériques. Plus précisément, cette conjecture se formule de la façon suivante.

Hypothèse 8.5. *Soit $\lambda \geq 2$ un entier. Pour tout nombre réel R tel que $2^\lambda \leq R \leq 2^{2\lambda}$, il existe un premier généralisé de Fermat $p = r^{2^\lambda} + 1$ tel que $R \leq r < \lambda^{2.5} R$.*

Le concept clé de notre algorithme est l'utilisation d'une chaîne de premiers généralisés de Fermat (de la forme $r^{2^\lambda} + 1$) pour gérer les appels récursifs. Cette variante est différente de la stratégie proposée dans [32]. Notre travail peut être compris comme étant une généralisation d'un rapport proposé par Fürer [25] (en 1989), qui proposait un algorithme reposant sur l'hypothèse qu'il existe une infinité de premiers de Fermat. Cependant, cette hypothèse est très probablement fausse, et donc notre approche permet de réparer ce problème.

Cette thèse est organisée de la façon suivante.

- Dans la **Partie I** on présente le travail qui a été réalisé sur la recherche de formules optimales et décrit dans [19]. Dans le **Chapitre 1**, on présente les outils théoriques et le cadre formel de cette partie de la thèse, qui a déjà été introduit dans [2]. Dans le **Chapitre 2**, on présente l'état de l'art, avec entre autres les algorithmes proposés dans [2] et [3]. On décrit les aspects théoriques et algorithmiques de notre amélioration dans le **Chapitre 3**, On applique cette amélioration à des exemples tels que ceux du produit court et du produit matriciel dans le **Chapitre 4**. On décrit dans le **Chapitre 5** comment calculer certains précalculs requis dans le **Chapitre 3**.

- Dans la **Partie II**, on présente le travail décrit dans [20], relatif à la recherche d'un algorithme asymptotiquement optimal pour la multiplication de grands entiers, basé sur la transformée de Fourier rapide et la stratégie de Fürer [26]. Dans **Chapitre 6**, on décrit la transformée de Fourier rapide et, en particulier, l'algorithme de Cooley-Tukey. Dans le **Chapitre 7**, on décrit l'algorithme de Fürer, basé sur la transformée de Fourier rapide, et permettant de multiplier des entiers de n bits en $O(n \cdot \log_2 n \cdot 2^{O(\log^* n)})$. On détaille les propriétés des premiers généralisés de Fermat dans le **Chapitre 8**. On propose un nouvel algorithme dans le **Chapitre 9**, basé sur la stratégie de Fürer et sur les premiers généralisés de Fermat, et on prouve qu'il multiplie des entiers de n bits en $O(n \cdot \log_2 n \cdot 4^{\log^* n})$. Dans le **Chapitre 10**, on discute des aspects pratiques d'une implémentation de notre algorithme de multiplications d'entiers.

Sommaire

Introduction

Various problems of symbolic computation use the multiplication as a building block in the algorithms that are developed to solve them. For example, problems in algorithmic number theory often require fast multiplication of large integers. The search of large primes or the computation of digits of π are such problems. The computation of resultants, gcd of polynomials, require efficient algorithms to multiply polynomials. Problems such as the discrete logarithm modulo a prime p often use the multiplication modulo p as a building block. This thesis presents algorithms to compute the product of various mathematical objects, such as large integers, polynomials, matrices... The algorithms solving these problems can be decomposed into two families: algorithms that are asymptotically fast and algorithms that are efficient for small sizes. For instance, considering the polynomials of degree d over a ring $\mathcal{R}$, finding an efficient algorithm when d grows falls into the first family, whereas finding the best algorithm when d is fixed falls in the second family. These problems are actually related, which is illustrated by the problem of integer multiplication and matrix multiplication.

The first improvement to the schoolbook algorithm multiplying n-bit integers with a quadratic complexity, i.e. with a complexity equal to $O(n^2)$, has been established in 1962, although Kolmogorov conjectured in 1952 that the quadratic complexity was optimal. This first improvement is due to Karatsuba and Ofman [37]. They transform two n-bit integers a and b into polynomials A and B of length two, obtained by splitting a and b into two halves. Thus, $A = a_0 + a_1X$ and $B = b_0 + b_1X$, where $A(2^{\lceil n/2 \rceil}) = a$ and $B(2^{\lceil n/2 \rceil}) = b$ and the bit-length of a_0, a_1, b_0, b_1 is roughly $n/2$. Karatsuba and Ofman reduce the problem of the multiplication of n-bit integers to the problem of the multiplication of linear polynomials whose coefficients have size $n/2$. The product $A \cdot B$ requires, to be computed, four multiplications using the schoolbook algorithm: a_0b_0, a_1b_0, a_0b_1, a_1b_1. With Karatsuba and Ofman's improvement, the coefficients of the product $A \cdot B$ can be retrieved from the computation of the three following non-scalar multiplications (multiplications by an element of the base ring are not taken into account): a_0b_0, $(a_0 + a_1)(b_0 + b_1)$, a_1b_1. We have the following formulas:

$$\begin{aligned} a_0b_0 &= a_0b_0, \\ a_0b_1 + a_1b_0 &= (a_0 + a_1)(b_0 + b_1) - a_0b_0 - a_1b_1, \\ a_1b_1 &= a_1b_1. \end{aligned}$$

These formulas are optimal if the number of non-scalar multiplications involved in their expression is minimal. In particular, Karatsuba and Ofman's algorithm can be used to improve the asymptotic binary complexity of the multiplication of two n-bit integers by applying it recursively: instead of $O(n^2)$ with the naive schoolbook algorithm, they obtain $O(n^{\log_2 3})$. It is common to refer to Karatsuba and Ofman's algorithm as "Karatsuba's algorithm". Therefore, we do it in the following.

Then, it is clear that, given a degree $d > 1$, finding the minimal amount of multiplications to compute the product of polynomials of degree d leads to even better complexities. For example, in [57], the integers are split into three parts, and the integer multiplication is reduced to the problem of multiplying polynomials of length three, which is realized via five multiplications. The complexity is then $O(n^{\log_3 5})$. One deduces from a set of formulas for a product of polynomials an algorithm for the product of integers. In fact, Karatsuba and Toom-Cook's algorithms can be seen as special cases of two problems.

The first way to see it is as an application of Lagrange interpolation method. Given a polynomial of degree d with coefficients lying in a field $\mathcal{R}$ containing $d+1$ distinct points, one can recover the coefficients of the polynomial from its evaluations at $d+1$ distinct points of the field $\mathcal{R}$, using Lagrange interpolation. For example, Karatsuba's algorithm can be seen as the evaluation of $A \cdot B$ at 0, 1 and the special point ∞ (the evaluation in ∞ gives the dominant term of a polynomial), since

$$A(0) \cdot B(0) = a_0 b_0,\ A(1) \cdot B(1) = (a_0 + a_1)(b_0 + b_1) \text{ and } A(\infty) \cdot B(\infty) = a_1 b_1.$$

Toom-Cook's algorithm can be seen as the evaluation of $A \cdot B$ at 0, -1, 1, 2 and ∞.

The second way to see it is, given polynomials of degree d with coefficients lying in a field $\mathcal{R}$, as the formulas minimizing the quantity of non-scalar multiplications required to compute their product. These non-scalar multiplications do not necessarily correspond to an evaluation of the product $A \cdot B$ at some points. Moreover, one can also consider the specific problem in a field $\mathbb{F}_q$ with $q < d+1$, which implies that one cannot use the Lagrange interpolation as above (we do not have $d+1$ distinct points in $\mathbb{F}_q$). We can approach the minimal number of non-scalar multiplications necessary for the computation of a product of polynomials of degree d by the "bilinear rank" of the "bilinear map" representing the product of polynomials. We define later more formally these notions. However, we can describe how to obtain the bilinear rank for the example of product of polynomials of degree d over $\mathbb{F}_2$. We formulate the problem in terms of matrices. The coefficients of the product of polynomials of degree two are given by the five bilinear forms:

$$\left(\begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix}, \begin{pmatrix} b_0 \\ b_1 \\ b_2 \end{pmatrix}\right) \mapsto a_0 b_0,\ \left(\begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix}, \begin{pmatrix} b_0 \\ b_1 \\ b_2 \end{pmatrix}\right) \mapsto a_1 b_0 + a_0 b_1,\ \left(\begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix}, \begin{pmatrix} b_0 \\ b_1 \\ b_2 \end{pmatrix}\right) \mapsto a_2 b_0 + a_1 b_1 + a_0 b_2,$$

$$\left(\begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix}, \begin{pmatrix} b_0 \\ b_1 \\ b_2 \end{pmatrix}\right) \mapsto a_2 b_1 + a_1 b_2 \text{ and } \left(\begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix}, \begin{pmatrix} b_0 \\ b_1 \\ b_2 \end{pmatrix}\right) \mapsto a_2 b_2.$$

These bilinear forms can be represented in the canonical basis by the matrices

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

The cardinality of the minimal set of rank-one matrices M_i such that each one of the previous matrices is a linear combination of the M_i's gives the bilinear rank of the product. For example,

from the six matrices

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

$$M_3 = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}, M_4 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}, M_5 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix},$$

representing the non-scalar multiplications a_0b_0, $(a_0+a_1)(b_0+b_1)$, a_1b_1, $(a_0+a_1+a_2)(b_0+b_1+b_2)$, a_2b_2 and $(a_1+a_2)(b_1+b_2)$, we deduce formulas to compute the coefficients of the product of polynomials of degree 2 using 6 multiplications. One can prove that there is no set of rank-one matrices of cardinality 5 allowing one to compute the coefficients of this product over $\mathbb{F}_2$. Thus, the bilinear rank of the product of polynomials of degree two over $\mathbb{F}_2$ is six. In that sense, the formulas obtained with the M_i's are optimal for the bilinear rank.

The product of any mathematical objects (matrices, polynomials,...) can described as a bilinear map. Finding optimal formulas (for the bilinear rank) for evaluating bilinear maps is a problem of an area called "algebraic complexity theory" [12, 11, 56, 36]. Finding optimal algorithms for the computation of the product of matrices or polynomials are important problems of the algebraic complexity theory: the complexity of the matrix product has been well studied in [56, 18, 43] and the complexity of polynomial multiplication in [37, 57, 52, 32].

For the product of matrices, the first result came in 1969: Strassen [56] proposed formulas improving on the cost of the product of two matrices 2×2, which, when applied recursively on large matrices, leads to a binary complexity $O(n^{\log_2 7})$ instead of $O(n^{\log_2 8}) = O(n^3)$. For a product of matrices

$$\begin{pmatrix} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \end{pmatrix} \text{ and } \begin{pmatrix} b_{0,0} & b_{0,1} \\ b_{1,0} & b_{1,1} \end{pmatrix},$$

Strassen proposed the following non-scalar products:

$$\pi_0 = (a_{0,0} + a_{1,1})(b_{0,0} + b_{1,1}), \pi_1 = (a_{1,0} + a_{1,1})b_{0,0}, \pi_2 = a_{0,0}(b_{0,1} - b_{1,1}), \pi_3 = a_{1,1}(b_{1,0} - b_{0,0}),$$

$$\pi_4 = (a_{0,0} + a_{0,1})b_{1,1}, \pi_5 = (a_{1,0} - a_{0,0})(b_{0,0} + b_{0,1}), \pi_6 = (a_{0,1} - a_{1,1})(b_{1,0} + b_{1,1}).$$

The associated formulas are:

$$\begin{aligned} a_{0,0}b_{0,0} + a_{0,1}b_{1,0} &= \pi_0 + \pi_3 - \pi_4 + \pi_6, \\ a_{0,0}b_{0,1} + a_{0,1}b_{1,1} &= \pi_2 + \pi_4, \\ a_{1,0}b_{0,0} + a_{1,1}b_{1,0} &= \pi_1 + \pi_3, \\ a_{1,0}b_{0,1} + a_{1,1}b_{1,1} &= \pi_0 - \pi_1 + \pi_2 + \pi_5. \end{aligned}$$

Denoting by ω the matrix exponent, the quantity such that $O(n^\omega)$ is the best asymptotic upper bound on the bilinear rank of the product of $n \times n$ matrices in $\mathbb{C}$, a lot of contributions [18, 15, 16, 43, 55, 61] improved the upper bound on ω. The best bound known is due to Williams and Le Gall [61] and is equal to 2.3729, although the bound proposed by Coppersmith and Winograd [18], equal to 2.376, is often used as a reference. It is conjectured in [15] that this exponent is

actually equal to two. However, it appears that none of these algorithms are practical and the naive algorithm is still widely in use, although one prefers Strassen's algorithm to multiply large matrices. When the size of the matrices to multiply is very large, we call Strassen's algorithm recursively, which explains the need to search optimal formulas for a matrix product slightly larger than the product of matrices 2×2 by 2×2 (ideally the product of matrices 3×3 by 3×3).

Smirnov describes in [54] practical algorithms for matrices of higher dimensions. Thus, one can notice that, most of the time, algorithms for matrix multiplication better than the Strassen's algorithm are unknown. For example, it is still unknown whether the minimal number of multiplications involved in the computation of the product of two matrices 3×3 with coefficients in $\mathbb{C}$ is equal to 23 or not (Laderman [40] was the first to propose an algorithm with 23 multiplications), although Bläser proved in [8] that 19 is a lower bound. The bound proposed by Bläser is a special case of a more general bound on any product of matrices of formats $m \times n$ and $n \times m$. It has been improved by Landsberg [41] by using algebraic geometry [42].

The main obstacle to finding optimal formulas is the fact that computing the bilinear rank of a bilinear map is known to be NP-hard [35]. The least-squares method seems to be one of the most popular [54] to approach this problem. Another way to decompose a bilinear map consists in using ingredients from algebraic geometry [5] and in finding a generalization of the singular value decomposition of matrices to general tensors. However, these methods do not work for bilinear maps over a finite field: they work on field of characteristic zero as $\mathbb{C}$ for the least-squares method or an algebraically closed field for the methods using geometry. In our context, we focus on finding optimal formulas of a bilinear map over a finite field. On a finite field, the possible formulas involving r non-scalar multiplications is finite. Therefore, we can use exhaustive search methods to find the bilinear rank of a bilinear map, and all the optimal formulas.

For the product of polynomials over finite fields, there exists an original method using algebraic curves [14], improving on the bilinear of the product of polynomials or the product of two elements of a finite field. In particular, the bilinear complexity of this strategy has been bounded in [50, 1, 48] and depends on the bilinear rank of the product of elements in $\mathbb{F}_{q^d}[y]/(y^\ell)$, where d and ℓ are positive integers. Thus, it should be possible to improve this bound for small values of d and ℓ with exhaustive search methods, since the multiplication of elements of $\mathbb{F}_{q^d}[y]/(y^\ell)$ is a bilinear map.

Montgomery proposes in [45] an algorithm to compute such a decomposition for the particular case of polynomials of small degree over a finite field. The author enumerates all possible formulas of some form, taking advantage of the fact that, on a finite field, the number of possible optimal formulas is finite. Then, he gets new formulas for the multiplication of polynomials of degree five, six and seven over $\mathbb{F}_2$. In [46], Oseledets proposes a heuristic approach to solve the bilinear rank problem for the polynomial product over $\mathbb{F}_2$. The author uses a formalization of the problem in terms of vector spaces and looks at formulas with some symmetries. Later, Barbulescu et al. propose in [2] a unified framework, developing the use of the vector spaces formalism, and compute the bilinear rank of different applications, such as the short product or the middle product over a finite field. Their algorithm allows one to generate all the possible optimal formulas of any bilinear map over a finite field. This work is the main inspiration of the first part of this thesis. In this part, we improve the time required to find all the optimal formulas for a given bilinear map, and to establish lower bounds for the bilinear rank, in particular for the matrix multiplication and short product over $\mathbb{F}_2$.

The improvement to the algorithm presented in [2] that we describe relies on the automorphism group stabilizing a bilinear map, and on the notion of "stem" of a vector space associated to such a bilinear map. Specifically, we compute all the decompositions for the short product of

polynomials P and Q modulo X^5 and the product of 3×2 by 2×3 matrices. The latter problem was out of reach with the method used in [2]. We prove, in particular, that the set of possible decompositions for this matrix product is essentially unique, up to the action of the automorphism group. Although it is difficult to obtain a sharp complexity analysis of our approach (it involves intrinsic properties of particular bilinear maps, whose impact in the final complexity is difficult to quantify), we give theoretical arguments explaining why our approach is better than [2].

When d is large, the scalar multiplications are not negligible anymore. Thus, even in fields as the complex field $\mathbb{C}$, in which we always have $d+1$ distinct points, the complexity of the evaluation at $d+1$ points is a problem to consider when d grows. Therefore, for product of polynomials when d is large, we study how to choose $d+1$ points so that we have the best complexity. This problem is related to establishing the asymptotic binary complexity of the multiplication of integers, via Kronecker-substitution. Several improvements have been achieved for the binary complexity of the multiplication of integers, in particular by Schönhage, Strassen, Fürer, Harvey, van der Hoeven and Lecerf [52, 26, 32]. These improvements rely on the complexity of the multiplication of polynomials of large degree and on an evaluation-interpolation scheme. However, their optimality is still an open question.

The naive algorithm or Karatsuba and Ofman's improvement leads to a polynomial-time algorithm to multiply integers. Toom-Cook's algorithm falls also into this category. The first algorithm to achieve what is called *quasi-linear* complexity is Schönhage and Strassen's [52, 51]: their algorithm multiply n-bit integers with a binary complexity equal to $O(n \cdot \log n \cdot \log^{(2)} n)$. The Schönhage-Strassen algorithm uses the fast Fourier transform (FFT) as a means to quickly evaluate a polynomial at the powers of a primitive root of unity [59, §8]. Moreover, the complexity is obtained by an appropriate choice of a ring $\mathcal{R}$ in which this evaluation is to be carried out. Namely, the choice $\mathcal{R} = \mathbb{Z}/(2^e+1)$, for e a suitable power of two, yields the complexity $O(n \cdot \log n \cdot \log\log n)$, while other natural choices for $\mathcal{R}$, such as $\mathcal{R} = \mathbb{C}$, appear to yield inferior performance at the time.

In 2007, M. Fürer observes that the ring $\mathcal{R} = \mathbb{C}[x]/(x^P+1)$, for P a suitable power of two, is particularly interesting [26]. Using this ring $\mathcal{R}$, it is possible to take advantage of large-radix FFT to obtain the improved complexity $O(n \cdot \log n \cdot 2^{O(\log^* n)})$ (in contrast, radix-2 FFT is sufficient for the Schönhage-Strassen algorithm). The notation $\log^*$ denotes the iterated logarithm (see §6.6). Fürer's result was an acclaimed improvement on the complexity of the Schönhage-Strassen algorithm, which had remained unbeaten for 35 years. An early extension of Fürer's work, proposed in [21], replaces the field $\mathbb{C}$ in the definition of $\mathcal{R}$ by a p-adic ring and reaches an identical asymptotic complexity. This p-adic variant can be expected to ease precision issues for potential implementations. Harvey, van der Hoeven and Lecerf in [32], and later Harvey and van der Hoeven in [30] propose new algorithms and a sharper analysis that makes the expression of the complexity more explicit, namely $O(n \cdot \log n \cdot 8^{\log^* n})$ and even $O(n \cdot \log n \cdot 4^{\log^* n})$ conjecturally. In comparison, they also show that a careful analysis of Fürer's original algorithm reaches the complexity $O(n \cdot \log n \cdot 16^{\log^* n})$. Very recently, in [31], Harvey and Van der Hoeven seem to have proven that there exists an algorithm with a complexity in $O(n \cdot \log n \cdot 4^{\log^* n})$, multiplying integers without using any conjecture of number theory. However, these algorithms achieving a bound similar to the one obtained by Fürer are, as it stands, perceived as theoretical results.

We describe in the second part of this thesis an algorithm reaching the complexity $O(n \cdot \log n \cdot 4^{\log^* n})$ and relying on a conjecture which can be regarded as an explicit version of the Bateman-Horn conjecture [4], supported by numerical evidence. Namely, this assumption is as follows.

Hypothesis 8.5. *Let $\lambda \geq 2$ be an integer. For any real number R such that $2^\lambda \leq R \leq 2^{2\lambda}$, there exists a generalized Fermat prime $p = r^{2^\lambda} + 1$ such that $R \leq r < \lambda^{2.5} R$.*

The key concept of our algorithm is the use of a chain of generalized Fermat primes (of the form $r^{2^\lambda} + 1$) to handle recursive calls. This variant differs from the strategy proposed in [32]. Some lineage can be drawn between our work and an early article by Fürer [25] (from 1989), which is dependent on the assumption that there exist infinitely many Fermat primes. The latter assumption, however, is widely believed to be wrong, so our variant fills a gap here.

This thesis is organized as follows.

- In **Part I** we present the work that has been done on finding optimal formulas, and described in [19]. In **Chapter 1**, we present the theoretical tools and the framework for this part of the thesis, corresponding to the framework introduced in [2]. In **Chapter 2**, we present the state of the art, such as the algorithm proposed in [2] and [3]. We describe the theoretical and algorithmic aspects of our improvement in **Chapter 3**, We apply this improvement to examples such as the short product and the matrix product in **Chapter 4**. We describe in **Chapter 5** how to realize some precomputations required by **Chapter 3**.
- In **Part II**, we present the work described in [20] related to the search of an asymptotically optimal algorithm for the multiplication of large integers, based on the fast Fourier transform and on the Fürer's strategy [26]. In **Chapter 6**, we describe the fast Fourier transform and, in particular the Cooley-Tukey algorithm. In **Chapter 7**, we describe the Fürer's algorithm, based on the fast Fourier transform, and allowing one to multiply n-bit integers in $O(n \cdot \log_2 n \cdot 2^{O(\log^* n)})$. We detail the properties of generalized Fermat primes in **Chapter 8**. We propose a new algorithm in **Chapter 9**, based on Fürer's strategy and on generalized Fermat primes and we prove it multiplies n-bit integers in $O(n \cdot \log_2 n \cdot 4^{\log^* n})$. In **Chapter 10**, we discuss the practical aspects of an implementation of the integer multiplication algorithm using generalized Fermat primes.

Part I

Bilinear rank

Notations

K	Field
$\mathbf{a}, \mathbf{b}, \ldots$	Elements of a vector space over K
$a, b, \ldots$	Elements of K
A, B	Polynomials over K
M, N	Matrices over K
$\mathcal{L}(K^m; K)$	Linear forms from K^m to K
$\mathcal{L}(K^m, K^n; K)$	Linear forms from $K^m \times K^n$ to K
$\mathcal{M}_{m,n}(K)$	$m \times n$ matrices over K
Φ, Ψ	Bilinear form of any rank
ϕ, ψ	Bilinear form of rank one
T, W, V	Subspaces of $\mathcal{L}(K^m, K^n; K)$
$\mathrm{GL}(K^m) \times \mathrm{GL}(K^n)$	Couples of invertible matrices
σ	Element of $\mathrm{GL}(K^m) \times \mathrm{GL}(K^n)$

Chapter 1

From bilinear maps to vector spaces

In this chapter, we recall the definition of the bilinear rank of a bilinear map. We choose the characterization given by de Groote [22] or Bürgisser et al. [12, Ch. 14]. The algorithms that are detailed in the following chapters use vector spaces generated by matrices. These vector spaces are obtained from bilinear maps such as the short product or the matrix product.

1.1 Bilinear rank

1.1.1 Bilinear forms

Let K be a field. We denote by $\mathcal{L}(K^m; K)$ the vector space of linear forms from K^m to K and by $\mathcal{L}(K^m, K^n; K)$ the vector space of bilinear forms from $K^m \times K^n$ to K.

Definition 1.1 (Rank-one bilinear form)**.** *Let $\Phi \in \mathcal{L}(K^m, K^n; K)$. We say that Φ is a rank-one bilinear form if there exist two linear forms $\alpha \in \mathcal{L}(K^m; K)$ and $\beta \in \mathcal{L}(K^n; K)$ such that Φ can be written as $\Phi : (\mathbf{a}, \mathbf{b}) \mapsto \alpha(\mathbf{a}) \cdot \beta(\mathbf{b})$.*

Definition 1.2 (Bilinear rank of bilinear forms)**.** *The bilinear rank of a bilinear form Φ, denoted by* $\mathrm{rk}(\Phi)$*, is defined as the minimal number of rank-one bilinear forms ϕ_i such that Φ can be written as a linear combination of the ϕ_i's. Then, a family $(\phi_i)_i$ of cardinality* $\mathrm{rk}(\Phi)$ *generating Φ is said to be an optimal decomposition of Φ.*

Thus, the bilinear rank of a bilinear form Φ is a number defined by the minimal size of a decomposition of the form

$$\Phi(\mathbf{a}, \mathbf{b}) = \sum_{0 \leq i < r} \alpha_i(a_0, \ldots, a_{m-1}) \beta_i(b_0, \ldots, b_{n-1}).$$

For $i \in \{0, \ldots, m-1\}$ and $j \in \{0, \ldots, n-1\}$, we denote by $e_{i,j}$ the bilinear forms $e_{i,j} : (\mathbf{a}, \mathbf{b}) \mapsto a_i b_j$. The $e_{i,j}$'s have rank one and form the canonical basis of $\mathcal{L}(K^m, K^n; K)$. This implies that any bilinear form can be expressed as a linear combination of bilinear forms of rank one and, consequently, the bilinear rank is well defined.

We have a matrix equivalent of Definition 1.2. Indeed, for any bilinear form $\Phi \in \mathcal{L}(K^m, K^n; K)$, there exists a matrix $M \in \mathcal{M}_{m,n}(K)$ such that $\Phi(\mathbf{a}, \mathbf{b}) = \mathbf{a}^{\mathrm{T}} \cdot M \cdot \mathbf{b}$ for any $\mathbf{a} \in K^m$ and $\mathbf{b} \in K^n$.

It is shown in [12, Ch. 14] that the usual linear algebraic rank of a bilinear form (which is the rank of the corresponding matrix M) coincides with its bilinear rank.

There is a one-to-one correspondence between rank-one bilinear forms and rank-one matrices. Thus, the number of rank-one matrices gives the number of rank-one bilinear forms. The rank-one matrices are the set of matrices that can be obtained as a product $X \cdot Y^{\mathrm{T}}$ of two non-zero column matrices $X \in \mathcal{M}_{m,1}, Y \in \mathcal{M}_{n,1}$. There are $(\#K^m - 1)$ nonzero matrices $X \in \mathcal{M}_{m,1}$ and $(\#K^n - 1)$ nonzero matrices $Y \in \mathcal{M}_{n,1}$. If $X \cdot Y^{\mathrm{T}} = X' \cdot Y'^{\mathrm{T}}$, there exists $\lambda \in K - \{0\}$ such that $X = \lambda X'$ and $Y = \frac{1}{\lambda} Y'$. Therefore, there are

$$\frac{(\#K^m - 1)(\#K^n - 1)}{(\#K - 1)}$$

distinct bilinear forms of rank one.

Example 1.3 (Bilinear forms of $\mathcal{L}(K^2, K^2; K)$)**.** *When $K = \mathbb{F}_2$, there are $(2^2 - 1)^2 = 9$ bilinear forms of rank one in $\mathcal{L}(K^2, K^2; K)$. Their matrix representations in the canonical basis are*

$$\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix}, \text{ and } \begin{pmatrix} 0 & 0 \\ 1 & 1 \end{pmatrix}.$$

Any bilinear form $\Phi \in \mathcal{L}(K^2, K^2; K)$ can be decomposed in the basis $\{e_{0,0}, e_{1,0}, e_{0,1}, e_{1,1}\}$:

$$\Phi = \epsilon_{0,0} \cdot e_{0,0} + \epsilon_{1,0} \cdot e_{1,0} + \epsilon_{0,1} \cdot e_{0,1} + \epsilon_{1,1} \cdot e_{1,1}.$$

Consequently, the matrix representation of Φ in the canonical basis is

$$\begin{pmatrix} \epsilon_{0,0} & \epsilon_{0,1} \\ \epsilon_{1,0} & \epsilon_{1,1} \end{pmatrix}.$$

1.1.2 Bilinear maps

We denote by $\mathcal{L}(K^m, K^n; K^\ell)$ the vector space of bilinear maps from $K^m \times K^n$ to K^ℓ. Any bilinear map $\mathbf{\Phi}$ from $K^m \times K^n$ to K^ℓ can also be seen as an element of $\mathcal{L}(K^m, K^n; K)^\ell$, whose coordinates are bilinear forms, denoted by $(\Phi_k)_{0 \le k < \ell}$.

We consider in the following chapters various examples of bilinear maps given by classical products:

- the product of polynomials of m and n terms, of $m+n-1$ terms (in this case $\ell = m+n-1$),
- the short product, which is the product of polynomials modulo X^ℓ,
- the circulant product, which is the product of polynomials modulo $X^\ell - 1$, and
- the matrix product (p, q, r), which is the product of matrices $p \times q$ by $q \times r$.

Example 1.4 (Short product of polynomials modulo X^3)**.** *We describe in this example the matrices associated to the coefficients of the short product of two polynomials modulo X^3.*

Let A and B be the polynomials $A = a_0 + a_1 X + a_2 X^2$ and $B = b_0 + b_1 X + b_2 X^2$. We

denote by C the polynomial $A \cdot B \bmod X^3$:

$$C = a_0b_0 + (a_0b_1 + a_1b_0)X + (a_0b_2 + a_1b_1 + a_2b_0)X^2.$$

We represent A and B as vectors of K^3 denoted by $\mathbf{a} = \begin{pmatrix} a_0 \\ a_1 \\ b_2 \end{pmatrix}$ and $\mathbf{b} = \begin{pmatrix} b_0 \\ b_1 \\ b_2 \end{pmatrix}$, respectively.

Let Φ_0, Φ_1 and Φ_2 be the bilinear forms defined as

$$\begin{aligned} \Phi_0 &: (\mathbf{a}, \mathbf{b}) \mapsto a_0b_0, \\ \Phi_1 &: (\mathbf{a}, \mathbf{b}) \mapsto a_0b_1 + a_1b_0, \\ \Phi_2 &: (\mathbf{a}, \mathbf{b}) \mapsto a_0b_2 + a_1b_1 + a_2b_0. \end{aligned}$$

With respect to the canonical basis $(e_i)_{0 \le i < 3}$ of K^3, the matrices M_k, representations of the Φ_k's, are

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

1.1.3 Multiplicative complexity and bilinear rank

We are interested in the intrinsic time required for an algorithm, whose input is

$$\mathbf{a} = \begin{pmatrix} a_0 \\ \vdots \\ a_{m-1} \end{pmatrix} \in K^m \text{ and } \mathbf{b} = \begin{pmatrix} b_0 \\ \vdots \\ b_{n-1} \end{pmatrix} \in K^n,$$

to compute $\mathbf{\Phi}(\mathbf{a}, \mathbf{b})$.

We consider straight-line arithmetic programs, which consist in a sequence of instructions $f_i \leftarrow u_i \diamond v_i$, for $i \in \{0, \ldots, s\}$, where $\diamond$ is an arithmetic operation among $\{+, -, \times, \div\}$ and where u_i and v_i either belong to $K \cup \{\text{indeterminates}\}$ or are previously computed f_j's (for $j < i$). A straight-line arithmetic program is said to compute a subset $\mathcal{S} \subset \mathcal{L}(K^m, K^n; K)$ if for any $\Phi \in \mathcal{S}$ there exists $f_i \in \{f_0, \ldots, f_s\}$ such that f_i can be expressed as the evaluation of Φ in the indeterminates that are used in the straight-line program.

In the straight-line arithmetic programs, the indeterminates are representing the linear forms $\mathbf{a} \mapsto a_i$ and $\mathbf{b} \mapsto b_i$.

Example 1.5 (Multiplication of linear polynomials)**.** *Let $A = a_0 + a_1X$ and $B = b_0 + b_1X$ be two polynomials over K. The product $A \cdot B$ is associated to the bilinear map $\mathbf{\Phi}$ taking as*

input the vectors $\mathbf{a} = \begin{pmatrix} a_0 \\ a_1 \end{pmatrix}$ *and* $\mathbf{b} = \begin{pmatrix} b_0 \\ b_1 \end{pmatrix}$ *such that*

$$\mathbf{\Phi} : (\mathbf{a}, \mathbf{b}) \mapsto \begin{pmatrix} a_0 b_0 \\ a_0 b_1 + a_1 b_0 \\ a_1 b_1 \end{pmatrix}.$$

Thus, $\mathbf{\Phi}$ *can be seen as an element of* $\mathcal{L}(K^2, K^2; K)^3$*, whose coordinates are the* 3 *bilinear forms*

$$\begin{aligned} &\Phi_0 : (\mathbf{a}, \mathbf{b}) \mapsto a_0 b_0, \\ &\Phi_1 : (\mathbf{a}, \mathbf{b}) \mapsto a_0 b_1 + a_1 b_0, \text{ and} \\ &\Phi_2 : (\mathbf{a}, \mathbf{b}) \mapsto a_1 b_1. \end{aligned}$$

The straight-line arithmetic program formed by the sequence

$$\begin{aligned} f_0 &\leftarrow a_0 \times b_0 \\ f_1 &\leftarrow a_1 \times b_0 \\ f_2 &\leftarrow a_0 \times b_1 \\ f_3 &\leftarrow a_1 \times b_1 \\ f_4 &\leftarrow f_1 + f_2 \end{aligned}$$

computes the set $\{\Phi_0, \Phi_1, \Phi_2\}$.

One can prove (see e.g. [22]) that, for a straight-line program computing a set of bilinear forms, we can assume that each instruction $f_i \leftarrow u_i \diamond v_i$ is either

1. an addition or subtraction,
2. a scalar multiplication ($u_i \in K$ or $v_i \in K$) or
3. multiplication of two linear forms in $\mathbf{a}$ and $\mathbf{b}$ ($u_i \in \mathcal{L}(K^{m+n}; K)$ and $v_i \in \mathcal{L}(K^{m+n}; K)$).

We do not consider divisions: in [12, Ch. 7], it is shown that divisions do not help for the computation of a set of quadratic polynomials and, in particular, bilinear forms. It follows that we obtain, from the straight-line program computing a bilinear map $\mathbf{\Phi}$, a decomposition of the form

$$\mathbf{\Phi} = \sum_{0 \le i < t} \alpha_i(a_0, \ldots, a_{m-1}, b_0, \ldots, b_{n-1}) \beta_i(a_0, \ldots, a_{m-1}, b_0, \ldots, b_{n-1}) \mathbf{c}_i,$$

where α_i and β_i are linear forms of $\mathcal{L}(K^{m+n}; K)$ and $\mathbf{c}_i \in K^\ell$.

Definition 1.6 (Multiplicative complexity)**.** *The multiplicative complexity of a bilinear map* $\mathbf{\Phi} = (\Phi_0, \ldots, \Phi_{\ell-1})$ *is the smallest number of non-scalar multiplications involved in a straight-line program computing* $\{\Phi_0, \ldots, \Phi_{\ell-1}\}$*. We ignore scalar multiplications, additions and subtractions because their cost is considered as negligible. We denote this multiplicative complexity by* $L(\mathbf{\Phi})$.

Ideally, for a bilinear map $\mathbf{\Phi}$, we would like to compute exactly $L(\mathbf{\Phi})$. To simplify the combinatorics of this computation, and its computational complexity, we approximate this number with the bilinear rank, denoted by $\mathrm{rk}(\mathbf{\Phi})$.

Definition 1.7 (Bilinear rank of bilinear maps)**.** *The rank of a bilinear map $\mathbf{\Phi}$ is the smallest integer r such that there exist r rank-one bilinear forms $(\phi_i)_{0\leq i<r}$ and r vectors $\mathbf{c}_i \in K^\ell$ such that*

$$\mathbf{\Phi} = \sum_{0\leq i<r} \phi_i \cdot \mathbf{c}_i.$$

Thus, the bilinear rank is a number defined by the minimal size of a decomposition of the form

$$\mathbf{\Phi}(\mathbf{a},\mathbf{b}) = \sum_{0\leq i<t} \alpha_i(a_0,\ldots,a_{m-1})\beta_i(b_0,\ldots,b_{n-1})\mathbf{c}_i.$$

Definition 1.7 is an extension of Definition 1.2 to bilinear maps. When $\ell = 1$, the rank of a bilinear map $\mathbf{\Phi} = (\Phi_0)$ is the bilinear rank of the bilinear form Φ_0 as defined in Definition 1.2.

Proposition 1.8 (Bounds on the bilinear rank [12, Ch. 14])**.** *The bilinear rank of any bilinear map $\mathbf{\Phi}$ satisfies*

$$L(\mathbf{\Phi}) \leq \mathrm{rk}(\mathbf{\Phi}) \leq 2L(\mathbf{\Phi}).$$

Proof. Since the decomposition of a bilinear map $\mathbf{\Phi}$ in terms of rank-one bilinear forms is a special case of its decomposition in terms of products of linear forms $\alpha_i, \beta_i \in \mathcal{L}(K^{m+n}; K)$, we have

$$L(\mathbf{\Phi}) \leq \mathrm{rk}(\mathbf{\Phi}).$$

There exist $\alpha_i, \beta_i \in \mathcal{L}(K^{m+n}; K)$ and vectors $\mathbf{c}_i \in K^\ell$ such that

$$\mathbf{\Phi}(\mathbf{a},\mathbf{b}) = \sum_{0\leq i<L(\mathbf{\Phi})} \alpha_i(a_0,\ldots,a_{m-1},b_0,\ldots,b_{n-1})\beta_i(a_0,\ldots,a_{m-1},b_0,\ldots,b_{n-1})\mathbf{c}_i.$$

By linearity, for $i \in \{0,\ldots,L(\mathbf{\Phi})-1\}$,

$$\alpha_i(a_0,\ldots,a_{m-1},b_0,\ldots,b_{n-1}) = \alpha_i(a_0,\ldots,a_{m-1},0,\ldots,0) + \alpha_i(0,\ldots,0,b_0,\ldots,b_{n-1})$$

and

$$\beta_i(a_0,\ldots,a_{m-1},b_0,\ldots,b_{n-1}) = \beta_i(a_0,\ldots,a_{m-1},0,\ldots,0) + \beta_i(0,\ldots,0,b_0,\ldots,b_{n-1}).$$

Using the fact that $\mathbf{\Phi}$ is a bilinear map, the sum of the quadratic terms is zero. Thus, we get the following equality:

$$\begin{aligned}\mathbf{\Phi}(\mathbf{a},\mathbf{b}) = &\sum_{0\leq i<L(\mathbf{\Phi})} \alpha_i(a_0,\ldots,a_{m-1},0,\ldots,0)\beta_i(0,\ldots,0,b_0,\ldots,b_{n-1})\mathbf{c}_i \\ &+ \sum_{0\leq i<L(\mathbf{\Phi})} \alpha_i(0,\ldots,0,b_0,\ldots,b_{n-1})\beta_i(a_0,\ldots,a_{m-1},0,\ldots,0)\mathbf{c}_i.\end{aligned}$$

Thus, we have a bilinear decomposition of length $2L(\mathbf{\Phi})$, which implies that

$$\mathrm{rk}(\mathbf{\Phi}) \leq 2L(\mathbf{\Phi}).$$

□

Consequently, the bilinear rank is no larger than twice the multiplicative complexity of a bilinear map. In Example 1.9, we describe a bilinear map for which the multiplicative complexity and the bilinear rank are distinct. Moreover, the notion of bilinear rank extends a well known mathematical invariant, which is the rank of a matrix.

Example 1.9 (Difference between multiplicative complexity and bilinear complexity)**.** *The product of matrices* 2×2 *by* 2×3 *is an example of product for which the multiplicative complexity and the bilinear rank are different in a field of characteristic different from* 2*. Indeed, Waksman proposed in [60] formulas allowing one to compute the product with* 10 *multiplications, whereas the bilinear rank is proved to be* 11 *in [34]. For* $\mathbf{a} \in \mathcal{M}_{2,2}$ *and* $\mathbf{b} \in \mathcal{M}_{2,3}$*, let*

$$\mathbf{\Phi} : (\mathbf{a}, \mathbf{b}) \mapsto \begin{pmatrix} a_{0,0}b_{0,0} + a_{0,1}b_{1,0} \\ a_{0,0}b_{0,1} + a_{0,1}b_{1,1} \\ a_{0,0}b_{0,2} + a_{0,1}b_{1,2} \\ a_{1,0}b_{0,0} + a_{1,1}b_{1,0} \\ a_{1,0}b_{0,1} + a_{1,1}b_{1,1} \\ a_{1,0}b_{0,2} + a_{1,1}b_{1,2} \end{pmatrix}.$$

Let $\Phi_0, \ldots, \Phi_9$ *be*

$$\begin{array}{rclrcl} \Phi_0 &=& \frac{1}{2}(a_{0,0} + b_{1,0})(a_{0,1} + b_{0,0}), & \Phi_1 &=& \frac{1}{2}(a_{0,0} + b_{1,1})(a_{0,1} + b_{0,1}), \\ \Phi_2 &=& \frac{1}{2}(a_{0,0} + b_{1,2})(a_{0,1} + b_{0,2}), & \Phi_3 &=& (a_{1,0} + b_{1,0})(a_{1,1} + b_{0,0}), \\ \Phi_4 &=& \frac{1}{2}(a_{1,0} + b_{1,1})(a_{1,1} + b_{0,1}), & \Phi_5 &=& (a_{1,0} + b_{1,2})(a_{1,1} + b_{0,2}), \\ \Phi_6 &=& \frac{1}{2}(a_{0,0} - b_{1,0})(-a_{0,1} + b_{0,0}), & \Phi_7 &=& \frac{1}{2}(a_{0,0} - b_{1,1})(-a_{0,1} + b_{0,1}), \\ \Phi_8 &=& \frac{1}{2}(a_{0,0} - b_{1,2})(-a_{0,1} + b_{0,2}), & \Phi_9 &=& \frac{1}{2}(a_{1,0} - b_{1,1})(-a_{1,1} + b_{0,1}). \end{array}$$

We have

$$\mathbf{\Phi}(\mathbf{a}, \mathbf{b}) = \begin{pmatrix} \Phi_0 + \Phi_6 \\ \Phi_1 + \Phi_7 \\ \Phi_2 + \Phi_8 \\ -\Phi_0 + \Phi_1 + \Phi_3 - \Phi_4 + \Phi_6 - \Phi_7 + \Phi_9 \\ \Phi_4 + \Phi_9 \\ \Phi_1 - \Phi_2 - \Phi_4 + \Phi_5 - \Phi_7 + \Phi_8 + \Phi_9 \end{pmatrix}.$$

Example 1.10 (Karatsuba formulas for the multiplication of linear polynomials [37])**.** *The bilinear map* $\mathbf{\Phi}$ *for the multiplication of linear polynomials is an element of* $\mathcal{L}(K^2, K^2; K)^3$ *whose coordinates are the* 3 *bilinear forms*

$$\begin{aligned} &\Phi_0 : (\mathbf{a}, \mathbf{b}) \mapsto a_0 b_0, \\ &\Phi_1 : (\mathbf{a}, \mathbf{b}) \mapsto a_0 b_1 + a_1 b_0, \text{ and} \\ &\Phi_2 : (\mathbf{a}, \mathbf{b}) \mapsto a_1 b_1. \end{aligned}$$

Let $\Psi \in \mathcal{L}(K^2, K^2; K)$ be such that $\Psi : (\mathbf{a}, \mathbf{b}) \mapsto (a_0 + a_1)(b_0 + b_1)$. Then, since $\Phi_1 = \Psi - \Phi_0 - \Phi_2$, we can rewrite $\mathbf{\Phi}$ as

$$\mathbf{\Phi} = \begin{pmatrix} \Phi_0 \\ \Phi_1 \\ \Phi_2 \end{pmatrix} = \Phi_0 \cdot \begin{pmatrix} 1 \\ -1 \\ 0 \end{pmatrix} + \Psi \cdot \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} + \Phi_2 \cdot \begin{pmatrix} 0 \\ -1 \\ 1 \end{pmatrix}.$$

The bilinear forms Φ_0, Ψ and Φ_2 all have rank one. Thus, we can deduce that the bilinear rank of $\mathbf{\Phi}$ is at most 3. Actually, one can show that the bilinear rank of $\mathbf{\Phi}$ is equal to 3.

1.2 A linear algebra problem

The approach of [2] to the computation of the rank of a bilinear map $\mathbf{\Phi} = (\Phi_0, \ldots, \Phi_{\ell-1})$ formulates the problem in terms of vector spaces: the algorithm considers the subspace $T = \mathrm{Span}(\{\Phi_0, \ldots, \Phi_{\ell-1}\}) \subset \mathcal{L}(K^m, K^n; K)$ rather than the bilinear map $\mathbf{\Phi} = (\Phi_0, \ldots, \Phi_{\ell-1})$. There is no loss of generality to prefer the vector space formalism rather than the bilinear map formalism. In both cases, the computation of the rank of bilinear map [35] is NP-hard. However, in [2] the authors show that we have a lower combinatorial complexity using their formalism. Consequently, we choose this formalism as well, and we need to extend the definition of the rank to subspaces of $\mathcal{L}(K^m, K^n; K)$.

Notation 1.11. *For T a subspace of $\mathcal{L}(K^m, K^n; K)$, we denote by $\mathscr{S}_{m,n,r}(T)$ the set of subspaces $V \subset \mathcal{L}(K^m, K^n; K)$ spanned by a free family of rank-one bilinear forms of size r and such that $T \subset V$.*

When $T = \mathrm{Span}(\emptyset)$, $\mathscr{S}_{m,n,r}(T)$ is the set of subspaces $V \subset \mathcal{L}(K^m, K^n; K)$ spanned by a free family of rank-one bilinear forms of size r and we denote it simply by $\mathscr{S}_{m,n,r}$.

When m and n are clear from the context, these sets are simply denoted by $\mathscr{S}_r(T)$ and $\mathscr{S}_r$, respectively.

We use Notation 1.11 to define the rank of a subspace $T \in \mathcal{L}(K^m, K^n; K)$ in Definition 1.12.

Definition 1.12 (Bilinear rank of a subspace of $\mathcal{L}(K^m, K^n; K)$)**.** *Let T be a subspace of $\mathcal{L}(K^m, K^n; K)$. The bilinear rank of T, denoted by $\mathrm{rk}(T)$, is the smallest r such that $\mathscr{S}_r(T) \neq \emptyset$. The set $\mathscr{S}_{\mathrm{rk}(T)}(T)$ is called* the set of optimal decompositions *of T.*

If $T = \mathrm{Span}(\{\Phi_0, \ldots, \Phi_{\ell-1}\})$, each Φ_i can be represented by a matrix M_i in the canonical basis. Thus, if T has rank r, there exists a set of r matrices $N_i \in \mathcal{M}_{m,n}(K)$ of rank one such that

$$\forall i \in \{0, \ldots, \ell - 1\}, \ M_i \in \mathrm{Span}(N_0, \ldots, N_{r-1}),$$

which is the matrix equivalent of Definition 1.12.

Example 1.13 (Subspace for the short product of polynomials modulo X^3)**.** *We describe in this example the subspace of $\mathcal{L}(K^3, K^3; K)$ associated to the short product of two polynomials modulo X^3. We describe elements of $\mathcal{L}(K^3, K^3; K)$ as matrices of $\mathcal{M}_{3,3}$ in the canonical basis, as it has been done in Example 1.4.*

The bilinear map $\mathbf{\Phi}$ corresponding to the short product is defined as the application

$$\mathbf{\Phi} : \left(\begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix}, \begin{pmatrix} b_0 \\ b_1 \\ b_2 \end{pmatrix} \right) \mapsto \begin{pmatrix} a_0 b_0 \\ a_0 b_1 + a_1 b_0 \\ a_0 b_2 + a_1 b_1 + a_2 b_0 \end{pmatrix}.$$

Thus, the subspace

$$\operatorname{Span}\left(\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} \right)$$

is the matrix representation of the subspace T of $\mathcal{L}(K^3, K^3; K)$ corresponding to $\mathbf{\Phi}$. Our aim is to compute its rank in the sense of Definition 1.12.

Proposition 1.14. *The rank of a subspace $T \in \mathcal{L}(K^m, K^n; K)$ is bounded as follow*

$$\dim(T) \leq \operatorname{rk}(T) \leq m \cdot n.$$

Proof. The first inequality comes from the fact that a basis of rank-one bilinear forms generating a subspace containing T has necessarily a size no smaller than $\dim(T)$. The second inequality comes from the fact that the cardinality of a basis of a subspace of $\mathcal{L}(K^m, K^n; K)$ is bounded by the cardinality of the canonical basis $\{e_{i,j}\}_{i,j}$. □

We need to prove that finding all the optimal formulas in the sense of the bilinear rank for a bilinear map $\mathbf{\Phi} \in \mathcal{L}(K^m, K^n; K)^\ell$ is equivalent to computing $\mathscr{S}_r(T)$. This can be rephrased as the fact that Definition 1.7 and Definition 1.12 coincide, which is stated in the following proposition.

Proposition 1.15 (Rank of a bilinear map and rank of a matrix space)**.** *Let*

$$\mathbf{\Phi} = (\Phi_0, \ldots, \Phi_{\ell-1})$$

be a bilinear map and $T = \operatorname{Span}(\Phi_0, \ldots, \Phi_{\ell-1}) \subset \mathcal{L}(K^m, K^n; K)$. We have

$$\operatorname{rk}(\mathbf{\Phi}) = \operatorname{rk}(T).$$

Proof. If r bilinear forms ϕ_i of rank one span a vector space containing T, each coordinate of $\mathbf{\Phi}$ is a linear combination of these bilinear forms and, consequently, there exist r vectors $\mathbf{c}_i \in K^\ell$ such that

$$\mathbf{\Phi} = \sum_{0 \leq i < r} \phi_i \cdot \mathbf{c}_i.$$

Conversely, the bilinear forms of such a decomposition generate a vector space that contains T. □

1.3 RP-automorphisms

We adapt the automorphism group defined in [12, Def. 14.11] to the case of subspaces of $\mathcal{L}(K^m, K^n; K)$, which we introduced in Section 1.2. We describe in this section the rank-preserving group of automorphisms σ acting on subspaces $T \subset \mathcal{L}(K^m, K^n; K)$, also referred to as the RP-automorphisms group.

Definition 1.16. *An element $\sigma = (\mu, \nu) \in \mathrm{GL}(K^m) \times \mathrm{GL}(K^n)$ acts on $\mathcal{L}(K^m, K^n; K)$ via*

$$\Phi \circ \sigma : (\mathbf{a}, \mathbf{b}) \mapsto \Phi(\mu(\mathbf{a}), \nu(\mathbf{b})).$$

When $m = n$, we have the transposition τ acting on any $\Phi \in \mathcal{L}(K^m, K^m; K)$, via

$$\Phi \circ \tau : (\mathbf{a}, \mathbf{b}) \mapsto \Phi(\mathbf{b}, \mathbf{a}).$$

We denote by $\mathrm{RPA}_{m,n}$ the smallest group containing $\mathrm{GL}(K^m) \times \mathrm{GL}(K^n)$ and, if $m = n$, the transposition τ. Its elements are called RP-automorphisms.

Proposition 1.17. *The action of $\mathrm{RPA}_{m,n}$ is a group action and its elements are all invertible.*

Proof. For $\sigma = (\mu, \nu), \sigma' = (\mu', \nu') \in \mathrm{GL}(K^m) \times \mathrm{GL}(K^n)$ and $\Phi \in \mathcal{L}(K^m, K^n; K)$, we have

$$\forall \mathbf{a}, \mathbf{b},\ ((\Phi \circ \sigma) \circ \sigma')(\mathbf{a}, \mathbf{b}) = (\Phi \circ \sigma)(\mu'(\mathbf{a}), \nu'(\mathbf{b})) = \Phi(\mu(\mu'(\mathbf{a})), \nu(\nu'(\mathbf{b}))) = (\Phi \circ (\sigma \circ \sigma'))(\mathbf{a}, \mathbf{b}).$$

It works as well if σ or σ' is the element τ. Thus, the action that we defined is indeed a group action. Since all the elements of $\mathrm{RPA}_{m,n}$ are invertible, we have automorphisms. □

Proposition 1.18 (RP-automorphisms preserve the rank)**.** *Let $\sigma \in \mathrm{RPA}_{m,n}$.*

- *For any $\Phi \in \mathcal{L}(K^m, K^n; K)$, we have $\mathrm{rk}(\Phi \circ \sigma) = \mathrm{rk}(\Phi)$.*
- *For any subspace $T \subset \mathcal{L}(K^m, K^n; K)$, we also have $\mathrm{rk}(T \circ \sigma) = \mathrm{rk}(T)$.*

Proof. First, let $\phi \in \mathcal{L}(K^m, K^n; K)$ of rank one. There exist $\alpha \in \mathcal{L}(K^m; K)$ and $\beta \in \mathcal{L}(K^n; K)$ such that $\phi : (\mathbf{a}, \mathbf{b}) \mapsto \alpha(\mathbf{a}) \cdot \beta(\mathbf{b})$. There exist $\mu \in \mathrm{GL}(K^m)$ and $\nu \in \mathrm{GL}(K^n)$ such that $\phi \circ \sigma : (\mathbf{a}, \mathbf{b}) \mapsto \alpha(\mu(\mathbf{a})) \cdot \beta(\nu(\mathbf{b}))$. Since $\alpha \circ \mu \in \mathcal{L}(K^m; K)$ and $\beta \circ \nu \in \mathcal{L}(K^n; K)$, $\phi \circ \sigma$ is a rank-one bilinear form.

Since the RP-automorphisms in Definition 1.16 preserve the rank of rank-one bilinear forms, by linearity and by definition of the rank of a bilinear form, it preserves the rank of any bilinear form. For any subspace $T \subset \mathcal{L}(K^m, K^n; K)$ and any $\sigma \in \mathrm{GL}(K^m) \times \mathrm{GL}(K^n)$, we have

$$\mathrm{rk}(T \circ \sigma) = \mathrm{rk}(T).$$

□

Proposition 1.19 (Action of $\mathrm{RPA}_{m,n}$)**.** *The elements of $\mathrm{RPA}_{m,n}$ are the only automorphisms of $\mathcal{L}(K^m, K^n; K)$ preserving the rank.*

Proof. In both cases $m \neq n$ and $m = n$, a proof with the formalism of tensors of order 3 can be found in [13]. □

Remark 1.20. *The group* $\mathrm{GL}(K^m) \times \mathrm{GL}(K^n)$ *is isomorphic to the group* $\mathrm{GL}_m(K) \times \mathrm{GL}_n(K)$*, acting on matrices* M *via*

$$M \cdot (X, Y) = X^{\mathrm{T}} \cdot M \cdot Y,$$

for any $(X, Y) \in \mathrm{GL}(K^m) \times \mathrm{GL}(K^n)$.

Thus, we often consider elements of $\mathrm{GL}(K^m) \times \mathrm{GL}(K^n)$ *as elements of* $\mathrm{GL}_m(K) \times \mathrm{GL}_n(K)$ *and conversely.*

Definition 1.21 (RP-isomorphic spaces)**.** *Let* W *and* W' *be two subspaces of* $\mathcal{L}(K^m, K^n; K)$*. We say that* W *and* W' *are* RP*-isomorphic if there exists* $\sigma \in \mathrm{RPA}_{m,n}$ *such that* $W = W' \circ \sigma$*.*

Example 1.22 (Action of $\mathrm{RPA}_{2,2}$)**.** *Let us consider the subspace* $V \subset \mathcal{L}(K^2, K^2; K)$ *generated by the bilinear forms represented by the matrices* M_1 *and* M_2 *defined as*

$$M_1 = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \text{ and } M_2 = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}.$$

We take $\sigma = (X, Y)$ *such that* $X = Y = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$.

Then, the subspace $V' = V \circ \sigma$ *is generated by* M_1' *and* M_2'*, with*

$$M_1' = X^{\mathrm{T}} \cdot M_1 \cdot Y = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \text{ and } M_2' = X^{\mathrm{T}} \cdot M_2 \cdot Y = \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}.$$

Since we will often refer to subgroups of $\mathrm{RPA}_{m,n}$ stabilizing elements of $\mathcal{L}(K^m, K^n; K)$ in the following, we define the notion of set-wise stabilizer.

Definition 1.23 (Set-wise stabilizer)**.** *For a subset* $\mathcal{T} \subset \mathcal{L}(K^m, K^n; K)$*, we denote by* $\mathrm{Stab}(\mathcal{T})$ *the subgroup of* $\mathrm{RPA}_{m,n}$ *stabilizing* $\mathcal{T}$*:*

$$\mathrm{Stab}(\mathcal{T}) = \{\sigma \in \mathrm{RPA}_{m,n} \mid \mathcal{T} \circ \sigma = \mathcal{T}\}.$$

We use the same notation for a subspace $T \subset \mathcal{L}(K^m, K^n; K)$*.*

In the rest of this work, we often refer to the "stabilizer" of a given set $\mathcal{T}$. Each time, we exclusively mean the set-wise stabilizer of $\mathcal{T}$, which is, in general, different from the point-wise stabilizer of $\mathcal{T}$. Indeed, the point-wise stabilizer of $\mathcal{T}$ is defined as $\{\sigma \in \mathrm{RPA}_{m,n} \mid \forall \Phi \in \mathcal{T},\ \Phi \circ \sigma = \Phi\}$.

Example 1.24 (Stabilizer for the short product modulo X^3)**.** *We describe in this example an element of the stabilizer of the subspace* T *generated by the bilinear forms*

$$\Phi_0 = a_0 b_0,\ \Phi_1 = a_1 b_0 + a_0 b_1,\ \text{and } \Phi_2 = a_2 b_0 + a_1 b_1 + a_0 b_2,$$

which are represented by the matrices

$$M_0 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \; M_1 = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \text{ and } M_2 = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}.$$

Let (X, Y) *be the couple*

$$(X, Y) = \left(\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix} \right).$$

We have

$$M_2 \cdot (X, Y) = X^{\mathrm{T}} \cdot M_2 \cdot Y = M_0 + M_1 + M_2 \in T.$$

Similarly, $M_1 \cdot (X, Y) = M_0 + M_1 \in T$ *and* $M_0 \cdot (X, Y) = M_0 \in T$*. Thus,* $(X, Y) \in \mathrm{Stab}(T)$*.*

A full description of the stabilizer for this particular example can be found in Section 4.1.1.

In our algorithms, we often use subgroups of $\mathrm{RPA}_{m,n}$ in the following context:

- we have a set $\mathcal{S}$ of subspaces of $\mathcal{L}(K^m, K^n; K)$,
- we have a subgroup H of $\mathrm{RPA}_{m,n}$ stabilizing $\mathcal{S}$, which means that for any $\sigma \in H$ and $V \in \mathcal{S}$, $V \circ \sigma \in \mathcal{S}$, and
- we need to enumerate one representative per equivalence class given by the group relation.

Notation 1.25. *Let* $\mathcal{S}$ *be a set of subspaces of* $\mathcal{L}(K^m, K^n; K)$ *and* $H \subset \mathrm{RPA}_{m,n}$ *stabilizing* $\mathcal{S}$*. We denote by* $\mathcal{S}/H = \{\{V \circ \sigma \mid \sigma \in H\} \mid V \in \mathcal{S}\}$ *the set of equivalence classes, or orbits, under the relation associated to the action of* H *on* $\mathcal{S}$*.*

Chapter 2

A review of existing exhaustive search algorithms

The main obstacle to finding optimal formulas (for the bilinear rank) is the fact that the decomposition of bilinear maps is NP-hard [35]. In terms of methods, the least-squares method seems to be one of the most popular ones [54]. Another way to decompose a bilinear map consists in using ingredients from geometry [5] and in finding a generalization of the singular value decomposition for matrices to general tensors. However, these methods are essentially used over an algebraically closed field K (e.g. $K = \mathbb{C}$) and are not meant to produce all the possible decompositions for a bilinear map. In our context, we need a method for computing a rank decomposition over a finite field not necessarily algebraically closed.

Montgomery proposes in [45] an algorithm to compute such a decomposition for the product of polynomials of small degree over a finite field. The author enumerates all possible formulas of some form, taking advantage of the fact that, on a finite field, the number of possible optimal formulas is bounded. This allows him to find new formulas for the multiplication of polynomials of 5, 6 and 7 terms over $\mathbb{F}_2$. It introduces the formalism of vector spaces and proposes an algorithm solving the bilinear rank problem by looking for symmetric formulas. In [46], Oseledets proposes a heuristic approach to solve the bilinear rank problem for the polynomial product over $\mathbb{F}_2$ and considers other products, such as the short product and the circulant product. The idea of Montgomery and Oseledets is to use the vector space formalism to search exhaustively for all the optimal formulas for products over a finite field. Later, Barbulescu et al. proposed in [2] an improvement on the exhaustive search of optimal formulas for a bilinear map, in a framework using the vector space formalism. In this chapter, we describe these algorithms, which allow one to prove lower bounds on the bilinear rank of bilinear maps.

2.1 Mongtomery's Algorithm

Montgomery proposes in [45] a first attempt at finding new formulas for the product of polynomials A and B of n-terms over a finite field K. We denote by $\mathbf{a}$ and $\mathbf{b}$ the vectors of K^n representing the polynomials A and B in the canonical basis.

Henceforth, we describe the algorithm for the special case $K = \mathbb{F}_2$. The ideas that are described can be adapted to finite fields of larger characteristic, although the computational cost make them not practical in this case.

The algorithm starts with the set of all the rank-one bilinear forms of $\mathcal{L}(K^n, K^n; K)$, which

has $(2^n-1)^2$ elements. The product of polynomials of n-terms is represented by the bilinear map $\mathbf{\Phi}_n \in \mathcal{L}(K^n, K^n; K^{2n-1})$ with

$$\mathbf{\Phi}_n : (\mathbf{a}, \mathbf{b}) \mapsto \begin{pmatrix} a_0b_0 \\ a_1b_0 + a_0b_1 \\ \vdots \\ a_{n-2}b_0 + \cdots + a_0b_{n-2} \\ a_{n-1}b_0 + \cdots + a_0b_{n-1} \\ a_{n-1}b_1 + \cdots + a_1b_{n-1} \\ \vdots \\ a_{n-1}b_{n-1} \end{pmatrix}.$$

We denote by T_n the vector space spanned by the coefficients of $\mathbf{\Phi}_n$.

Example 2.1 (Subspace representing $\mathbf{\Phi}_3$)**.** *The subspace T_3 corresponding to $\mathbf{\Phi}_3$ is*

$$T_3 = \text{Span}\left(\begin{pmatrix}1&0&0\\0&0&0\\0&0&0\end{pmatrix}, \begin{pmatrix}0&1&0\\1&0&0\\0&0&0\end{pmatrix}, \begin{pmatrix}0&0&1\\0&1&0\\1&0&0\end{pmatrix}, \begin{pmatrix}0&0&0\\0&0&1\\0&1&0\end{pmatrix}, \begin{pmatrix}0&0&0\\0&0&0\\0&0&1\end{pmatrix}\right).$$

Exhaustive search of formulas for the product of polynomials. Taking the formalism proposed in Chapter 1 into account, the general algorithmic strategy Montgomery proposes to compute the bilinear rank of the bilinear map representing a polynomial product of degree $n-1$ is stated as follows:

1. compute the set of rank-one matrices of $\mathcal{M}_n$, of cardinality $(2^n-1)^2$;
2. start with the known lower bound $r = 2n-1$ on the bilinear rank;
3. compute all the $\binom{(2^n-1)^2}{r}$ sets of r rank-one matrices;
4. select the sets generating a vector space containing T_n;
5. if no such set is found at the previous step, increment r and return to Step 2;
6. r is the bilinear rank and the selected sets produce optimal formulas for the polynomial product.

The strategy described previously actually produces all the optimal formulas for the bilinear decomposition of $\mathbf{\Phi}_n$. For $n = 5$, we are led to increase r until 13. There are $2.1 \cdot 10^8$ ways to select 13 rank-one matrices among $(2^n-1)^2$ rank-one bilinear forms. For $n = 6$ or $n = 7$, we have roughly $1.0 \cdot 10^{15}$ and $2.5 \cdot 10^{24}$ possibilities, which is too large to be enumerated.

Thus, Montgomery adapts this algorithm to find at least one set of optimal formulas for the product of n-terms polynomials. He proposes three constraints on the set $\mathcal{S}$ of bilinear forms of rank one that are selected:

1. for each bilinear form $\Phi \in \mathcal{S}$, the corresponding matrix M in the canonical basis should be symmetric: $M = M^{\mathrm{T}}$;

2. $\mathcal{S}$ contains the bilinear forms given by the coefficients a_0b_0, $a_{n-1}b_{n-1}$ and $(a_0 + \cdots + a_{n-1})(b_0 + \cdots + b_{n-1})$;

3. $\mathcal{S}$ should satisfy $\mathcal{S} \circ \sigma = \mathcal{S}$ where $\sigma \in \mathrm{GL}(K^n) \times \mathrm{GL}(K^n)$ is represented by the pair (P, P) with $P \in \mathrm{GL}_n(\mathbb{F}_2)$ the matrix

$$\begin{bmatrix} 0 & \cdots & 0 & 1 \\ \vdots & \iddots & \iddots & 0 \\ 0 & \iddots & \iddots & \vdots \\ 1 & 0 & \cdots & 0 \end{bmatrix}$$

because the vector space T also satisfies $T \circ \sigma = T$.

For $n = 6$, the lower bound on the size of a decomposition with the given constraints is 17. The first and the second constraints lead to sets $\mathcal{S}$ containing the 3 bilinear forms given by the coefficients a_0b_0, a_5b_5 and $(a_0 + \cdots + a_6)(b_0 + \cdots + b_6)$ and 14 bilinear forms among the $(2^6 - 1) - 3 = 60$ remaining rank-one bilinear forms represented by a symmetric matrix. Among the 60 matrices, 6 are invariant under the action of σ and 54 are not, which gives 27 pairs of bilinear forms (ϕ, ψ) such that $\phi = \psi \circ \sigma$. Thus, for i an even integer in $\{0, \ldots, 6\}$, where 6 is the number of invariant matrices under the action of σ, the third constraint splits the 14 bilinear forms required in $\mathcal{S}$ into i bilinear forms invariant under the action of σ and $(14 - i)/2$ pairs of bilinear forms (ϕ, ψ) such that $\phi = \psi \circ \sigma$. The search is clearly not exhaustive.

We have to enumerate

$$\sum_{\substack{0 \le i \le 6, \\ i \text{ even}}} \binom{6}{i} \cdot \binom{27}{(17 - 3 - i)/2} = 6.6 \cdot 10^6$$

possible sets.

2.2 Exhaustive search for any bilinear map

We describe in this section how to generalize the previous approach. In [46], Oseledets develops the idea of Montgomery and generalizes it to any kind of bilinear map. The naive algorithms that are described in this section can be considered as a reformulation of Oseledets' exhaustive search. However, the author proposes also heuristics to decompose certain bilinear maps given by the product of polynomials modulo X^n (short product) and the product modulo $X^n - 1$ (circulant product).

We work over a finite field K. We consider a bilinear map $\mathbf{\Phi} \in \mathcal{L}(K^m, K^n; K^\ell)$, represented by the subspace $T \subset \mathcal{L}(K^m, K^n; K)$ of dimension ℓ.

General strategy for computing the bilinear rank. The algorithmic strategy we use to compute the bilinear rank of a bilinear map is stated as follows:

- start with the known lower bound $r = \ell$ on the bilinear rank;
- compute $\mathscr{S}_r(T)$;
- if $\mathscr{S}_r(T) = \emptyset$, increment r and return to the previous step;

- r is the bilinear rank and $\mathscr{S}_r(T)$ the set of optimal decompositions.

The general strategy requires an algorithm to compute $\mathscr{S}_r(T)$. The algorithms that are considered require a test to determine whether, for V a subspace of $\mathcal{L}(K^m, K^n; K)$ of dimension r, we have $V \in \mathscr{S}_r$: we denote this test by HasRankOneBasis. A naive method to perform this test is described in Algorithm 1. We could think of other methods based on solving bilinear systems, but it does not seem efficient in our applications. However, an optimized version of this algorithm is used for particular bilinear maps (product of 2×3 by 3×2 matrices for example described in Section 4.3.3).

This method assumes that we have computed the the set $\mathcal{G}_{m,n} \simeq \mathscr{S}_{m,n,1}$ of rank-one bilinear forms of $\mathcal{L}(K^m, K^n; K)$ up to a multiplicative factor. This set has cardinality

$$c_{m,n} = \#\mathcal{G}_{m,n} = \frac{(\#K^m - 1)(\#K^n - 1)}{(\#K - 1)^2}.$$

Algorithm 1 HasRankOneBasis (naive method)

Input: A subspace $V \subset \mathcal{L}(K^m, K^n; K)$, $\mathcal{G}_{m,n}$
Output: Boolean indicating whether $V \in \mathscr{S}_{\dim(V)}$
1: $\mathcal{H} \leftarrow \mathcal{G}_{m,n} \cap V$ ▷ $\mathcal{G}_{m,n}$ is the set of rank-one bilinear forms of $\mathcal{L}(K^m, K^n; K)$
2: **if** $\dim(\mathrm{Span}(\mathcal{H})) = \dim(V)$ **then**
3: **return true**
4: **else**
5: **return false**
6: **end if**

The naive version of HasRankOneBasis computes the set of rank-one bilinear forms of a vector space V and computes the dimension of the vector space generated by this space. Consequently, the complexity is given by $c_{m,n}$ membership tests to V (which can be realized as a Gauss reduction of a bilinear form of $\mathcal{G}_{m,n}$ by a matrix given by a basis of V). Generically, the cost of Algorithm HasRankOneBasis is thus $O(mn \cdot \dim(V) \cdot c_{m,n})$.

2.2.1 Naive algorithms

The first method that comes to mind is to compute the $\binom{c_{m,n}}{r}$ possible sets of r bilinear forms and to test whether T is contained in their span. Algorithm 2 is a proposed formalisation of this algorithm.

Algorithm 2 IterativeExhaustiveSearch

Input: $T \subset \mathcal{L}(K^m, K^n; K)$ of dimension ℓ, $\mathcal{G}_{m,n}$, an integer r
Output: $\mathscr{S}_r(T)$
$\mathcal{S} \leftarrow \emptyset$
for $\mathcal{V} \subset \mathcal{G}_{m,n}$ such that $\#\mathcal{V} = r$ **do**
if $\dim(\mathrm{Span}(\mathcal{V})) = r$ and $T \subset \mathrm{Span}(\mathcal{V})$ **then**
$\mathcal{S} \leftarrow \mathcal{S} \cup \{\mathrm{Span}(\mathcal{V})\}$
end if
end for
return $\mathcal{S}$

Algorithm 3 RecursiveExhaustiveSearch

Input: $T \subset \mathcal{L}(K^m, K^n; K)$ of dimension ℓ, $\mathcal{G}_{m,n}$, an integer r
Output: $\mathscr{S}_r(T)$

```
1: function ExpandSubspace(V, H, r)
2:     if dim V = r and T ⊂ V then
3:         return {V}
4:     else
5:         S ← ∅
6:         for i ∈ {0, ..., #H − 1} do                    ▷ H = {φ_i | i ∈ [0, #H − 1]}
7:             H' ← {φ_{i+1}, ..., φ_{#H−1}}
8:             S ← S ∪ ExpandSubspace(V ⊕ Span(φ_i), H', r)
9:         end for
10:        return S
11:    end if
12: end function
13: return ExpandSubspace(Span(∅), G_{m,n}, 0, r)
```

Algorithm IterativeExhaustiveSearch and RecursiveExhaustiveSearch can be considered as respectively the iterative and the recursive version of the same strategy. The recursive calls in RecursiveExhaustiveSearch can be represented with a tree in which each node at depth k corresponds to a vector space $V_{i_0,i_1,\ldots,i_{k-1}}$ of dimension k generated by a basis of rank-one bilinear forms $\phi_{i_0}, \phi_{i_1}, \ldots, \phi_{i_{k-1}}$. For example, assuming that the initial set of rank-one bilinear forms is $\{\phi_0, \phi_1, \phi_2, \phi_3\}$, we would obtain generically for $r = 3$ the tree given in Figure 2.2.

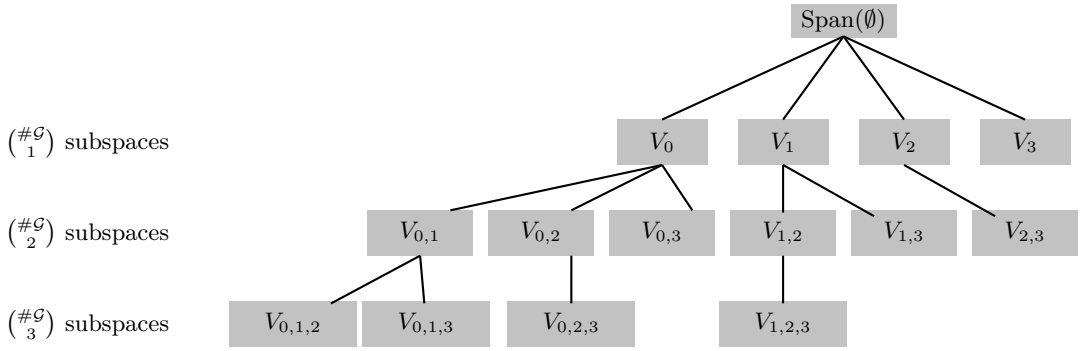

Figure 2.1: Subspaces V visited by recursive calls of Algorithm 3 up to $k = 3$

The cost of IterativeExhaustiveSearch and RecursiveExhaustiveSearch is smaller than the cost of $\binom{c_{m,n}}{r}$ inclusion tests. The inclusion test $T \subset V$ is computed with a basis of T formed by ℓ vectors of K^{m+n}, whose elements are reduced via Gauss reduction by a matrix of $\mathcal{M}_{r,m+n}$ representing a basis of V. Thus, the whole cost is smaller than the cost of $\ell\binom{c_{m,n}}{r}$ Gauss reductions of a vector of K^{m+n} by a matrix of $\mathcal{M}_{r,m+n}$.

Algorithm RecursiveExhaustiveSearch can be improved by taking into account the fact that, in Figure 2.1, we may have $V_{0,1} = V_{0,2}$. In this scenario, the recursive call corresponding to the node $V_{0,3}$ is facultative. Algorithm ImprovedRecursiveExhaustiveSearch takes into account this observation.

Algorithm 4 ImprovedRecursiveExhaustiveSearch

Input: $T \subset \mathcal{L}(K^m, K^n; K)$ of dimension ℓ, $\mathcal{G}_{m,n}$, an integer r
Output: $\mathscr{S}_r(T)$

```
1: function ExpandSubspace(V, H, d, r)
2:     if d = r and dim V = r and T ⊂ V then
3:         return {V}
4:     else
5:         S ← ∅
6:         for i ∈ {0, ..., #H − 1} do                    ▷ H = {φ_i | i ∈ [0, #H − 1]}
7:             H' ← {φ_{i+1}, ..., φ_{#H−1}} mod φ_i      ▷ Gauss reduction modulo φ_i
8:             S ← S ∪ ExpandSubspace(V ⊕ Span(φ_i), H', d + 1, r)
9:         end for
10:        return S
11:    end if
12: end function
13: return ExpandSubspace(Span(∅), G_{m,n}, 0, r)
```

The refinement is the Gauss reduction on Line 7. The set $\mathcal{H}'$ is a subset of $\mathcal{H}$ of vectors that form a set of representatives for the classes given by the Gauss reduction modulo ϕ_i. More specifically, the elements of $\mathcal{H}$ are represented as vectors of K^{m+n} and we reduce these vectors by a vector ϕ_i, seen as a matrix of $\mathcal{M}_{1,m+n}$.

Example 2.2 (Gauss reduction)**.** *Taking $m = n = 2$, we show in this example how the bilinear forms ϕ_0, ϕ_1, ϕ_2 and ϕ_3, seen as the elements of $\mathbb{F}_2^{2+2} = \mathbb{F}_2^4$*

$$(1,0,0,0),\ (1,1,0,0),\ (1,1,0,1) \text{ and } (0,1,1,0),$$

respectively, are reduced row-wise by the element represented by the 1×4 matrix

$$\begin{pmatrix} 1 & 0 & 1 & 0 \end{pmatrix}.$$

In this scenario we obtain from the set $\mathcal{H} = \{\phi_0, \ldots, \phi_3\}$ the set

$$\mathcal{H}' = \{(0,0,1,0), (0,1,1,0), (0,1,1,1)\}.$$

In the presented version, the cardinality of $\mathcal{H}$ decreases at each call due to this modulo relation.

The algorithms BDEZ and BDEZStab that follow use the same strategy as ImprovedRecursiveExhaustiveSearch. Their cost is different because the test on Line 2 is not the same (it uses HasRankOneBasis) and the initial call to ExpandSubspace takes different parameters.

2.2.2 Oseledets' heuristics

For certain bilinear maps, Oseledets proposes to use the presence of some rank-one bilinear forms in the initial vector space T, during the exhaustive search process. Indeed, he observes that for a vector space V such that $V \in \mathscr{S}_r(T)$, if U is a subspace of T generated by rank-one bilinear forms, then we can construct V via a partial exhaustive search. Montgomery already considers with a similar idea the case of the polynomial product in [45]. Indeed, he is looking for sets of rank-one bilinear forms containing

- a_0b_0, which is the coefficient of $[X^0]$ in the polynomial product $A(X) \cdot B(X)$,
- $a_{n-1}b_{n-1}$, which is the coefficient of $[X^{2n-2}]$ and
- $(a_0 + \cdots + a_{n-1})(b_0 + \cdots + b_{n-1})$, which is the sum of the coefficients of the $[X^i]$'s.

The idea is the following, assuming that the estimated rank of a subspace T is r, as in the general strategy:

- find the rank-one bilinear forms in T and denote their span by U;
- enumerate all subspaces $W \in \mathscr{S}_{r-\dim(U)}$ and test whether $U \oplus W$ contains T.

Proposition 2.3. *Let T be a subspace of dimension ℓ of $\mathcal{L}(K^m, K^n; K)$, and let $r \geq \ell$ be an integer. Let $U \subset T$ be the subspace containing all the rank-one bilinear forms of T. For any $V \in \mathscr{S}_r(T)$, there exists $W \in \mathscr{S}_{r-\dim(U)}$ such that $T \subset U \oplus W$.*

Proof. Let $V \in \mathscr{S}_r(T)$. Then $U \subset V$. Let $\mathcal{B}$ be a basis of V of rank-one bilinear forms. Let $\mathcal{U}$ be a basis of U of rank-one bilinear forms. We can complete $\mathcal{U}$ with elements of $\mathcal{B}$ and obtain a new basis of V containing $\mathcal{U}$. The elements added to $\mathcal{U}$ form a free family generating a subspace $W \in \mathscr{S}_{r-\dim(U)}$. □

Moreover, Oseledets proposes a heuristic, in the case where the exhaustive search is too expensive. The idea is to take a bilinear form Φ of T, find rank-one bilinear forms generating Φ, and add these rank-one bilinear forms to the bases enumerated during the search, which is not exhaustive anymore.

Example 2.4. *For the product of polynomial of degree 2, the bilinear form represented by*

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

is in the target T. It can be decomposed as a sum of the matrices

$$\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \text{ and } \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}.$$

The search then consists in finding complements of

$$\text{Span}\left(\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}\right).$$

Of course, this heuristic may lead to an overestimation of the rank.

2.2.3 The BDEZ Algorithm (Barbulescu, Detrey, Estibals, Zimmermann)

We describe in this section the algorithm [2, Alg. 1], which is an improvement of the recursive algorithm RecursiveExhaustiveSearch. It computes $\mathscr{S}_r(T)$ for a given subspace $T \subset \mathcal{L}(K^m, K^n; K)$ of dimension ℓ.

In order to get all the vector spaces $V \in \mathscr{S}_r$ such that $T \subset V$, we compute the vector spaces $W \in \mathscr{S}_{r-\ell}$ such that $T \oplus W \in \mathscr{S}_r$. In other terms, instead of enumerating all the elements of $\mathscr{S}_r$, we rather enumerate complementary subspaces of T in $\mathscr{S}_{r-\ell}$. This restriction can be done thanks to Proposition [2, Prop. 1], reformulated as Proposition 2.5 using the notations of Chapter 1.

Proposition 2.5. *Let T be a subspace of dimension ℓ of $\mathcal{L}(K^m, K^n; K)$, and let $r \geq \ell$ be an integer. For any $V \in \mathscr{S}_r(T)$, there exists $W \in \mathscr{S}_{r-\ell}$ such that $T \oplus W = V$.*

Proof. Let $\mathcal{B}$ be a basis of V composed of rank-one bilinear forms. We define inductively a sequence of subspaces $(W_i)_{0 \leq i \leq r-\ell}$, such that for any i we have $W_i \in \mathscr{S}_i$, as follows.

- The set W_0 is the null subspace and satisfies $T \oplus W_0 = T \subset V$ and $\dim T \oplus W_0 = \ell$.
- For $i \in \{1, \ldots, r-\ell\}$, assuming that $T \oplus W_{i-1} \subset V$ and $\dim(T \oplus W_{i-1}) = \ell + i - 1$, there exists $\phi \in \mathcal{B}$ such that $\phi \notin T \oplus W_{i-1}$ (otherwise $T \oplus W_{i-1} = V$ and $\dim V \leq r-1$, which is a contradiction). Then, we define W_i as $W_i = W_{i-1} \oplus \mathrm{Span}(\phi)$. The subspace W_i satisfies $T \oplus W_i \subset V$, $\dim(T \oplus W_i) = \ell + i$ and $W_i \in \mathscr{S}_i$.

Taking $W = W_{r-\ell}$, Proposition 2.5 is proved. □

Algorithm BDEZ can be described as a a recursive optimized version of RecursiveExhaustiveSearch. It constructs all the sets of cardinality $r - \ell$ of independent bilinear forms of rank one. The input of BDEZ is: a target T of dimension ℓ, the set of rank-one bilinear forms up to a multiplicative factor and an integer r (r is a lower bound on the rank of T, as explained in the global strategy).

Algorithm 5 BDEZ

Input: $T \subset \mathcal{L}(K^m, K^n; K)$ of dimension ℓ, $\mathcal{G}_{m,n}$, an integer r
Output: $\mathscr{S}_r(T)$

```
function ExpandSubspace(V, H, d, r)
    if d = r and dim V = r and HasRankOneBasis(V) then
        return {V}
    else
        S ← ∅
        for i ∈ {0, ..., #H − 1} do                    ▷ H = {φ_i | i ∈ [0, #H − 1]}
            H' ← {φ_{i+1}, ..., φ_{#H−1}} mod φ_i      ▷ Gauss reduction modulo φ_i
            S ← S ∪ ExpandSubspace(V ⊕ Span(φ_i), H', d + 1, r)
        end for
        return S
    end if
end function
return ExpandSubspace(T, G mod T, ℓ, r)    ▷ Gauss reduction of G modulo a basis of T
```

This algorithm is represented by a tree in which each node at depth k corresponds to a vector space $T \oplus W_{i_0,i_1,\ldots,i_{k-1}}$ of dimension k generated by a basis of T and rank-one bilinear forms $\phi_{i_0}, \phi_{i_1}, \ldots, \phi_{i_{k-1}}$. For example, assuming that the initial set of rank-one bilinear forms is $\{\phi_0, \phi_1, \phi_2, \phi_3\}$ and ignoring the reduction computed on Line 7, we would obtain generically for $r - \ell = 3$ the tree given in Figure 2.2.

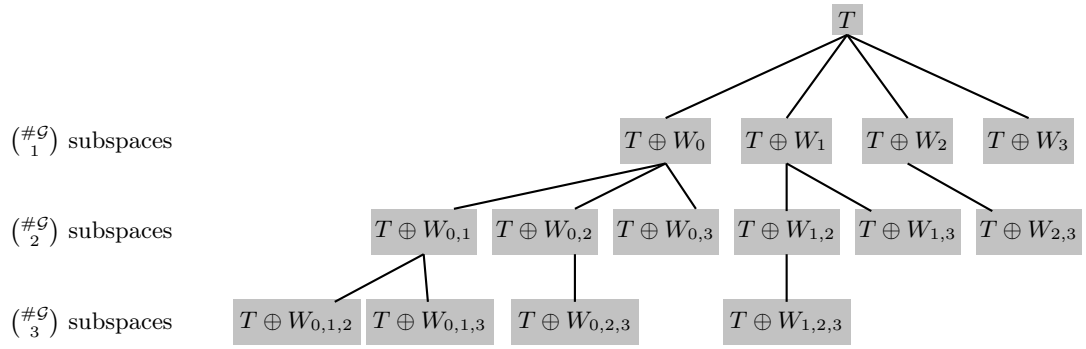

Figure 2.2: Subspaces V visited by recursive calls of Algorithm 5 up to $k = 3$

2.2.4 Improving on BDEZ using RP-automorphisms

We present in this section, with kind permission from the authors, an unpublished improvement [3] to Algorithm BDEZ. We use the RP-automorphisms defined in Chapter 1.

The algorithmic improvement comes from the fact that, for any target space $T \subset \mathcal{L}(K^m, K^n; K)$ of dimension ℓ and any integer $r \geq \ell$, we have

$$\forall \sigma \in \mathrm{Stab}(T),\ \mathscr{S}_r(T) \circ \sigma = \mathscr{S}_r(T),$$

because σ preserves the rank (see Proposition 1.18). Thus, we can restrict our interest to the computation of the quotient $\mathscr{S}_r(T)/\mathrm{Stab}(T)$ instead of $\mathscr{S}_r(T)$.

In order to enumerate the elements of $\mathscr{S}_r(T)$, it is sufficient to obtain a set of representatives of $\mathscr{S}_r(T)/\mathrm{Stab}(T)$, from which one can recover the whole orbits through the group action of $\mathrm{Stab}(T)$. Moreover, we can compute $\mathscr{S}_r(T)/\mathrm{Stab}(T)$ faster than $\mathscr{S}_r(T)$. Thus, we adapt our general strategy to this idea.

General strategy for computing the bilinear rank using RP-automorphisms The new algorithmic strategy we are considering is stated as follows, for a target subspace $T \subset \mathcal{L}(K^m, K^n; K)$ of dimension ℓ and the associated subgroup $\mathrm{Stab}(T)$ of RP-automorphisms stabilizing T:

- start with the known lower bound $r = \ell$ on the bilinear rank;
- compute a set of representatives of $\mathscr{S}_r(T)/\mathrm{Stab}(T)$ (the set $\mathscr{S}_r(T)$ up to the action of $\mathrm{Stab}(T)$);
- if $\mathscr{S}_r(T)/\mathrm{Stab}(T) = \emptyset$, increment r and return to the previous step;
- recover $\mathscr{S}_r(T)$ using the action of $\mathrm{Stab}(T)$;
- r is the bilinear rank and $\mathscr{S}_r(T)$ the set of optimal decompositions.

Algorithm BDEZStab is a recursive approach for the computation of a set of representatives of $\mathscr{S}_r(T)/\mathrm{Stab}(T)$. The input of BDEZStab is: a target subspace T of dimension ℓ, the set of rank-one bilinear forms of $\mathcal{L}(K^m, K^n; K)$ up to a multiplicative factor, the group $\mathrm{Stab}(T)$ and an integer $r \geq \ell$.

Algorithm 6 BDEZStab

Input: $T \subset \mathcal{L}(K^m, K^n; K)$, the stabilizer $\mathrm{Stab}(T)$, an integer r
Output: A set of representatives of $\mathscr{S}_r(T)/\mathrm{Stab}(T)$

1: **function** ExpandSubspace$(V, \mathcal{H}, U, d, r)$ ▷ $V \subset \mathcal{L}(K^m, K^n; K), \mathcal{H} \subset \mathcal{G}, U \subset \mathrm{Stab}(T), r \in \mathbb{N}$
2: **if** $d = r$ and $\dim V = r$ and HasRankOneBasis(V) **then**
3: **return** $\{V\}$
4: **else**
5: $\mathcal{S} \leftarrow \emptyset$
6: $\mathcal{O} \leftarrow \mathcal{H}/_U$ ▷ ϕ and ϕ' lie in the same orbit if $V \oplus \mathrm{Span}(\phi) = (V \oplus \mathrm{Span}(\phi')) \circ \sigma$
7: **for** $i \in \{0, \ldots, \#\mathcal{O} - 1\}$ **do** ▷ $\mathcal{O} = \{O_i \mid i \in \{0, \ldots, \#\mathcal{O} - 1\}\}$
8: $\phi \leftarrow$ Representative(O_i) ▷ Choose a representative of the orbit O_i
9: $U' \leftarrow \mathrm{Stab}\,(V \oplus \mathrm{Span}(\{\phi\})) \cap U$
10: $\mathcal{H}' \leftarrow \cup_{j \geq i} O_j$
11: $\mathcal{S} \leftarrow \mathcal{S} \cup$ ExpandSubspace$(V, \mathcal{H}', U', d + 1, r)$
12: **end for**
13: **return** $\mathcal{S}$
14: **end if**
15: **end function**
16: **return** ExpandSubspace$(T, \mathcal{G}, \mathrm{Stab}(T), \ell, r)$

On Line 6, we split the set $\mathcal{H}$ into orbits and we have a basis of V forming a $\dim(V) \times mn$ matrix M. We proceed as follow:

1. if $\mathcal{H}$ is not empty, pick an element ϕ of $\mathcal{H}$,
2. compute its orbit O under the action of U and reduce its elements by M,
3. remove the elements ψ of $\mathcal{H}$ such that their reduction by M is in O,
4. add to a list $\mathcal{O}$ the orbit O and return to Step 1.

Figure 2.3 describes this recursive approach using a tree and illustrates how some branches are pruned, relying on Proposition 2.6. We assume that the initial set of rank-one bilinear forms is $\{\phi_0, \phi_1, \phi_2, \phi_3\}$ and that we have $\sigma \in \mathrm{Stab}(T)$ such that $\sigma(\phi_0) = \phi_1$, $\sigma(\phi_1) = \phi_0$, $\sigma(\phi_2) = \phi_3$ and $\sigma(\phi_3) = \phi_2$.

Proposition 2.6. *Let T and V be subspaces of $\mathcal{L}(K^m, K^n; K)$ such that $V \in \mathscr{S}_r(T)$. Then, given a bilinear form ϕ of rank one, if V satisfies $V \cap (\{\phi\} \circ \mathrm{Stab}(T)) \neq \emptyset$, there exists an element V' in the equivalence class of V for the action of $\mathrm{Stab}(T)$ and such that $\phi \in V'$.*

Proof. There exists $\sigma \in \mathrm{Stab}(T)$ such that $\phi \circ \sigma \in V$. We can then take $V' = V \circ (\sigma^{-1})$, which meets all the conditions. □

The particularity of BDEZStab is that, instead of enumerating all the elements of $\mathcal{H}$ as in BDEZ, we restrict ourselves to one element per equivalence class for the action of $U \subset \mathrm{Stab}(V)$.

We use in particular the fact that the additional computations such as stabilizers on Line 9 are negligible, compared to the speed-up obtained by pruning branches in BDEZ. Heuristically, BDEZStab is faster than BDEZ by a factor $\#\,\mathrm{Stab}\,(T)$. This method constitutes the state of the art for the current work: our contribution is compared to the performance of this algorithm.

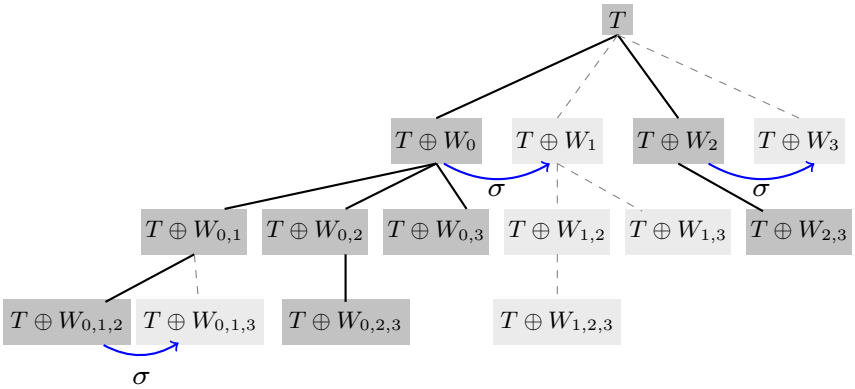

Figure 2.3: Pruning branches in an exhaustive search using RP-automorphisms.

Chapter 3

A new algorithm using coverings

Our contribution consists in reducing the number of vector spaces W that we need to enumerate in order to find those that satisfy $T \oplus W \in \mathscr{S}_r$, where T is the vector space corresponding to a bilinear map. To this effect, we restrict ourselves to vector spaces W satisfying some properties intrinsic to T. In this chapter, the definition and theoretical aspects of the set of vector spaces that satisfy these properties are treated. We give several examples, such as the short product, the circulant product and the matrix product, in Chapter 4.

3.1 Intermediate algorithms

Algorithms ImprovedRecursiveExhaustiveSearch and BDEZ, presented in Section 2.2.3 and 2.2.4, can be seen as two extreme cases of a more general set of strategies. Indeed, ImprovedRecursiveExhaustiveSearch and BDEZ are both associated to a tree whose leaves are labeled by $\binom{c_{m,n}}{d}$ elements (not necessarily distinct) of $\mathscr{S}_d$, d being a positive integer. In the case of ImprovedRecursiveExhaustiveSearch, $d = r$ where r is a candidate for the rank of a bilinear map $\mathbf{\Phi}$ represented by the subspace T. In the case of BDEZ, $d = r - \ell$, where ℓ is the dimension of T. Both algorithms call ExpandSubspace, but with different inputs.

Between RecursiveExhaustiveSearch and BDEZ, we can define intermediate algorithms, involving a parameter $k \in \{0, \dots, \ell\}$, consisting in enumerating $W \in \mathscr{S}_{r-\ell+k}$ and then selecting the subspaces, which satisfy

$$T + W \in \mathscr{S}_r.$$

At first sight, since $\#\mathscr{S}_{r-\ell+k} > \#\mathscr{S}_{r-\ell}$, BDEZ should be preferred to any of those intermediate algorithms. However, for vector spaces T associated to specific bilinear maps, there exists a positive integer k such that any $V \in \mathscr{S}_r(T)$ can be decomposed as

$$V = T + W$$

where $W \in \mathscr{S}_{r-\ell+k}$ and $T \cap W$ satisfies some property, intrinsic to T, such as containing a particular subspace F of dimension k, or in broader generality containing one subspace F_i among a finite set $\{F_0, \dots, F_{g-1}\}$ that is determined by T. Later on in this work, we will make this notion more precise and call such a finite set a *stem*. It allows one to compute a covering of $\mathscr{S}_r(T)$, which is why we use this abuse of terminology.

Our main contribution uses the fact that, for a well chosen subspace $F \subset T$ of dimension k, we may have

$$\#\mathscr{S}_{r-\ell+k}(F) \geq \#\mathscr{S}_{r-\ell}. \tag{3.1}$$

Thus, for specific T's, for which we know a set of subspaces F_i satisfying the Inequality 3.1 along with an algorithm computing the sets $\mathscr{S}_{r-\ell+k}(F_i)$ faster than $\mathscr{S}_{r-\ell}$, we are able to deduce the set $\mathscr{S}_r(T)$ faster than BDEZ. Note that this method relies on the knowledge of the subspaces F_i, which form, in fact, the so-called stem, as defined formally in the next section. Thus, the complexity of our method is difficult to establish, as it takes into account the algebraic specificities of the bilinear maps that are considered, such as the short product or the matrix product.

3.2 Coverings of optimal decompositions

First, our strategy consists in constructing, for any $r \geq \ell$, g sets $\{\mathcal{E}_{i,r}\}_i$ that are g subsets of $\mathscr{S}_{r-\ell+k_i}$, where k_i is a non negative integer, satisfying some property described in Definition 3.1.

Definition 3.1 (Covering of a vector space)**.** *Let T be a subspace of $\mathrm{GL}(K^m) \times \mathrm{GL}(K^n)$ of dimension ℓ. Let $(\mathcal{E}_{i,r})_{0\leq i<g}$ be a family of subsets parameterized by an integer $r \geq \ell$ and where $\mathcal{E}_{i,r} \subset \mathscr{S}_{r-\ell+k_i}$, k_i being a positive integer, for all $i \in \{0,\dots,g-1\}$,. Then, $(\mathcal{E}_{i,r})_{0\leq i<g}$ is said to be a covering of T if and only if, for any vector space $W \in \mathscr{S}_{r-\ell}$ such that $T \oplus W \in \mathscr{S}_r$, there exist an index $i \in \{0,\dots,g-1\}$, a subspace $V \in \mathcal{E}_{i,r}$, and an* RP*-automorphism $\sigma \in \mathrm{Stab}(T)$ such that $T + (V \circ \sigma) = T \oplus W$.*

Proposition 3.2. *Given T as above and a covering $(\mathcal{E}_{i,r})_{0\leq i<g}$ of T, then, for any $r \geq \ell$, we have*

$$\mathscr{S}_r(T) \subset \{T + V \mid \exists i \in \{0,\dots,g-1\},\ V \in \mathcal{E}_{i,r} \circ \mathrm{Stab}(T)\}.$$

Proof. Let $V \in \mathscr{S}_r(T)$. By Proposition 2.5, there exists $W \in \mathscr{S}_{r-\ell}$ such that $T \oplus W = U$. Then, by Definition 3.1, there exist an index $i \in \{0,\dots,g-1\}$, a subspace $V \in \mathcal{E}_{i,r}$, and an RP-automorphism $\sigma \in \mathrm{Stab}(T)$ such that $T + (V \circ \sigma) = T \oplus W$. Taking $V' = V \circ \sigma$, we thus have $U = T + V'$ and $V' \in \mathcal{E}_{i,r} \circ \mathrm{Stab}(T)$, which proves the inclusion. □

Thus, assuming that we have a method for computing the $\mathcal{E}_{i,r}$'s, we are able to cover the whole set $\mathscr{S}_r(T)$. For example, the family composed of the single set $\mathcal{E}_{0,r} = \mathscr{S}_{r-\ell}/\mathrm{Stab}(T)$ is a covering of T and an algorithm is known for its enumeration (namely, BDEZStab). We describe below how we construct the $\mathcal{E}_{i,r}$'s that we use in practice.

Definition 3.3 (Stem of a vector space)**.** *For a vector space T, a family $(F_i)_{0\leq i<g}$ of g subspaces $F_i \subset T$ of dimension k_i is said to be a* stem *of T if and only if, for any basis $\mathcal{B}$ of T, there exist $i \in \{0,\dots,g-1\}$, an* RP*-automorphism $\sigma \in \mathrm{Stab}(T)$ and a free family $\mathcal{F} \subset \mathcal{B}$ of size k_i such that*

$$\mathrm{Span}(\mathcal{F}) \circ \sigma = F_i.$$

Proposition 3.4. *For a vector space T, a stem of T given by g subspaces $F_i \subset T$, and g subgroups $U_i \subset \mathrm{Stab}(T) \cap \mathrm{Stab}(F_i)$, the set of $\mathcal{E}_{i,r}$'s, such that $\mathcal{E}_{i,r}$ is a set of representatives of the quotient $\mathscr{S}_{r-\ell+k_i}(F_i)/U_i$, is a covering of T.*

Proof. Let $W \in \mathscr{S}_{r-\ell}$ be such that $T \oplus W \in \mathscr{S}_r$. Take a basis $\mathcal{W}$ of W, and complete it into a basis of $T \oplus W$ using ℓ rank-one bilinear forms, denoted by $\{\psi_i\}_{0\leq i<\ell}$. For all $i \in \{0,\dots,\ell-1\}$, write $\psi_i = t_i + w_i$, with $t_i \in T$ and $w_i \in W$.

The t_i's are linearly independent. Otherwise, there would exist coefficients $(\lambda_i)_{0\leq i<\ell}$ such that $\sum_{i=0}^{\ell-1}\lambda_i t_i = 0$, whence $\sum_{i=0}^{\ell-1}\lambda_i\psi_i = \sum_{i=0}^{\ell-1}\lambda_i w_i$, which would then contradict the fact that $\{\psi_i\}_{0\leq i<\ell}$ completes $\mathcal{W}$ into a basis of $T\oplus W$.

Consequently, $\mathcal{B} = \{t_i\}_{0\leq i<\ell}$ is a free family of ℓ vectors of T and, as $\dim(T) = \ell$, $\mathcal{B}$ is a basis of T. Then, by Definition 3.3, there exist an index $i \in \{0,\dots,g-1\}$, a subset $\mathcal{F}\subset\mathcal{B}$ of size $k_i = \dim(F_i)$, and an RP-automorphism $\sigma\in\mathrm{Stab}(T)$ such that $\mathrm{Span}(\mathcal{F})\circ\sigma = F_i$.

Let $V = W\oplus\mathrm{Span}(\mathcal{F})$. Writing $\mathcal{F} = \{t_i\}_{i\in I}$, with $I\subset\{0,\dots,\ell-1\}$, we define $\mathcal{F}' = \{\psi_i\}_{i\in I}$. Since $\psi_i = t_i + w_i$ and $\mathrm{Span}(\mathcal{F}')\in\mathscr{S}_{k_i}$, we have $V = W\oplus\mathrm{Span}(\mathcal{F}) = W\oplus\mathrm{Span}(\mathcal{F}')\in\mathscr{S}_{r-\ell+k_i}$.

Now, consider $V' = V\circ\sigma = (W\oplus\mathrm{Span}(\mathcal{F}))\circ\sigma$: we also have $V'\in\mathscr{S}_{r-\ell+k_i}$, as RP-automorphisms preserve the bilinear rank, and $F_i = \mathrm{Span}(\mathcal{F})\circ\sigma\subset V'$, whence $V'\in\mathscr{S}_{r-\ell+k_i}(F_i)$.

Finally, let $V''\in\mathcal{E}_{i,r}$ be a representative of the equivalence class of V' in the quotient set $\mathscr{S}_{r-\ell+k_i}(F_i)/U_i$: there exists an RP-automorphism $\gamma\in U_i$ such that $V'' = V'\circ\gamma$. We then have

$$T + (V''\circ\gamma^{-1}\circ\sigma^{-1}) = T + (V'\circ\sigma^{-1}) = T + V = T + (W\oplus\mathrm{Span}(\mathcal{F})) = T\oplus W$$

where the last equality comes from the fact that $\mathrm{Span}(\mathcal{F})\subset T$. Finally, as $\gamma^{-1}\circ\sigma^{-1}\in\mathrm{Stab}(T)$, this proves the result. □

Given T and a stem of T, we can derive a new algorithm that computes $\mathscr{S}_r(T)$ via the computation of some intermediate sets $\mathcal{E}_{i,r} = \mathscr{S}_{r-\ell+k_i}(F_i)/U_i$ for $i\in\{0,\dots,g-1\}$.

Example 3.5 (Two examples of stems)**.** *For any vector space T, let $\mathcal{B}$ be a basis of T. There exists a subset of $\mathcal{B}$ generating T (namely, $\mathcal{B}$): $\{T\}$ is a stem of T. There exists also a subset of $\mathcal{B}$ generating* $\mathrm{Span}(\emptyset)$ *(namely, $\emptyset$): $\{\mathrm{Span}(\emptyset)\}$ is a stem of T.*

- *An enumeration algorithm that uses $\{T\}$ as a stem amounts to computing $\mathscr{S}_r(T)/\mathrm{Stab}(T)$. In this case, we did not decompose the original problem into simpler problems.*
- *If the stem chosen is the set $\{\mathrm{Span}(\emptyset)\}$, this is equivalent to enumerate a set of representatives of the quotient $\mathscr{S}_{r-\ell}(\mathrm{Span}(\emptyset))/\mathrm{Stab}(T)$. For this purpose, no better methods than* BDEZStab *is known.*

Thus, BDEZStab can be seen as an approach derived from the stem $\{\mathrm{Span}(\emptyset)\}$. We propose here other strategies that are derived from stems, given by sets of subspaces $F_i\subset T$ of dimension k_i. The enumeration of a set $\mathscr{S}_{r-\ell+k_i}(F_i)$ is interesting in practice if its cardinality is less than $\#\mathscr{S}_{r-\ell}$. However, its cost depends also on the algorithms used for the computation of quotients and stabilizers and on how large k_i is, which is detailed below.

No automatic method is known to determine, how to choose a stem for a given vector space T: we have to provide a stem for each T. This task has to be done by hand specifically for each bilinear map. In Chapter 4, we provide examples of such families for several bilinear maps.

Let $\mathcal{F}_i$ be a basis of F_i. Our strategy assumes that we have a finite representation of a group U_i such that $U_i\subset\mathrm{Stab}(T)\cap\mathrm{Stab}(F_i)$. In Proposition 3.4, the larger the groups U_i are, the smaller the $\mathcal{E}_{i,r}$'s are. And we prefer to keep the $\mathcal{E}_{i,r}$'s as small as possible, since it gives smaller sets to enumerate. Thus, this should lead us to choose $U_i = \mathrm{Stab}(T)\cap\mathrm{Stab}(F_i)$. However, in practice, the method used in our implementation is specialized to the choice $U_i = \mathrm{Stab}(T)\cap\mathrm{Stab}(\mathcal{F}_i)\subset\mathrm{Stab}(T)\cap\mathrm{Stab}(F_i)$ (we have $\mathrm{Stab}(\mathcal{F}_i)\subset\mathrm{Stab}(F_i)$) because only in this case do we have a practical algorithm to enumerate a set of representatives for the quotient $\mathscr{S}_{r-\ell+k_i}(F_i)/U_i$.

Notation 3.6. *For a free family $\mathcal{F}$ of k bilinear forms and a positive integer d, we let*

$$\tilde{\mathscr{S}}_{d+k}(\mathcal{F}) = \mathscr{S}_{d+k}(\mathrm{Span}(\mathcal{F}))/\mathrm{Stab}(\mathcal{F}).$$

In order to enumerate sets of the form $\mathscr{S}_{r-\ell+k_i}(F_i)/\mathrm{Stab}(T) \cap \mathrm{Stab}(\mathcal{F}_i)$, we adopt a three-step strategy.

Remark 3.7. *This strategy requires the precomputation of a set of representatives of the quotient*

$$\mathscr{S}_{r-\ell+k_i}/\mathrm{GL}(K^m) \times \mathrm{GL}(K^n).$$

Chapter 5 describes how to compute such a set. We may also precompute the quotient

$$\mathscr{S}_{r-\ell+k_i}/\mathrm{RPA}_{m,n},$$

which is smaller. It should be used in an optimized implementation. In our applications, it is not the main obstacle and the loss can often be recovered if the vector space of a bilinear map is invariant under the action of the transposition.

However, there is a practical limit on their dimension k_i, due to the precomputations that are used in our method and that constitute a bottleneck. Assuming that

$$\#\left(\mathscr{S}_d/\mathrm{GL}(K^d) \times \mathrm{GL}(K^d)\right)$$

behaves as $(d!)^{1.1}$ over $\mathbb{F}_2$ (which is an empirical estimate), storing a set of representatives of

$$\mathscr{S}_d/\mathrm{GL}(K^d) \times \mathrm{GL}(K^d)$$

for $d = 13$ would require 15 terabytes for instance. Consequently, given the largest "d" for which we are able to compute in practice

$$\mathscr{S}_d/\mathrm{GL}(K^m) \times \mathrm{GL}(K^n),$$

we have a practical constraint on how large the $r-\ell+k_i$'s may be: we should have $r-\ell+k_i \leq d$ for all i.

The first step consists in computing $\tilde{\mathscr{S}}_{r-\ell+k_i}(\mathcal{F}_i)$ and is detailed in Section 3.3. The second step applies the action of the left transversal

$$\mathrm{Stab}(\mathcal{F}_i)/\mathrm{Stab}(T) \cap \mathrm{Stab}(\mathcal{F}_i),$$

which can be computed using the algorithms proposed in [33] for example. The third step consists in calling HasRankOneBasis for each element of the previous set.

We describe in Algorithm CoveringSetsMethod the global strategy to find optimal formulas for T in the sense of the bilinear rank, that is, to enumerate $\mathscr{S}_r(T)$ given a stem. We assume that we are given a subspace T and a set of g free families $\mathcal{F}_0, \ldots, \mathcal{F}_{g-1}$ of T such that $\{\mathrm{Span}(\mathcal{F}_i)\}_i$ forms a stem of T.

Algorithm 7 CoveringSetsMethod

Input: $T, \{\mathcal{F}_i\}_{0\le i<g}, \left\{\mathscr{S}_{r-\ell+k_i}/\mathrm{GL}(K^m)\times\mathrm{GL}(K^n)\right\}_{0\le i<g}, r$
Output: $\mathscr{S}_r(T)$
1: $\mathcal{S}\leftarrow\emptyset$
2: **for** $i\in\{0,\ldots,g-1\}$ **do**
3: $\quad\mathcal{Q}\leftarrow\tilde{\mathscr{S}}_{r-\ell+k_i}(\mathcal{F}_i)$, obtained from $\mathscr{S}_{r-\ell+k_i}/\mathrm{GL}(K^m)\times\mathrm{GL}(K^n)$ ▷ See Algorithm 8 in 3.3
4: $\quad\mathcal{L}\leftarrow\mathrm{Stab}(\mathcal{F}_i)/\mathrm{Stab}(T)\cap\mathrm{Stab}(\mathcal{F}_i)$ ▷ use e.g. [33]
5: $\quad$**for** $\sigma\in\mathcal{L}, W\in\mathcal{Q}$ **do**
6: $\quad\quad$**if** HasRankOneBasis$(T+(W\circ\sigma))$ **then**
7: $\quad\quad\quad\mathcal{S}\leftarrow\mathcal{S}\cup\{T+(W\circ\sigma)\}$
8: $\quad\quad$**end if**
9: $\quad$**end for**
10: **end for**
11: **return** $\bigcup_{V\in\mathcal{S}} V\circ\mathrm{Stab}(T)$

The computation of the quotient $\mathcal{Q}$ on Line 3 of CoveringSetsMethod is detailed in Section 3.3.

3.3 Enumerating the elements of $\tilde{\mathscr{S}}_{r-\ell+k}(\mathcal{F})$

In this section, T is a vector space of dimension ℓ. For $\Psi_0,\ldots,\Psi_{k-1}$ bilinear forms of $\mathcal{L}(K^m,K^n;K)$, Algorithm 8 explains how to compute a set of representatives of the quotient

$$\tilde{\mathscr{S}}_{r-\ell+k}(\{\Psi_0,\ldots,\Psi_{k-1}\}) = \mathscr{S}_{r-\ell+k}(\mathrm{Span}(\{\Psi_0,\ldots,\Psi_{k-1}\}))/\mathrm{Stab}(\{\Psi_0,\ldots,\Psi_{k-1}\}),$$

which is required in Algorithm CoveringSetsMethod of Section 3.2.

Algorithm 8 IntermediateSetViaQuotientComputation

Input: Ω a set of representatives of $\mathscr{S}_{r-\ell+k}/\mathrm{GL}(K^m)\times\mathrm{GL}(K^n), \{\Psi_0,\ldots,\Psi_{k-1}\}$
Output: One representative per orbit of $\tilde{\mathscr{S}}_{r-\ell+k}(\{\Psi_0,\ldots,\Psi_{k-1}\})$, defined as above
1: $\mathcal{Q}\leftarrow\emptyset$
2: **for** $W\in\Omega$ **do**
3: $\quad$**for** $\{\{\Phi_0,\ldots,\Phi_{k-1}\}\subset W \mid \forall t,\ \mathrm{rk}(\Phi_t)=\mathrm{rk}(\Psi_t)\}/\mathrm{Stab}(W)$ **do** ▷ See Algorithm 9
4: $\quad\quad$**if** $\exists\sigma\in\mathrm{GL}(K^m)\times\mathrm{GL}(K^n),\ \{\Phi_0,\ldots,\Phi_{k-1}\}\circ\sigma=\{\Psi_0,\ldots,\Psi_{k-1}\}$ **then**
5: $\quad\quad\quad\mathcal{Q}\leftarrow\mathcal{Q}\cup\{W\circ\sigma\}$
6: $\quad\quad$**end if**
7: $\quad$**end for**
8: **end for**
9: **return** $\mathcal{Q}$

Correctness of Algorithm 8. By construction, any element of $\mathcal{Q}$ is an element of

$$\mathscr{S}_{r-\ell+k}(\{\Psi_0,\ldots,\Psi_{k-1}\}).$$

- First, we prove that any orbit of $\tilde{\mathscr{S}}_{r-\ell+k}(\{\Psi_0,\ldots,\Psi_{k-1}\})$ has a representative in $\mathcal{Q}$.

  Let W' be a representative of an orbit in $\tilde{\mathscr{S}}_{r-\ell+k}(\{\Psi_0,\ldots,\Psi_{k-1}\})$. There exist $\sigma\in\mathrm{Stab}(\{\Psi_0,\ldots,\Psi_{k-1}\})$ and $W\in\mathcal{Q}$ such that $W\circ\sigma=W'$. Thus, we have $\{\Psi_0,\ldots,\Psi_{k-1}\}\circ\sigma^{-1}\subset$

W and the set $\left\{\Psi_0 \circ \sigma^{-1}, \ldots, \Psi_{k-1} \circ \sigma^{-1}\right\}$ satisfies the predicate on Line 4. Any σ' such that $\{\Psi_0, \ldots, \Psi_{k-1}\} \circ \sigma^{-1} \circ \sigma' = \{\Psi_0, \ldots, \Psi_{k-1}\}$ satisfies

$$\sigma' \in \sigma \circ \mathrm{Stab}(\{\Psi_0, \ldots, \Psi_{k-1}\}),$$

which means that an element of $W \circ \sigma \circ \mathrm{Stab}(\{\Psi_0, \ldots, \Psi_{k-1}\}) = W' \circ \mathrm{Stab}(\{\Psi_0, \ldots, \Psi_{k-1}\})$ is included in the list returned by Algorithm 8. Thus, the list returned contains at least one representative per orbit of $\tilde{\mathscr{S}}_{r-\ell+k}(\{\Psi_0, \ldots, \Psi_{k-1}\})$.

- In the following, we prove that each orbit of $\tilde{\mathscr{S}}_{r-\ell+k}(\{\Psi_0, \ldots, \Psi_{k-1}\})$ has a unique representative in $\mathcal{Q}$.

  Assume that there exist $W, W' \in \mathcal{Q}$ and $\gamma \in \mathrm{Stab}(\{\Psi_0, \ldots, \Psi_{k-1}\})$ such that $W = W' \circ \gamma$. By construction, there exists $W_0, W_0' \in \mathscr{S}_{r-\ell+k}$ and $\sigma, \sigma' \in \mathrm{GL}(K^m) \times \mathrm{GL}(K^n)$ such that $W = W_0 \circ \sigma$ and $W' = W_0' \circ \sigma'$. Then $W_0' = W_0 \circ \sigma \circ \gamma^{-1} \circ \sigma'^{-1}$, whence $W_0' = W_0$ as on Line 2 of Algorithm 8 we enumerate only one representative of each orbit of $\mathscr{S}_{r-\ell+k}/\mathrm{GL}(K^m) \times \mathrm{GL}(K^n)$. Thus, $\sigma \circ \gamma^{-1} \circ \sigma'^{-1} \in \mathrm{Stab}(W_0)$. Still by construction, there exists $\{\Phi_0, \ldots, \Phi_{k-1}\}$ and $\{\Phi_0', \ldots, \Phi_{k-1}'\} \subset W_0$ such that

  $$\{\Phi_0, \ldots, \Phi_{k-1}\} \circ \sigma = \{\Psi_0, \ldots, \Psi_{k-1}\}$$

  and

  $$\{\Phi_0', \ldots, \Phi_{k-1}'\} \circ \sigma' = \{\Psi_0, \ldots, \Psi_{k-1}\}.$$

  Then,

  $$\{\Phi_0', \ldots, \Phi_{k-1}'\} = \{\Psi_0, \ldots, \Psi_{k-1}\} \circ \sigma'^{-1} = \{\Psi_0, \ldots, \Psi_{k-1}\} \circ \gamma^{-1} \circ \sigma'^{-1} = \{\Phi_0, \ldots, \Phi_{k-1}\} \circ \sigma \circ \gamma^{-1} \circ \sigma'^{-1}$$

  and $\{\Phi_0, \ldots, \Phi_{k-1}\}$ is in the same orbit as $\{\Phi_0', \ldots, \Phi_{k-1}'\}$ under the action of $\mathrm{Stab}(W_0)$, which is contradictory with the definition of the quotient on Line 3.

□

On Line 3 of Algorithm 8, we select all tuples of size k of elements $\Phi_0, \ldots, \Phi_{k-1}$ such that $\mathrm{rk}(\Phi_i) = \mathrm{rk}(\Psi_i)$ for any $i \in \{0, \ldots, k-1\}$. The different tuples are enumerated up to the action of $\mathrm{Stab}(W)$. We proceed like described in Algorithm EnumerateTuples. We use a backtracking algorithm, similar to BDEZ, for which the recursive depth is bounded by k.

The number of recursive calls is bounded by $\#W$ for this algorithm. Consequently, the complexity is bounded by $(\#W)^k$, without taking into account the gain due to the use of RP-automorphisms. The algorithms described in [33] allows one to compute the orbits on Line 6. In our applications, k and $\dim(W)$ are small.

Testing the predicate of Line 4 of Algorithm 8 is a problem generalizing the problem of [12, Ch. 19] and [36]: given two pairs (M_0, M_1) and (N_0, N_1) of $(\mathcal{M}_{m,n})^2$, determine whether there exists two invertible matrices X and Y such that $(X^{\mathrm{T}} M_0 Y, X^{\mathrm{T}} M_1 Y) = (N_0, N_1)$. This can be done by computing a Weierstrass-Kronecker canonical form for (M_0, M_1). When we consider more than 2 matrices, for example 3 matrices (M_0, M_1, M_2) sent on (N_0, N_1, N_2), we compute (X, Y) such that (M_0, M_1) is sent on (N_0, N_1) and we compose it with elements of $(\mathrm{Stab}\, M_0 \cap \mathrm{Stab}\, M_1)/\mathrm{Stab}\, M_2$, computed with the algorithms proposed in [33] for example. The complexity for finding all the RP-automorphisms σ in IntermediateSetViaQuotientComputation is bounded by the cardinality of $\mathscr{S}_{r-\ell+k}$ (which is comparable to BDEZ) by construction, and is hard to estimate more precisely. In our applications, it appears to be negligible compared to BDEZ.

Algorithm 9 EnumerateTuples

Input: k, W, $\mathrm{Stab}(W)$, $\Psi_0, \ldots, \Psi_{k-1}$
Output: One representative per orbit of $\{\{\Phi_0, \ldots, \Phi_{k-1}\} \subset W \mid \forall t,\ \mathrm{rk}(\Phi_t) = \mathrm{rk}(\Psi_t)\}/\mathrm{Stab}(W)$

```
function RecursiveEnumeration(Φ, C, i, U) ▷ C is a partition of the non zero elements of W, U is a subgroup of Stab(W)
    if i = k then
        return Φ
    else
        S ← ∅
        C' ← ⋃_{O∈C} {O/U}
        for j ∈ {0, ..., #C' − 1} do                          ▷ C' = {O_j | j ∈ [0, #C' − 1]}
            Φ[i] ← Representative(O_j)
            if rk(Φ[i]) = rk(Ψ_i) then
                S ← S ∪ RecursiveEnumeration(t, {O_j, ..., O_{#C'−1}}, i + 1, U ∩ Stab(Φ[i]))
            end if
        end for
        return S
    end if
end function
C ← {{Φ ∈ W | Φ ≠ 0}}
Φ ← ( )                                                       ▷ Empty vector of size k
return RecursiveEnumeration(Φ, C, 0, Stab(W))
```

Chapter 4

Application to examples of bilinear maps

An implementation in Magma V2.21-3 [9] of the algorithms BDEZ, BDEZStab and the new method has been done and is available at the address

http://karancode.gforge.inria.fr.

We provide in this section the stems corresponding to certain bilinear maps and we compare the timings obtained for them, using different algorithms. Our Magma implementation of the algorithm is clearly slower than the original C version described in [2]. However, since we are interested in the speed-up obtained from our work, we need a fair approach. We show in particular that Algorithm BDEZStab, although it has not been written in a multi-threaded version and in C, improves considerably on the timings estimated in [2]. The new algorithm described in Chapter 3 is denoted by CoveringSetsMethod: compared to Algorithm BDEZStab, it constitutes a huge speed-up on particular instances of the bilinear rank problem. All the timings presented in this section have been done on a single core 3.3 GHz Intel Core i5-4590.

We need a few notations to denote the various bilinear maps we are interested in:

- $\mathsf{MatProd}_{(p,q,r)}$ denotes the product of matrices $p \times q$ by $q \times r$,
- ShProd_ℓ denotes the product of polynomials modulo X^ℓ,
- $\mathsf{CircProd}_\ell$ the product of polynomials modulo $X^\ell - 1$.

We describe in Table 4.1 timings for various bilinear maps and the implementations of BDEZ and BDEZStab. The number of tests represents the number of calls to HasRankOneBasis.

It is possible to estimate the time it would take to obtain a result for a bilinear rank problem out of reach for BDEZ or BDEZStab. We denote by $\mathcal{N}_t$ the number of calls to HasRankOneBasis in these algorithms when the input r is equal to $\ell + t$. (ℓ is the dimension of the vector space T corresponding to the bilinear map). Since when r is too large, BDEZ is too expensive, there is a practical limit on the known values of $\mathcal{N}_t$, t being a positive integer. We consider the ratio $\lceil \frac{\mathcal{N}_t}{\mathcal{N}_{t-1}} \rceil$ to estimate $\mathcal{N}_{t+1}$. Assuming that this ratio decreases with t, which seems to hold empirically, we have

$$\mathcal{N}_{t+1} \leq \left\lceil \frac{\mathcal{N}_t}{\mathcal{N}_{t-1}} \right\rceil \cdot \mathcal{N}_t, \tag{4.1}$$

t being a positive integer of $\{1, \ldots, r-\ell\}$.

Thus, we are able to predict timings for bilinear maps indicated in Table 4.1 via to this assumption, which allows us to compare Algorithm BDEZ to other approaches for problems of larger sizes. We estimate the number of tests by computing

$$\mathcal{N}_t \cdot \left\lceil \frac{\mathcal{N}_t}{\mathcal{N}_{t-1}} \right\rceil^{r-\ell-t}$$

where $r-\ell$ is the difference $\mathrm{rk}(T) - \dim(T)$ for T representing a bilinear map and t is the largest integer for which we are able to compute $\mathcal{N}_t$. The time can be estimated with a similar technique. We observe that the speed-up seems to match with $\#\,\mathrm{Stab}(T)$, as expected. The estimated values in Table 4.1 relying on BDEZStab have not been effectively done because the implementation of CoveringSetsMethod allowed us to obtain more results, more efficiently. The estimations rely on the heuristic given by the Inequality 4.1. In the global strategy, we were increasing progressively the lower bound r on the rank, before running BDEZ, BDEZStab or CoveringSetsMethod. For $r < \mathrm{rk}(T)$, the time spent in those algorithms is negligible, because of the exponential growth of their complexity.

It is not clear how to estimate timings for our approach CoveringSetsMethod beyond what has been done and reported in Table 4.1. However, for the set of bilinear maps for which CoveringSetsMethod allows one to compute all the optimal formulas, we observe a clear speed-up compared to BDEZStab. For example, in order to compute the bilinear rank of bilinear maps of larger degrees using this method, we need to be able to compute and store all the elements of

$$\mathscr{S}_{10}/\mathrm{GL}(K^{10}) \times \mathrm{GL}(K^{10})$$

for ShProd_6 (and even more for other bilinear maps), which has not been done yet and requires an optimized implementation of the algorithm described in Chapter 5, combined with a classification of its orbits.

Henceforth, we describe how we computed optimal formulas for bilinear maps given in Table 4.1 via our approach. We provide some technical details, specific to each bilinear map, necessary for an implementation.

4.1 Application to the short product

4.1.1 Structure of the short product

Let ℓ be a positive integer, let $\mathbf{\Phi}$ be the bilinear map $\mathbf{\Phi} \in \mathcal{L}(K^\ell, K^\ell; K^\ell)$ defined by the short product

$$\mathbf{\Phi} : \begin{pmatrix} a_0 \\ \vdots \\ a_{\ell-1} \end{pmatrix}, \begin{pmatrix} b_0 \\ \vdots \\ b_{\ell-1} \end{pmatrix} \mapsto \begin{pmatrix} c_0 \\ \vdots \\ c_{\ell-1} \end{pmatrix}$$

such that $\sum_{0\le i<\ell} c_i X^{\ell-1-i} = (\sum_{0\le i<\ell} a_i X^i)(\sum_{0\le i<\ell} b_i X^{\ell-1-i}) \mod X^\ell$. Let T be the subspace of $\mathcal{L}(K^\ell, K^\ell; K)$ spanned by the ℓ bilinear forms that are the coordinates of $\mathbf{\Phi}$, denoted by $\Phi_0, \ldots, \Phi_{\ell-1}$.

The matrix representing the element $\sum_{0\le i<\ell} m_i \Phi_i \in T$, where $m_i \in K$, is

bilinear map	rank	algorithm	nb. of tests	time (s)
ShProd_3	5	BDEZ	$5.9 \cdot 10^2$	$1.4 \cdot 10^{-1}$
		BDEZStab	$3.4 \cdot 10$	0.0
ShProd_4	8	BDEZ	$5.2 \cdot 10^7$	$4.3 \cdot 10^3$
		BDEZStab	$3.1 \cdot 10^5$	$2.7 \cdot 10$
		CoveringSetsMethod	$\mathbf{2.8 \cdot 10^2}$	**3.0**
ShProd_5	11	BDEZ	$1.8 \cdot 10^{16}$ (est.)	$5.7 \cdot 10^{12}$ (est.)
		BDEZStab	$6.9 \cdot 10^{11}$ (est.)	$2.2 \cdot 10^8$ (est.)
		CoveringSetsMethod	$\mathbf{6.3 \cdot 10^6}$	$\mathbf{2.4 \cdot 10^3}$
ShProd_6	14	BDEZ	$3.9 \cdot 10^{26}$ (est.)	$4.7 \cdot 10^{23}$ (est.)
		BDEZStab	$2.0 \cdot 10^{19}$ (est.)	$2.7 \cdot 10^{16}$ (est.)
$\mathsf{CircProd}_3$	4	BDEZ	36	0.0
		BDEZStab	6	$0.1 \cdot 10^{-2}$
$\mathsf{CircProd}_4$	8	BDEZ	$5.2 \cdot 10^7$	$4.3 \cdot 10^3$
		BDEZStab	$3.1 \cdot 10^5$	$2.7 \cdot 10$
$\mathsf{CircProd}_5$	10	BDEZ	$4.0 \cdot 10^{13}$ (est.)	$1.2 \cdot 10^{10}$ (est.)
		BDEZStab	$1.0 \cdot 10^{10}$ (est.)	$3.5 \cdot 10^6$ (est.)
		CoveringSetsMethod	$\mathbf{8.8 \cdot 10^8}$	$\mathbf{5.4 \cdot 10^3}$
$\mathsf{CircProd}_6$	12	BDEZ	$1.0 \cdot 10^{20}$ (est.)	$1.3 \cdot 10^{17}$ (est.)
		BDEZStab	$1.1 \cdot 10^{15}$ (est.)	$1.5 \cdot 10^{12}$ (est.)
$\mathsf{MatProd}_{(2,2,2)}$	7	BDEZ	$1.05 \cdot 10^6$	$8.5 \cdot 10$
		BDEZStab	$6.8 \cdot 10^3$	$5.0 \cdot 10^{-1}$
$\mathsf{MatProd}_{(3,2,3)}$	15	BDEZ	$9.2 \cdot 10^{19}$ (est.)	$1.1 \cdot 10^{17}$ (est.)
		BDEZStab	$2.6 \cdot 10^{13}$ (est.)	$3.4 \cdot 10^{10}$ (est.)
		CoveringSetsMethod	$\mathbf{1.6 \cdot 10^9}$	$\mathbf{8.5 \cdot 10^5}$
$\mathsf{MatProd}_{(2,3,2)}$	11	BDEZ	$2.3 \cdot 10^{23}$ (est.)	$2.7 \cdot 10^{20}$ (est.)
		BDEZStab	$4.6 \cdot 10^{18}$ (est.)	$5.4 \cdot 10^{15}$ (est.)
		CoveringSetsMethod	$\mathbf{6.3 \cdot 10^{10}}$	$\mathbf{4.1 \cdot 10^6}$

Table 4.1: Number of tests and timings obtained with Algorithms BDEZ, BDEZStab and CoveringSetsMethod for various bilinear maps over $K = \mathbb{F}_2$.

$$M(m_0,\dots,m_{\ell-1}) = \begin{bmatrix} m_0 & m_1 & \cdots & m_{\ell-1} \\ 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & m_1 \\ 0 & \cdots & 0 & m_0 \end{bmatrix},$$

in the canonical basis. This matrix is an upper triangular Toeplitz matrix.

Let N be the matrix $M(0,\dots,1,0)$. The matrix N is a nilpotent matrix such that

$$\forall j \in \{0,\dots,\ell-1\},\ M(0,\dots,0,1,\underbrace{0,\dots,0}_{j \text{ zeros}}) = N^{\ell-1-j},$$

and $N^\ell = 0$. The elements of the algebra $K[N]$ are the upper triangular Toeplitz matrices and $K[N] \cong K[X]/(X^\ell)$.

We provide in Theorem 4.1 a useful property describing the action of $\mathrm{Stab}(T)$ on T.

Theorem 4.1. *Let any integer $\ell \geq 2$:*

1. *the orbit of the identity matrix $I = N^0$ for the action of $\mathrm{Stab}(T)$ is the set of invertible matrices of T;*
2. *the orbit of N for the action of $\mathrm{Stab}(T) \cap \mathrm{Stab}(I)$ is the set of nilpotent matrices of T;*
3. *for any pair (Ψ, Ψ') of elements of T such that $\mathrm{rk}(\Psi) = \ell$ and $\mathrm{rk}(\Psi') = \ell - 1$, there exists $\sigma \in \mathrm{Stab}(T)$ such that*
$$(\Psi \circ \sigma, \Psi' \circ \sigma) = (I, N);$$
4. *we have $\mathrm{Stab}(I) \cap \mathrm{Stab}(N) \subset \mathrm{Stab}(T)$ and the cardinality of $\mathrm{Stab}(T)$ is $(\#K)^{3\ell-4}(\#K-1)^3$.*

Proof. • First, we prove that, for any element $M \in T$ of rank ℓ, there exists $R \in K[X]$ a polynomial of degree at most $\ell - 1$ such that $R(0) \neq 0$ and $R(N) = M$, and that

$$\exists N_1 \in \mathrm{Stab}(T),\ I \cdot N_1 = R(N).$$

Any element in the orbit of I has rank ℓ and any element of rank ℓ in T is associated to a polynomial $R \in K[X]$ evaluated in N of degree $\ell - 1$ such that $R(0) \neq 0$. It remains to prove that the orbit of I corresponds exactly to the set of rank-ℓ elements. Given $R \in K[X]$ such that $R(0) \neq 0$, we denote by $N_1(R)$ the element $(I, R(N))$. This element is in $\mathrm{Stab}(T)$ because, for any S, we have $R(N)S(N) = (RS \bmod X^\ell)(N)$, which is a polynomial evaluated in N of degree at most $\ell - 1$. We have:

$$I \cdot N_1(R) = (I)^{\mathrm{T}} \cdot I \cdot R(N) = R(N).$$

• We prove that, for any element $M \in T$ of rank $\ell - 1$, there exists $R \in K[X]$ a polynomial of degree at most $\ell - 1$ such that $R(0) = 0$, $R'(0) \neq 0$ and $R(N) = M$, and that

$$\exists N_2 \in \mathrm{Stab}(I) \cap \mathrm{Stab}(T),\ N \cdot N_2 = R(N).$$

An element of the orbit of N is an element of rank $\ell-1$ and an element of rank $\ell-1$ in T is associated to a polynomial $R \in K[X]$ evaluated in N of degree at most $\ell-1$ such that $R(0)=0$ and $R'(0)\neq 0$. It remains to prove that N can mapped to any element of rank $\ell-1$ via the action of $\mathrm{Stab}(I)\cap\mathrm{Stab}(T)$.

Let e_ℓ be the vector $(0,\cdots,0,1)$, such that $R(N)\cdot e_\ell$ corresponds to the last colum of $R(N)$. We have $R(N)^{\ell-1}e_\ell \neq 0$. Thus, let $P(N)$ be the matrix whose columns are given by the tuple $(R(N)^{\ell-1}\cdot e_\ell, R(N)^{\ell-2}\cdot e_\ell,\ldots,R(N)\cdot e_\ell, e_\ell)$. We have $R(N)P(N) = (0, R(N)^{\ell-1}\cdot e_\ell,\ldots,R(N)^2\cdot e_\ell, R(N)\cdot e_\ell) = R(N)$ and $P(N)N = (0, R(N)^{\ell-1}\cdot e_\ell, R(N)^{\ell-2}\cdot e_\ell,\ldots,R(N)\cdot e_\ell)$. Consequently, we have $R(N)P(N) = P(N)N$ and $P(N)^{-1}R(N)P(N) = N$. We take $N_2(R) = (P(N)^{\mathrm{T}}, P(N)^{-1})$:

$$N\cdot N_2(R) = R(N) \text{ and } N_2(R)\in \mathrm{Stab}(I)\cap\mathrm{Stab}(T).$$

- Let (Ψ,Ψ') be a couple of elements of T such that $\mathrm{rk}(\Psi)=\ell$ and $\mathrm{rk}(\Psi')=\ell-1$. Let (P,P') be the corresponding matrices. According to the previous points, there exist $N_1\in\mathrm{Stab}(T)$ such that $I\cdot N_1 = P$ and $N_2\in\mathrm{Stab}(T)\cap\mathrm{Stab}(I)$ such that $N\cdot N_2 = P'\cdot N_1^{-1}$. Consequently, we have
$$(I,N) = (P\cdot N_1^{-1}\cdot N_2^{-1}, P'\cdot N_1^{-1}\cdot N_2^{-1}).$$
- We prove that we have $\mathrm{Stab}(I)\cap\mathrm{Stab}(N)\subset\mathrm{Stab}(T)$ and that, for any $N_3\in\mathrm{Stab}(I)\cap\mathrm{Stab}(N)$, there exists $R\in K[X]$ a polynomial of degree at most $\ell-1$ such that $R(0)\neq 0$ and
$$N_3 = ((R(N)^{-1})^{\mathrm{T}}, R(N)).$$
Let $N_3\in\mathrm{Stab}(I)\cap\mathrm{Stab}(N)$. Since $N_3\in\mathrm{Stab}(I)$, there exists $P\in\mathrm{GL}_\ell$ such that $N_3 = ((P^{-1})^{\mathrm{T}}, P)$ and, since $N_3\in\mathrm{Stab}(N)$, $P^{-1}NP = N$. We have $PN = NP$.
Multiplying a matrix by N on the left shifts the rows upward and multiplying N on the right shifts the columns on the right. Therefore, denoting by p_{ij} the coefficients of P, with $p_{00}\neq 0$ and $p_{i0}=0$ for $i\geq 1$, we have
$$\forall (i,j)\in\{1,\ldots,\ell-1\}\times\{0,\ldots,\ell-1\}, p_{i,j} = p_{i+1,j+1}.$$
More particularly, P is equal to the evaluation in N of a polynomial R such that $R(0)\neq 0$, from which we deduce that
$$N_3 = ((R(N)^{-1})^{\mathrm{T}}, R(N)) \text{ and } N_3\in\mathrm{Stab}(T).$$
Given the form of the elements of $\mathrm{Stab}(I)\cap\mathrm{Stab}(N)$, its cardinality is equal to the number of polynomials R of degree at most $\ell-1$ such that $R(0)\neq 0$, which is $\#K^{\ell-1}(K-1)$. Combining with the fact that there are $\#K^{\ell-1}(K-1)\cdot\#K^{\ell-2}(K-1)$ couples (Ψ,Ψ') of elements of T such that $\mathrm{rk}(\Psi)=\ell$ and $\mathrm{rk}(\Psi')=\ell-1$, we have $\#\mathrm{Stab}(T) = \#K^{3\ell-4}(\#K-1)^3$.

□

4.1.2 Stem of the short product

We consider in this section the example of the short product

$$\mathbf{\Phi}_\ell : (A,B)\longmapsto A\cdot B \bmod X^\ell = \begin{pmatrix} a_{\ell-1}b_0+\cdots+a_0b_{\ell-1}\\ \vdots\\ a_1b_0+a_0b_1\\ a_0b_0\end{pmatrix}$$

we denote by $\Phi_0, \ldots, \Phi_{\ell-1}$ the bilinear forms such that

$$\forall i \geq 0,\ \Phi_i(A,B) = \sum_{j \in \{0,\ldots,\ell-1-i\}} a_j b_{j+i}$$

and by T the subspace $\mathrm{Span}(\Phi_0, \ldots, \Phi_{\ell-1})$.

In order to produce a covering of the vector spaces W satisfying $T \oplus W \in \mathscr{S}_r(T)$ that we compute with CoveringSetsMethod, we need a stem of T. This stem is given in Proposition 4.2.

Proposition 4.2 (Stem of short product)**.** *Let ℓ and $r \geq \ell$. The set $\{\mathrm{Span}(\Phi_0, \Phi_1)\}$is a stem of T: for any basis $\mathcal{B}$ of T, there exists $\sigma \in \mathrm{Stab}(T)$ and $\mathcal{U} \subset \mathcal{B}$ of cardinality 2 such that*

$$\mathrm{Span}(\mathcal{U}) \circ \sigma = \mathrm{Span}(\Phi_0, \Phi_1).$$

Proof. We first observe that for any $\Phi \in \mathrm{Span}(\Phi_{\ell-1-i}, \ldots, \Phi_{\ell-1})$, $\mathrm{rk}(\Phi) \leq i+1$. Therefore, any element of rank ℓ in T has a nonzero coordinate over Φ_0 in its decomposition over the basis $(\Phi_0, \ldots, \Phi_{\ell-1})$ and, reciprocally, any element having a nonzero coordinate over Φ_0 has rank ℓ. Thus, a basis $\mathcal{B}$ of T necessarily contains an element of rank ℓ denoted by Ψ. The element Ψ has a nonzero coordinate over Φ_0, when we decompose it over $\{\Phi_0, \ldots, \Phi_{\ell-1}\}$. Similarly, there exist $\Psi' \in \mathcal{B}$ and $\lambda \in K$ for which $\Psi' - \lambda\Psi$ has rank $\ell - 1$.

We then use Theorem 4.1 to find an element $\sigma \in \mathrm{Stab}(T)$ such that

$$(\Psi \circ \sigma, \Psi' \circ \sigma) = (\Phi_0, \Phi_1) \text{ or } (\Psi \circ \sigma, (\Psi - \lambda\Psi') \circ \sigma) = (\Phi_0, \Phi_1),$$

which concludes. □

We give in Table 4.2 the cardinality of coverings of $\mathscr{S}_r(T)$ given by Proposition 4.2 for the example T:

set	**cardinality**
$\mathscr{S}_2(\mathrm{Span}(\emptyset)) = \mathscr{S}_2$	980
$\mathscr{S}_3(\mathrm{Span}(\Phi_0))$	28
$\mathscr{S}_4(\mathrm{Span}(\Phi_0, \Phi_1))$	6

Table 4.2: Comparison of the cardinality for $\ell = 3$ of three coverings of T for $K = \mathbb{F}_2$.

We recall that we need to compute the set $\mathcal{Q} = \tilde{\mathscr{S}}_{r-\ell+2}(\{\Phi_0, \Phi_1\})$ for a given integer r. If we take $\ell = 3$, we can represent Φ_0 and Φ_1 by the matrices

$$I = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \text{ and } N = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}.$$

Thus, for a given couple (M_0, M_1) of matrices representing bilinear forms of a subspace $W \in \mathscr{S}_{r-\ell+2}/\mathrm{GL}(K^\ell) \times \mathrm{GL}(K^\ell)$, we are looking for invertible matrices X and Y such that

$$X^{\mathrm{T}} M_0 Y = I \text{ and } X^{\mathrm{T}} M_1 Y = N,$$

Algorithm 10 IntermediateSetViaQuotientComputation (Short product)

Input: $\mathscr{S}_{r-\ell+2}/\mathrm{GL}(K^\ell) \times \mathrm{GL}(K^\ell)$
Output: One representative per orbit of $\mathcal{Q}$, defined as above
1: $\mathcal{Q} \leftarrow \emptyset$
2: **for** $W \in \mathscr{S}_{\ell,\ell,r-\ell+2}/\mathrm{GL}(K^\ell) \times \mathrm{GL}(K^\ell)$ **do**
3: **for** $\Psi \in \{\Phi \in W \mid \mathrm{rk}(\Phi) = \ell\}/\mathrm{Stab}(W)$ **do**
4: Let σ such that $\Psi \circ \sigma = I$ ▷ We obtain σ via a Gauss reduction
5: $W' \leftarrow W \circ \sigma$
6: **for** $\Psi' \in \{\Phi \in W' \mid \mathrm{rk}(\Phi) = \ell - 1\}/\mathrm{Stab}(W') \cap \mathrm{Stab}(I)$ **do**
7: **if** $\exists \sigma' \in \mathrm{Stab}(I),\ \Psi' \circ \sigma' = N$ **then** ▷ Using that N and Ψ' are similar
8: $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{W \circ \sigma \circ \sigma'\}$
9: **end if**
10: **end for**
11: **end for**
12: **end for**
13: **return** $\mathcal{Q}$

which is done in Algorithm 10. As it is precised on Line 4 of Algorithm 10, we find X and Y such that $X^{\mathrm{T}} M_0 Y = I$ via Gauss reduction. Then, we need to check whether $X^{\mathrm{T}} M_1 Y$ and N are similar or not ($(X^{\mathrm{T}} M_1 Y)^\ell$ should be the null matrix for this purpose), as done on Line 7 of Algorithm 10.

Once we have computed $\mathcal{Q}$, it remains to compute the left transversal

$$\mathcal{L} = \mathrm{Stab}(\{I, N\})/\mathrm{Stab}(T) \cap \mathrm{Stab}(\{I, N\})$$

and to compute $\mathcal{Q} \circ \mathcal{L}$. According to Theorem 4.1, we have $\#\mathcal{L} = 1$, which means that Algorithm 10 actually returns $\tilde{\mathscr{S}}_{r-\ell+2}(\{I, N\}) \circ \mathcal{L}$.

In terms of complexity, we do not have explicit bounds. However, we can state that the complexity depends linearly on $\#\mathscr{S}_{r-\ell+2}/\mathrm{GL}(K^\ell) \times \mathrm{GL}(K^\ell)$ and on the number of pairs of bilinear forms (Φ, Ψ) per element of $\mathscr{S}_{r-\ell+2}/\mathrm{GL}(K^\ell) \times \mathrm{GL}(K^\ell)$ such that $\mathrm{rk}(\Phi) = \ell$ and $\mathrm{rk}(\Psi) = \ell - 1$.

We managed to obtain all the elements of $\mathscr{S}_r(T)$, where T is the vector space generated by the bilinear forms associated to ShProd_ℓ for $\ell = 4$ and $\ell = 5$ and $r = \mathrm{rk}(T)$.

map	$\mathcal{F}$	d	$\#\tilde{\mathscr{S}}_d(\{\mathcal{F}\})$	tests	step 1 (s)	step 2 (s)	step 3 (s)	sol.
ShProd_4	$\{\Phi_0, \Phi_1\}$	6	$2.8 \cdot 10^2$	$2.8 \cdot 10^2$	2.8	$1.0 \cdot 10^{-2}$	$2.0 \cdot 10^{-2}$	83
ShProd_5	$\{\Phi_0, \Phi_1\}$	8	$6.3 \cdot 10^6$	$6.3 \cdot 10^6$	$1.8 \cdot 10^3$	3.9	$2.8 \cdot 10^2$	3,168

Table 4.3: Short product analysis

We observe in Table 4.3 that the cost of Step 2 and Step 3 grows linearly with the number of tests. The cost of Step 2 is indeed negligible for ShProd_5. For ShProd_4, it seems less clear: it is probably not precise enough for this magnitude.

map	total solutions	equivalence classes
ShProd_4	1,440	220
ShProd_5	146,944	11,424

Table 4.4: Vector spaces found for the short product

The last column of Table 4.4 describes the number of equivalence classes of vector spaces in $\mathscr{S}_r(T)$, with respect to the group $\mathrm{Stab}(T)$.

Example 4.3 (Example of a minimal set of non-scalar products for the short product modulo X^5)**.** *The matrices*

$$\begin{pmatrix} 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix},$$

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix},$$

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix},$$

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

generate, in the canonical basis, the coefficients of $\mathbf{\Phi}$*.*

4.2 A stem of the optimal decompositions for the circulant product modulo (X^5-1)

We present in this section how to find, with our approach, optimal decompositions of the polynomial product modulo (X^5-1) over $\mathbb{F}_2$. We denote by T the target space spanned by the coefficients Φ_i of the bilinear map

$$\mathbf{\Phi} : (A,B) \mapsto A \cdot B \bmod (X^5-1) = \begin{pmatrix} \Phi_0 \\ \Phi_1 \\ \Phi_2 \\ \Phi_3 \\ \Phi_4 \end{pmatrix} = \begin{pmatrix} a_4b_1 + a_3b_2 + a_2b_3 + a_1b_4 + a_0b_0 \\ a_4b_2 + a_3b_3 + a_2b_4 + a_1b_0 + a_0b_1 \\ a_4b_3 + a_3b_4 + a_2b_0 + a_1b_1 + a_0b_2 \\ a_4b_4 + a_3b_0 + a_2b_1 + a_1b_2 + a_0b_3 \\ a_4b_0 + a_3b_1 + a_2b_2 + a_1b_3 + a_0b_4 \end{pmatrix}.$$

The subspace T has the following structure:

$$T = \mathrm{Span}(\phi) \oplus H,$$

where $\phi = \Phi_0 + \Phi_1 + \Phi_2 + \Phi_3 + \Phi_4$ and H is a hyperplane containing all the bilinear forms of T of rank 4. For any $\psi \in H$ such that $\psi \neq 0$, we have $\mathrm{rk}(\psi + \phi) = 5$.

Moreover, the action of $\mathrm{Stab}(T)$ on $H - \{0\}$ is transitive (observed empirically), which means that all the elements of rank 4 are in the same orbit. Consequently, it is also transitive on $\mathrm{Span}(\phi) \oplus H$ and all the elements of rank 5 are in the same orbit.

Proposition 4.4 (Stem of the circulant product)**.** *The set* $\{\mathrm{Span}(\{\Phi_4\}), H\}$ *is a stem of* T*: for any basis* $\mathcal{B}$ *of* T*, there exists* $\mathcal{U} \subset \mathcal{B}$ *and* $\sigma \in \mathrm{Stab}(T)$ *such that*

$$\mathrm{Span}(\mathcal{U}) \circ \sigma = \mathrm{Span}(\Phi_4)$$

or

$$\mathrm{Span}(\mathcal{U}) = H.$$

We deduce from Proposition 4.4 that we need algorithms to compute the following sets:

- $\mathscr{S}_6(\mathrm{Span}(\Phi_4))$ and
- $\mathscr{S}_9(H)$.

Of course, using the stabilizer of T, we reduce our problem to the computation of

- $\tilde{\mathscr{S}}_6(\{\Phi_4\})$ and
- $\mathscr{S}_9(H)/\mathrm{Stab}(H)$ (any element $V \in \mathcal{E}_1$ satisfies $T \subset V + \mathrm{Span}(\phi) \in \mathscr{S}_{10}$).

The particularity of our strategy for the circulant product is that, in order to compute $\mathcal{E}_1$, we propose a stem of H.

Proposition 4.5 (Stem of H)**.** *The set* $\{(\Phi_0 + \Phi_1 + \Phi_2 + \Phi_3)\}$ *is a stem of* H*: for any basis* $\mathcal{B}$ *of* H*, there exists* $\mathcal{U} \subset \mathcal{B}$ *of cardinality* 1 *and* $\sigma \in \mathrm{Stab}(H) = \mathrm{Stab}(T)$ *such that*

$$\mathrm{Span}(\mathcal{U}) \circ \sigma = \mathrm{Span}(\Phi_0 + \Phi_1 + \Phi_2 + \Phi_3).$$

Proof. Any non zero element of H has rank 4 and is in the orbit of $\Phi_0 + \Phi_1 + \Phi_2 + \Phi_3$ for the action of $\mathrm{Stab}(H)$. □

Thus, we deduce the set $\mathscr{S}_9(H)/\mathrm{Stab}(H)$ from the enumeration of

$$\tilde{\mathscr{S}_6}(\{\Phi_0 + \Phi_1 + \Phi_2 + \Phi_3\}).$$

$\mathcal{F}$	d	$\#\tilde{\mathscr{S}_d}(\{\mathcal{F}\})$	tests	step 1 (s)	step 2 (s)	step 3 (s)	sol.
$\{\Phi_0\}$	6	$5.2 \cdot 10$	$8.7 \cdot 10^6$	$6.9 \cdot 10$	3.7	$3.1 \cdot 10^3$	0
$\{\sum_{0 \leq i < 4} \Phi_i\}$	6	$2.0 \cdot 10^3$	$6.7 \cdot 10^5$	$6.7 \cdot 10$	$5.0 \cdot 10$	$1.9 \cdot 10^2$	264

Table 4.5: Circulant product

We have in Table 4.5 the timings the procedure described in Section 3.3. The set $\mathscr{S}_{10}(T)$ contains 2,025 elements divided in 9 equivalence classes of solutions. Interestingly, the set $\{\Phi_0\}$ does not correspond to any element of $\mathscr{S}_{10}(T)$. It means that, for a basis $\mathcal{B}$ of bilinear forms of rank one containing ϕ and generating a subspace of $\mathscr{S}_{10}(T)$, the coordinate of the elements of rank 4 on ϕ is zero.

4.3 Application to matrix products

We denote by $\mathbf{\Phi}_{p,q,r}$ the bilinear map corresponding to the $p \times q$ by $q \times r$ matrix product:

$$\begin{array}{rccc} \mathbf{\Phi}_{p,q,r} : & \mathcal{M}_{p,q}(K) \times \mathcal{M}_{q,r}(K) & \longrightarrow & \mathcal{M}_{p,r}(K) \\ & (A, B) & \longmapsto & A \cdot B \end{array}.$$

We denote by $\Phi_{i,j}$ the bilinear forms such that $\Phi_{i,j}(A, B)$ is the coefficient (i, j) of $\mathbf{\Phi}_{p,q,r}(A, B)$ for $i \in \{0, \ldots, p-1\}, j \in \{0, \ldots, r-1\}$. The elements $\Phi_{i,j}$ satisfy $\Phi_{i,j}(A, B) = \sum_{0 \leq h < q} a_{i,h} b_{h,j}$.

The bilinear map $\mathbf{\Phi}_{p,q,r}$ is represented by a subspace of $\mathcal{L}(K^{pq}, K^{qr}; K)$ denoted by

$$T_{p,q,r} = \mathrm{Span}((\Phi_{i,j})_{i,j}).$$

In order to represent the elements of $T_{p,q,r}$ in terms of matrices of $\mathcal{M}_{pq,qr}$, we need an order on the $a_{i,h}$'s and $b_{h,j}$'s.

- For the $a_{i,h}$'s, we fix the following order: $a_{i,h} \leq a_{i',h'}$ if $i \leq i'$ or $i = i'$ and $h \leq h'$, which is the row-major order.
- For the $b_{h,j}$'s, we fix the following order: $b_{h,j} \leq b_{h',j'}$ if $j \leq j'$ or $j = j'$ and $h \leq h'$, which is the column-major order.

Then, in the bases of $\mathcal{M}_{p,q}$ and $\mathcal{M}_{q,r}$ given by the $a_{i,h}$'s and $b_{h,j}$'s ordered as above, the elements of $T_{p,q,r}$ can be represented as matrices of $\mathcal{M}_{pq,qr}$ divided in blocks of size $q \times q$ equal to I_q the identity matrix of $\mathcal{M}_{q,q}$. Consequently, this space is isomorphic to $\mathcal{M}_{p,r} \otimes I_q$ and all the elements of $T_{p,q,r}$ have a rank which is multiple of q.

Example 4.6 (Matrix representation of elements of $T_{2,2,2}$)**.** *The elements of $T_{2,2,2}$ are represented by matrices of $\mathcal{M}_{4,4}$ spanned by*

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

corresponding to the coefficients $a_{0,0}b_{0,0} + a_{0,1}b_{1,0}$, $a_{0,0}b_{0,1} + a_{0,1}b_{1,1}$, $a_{1,0}b_{0,0} + a_{1,1}b_{1,0}$ and $a_{1,0}b_{0,1} + a_{1,1}b_{1,1}$, respectively. The previous matrices can also be expressed as

$$\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \otimes I_2, \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \otimes I_2, \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} \otimes I_2, \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \otimes I_2,$$

respectively.

Let (e_i), (f_h) and (g_j) be the canonical bases of K^p, K^q and K^r. The subspace $T_{p,q,r}$ can be easily characterized with the tensor notation: it is generated by the vectors, for $i \in \{0, \dots, p-1\}, j \in \{0, \dots, r-1\}$,

$$\Phi_{i,j} = \sum_{0 \le h < q} e_i \otimes f_h \otimes f_h \otimes g_j.$$

Theorem 4.7. *Stabilizer of the matrix product For the group action $M \cdot (X, Y) \mapsto X^{\mathrm{T}}MY$, the subgroup stabilizing the vector space $T_{p,q,r}$ can be described as the group given by the pairs $(P \otimes R^{\mathrm{T}}, Q \otimes (R^{-1}))$ for $P \in \mathrm{GL}_p$, $R \in \mathrm{GL}_q$, and $Q \in \mathrm{GL}_r$.*

Proof. We denote by $T_{p,q,r}$ the vector space given by the product of matrices $p \times q$ by $q \times r$, which is isomorphic to $\mathcal{M}_{p,r} \otimes I_q$ (we do not use the canonical basis for this representation). For the group action $M \cdot (X, Y) \mapsto X^{\mathrm{T}}MY$, we want to prove that the subgroup stabilizing the vector space $T_{p,q,r}$ is isomorphic to $\mathrm{Stab}(\mathcal{M}_{p,r}) \otimes \mathrm{Stab}(I_q)$.

Let (X, Y) be a pair of invertible matrices such that $X^{\mathrm{T}}T_{p,q,r}Y = T_{p,q,r}$. For any $i \in \{0, \dots, p-1\}$ and $j \in \{0, \dots, q-1\}$, we denote by $M_{i,j}$ the matrix $X^{\mathrm{T}} \cdot (e_{i,j}) \cdot Y$, where $e_{i,j}$ is the canonical basis of $\mathcal{M}_{p,r}$. Denoting by $X_{i,h}$ the $q \times q$ blocks of X and $Y_{\ell,j}$ the $q \times q$ blocks of Y, we have $M_{i,j} = (X_{i,h}Y_{j,\ell})_{h,\ell}$ for any i and j. Consequently, since $X^{\mathrm{T}} \cdot (e_{i,j}) \cdot Y \in T_{p,q,r}$, we have

$$\forall i, j, h, \ell, \ X_{i,h}Y_{j,\ell} \in \mathrm{Span}(\{I_q\}). \tag{4.2}$$

Let (i, h) such that $X_{i,h}$ is not null and j any integer in $\{0, \dots, q-1\}$. We have the inclusion

$$X_{i,h} \cdot \mathrm{Span}(\{Y_{j,0}, \dots, Y_{j,q-1}\}) \subset \mathrm{Span}(\{I_q\})$$

and, since Y is invertible, we even have the equality. Thus, for any (i, h) such that $X_{i,h}$ is not null, we have shown that $X_{i,h}$ is invertible. We have the same property for the blocks of Y.

Combining the fact that the blocks of X and Y that are not null are invertible and Equation (4.2), we can conclude that the stabilizer of $T_{p,q,r}$ is generated by matrices (X, Y) such that there exists $g \in \mathrm{GL}_q$ satisfying

$$X^{\mathrm{T}} \in \mathrm{GL}_p \otimes g \text{ and } Y \in \mathrm{GL}_r \otimes g^{-1}.$$

□

Corollary 4.8. *The elements of $T_{p,q,r}$ of a given rank lie in the same orbit under the action of* $\mathrm{Stab}(T_{p,q,r})$.

Example 4.9 (Action of the stabilizer of $T_{2,2,2}$)**.** *The stabilizer of $T_{2,2,2}$ is generated by the following elements of* $\mathrm{GL}(K^4) \times \mathrm{GL}(K^4)$*:*

$$\left(\begin{pmatrix}1&0&1&0\\0&1&0&1\\0&0&1&0\\0&0&0&1\end{pmatrix},\begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{pmatrix}\right),\left(\begin{pmatrix}0&0&1&0\\0&0&0&1\\1&0&0&0\\0&1&0&0\end{pmatrix},\begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{pmatrix}\right),$$

$$\left(\begin{pmatrix}1&1&0&0\\0&1&0&0\\0&0&1&1\\0&0&0&1\end{pmatrix},\begin{pmatrix}1&1&0&0\\0&1&0&0\\0&0&1&1\\0&0&0&1\end{pmatrix}\right),\left(\begin{pmatrix}0&1&0&0\\1&0&0&0\\0&0&0&1\\0&0&1&0\end{pmatrix},\begin{pmatrix}0&1&0&0\\1&0&0&0\\0&0&0&1\\0&0&1&0\end{pmatrix}\right),$$

$$\left(\begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{pmatrix},\begin{pmatrix}1&0&1&0\\0&1&0&1\\0&0&1&0\\0&0&0&1\end{pmatrix}\right),\left(\begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{pmatrix},\begin{pmatrix}0&0&1&0\\0&0&0&1\\1&0&0&0\\0&1&0&0\end{pmatrix}\right).$$

The vector space of $T_{2,2,2}$ is isomorphic to $\mathcal{M}_{2,2} \otimes I_2$. Thus the elements of $T_{2,2,2}$ have rank 0, 2 *or* 4.

Via the action of $\mathrm{Stab}(T_{2,2,2})$*, all the elements of rank* 2 *can all be mapped to the element*

$$\begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&0&0\\0&0&0&0\end{pmatrix}.$$

Similarly, via the action of $\mathrm{Stab}(T_{2,2,2})$*, all the elements of rank* 4 *can all be mapped to the element*

$$\begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{pmatrix}.$$

4.3.1 Decomposition of the matrix product $(3, 2, 3)$

The rank of this bilinear map is known [34]: it has rank 15. However, all the optimal formulas are not known. The target subspace of $\mathcal{L}(K^6, K^6; K)$ considered is denoted by

$$T = \mathrm{Span}(\{\Phi_{i,j} \mid i, j \in \{0, 1, 2\}\}).$$

The elements of T can be represented as matrices of $\mathcal{M}_{6,6}$ divided in blocks of size 2×2 equal to

$$\begin{pmatrix} \delta & 0 \\ 0 & \delta \end{pmatrix}, \delta \in K.$$

We produce a stem of T given by five subspaces of T.

Proposition 4.10 (Stem of the matrix product)**.** *The set*

$$\mathcal{C} = \{\mathrm{Span}(\{\Phi_{0,0}+\Phi_{1,1}+\Phi_{2,2}\}), \mathrm{Span}(\{\Phi_{0,0}+\Phi_{1,1}, \Phi_{0,1}+\Phi_{2,2}\}), \mathrm{Span}(\{\Phi_{0,0}+\Phi_{1,1}, \Phi_{1,1}+\Phi_{2,2}\}),$$
$$\mathrm{Span}(\{\Phi_{0,0} + \Phi_{1,1}, \Phi_{2,2}\}), \mathrm{Span}(\{\Phi_{0,0}, \Phi_{1,1}, \Phi_{2,2}\})\}$$

is a stem of T: for any basis $\mathcal{B}$ of T, there exists $\mathcal{U} \subset \mathcal{B}$ and $\sigma \in \mathrm{Stab}(T)$ such that

$$\mathrm{Span}(\mathcal{U}) \circ \sigma \in \mathcal{C}.$$

Proof. Let $\mathcal{B}$ be a basis of T.

- If there exists an element Φ of rank 6 in $\mathcal{B}$, then there exists $\sigma \in \mathrm{Stab}(T)$ such that $\Phi_{0,0} + \Phi_{1,1} + \Phi_{2,2} \in \mathcal{B} \circ \sigma$. Otherwise, any element Φ of $\mathcal{B}$ has rank smaller or equal to 4 and we have to distinguish two cases.
- If there exists an element Φ of rank 4, there exists σ such that $\Phi_{0,0} + \Phi_{1,1} \in \mathcal{B} \circ \sigma$ and, consequently, there exists another element $\Phi' \in \mathcal{B}$ of rank 2 or 4 whose coordinate over $\Phi_{2,2}$ in the basis $(\Phi_{i,j})_{i,j}$ is not null: we need to look at the possible orbits in which Φ' is included under the action of the subgroup of $\mathrm{Stab}(T)$ preserving the fact that Φ is in the orbit of $\Phi_{0,0} + \Phi_{1,1}$. We can prove that there exist 3 such orbits and that there exists $\sigma \in \mathrm{Stab}(T)$ and $\mathcal{U} \subset \mathcal{B}$ of cardinality 2 such that

$$\mathcal{U} \circ \sigma = \begin{cases} \{\Phi_{0,0} + \Phi_{1,1}, \Phi_{0,1} + \Phi_{2,2}\} \\ \text{or} \\ \{\Phi_{0,0} + \Phi_{1,1}, \Phi_{1,1} + \Phi_{2,2}\} \\ \text{or} \\ \{\Phi_{0,0} + \Phi_{1,1}, \Phi_{2,2}\}. \end{cases}$$

- All the elements of $\mathcal{B}$ have rank 2 and there exists $\mathcal{U} \subset \mathcal{B}$ and $\sigma \in \mathrm{Stab}(T)$ such that

$$\mathcal{U} \circ \sigma = \{\Phi_{0,0}, \Phi_{1,1}, \Phi_{2,2}\}.$$

□

In conclusion, we consider 5 intermediate sets:

- $\tilde{\mathscr{S}_7}(\{\Phi_{0,0}+\Phi_{1,1}+\Phi_{2,2}\})$, set of subspaces containing the diagonal element $\Phi_{0,0}+\Phi_{1,1}+\Phi_{2,2}$,

- $\tilde{\mathscr{S}}_8(\{\Phi_{0,0}+\Phi_{1,1}, \Phi_{0,1}+\Phi_{2,2}\})$, set of subspaces containing $\Phi_{0,0}+\Phi_{1,1}$ and $\Phi_{0,1}+\Phi_{2,2}$,
- $\tilde{\mathscr{S}}_8(\{\Phi_{0,0}+\Phi_{1,1}, \Phi_{1,1}+\Phi_{2,2}\})$, set of subspaces containing $\Phi_{0,0}+\Phi_{1,1}$ and $\Phi_{1,1}+\Phi_{2,2}$,
- $\tilde{\mathscr{S}}_8(\{\Phi_{0,0}+\Phi_{1,1}, \Phi_{2,2}\})$, set of subspaces containing $\Phi_{0,0}+\Phi_{1,1}$ and $\Phi_{2,2}$,
- $\tilde{\mathscr{S}}_9(\{\Phi_{0,0}, \Phi_{1,1}, \Phi_{2,2}\})$, set of subspaces containing $\Phi_{0,0}$, $\Phi_{1,1}$ and $\Phi_{2,2}$.

Moreover, being able to decompose a matrix product of larger dimensions requires to improve on the theoretical aspect of our strategy, since the size of the required set

$$\mathscr{S}_{15}/\mathrm{GL}(K^9)\times\mathrm{GL}(K^9)$$

is expected to be too large, based on the apparent exponential growth of the progression of the sets described in Table 5.2.

As described in Section 3.3, we need to precompute the quotients

$$\mathscr{S}_{6+k}/\mathrm{GL}(K^{6+k})\times\mathrm{GL}(K^{6+k})$$

for $k\in\{1,2,3\}$, and we prove that we can restrict ourselves to subspaces containing at least one element of rank 6. The techniques for computing theses subsets are described in Chapter 5.

The sets returned by IntermediateSetViaQuotientComputation in Section 3.3, are computed in $1.6\cdot 10^5$ seconds. They are defined as the following sets: We describe in Table 4.6 how we apply our method to the sets

- $\{\Phi_{0,0}+\Phi_{1,1}+\Phi_{2,2}\}$,
- $\{\Phi_{0,0}+\Phi_{1,1}, \Phi_{0,1}+\Phi_{2,2}\}$,
- $\{\Phi_{0,0}+\Phi_{1,1}, \Phi_{1,1}+\Phi_{2,2}\}$,
- $\{\Phi_{0,0}+\Phi_{1,1}, \Phi_{2,2}\}$.

$\mathcal{F}$	d	$\#\tilde{\mathscr{S}}_d(\{\mathcal{F}\})$	tests	step 1 (s)	step 2 (s)	step 3 (s)	sol.
$\left\{\begin{matrix}\Phi_{0,0}+\Phi_{1,1}\\ +\Phi_{2,2}\end{matrix}\right\}$	7	$8.8\cdot 10$	$1.2\cdot 10^8$	2.7	$2.8\cdot 10^4$	$2.0\cdot 10^5$	5
$\left\{\begin{matrix}\Phi_{0,0}+\Phi_{1,1},\\ \Phi_{0,1}+\Phi_{2,2}\end{matrix}\right\}$	8	$7.5\cdot 10^5$	$2.2\cdot 10^7$	$1.2\cdot 10^3$	$1.2\cdot 10^4$	$3.3\cdot 10^5$	13
$\left\{\begin{matrix}\Phi_{0,0}+\Phi_{1,1},\\ \Phi_{1,1}+\Phi_{2,2}\end{matrix}\right\}$	8	$1.0\cdot 10^4$	$2.8\cdot 10^5$	$9.3\cdot 10^2$	9.9	$4.1\cdot 10^2$	1
$\left\{\begin{matrix}\Phi_{0,0}+\Phi_{1,1},\\ \Phi_{2,2}\end{matrix}\right\}$	8	$2.7\cdot 10^5$	$5.9\cdot 10^8$	$5.3\cdot 10^2$	$4.9\cdot 10^4$	$9.1\cdot 10^5$	46

Table 4.6: Matrix product analysis

Instead of computing $\tilde{\mathscr{S}}_9(\{\Phi_{0,0}, \Phi_{1,1}, \Phi_{2,2}\})$, we actually compute a subset $\mathcal{E}'$ given by a special trick described in Section 4.3.2.

$\mathcal{F}$	d	$\#\mathcal{E}'$	tests	step 1 (s)	step 2 (s)	step 3 (s)	sol.
$\{\Phi_{0,0}, \Phi_{1,1}, \Phi_{2,2}\}$	9	$2.5 \cdot 10^7$	$9.1 \cdot 10^8$	$1.5 \cdot 10^5$	$4.1 \cdot 10^3$	$1.3 \cdot 10^6$	2

Table 4.7: Timings for $\mathcal{E}'$

In conclusion, we are able to decompose $\mathbf{\Phi}_{3,2,3}$ over $\mathbb{F}_2$ and to give all the possible optimal decompositions. We have a speed-up of 10^4 compared to our implementation of Algorithm BDEZStab. Although the rank of this bilinear map was already known thanks to Hopcroft and Kerr [34], determining all the possible optimal decompositions was not a well studied problem to our knowledge. Furthermore, we obtain an alternative proof over $K = \mathbb{F}_2$ of the fact that the bilinear rank of $\mathbf{\Phi}_{3,2,3}$ is 15.

We prove with our algorithm that there is only one class of equivalence of vector spaces $W \in \mathscr{S}_{6,6,15}$ containing T, for the group action induced by $\mathrm{Stab}(T)$. It is interesting to note that this is also the case for $T_{2,2,2}$. We do not have this kind of result for the short product for example.

The matrices

$$\begin{pmatrix} 0&0&0&0&0&0\\ 1&1&1&1&0&0\\ 0&0&0&0&0&0\\ 1&1&1&1&0&0\\ 0&0&0&0&0&0\\ 0&0&0&0&0&0 \end{pmatrix}, \begin{pmatrix} 0&0&0&0&1&0\\ 0&0&0&0&0&0\\ 0&0&0&0&1&0\\ 0&0&0&0&0&0\\ 0&0&0&0&1&0\\ 0&0&0&0&0&0 \end{pmatrix}, \begin{pmatrix} 0&0&0&0&0&0\\ 0&0&0&0&0&0\\ 0&0&0&0&0&0\\ 0&0&1&1&0&0\\ 0&0&0&0&0&0\\ 0&0&1&1&0&0 \end{pmatrix},$$

$$\begin{pmatrix} 0&0&1&1&0&1\\ 0&0&0&0&0&0\\ 0&0&0&0&0&0\\ 0&0&0&1&1&1\\ 0&0&0&0&0&0\\ 0&0&0&1&1&1 \end{pmatrix}, \begin{pmatrix} 1&0&1&1&0&1\\ 1&0&1&1&0&1\\ 0&0&0&0&0&0\\ 1&0&1&1&0&1\\ 0&0&0&0&0&0\\ 0&0&0&0&0&0 \end{pmatrix}, \begin{pmatrix} 0&0&0&0&0&0\\ 1&1&1&1&0&1\\ 0&0&0&0&0&0\\ 1&1&1&1&0&1\\ 1&1&1&1&0&1\\ 0&0&0&0&0&0 \end{pmatrix},$$

$$\begin{pmatrix} 0&0&1&1&1&1\\ 0&0&0&0&0&0\\ 0&0&0&0&0&0\\ 0&0&0&0&0&0\\ 0&0&0&0&0&0\\ 0&0&0&0&0&0 \end{pmatrix}, \begin{pmatrix} 0&0&0&0&0&0\\ 0&0&0&0&0&1\\ 0&0&0&0&0&0\\ 0&0&0&0&0&1\\ 0&0&0&0&0&0\\ 0&0&0&0&0&1 \end{pmatrix}, \begin{pmatrix} 0&0&0&0&0&0\\ 0&0&0&0&0&0\\ 0&0&0&0&0&0\\ 0&0&0&0&0&0\\ 0&1&0&1&0&1\\ 0&1&0&1&0&1 \end{pmatrix},$$

$$\begin{pmatrix} 1&0&1&0&0&0\\ 1&0&1&0&0&0\\ 1&0&1&0&0&0\\ 1&0&1&0&0&0\\ 0&0&0&0&0&0\\ 0&0&0&0&0&0 \end{pmatrix}, \begin{pmatrix} 0&0&0&1&0&1\\ 0&0&0&1&0&1\\ 0&0&0&0&0&0\\ 0&0&0&0&0&0\\ 0&0&0&0&0&0\\ 0&0&0&0&0&0 \end{pmatrix}, \begin{pmatrix} 0&0&0&0&0&0\\ 0&0&0&0&0&0\\ 0&0&0&0&0&0\\ 0&0&0&0&0&0\\ 1&1&1&1&1&1\\ 0&0&0&0&0&0 \end{pmatrix},$$

$$\begin{pmatrix} 0&0&0&0&0&0\\ 0&0&0&0&0&0\\ 0&0&0&0&0&0\\ 0&1&1&1&0&1\\ 0&1&1&1&0&1\\ 0&1&1&1&0&1 \end{pmatrix}, \begin{pmatrix} 0&0&0&0&0&0\\ 0&0&0&0&0&0\\ 0&0&0&0&0&0\\ 0&1&0&0&0&1\\ 0&0&0&0&0&0\\ 0&0&0&0&0&0 \end{pmatrix}, \begin{pmatrix} 0&0&0&0&0&0\\ 0&0&0&0&0&0\\ 0&0&1&0&0&0\\ 0&0&1&0&0&0\\ 0&0&1&0&0&0\\ 0&0&1&0&0&0 \end{pmatrix}$$

form a set of non-scalar products leading to optimal formulas for the matrix product. We retrieve all the subspaces of $\mathscr{S}_{15}(T)$ via the action of the group $\mathrm{Stab}(T)$.

4.3.2 Using the Hamming weight for the matrix product

We describe in this section a trick allowing one to speed-up the execution of our approach for the matrix product. However, this part is technical and can be skipped on a first read.

We still denote by T the subspace of $\mathcal{L}(K^6, K^6; K)$ corresponding to the coefficients of the product of 3×2 by 2×3 matrices. We remind the stem of T that we consider: the set

$$\mathcal{C} = \{\mathrm{Span}(\{\Phi_{0,0}+\Phi_{1,1}+\Phi_{2,2}\}), \mathrm{Span}(\{\Phi_{0,0}+\Phi_{1,1}, \Phi_{0,1}+\Phi_{2,2}\}), \mathrm{Span}(\{\Phi_{0,0}+\Phi_{1,1}, \Phi_{1,1}+\Phi_{2,2}\}),$$
$$\mathrm{Span}(\{\Phi_{0,0}+\Phi_{1,1}, \Phi_{2,2}\}), \mathrm{Span}(\{\Phi_{0,0}, \Phi_{1,1}, \Phi_{2,2}\})\}.$$

We define the following sets:

- $\mathcal{E}_0 = \mathscr{S}_7(\mathrm{Span}(\Phi_{0,0} + \Phi_{1,1} + \Phi_{2,2}))$,
- $\mathcal{E}_1 = \mathscr{S}_8(\mathrm{Span}(\Phi_{0,0} + \Phi_{1,1}, \Phi_{0,1} + \Phi_{2,2}))$,
- $\mathcal{E}_2 = \mathscr{S}_8(\mathrm{Span}(\Phi_{0,0} + \Phi_{1,1}, \Phi_{1,1} + \Phi_{2,2}))$,
- $\mathcal{E}_3 = \mathscr{S}_8(\mathrm{Span}(\Phi_{0,0} + \Phi_{1,1}, \Phi_{2,2}))$ and
- $\mathcal{E}_4 = \mathscr{S}_9(\mathrm{Span}(\Phi_{0,0}, \Phi_{1,1}, \Phi_{2,2}))$.

In theory, we have to enumerate the elements of the sets $\tilde{\mathscr{S}_7}(\{\Phi_{0,0} + \Phi_{1,1} + \Phi_{2,2}\})$, $\tilde{\mathscr{S}_8}(\{\Phi_{0,0} + \Phi_{1,1}, \Phi_{0,1}+\Phi_{2,2}\})$, $\tilde{\mathscr{S}_8}(\{\Phi_{0,0}+\Phi_{1,1}, \Phi_{1,1}+\Phi_{2,2}\})$, $\tilde{\mathscr{S}_8}(\{\Phi_{0,0}+\Phi_{1,1}, \Phi_{2,2}\})$ and $\tilde{\mathscr{S}_9}(\{\Phi_{0,0}, \Phi_{1,1}, \Phi_{2,2}\})$, denoted by $\tilde{\mathcal{E}}_0$, $\tilde{\mathcal{E}}_1$, $\tilde{\mathcal{E}}_2$, $\tilde{\mathcal{E}}_3$ and $\tilde{\mathcal{E}}_4$, respectively. However, one should notice that, given $V \in \mathscr{S}_{15}(T)$ such that

$$\exists W \in \tilde{\mathcal{E}}_4, \sigma \in \mathrm{Stab}(\{\Phi_{0,0}, \Phi_{1,1}, \Phi_{2,2}\}),\ V = T + W \circ \sigma,$$

it may happen that there exists $W' \subset V$ such that

$$V = T + W'$$

and

$$\exists \sigma \in \mathrm{Stab}(T),\ W' \circ \sigma = \begin{cases} \mathscr{S}_7(\mathrm{Span}(\Phi_{0,0} + \Phi_{1,1} + \Phi_{2,2})) \\ \text{or} \\ \mathscr{S}_8(\mathrm{Span}(\Phi_{0,0} + \Phi_{1,1}, \Phi_{0,1} + \Phi_{2,2})) \\ \text{or} \\ \mathscr{S}_8(\mathrm{Span}(\Phi_{0,0} + \Phi_{1,1}, \Phi_{1,1} + \Phi_{2,2})) \\ \text{or} \\ \mathscr{S}_8(\mathrm{Span}(\Phi_{0,0} + \Phi_{1,1}, \Phi_{2,2})) \end{cases}.$$

In other terms, the V's corresponding to the 5 sets to enumerate do not form a partition of $\mathscr{S}_{15}(T)$.

Thus, we propose, if possible, to enumerate a subset of $\mathcal{E}_4$, rather than the whole set, without losing exhaustivity. The strategy that is proposed is related to the notion of Hamming weight of the elements $\Phi_{0,0}$, $\Phi_{1,1}$ and $\Phi_{2,2}$.

Definition 4.11 (Hamming weight for $\mathscr{S}_d$)**.** *Let $W \in \mathscr{S}_d$ and $\mathcal{B} = (\psi_0, \dots, \psi_{d-1})$ a basis of rank-one bilinear forms of W. Any $x \in W$ has a unique decomposition over $\mathcal{B}$:*

$$x = \sum_t \lambda_t \cdot \psi_t.$$

We define its Hamming weight over $\mathcal{B}$ as

$$\mathbb{H}_{\mathcal{B}}(x) = \#\{t \in \{0, \dots, d-1\} \mid \lambda_t \neq 0\}.$$

We can extend the definition of the Hamming weight to any subset $\mathcal{S}$ of W:

$$\mathbb{H}_{\mathcal{B}}(\mathcal{S}) = \min(\{\#I \mid I \subset \{0, \dots, d-1\}, \mathcal{S} \subset \mathrm{Span}((\psi_t)_{t \in I})\}).$$

The Hamming weight over some basis has a useful property related to the bilinear rank stated in Lemma 4.12.

Lemma 4.12. *Let $W \in \mathscr{S}_d$ and $\mathcal{B}$ a basis of W composed of rank-one bilinear forms. For any subset $\mathcal{S}$ of W, we have*

$$\mathrm{rk}(\mathrm{Span}(\mathcal{S})) \leq \mathbb{H}_{\mathcal{B}}(\mathcal{S}).$$

Proof. Clear from the definition of the rank of a set $\mathcal{S}$ given in Definition 1.12. □

We describe in Theorem 4.13 what is the subset of $\tilde{\mathcal{E}}_4$ that we consider.

Theorem 4.13. *Let W be a subspace such that $W \in \tilde{\mathcal{E}}_4$ and $T + W \in \mathscr{S}_{15}$ and let $\mathcal{B}$ be a basis of W composed of rank-one bilinear forms. Let $\tilde{\mathcal{E}}'$ be the subset of elements $W \in \tilde{\mathcal{E}}_4$ such that*

$$\mathbb{H}_{\mathcal{B}}(\Phi_{0,0} + \Phi_{1,1} + \Phi_{2,2}) = \mathbb{H}_{\mathcal{B}}(\Phi_{0,0}) + \mathbb{H}_{\mathcal{B}}(\Phi_{1,1}) + \mathbb{H}_{\mathcal{B}}(\Phi_{2,2}) \text{ and } \mathbb{H}_{\mathcal{B}}(\Phi_{0,0} + \Phi_{1,1} + \Phi_{2,2}) > 6.$$

We obtain all the elements of $\mathscr{S}_{15}(T)$ via the enumeration of $\tilde{\mathcal{E}}_0$, $\tilde{\mathcal{E}}_1$, $\tilde{\mathcal{E}}_2$, $\tilde{\mathcal{E}}_3$ and $\tilde{\mathcal{E}}'$.

We prove Theorem 4.13 within 2 steps:

1. We prove in Lemma 4.14 that if $\mathbb{H}_{\mathcal{B}}(\Phi_{0,0}+\Phi_{1,1}+\Phi_{2,2}) \neq \mathbb{H}_{\mathcal{B}}(\Phi_{0,0})+\mathbb{H}_{\mathcal{B}}(\Phi_{1,1})+\mathbb{H}_{\mathcal{B}}(\Phi_{2,2})$, a subspace V obtained as $V = T + W \circ \sigma$ can also be obtained as $V = T + W' \circ \sigma$, with $W' \in \tilde{\mathcal{E}}_0$, $\tilde{\mathcal{E}}_1$ or $\tilde{\mathcal{E}}_3$.

2. Otherwise, if $\mathbb{H}_{\mathcal{B}}(\Phi_{0,0}+\Phi_{1,1}+\Phi_{2,2}) = \mathbb{H}_{\mathcal{B}}(\Phi_{0,0})+\mathbb{H}_{\mathcal{B}}(\Phi_{1,1})+\mathbb{H}_{\mathcal{B}}(\Phi_{2,2})$, it remains to prove that we do not lose in generality if we assume that $\mathbb{H}_{\mathcal{B}}(\Phi_{0,0} + \Phi_{1,1} + \Phi_{2,2}) > 6$, which is done in Lemma 4.15.

Lemma 4.14. *Let $W \in \mathscr{S}_9$ and let $V = T + W$. We assume that $T + W \in \mathscr{S}_{15}(T)$ and $\mathrm{Span}(\{\Phi_{0,0}, \Phi_{1,1}, \Phi_{2,2}\}) \subset W$. Let $\mathcal{B}$ be a basis of rank-one bilinear forms of W. If $\mathbb{H}_{\mathcal{B}}(\Phi_{0,0} + \Phi_{1,1}+\Phi_{2,2}) \neq \mathbb{H}_{\mathcal{B}}(\Phi_{0,0})+\mathbb{H}_{\mathcal{B}}(\Phi_{1,1})+\mathbb{H}_{\mathcal{B}}(\Phi_{2,2})$, there exists $W' \subset W$ such that $V = T+W'$ and there exists $\sigma' \in \mathrm{Stab}(T)$ such that $W' \circ \sigma' \in \mathcal{E}_0, \mathcal{E}_1$ or $\mathcal{E}_3$.*

Proof. We have by hypothesis $\mathbb{H}_{\mathcal{B}}(\Phi_{0,0} + \Phi_{1,1} + \Phi_{2,2}) < \mathbb{H}_{\mathcal{B}}(\Phi_{0,0}) + \mathbb{H}_{\mathcal{B}}(\Phi_{1,1}) + \mathbb{H}_{\mathcal{B}}(\Phi_{2,2})$. Thus, there exist two elements $\Psi \in \mathcal{B}$ and $\Phi \in \{\Phi_{0,0}, \Phi_{1,1}, \Phi_{2,2}\}$ such that the coordinate of Φ on Ψ is not zero and the coordinates of $\Phi_{0,0} + \Phi_{1,1} + \Phi_{2,2}$ on Ψ is zero. By considering the vector space $W' = \mathrm{Span}(\mathcal{B} - \{\Psi\})$, we have $W' \in \mathscr{S}_8$. Moreover, we have $W = \mathrm{Span}(\{\Psi\}) \oplus W' = \mathrm{Span}(\{\Phi\}) \oplus W'$ and $T + (\mathrm{Span}(\{\Phi\}) \oplus W') =\subset T + W'$. Thus, $\dim(T+W') = \dim(T+W) = 15$. Consequently, $\dim(T \cap W') = 2$.

If there exists in $T \cap W'$ two elements Φ_1 and Φ_2 of rank smaller or equal to 4 such that $\Phi_1 + \Phi_2 = \Phi_{0,0} + \Phi_{1,1} + \Phi_{2,2}$, then

$$\exists \sigma \in \mathrm{Stab}(T),\ W' \circ \sigma \in \begin{cases} \mathscr{S}_8(\mathrm{Span}(\Phi_{0,0} + \Phi_{1,1}, \Phi_{0,1} + \Phi_{2,2})) \\ \text{or} \\ \mathscr{S}_8(\mathrm{Span}(\Phi_{0,0} + \Phi_{1,1}, \Phi_{2,2})) \end{cases}.$$

Otherwise, there exists $W'' \subset W'$ such that

$$\exists \sigma \in \mathrm{Stab}(T),\ W'' \circ \sigma \in \mathscr{S}_7(\mathrm{Span}(\Phi_{0,0} + \Phi_{1,1} + \Phi_{2,2}))$$

and $T + W'' \in \mathscr{S}_{15}$, which concludes. □

Lemma 4.15. *Let $V \in \mathscr{S}_{15}(T)$. The subspace V satisfies hypotheses **H1** and **H2** state as follows.*

H1: *For any $W \subset V$ such that there exists $\sigma \in \mathrm{Stab}(T)$ satisfying $W \circ \sigma \in \mathscr{S}_9(\mathrm{Span}(\Phi_{0,0}, \Phi_{1,1}, \Phi_{2,2}))$ and $T + W \in \mathscr{S}_{15}$, we have, for any basis $\mathcal{B}$ of rank-one bilinear forms of W,*

$$\mathbb{H}_{\mathcal{B}}(\Phi_{0,0} + \Phi_{1,1} + \Phi_{2,2}) = \mathbb{H}_{\mathcal{B}}(\Phi_{0,0}) + \mathbb{H}_{\mathcal{B}}(\Phi_{1,1}) + \mathbb{H}_{\mathcal{B}}(\Phi_{2,2}).$$

H2: *There do not exist $W \subset V$ and $\sigma \in \mathrm{Stab}(T)$ such that $W \circ \sigma \in \mathcal{E}_0, \mathcal{E}_1$, $\mathcal{E}_2$ or $\mathcal{E}_3$ and $V = T + W$ (in other terms, V can not be obtained via the enumeration of $\tilde{\mathcal{E}}_0$, $\tilde{\mathcal{E}}_1$, $\tilde{\mathcal{E}}_2$ or $\tilde{\mathcal{E}}_3$).*

Then, there exists $W' \subset V$ and $\sigma' \in \mathrm{Stab}(T)$ such that $W' \circ \sigma' \in \mathscr{S}_9(\mathrm{Span}(\Phi_{0,0}, \Phi_{1,1}, \Phi_{2,2}))$, $T + W' \in \mathscr{S}_{15}$, and W' has a basis $\mathcal{B}'$ of rank-one bilinear forms such that

$$\mathbb{H}_{\mathcal{B}' \circ \sigma'}(\Phi_{0,0} + \Phi_{1,1} + \Phi_{2,2}) > 6.$$

Proof. Let $W \in \mathscr{S}_6$ be such that $T \oplus W \in \mathscr{S}_{15}$. Take a basis $\mathcal{W}$ of W, and complete it into a basis $\mathcal{B}$ of $T \oplus W$ using 9 rank-one bilinear forms, denoted by $\{\psi_i\}_{0 \leq i < 9}$. For all $i \in \{0, \ldots, 8\}$, write $\psi_i = \Phi_i + \Psi_i$, with $\Phi_i \in T$ and $\Psi_i \in W$. The Φ_i's form a basis of T. In our context, we

are concerned by the case $\mathrm{rk}(\Phi_i) = 2$ for any i (otherwise **H2** is not satisfied). Since we assume Hypothesis **H1**, it is enough to prove that there exists i such that $\mathbb{H}_{\mathcal{B}}(\Phi_i) > 2$.

There is necessarily a couple $(\Phi_i, \Phi_{i'})$ such that

$$\mathbb{H}_{\mathcal{B}}(\Phi_i + \Phi_{i'}) < \mathbb{H}_{\mathcal{B}}(\Phi_i) + \mathbb{H}_{\mathcal{B}}(\Phi_{i'}).$$

Otherwise, we would have $\#\mathcal{B} \geq 2 \cdot 9 = 18 \neq 15 = \mathrm{rk}(T)$.

If $\mathbb{H}_{\mathcal{B}}(\Phi_i) = 2$ and $\mathbb{H}_{\mathcal{B}}(\Phi_{i'}) = 2$, then

$$\mathbb{H}_{\mathcal{B}}(\{\Phi_i, \Phi_{i'}\}) \leq 3$$

and, by Lemma 4.12, $\mathrm{rk}(\{\Phi_i, \Phi_{i'}\}) \leq 3$. Henceforth, we prove that this is contradictory, because $\mathrm{rk}(\{\Phi_i, \Phi_{i'}\}) = 4$. Indeed, there are two cases.

- If $\Phi_i + \Phi_{i'}$ has rank 4, the conclusion follows.
- If $\mathrm{Span}(\Phi_i, \Phi_{i'})$ is isomorphic to $T_{2,2,1}$, whose rank is equal to 4: $T_{2,2,1}$ and $T_{2,1,2}$ have the same rank according to [22] and $T_{2,1,2}$ is a vector space of dimension one generated by a bilinear form of rank 4.

Consequently, $\mathbb{H}_{\mathcal{B}}(\Phi_i) > 2$ or $\mathbb{H}_{\mathcal{B}}(\Phi_{i'}) > 2$. □

4.3.3 Matrix product $(2, 3, 2)$

We denote by $\mathbf{\Phi}_{2,3,2}$ the bilinear map

$$\begin{array}{rccc} \mathbf{\Phi}_{2,3,2}: & \mathcal{M}_{2,3}(\mathbb{F}_2) \times \mathcal{M}_{3,2}(\mathbb{F}_2) & \longrightarrow & \mathcal{M}_{2,2}(\mathbb{F}_2) \\ & (A, B) & \longmapsto & A \cdot B \end{array}$$

and $\Phi_{i,j}$ its coefficients. We denote by T the corresponding vector space.

Proposition 4.16 (Stem of the matrix product 2×3 by 3×2)**.** *The set*

$$\mathcal{C} = \{\mathrm{Span}(\{\Phi_{0,0} + \Phi_{1,1}, \Phi_{0,0} + \Phi_{0,1} + \Phi_{1,0}\}), \mathrm{Span}(\{\Phi_{0,0} + \Phi_{1,1}, \Phi_{0,0}\})\}$$

is a stem of T: for any basis $\mathcal{B}$ of T, there exists $\mathcal{U} \subset \mathcal{B}$ and $\sigma \in \mathrm{Stab}(T)$ such that

$$\mathrm{Span}(\mathcal{U}) \circ \sigma \in \mathcal{C}.$$

Proof. Let $\mathcal{B}$ be a basis of T. We have to distinguish 2 cases:

- there exists an element of rank 6 in $\mathcal{B}$;
- any element of $\mathcal{B}$ has rank 3.

In the first case, there exists $\sigma \in \mathrm{Stab}(T)$ such that $\mathcal{B} \circ \sigma = (\Phi_{0,0} + \Phi_{1,1}, \Psi_1, \Psi_2, \Psi_3)$. We can assume that the coordinates of Ψ_1 in the basis $(\Phi_{0,0}, \Phi_{1,1}, \Phi_{0,1}, \Phi_{1,0})$ have the following shape:

$$(1, 0, \lambda_2, \lambda_3) \text{ or } (0, 1, \lambda_2, \lambda_3),$$

because otherwise every element of the basis $\mathcal{B}$ would have coordinates of the shape $(\lambda_0, \lambda_0, \lambda_2, \lambda_3)$. Up to the action of $\mathrm{Stab}(\Phi_{0,0} + \Phi_{1,1}) \cap \mathrm{Stab}(T)$, we can assume that we are in the first case.

- If Ψ_1 has rank three, using the action of $\mathrm{Stab}(\Phi_{0,0}+\Phi_{1,1})\cap\mathrm{Stab}(T)$, we are brought to one possibility satisfying the good shape: $\Psi_1=\Phi_{0,0}$.
- If Ψ_1 has rank 6, because of its shape, it is necessarily equal to $\Phi_{0,0}+\Phi_{0,1}+\Phi_{1,0}$.

In the second case, there exists $\sigma\in\mathrm{Stab}(T)$ such that $\mathcal{B}\circ\sigma=(\Phi_{0,0},\Psi_1,\Psi_2,\Psi_3)$. There exists $i\in\{1,\dots,3\}$ such that the coordinates of Ψ_i in the basis $(\Phi_{0,0},\Phi_{1,1},\Phi_{0,1},\Phi_{1,0})$ have the following shape:

$$(\lambda_0,1,\lambda_2,\lambda_3).$$

We can assume that $i=1$. Then, there exists $\sigma'\in\mathrm{Stab}(T)$ such that $\mathcal{B}\circ\sigma'=(\Phi_{0,0},\Phi_{1,1},\Psi_2,\Psi_3)$.

It may seem that we have a stem of 3 sets given by $\mathscr{S}_9(\mathrm{Span}(\Phi_{0,0}+\Phi_{1,1},\Phi_{0,0}))$, $\mathscr{S}_9(\mathrm{Span}(\Phi_{0,0}+\Phi_{1,1},\Phi_{0,0}+\Phi_{1,0}+\Phi_{0,1}))$ and $\mathscr{S}_9(\mathrm{Span}(\Phi_{0,0},\Phi_{1,1}))$. Actually the first and the third sets are equal. □

Thus, we compute the following sets, corresponding to the quotient $\mathcal{Q}$ computed with IntermediateSetViaQuotientComputation in Section 3.3:

- $\tilde{\mathscr{S}}_9(\{\Phi_{0,0}+\Phi_{1,1},\Phi_{0,0}+\Phi_{0,1}+\Phi_{1,0}\})$,
- $\tilde{\mathscr{S}}_9(\{\Phi_{0,0}+\Phi_{1,1},\Phi_{0,0}\})$.

$\mathcal{F}$	d	$\#\tilde{\mathscr{S}}_d(\{\mathcal{F}\})$	tests	step 1 (s)	step 2 (s)	step 3 (s)	sol.
$\left\{\begin{matrix}\Phi_{0,0}+\Phi_{1,1},\\ \Phi_{0,0}+\Phi_{0,1}\\ +\Phi_{1,0}\end{matrix}\right\}$	9	139	$5.0\cdot10^4$	$8.4\cdot10^4$	$3.2\cdot10^{-1}$	$7.3\cdot10$	44
$\left\{\begin{matrix}\Phi_{0,0}+\Phi_{1,1},\\ \Phi_{0,0}\end{matrix}\right\}$	9	$3.8\cdot10^8$	$6.3\cdot10^{10}$	$1.0\cdot10^5$	$1.0\cdot10^3$	$3.7\cdot10^6$	5,614

Table 4.8: Product of 2×3 by 3×2 matrices analysis

We obtained a speed-up of 10^9 compared to our implementation of BDEZStab, and we found 1,096,452 elements of $\mathscr{S}_{11}(T)$, divided in 196 equivalence classes of solutions with respect to the action of $\mathrm{Stab}(T)$. The computations described in Table 4.8 used an improved Step 3 for the family $\{\Phi_{0,0}+\Phi_{1,1},\Phi_{0,0}\}$: we select a subset of the transversal that is applied. This improvement takes into account two facts. First, the quotient $\mathrm{Stab}(\{\Phi_{0,0},\Phi_{1,1}\})/\mathrm{Stab}(T)\cap\mathrm{Stab}(\{\Phi_{0,0},\Phi_{1,1}\})$ can be represented by the set of representatives given by the pairs of the form

$$\left(\begin{pmatrix}P^{\mathrm{T}}&0\\0&I\end{pmatrix},\begin{pmatrix}P^{-1}&0\\0&I\end{pmatrix}\right),$$

$P\in\mathrm{GL}_3$. In particular, we know the action of such a pair on a matrix $M\in\mathcal{M}_{6,6}$ composed of four block $M_{0,0},M_{1,0},M_{0,1},M_{1,1}$ gives

$$\begin{pmatrix}PM_{0,0}P^{-1}&PM_{0,1}\\M_{1,0}P^{-1}&M_{1,1}\end{pmatrix}.$$

Second, given a subspace W of $\tilde{\mathscr{S}}_9(\{\Phi_{0,0}+\Phi_{1,1},\Phi_{0,0}\})$, there exists two elements $\phi_0,\phi_1 \in T+W$ of rank one such that $T+W = W \oplus \mathrm{Span}(\{\phi_0,\phi_1\})$. Thus, we have two non zero elements Ψ_0 and Ψ_1 in T such that there exist $\Psi_0',\Psi_1' \in W$ satisfying

$$\Psi_0 = \phi_0 + \Psi_0' \text{ and } \Psi_1 = \phi_1 + \Psi_1'.$$

The elements Ψ_0 and Ψ_1 have a non zero coordinate at $\Phi_{0,1}$ or $\Phi_{0,1}$ over the basis

$$\{\Phi_{0,0},\Phi_{1,0},\Phi_{0,1},\Phi_{1,1}\}.$$

Thus, we keep the elements $W\circ\sigma$ such that $W \in \tilde{\mathscr{S}}_9(\{\Phi_{0,0}+\Phi_{1,1},\Phi_{0,0}\})$ and $\sigma \in \mathrm{Stab}(T) \cap \mathrm{Stab}(\{\Phi_{0,0}+\Phi_{1,1},\Phi_{0,0}\})$ for which there exist $\Phi_0,\Phi_1 \in W\circ\sigma$ for which their matrix representation in $\mathcal{M}_{6,6}$ is composed of four blocks $M_{0,0},M_{1,0},M_{0,1},M_{1,1}$ such that $\mathrm{rk}(M_{0,0}) \leq 1$ and $\mathrm{rk}(M_{1,1}) \leq 1$.

Chapter 5

Algorithm for the precomputation required in the new method

In this section, we propose an approach for computing the set $\mathscr{S}_{m,n,d}/\mathrm{GL}(K^m) \times \mathrm{GL}(K^n)$, required by the algorithm described in Section 3.3. Its cost is at least exponential in d, m and n and difficult to estimate precisely.

Notation 5.1. *We denote by Ω_d the quotient $\mathscr{S}_{d,d,d}/\mathrm{GL}(K^d) \times \mathrm{GL}(K^d)$ for any $d \geq 1$.*

5.1 Representation of elements of Ω_d

First, we describe how we represent elements of $\mathscr{S}_{m,n,d}$ and we prove that, given the knowledge of Ω_d, we can deduce the elements of $\mathscr{S}_{m,n,d}/\mathrm{GL}(K^m) \times \mathrm{GL}(K^n)$ for any m and n from this precomputation.

Let W be an element of $\mathscr{S}_{m,n,d}$. There exist d rank-one bilinear forms $\phi_t : (\mathbf{a}, \mathbf{b}) \mapsto \alpha_t(\mathbf{a}) \cdot \beta_t(\mathbf{b})$ such that $W = \mathrm{Span}\left((\phi_t)_{t\in\{0,\ldots,d-1\}}\right)$. In the canonical basis of K^m and K^n, we represent α_t and β_t as matrices of $\mathcal{M}_{1,m}$ and $\mathcal{M}_{1,n}$. Thus, there exist two matrices $U \in \mathcal{M}_{d,m}$ and $V \in \mathcal{M}_{d,n}$, whose rows are given by the linear forms α_t and β_t respectively, and W can be represented by the pair (U, V). Such a representation is not unique (for example, any permutation of the rows of (U, V) gives a valid representation). In particular, for a pair of matrices (U, V) representing some vector space W, there exists $\sigma = \mu \times \nu$ in $\mathrm{GL}(K^m) \times \mathrm{GL}(K^n)$ such that the pair of matrices U', V', such that $(U', V') = (U \circ \mu, V \circ \nu)$ represents $W \circ \sigma$, are the reduced column echelon form of the matrices U and V, respectively.

Example 5.2. *Let us consider the vector space W of $\mathscr{S}_{3,4,6}$ generated by the rank-one*

bilinear forms represented by

$$M_1 = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, M_2 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}, M_3 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix},$$
$$M_4 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, M_5 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, M_6 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

The couple of matrices (U, V) *associated to* W *is*

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Assuming that we have a representation of the elements of $\mathscr{S}_{d,d,d}/\mathrm{GL}(K^d) \times \mathrm{GL}(K^d)$ in terms of pairs of matrices $(U, V) \in \mathcal{M}_{d,d} \times \mathcal{M}_{d,d}$ in reduced column echelon form, we obtain all the elements of

$$\mathscr{S}_{m,n,d}/\mathrm{GL}(K^m) \times \mathrm{GL}(K^n)$$

by considering the subset Ω'_d of $\Omega_d = \mathscr{S}_{d,d,d}/\mathrm{GL}(K^d) \times \mathrm{GL}(K^d)$ of elements represented by matrices (U, V) in reduced column echelon form such that $\mathrm{rk}(U) \leq \min(m, d)$ and $\mathrm{rk}(V) \leq \min(n, d)$. Given m and n, a set of representatives for

$$\mathscr{S}_{m,n,d}/\mathrm{GL}(K^m) \times \mathrm{GL}(K^n)$$

can be seen as matrices $(U', V') \in \mathcal{M}_{d,m} \times \mathcal{M}_{d,n}$ in reduced column echelon form and for which there exists matrices $(U, V) \in \mathcal{M}^2_{d,d}$, representing an element of Ω'_d, obtained by adding $d-m$ an $d-n$ zero columns to U' and V', respectively, or by removing zero columns if $d < m$ or $d < n$.

5.2 Method for the computation of a set of representatives of the quotient

Our strategy consists in deducing Ω_d from the computation of Ω_{d-1}. Algorithm 11 describes this strategy: for each vector space W of Ω_{d-1}, we extend it to a vector space of $\mathcal{L}(K^d, K^d; K)$ by adding a zero-column to the corresponding U and V matrices, and we consider the vector spaces $W \oplus \mathrm{Span}(\phi)$ that can be obtained by adding an element ϕ of rank one. We remove from the set of $W \oplus \mathrm{Span}(\phi)$ the vector spaces that are RP-isomorphic via an isomorphism test. We determine whether two vector spaces W' and W are RP-isomorphic if there exists a basis of W' of rank-one bilinear forms such that the corresponding couple of matrices (U', V') in reduced column echelon form is equal to (U, V). The complexity of this approach depends on the number

of bases of rank-one bilinear forms of W, which, compared to d, is not large generically. However, there are degenerate cases for which the nomber of bases is very large (exponential in d^2). These cases require specific code to recognize them and to treat them separately.

We denote by Extend the algorithm transforming a set of subspaces of $\mathcal{L}(K^{d-1}, K^{d-1}; K)$ into subspaces of $\mathcal{L}(K^d, K^d; K)$ with zero-padding.

Algorithm 11 IterativeQuotientsComputation

Input: Ω_{d-1}, a set $\mathcal{G}$ of rank-one bilinear forms
Output: Ω_d

1: $\widehat{\Omega}_{d-1} \leftarrow \mathsf{Extend}(\Omega_{d-1})$ ▷ We compute extensions of elements of Ω_{d-1} in $\mathcal{L}(K^d, K^d; K)$
2: $\mathcal{L} \leftarrow \emptyset$
3: **for** $W \in \widehat{\Omega}_{d-1}$ **do**
4: $\quad \mathcal{H} \leftarrow \mathcal{G}/_{\mathrm{Stab}\, W}$
5: $\quad$ **for** $h \in \mathcal{H}$ **do**
6: $\quad\quad \mathcal{L} \leftarrow \mathcal{L} \cup \{W \oplus \mathrm{Span}(h)\}$
7: $\quad$ **end for**
8: **end for**
9: **return** $\mathcal{L}/_{\mathrm{GL}(K^d) \times \mathrm{GL}(K^d)}$ ▷ We remove RP-isomorphic vector spaces in $\mathcal{L}$

The naive algorithm which checks for each pair of elements of the set $\mathcal{L}$ whether or not they are RP-isomorphic, computed in Line 9 of Algorithm 11, can be improved. Indeed, we propose to compute invariants for the group action induced by $\mathrm{GL}(K^d) \times \mathrm{GL}(K^d)$ and to compare subspaces having the same invariants. For example, for $W \in \mathscr{S}_{d,d,d}$, we consider the polynomial $P_W = \sum_{0 \leq t \leq d} p_{W,t} X^t$ such that

$$\forall t \geq 0,\ p_{W,t} = \#\{\phi \in W |\ \mathrm{rk}(\phi) = t\}.$$

Therefore, for any $\sigma \in \mathrm{GL}(K^d) \times \mathrm{GL}(K^d)$, $P_{W \circ \sigma} = P_W$.

In Algorithm 11 we regroup the vector spaces of $\mathcal{L}$ which have the same invariants. For two subspaces W and W' having the same invariants, we need to explain how we decide whether they are RP-isomorphic or not. We describe the procedure in Algorithm IsomorphismTest.

Algorithm 12 IsomorphismTest

Input: W and W' in $\mathscr{S}_d$
Output: **true** if and only if W and W' are RP-isomorphic

We represent W' by a pair (U', V') in column echelon form
for $\mathcal{B} \subset W$, basis of rank-one bilinear forms **do**
$\quad$ We represent W by a pair (U, V) in column echelon form given by the basis $\mathcal{B}$
$\quad$ **for** $\sigma \in \mathfrak{S}_d$ **do** ▷ $\mathfrak{S}_d$ are the row permutations of $\mathcal{M}_{d,d}$
$\quad\quad$ **if** The column echelon form of $(U \circ \sigma, V \circ \sigma)$ is equal to (U', V') **then**
$\quad\quad\quad$ **return true**
$\quad\quad$ **end if**
$\quad$ **end for**
end for
return false

Thus, for a vector space $W \in \mathscr{S}_d$ and $\mathcal{H} \subset W$ the set of rank-one bilinear forms in W, the complexity of IsomorphismTest is given by $\binom{\#\mathcal{H}}{d}$, an approximation of the number of possible

bases of rank-one bilinear forms, times $d!$ times the cost of a Gauss reduction of two matrices of $\mathcal{M}_{d,d}$. If $\#\mathcal{H}$ is too large, we apply the idea given in Section 5.3.

5.3 Improvement to the strategy using classification of vector spaces

This section is dedicated to a classification of the elements of Ω_d which allows one to improve the isomorphism test and the computation of a stabilizer depending on the type.

Proposition 5.3. *Let $d \geq 1$. There exists $W \in \Omega_d$ such that there are $\#\,\mathrm{GL}(d,K)$ free families $\mathcal{B} = (\phi_0, \ldots, \phi_{d-1})$ with $\mathrm{rk}(\phi_j) = 1$ for any j, and such that $W = \mathrm{Span}(\mathcal{B})$.*

Proof. Let $\mathbf{e}_0, \ldots, \mathbf{e}_{d-1}$ be a basis of K^d. We consider $\alpha_j \in \mathcal{L}(K^d; K)$ such that $\alpha_j(\mathbf{e}_k) = 0$ for $k \neq j$ and $\alpha_j(\mathbf{e}_j) = 1$, and $\beta \in \mathcal{L}(K^d; K)$ such that $\beta(\mathbf{e}_0) = 1$ and $\beta(\mathbf{e}_k) = 0$ for $k \geq 1$. Let $W = \mathrm{Span}(\alpha_0 \cdot \beta, \alpha_1 \cdot \beta, \ldots, \alpha_{d-1} \cdot \beta)$. Any $\phi \in W$ of rank one can be decomposed with a linear form $\alpha \in\in \mathcal{L}(K^d; K)$ and the linear form already defined β: $\phi = \alpha \cdot v$.

Thus, we have all the possible bases by considering $(\alpha_0 \circ \sigma)\beta, (\alpha_1 \circ \sigma)v, \ldots, (\alpha_{d-1} \circ \sigma)\beta$ for $\sigma \in \mathrm{GL}(d, K)$. □

According to Section 5.1, the complexity of testing whether two vector spaces W and W' are RP-isomorphic depends on the number of bases of W of bilinear forms of rank one. The proposition 5.3 implies that by using the procedure described in Section 5.1, we possibly have to enumerate $\#\,\mathrm{GL}(d, K)$ bases, which is too large for our applications.

However, it appears that the number of bases per $W \in \Omega_d$ is, in general, smaller than $\#\,\mathrm{GL}(d, K)$ by an order of magnitude. Moreover, we can deduce directly that a vector space having $\#\,\mathrm{GL}(d, K)$ bases has a stabilizer RP-isomorphic to $\mathrm{GL}(d, K)$, given as a set of generators. Thus, it is possible to adapt our strategy to vector spaces having a lot of bases and to compute directly a set of generators.

Definition 5.4. *Let $W \in \Omega_d$, E a subspace of W and $\mathcal{H}$ be the set of elements of W of rank one.*

We say that E is a left (respectively right) degenerate space if

- *it is generated by elements $h \in \mathcal{H}$,*
- *there exists $\alpha \in \mathcal{L}(K^d; K)$ (respectively β) such that for any $\Phi \in \mathcal{H} \cap E$, there exists $\beta \in \mathcal{L}(K^d; K)$ (respectively α) such that $\Phi(\mathbf{a}, \mathbf{b}) = \alpha(\mathbf{a}) \cdot \beta(\mathbf{b})$,*
- *its dimension is maximal over all subspaces having the two previous properties.*

The definition of a degenerate space permits to restrict the number of bases considered, since two left (or right) degenerate spaces of the same dimension are trivially RP-isomorphic. Therefore, in order to check the existence of a transformation σ such that $W = W' \circ \sigma$ for W and W' two elements of $\mathscr{S}_{d,d,d}$, it is interesting to check for a σ mapping a degenerate space onto another degenerate space.

Proposition 5.5. *Let W and W' two elements of Ω_d. If there exists a unique left (respec-*

tively right) degenerate space $E \in W$ and if W and W' are RP*-isomorphic, then there exists a unique left (respectively right) degenerate space $E' \in W'$ such that* $\dim E' = \dim E$.

Thus, by using the proposition 5.5, we can restrict the search of a σ mapping a vector space W onto another one W' by considering the elements sending the unique degenerate space of W (if it is unique) on the unique degenerate space of W'.

It should be possible to take into account the case of several degenerate spaces. However, trying to establish the optimal strategy in that case goes beyond the scope of the current work.

Example 5.6. *We give here an example of two vector spaces W and W' of $\mathcal{L}(K^3, K^4; K)$ containing a left degenerate space of dimension* 4*. The vector space W is generated by the bilinear forms represented by the matrices*

$$M_0 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, M_1 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, M_2 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix},$$
$$M_3 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}, M_4 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}, M_5 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

and the vector space W' is generated by

$$N_0 = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, N_1 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}, N_2 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix},$$
$$N_3 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, N_4 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, N_5 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

These vector spaces are RP*-isomorphic and any isomorphism σ such that $W \circ \sigma = W'$ satisfies*

$$\mathrm{Span}(M_2, M_3, M_4, M_5) \circ \sigma = \mathrm{Span}(N_2, N_3, N_4, N_5).$$

The vector spaces $\mathrm{Span}(M_2, M_3, M_4, M_5)$ *and* $\mathrm{Span}(N_2, N_3, N_4, N_5)$ *are degenerate spaces of dimension* 4*.*

We improve on the algorithm given by IsomorphismTest for vector spaces W and W' containing degenerate spaces E and E', respectively, such that E' is the unique degenerate space of dimension $\dim(E)$ of W'.

Algorithm 13 IsomorphismTestDegenerate

Input: W and W' in $\mathscr{S}_d$, E and E' degenerate spaces of W and W' respectively
Output: **true** if and only if W and W' are RP-isomorphic

We represent W' by a pair (U', V') in column echelon form such that the $\dim(E)$ first rows of U and V form a basis of E'
for $\mathcal{B} \subset W$, basis of rank-one bilinear forms containing a basis of E **do**
    We represent W by a pair (U, V) in column echelon form given by the basis $\mathcal{B}$ such that the $\dim(E)$ first rows of U and V form a basis of E
    for $\sigma \in \mathfrak{S}_d(\dim E, \ldots, d-1)$ **do** ▷ Permutations of the $d - \dim(E)$ last rows
        if The column echelon form of $(U \circ \sigma, V \circ \sigma)$ is equal to (U', V') **then**
            return true
        end if
    end for
end for
return false

Since we fixed the $\dim(E)$ first rows of U and V in IsomorphismTestDegenerate, if $\mathcal{H}$ is the set of rank-one bilinear forms of W, the complexity is reduced to $\binom{\#\mathcal{H}}{d-\dim(E)}$ times $(d-\dim(E))!$ times the cost of a Gauss reduction.

5.4 Experimental results

We have been able to compute Ω_d for $i \in \{1, \ldots, 8\}$ and $K = \mathbb{F}_2$ with an implementation in Magma V2.21-3 [9][1].

We observe in Table 5.1 that it is more likely that a vector space Ω_d has d bilinear forms of rank one rather than $2^d - 1$, although we have $d!$ possible bases in the first scenario and $d! \cdot \binom{2^d-1}{d}$ in the second one: the most common scenario is the nicer one in terms of computation. This is why it is interesting to consider the notion of degenerate space introduced in Section 5.3, which allows one to have a specific strategy for the rare cases where the number of possible basis is too large.

In Table 5.2, we give the time required to compute each Ω_i with our method, its cardinality, and we compare it with a theoretical upper bound. We observe that an upper bound on $\#\Omega_d$ with the good order of magnitude is hard to obtain. Indeed, we are only able to say that $\#\Omega_d$ is bounded by the quantity

$$\left(\frac{\#K^d - 1}{\#K - 1}\right)^2 \cdot \#\Omega_{d-1},$$

corresponding to the number of possible rank-one bilinear forms that we add to elements of Ω_{d-1} to obtain an element of Ω_i. This formula leads recursively to the following bound:

$$\#\Omega_d \leq \frac{1}{(\#K-1)^d} \left(\prod_{t \in \{1,\ldots,d\}} (\#K^t - 1) \right)^2.$$

This upper bound differs by a huge factor from the true cardinality of Ω_d and cannot consequently be used in a complexity analysis.

[1] The code of this implementation can be found at the address http://karancode.gforge.inria.fr

#elts of Ω_2	N	#elts of Ω_6	N	#elts of Ω_7	N
1	2	262	6	3315	7
2	3	295	7	3569	8
#elts of Ω_3	N	150	8	1993	9
4	3	77	9	818	10
2	4	63	10	593	11
1	5	44	11	383	12
2	7	17	12	226	13
#elts of Ω_4	N	16	13	77	14
10	4	6	14	84	15
8	5	6	15	26	16
4	6	11	17	24	17
2	7	6	18	47	18
2	8	4	19	42	19
3	9	6	21	14	20
2	15	2	32	23	21
#elts of Ω_5	N	2	33	10	22
38	5	2	63	8	23
43	6			6	25
21	7			3	29
6	8			8	33
13	9			6	34
7	10			4	35
4	11			4	37
3	13			2	64
2	16			2	65
2	17			2	127
2	31				

Table 5.1: Number of elements of Ω_d over $K = \mathbb{F}_2$ having N bilinear forms of rank one, when this number is not zero.

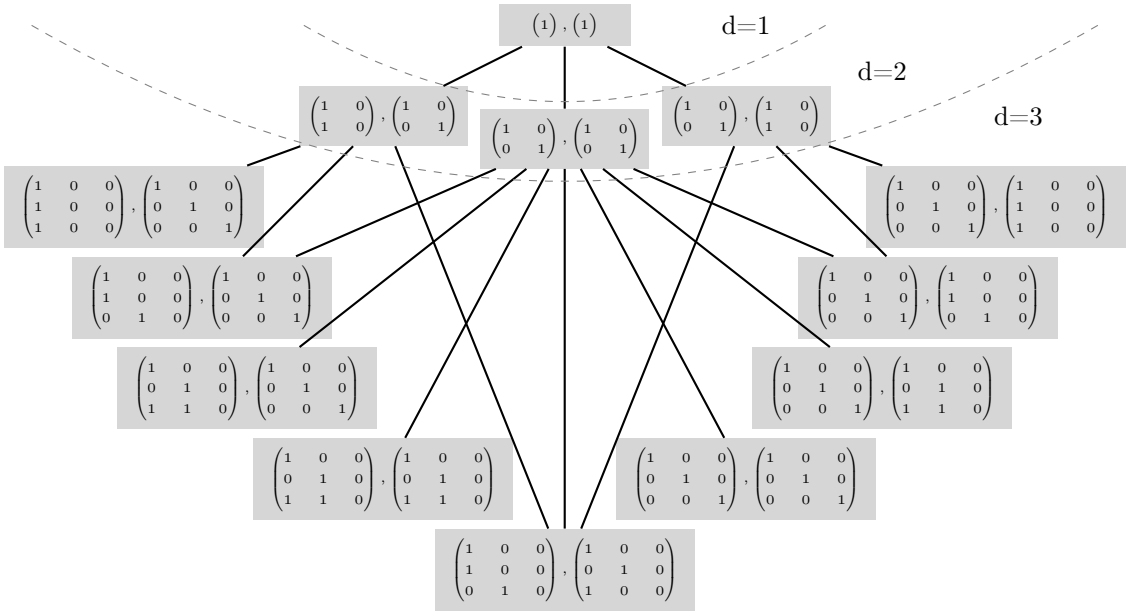

Figure 5.1: Partially ordered structure of the Ω_d for $d \leq 3$ and $K = \mathbb{F}_2$.

set	Ω_1	Ω_2	Ω_3	Ω_4	Ω_5	Ω_6	Ω_7	Ω_8
cardinality	1	3	9	31	141	969	11,289	265,577
upper bound	1	9	$4.4 \cdot 10^2$	$9.9 \cdot 10^4$	$9.5 \cdot 10^7$	$3.8 \cdot 10^{11}$	$6.1 \cdot 10^{15}$	$4.0 \cdot 10^{20}$
time (s)	0	0.0	0.0	$1.8 \cdot 10^{-1}$	1.5	$1.8 \cdot 10$	$4.7 \cdot 10^2$	$1.8 \cdot 10^4$

Table 5.2: Timings for our approach to compute the sets Ω_d over $K = \mathbb{F}_2$ on a single core of a 3.3 GHz Intel Core i5-4590.

We provide in Figure 5.1 how subspaces of Ω_3 over $\mathbb{F}_2$ are related to Ω_2 and Ω_1 by using its poset structure. We represent an element of Ω_d by a couple of matrices (U, V) of $M^2_{d,d}$.

Part II

Asymptotic multiplication

Notations

n	bit-length of the integers multiplied
N	length of the polynomials multiplied
p, q	primes
ω	root of unity
$\log$	natural logarithm
$\log^{(i)}$	i-th iterate of the log function
$\log_2$	log in radix two
$\log^*$	iterated logarithm function (defined in Section 6.6)
li	$x \mapsto \int_2^x \frac{\mathrm{d}t}{\log t}$
$P(r, \lambda)$	generalized Fermat number $r^{2^\lambda} + 1$
$\mathcal{P}(x)$	set of primes smaller than x
$\pi(x, q, r)$	number of primes $\leq x$ congruent to r mod q
$\mathsf{M}_{\mathcal{R}}$	cost of multiplying $a \in \mathcal{R}$ by $b \in \mathcal{R}$, with no auxiliary inputs

Chapter 6

Fast Fourier Transform

In this chapter, we describe the paradigm used to multiply large integers. This paradigm relies on the evaluation-interpolation scheme. The methods described in this chapter imply transforming integers into polynomials and, then, multiplying polynomials. We have some freedom about the choice of the ring of the coefficients of the polynomials and on their degrees.

6.1 Integers to polynomials

Let a and b be non negative integers to be multiplied. Let n be an upper bound on the bit-length of these integers. Standard substitution techniques (see e.g. [6]) allow one to compute c via the computation of the product $C(x) = A(x)B(x)$, where A and B are univariate polynomials related to a and b. Polynomials are taken over some well-chosen ring $\mathcal{R}$. (In this chapter, we do not explicitly fix a choice for $\mathcal{R}$.)

We obtain the polynomials A and B using the representation of a and b. Assuming that we have their radix-2 representation, a and b can be seen as sequences of elements in $\{0, 1\}$. We have

$$a = \sum_{0 \leq i < n} a_i 2^i \text{ and } b = \sum_{0 \leq i < n} b_i 2^i.$$

Let $\eta = 2^t$ be a power of two. We have

$$a = \sum_{0 \leq i < \lceil n/t \rceil} \left(\sum_{0 \leq j < t} a_{it+j} 2^j \right) \eta^i \text{ and } b = \sum_{0 \leq i < \lceil n/t \rceil} \left(\sum_{0 \leq j < t} b_{it+j} 2^j \right) \eta^i.$$

Via zero-padding on a and b, we can assume that there is no constraint on $t = \log_2 \eta$ except to be less than n. Denoting by $\tilde{a}_i$ and $\tilde{b}_i$ the integers $\sum_{0 \leq j < t} a_{it+j} 2^j$ and $\sum_{0 \leq j < t} b_{it+j} 2^j$, we consider the polynomials

$$A(X) = \sum_{0 \leq i < n/t} \tilde{a}_i X^i \text{ and } B(X) = \sum_{0 \leq i < n/t} \tilde{b}_i X^i.$$

A procedure allowing one to multiply the polynomials $A, B \in \mathbb{Z}[X]$ produces a polynomial $C = A \cdot B$ satisfying

$$C(\eta) = A(\eta) \cdot B(\eta) = a \cdot b = c.$$

The length of A and B is $\lceil n/t \rceil$. Thus, the length of C is $2\lceil n/t \rceil - 1$.

We denote by N the smallest power of two such that $2\lceil n/t \rceil - 1 \leq N$. The integer N is an upper bound on the length of the polynomial C. In practice it may be interesting to consider that N is not a power of two, as it has been studied for example in [58]. From an asymptotic point of view in the deterministic multi-tape Turing model, we are allowed to restrict the problem to the case where N is a power of two. Assuming that the coefficients of C are represented in radix 2, we have a procedure computing a radix-2 representation of c from that of C in $O(n)$.

A procedure to multiply the polynomials A and B is required. We see A and B as elements of $\mathcal{R}[X]$ for some ring $\mathcal{R}$ for which the elements have a finite representation on a binary machine. For example, the ring $\mathcal{R}$ can be $\mathbb{Z}/2^k\mathbb{Z}$ for some positive integer k, or $\mathbb{C}$, the complex field, with finite precision. The main constraint on $\mathcal{R}$ is to allow one to represent correctly the coefficients of C. For the example given by the ring $\mathbb{Z}/2^k\mathbb{Z}$, it is required for 2^k to be larger than all the coefficients of C. This constraint can be summarized via the following inequality:

$$2^k \geq \frac{N+1}{2}\eta^2.$$

We prefer to use the inequality 6.1

$$2^k \geq N\eta^2, \tag{6.1}$$

which is allowed because N is a power of two.

Reciprocally, Kronecker and Schönhage proposed in [39, 51] to multiply polynomials by transforming them into integers. Thus, the complexity of multiplication of polynomials is related to the complexity of the multiplication of integers. We refer to this method as "Kronecker substitution".

We can describe the algorithm as follows. Let A and B be polynomials of $\mathbb{Z}[X]$ of length $N/2$, with coefficients bounded by $\eta = 2^t$. Then, we consider the integers a and b such that $a = A(2^k)$ and $b = B(2^k)$, where $2^k = N \cdot \eta^2$. We compute $c = a \cdot b$. We read the coefficients of C from c by splitting c into pieces of bit-length k.

For example, if we multiply $1+X$ by $X+X^2$, we obtain, in radix two with the least significant bit at the left, the integers

$$(100010000000) \text{ and } (000010001000),$$

since $k = 2 + \log_2 4 = 4$. The product c is represented by

$$(000010000010010000000).$$

When we split c into pieces of bit-length four, we obtain a polynomial C equal to

$$C = (0000) + (1000)X + (0100)X^2 + (1000)X^3 + (0000)X^4 = X + 2X^2 + X^3.$$

6.2 Evaluation-Interpolation

We describe how to compute the product of two polynomials $A, B \in \mathcal{R}[X]$. For large degrees, we use the evaluation-interpolation scheme. The principle is to use the fact that for two sequences (x_i) and (y_i) of N elements of $\mathcal{R}$, such that the x_i's are distinct, there exists exactly one polynomial C of degree at most $N-1$ such that $C(x_i) = y_i$ (by hypothesis, the ring $\mathcal{R}$ contains N distinct elements). This polynomial C can be obtained via the Lagrange interpolation. Using this fact, a polynomial C such that $C = A \cdot B$ can be computed via 3 steps:

1. compute the evaluation of A and B on N points x_i;
2. compute the point-wise products $A(x_i) \cdot B(x_i)$;
3. compute C as the unique polynomial such that $C(x_i) = A(x_i) \cdot B(x_i)$.

This general strategy used to multiply polynomials is called the evaluation-interpolation scheme. Such a strategy is described in Algorithm 14, where we highlight the possibility of computing the product $C(x) = A(x)B(x)$ by multipoint evaluation and interpolation if the ring $\mathcal{R}$ in which computations take place provides a nice and sufficiently large set of interpolation points.

Algorithm 14 Multiply in $\mathbb{Z}$ via multipoint evaluation of polynomials

Input: a, b two integers to be multiplied; we let $n \geq \max(\log_2 |a|, \log_2 |b|)$
η a power of two; we let $N = \lceil 2n/\log_2 \eta \rceil$
$\mathcal{R}$ a ring such that there exists an injective function mapping integers below $N\eta^2$ to $\mathcal{R}$
$\mathcal{S} \subset \mathcal{R}$ a set of N evaluation points
Output: $c = a \cdot b$
function MultiplyIntegersViaMultipointEvaluation$(a, b, \eta, \mathcal{R})$
    Let $A(x) \in \mathbb{Z}[X]$ with positive coefficients less than η and such that $A(\eta) = a$
    Let $B(x) \in \mathbb{Z}[X]$ with positive coefficients less than η and such that $B(\eta) = b$
    A and B are seen as elements of $\mathcal{R}[X]$
    $\hat{A} \leftarrow$ MultiEvaluation$(A, \mathcal{S})$; define $\hat{B}$ likewise.
    $\hat{C} \leftarrow$ PointwiseProduct$(\hat{A}, \hat{B})$.
    $C \leftarrow$ Interpolation$(\hat{C}, \mathcal{S})$
    Reinterpret C as a polynomial in $\mathbb{Z}$.
    return $c = C(\eta)$.
end function

The procedure followed by Algorithm 14 is in fact quite general, and can be applied to a wider range of bilinear operations than just integer multiplication. For example, one can adapt this algorithm to multiply polynomials or power series in various rings, or to compute other operations such as middle products or dot products. The last example of the dot product is archetypal of the situation where results of the MultiEvaluation step (as e.g. $\hat{A}$ in Algorithm 14) are used more than once. The conditions on $\mathcal{R}$ that are used to guard against possible overflow must be adjusted accordingly. For example, if $\mathcal{R}$ is chosen with the form $\mathbb{Z}/2^m\mathbb{Z}$, the bound given by the inequality 6.1 implies that 2^m should be larger than $N\eta^2$.

6.3 Discrete Fourier Transform

We now discuss how multi-evaluation can be performed efficiently. This depends first and foremost on the number of evaluation points N and on the ring $\mathcal{R}$. FFT algorithms are special-purpose algorithms adapted to evaluation points chosen among roots of unity in $\mathcal{R}$. In order to allow $\mathcal{R}$ to be a non-integral ring, we need the following definition.

Definition 6.1. *Let $N \geq 1$ be an integer, and $\mathcal{R}$ be a ring of characteristic zero or characteristic coprime to N, containing an N-th root of unity ω. We say that ω is a principal N-th root of unity if* $\forall i \in [\![1, N-1]\!], \sum_{j=0}^{N-1} \omega^{ij} = 0$.

The notion of principal root of unity is stricter than the classical notion of primitive root, and provides the suitable generalization to non-integral rings. For example in $\mathbb{C} \times \mathbb{C}$, the element $(1, i)$ is a primitive 4-th root of unity but not a principal 4-th root of unity.

Using the set of powers of ω as a set of evaluation points, we define the discrete Fourier transform (DFT) as follows.

Definition 6.2 (Discrete Fourier Transform (DFT))**.** *Let $\mathcal{R}$ be a ring with ω a principal N-th root of unity. The DFT of length N and base root ω over $\mathcal{R}$ is the ring morphism* $\mathrm{DFT}_{N,w}$ *defined as:*

$$\begin{cases} \mathcal{R}[X]/(X^N - 1) & \to & \mathcal{R}[X]/(X - 1) \times \mathcal{R}[X]/(X - \omega) \times \cdots \times \mathcal{R}[X]/(X - \omega^{N-1}) \\ P & \mapsto & (P(1), P(\omega), \ldots, P(\omega^{N-1})). \end{cases}$$

We customarily write a DFT of length N of a polynomial P as the polynomial $\hat{P}$ of degree less than $N - 1$ defined as

$$\hat{P} = \mathrm{DFT}_{N,\omega}(P) = P(1) + P(\omega)X + \cdots + P(\omega^{N-1})X^{N-1}.$$

The DFT of a polynomial P can be understood as the product of a Vandermonde matrix

$$V(\omega) = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^{N-1} \\ 1 & \omega^2 & \omega^4 & \cdots & \omega^{2(N-1)} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & \omega^{N-2} & \omega^{2(N-2)} & \ldots & \omega^{(N-2)(N-1)} \\ 1 & \omega^{N-1} & \omega^{2(N-1)} & \ldots & \omega^{(N-1)(N-1)} \end{pmatrix}$$

with the vector

$$\begin{pmatrix} P_0 \\ P_1 \\ \vdots \\ P_{N-1} \end{pmatrix}$$

given by the coefficients of P.

The discrete Fourier Transform is convenient because the inverse of a DFT is also a DFT scaled by $1/N$ (see e.g. [59, §8]).

Proposition 6.3. *Let P be a polynomial with coefficients in $\mathcal{R}$ and ω a principal root of unity of $\mathcal{R}$. We have* $\mathrm{DFT}_{N,\omega^{-1}}(\mathrm{DFT}_{N,\omega}(P)) = N \cdot P$.

Proof. Let M be the matrix $V(\omega^{-1}) \cdot V(\omega)$. The coefficient (i, j) of M is $\sum_{0 \leq k < N} \omega^{(-i+j)k}$. By definition of a principal root of unity, if $i \neq j$, this coefficient is zero. Otherwise, it is equal to N. Thus, M is a diagonal matrix equal to N times the identity matrix. □

Corollary 6.4. *The morphism* $\mathrm{DFT}_{N,\omega}$ *is, in fact, a ring isomorphism.*

Proof. The integer N is supposed to be invertible in the ring $\mathcal{R}$. Consequently, $\mathrm{DFT}_{N,\omega}$ is invertible. □

Thus, the task of interpolating a polynomial A from its multi-evaluation $\hat{A}$ can be done with essentially the same algorithm.

The isomorphism given by the DFT of a polynomial of length N is an isomorphism between $\mathcal{R}[X]/(X^N-1)$ and $\mathcal{R}^N$. In our applications, it is also useful to have an isomorphism between $\mathcal{R}[X]/(X^N+1)$ and $\mathcal{R}^N$. Thus, we define the Half-DFT, which is an isomorphism derived from the one given by the DFT.

Definition 6.5 (Half discrete Fourier Transform (DFT))**.** *Let* $\mathcal{R}$ *be a ring with* ω *a principal* $2N$*-th root of unity. The Half-DFT of length* N *and base root* ω *over* $\mathcal{R}$ *is the ring isomorphism* $\text{Half-DFT}_{N,\omega}$ *defined as:*

$$\begin{cases} \mathcal{R}[X]/(X^N+1) & \to & \mathcal{R}[X]/(X-\omega) \times \mathcal{R}[X]/(X-\omega^3) \times \cdots \times \mathcal{R}[X]/(X-\omega^{2N-1}) \\ P & \mapsto & (P(\omega), P(\omega^3), \ldots, P(\omega^{2N-1})). \end{cases}$$

The fact that $(X^N+1) = \prod_{0\le i<N}(X-\omega^{2i+1})$ follows from

$$(X^{2N}-1) = \prod_{0\le i<2N}(X-\omega^i)$$

and

$$(X^N-1) = \prod_{0\le i<N}(X-\omega^{2i}).$$

Half-DFTs are used for polynomial products modulo X^N+1, as opposed to X^N-1. Such convolutions are called *negacyclic*. The half-DFT of a polynomial A is related to the DFT of the polynomial $A(\omega X)$ via

$$\text{Half-DFT}_{N,\omega}(A(X)) = \mathrm{DFT}_{N,\omega^2}(A(\omega X)).$$

Thus, an algorithm computing a DFT can be used to compute a half-DFT, with N additional multiplications involved in the computation of the polynomial $A(\omega X)$.

6.4 Cooley-Tukey FFT

Cooley and Tukey showed in [17] how a DFT of composite order $N = N_1N_2$ can be computed. This algorithm is also sometimes called "matrix Fourier algorithm", alluding to the fact that it performs N_2 "column-wise" transforms of length N_1, followed by N_1 "row-wise" transforms of length N_2. It is described in Algorithm 15.

The notation $\mathrm{DFT}_{N,\omega}$ denotes a mathematical object rather than an algorithm. Therefore, we need to detail how recursive computations of $\mathrm{DFT}_{N_k,\omega_k}$ are handled in Algorithm 15. Two approaches are rather typical instantiations of the Cooley-Tukey algorithm when the length N is a power of two:

Algorithm 15 General Cooley-Tukey FFT of order $N = N_1 N_2$

Input: $A = \sum_{i=0}^{N-1} a_i X^i \in \mathcal{R}[X]/(X^N - 1)$ and ω a principal N-th root of unity. Let $\omega_1 = \omega^{N_2}$ and $\omega_2 = \omega^{N_1}$.
Output: $\hat{A} = \mathrm{DFT}_{N,\omega}(A) = A(1) + A(\omega)X + \cdots + A(\omega^{N-1})X^{N-1}$
function LargeRadixCooleyTukeyFFT(A,N_1,N_2,ω)
    Let $(B_j(X))_j \in \mathcal{R}[X]^{N_2}$ be such that $A(X) = \sum_{j<N_2} B_j(X^{N_2})X^j$ ▷ $\forall j,\ \deg B_j \le N_1$
    for $j \in \{0, \ldots, N_2 - 1\}$ **do**
        $\hat{B_j} \leftarrow \mathrm{DFT}_{N_1,\omega_1}(B_j)$ ▷ $w_1 = w^{N_2}$ is a principal N_1-st root
        $\hat{B_j} \leftarrow \hat{B_j}(\omega^j X)$
    end for
    Let $(S_i(Y))_i \in \mathcal{R}[Y]^{N_1}$ be such that $\sum_j \hat{B_j}(X)Y^j = \sum_i S_i(Y)X^i$ ▷ $\forall i,\ \deg S_i \le N_2$
    for $i \in \{0, \ldots, N_1 - 1\}$ **do**
        $\hat{S_i}(Y) \leftarrow \mathrm{DFT}_{N_2,\omega_2}(S_i)$ ▷ $w_2 = w^{N_1}$ is a principal N_2-nd root
    end for
    return $\sum_{i<N_1} \hat{S_i}(X^{N_1})X^i$
end function

- "radix-two FFT": For a length $N = 2^k$, compute $N_2 = 2$ transforms of length $N_1 = 2^{k-1}$ (often called "butterflies"), then recurse with two transforms of length 2. We use the notation RadixTwoCooleyTukeyFFT(N, ω, A) for this algorithm. The radix-two Cooley Tukey has a particular form, described in Algorithm 16, using the fact that $\omega^{N/2} = -1$.
- "large-radix FFT": More generally, for a length $N = 2^{uq+r}$ with $r < u$, and $q > 0$, compute $N_2 = N/2^u$ transforms of length $N_1 = 2^u$, then recurse with transforms of length $N_2 = N/2^u$. When all recursive calls are unrolled, we see that the computation is based on transforms of length 2^u (or 2^r at the very end of the recursion). Those are done with RadixTwoCooleyTukeyFFT. We use the notation LargeRadixCooleyTukeyFFT$(A, N, \omega, 2^u)$ for this algorithm.

It is clear that the latter approach specializes to the former when $u = 1$.

Algorithm 16 Radix-two Cooley-Tukey FFT of order $N = 2^u$

1: **Input:** $A = \sum_{i=0}^{N-1} a_i X^i \in \mathcal{R}[X]/(X^N - 1)$ and ω a principal N-th root of unity.
2: **Output:** $\hat{A} = \mathrm{DFT}_{N,\omega}(A) = A(1) + A(\omega)X + \cdots + A(\omega^{N-1})X^{N-1}$
3: **function** RadixTwoCooleyTukeyFFT(A,N,ω)
4: **if** $N = 2$ **then**
5: **return** $a_0 + a_1 + (a_0 - a_1)X$
6: **else**
7: $B_0 \leftarrow a_0 + a_2 X + \cdots + a_{N-2} X^{N/2-1}$
8: $B_1 \leftarrow a_1 + a_3 X + \cdots + a_{N-1} X^{N/2-1}$
9: $\hat{B}_0 \leftarrow \mathsf{RadixTwoCooleyTukeyFFT}(B_0, N/2, \omega^2)$
10: $\hat{B}_1 \leftarrow \mathsf{RadixTwoCooleyTukeyFFT}(B_1, N/2, \omega^2)$
11: $\hat{B}_1 \leftarrow \hat{B}_1(\omega X)$
12: Let $(S_i(Y))_i \in \mathcal{R}[Y]^{N/2}$ be such that $\hat{B}_0(X) + \hat{B}_1(X)Y = \sum_{0 \le i < N/2} S_i(Y) X^i$
13: **for** $0 \le i < N/2$ **do**
14: $\hat{S}_i(Y) \leftarrow \mathsf{RadixTwoCooleyTukeyFFT}(S_i, 2, -1)$
15: **end for**
16: **return** $\sum_{0 \le i < N/2} \hat{S}_i(X^{N/2}) X^i$
17: **end if**
18: **end function**

For $N = 16$, we can illustrate RadixTwoCooleyTukeyFFT by a butterfly diagram given in Figure 6.1. We order the A_i's in the bitreverse order, which is the order obtained in RadixTwoCooleyTukeyFFT after the reordering obtained by the successive application of lines 7 and 8.

The complexity of Algorithm 16 requires to count:

- the additions and subtractions,
- the multiplications,
- the memory accesses.

Denoting by $L(N)$ the additions and subtractions in RadixTwoCooleyTukeyFFT, we have

$$L(N) = N + 2L(N/2)$$

if $N > 2$, and $L(2) = 2$. The resolution of the recursive equation leads to $L(N) = N \log_2 N$.

Denoting by $C(N)$ the multiplications in RadixTwoCooleyTukeyFFT, we have

$$C(N) = N/2 + 2C(N/2)$$

if $N > 2$, corresponding to computation of $\hat{B}_1(\omega X)$ on Line 11 of RadixTwoCooleyTukeyFFT ($\hat{B}_1$ being a polynomial of degree $N/2$), and $C(2) = 0$. The resolution of the recursive equation leads to $C(N) = \frac{N}{2}(\log_2 N - 1)$.

In RadixTwoCooleyTukeyFFT, the algorithmic aspects related to the storage of the roots of unity are not detailed. If we store the powers of ω in an array of length N, depending on how this storage is done, the distance between roots of unity may impact the complexity. Since transforms of sizes 2 or 4 imply less than 1 multiplication by a root of unity on a multitape Turing machine, we go to transforms of size 8 to give an example. We compute $N/8$ transforms of size 8. A

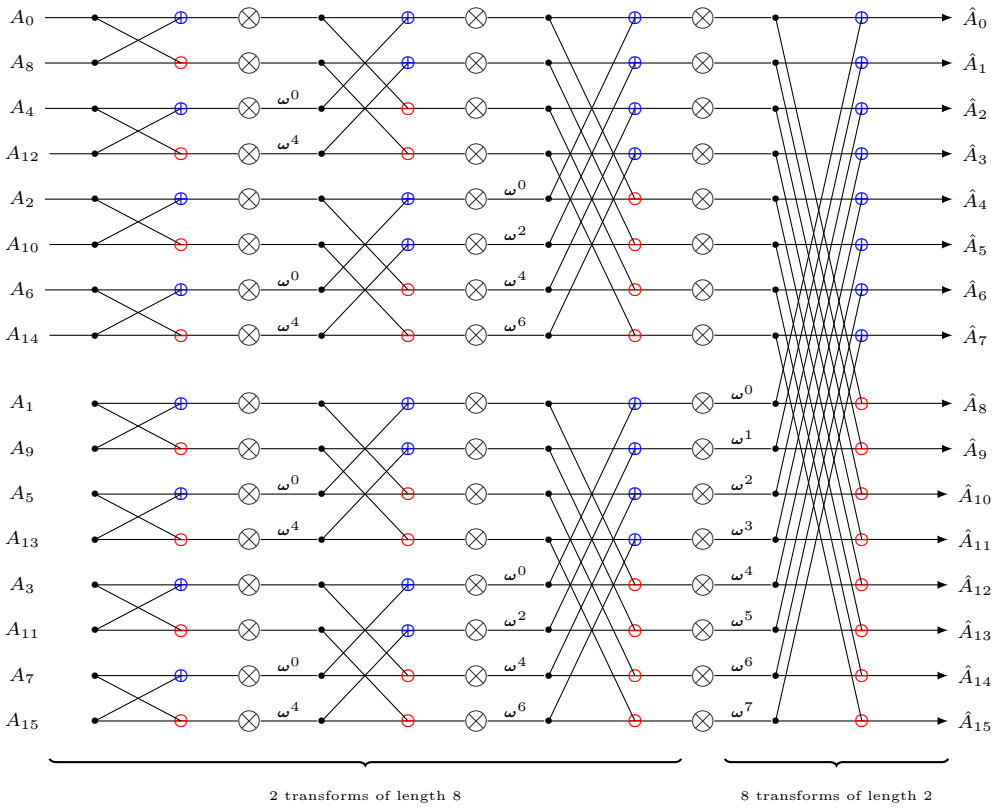

Figure 6.1: Butterfly of a 16-point FFT

transform of size 8 implies to have access to 8-th roots of unity. If the distance between two 8-th roots of unity is $O(N/8)$, then we have an additional cost of the order $O(N^2)$. Thus, we have to store the roots of unity so that the distance between high powers of a root of unity should be small. Two solutions are possible.

- We store them in an array of length N. For each recursive call, we assume that we construct an array of length $N/2$ containing the powers of ω^2.
- Alternatively, we precompute an array of size $N + N/2 + N/4 + \cdots + 2$ containing the N powers of ω, then the $N/2$ powers of ω^2, ..., the two powers of -1.

In both cases, the cost of the memory accesses is $O(N \log_2 N)$, because the cost of enumerating the $N/2^k$-th roots of unity, for some $k \leq \log_2 N$, is equal to $O(2^k)$.

6.5 Complexity of integer multiplication

Notation 6.6. *We denote by $\mathsf{M}(n)$ the cost of the multiplication of two n-bit integers in the deterministic multitape Turing model [47], also called bit complexity.*

By combining the evaluation-interpolation scheme of §6.1 with FFT-based multi-evaluation and interpolation as in §6.4, we obtain quasi-linear integer multiplication algorithms. We identify several tasks whose costs contribute to the bit complexity of such algorithms.

- conversion of the input integers to polynomials in $\mathcal{R}[X]$;
- multiplications by roots of unity in the FFT computation;
- linear operations in the FFT computation (additions, etc);
- point-wise products of elements of $\mathcal{R}$. Recursive calls to the integer multiplication algorithm are of course possible;
- recovery of the resulting integer from the computed polynomial.

Algorithm 14 chooses η a power of two so that the first and last steps above have linear complexity (at least provided that elements in $\mathcal{R}$ are represented in a straightforward way). If we go into more detail, $\mathsf{M}(n)$ then expresses as $\mathsf{M}(n) = O(C(N) \cdot K_{\mathrm{FFT}}(\mathcal{R})) + O(N \cdot K_{\mathrm{PW}}(\mathcal{R}))$, with the following notations.

- $K_{\mathrm{FFT}}(\mathcal{R})$ denotes the cost for the multiplications by powers of ω in $\mathcal{R}$ that occur within the FFT computation.
- $K_{\mathrm{PW}}(\mathcal{R})$ denotes the binary cost for the point-wise products in $\mathcal{R}$.

The costs $K_{\mathrm{PW}}(\mathcal{R})$ and $K_{\mathrm{FFT}}(\mathcal{R})$ are not necessarily equal.

6.6 Choice of the base ring

Depending on $\mathcal{R}$, the bit complexity estimates of §6.5 can be made more precise. Some rings have special principal roots of unity that allow faster operations (multiplication in $\mathcal{R}$) than others. Several choices for $\mathcal{R}$ are discussed in [52]. We describe their important characteristics when the goal is to multiply two n-bit integers.

We denote by $\log^*$ the iterated logarithm function, defined recursively by $\log^* x = 0$ for any real number $x \leq 1$, and by $\log^* x = 1 + \log^*(\log x)$ for $x \geq 1$.

6.6.1 Complex FFT

The choice $\mathcal{R} = \mathbb{C}$ might seem natural because roots of unity are plenty. For a polynomial of length N, we can use the principal root of unity $\omega = e^{2i\pi/N}$. For any k such that k and N are coprime, ω^k is an N-th principal root of unity.

The precision required calls for some analysis. The asymptotic complexity depends on the best choice for this precision. A precision of $t = O(\log_2 n)$ bits is compatible with a transform length N where N is the smallest power of two larger than $(\frac{n}{\log_2 n})$ (see [52, §3]), in the sense that the polynomials that we multiply can be represented on tN bits and the product would not be correct if t were smaller (thus, $t = O(\log_2 n)$ is optimal). We represent the elements of $\mathbb{C}$ as linear polynomials with coefficients in $\mathbb{Z}$. Using the Kronecker substitution for the multiplications, we are led to store the elements of $\mathbb{C}$ on $4t + 2 = O(t)$ bits. The $4t + 2$ bits include the zero padding due to the Kronecker substitution.

Assuming that we use RadixTwoCooleyTukeyFFT, we have $N \log_2 N$ additions and subtractions in $\mathbb{C}$, whose cost is $O(t)$ each. Thus, the binary cost of the additions and subtractions is $O(Nt \log_2 N) = O(n \log_2 n)$.

We have $N \log_2 N$ memory accesses to elements of size $O(t)$. Thus, the binary cost of the memory accesses is also $O(n \log_2 n)$.

Finally, we have $O(N \log_2 N)$ multiplications of integers of size $O(t)$. Thus, the recursive equation corresponding to the FFT over the complex field is:

$$\mathsf{M}(n) = (3N \log_2 N + N)\mathsf{M}(t) + O(n \log_2 n).$$

We simplify the equation by considering that $n \log_2 n \leq N(\log_2 N)\mathsf{M}(t)$: we use the fact that $\mathsf{M}(t)$ is more expensive than the cost of an addition, which is linear in t. Thus, we have

$$\mathsf{M}(n) = O(N \log_2 N \cdot \mathsf{M}(\log_2 n)) = O(n \cdot \mathsf{M}(\log_2 n)),$$

so that

$$\mathsf{M}(n) = 2^{O(\log_2^* n)} \cdot n \cdot \log_2 n \cdot \log_2^{(2)} n \cdot \log_2^{(3)} n \cdot \ldots,$$

where the number of recursive calls is $\log_2^* n + O(1)$.

6.6.2 Schönhage-Strassen algorithm

Schönhage and Strassen proposed in [52] and in [51] two recursive algorithms to multiply elements of a ring of the form $\mathbb{Z}/(2^m + 1)\mathbb{Z}$, where m is a power of two. Note that, in particular, the Schönhage-Strassen algorithms can be used to compute the product of n-bit integers for any $n \leq \frac{m}{2}$. We describe these algorithms in the following paragraphs.

These algorithms are parametrized by integers t, k and e, depending on m. We split the m-bit elements of $\mathbb{Z}/(2^m + 1)\mathbb{Z}$ into pieces of size t, obtaining polynomials of length denoted by $N = m/t$. We consider that the coefficients of these polynomials live in $\mathcal{R} = \mathbb{Z}/\big(2^k(2^e + 1)\big)\,\mathbb{Z}$, with e chosen such that N divides e and k "negligible", compared to m (say $k \approx \log_2 m$).

We multiply elements of $\mathcal{R}$ using Chinese remainder theorem (CRT), which states that

$$\mathcal{R} \simeq \mathbb{Z}/2^k\mathbb{Z} \times \mathbb{Z}/(2^e + 1)\mathbb{Z}.$$

- To muliply polynomials of length N over $\mathbb{Z}/2^k\mathbb{Z}$, we use Kronecker substitution and any subquadratic algorithm to multiply integers.
- In $\mathbb{Z}/(2^e + 1)\mathbb{Z}$, we use the fact that, since 2 is a $2e$-th principal root of unity in $\mathbb{Z}/(2^e + 1)\mathbb{Z}$ and N divides e, $2^{e/N}$ is also a $2N$-th principal root of unity in $\mathbb{Z}/(2^e + 1)\mathbb{Z}$. Thus, we perform an FFT for the multiplication of polynomials of length N over $\mathbb{Z}/(2^e + 1)\mathbb{Z}$.

During the FFT, the cost of linear operations such as additions, subtractions or memory accesses is

$$O(Nt \log_2 N) = O(m \log_2 m).$$

Since $2^{2t/N}$ is a principal $2N$-th principal root of unity in $\mathbb{Z}/(2^e+1)\mathbb{Z}$, all the multiplications in RadixTwoCooleyTukeyFFT are multiplications by powers of two: on a binary machine, this cost is linear. Thus, the cost of the multiplications during the FFT is also $O(m \log_2 m)$. The difficult part is concentrated in the point-wise product of the evaluation-interpolation scheme.

We can summarize the procedure as follows.

1. Two elements x and y of $\mathcal{R}$ are decomposed in $\mathbb{Z}/2^k\mathbb{Z}$ and $\mathbb{Z}/(2^e+1)\mathbb{Z}$: we obtain polynomials to multiply over two different rings;

2. we compute the products in $\mathbb{Z}/2^k\mathbb{Z}$ using Karatsuba's algorithm: since $k \approx \log_2 m$, we multiply polynomials of length N and whose coefficients have bit-length k, which has a cost $O(\sqrt{m} \log_2 m)^{\log_2 3} = o(m \log_2 m)$ if we use Kronecker-substitution and Karatsuba's algorithm;

3. we compute the products in $\mathbb{Z}/(2^e+1)\mathbb{Z}$ by calling recursively the Schönhage-Strassen algorithm;

4. we recompose the two products in $\mathcal{R}$.

The cost of the decomposition and the recomposition is linear, using the Bézout identity

$$(2^e+1) - 2^k \cdot 2^{e-k} = 1.$$

We precise in the following how to choose the parameters t, k and e such that the complexity of the product modulo (2^m+1) is $O(m \log m \log^{(2)} m)$.

- In [52], the parameters are chosen as follows:
  - t is the largest power of two such that $t \leq \sqrt{m}$;
  - $k = \log_2 N$;
  - $e = 2t$;
  - $\mathcal{R} = \mathbb{Z}/\big(2^k(2^e+1)\big)\,\mathbb{Z}$.

  First, note that $N = m/t \geq \sqrt{m}$ and that $N \mid 2t$. Second, since $N(2^{2t}+1) > N \cdot 2^{2t}$, all the coefficients of the product modulo X^N+1 of two polynomials of length N and whose coefficients have bit-length at most t can be represented in the ring $\mathcal{R}$.

  Denoting by $\mathsf{C}(m)$ the complexity of the multiplication of two elements of $\mathbb{Z}/(2^m+1)\mathbb{Z}$, this leads to the equation

$$\mathsf{C}(m) = O(m \log_2 m) + N\mathsf{C}(2t).$$

  For $m \geq 4$, there exists a constant d such that

$$\frac{\mathsf{C}(m)}{m \log_2 m} \leq d + \frac{\mathsf{C}(2t)}{t \log_2 m} \leq d + \frac{\mathsf{C}(2t)}{2t \log_2 t} = d + \frac{\mathsf{C}(2t)}{(2t \log_2 2t)\left(1 - \frac{1}{\log_2 2t}\right)}. \tag{6.2}$$

First, we have the following intermediate bound on $\frac{\mathsf{C}(m)}{m\log_2 m}$:

$$\forall m \geq 4,\ \frac{\mathsf{C}(m)}{m\log_2 m} \leq d + 2\frac{\mathsf{C}(2t)}{2t\log_2 2t} = O(2^{\log_2^{(2)} m}) = O(\log_2 m). \tag{6.3}$$

For $m < 4$, we have $\frac{\mathsf{C}(m)}{m\log_2 m} = O(1)$, which means that we have the bound above for any m.

Returning to Equation (6.2), we have for $m \geq 4$,

$$\frac{\mathsf{C}(m)}{m\log_2 m} \leq d + \frac{\mathsf{C}(2t)}{t\log_2 m} \leq d + \frac{\mathsf{C}(2t)}{(2t\log_2 2t)\left(1-\frac{1}{\log_2 2t}\right)} \leq d + \frac{\mathsf{C}(2t)}{2t\log_2 2t}\left(1+\frac{2}{\log_2 2t}\right).$$

Thus, reusing the intermediate bound obtained in Equation (6.3), there exists a constant d' such that

$$\frac{\mathsf{C}(m)}{m\log_2 m} \leq d' + \frac{\mathsf{C}(2t)}{2t\log_2 2t}.$$

By developing the recursive equation, we have

$$\frac{\mathsf{C}(m)}{m\log_2 m} = O(\log_2^{(2)} m)$$

and we obtain the complexity $\mathsf{C}(m) = O(m\cdot\log_2 m\cdot\log_2^{(2)} m)$. It follows that the multiplication of two n-bit integers, for $n \leq \frac{m}{2}$, using the Schönhage-Strassen algorithm has cost of $O(n\cdot\log_2 n\cdot\log_2^{(2)} n)$.

- In [51], Schönhage proposes an alternative avoiding the use of the CRT. We can summarize the choice for the parameters as follows:
  - t is the largest power of two such that $t \leq \sqrt{m}\log_2 m$;
  - $k = 0$;
  - e is the least integer such that $2^e \geq N2^{2t}$ and that $N \mid e$;
  - $\mathcal{R} = \mathbb{Z}/(2^e+1)\,\mathbb{Z}$.

  Note that we have $e \leq 2t + N + \log_2 N \leq 2t + 2N$.

  Still denoting by $\mathsf{C}(m)$ the complexity of the multiplication of two elements of $\mathbb{Z}/(2^m+1)\mathbb{Z}$, we have the equation

$$\mathsf{C}(m) = O(m\log_2 m) + N\mathsf{C}(e).$$

  For $m \geq 4$, there exists a constant d such that

$$\frac{\mathsf{C}(m)}{m\log_2 m} \leq d + \frac{\mathsf{C}(e)}{t\log_2 m} \leq d + \frac{\mathsf{C}(e)}{e\log_2 e}\left(\frac{2(t+N)\log_2 2(t+N)}{t\log_2 m}\right). \tag{6.4}$$

  We have

$$\begin{aligned}\frac{2(t+N)\log_2 2(t+N)}{t\log_2 m} &= 2\frac{\log_2 t}{\log_2 m} + 2\frac{\log_2(2+2N/t)}{\log_2 m} + 2\frac{N\log_2 t}{t\log_2 m} + 2\frac{N\log_2(2+2N/t)}{t\log_2 m}\\ &= 2\frac{\log_2 t}{\log_2 m} + O\left(\frac{1}{\log_2 m}\right)\\ &= 1 + O\left(\frac{\log_2^{(2)} m}{\log_2 m}\right).\end{aligned}$$

We obtain the following equation:

$$\frac{\mathsf{C}(m)}{m\log_2 m} \leq d + \frac{\mathsf{C}(e)}{e\log_2 e}\left(1 + O\left(\frac{\log_2^{(2)} m}{\log_2 m}\right)\right). \tag{6.5}$$

There exists m_0 such that for $m \geq m_0$, $O\left(\frac{\log_2^{(2)} m}{\log_2 m}\right) \leq 1$. We obtain the following intermediate bound:

$$\frac{\mathsf{C}(m)}{m\log_2 m} \leq d + 2\frac{\mathsf{C}(e)}{e\log_2 e} \leq O(2^{\log_2^{(2)} m}) = O(\log_2 m). \tag{6.6}$$

We inject the bound of Equation (6.6) into Equation (6.5):

$$\frac{\mathsf{C}(m)}{m\log_2 m} \leq d + \frac{\mathsf{C}(e)}{e\log_2 e}\left(1 + O\left(\frac{\log_2^{(2)} m}{\log_2 m}\right)\right) = O(\log_2^{(2)} m) + \frac{\mathsf{C}(e)}{e\log_2 e}. \tag{6.7}$$

By developping the previous inequality, we obtain

$$\frac{\mathsf{C}(m)}{m\log_2 m} \leq O(\log_2^{(2)} m)^2.$$

By injecting again the bound obtained for $\frac{\mathsf{C}(m)}{m\log_2 m}$ into Equation (6.5), we obtain the wanted result.

- It is often considered, as in [27], that it is sufficient to take
  - N the largest power of two such that $N \leq \sqrt{m}$;
  - $k = 0$;
  - e minimal such that $e \geq 2t + \log_2 N$ and that $N \mid e$;
  - $\mathcal{R} = \mathbb{Z}/(2^e + 1)\,\mathbb{Z}$.

  For practical implementations, these parameters can be chosen. However, they do not correspond to an algorithm with an asymptotic complexity equal to the complexity of Schonhage-Strassen algorithm. Indeed, since N divides e and, consequently, $2t$, we have $e = 2t+N \geq 2t+t = 3t$. We denote by $\mathsf{C}(m)$ the complexity of the multiplication algorithm using recursively these parameters to multiply m-bit integers. There exists a constant d such that

$$\mathsf{C}(m) \geq dm\log_2 m + N\mathsf{C}(3t).$$

  We denote by $\mathsf{L}(m)$ the solution of

$$\mathsf{L}(m) = dm\log_2 m + N\mathsf{L}(3t).$$

  $\mathsf{L}(m)$ is a lower bound on the cost $\mathsf{C}(m)$ of the algorithm multiplying integers without CRT. We have

$$\mathsf{L}(m) \geq dm\log_2 m + \frac{3}{2}dm\log_2 m + \cdots + \left(\frac{3}{2}\right)^{\log_2^{(2)} m} dm\log_2 m.$$

  There exists d' such that $\mathsf{L}(m) \geq d'm(\log_2 m)^{1+\log_2(\frac{3}{2})}$, which is not the correct complexity. In the literature [27, 59, 38], this detail is not covered.

Chapter 7

Fürer-like algorithms

Fürer proposed two distinct algorithms: one in [25] and, some 20 years later, in [26]. The scheme proposed in [25] relies on the assumption that there exist infinitely many Fermat primes, which is unfortunately widely believed to be wrong.

7.1 Fürer's algorithm in a complex modular ring

In this section, we describe an improved version of the algorithm developed in [26]. In particular, the ring $\mathcal{R}$, described in [26, Section 4], is a complex ring of the form $\mathbb{C}[x]/(x^{2^\lambda}+1)$. We prove here that we can work in a real ring of the form $\mathbb{R}[x]/(x^{2^\lambda}+1)$.

Fürer's algorithm combines the advantages of the complex field $\mathbb{C}$ and of the modular ring $\mathbb{Z}/(2^e+1)\mathbb{Z}$, for a multiplication of n-bit integers. These rings have orthogonal advantages and drawbacks.

- In the modular ring, there exists a particular root of unity, namely 2, for which the multiplications by its powers are linear in e, because they are equivalent to negacyclic shifts. In this ring, $e \approx \sqrt{n}$.
- In the complex field, there exists a particular power of the roots of unity for which the multiplications are also "easy" to compute: these roots are the powers of i. This can be used in a radix-4 FFT: the transforms of size 4 involve multiplications by powers of i, which are linear. Although there are fewer particular powers of roots of unity, the bit-length of the multiplications is approximately $\log_2 n$ which is less than $\sqrt{n}$, by an order of magnitude. Thus, the recursive calls are done on smaller sizes, but the quantity of expensive multiplications is larger.

Fürer proposed in [26] to use the ring $\mathcal{R} = \mathbb{C}[x]/(x^{2^\lambda}+1)$, which has a natural principal $2^{\lambda+1}$-th root of unity, namely x. Let $\zeta = \exp(i\pi/2^\lambda)$ and $\mathcal{R}_j = \mathbb{C}[x]/\left(x-\zeta^{2j+1}\right)$. The ring $\mathcal{R}$ is isomorphic to $\prod_{0\le j<2^\lambda} \mathcal{R}_j$. For any integer N which is a multiple of $2^{\lambda+1}$ (and in particular for powers of two of higher order), we define ω_N as the unique element of $\mathcal{R}$ that maps to $\exp(2(2j+1)i\pi/N)$ in $\mathcal{R}_j$. Lagrange interpolation can be used to compute ω_N explicitly. We verify easily that:

- ω_N is a principal N-th root of unity;
- by construction, $\omega_N^{N/2^{\lambda+1}}$ maps to $x = \exp((2j+1)i\pi/2^\lambda)$ in $\mathcal{R}_j$, so that $\omega_N^{N/2^{\lambda+1}} = x$ in $\mathcal{R}$.

The latter point implies that among powers of ω_N, some enjoy particularly easy operations. When N is clear from the context, ω_N is simply denoted by ω.

We consider a radix-$2^{\lambda+1}$ Cooley-Tukey FFT described by Algorithm 17. The key observation is that the "inner" transforms of length $2^{\lambda+1}$ that are computed on Line 14 within the recursion involve multiplications by roots of unity which are then multiplications by powers of $\omega^K = x \in \mathcal{R}$, and therefore inexpensive, since they can done via negacyclic shifts. We can count the few remaining *expensive* multiplications that occur within the recursion when $N \geq 2^{\lambda+1}$. Those correspond to the scaling operation $\hat{B}_k \leftarrow \hat{B}_k(\omega^k X)$ on Line 15 of Algorithm 17. Their count $E(N)$ satisfies

$$E(N) = 2^{\lambda+1} E\left(\frac{N}{2^{\lambda+1}}\right) + N, \tag{7.1}$$

because we have $K = N/2^{\lambda+1}$ polynomials of degree $J = 2^{\lambda+1}$ that are composed with a power of ω ($K \cdot J = N$ multiplications) followed by J "outer" transforms of size K on Line 19. It follows that $E(N) = N(\lceil \log_{2^{\lambda+1}} N \rceil - 1)$.

Algorithm 17 Radix-$2^{\lambda+1}$ Cooley-Tukey FFT of order N, a power of two

1: **Input:** $A = \sum_{i=0}^{N-1} a_i X^i \in \mathcal{R}[X]/(X^N - 1)$ and ω a principal N-th root of unity.
2: **Output:** $\hat{A} = \mathrm{DFT}_{N,\omega}(A) = A(1) + A(\omega)X + \cdots + A(\omega^{N-1})X^{N-1}$
3: **function** LargerRadixCooleyTukeyFFT(A,N,λ,ω)
4: **if** $N \leq 2$ **then**
5: **return** $a_0 + a_1 + (a_0 - a_1)X$
6: **else**
7: $J \leftarrow 2$
8: **if** $N \geq 2^{\lambda+1}$ **then**
9: $J \leftarrow 2^{\lambda+1}$
10: **end if**
11: $K \leftarrow N/J$
12: **for** $0 \leq k < K$ **do**
13: Let B_k be the polynomial given by the coefficients $(a_{jK+k})_{0 \leq j < J}$
14: $\hat{B}_k \leftarrow$ RadixTwoCooleyTukeyFFT(B_k, J, ω^K)
15: $\hat{B}_k \leftarrow \hat{B}_k(\omega^k X)$
16: **end for**
17: Let $(S_j(Y))_j \in \mathcal{R}[Y]^J$ be such that $\sum_{0 \leq k < K} \hat{B}_k(X) Y^k = \sum_{0 \leq j < J} S_j(Y) X^j$
18: **for** $0 \leq j < J$ **do**
19: $\hat{S}_j(Y) \leftarrow$ LargerRadixCooleyTukeyFFT$(S_j, K, \lambda, \omega^J)$
20: **end for**
21: **return** $\sum_{0 \leq j < J} \hat{S}_j(X^J) X^j$
22: **end if**
23: **end function**

We describe elements of $\mathbb{C}$ with a fixed precision. To multiply n-bit integers, Fürer selects $\lambda = \lceil \log_2^{(2)} n \rceil$ and proves that precision $O(\log_2 n)$ is sufficient for the coefficients of the elements of $\mathcal{R}$ that occur in the computation. The integers to be multiplied are split into pieces of $2^{2\lambda-1}$ bits. Each piece of $2^{2\lambda-1}$ bits is then transformed into a polynomial of length $2^{\lambda-1}$ whose coefficients are encoded on 2^λ bits. These polynomials are seen as elements of $\mathcal{R}$. Moreover, the transform length is $N = 4n/(\log_2 n)^2$. Since we need to have $2^{\lambda+1} \mid 2N$, we should have $n \geq 2^{10}$. This decomposition is described in Algorithm 18 (FurerComplexMul).

Algorithm 18 Multiplication of integers with Fürer's algorithm

Input: a and b two n-bit integers, where n is a power of two and $n \geq 2^{10}$
Output: $a \cdot b \mod (2^{2n}+1) = a \cdot b$
function FurerComplexMul(a,b,n)
  Let $\lambda = \lceil \log_2^{(2)} n \rceil$, $\eta = 2^{2^{2\lambda-1}}$, $N = 2n/\log_2 \eta = 2n/2^{2\lambda-1}$,
  Let $\mathcal{R} = \mathbb{C}[x]/(x^{2^\lambda}+1)$, and $\omega = \omega_{2N}$ as in §7.1.
  Let $A_0(X) \in \mathbb{Z}[X]$ with non negative coefficients less than η be such that $A_0(\eta) = a$; define $B_0(X)$ likewise.
  Let $\tilde{A}(X,x) \in \mathbb{C}[X,x]$ be such that $\tilde{A}(X, 2^{2^\lambda}) = A_0(X)$. Define $\tilde{B}$ likewise.
  Map $\tilde{A}$ and $\tilde{B}$ to polynomials A and B in $\mathcal{R}[X]/(X^N+1)$
  $\hat{A} \leftarrow$ LargerRadixCooleyTukeyFFT$(A(\omega X), N, \lambda, \omega^2)$ ▷ computes Half-DFT$_{N,\omega}(A)$
  $\hat{B} \leftarrow$ LargerRadixCooleyTukeyFFT$(B(\omega X), N, \lambda, \omega^2))$ ▷ computes Half-DFT$_{N,\omega}(B)$
  $\hat{C} \leftarrow$ PointwiseProduct$(\hat{A}, \hat{B})$
  $C \leftarrow \frac{1}{N}$LargerRadixCooleyTukeyFFT$(\hat{C}(\omega^{-1}X), N, \lambda, \omega^{-2})$ ▷ computes Half-IFT$_{N,\omega}(\hat{C})$
  Lift C to $\tilde{C} \in \mathbb{C}[X,x]$
  return $\tilde{C}(\eta, 2^\lambda)$
end function

7.2 Precision of the modular ring and complexity analysis

We show in this section that there exists in $\mathcal{R} = \mathbb{C}[x]/(x^{2^\lambda}+1)$ a principal root of unity $\omega(x)$ such that the coefficients of ω are all in $\mathbb{R}$, when $\lambda \geq 1$. Consequently, we can represent, using this principal root of unity, all the coefficients of the elements of $\mathcal{R}$ used in Fürer's algorithm assuming that they live in $\mathbb{R}$. When $\lambda = 0$, the proposed construction does not work. We describe the precision required to store elements of $\mathcal{R}$ assuming that the coefficients are in $\mathbb{R}$ and we describe the complexity analysis given by Fürer in [26].

7.2.1 From a complex ring to real ring

First, we need an explicit expression of a polynomial obtained by Lagrange interpolation at the points $(\zeta^{2j+1})_{0 \leq j < 2^\lambda}$. Therefore, we give the expression of the corresponding Lagrange basis.

Lemma 7.1. *The Lagrange polynomial basis $(\ell_j)_{0 \leq j < 2^\lambda}$ corresponding to the evaluation points $(\zeta^{2j+1}))_{0 \leq j < 2^\lambda}$ (i.e., evaluation in the $\mathcal{R}_j$'s) is given by*

$$\ell_j = \frac{1}{2^\lambda} \sum_{t=0}^{2^\lambda - 1} \left(\frac{x}{\zeta^{2j+1}} \right)^t .$$

Proof. For any $j \in \{0, \ldots, 2^\lambda - 1\}$,

$$\ell_j = \frac{f(x)}{f'(\zeta^{2j+1})(x - \zeta^{2j+1})},$$

with $f(x) = \prod_{k=0}^{2^\lambda - 1}(x - \zeta^{2k+1}) = x^{2^\lambda} + 1$.

Note that $(\zeta^{2j+1})^{2^\lambda} = (\zeta^{2^\lambda})^{2j+1} = (-1)^{2j+1} = -1$. Thus, we have

$$f(x) = 1 + x^{2^\lambda} = 1 - \left(\frac{x}{\zeta^{2j+1}} \right)^{2^\lambda} \text{ and } f'(\zeta^{2j+1}) = -\frac{2^\lambda}{\zeta^{2j+1}}.$$

Therefore,

$$\ell_j = \frac{1}{2^\lambda} \frac{\left(\frac{x}{\zeta^{2j+1}}\right)^{2^\lambda} - 1}{\frac{x}{\zeta^{2j+1}} - 1} = \frac{1}{2^\lambda} \sum_{t=0}^{2^\lambda - 1} \left(\frac{x}{\zeta^{2j+1}}\right)^t.$$

□

We suppose that we have N such that $2^\lambda \mid N$. Let $\sigma = \exp(i\pi/N) \in \mathbb{C}$: it is a principal $2N$-th root of unity and $\zeta = \sigma^{N/2^\lambda}$. Let $\omega \in \mathcal{R}$ such that ω maps to σ^{2j+1-2^λ} in $\mathcal{R}_j$. In other words,

$$\omega(\zeta^{2j+1}) = \sigma^{2j+1-2^\lambda} = \frac{\sigma^{2j+1}}{\sigma^{2^\lambda}}.$$

Then, ω is a $2N$-th principal root of unity if and only if $1 - 2^\lambda$ is odd, since $\omega^N = 1$ if and only if if $1-2^\lambda$ is odd. We want to show that the coefficients of ω are real. For this purpose, we compute explicitly these coefficients. Using Lagrange interpolation to compute $\omega = \sum_{0 \le j < 2^\lambda} \sigma^{2j+1-2^\lambda} \ell_j$, we have an explicit expression of its coefficients with the lemma above.

Proposition 7.2. *For $\lambda \ge 1$, the $\omega \in \mathcal{R}$ is a principal $2N$-th root of unity and, for all $k \in \mathbb{Z}$,*

$$\omega^k = \begin{cases} (-x)^{k'} = (-1)^{k' + \lfloor k'/2 \rfloor} x^{k' \bmod 2^\lambda} & \text{if } k = k' N/2^\lambda, \text{ and} \\ \dfrac{1}{2^\lambda} \displaystyle\sum_{t=0}^{2^\lambda - 1} \frac{\sin\left(\frac{k 2^\lambda \pi}{N}\right)}{\sin\left(\frac{k\pi}{N} - \frac{t\pi}{2^\lambda}\right)} x^t & \text{otherwise.} \end{cases}$$

Proof.

- Since ω maps to a principal $2N$-th root of unity in each $\mathcal{R}_j$, then ω is also a principal $2N$-th root of unity in $\mathcal{R} \cong \prod_{j=0}^{2^\lambda - 1} \mathcal{R}_j$.
- Assume $k = k'N/2^\lambda$. Then, in each $\mathcal{R}_j$, we have

$$\omega(\zeta^{2j+1})^k = \left(\frac{\sigma^{2j+1}}{\sigma^{2^\lambda}}\right)^k = \left(\frac{(\sigma^{N/2^\lambda})^{2j+1}}{\sigma^N}\right)^{k'} = (-\zeta^{2j+1})^{k'} \equiv (-x)^{k'} \pmod{x - \zeta^{2j+1}}.$$

  Therefore, in $\mathcal{R}$, we have $\omega^k = (-x)^{k'} = (-1)^{k'} x^{k'} = (-1)^{k' + \lfloor k'/2^\lambda \rfloor} x^{k' \bmod 2^\lambda}$, as $x^{2^\lambda} = -1$.
- If k is not divisible by $N/2^\lambda$, we compute ω^k by Lagrange interpolation:

$$\begin{aligned} \omega^k &= \sum_{j=0}^{2^\lambda - 1} \omega(\zeta^{2j+1})^k \ell_j &&= \sum_{j=0}^{2^\lambda - 1} \frac{\sigma^{k(2j+1)}}{\sigma^{k 2^\lambda}} \ell_j \\ &= \frac{1}{2^\lambda \sigma^{k 2^\lambda}} \sum_{j=0}^{2^\lambda - 1} \sum_{t=0}^{2^\lambda - 1} \left(\frac{\sigma^k}{\zeta^t}\right)^{2j+1} x^t &&= \frac{1}{2^\lambda \sigma^{k 2^\lambda}} \sum_{t=0}^{2^\lambda - 1} \left(\frac{\sigma^k}{\zeta^t}\right) \frac{\left(\frac{\sigma^k}{\zeta^t}\right)^{2^{\lambda+1}} - 1}{\left(\frac{\sigma^k}{\zeta^t}\right)^2 - 1} x^t, \end{aligned}$$

  where the last equality comes from the fact that $(\sigma^k/\zeta^t)^2 \ne 1$, for all $0 \le t < 2^\lambda$.

  Indeed, let us assume that there exists $0 \le t < 2^\lambda$ such that $(\sigma^k/\zeta^t)^2 = 1$. Then, since $\zeta = \sigma^{N/2^\lambda}$, we have $\sigma^{2k - 2tN/2^\lambda} = 1$. As σ is a principal $2N$-th root of unity, we obtain $2k \equiv 2tN/2^\lambda \pmod{2N}$, whence $k \equiv tN/2^\lambda \pmod{N}$ and, finally, $k \equiv 0 \pmod{N/2^\lambda}$, which contradicts the assumption that k is not divisible by $N/2^\lambda$.

Finally, since ζ is a $2^{\lambda+1}$-st root of unity, we can rewrite ω^k as

$$\begin{aligned}\omega^k = & \frac{1}{2^\lambda \sigma^{k2^\lambda}} \sum_{t=0}^{2^\lambda-1} \left(\frac{\sigma^k}{\zeta^t}\right) \frac{\sigma^{k2^{\lambda+1}}-1}{\left(\frac{\sigma^k}{\zeta^t}\right)^2-1} x^t & = \frac{1}{2^\lambda} \sum_{t=0}^{2^\lambda-1} \frac{\sigma^{k2^\lambda}-\sigma^{-k2^\lambda}}{\left(\frac{\sigma^k}{\zeta^t}\right)-\left(\frac{\sigma^k}{\zeta^t}\right)^{-1}} x^t \\ = & \frac{1}{2^\lambda} \sum_{t=0}^{2^\lambda-1} \frac{\mathrm{Im}(\sigma^{k2^\lambda})}{\mathrm{Im}\left(\frac{\sigma^k}{\zeta^t}\right)} x^t & = \frac{1}{2^\lambda} \sum_{t=0}^{2^\lambda-1} \frac{\sin\left(\frac{k2^\lambda\pi}{N}\right)}{\sin\left(\frac{k\pi}{N}-\frac{t\pi}{2^\lambda}\right)} x^t.\end{aligned}$$

□

Given the expression of ω^k, we observe that its coefficients are all real.

Corollary 7.3. *For all $k \in \mathbb{Z}$, $\omega^k \in \mathbb{R}[x]/(x^{2^\lambda}+1)$.*

7.2.2 Norm

In this section, we compute the L_1-norm of the roots of unity ω^k defined in Section 7.2.1.

Definition 7.4. *The L_1-norm of an element $A = \sum_j a_j x^j \in \mathbb{R}[x]/(x^{2^\lambda}+1)$ is defined as*

$$\|A\|_1 = \sum_{0 \le j < 2^\lambda} |a_j|.$$

This norm is used to determine how many bits of precision are required to represent the coefficients of the roots of unity that we consider. In Fürer's article, the norm that is used is the L_2-norm and it is bounded by 2^λ.

Lemma 7.5. *For all $A \in \mathcal{R}[x]/(x^{2^\lambda}+1)$, $\|A \cdot x\|_1 = \|A\|_1$.*

Proof. Writing $A = \sum_{t=0}^{2^\lambda-1}$, we have $A \cdot x = \sum_{t=1}^{2^\lambda-1} a_{t-1}x^t + a_{2^\lambda-1}x^{2^\lambda} = \sum_{t=1}^{2^\lambda-1} a_{t-1}x^t - a_{2^\lambda-1}$, since $x^{2^\lambda} = -1$. Therefore, $\|A \cdot x\|_1 = \sum_{t=0}^{2^\lambda-1} |a_t| = \|A\|_1$. □

Consider ω defined as in Section 7.2.1. We want to bound the L_1-norm of the ω^k's. We use the previous proposition to reduce the problem to the set of ω^k's for which $k \in \{0, \ldots, N/2^\lambda - 1\}$.

Proposition 7.6. *For all $k \in \mathbb{Z}$, let $k_0 = k \bmod (N/2^\lambda)$, with $0 \le k_0 < N/2^\lambda$. Then,*

$$\|\omega^k\|_1 = \|\omega^{k_0}\|_1 = \begin{cases} 1 & \text{if } k_0 = 0, \text{ and} \\ \frac{1}{2^\lambda} \sum_{t=0}^{2^\lambda-1} \frac{\sin\left(\frac{k_0 2^\lambda \pi}{N}\right)}{\sin\left(\frac{k_0\pi}{N}+\frac{t\pi}{2^\lambda}\right)} & \text{otherwise.} \end{cases}$$

Proof. Let $k \in \mathbb{Z}$. Write $k = k_1 N/2^\lambda + k_0$, with $k_1 = \lfloor k2^\lambda/N \rfloor$ and $k_0 = k \bmod (N/2^\lambda) = k - k_1 N/2^\lambda$. Then, according to Proposition 7.2, $\omega^k = (-1)^{k_1}\omega^{k_0}$ and, by Lemma 7.5, we obtain

$$\|\omega^k\|_1 = \|\omega^{k_0}\|_1 = \begin{cases} 1 & \text{if } k_0 = 0, \text{ and} \\ \frac{1}{2^\lambda} \sum_{t=0}^{2^\lambda-1} \frac{\left|\sin\left(\frac{k_0 2^\lambda \pi}{N}\right)\right|}{\left|\sin\left(\frac{k_0\pi}{N}-\frac{t\pi}{2^\lambda}\right)\right|} & \text{otherwise.} \end{cases}$$

Since $0 \leq k_0 < N/2^\lambda$ by construction, $\sin(k_0 2^\lambda \pi/N) \geq 0$. Furthermore, for all $t \in \{1, \ldots, 2^\lambda - 1\}$, we have $-1 < k_0/N - t/2^\lambda < 0$, and

$$\left|\sin\left(\frac{k_0\pi}{N} - \frac{t\pi}{2^\lambda}\right)\right| = \sin\left(\frac{t\pi}{2^\lambda} - \frac{k_0\pi}{N}\right) = \sin\left(\frac{(2^\lambda - t)\pi}{2^\lambda} + \frac{k_0\pi}{N}\right).$$

Additionally, for $t = 0$, we have

$$\left|\sin\left(\frac{k_0\pi}{N} - \frac{t\pi}{2^\lambda}\right)\right| = \sin\left(\frac{k_0\pi}{N}\right).$$

Finally, the result follows from the fact that, for $k_0 \neq 0$,

$$\sum_{t=0}^{2^\lambda-1} \frac{1}{\left|\sin\left(\frac{k_0\pi}{N} - \frac{t\pi}{2^\lambda}\right)\right|} = \sum_{t=1}^{2^\lambda-1} \frac{1}{\sin\left(\frac{(2^\lambda-t)\pi}{2^\lambda} + \frac{k_0\pi}{N}\right)} + \frac{1}{\sin\left(\frac{k_0\pi}{N}\right)} = \sum_{t=0}^{2^\lambda-1} \frac{1}{\sin\left(\frac{k_0\pi}{N} + \frac{t\pi}{2^\lambda}\right)}.$$

□

Proposition 7.7. *Let $N \geq 1$. For $\lambda \geq 1$, we denote by $n(\lambda)$ the maximum*

$$\max_{0 \leq k < N/2^\lambda} (\|\omega^k\|_1).$$

We have $n(\lambda) = \frac{2}{\pi}\lambda \log 2 + O(1)$.

Proof. In the following, we assume that $\lambda \geq 1$. First, we show that the root of unity having the largest L_1-norm is the one for which $k = \frac{N}{2^{\lambda+1}}$. We group the terms of index j and $2^\lambda - 1 - j$. We want the k for which the sum $\frac{1}{\sin\left(\frac{k_0\pi}{N} + \frac{t\pi}{2^\lambda}\right)} + \frac{1}{\sin\left(\frac{(t+1)\pi}{2^\lambda} - \frac{k_0\pi}{N}\right)}$ is maximal.

For $t \in \{0, \ldots, 2^\lambda - 1\}$, we define the function $f_t(u)$ for $-\frac{1}{2} < u < \frac{1}{2}$ as

$$f_t : u \mapsto \frac{\cos(u\pi)}{\sin\left(\left(t + \frac{1}{2} + u\right)\frac{\pi}{2^\lambda}\right)} + \frac{\cos(u\pi)}{\sin\left(\left(t + \frac{1}{2} - u\right)\frac{\pi}{2^\lambda}\right)}.$$

Then, for all $0 < k_0 < N/2^\lambda$, we have

$$\|\omega^{k_0}\|_1 = \frac{1}{2^\lambda} \sum_{t=0}^{2^{\lambda-1}-1} f_t\left(\frac{k_0 2^\lambda}{N} - \frac{1}{2}\right).$$

We let g_t be

$$g_t : u \mapsto \frac{\cos(u\pi)}{\sin\left(\left(t + \frac{1}{2} + u\right)\frac{\pi}{2^\lambda}\right)}.$$

We have

$$f_t(u) = g_t(u) + g_t(-u).$$

Thus, f_t is an even function and $f_t'(0) = 0$. We will prove that it is the only zero for $u \in]-\frac{1}{2}, \frac{1}{2}[$ and that it is a maximum. We restrict the problem to the interval $]0, \frac{1}{2}[$ by using the fact that f_t is an even function.

We want to prove that, for $u \in]0, \frac{1}{2}[$, we have

$$0 \geq f_t(u) - f_t(0) = \frac{2\cos(u\pi)\cos(\frac{u\pi}{2^\lambda})\sin^2\left(\left(t+\frac{1}{2}\right)\frac{\pi}{2^\lambda}\right) - 2\left(\sin^2\left(\left(t+\frac{1}{2}\right)\frac{\pi}{2^\lambda}\right) - \sin^2\left(\frac{u\pi}{2^\lambda}\right)\right)}{\sin\left(\left(t+\frac{1}{2}+u\right)\frac{\pi}{2^\lambda}\right)\sin\left(\left(t+\frac{1}{2}-u\right)\frac{\pi}{2^\lambda}\right)\sin\left(\left(t+\frac{1}{2}\right)\frac{\pi}{2^\lambda}\right)}.$$

Denoting by h_t the function

$$h_t : u \mapsto \cos(u\pi)\cos(\frac{u\pi}{2^\lambda}) + \frac{\sin^2\left(\frac{u\pi}{2^\lambda}\right)}{\sin^2\left(\left(t+\frac{1}{2}\right)\frac{\pi}{2^\lambda}\right)} - 1,$$

we have

$$f_t(u) - f_t(0) = \frac{2\sin^2\left(\left(t+\frac{1}{2}\right)\frac{\pi}{2^\lambda}\right)h_t(u)}{\sin\left(\left(t+\frac{1}{2}+u\right)\frac{\pi}{2^\lambda}\right)\sin\left(\left(t+\frac{1}{2}-u\right)\frac{\pi}{2^\lambda}\right)\sin\left(\left(t+\frac{1}{2}\right)\frac{\pi}{2^\lambda}\right)}$$

and, therefore, we only need to prove that for $u \in]0, \frac{1}{2}[$, $h_t(u) \leq 0$. Since sine is an increasing function on $]0, \pi/2[$, we have $h_t(u) \leq h_0(u)$ for any $u \in]0, 1/2[$.

We observe that $h_0(0) = 0$ and $h_0(1/2) = 0$. Thus, if we prove that the sign of h_0 is constant on $]0, 1/2[$ and that it is increasing on $]1/2 - \epsilon, 1/2[$, we prove that it is negative on this interval. To do that, we prove that h_0' has at most one zero in this interval, which implies that h_0 has no zero in the same interval.

We have

$$\forall u \in]0, 1/2[,\ h_0'(u) = -\pi\sin(u\pi)\cos\left(\frac{u\pi}{2^\lambda}\right) - \frac{\pi}{2^\lambda}\sin\left(\frac{u\pi}{2^\lambda}\right)\cos(u\pi) + \frac{2\pi}{2^\lambda}\frac{\cos\left(\frac{u\pi}{2^\lambda}\right)\sin\left(\frac{u\pi}{2^\lambda}\right)}{\sin^2\left(\frac{\pi}{2^{\lambda+1}}\right)}.$$

Thus,

$$\forall u \in]0, 1/2[,\ h_0'(u) = \cos\left(\frac{u\pi}{2^\lambda}\right)\sin\left(\frac{u\pi}{2^\lambda}\right)\left(-\pi\frac{\sin(u\pi)}{\sin\left(\frac{u\pi}{2^\lambda}\right)} - \frac{\pi}{2^\lambda}\frac{\cos(u\pi)}{\cos\left(\frac{u\pi}{2^\lambda}\right)} + \frac{2\pi}{2^\lambda}\frac{1}{\sin^2\left(\frac{\pi}{2^{\lambda+1}}\right)}\right).$$

The functions $x \mapsto \frac{\sin(u\pi)}{\sin\left(\frac{u\pi}{2^\lambda}\right)}$ and $x \mapsto \frac{\cos(u\pi)}{\cos\left(\frac{u\pi}{2^\lambda}\right)}$ are both decreasing functions on $]0, 1/2[$. Consequently, h_0' has at most one zero on $]0, 1/2[$.

Moreover, we have $h_0'(1/2) > 0$, since

$$2 - 2^\lambda \sin\left(\frac{\pi}{2^{\lambda+1}}\right) > 0.$$

Consequently, $h_0(u) < 0$ on $]0, 1/2[$.

Thus, the power of ω having the largest L_1 norm is the $\frac{N}{2^{\lambda+1}}$-th power. We have

$$\begin{aligned}\|\omega^{\frac{N}{2^{\lambda+1}}}\|_1 &= \frac{1}{2^{\lambda-1}}\sum_{t=0}^{2^{\lambda-1}-1}\frac{1}{\sin(\frac{\pi}{2^{\lambda+1}} + \frac{t\pi}{2^\lambda})} \\ &= \frac{1}{2^{\lambda-1}}\sum_{t=1}^{2^{\lambda-1}-1}\frac{1}{\sin(\frac{\pi}{2^{\lambda+1}} + \frac{t\pi}{2^\lambda})} + \frac{1}{2^{\lambda-1}}\frac{1}{\sin(\pi/2^{\lambda+1})}.\end{aligned}$$

We use the fact that the function $\sin(x)$ grows with x for $x \in [0; \pi/2]$ to compare the sum with

an integral:

$$\begin{aligned}
\sum_{t=0}^{2^\lambda-1} \frac{1}{2^\lambda \sin(\frac{(t+1)\pi}{2^\lambda} - \frac{\pi}{2^{\lambda+1}})} &\leq \frac{1}{2^{\lambda-1}} \sum_{t=1}^{2^{\lambda-1}-1} \int_{t-1}^{t} \frac{1}{\sin(\frac{\pi}{2^{\lambda+1}} + \frac{u\pi}{2^\lambda})} \mathrm{d}u + \frac{1}{2^{\lambda-1}} \frac{1}{\sin(\pi/2^{\lambda+1})} \\
&\leq \frac{1}{2^{\lambda-1}} \int_{\pi/2^{\lambda+1}}^{\pi/2-\pi/2^{\lambda+1}} \frac{2^\lambda}{\pi \sin(x)} \mathrm{d}x + \frac{1}{2^{\lambda-1}} \frac{1}{\sin(\pi/2^{\lambda+1})} \\
&\leq \frac{2}{\pi} \left[\log\left(\sqrt{\frac{1-\sin(\pi/2^{\lambda+1})}{1+\sin(\pi/2^{\lambda+1})}} \right) - \log\left(\sqrt{\frac{1-\cos(\pi/2^{\lambda+1})}{1+\cos(\pi/2^{\lambda+1})}} \right) \right] + \\
&\quad \frac{1}{2^{\lambda-1}} \frac{1}{\sin(\pi/2^{\lambda+1})} \\
&\leq \frac{2}{\pi} \left[\log\left(\sqrt{\frac{\sin^2\left(\frac{1}{2}\left(\pi/2 - \pi/2^{\lambda+1}\right)\right)}{\cos^2\left(\frac{1}{2}\left(\pi/2 - \pi/2^{\lambda+1}\right)\right)}} \right) - \log\left(\sqrt{\frac{\sin^2(\pi/2^{\lambda+2})}{\cos^2(\pi/2^{\lambda+2})}} \right) \right] \\
&\quad + \frac{1}{2^{\lambda-1}} \frac{1}{\frac{2}{\pi}\frac{\pi}{2^{\lambda+1}}} \\
&\leq \frac{2}{\pi} \left[\log(\tan(\pi/4)) + \log(\cos(\pi/2^{\lambda+2})) - \log(\sin(\pi/2^{\lambda+2})) \right] + 2 \\
&\leq \frac{2}{\pi} \left[0 - \log\left(\frac{2}{\pi} \cdot \frac{\pi}{2^{\lambda+1}} \right) \right] + 2 \\
&\leq \frac{2}{\pi} \lambda \log(2) + 2.
\end{aligned}$$

□

The expression $\frac{1}{2}\lambda$ is a simple upper bound on the dominant term of the previous sum. Fürer's bound of the L_2 norm of the root of unity was 2^λ, which means that we have an exponentially better bound.

7.2.3 Precision

We need to be able to estimate the number of bits required to store the elements of $\mathcal{R}$ and to represent correctly the coefficients of the product C of the polynomials A and B in Algorithm 18. We use the same error analysis as in [26]. As in [26], in a practical implementation, we can assume that we use integer arithmetic and an appropriate scale with a power of two at each steps of the FFT. More details about this appropriate scale are given below.

We compute fast Fourier transforms over the ring $\mathcal{R} = \mathbb{R}[x]/(x^{2^\lambda} + 1)$. We represent the elements of $\mathcal{R}$ as polynomials with real coefficients with fixed precision. We want to determine the numerical precision needed for these coefficients. First, we start with integer coefficients of bit-length 2^λ. We apply two Half-DFTs, followed by a point-wise product, and then an inverse Half-DFT. At the end, the absolute error should be less than $1/2$ and the precision, denoted by S, follows.

Thus, we do the computations with S bits of precision, where S has to be determined. The coefficients are stored on S bits and we denote by V the number of bits before the binary point. The quantity $S - V$ is the number of bits after the binary point. During the algorithm, V varies.

For a given step of Fürer's algorithm, V is uniform for the set of elements that are involved in the following computations, except for the principal roots of unity. Thus, V can be thought as a static data at each step of Fürer's algorithm (each level of the FFT butterflies, point-wise product, first step of Half-DFT, last step of inverse of Half-DFT).

As it has been said, we suppose that we use integer arithmetic. For an element $Z \in \mathbb{Z}$, we denote its representation on a binary machine by $\overline{Z}$ and we let the error be

$$\epsilon(Z) = \overline{Z} - Z.$$

Elements $Z \in \mathbb{R}$ are represented as $\overline{Z} = m_Z 2^{-E_Z}$, where $m_Z \in \mathbb{Z}$ is such that $m_Z \leq 2^S$ and $E_Z \in \mathbb{N}$ is such that $S - E_Z$ is the number of bits before the binary point. We have the following arithmetic, for elements Z and Z' such that $E_Z = E_{Z'}$ for the addition:

$$\overline{Z} + \overline{Z'} = \left\lfloor \frac{m_Z + m_{Z'}}{2} \right\rfloor 2^{-E_Z+1}$$

and

$$\overline{Z} \cdot \overline{Z'} = \left\lfloor \frac{m_Z \cdot m_{Z'}}{2^S} \right\rfloor 2^{-E_Z - E_{Z'} + S}.$$

We consider that an addition or subtraction doubles the absolute value and increases V by one. However, multiplications by powers of x do not introduce errors: the maximum of the absolute values of the coefficients of $\overline{A \cdot x} - A \cdot x$ is the same as for $\overline{A} - A$ for any $A \in \mathcal{R}$.

The fact that the roots of unity in $\mathcal{R}$ have coefficients in $\mathbb{R}$ and that their L_1-norm is bounded by $n(\lambda)$ allows us to have a better precision than in [26]. Fürer bounds the infinity norm of its root of unity by one. We need the same result.

Lemma 7.8 (L_2-norm of roots of unity)**.** *The L_2-norm of ω is $\|\omega\|_2 = 1$.*

Proof. The proof is the same as in [26]:

- the value of $\omega(x)$ at all the principal $2^{\lambda+1}$-st roots of unity is a complex number of absolute value one;
- the polynomial $B \in \mathcal{R}$ whose coefficients are these values has L_2-norm equal to $\sqrt{2^\lambda}$;
- the coefficients of $\omega(x)$ are obtained via Half-DFT$^{-1}_{2^\lambda,\zeta}(B)$;
- $\sqrt{2^\lambda}$ Half-DFT$^{-1}_{2^\lambda,\zeta}$ is a unitary matrix: it preserves the L_2-norm.

□

Corollary 7.9. *We have $\|\omega^k\|_\infty \leq 1$ for any $k \geq 0$.*

Fürer shows in [26] that the precision S required to represent these coefficients is greater than

$$9 + 2^{\lambda+1} + 4\lambda + \log_2^{(2)}(2N) + 5\log_2 N \sim 5\log_2 N + 2^{\lambda+1}.$$

We do not prove this result because we obtain a better bound below: we obtain

$$1 + \log_2 15 + \log_2 N(2 + 3\frac{\log_2 n(\lambda)}{\log_2 2^\lambda}) + \lambda + 2^{\lambda+1} + \log_2(L+1) + \log_2^{(2)} N \sim (2 + o(1))\log_2 N + 2^{\lambda+1}.$$

We use exactly the same error analysis as Fürer. However, whereas Fürer used 2^λ as an upper bound on the L_1-norm of his principal roots of unity of $\mathbb{C}[x]/\left(x^{2^\lambda}+1\right)$, we use a more precise bound, namely $\tilde{n}(\lambda)$, defined as the least power of two greater than or equal to $n(\lambda)$. Furthermore, since he works over $\mathbb{C}$, he has an additional factor two. We need to reexpress the lemmas proved by Fürer in [26] with $\tilde{n}(\lambda)$, over $\mathbb{R}$. The bound that we obtain is a direct consequence of this injection.

As in [26], we say that, at some stage of an algorithm, we have a value bound v and an error bound e, if the coefficients r of elements of $\mathcal{R}$ involved in the computations of this stage satisfy

$$|r| \leq v \text{ and } |\epsilon(r)| \leq e.$$

Given ω^k, according to Corollary 7.9, 1 is a value bound on the coefficients of ω^k and 2^{-S} is an error bound on the coefficients of ω^k.

The bounds that are used are always powers of two. If we have a value bound v, we do all the computations with $V = \log_2 v$ bits before the binary point.

Lemma 7.10 (Lemma 6.1 in [26]). *Let $C \in \mathcal{R}$ be represented by $\overline{C}$, such that $\|C\|_\infty < v_C$ and $\|C - \overline{C}\|_\infty < e_C$. Let $k \in \{0, \ldots, 2N-1\}$. From $\overline{C}$ and $\overline{\omega^k}$ we compute a machine representation $\overline{D}$ of the product $D = C \cdot \omega^k$, such that v_D is a value bound and e_D is an error bound on the coefficients of D. Then,*

$$v_D = \tilde{n}(\lambda) v_C$$

is a value bound and

$$e_D = \tilde{n}(\lambda) e_C + 2^\lambda v_C 2^{-S+1}$$

is an error bound after the multiplication.

Proof. Let k be a non negative integer. We denote by $\omega_{k,i}$ the coefficients of ω^k. We showed that $\sum_i |\omega_{k,i}| \leq \tilde{n}(\lambda)$. We have

$$\overline{C} \cdot \overline{\omega^k} = \sum_{i=0}^{2^\lambda-1} \left(\sum_{j=0}^{i-1} \overline{c_j}\overline{\omega_{k,i-j}} - \sum_{j=i}^{2^\lambda-1} \overline{c_j}\overline{\omega_{k,2^\lambda+i-j}} \right) x^i.$$

We represent the $\overline{c_j}$'s by $c_j + \epsilon(c_j)$. We have the same decomposition for the coefficients of the root of unity $\omega_{k,j} + \epsilon(\omega_{k,j})$. We decompose $\overline{C} \cdot \overline{\omega}$ into three parts:

$$\begin{aligned}
\overline{C} \cdot \overline{\omega} = &\sum_{i=0}^{2^\lambda-1} \left(\sum_{j=0}^{i-1} c_j \omega_{k,i-j} - \sum_{j=i}^{2^\lambda-1} c_j \omega_{k,2^\lambda+i-j} \right) x^i \\
&+ \sum_{i=0}^{2^\lambda-1} \left(\sum_{j=0}^{i-1} \epsilon(c_j)\overline{\omega_{k,i-j}} - \sum_{j=i}^{2^\lambda-1} \epsilon(c_j)\overline{\omega_{k,2^\lambda+i-j}} \right) x^i \\
&+ \sum_{i=0}^{2^\lambda-1} \left(\sum_{j=0}^{i-1} c_j \epsilon(\omega_{k,i-j}) - \sum_{j=i}^{2^\lambda-1} c_j \epsilon(\omega_{k,2^\lambda+i-j}) \right) x^i.
\end{aligned}$$

The first sum is the theoretical polynomial to be approximated. We can bound it by $\tilde{n}(\lambda) v_C$, where $\tilde{n}(\lambda)$ is an upper bound on the L_1-norm of ω^k.

The coefficients of the second sum can be bounded in absolute value by $e_C\tilde{n}(\lambda)$:

$$|\overline{\omega_{k,i}}| \leq |\omega_{k,i}|$$

because of the truncature and

$$\left|\sum_{j=0}^{i-1}\epsilon(c_j)\overline{\omega_{k,i-j}} - \sum_{j=i}^{2^\lambda-1}\epsilon(c_j)\overline{\omega_{k,2^\lambda+i-j}}\right| \leq e_C\left(\sum_{j=0}^{2^\lambda-1}|\omega_{k,j}|\right).$$

We bound the coefficients of the third sum by $2^\lambda v_C 2^{-S}$ since the $\epsilon(\omega_{k,j})$'s satisfy $\epsilon(\omega_{k,j}) < 2^{-S}$. Moreover, we introduce a new error due to the rounding on each coefficient of the product. Thus, we add $v_D 2^{-S} \leq 2^\lambda v_C 2^{-S}$ to the previous sum. □

We do not prove the following lemmas, since the proofs are exactly the same as in [26], although we have to take into account the fact that we work with reals and that we obtained new bounds in Lemma 7.10.

Lemma 7.11 (Lemma 6.2 in [26]). *Let N be a power of two such that $N \geq 2$ and λ be a non negative integer. Let $L = \lceil(\log_2 N)/(\lambda+1)\rceil - 1$ be the number of levels with computationally intensive principal roots of unity in Fürer FFT. If the input A of an N-point FFT has a value bound v_A and an error bound e_A, then the output B has a value bound*

$$v_B = N\tilde{n}(\lambda)^L v_A$$

and an error bound

$$e_B = N\tilde{n}(\lambda)^L\left(e_A + v_A 2^{-S}\left(\log_2 N + 2L\frac{2^\lambda}{\tilde{n}(\lambda)}\right)\right)$$

We need to estimate the error produced by the point-wise product. Thus, given C and C' elements of $\mathcal{R}$ with v_C and e_C their value bound and their error bound, the problem is to estimate the error on the product.

Lemma 7.12 (Lemma 6.3 in [26]). *If v_C is a value bound and e_C, such that $v_C 2^{-S} \leq e_C$, is an error bound for C and C' before their multiplications, then $v_D = 2^\lambda v_C^2$ is a value bound and $e_D = 2^{\lambda+2} v_C e_C$ is an error bound on the coefficients of $D = C \cdot C'$.*

Lemma 7.13 (Lemma 6.4 in [26]). *For $\lambda = \lceil\log_2^{(2)} n\rceil$ and $N = 2n/2^{2\lambda-1}$, precision $S \geq \log_2 15 + \log_2 N\left(2 + 3\frac{\log_2\tilde{n}(\lambda)}{\log_2 2^\lambda}\right) + \lambda + 2^{\lambda+1} + \log_2(L+1) + \log_2^{(2)} N + 1$ is sufficient for the multiplication of integers of bit-length n.*

We summarize in Table 7.1 the evolution of the value and error bounds, at different stages of Fürer's algorithm. These stages are as follow:

1. Start;
2. After first level of Half-FFT;
3. After N-point FFT;

4. After point-wise products;
5. After Inverse-FFT;
6. After last level of Inverse-Half-FFT.

Stage	Value bound	Absolute error bound
1	2^{2^λ}	0
2	$\tilde{n}(\lambda)2^{2^\lambda}$	$2^\lambda 2^{2^\lambda} 2^{-S+1}$
3	$N\tilde{n}(\lambda)^{L+1}2^{2^\lambda}$	$N\tilde{n}(\lambda)^{L+1}2^{2^\lambda}2^{-S}\left(\log_2 N + 2(L+1)2^\lambda/\tilde{n}(\lambda)\right)$
4	$2^\lambda N^2\tilde{n}(\lambda)^{2(L+1)}2^{2^\lambda}$	$2^{\lambda+2}N^2\tilde{n}(\lambda)^{2(L+1)}2^{2^{\lambda+1}}2^{-S}\left(\log_2 N + 2(L+1)2^\lambda/\tilde{n}(\lambda)\right)$
5	$2^\lambda N^2\tilde{n}(\lambda)^{3L+2}2^{2^{\lambda+1}}$	$\tilde{n}(\lambda)^{3L+2}2^\lambda N^{2^{2^{\lambda+1}+1}}2^{-S}\left(5\log_2 N + (10L+8)2^\lambda/\tilde{n}(\lambda)\right)$
6	$2^\lambda N^2\tilde{n}(\lambda)^{3(L+1)}2^{2^{\lambda+1}}$	$\tilde{n}(\lambda)^{3(L+1)}2^\lambda N^{2^{2^{\lambda+1}+1}}2^{-S}\left(5\log_2 N + (10L+8)2^\lambda/\tilde{n}(\lambda)\right)$

Table 7.1: Value and error bounds.

In Fürer's article, the bound on the precision was

$$9 + 2^{\lambda+1} + 4\lambda + \log_2^{(2)}(2N) + 5\log_2 N \sim 5\log_2 N + 2^{\lambda+1}.$$

We proved that this bound can be actually taken as

$$1+\log_2 15+\log_2 N(2+3\frac{\log_2 n(\lambda)}{\log_2 2^\lambda})+\log_2 2^\lambda+2^{\lambda+1}+\log_2(L+1)+\log_2^{(2)} N \sim (2+o(1))\log_2 N+2^{\lambda+1}.$$

Thus, we have improved this bound by a factor almost equal to 2, when looking the coefficient of the dominant term $\log_2 N$.

7.2.4 Complexity analysis

Some non-trivial multiplications by elements of $\mathcal{R}$ are needed in Algorithm 18: $3E(N)$ multiplications in recursive calls, and $4N$ multiplications by scaling factors (because of the negacyclic convolution) and pointwise products. For these, we use Kronecker substitution: we encode elements of $\mathcal{R}$ as integers of bit length $O((\log_2 n)^2)$, and then call recursively FurerComplexMul. Other multiplications by roots of unity are cheap. Their number is $O(N \log N)$, and their cost is linear in the size of elements of $\mathcal{R}$, that is $O(2^\lambda \log n)$. Additionally, all implicit rearrangement costs of Algorithm 18 (see §6.4) are also within this same bound. We get the following equation for $\mathsf{M}(n)$:

$$\mathsf{M}(n) = N(3\lceil \log_{2^{\lambda+1}} N\rceil + 1)\cdot \mathsf{M}(O(\log n)^2) + O(N\log N \cdot 2^\lambda \log n). \tag{7.2}$$

Fürer proves by induction that this equation leads to $\mathsf{M}(n) \le n\log n(2^{c\log^*\sqrt[4]{n}} - d)$ for some constants $c, d > 0$, so that

$$\mathsf{M}(n) = n\cdot\log n\cdot 2^{O(\log^* n)},$$

where $\log^*$ is the iterated logarithm function defined in Section 6.6. This trick allows one to have an exponent $\log_2^*\sqrt[4]{n}$ decreasing at each recursive call to the integer multiplication algorithm.

We assume that there exists n_0' satisfying $n \geq n_0' \geq n_0 \geq 16$, where n_0 is the bit-length below which we apply a multiplication algorithm of worse asymptotic complexity. We suppose that Fürer's assertion is true for any $k \in \{2, \ldots, n\}$. Then, we obtain from

$$\mathsf{M}(n) \leq N(3\lceil \log_{2^{\lambda+1}} \rceil N + 1)\mathsf{M}(a(\log_2 N)^2) + b(N(\log_2 N)^3)$$

the relation

$$\mathsf{M}(n) \leq 4N\lceil \log_{2^{\lambda+1}} N\rceil a(\log_2 N)^2 \log_2(O(\log_2 N)^2)(2^{c(\log_2^* \sqrt[4]{O(\log_2 N)^2})} - d) + b(N(\log_2 N)^3),$$

by induction.

We choose n_0' such that $\sqrt[4]{(\log_2 N)^2} \leq \frac{1}{4}\log_2 n$. Then,

$$\begin{aligned}\mathsf{M}(n) &\leq 8\frac{n}{\log_2^2 n}\log_{2^{\lambda+1}} n a(\log_2^2 n) 2\log_2^{(2)}(2^{c(\log_2^*(\frac{1}{4}\log_2 n))} - d) + 2bn\log_2 n \\ &= 16an\log_2 n \frac{\log_2^{(2)} n}{\log_2 2^{\lambda+1}}(2^{c(\log_2^* \sqrt[4]{n}-1)} - d) + b2n\log_2 n.\end{aligned}$$

We notice that $\log_2^{(2)} n \leq \lambda + 1 \leq \log_2^{(2)} n + 2 \leq 2\log_2^{(2)} n$. By choosing c and d such that $16a \leq 2^c$ and $-8ad + 2b \leq d$ (and c sufficiently large to initialize the induction), we obtain the final result. Compared to original Fürer's algorithm, using a polynomial ring with real coefficients and an improved precision involves a different coefficient a. A modification of this coefficient a involves a modification of the constant c and hence the constant hidden in the term $O(\log^* n)$ of the final complexity.

7.3 Modular ring

A modular version of Fürer's algorithm has been proposed in [21]. Rather than considering the ring $\mathbb{C}[x]/(x^{2^\lambda} + 1)$, the authors proposed the ring $\mathbb{Z}[x]/(p^c, x^{2^\lambda} + 1)$ for a well-chosen prime p, and a constant c. It can be understood as a p-adic version of Fürer's algorithm, with a fixed precision. When they split the input integers a and b, they decompose them into N chunks, where N is no longer $N = O(\frac{n}{\log_2^2 n})$: they choose N such that $N^k = O\left(\frac{n}{(\log_2 n)^2}\right)$ for a constant k that will be detailed later.

Thus, they decompose a as $a = a_0 + ... + a_{N^k-1}2^{\left(\frac{n}{N^k}\right)(N^k-1)}$, from which they deduce a k-variate polynomial $A(X_1, ..., X_k) = \sum_{i=0}^{N^k-1} a_i X_1^{i_1} ... X_k^{i_k}$, where $(i_k ... i_1)$ is the decomposition of i in radix N (to recover a, they evaluate the X_i's at $2^{\left(\frac{n}{N^k}\right)\left(N^i-1\right)}$). According to this decomposition, they need to choose c such that $p^c > N^k 2^\lambda 2^{2^{\lambda+1}}$.

The ring $\mathbb{Z}/p^c\mathbb{Z}$ should contain a $2N$-th principal root of unity. For this purpose, they have to check that $2N \mid (p-1)$. Thus, they are looking for a prime p in the set $\{1 + 2iN \mid i \geq 0\}$. They reduce the search by using the following theorem.

Theorem 7.14 (Linnik's theorem [44]). *There exist constants l and L such that for d and m coprime integers, the smallest prime p satisfying $p \equiv d \bmod m$ is less than lm^L.*

Let now k be an integer greater than L. Since $N^k = o(n)$, $N^L = O(N^k) = o(n)$. Thus, according to Linnik's theorem, they can find $p \equiv 1 \bmod 2N$ with a cost no larger than to $O(N^L) = o(n)$.

It remains to find a $2N$-th principal root of unity in $\mathbb{Z}/p^c\mathbb{Z}$. For this purpose, it is necessary to find a generator of $\mathbb{F}_p$, which is a $(p-1)$-th root of unity in $\mathbb{F}_p$, with a deterministic complexity equal to $O(p^{\frac{1}{4}+\epsilon})$, according to [53], for any $\epsilon > 0$. One can deduce from it a $(p-1)$-th root of unity in $\mathbb{Z}/p^c\mathbb{Z}$ by using Hensel lifting.

Theorem 7.15 (Hensel Lifting). *If ζ is a $(p-1)$-th principal root of unity in $\mathbb{Z}/p^k\mathbb{Z}$, then there exists a $(p-1)$-th principal root of unity ζ' in $\mathbb{Z}/p^{k+1}\mathbb{Z}$ satisfying $\zeta' \equiv \zeta \bmod p^k$. This root of unity is given by $\zeta' = \zeta - \frac{f(\zeta)}{f'(\zeta)}$, where $f(x) = x^{p-1} - 1$*

Once the root of unity is obtained, it remains to apply the Lagrange interpolation used in Fürer's version to get a root of unity ω in $\mathbb{Z}[x]/(p^c, x^{2^\lambda} + 1)$ such that $\omega^{N/2^{\lambda+1}} = x$.

Thus, the authors use the same algorithm as Fürer, although they work with k-variate polynomials, where $k \geqslant 7$ and $c > 5(k+1)$, which implies to work in the ring $(\mathbb{Z}[x]/(p^c, x^{2^\lambda} + 1))(X_1, \ldots, X_{k-1}) = R(X_1, \ldots, X_{k-1})$, but does not change radically the complexity. They use several FFT: if the lengths are $N_1, N_2, \ldots, N_k$, they compute $N_2 \cdots N_k$ FFT's of length N_1, then $N_1 N_3 \cdots N_k$ FFT's of length N_2, etc. They finish with $N_1 \cdots N_{k-1}$ FFT's of length N_k. Those FFT's use the ring $\mathbb{Z}[x]/(p^c, x^{2^\lambda} + 1))$ and Fürer's strategy.

However, using k-variate polynomials instead of univariate polynomials introduce constants in the term $O(\log^* n)$ of the final complexity, due to the fact that the number of nonzero coefficients of the product of k-variate polynomials is larger than the number of coefficients of these k-variate polynomials (at most 2^k times). These constants have a practical impact and can be improved by considering that the computation of the prime p is a precomputation. Thus, rather than using k-variate polynomials, one can determine randomly the prime p with a cost $O(\log_2^3 N)$. and work in $(\mathbb{Z}[x]/(p^c, x^{2^\lambda} + 1))$, but the algorithm is not deterministic anymore.

7.4 Bluestein's chirp transform

Harvey, van der Hoeven and Lecerf in [32], and Harvey and van der Hoeven in [30] propose new algorithms achieving complexity bounds similar to the one that Fürer gets, and improve on the constant hidden in $O(\log_2^* n)$. Their improvement yields an asymptotic equation similar to the improvement in this part. The algorithms in [32, 30] rely on Bluestein's chirp transform [7].

Bluestein's chirp transform allows one to compute an L-point DFT as a negacyclic product of two polynomials $\tilde{A}$ and B. Let $A = \sum_{0 \leq i < L} a_i X^i$ be a polynomial of length L in $\mathcal{R}$ and $\omega \in \mathcal{R}$ be an L-th principal root of unity. We assume that L is even and that there exists $\nu \in \mathcal{R}$ such that $\nu^2 = \omega$. The authors have

$$A(\omega^k) = \sum_{0 \leq i < L} a_i \omega^{ik}.$$

Using the identity

$$ik = -\frac{1}{2}(i-k)^2 + \frac{i^2}{2} + \frac{k^2}{2},$$

they have

$$A(\omega^k) = \nu^{-k^2} \sum_{0 \leq i < L} \left(a_i \nu^{-i^2}\right) \nu^{(k-i)^2}.$$

Thus, denoting by $\tilde{A}$ and B the polynomials

$$\tilde{A} = \sum_{0 \leq i < L} a_i \nu^{-i^2} X^i \text{ and } B = \sum_{0 \leq i < L} \nu^{i^2} X^i,$$

they compute $C = \tilde{A} \cdot B \bmod X^L + 1$ and they obtain $A(\omega^k)$ multiplying the k-th coefficient of C by ν^{-k^2}.

Algorithm 19 DFT of polynomial with Bluestein's chirp transform

1: **Input:** $A = \sum_{i=0}^{L-1} a_i X^i \in \mathcal{R}[X]/X^L - 1$, ν a principal $2L$-th root of unity and $\omega = \nu^2$.
2: **Output:** $\hat{A} = \mathrm{DFT}_{L,\omega}(A) = A(1) + A(\omega)X + \cdots + A(\omega^{L-1})X^{L-1}$
3: **function** BluesteinTransform(A,L,ν)
4: Let $\tilde{A}$ be $\sum_{0 \le i < L} a_i \nu^{-i^2} X^i$
5: Let B be $\sum_{0 \le i < L} \nu^{i^2} X^i$
6: Let $C = \tilde{A} \cdot B \bmod X^L + 1$ computed with Kronecker substitution
7: **return** $A(1) + \cdots + A(\omega^{L-1})X^{L-1}$ by scaling the k-th coefficient of C by ν^{-k^2}.
8: **end function**

The idea developed in [32] implies to use, instead of a radix-2^λ FFT, another method described in Algorithm 20. On Line 11 of Algorithm 20, they use Bluestein's chirp transform instead of

Algorithm 20 FFT of order a power of two N using Bluestein's chirp transform

1: **Input:** $A = \sum_{i=0}^{N-1} a_i X^i \in \mathcal{R}[X]/X^N - 1$, ν a principal $2L$-th root of unity and $\omega = \nu^2$.
2: **Output:** $\hat{A} = \mathrm{DFT}_{N,\omega}(A) = A(1) + A(\omega)X + \cdots + A(\omega^{N-1})X^{N-1}$
3: **function** BluesteinFFT(A,N,λ,ν)
4: **if** $N < 2^\lambda$ **then**
5: Compute $\hat{A}$ with BluesteinTransform and a Kronecker substitution
6: **else**
7: $J \leftarrow 2^\lambda$
8: $K \leftarrow N/J$
9: **for** $0 \le k < K$ **do**
10: Let B_k be the polynomial given by the coefficients $(a_{jK+k})_j$
11: $\hat{B_k} \leftarrow$ BluesteinTransform(B_k, J, ν^K)
12: $\hat{B_k} \leftarrow \hat{B_k}(\omega^k X)$
13: **end for**
14: Let $(S_j(Y))_j \in \mathcal{R}[Y]^J$ be such that $\sum_{0 \le k < K} \hat{B_k}(X)Y^k = \sum_{0 \le j < J} S_j(Y)X^j$
15: **for** $0 \le j < J$ **do**
16: $\hat{S_j}(Y) \leftarrow$ BluesteinFFT(S_j, K, λ, ν^J) ▷ Recursive call
17: **end for**
18: **return** $\sum_{0 \le i < J} \hat{S}_i(X^J)X^i$
19: **end if**
20: **end function**

a radix-2 Cooley-Tukey FFT: they compute a negacyclic product of two polynomials with the Kronecker substitution. In this version, the ring $\mathcal{R}$ can be any ring containing an N-th principal root of unity ($\mathbb{C}$ for example) and λ can be taken larger than in Fürer's algorithm: rather than $\lambda \approx \log_2^{(2)} n$, they can choose it as $\lambda \approx (\log_2^{(2)} n)^2 + O(\log_2^{(2)} n)$. Rather than concentrating the expensive operations in the multiplications on Line 12, there is a trade-off between the complexity of the N multiplications in $\mathcal{R}$ on Line 12 and the $N/2^\lambda$ multiplications in $\mathcal{R}[x]/(x^{2^\lambda} + 1)$.

In order to multiply two n-bit integers a and b, they split these integers into $N/2$ pieces of

size s. Assuming that the precision required in $\mathbb{C}$ is p, the complexity equation is

$$\mathsf{M}(n) = O\left(N\frac{\log N}{\lambda}\mathsf{M}(O(p))\right) + O\left(\frac{N\log_2 N}{\lambda 2^\lambda}\mathsf{M}(O(2^\lambda p))\right) + O(n\log_2 n).$$

Using the radix-2 Cooley-Tukey algorithm, the second term in the equation is $O(N\log_2 N\mathsf{M}(O(p)))$. Taking $\lambda = \log_2 n\log_2^{(2)} n + O(\log_2 n)$, the first term is negligible and the second term is dominant. They obtain $O(n\log_2 n 2^{O(\log_2^* n)})$ as a solution of the recursive equation.

Chapter 8

Generalized Fermat primes

This chapter defines admissible generalized Fermat numbers. Our main use case will be when such numbers are prime, and we define a descending chain of such primes.

Definition 8.1. *A* generalized Fermat number *is an integer of the form $r^{2^\lambda} + 1$, where λ and r are two positive integers. We use the shorthand notation $P(r, \lambda)$ for such numbers.*

For notational ease, throughout this chapter, whenever we mention a generalized Fermat number p, we actually consider the pair (r, λ) rather than the number p alone. For this reason, it shall be understood without further mention that r and λ are implicit data that is unequivocally attached to p, which is underlined by the fact that we favor the expression "let $p = P(r, \lambda)$ be a generalized Fermat number".

8.1 Abundance of generalized Fermat primes

Asymptotically, the existence of generalized Fermat *primes* in integer intervals can be obtained via the Bateman-Horn conjecture [4]. For real numbers $A < B$ and an integer $\lambda \geq 1$, we let $\Delta(\lambda, A, B)$ denote the number of integers $r \in [A, B)$ such that $p = P(r, \lambda) = r^{2^\lambda} + 1$ is a generalized Fermat prime. The following lemma captures the asymptotic behaviour of Δ in specific intervals. However it will be of little use *per se* but to define some notations.

Lemma 8.2. *Fix an integer $\lambda \geq 1$. Let $\alpha > 1$ be a real number (possibly depending on λ). If the Bateman-Horn conjecture holds for the function $f(x) = x^{2^\lambda} + 1$, then*

$$\Delta(\lambda, R, \alpha R) \sim \frac{C_\lambda}{2^\lambda}(\mathrm{li}(\alpha R) - \mathrm{li}(R))$$

as $R \to \infty$, where we used the notations:

$$\mathrm{li}(x) = \int_2^x \frac{\mathrm{d}t}{\log t}, \quad C_\lambda = \frac{1}{2} \prod_{p \ prime} \frac{1 - \chi_\lambda(p)/p}{1 - 1/p}, \quad \chi_\lambda(p) = \begin{cases} 2^\lambda & \text{if } 2^{\lambda+1} \mid p - 1, \\ 0 & \text{otherwise.} \end{cases}$$

Proof. Bateman and Horn [4] define the constant C_λ as above, and conjecture that as R grows, we have

$$\Delta(\lambda, 1, R) \sim \frac{C_\lambda}{2^\lambda} \mathrm{li}(R) \sim \frac{C_\lambda}{2^\lambda} \cdot \frac{R}{\log R}.$$

Let $\epsilon > 0$. Assuming the Bateman-Horn conjecture holds, we have for R large enough

$$\left| \frac{\Delta(\lambda, R, \alpha R)}{\frac{C_\lambda}{2^\lambda}(\mathrm{li}(\alpha R) - \mathrm{li}(R))} - 1 \right| < \epsilon \cdot \frac{1 + \mathrm{li}(R)/\mathrm{li}(\alpha R)}{1 - \mathrm{li}(R)/\mathrm{li}(\alpha R)}.$$

Now since $\mathrm{li}(x) \sim x/\log x$ and $\alpha > 1$, the right-hand side above converges to a positive constant as $R \to \infty$. This proves the claim. □

We now go through several steps to provide heuristic arguments supporting the existence of sufficiently many generalized Fermat primes in our ranges of interest. Our attention first goes to the asymptotic estimate on the right-hand side in Lemma 8.2, and to how it evolves as $\lambda \to \infty$, for some specific choices of α and R. Table 8.1 indicates some experimental values for the constant C_λ (the same data has also been collected by [23]). While the observation of Table 8.1 would support the empirical claim that C_λ increases as λ increases, a proof of such a statement has eluded us. We prove Proposition 8.3 below, which is a much weaker statement. We then choose α and R so that the estimate of Lemma 8.2 can be shown to tend to infinity (Proposition 8.4).

Proposition 8.3. *Let C_λ be as in Lemma 8.2. We have $\frac{1}{\lambda} = O(C_\lambda)$.*

Proof. We prove that there exists an absolute constant $C > 0$ such that $C_\lambda \geq \frac{C}{\lambda}$ for any $\lambda \geq 1$, where C_λ is defined as in Lemma 8.2.

The idea is to rely on the proof of the main theorem of [49, §2], and to use the main result of [24] for arithmetic progressions with "powerful moduli", since we consider arithmetic progressions $(q \cdot k + r)_k$ where q is a power of two.

Let $\mathcal{P}(x)$ be the set of primes smaller than x, and extend the notation of Lemma 8.2 to define

$$C_\lambda(x) = \frac{1}{2} \prod_{p \in \mathcal{P}(x)} \frac{1 - \chi_\lambda(p)/p}{1 - 1/p}.$$

Throughout this proof, we use the shorthand notation $q = 2^{\lambda+1}$. Let $\pi(x, q, r)$ be the number of primes $\leq x$ congruent to r mod q. Let $G(x) = \pi(x, q, 1)$. We will use twice the Brun-Titchmarsh inequality, which says that

$$G(x) = \pi(x, q, 1) \leq 2(x/\phi(q))/\log(x/q) = 4x/(q \log(x/q)).$$

Let now $g(t) = G(t) - G(t-q)$. By construction, G and g are constant on intervals $[1+qi, 1+qi+q)$ for any integer $i \geq 0$, and $g(t)$ is equal to to 1 or 0 on that interval depending on whether $1 + qi$ is prime or not. Hence

$$g(1 + qi) = \frac{1}{q} \int_{1+qi}^{1+qi+q} g(t)\,\mathrm{d}t \quad \text{and} \quad G(1 + qi) = \frac{1}{q} \int_{0}^{1+qi+q} g(t)\,\mathrm{d}t.$$

Let $F(x) = -\log(1 - 2^\lambda/x) = -\log(1 - q/(2x))$, which is a decreasing, convex, and non-negative function defined for $x > q/2$. Furthermore, since $\lambda \geq 0$, for $x \geq 1 + q/2$ we have $F(x) \leq \log 2$. Our goal is to find an asymptotic lower bound for the (logarithm of the) numerator of $C_\lambda(x)$ for large x (we impose $x \geq 1 + 2q$ below). Equivalently, we seek an upper bound for $\mathcal{S}(x) = \sum_{i=1}^{N} F(1 + qi) g(1 + qi)$, where we set $N = \lfloor (x - 1)/q \rfloor$. Throughout the proof below,

implicit constants $O(1)$ are uniform on λ —possibly for x larger than some bound that depends on λ, but that is not an issue since $C_\lambda = \lim_{x\to\infty} C_\lambda(x)$.

$$\begin{aligned}
\mathcal{S}(x) &= \sum_{i=1}^{N} F(1+qi)g(1+qi) = \sum_{i=1}^{N} F(1+qi)\frac{1}{q}\int_{1+qi}^{1+qi+q} g(t)\,\mathrm{d}t \\
&\le \frac{1}{q}\sum_{i=1}^{N}\int_{1+qi}^{1+qi+q} F(t-q/2)g(t)\,\mathrm{d}t \quad \text{(because } F \text{ is convex)} \\
&\le \frac{1}{q}\int_{1+q}^{x+q} F(t-q/2)g(t)\,\mathrm{d}t + \underbrace{\frac{1}{q}\int_{x+q}^{1+qN+q} F(t-q/2)g(t)\,\mathrm{d}t}_{O(1)} \\
&\le F(x+q/2)G(x) - F(1+q/2)G(1+q) - \int_1^x F'(t+q/2)G(t)\,\mathrm{d}t + O(1).
\end{aligned}$$

Since $F(x+q/2) \le \frac{q}{2x}$ and $G(x) \le 4x/(q\log x/q))$, the first summand is bounded by $2/\log 2$ for $x \ge 2q$. Since $G(1+q) \le 1$ and $G(t) = 0$ for $t < 2$ we have:

$$\mathcal{S}(x) \le O(1) + \int_2^x \frac{q\pi(t,q,1)}{t(2t+q)}\,\mathrm{d}t \le O(1) + \int_2^x \frac{q\pi(t,q,1)}{2t^2}\,\mathrm{d}t.$$

Elliott [24] proved a theorem that relates $\pi(x,q,r)$ to its asymptotic estimate We state a very weak form of it, namely that there exists an absolute constant K such that for any $\lambda \ge 0$ and t such that

$$\min(t^{1/3}\exp(-(\log\log t)^3), t^{1/2}\exp(-8\log\log t)) \ge q,$$

we have

$$\left|\pi(t,q,1) - \frac{2t}{q\log t}\right| < \frac{Kt}{q(\log t)^2}. \tag{8.1}$$

The condition above on t can be simplified. There exists an absolute constant H such that for any $x > 1$

$$\min(x^{1/3}\exp(-(\log\log x)^3), x^{1/2}\exp(-8\log\log x)) \ge (Hx)^{1/4}.$$

Thus, for $t \ge q^4/H$, Equation (8.1) holds. We rewrite the upper bound on $\mathcal{S}(x)$:

$$\mathcal{S}(x) \le O(1) + \underbrace{\int_2^{q^4/H} \frac{q\pi(t,q,1)}{2t^2}\,\mathrm{d}t}_{I_0(\lambda)} + \underbrace{\int_{q^4/H}^{x} \frac{q\pi(t,q,1)}{2t^2}\,\mathrm{d}t}_{I_1(\lambda,x)}.$$

For $I_0(\lambda)$, by Brun-Titchmarsh we have

$$\begin{aligned}
I_0(\lambda) &\le \int_2^{q^4/H} \frac{2}{t\log(t/q)}\,\mathrm{d}t \le \int_2^{q^3/H} \frac{2}{u\log(u)}\,\mathrm{d}u \\
&\le 2\log\log(q^3/H) \le 2\log\lambda + O(1).
\end{aligned}$$

λ	C_λ	λ	C_λ	λ	C_λ	λ	C_λ	λ	C_λ
1	1.37	5	3.61	9	7.49	13	8.47	17	11.00
2	2.68	6	3.94	10	8.02	14	8.01	18	13.01
3	2.09	7	3.11	11	7.23	15	5.80	19	13.06
4	3.67	8	7.43	12	8.43	16	11.20	20	14.45

Table 8.1: Approximations of the infinite product C_λ, as defined by Lemma 8.2. The computation was done by enumerating all primes below 10^{11}, with resulting values rounded to nearest. Proposition 8.3 shows that $\frac{1}{\lambda} = O(C_\lambda)$.

We use Elliott's theorem to bound $I_1(\lambda, x)$ (using the notations of Equation (8.1)):

$$\begin{aligned}\left| I_1(\lambda,x) - \int_{q^4/H}^{x} \frac{1}{t\log t}\mathrm{d}t \right| &\leq \int_{q^4/H}^{x} \frac{K}{t(\log t)^2}\mathrm{d}t \\ &\leq -\frac{K}{\log x} + \frac{K}{\log(q^4/H)} = O(1) \\ \text{so that } I_1(\lambda,x) &\leq \log\log x - \log\log(q^4/H) + O(1) \\ &\leq \log\log x - \log\lambda + O(1).\end{aligned}$$

Combining the bounds on I_0 and I_1, we have obtained:

$$\mathcal{S}(x) \leq \log\log x + \log\lambda + O(1).$$

The lower bound on $C_\lambda(x)$ follows: indeed, we have

$$\begin{aligned}-\log C_\lambda(x) &= \sum_{p\in\mathcal{P}(x)} \log\left(1-\frac{1}{p}\right) + \mathcal{S}(x), \\ &\leq (-\gamma - \log\log x + o(1)) + (\log\log x + \log\lambda + O(1)), \\ \log C_\lambda(x) &\geq O(1) - \log\lambda.\end{aligned}$$

Hence $C_\lambda(x) \geq A/\lambda$ for some absolute constant A, and x large enough. It follows that $C_\lambda \geq A/\lambda$, as claimed. We notice that the multiplier affecting $\log\lambda$ above, and hence the exponent of λ in our lower bound, can be directly traced to the use of the Brun-Titchmarsh inequality in bounding $I_0(\lambda)$. □

Proposition 8.4. *We use the same notations as in Lemma 8.2. Let $a(\lambda)$ be a real-valued function such that $a(\lambda) \geq \kappa\lambda^{2+\epsilon}$ for two positive constants κ, ϵ. Then the asymptotic estimate of Lemma 8.2, when formulated for $R = 2^\lambda$ and $\alpha = a(\lambda)$ is such that, as $\lambda \to \infty$:*

$$\frac{C_\lambda}{2^\lambda}\left(\mathrm{li}(a(\lambda)\cdot 2^\lambda) - \mathrm{li}(2^\lambda)\right) \longrightarrow \infty.$$

Proof. A lower bound for $(\mathrm{li}(\alpha R) - \mathrm{li}(R))$ is $(\alpha - 1)R/\log(\alpha R)$. We have

$$\frac{C_\lambda}{2^\lambda}\left(\mathrm{li}(a(\lambda)\cdot 2^\lambda) - \mathrm{li}(2^\lambda)\right) \geq \frac{C_\lambda}{2\log 2}\cdot\frac{\kappa\lambda^{2+\epsilon}-1}{\lambda} \geq \lambda C_\lambda \cdot \frac{\kappa\lambda^\epsilon - 1/\lambda^2}{2\log 2}.$$

The claim follows, since $\frac{1}{\lambda} = O(C_\lambda)$ implies that $\lambda C_\lambda \lambda^\epsilon$ tends to ∞. □

Our heuristic claim is that for $\alpha = \lambda^{2.5}$ (which fulfills the conditions of Proposition 8.4), the estimate of Lemma 8.2 is accurate enough, as early as for $R = 2^\lambda$.

Hypothesis 8.5. *Let $\lambda \geq 2$ be an integer. For any real number R such that $2^\lambda \leq R \leq 2^{2\lambda}$, we have $\Delta(\lambda, R, \lambda^{2.5}R) \geq 1$. In other words, there exists a generalized Fermat prime $p = P(r,\lambda)$ such that $R \leq r < \lambda^{2.5}R$.*

Both the constant C_λ, as well as the accordance of the prime count $\Delta(\lambda, 1, B)$ with the asymptotic estimate given by the Bateman-Horn conjecture, have been studied by [23]. While the experiments of [23] do support the validity of the Bateman-Horn conjecture even for primes not very large, we provide independent experimental data to support Hypothesis 8.5. We computed numerically the value $\Delta(\lambda, 2^\lambda, \lambda^{2.5}2^\lambda)$, as well as the estimate given by Lemma 8.2. We chose to restrict the verification to $R = 2^\lambda$ because this is empirically the hardest case.

To obtain $\Delta(\lambda, 2^\lambda, \lambda^{2.5}2^\lambda)$, we used a simple primality proof algorithm based on Pocklington's theorem, in Las Vegas probabilistic time. The result of our experiments is given in Table 8.2. For each potential divisor q of $N - 1 = r^{2^\lambda}$ and each prime divisor s of r, we prove that the s-valuation of q is exactly 2^λ times the s-valuation of r, whence it follows eventually that N is prime. The probabilistic time comes from the fact that we use random picks to find a generator of the s-Sylow subgroup. Most computations were short, except for $\lambda = 12$ where we had to use some non-trivial amount of computing power. We reported the "total" number of candidate values for r in Table 2. It is easy to filter out r's that are roots of $x^{2^\lambda} + 1$ mod smallish primes congruent to 1 modulo $2^{\lambda+1}$. With a small expense in computation time, we can roughly halve the number of candidates. Yet we did not take this optimization into account when mentioning the number of candidates.

Hypothesis 8.5 is in fact stronger than what would be strictly necessary to reach the asymptotic complexity we claim in this thesis. Proposition 8.4 led us to choose α as a polynomial of degree at least two, and our particular choice $\alpha = \lambda^{2.5}$ has the advantage that the data in Table 8.2 has no corner cases for small values of λ (in particular for $\lambda = 3$). That factor could be replaced by any polynomial in λ, this would not affect the fact that the bounds used in Proposition 9.6 have a numerator in $O(\log_2 \lambda)$. We note, however, that using a polynomial with a more rapid growth would invalidate some inequalities for small values of λ, so that we would only be able to state our algorithm for larger values of λ.

Throughout the rest of the thesis, Hypothesis 8.5 is tacitly assumed.

8.2 Chains of generalized Fermat primes

Some generalized Fermat numbers, defined below, play a key role in this thesis.

Definition 8.6 (Admissible generalized Fermat number)**.** *A generalized Fermat number $p = P(r,\lambda)$ is called* admissible *whenever $\lambda \geq 4$ and r is such that $2^\lambda \leq r < 2^{2\lambda}\lambda^{2.5}$.*

λ	Candidates	Primes	Estimate	λ	Candidates	Primes	Estimate
1	0	0	0	7	8233	42	46
2	9	4	5	8	$2.3e4$	126	138
3	58	1	8	9	$6.2e4$	184	170
4	248	24	22	10	$1.6e5$	224	218
5	878	31	30	11	$4.1e5$	227	230
6	2789	57	45	12	$1.0e6$	≥ 307	312

Table 8.2: Number of generalized Fermat primes $r^{2^\lambda}+1$ with $r \in [R, \lambda^{2.5}R)$ with $R = 2^\lambda$ (only even r are counted as candidates), compared to the asymptotic estimate of Lemma 8.2. Hypothesis 8.5 asserts that the third column is never zero for $\lambda \geq 2$.

Definition 8.6 captures the primes whose existence is asserted by Hypothesis 8.5 (it is easy to observe that these are admissible when $\lambda \geq 4$), as well as generalized Fermat numbers that are subject to the same bounds.

The following proposition shows how from admissible generalized Fermat numbers (not necessarily prime), we can build smaller generalized Fermat primes. For large enough inputs, these smaller primes are in turn admissible, so that this construction can be used another time.

Proposition 8.7. *Let $\lambda \geq 4$, and let $p = P(r, \lambda) = r^{2^\lambda} + 1$ be an admissible generalized Fermat number. A smaller generalized Fermat prime denoted* smallerprime(p) *and an integer* batchsize(p) *are defined as follows.*

Let $\lambda' = \left\lceil \log_2^{(3)} p \right\rceil$. Let $\phi(k) = 2^{k+1} \log_2 r + \lambda - k$. There exists a power of two β such that the following conditions hold:

i. $0 \leq \log_2 \beta < \lambda'$,

ii. $\lambda' 2^{\lambda'} \leq \phi(\log_2 \beta) \leq 2\lambda' 2^{\lambda'}$,

iii. Given $R' = 2^{\phi(\log_2 \beta)/2^{\lambda'}}$, there exists an integer $r' \in [R', \lambda'^{2.5} R')$ such that $p' = P(r', \lambda') = r'^{2^{\lambda'}} + 1$ is a generalized Fermat prime.

Given β and p' as above, we let smallerprime$(p) = p'$ *and* batchsize$(p) = \beta$. *Furthermore, if $\lambda' \geq 4$, then p' is admissible too.*

In anticipation for the proof of Proposition 8.7, we prove the following bounds.

Lemma 8.8. *Let λ and λ' be as in Proposition 8.7. We have*

$$\log_2(\lambda + \log_2 \lambda) \leq \lambda' < 3 \log_2 \lambda - 1 < \lambda.$$

Proof. Since p is admissible, we have

$$2^{\lambda'} \geq \log_2(2^\lambda \log_2 r) \geq \lambda + \log_2^{(2)} r \geq \lambda + \log_2 \lambda.$$

λ	λ'
$3 \le \lambda \le 4$	3
$\lambda = 5$	3, 4
$6 \le \lambda \le 11$	4

λ	λ'
$\lambda = 12$	4, 5
$13 \le \lambda \le 26$	5
$\lambda = 27$	5, 6

λ	λ'
$28 \le \lambda \le 56$	6
$57 \le \lambda \le 58$	6, 7
$59 \le \lambda$	≥ 7

Table 8.3: Possible values for $\lambda' = \left\lceil \log_2^{(3)} p \right\rceil$ for $p = P(r, \lambda)$ an admissible generalized Fermat number, using the bounds $\log_2(\lambda + \log_2 \lambda) \le \lambda' \le 1 + \log_2^{(2)}(1 + 2^\lambda(2\lambda + 2.5 \log_2 \lambda))$.

In the other direction, the condition on p being admissible gives the following uniform bound on λ' (we first bound p by $2r^{2^\lambda}$):

$$\lambda' \le 1 + \log_2^{(2)}(1 + 2^\lambda(2\lambda + 2.5 \log_2 \lambda)).$$

An unilluminating calculation shows that this right hand side is indeed bounded by $3 \log_2 \lambda - 1$ for all $\lambda \ge 3$, and then by λ for all $\lambda \ge 4$. □

The lower bound given by Lemma 8.8 is most useful now, and gives in fact the correct order of magnitude for λ'. The upper bound is much coarser and will be used in §9.5. Possible values for λ' are given in Table 8.3. In particular, $\lambda \ge 4$ implies $\lambda' \ge 3$.

Proof of Proposition 8.7. The function ϕ is easily seen to satisfy $\phi(k) \le 2\phi(k-1)$ for any integer $k \le \lambda + 2$. As a consequence, the intervals $[\phi(k), 2\phi(k)]$, for k ranging from 0 to $\lambda' - 1$, form a covering of the interval $[\phi(0), \phi(\lambda')]$.

We prove $\phi(0) \le 2\lambda' 2^{\lambda'} \le \phi(\lambda')$, which will directly entail that $2\lambda' 2^{\lambda'}$ is within one of the above intervals that form a covering.

The bound $2\lambda' 2^{\lambda'} \le \phi(\lambda')$ is a consequence of $\lambda \ge \lambda'$:

$$\phi(\lambda') \ge 2^{\lambda'+1} \log_2 r \ge 2^{\lambda'+1} \lambda \ge 2^{\lambda'+1} \lambda'.$$

The proof that $2\lambda' 2^{\lambda'} \ge \phi(0)$ is based on calculus. Lower and upper bounds for $2\lambda' 2^{\lambda'}$ and $\phi(0)$ are

$$\begin{aligned} 2\lambda' 2^{\lambda'} &\ge 2(\lambda + \log_2 \lambda) \log_2(\lambda + \log_2 \lambda) \ge (\lambda + \log_2 \lambda) \log_2(36), \\ \phi(0) &\le \lambda + 2(2\lambda + 2.5 \log_2 \lambda) = 5(\lambda + \log_2 \lambda). \end{aligned}$$

We have proved that there exists an integer k such that $0 \le k < \lambda'$, and that $\phi(k) \le 2\lambda' 2^{\lambda'} \le 2\phi(k)$. Let $\beta = 2^k$, so that (i) holds. We have that

$$\lambda' \le \frac{\phi(\log_2 \beta)}{2^{\lambda'}} \le 2\lambda'.$$

This implies (ii). Finally, $R' = 2^{\frac{\phi(\log_2 \beta)}{2^{\lambda'}}}$ is such that $2^{\lambda'} \le R' \le 2^{2\lambda'}$. Hypothesis 8.5 then implies (iii), and concludes the proof. Admissibility of p' follows from Definition 8.6. □

The following technical lemma provides useful bounds for $p' = \text{smallerprime}(p)$.

Lemma 8.9. *Let $p = P(r, \lambda)$ be as in Proposition 8.7. Let $\beta = \text{batchsize}(p)$ and $p' = \text{smallerprime}(p)$. We have*

i. $1 \leq \frac{\log_2 p'}{2\beta \log_2 r} \leq \min(1 + \frac{4 \log_2 \lambda'}{\lambda' - 1}, \frac{7}{2})$. *In particular,* $\frac{\log_2 p'}{\beta \log_2 r} = 2 + o(1)$.

ii. $\lambda' + \log_2 \lambda' \leq \log_2^{(2)} p' \leq \lambda' + \log_2 \lambda' + 2$.

Proof. We follow the notations of Proposition 8.7. The lower bound in (i) is easy:

$$\log_2 p' \geq 2^{\lambda'} \log_2 R' \geq \phi(\log_2 \beta) \geq 2\beta \log_2 r.$$

The upper bound requires more work. On the one hand, Lemma 8.8 gives $2^{\lambda'} > \lambda$, whence

$$\begin{aligned} 2\beta \log_2 r + 2^{\lambda'} &\geq \phi(\log_2 \beta) \geq \lambda' 2^{\lambda'} \\ 2\beta \log_2 r &\geq (\lambda' - 1) 2^{\lambda'}. \end{aligned}$$

And on the other hand, we can bound $\log_2 p'$ as follows.

$$\begin{aligned} \log_2 p' &= \log_2((p' - 1) + 1) = \log_2(p' - 1) + \log_2(1 + 1/(p' - 1)) \\ &\leq 2^{\lambda'} \log_2 r' + 1 \leq 2^{\lambda'} \log_2 R' + 2^{\lambda'} \log_2(\lambda'^{2.5}) + 1 \\ &\leq \phi(\log_2 \beta) + 2^{\lambda'} \log_2(\lambda'^{2.5}) + 1 \end{aligned} \tag{8.2}$$

by the definition of R'. Using now Lemma 8.8 and $2^{\lambda'} \geq \lambda + \log_2 \lambda \geq \lambda + 2$ we have

$$\begin{aligned} \log_2 p' &\leq 2\beta \log_2 r + (2^{\lambda'} - 2) + 2^{\lambda'} \log_2(\lambda'^{2.5}) + 1 \\ &\leq 2\beta \log_2 r + 2^{\lambda'} \cdot \min(4 \log_2 \lambda', 5(\lambda' - 1)/2) \quad \text{since } \lambda' \geq 3. \end{aligned}$$

The upper bound on the last line is obtained by calculus. We have thus proved (i).

The lower bound in statement (ii) is trivial. The upper bound is derived from inequality (8.2) above. By (ii) in Proposition 8.7, we have $\frac{\phi(\log_2 \beta)}{2^{\lambda'}} \leq 2\lambda'$, whence

$$\begin{aligned} \log_2 p' &\leq 2\lambda' 2^{\lambda'} + 2^{\lambda'} \log_2(\lambda'^{2.5}) + 1. \\ \log_2^{(2)} p' &\leq \log_2 \left(1 + 2^{\lambda'}(2\lambda' + \log_2(\lambda'^{2.5}))\right) \\ &\leq \lambda' + \log_2 \lambda' + 2 \text{ since } \lambda' \geq 3. \end{aligned}$$

Again, this last upper bound is verified by calculus. □

Chapter 9

Integer multiplication algorithm using generalized Fermat primes

We now see how we can design an asymptotically fast integer multiplication algorithm that uses rings of integers modulo generalized Fermat primes.

Throughout this chapter, our preferred representation for elements of a ring $\mathcal{R}$ of integers modulo a generalized Fermat number $p = P(r, \lambda)$ is the representation *in radix* r. Namely, $a \in \mathcal{R}$ is represented as a 2^λ-uple $(a_0, \ldots, a_{2^\lambda-1})$ such that $a = \sum_{j<2^\lambda} a_j r^j$ and $0 \leq a_j < r$. This representation does not cover the case $a = -1$, and we need an *ad hoc* exceptional representation for this case (possible representation choices are plenty – one extra bit is enough). Conversions between binary representation and radix r representation can be done in linear time when r is a power of two, but we also need to deal with the general case. Recursive base conversion algorithms (see [10, §1.7.2]), do this in quasi-linear time $O(\lambda \mathsf{M}(\log p))$ (this holds both for ways, both *to* and *from* representation in radix r). Additions and subtractions in $\mathcal{R}$ using this representation are linear. This chapter is concerned with the complexity of multiplication in $\mathcal{R}$. We denote this cost by $\mathsf{M}_\mathcal{R}$.

9.1 Preliminaries: transforms

The following definition extends concepts defined in Proposition 8.7 and defines useful data for our algorithms.

Definition 9.1 (smallerring($\mathcal{R}$))**.** *Let $\lambda \geq 4$. Let $p = P(r, \lambda) = r^{2^\lambda} + 1$ be an admissible generalized Fermat number, and let $\mathcal{R} = \mathbb{Z}/p\mathbb{Z}$. Following Proposition 8.7 we let* smallerring($\mathcal{R}$) *be the triple $(\mathcal{R}', N', \omega')$ defined as follows:*

- $\mathcal{R}' = \mathbb{Z}/p'\mathbb{Z}$, *with* $p' = P(r', \lambda') =$ smallerprime(p).
- $N' = 2^\lambda /$ batchsize(p). *(N' is a power of two.)*
- ω' *is a primitive $2N'$-th root of unity in $\mathcal{R}'$.*

For the root ω' to be well defined above, we need the following property.

Lemma 9.2. *Using the notations above, $\mathcal{R}'$ has a primitive $2N'$-th root of unity.*

Proof. Notice first that $\lambda' \geq 3$ so that $p' \geq 2^{3 \cdot 2^3}$, and that $p' = r'^{2^{\lambda'}} + 1$ is prime, so that in particular r' is even. For $2N'$ to divide $p' - 1$, it suffices to check that $2N'$ divides $2^{2^{\lambda'}}$. We have

$$\log_2(2N') = \lambda + 1 - \log_2 \beta$$
$$2^{\lambda'} \geq \lambda + \log_2^{(2)} r,$$

so that it is sufficient to check that $\log_2^{(2)} r \geq 1$, which holds as soon as $\lambda \geq 2$. □

The algorithms described in the remainder of this chapter all assume that the sequences of rings and auxiliary data defined by Definition 9.1 are computed in advance, for all levels of the recursion. We assume that a tape of our Turing machine is devoted to that data, stored one level after another. The size of the data smallerring($\mathcal{R}'$) is clearly $O(\log p')$.

9.2 New algorithms

We now describe two new algorithms that are dependent on each other. Both aim at computing products of elements of $\mathcal{R}$.

- One algorithm that computes "transforms" of elements of $\mathcal{R}$, denoting the transform of an element $a \in \mathcal{R}$ by $\mathcal{T}_{\mathcal{R}}(a)$. Internally, this algorithm multiplies elements of $\mathcal{R}'$.
- One algorithm that multiplies elements of $\mathcal{R}$. This algorithm uses the transforms computed by the previous algorithm.

We begin with Algorithm 21 (TransformR), which computes transforms. We can state it thanks to Lemma 9.2.

Algorithm 21 Transform $\mathcal{T}_{\mathcal{R}}(a)$ of $a \in \mathcal{R} = \mathbb{Z}/p\mathbb{Z}$, with $p = r^{2^\lambda} + 1$ admissible (not necessarily prime), $\lambda \geq 4$. (Algorithm without precomputations.)

1: **function** TransformR(a)
2: **Input:** $a \in \mathcal{R}$, represented in radix r.
3: **Output:** $\mathcal{T}_{\mathcal{R}}(a)$, a vector of N' elements of $\mathcal{R}'$, represented in radix r'
4: Let $\beta =$ batchsize(p), and $(\mathcal{R}', N', \omega') =$ smallerring($\mathcal{R}$).
5: Let $\tilde{A}(X) \in \mathbb{Z}[X]$ with positive coefficients below r^β be such that $\tilde{A}(r^\beta) = a$;
6: Map $\tilde{A}$ to $A \in \mathcal{R}'[X]/(X^{N'} + 1)$.
7: Rewrite coefficients of A in radix r'.
8: **return** Half-DFT$_{N',\omega'}(A) =$ LargerRadixCooleyTukeyFFT$(A(\omega' X), N', \lambda', \omega'^2)$.
9: **end function**

Our complexity analysis will need to reason on the set of transforms of roots of unity that are used by Algorithm 21. We define it as follows:

Definition 9.3 ($\mathcal{W}(\mathcal{R})$, vector of precomputed transforms useful for $\mathcal{T}_{\mathcal{R}}$)**.** *Fix notations as in Definition 9.1. We let $\mathcal{W}(\mathcal{R})$ denote the vector defined as:*

$$\mathcal{W}(\mathcal{R}) = \{\mathcal{T}_{\mathcal{R}'}(\omega'^{2i+1}),\ i \in [\![0, \frac{N'}{2^{\lambda'+1}} - 1]\!]\}$$

where $\mathcal{T}_{\mathcal{R}'}$ is defined as in Algorithm 21 (albeit using $\mathcal{R}'$ as an input ring).

Complexity of Algorithm 21, with or without precomputations

We define the following costs. The analysis of $\mathsf{M}_{\mathcal{R}}$ and $\mathsf{M}'_{\mathcal{R}}$ will be done in §9.3.

- $\mathsf{M}_{\mathcal{R}}$: cost of multiplying $a \in \mathcal{R}$ by $b \in \mathcal{R}$, with no auxiliary inputs.
- $\mathsf{M}'_{\mathcal{R}}$: cost of the same computation, with $\mathcal{T}_{\mathcal{R}}(b)$ known.
- $\mathsf{T}_{\mathcal{R}}$: cost of computing $\mathcal{T}_{\mathcal{R}}$ with Algorithm 21.
- $\mathsf{T}'_{\mathcal{R}}$: cost of computing $\mathcal{T}_{\mathcal{R}}$ with Algorithm 21, aided with the auxiliary knowledge of $\mathcal{W}(\mathcal{R})$.
- $\mathsf{W}_{\mathcal{R}}$: cost of computing $\mathcal{W}(\mathcal{R})$.

We begin with $\mathcal{T}_{\mathcal{R}}$. Algorithm 21 uses base conversions on lines 5 and 7. Both operations perform $N' = 2^{\lambda}/\beta$ conversions, and the respective costs per conversion in each case are $O(\log\beta \cdot \mathsf{M}(\beta\log r))$ and $O(\lambda' \cdot \mathsf{M}(\beta\log r))$ (in these complexity estimates, $\mathsf{M}(n)$ can be taken as the complexity obtained for multiplying integers by the Schönhage-Strassen algorithm, for example). By Proposition 8.7 we have $\log\beta \leq \lambda'$, and by Lemma 8.9 we have $\log p' = \Theta(\beta\log r)$, so that the overall base conversion costs in Algorithm 21 can be expressed as $O(N'\lambda' \cdot \mathsf{M}(\log p'))$.

The computation of the Half-DFT on line 8 of Algorithm 21 involve $N'\log N'$ multiplication by roots of unity in $\mathcal{R}'$, of which only $(E(N') + N')$ exceed a linear cost (using the notation of Section 7.1). We have

$$\mathsf{T}_{\mathcal{R}} = (E(N') + N')\mathsf{M}_{\mathcal{R}'} + O\left(N'\log N'\log p' + N'\lambda' \cdot \mathsf{M}(\log p')\right).$$

We now turn to the analysis of $\mathsf{T}'_{\mathcal{R}}$. If the vector $\mathcal{W}(\mathcal{R}) = \{\mathcal{T}_{\mathcal{R}'}(\omega^{2i+1}),\ i \in [\![0, \frac{N'}{2^{\lambda'+1}} - 1]\!]\}$ is known, then the computation of $\mathcal{T}_{\mathcal{R}}$ can be done a bit faster: the $(E(N') + N')$ "expensive" multiplications by roots of unity in $\mathcal{R}'$ do not need to recompute the transforms of the roots. They may thus use a somewhat faster algorithm for multiplication in $\mathcal{R}'$. We defined above its cost as $\mathsf{M}'_{\mathcal{R}}$, and we have:

$$\mathsf{T}'_{\mathcal{R}} = (E(N') + N')\mathsf{M}'_{\mathcal{R}'} + O\left(N'\log N'\log p' + N'\lambda' \cdot \mathsf{M}(\log p')\right).$$

Finally, we give the cost $\mathsf{W}_{\mathcal{R}}$ of computing $\mathcal{W}(\mathcal{R})$. Here, we do *not* recursively use $\mathcal{W}(\mathcal{R}')$ to compute the different elements. We do however use the knowledge of the root of unity ω' (it belongs to the precomputed data smallerring($\mathcal{R}$)). We use the fact that we can deduce the transform of ω'^{k} from the binary decomposition of k and the computation of the transforms of the powers of the form ω'^{2^j}, as in the fast exponentation algorithms. To compute $\mathcal{W}(\mathcal{R})$, we first compute $\mathcal{T}_{\mathcal{R}'}(\omega')$ and $\mathcal{T}_{\mathcal{R}'}(\omega'^2)$, which cost $2\mathsf{T}_{\mathcal{R}'}$. Then we do successive point-wise multiplications by the vector $\mathcal{T}_{\mathcal{R}'}(\omega'^2)$ to obtain the transforms of the other roots. For each of the $N'/2^{\lambda'+1} - 1$ transforms to be inferred this way, we need N'' multiplications in $\mathcal{R}''$, where we temporarily set $(\mathcal{R}'', N'', \omega'') = \text{smallerring}(\mathcal{R}')$. Therefore we have

$$\mathsf{W}_{\mathcal{R}} \leq 2\mathsf{T}_{\mathcal{R}'} + (N'/2^{\lambda'+1} - 1)N''\mathsf{M}_{\mathcal{R}''} \leq N'\mathsf{T}_{\mathcal{R}'}.$$

Without further detail, we also claim that the inverse transform $\mathcal{T}_{\mathcal{R}}^{-1}$ can be computed with the same cost as $\mathcal{T}_{\mathcal{R}}$.

Algorithm 22 Multiplication in $\mathcal{R} = \mathbb{Z}/p\mathbb{Z}$, with $p = r^{2^\lambda} + 1$ admissible, $\lambda \geq 4$.
p is not necessarily prime.

We use the notations $\mathcal{T}_\mathcal{R}$, $\mathcal{W}_\mathcal{R}$ as in §9.2.
1: **function** MulR(a,$T_\mathcal{R}(b)$)
2: **Input:** $a \in \mathcal{R}$, represented in radix r; $\mathcal{T}_\mathcal{R}(b)$ for some $b \in \mathcal{R}$.
3: **Output:** $a \cdot b \mod p$, represented in radix r
4: Let $\beta = \text{batchsize}(p)$, and $(\mathcal{R}', N', \omega') = \text{smallerring}(\mathcal{R})$.
5: Compute $W = \mathcal{W}(\mathcal{R})$ using Algorithm TransformR.
6: Compute $\mathcal{T}_\mathcal{R}(a)$ using Algorithm TransformR and W as auxiliary data.
7: Compute $\gamma = \mathcal{T}_\mathcal{R}(a) * \mathcal{T}_\mathcal{R}(b)$ ▷ point-wise products of elements of $\mathcal{R}'$.
8: Compute $c = \mathcal{T}_\mathcal{R}^{-1}(\gamma)$ as follows:
9: $C \leftarrow \text{Half-IFT}_{N',\omega'}(\gamma) \in \mathcal{R}'[X]/(X^{N'} + 1)$ using W as auxiliary data.
10: Lift C to $\tilde{C} \in \mathbb{Z}[X]$ as follows:
11: **for** $i \in [\![0, N' - 1]\!]$ **do**
12: Lift coefficient of degree i to $[\![-(N' - 1 - i)r^{2\beta}, (i + 1)r^{2\beta}[\![$.
13: **end for**
14: Rewrite coefficients of $\tilde{C}$ as signed integers in radix r.
15: Compute $\tilde{C}(r^\beta) = c$. ▷ The result is defined modulo $(r^\beta)^{N'} + 1 = p$.
16: **return** c
17: **end function**

9.3 Multiplication modulo generalized Fermat numbers

Using Algorithm 21 (TransformR), we can now state Algorithm 22 (MulR). Its validity depends on the following lemma:

Lemma 9.4. *Let notations be as in Algorithm 22. Let $\tilde{A}, \tilde{B}$ be polynomials in $\mathbb{Z}[X]$ of degree less than N' and with positive coefficients below r^β such that, A and B being their respective images in $\mathcal{R}'[X]/(X^{N'} + 1)$, we have $\mathcal{T}_\mathcal{R}(a) = \text{Half-DFT}_{N',\omega'}(A)$ and $\mathcal{T}_\mathcal{R}(b) = \text{Half-DFT}_{N',\omega'}(B)$ on line 7 of Algorithm 22.*

i. Both $\tilde{A}$ and $\tilde{B}$ are uniquely defined from $\mathcal{T}_\mathcal{R}(a)$ and $\mathcal{T}_\mathcal{R}(b)$.

ii. The polynomial $\tilde{C}$ is equal to $\tilde{A} \cdot \tilde{B} \mod X^{N'} + 1$.

iii. c is equal to $ab \mod p$.

Proof. We prove (i) for $\mathcal{T}_\mathcal{R}(a)$, the same reasoning holds for $\mathcal{T}_\mathcal{R}(b)$. The polynomial $A \in \mathcal{R}'[X]/(X^{N'} + 1)$ is uniquely defined because Half-DFT is an isomorphism. Now since $\mathcal{T}_\mathcal{R}(a)$ is computed from an element a of $\mathcal{R}$, line 5 of Algorithm 21 has unambiguously computed a polynomial $\tilde{A}$, which meets the conditions. Since there is a unique lift of A to $\mathbb{Z}[X]$ that has degree less than N' and positive coefficients below p', this lift is then necessarily the same as $\tilde{A}$.

Statement (ii) holds modulo p' by construction, but we must make sure that the lift on lines 10-13 of Algorithm 22 computes the correct product over the integers. To do so, we compute a bound for the coefficients of the product $\tilde{A} \cdot \tilde{B} \mod X^{N'} + 1$. Both operands have at most N' coefficients. The coefficient of degree i of their product modulo $X^{N'} + 1$ lies within the interval $[\![-(N' - 1 - i)(r^\beta)^2, (i + 1)(r^\beta)^2[\![$ (actually with the lower endpoint open for $i < N' - 1$), which has width $N'(r^\beta)^2$. The base 2 logarithm of this latter value is $2\beta \log_2 r + \lambda - \log_2 \beta = \phi(\log_2 \beta)$, following the notation of Proposition 8.7. Now again following notations of Proposition 8.7, we

have $p' \geq {R'}^{2^{\lambda'}} \geq 2^{\phi(\log_2 \beta)} \geq N'(r^\beta)^2$. Thus, the coefficient c_i of degree i of $\tilde{A} \cdot \tilde{B} \bmod X^{N'} + 1$ is lifted to a unique signed representative modulo p' on line 10. This proves the claim.[2]

Statement (*iii*) follows: by (*ii*), we have that $\tilde{C} = \tilde{A} \cdot \tilde{B} \mod X^{N'} + 1$. By evaluating at r^β, we obtain the result $c = ab$ modulo $(r^\beta)^{N'} + 1 = p$. □

Complexity analysis of Algorithm 22

We first mention that the relative costs of multiplications and transforms, with or without precomputations, satisfy the following equations.

$$2\mathsf{T}'_{\mathcal{R}} \leq \mathsf{M}'_{\mathcal{R}} \leq \mathsf{M}_{\mathcal{R}} \leq \mathsf{M}'_{\mathcal{R}} + \mathsf{T}'_{\mathcal{R}} \leq \frac{3}{2}\mathsf{M}'_{\mathcal{R}} \quad \text{and} \quad \mathsf{T}_{\mathcal{R}} \leq \mathsf{M}_{\mathcal{R}}.$$

(To get $\mathsf{M}_{\mathcal{R}} \leq \mathsf{M}'_{\mathcal{R}} + \mathsf{T}'_{\mathcal{R}}$, it suffices to *first* compute $\mathcal{W}(\mathcal{R})$, and then $\mathcal{T}_{\mathcal{R}}(b)$.)

On line 10, Algorithm 22 converts between representation in radix r' and binary representation. On line 14 the conversion is between binary representation and representation in radix r. As with Algorithm 21, we can do this in time $O(N'\lambda' \cdot \mathsf{M}(\log p'))$. Point-wise products, on line 7, use a variation of Algorithm 22, where there is no auxiliary input, recursively (thus exploiting the fact that the coefficients of $\mathcal{T}_{\mathcal{R}}(a)$ and $\mathcal{T}_{\mathcal{R}}(b)$ are represented in radix r'). And last but not least, the most important aspect of the complexity of Algorithm 22 is that since we compute $\mathcal{W}(\mathcal{R})$, the transforms $\mathcal{T}_{\mathcal{R}}(a)$ and $\mathcal{T}_{\mathcal{R}}^{-1}(\gamma)$ can take advantage of it. We thus have:

$$\mathsf{M}'_{\mathcal{R}} = \mathsf{W}_{\mathcal{R}} + 2\mathsf{T}'_{\mathcal{R}} + N'\mathsf{M}_{\mathcal{R}'} + O(N'\lambda' \cdot \mathsf{M}(\log p')) + O(\log p).$$

We now use the various expressions obtained in §9.2 to rewrite this. We use the coarse bounds $\mathsf{T}_{\mathcal{R}'} \leq \mathsf{M}_{\mathcal{R}'} \leq \frac{3}{2}\mathsf{M}'_{\mathcal{R}'}$. We have

$$\begin{aligned}
\mathsf{M}'_{\mathcal{R}} &\leq N'\mathsf{M}_{\mathcal{R}'} + 2E(N')\mathsf{M}'_{\mathcal{R}'} + 2N'\mathsf{M}'_{\mathcal{R}'} + N'\mathsf{M}_{\mathcal{R}'} \\
&\qquad + O(N' \cdot \lambda' \cdot \mathsf{M}(\log p')) + O(N' \cdot \log N' \cdot \log p') + O(\log p) \\
&\leq 5N'\mathsf{M}'_{\mathcal{R}'} + 2E(N')\mathsf{M}'_{\mathcal{R}'} \\
&\qquad + O(N' \cdot \lambda' \cdot \mathsf{M}(\log p')) + O(N' \cdot \log N' \cdot \log p') + O(\log p) \\
&\leq 2N' \cdot (3 + \log_{2^{\lambda'+1}} N') \cdot \mathsf{M}'_{\mathcal{R}'} \\
&\qquad + O(N' \cdot \lambda' \cdot \mathsf{M}(\log p')) + O(N' \cdot \log N' \cdot \log p') + O(\log p)
\end{aligned}$$

where we used $E(N') \leq N' \log_{2^{\lambda'+1}} N'$ and $5/2 < 3$. Algorithm 22 also needs to move the head of tape of precomputed data by the size of the current data smallerring($\mathcal{R}$). The corresponding overhead $O(\log p')$ is easily subsumed within the lower-order terms above.

9.4 Multiplication in $\mathbb{Z}$ using multiplication in $\mathcal{R}$

We can build on Algorithm 22 to obtain an integer multiplication algorithm for n-bit integers a and b.

Note however that we avoid the following simple approach because it does not work complexity-wise: we do not multiply a and b by considering them as elements of $\mathbb{Z}/p\mathbb{Z}$ for p an admissible generalized Fermat number such that $p \geq 2^{2n}$. There are two reasons for that. First, doing so

[2] On lines 10-13 of Algorithm 22, intervals depend on the degree so that we can do without a needlessly coarse lower bound $2N'(r^\beta)^2 \leq p'$. It would be possible to adjust the definition of ϕ in Proposition 8.7, as well as the corresponding proofs, so that that coarser inequality holds.

for p an admissible generalized Fermat *prime* is out of question: unless we consider that p is given beforehand, computing it is likely to be more expensive than computing a product of bit length $\log_2 p$, and would therefore appear dominant, maybe prohibitive even for a precomputation. Fortunately, Algorithm 22 (MulR) does not require that p be prime, and therefore this difficulty can easily be circumvented. For example we may select λ such that $\lambda 2^\lambda \geq 2n$, and then set $p = P(2^\lambda, \lambda)$. The second issue is harder to deal with: in the ring $\mathcal{R}'$ used by Algorithm 22, we need to find $2N'$-th roots of unity, and for this we need a quadratic nonresidue in $\mathcal{R}'$ (which generates the 2-Sylow subgroup of $\mathcal{R}'$). Alas, if our first (non-prime) modulus p is such that $\log_2 p \geq 2n$, then in Proposition 8.7 we have $\lambda' = \lceil \log_2^{(3)} p \rceil \geq \log_2^{(2)} n$, so that the upper bound on $\log_2 p'$ that we obtain from Lemma 8.9 is at least as large as $\log_2 n \cdot \log_2^{(2)} n$. If we can use only deterministic exponential-time algorithms to search for a multiplicative generator for $\mathcal{R}'$, then the complexity of this search exceeds the overall complexity of integer multiplication.

Similar (but subtly different) issues were already encountered by Harvey, van der Hoeven and Lecerf. The workarounds proposed in [32, §8] also apply here.

- Either we assume the generalized Riemann hypothesis, in which case a quadratic nonresidue in $\mathcal{R}'$ can be found in polynomial time.
- Or we do the top-level multiplication with one round of Fürer's algorithm. Multiplication in the ring $\mathbb{C}[X]/X^{2^\lambda} + 1$ that is used by Algorithm 18 (FurerComplexMul) reduces to multiplication of integers of bit length $n_0 = O((\log n)^2)$, with n denoting the bit length of the integers a and b (see Equation (7.2)). These integers are then multiplied by Algorithm 22 (MulR), for a suitable modulus p_0 (not necessarily prime).

The latter strategy is given by Algorithm 23 (MulZ). Note that since we build upon Algorithm 18, we force the bit length n to be rounded up to a power of two.

Algorithm 23 Multiplication of integers in $\mathbb{Z}$

1: **Input:** a, b two positive n-bit integers, n being a power of two.
2: **Output:** $c = a \cdot b$
3: **function** MulZ(a,b)
4: Let n_0 be such that all internal multiplications in FurerComplexMul$(\cdot, \cdot, n)$ may be done by multiplying two n_0-bit integers. (As per the analysis of FurerComplexMul, we have $n_0 = O((\log_2 n)^2)$.)
5: Let λ_0 be the smallest integer such that $2n_0 \leq \lambda_0 2^{\lambda_0}$.
6: Let $p_0 = P(2^{\lambda_0}, \lambda_0) = 2^{\lambda_0 2^{\lambda_0}} + 1$.
7: **return** $c =$ FurerComplexMul(a, b, n), where all internal multiplications are done with Algorithm 22 (MulR), in the ring $\mathcal{R}_0 = \mathbb{Z}/p_0\mathbb{Z}$.
8: **end function**

It is easy to see that p_0 in Algorithm 23 is an admissible generalized Fermat number. As for the determination of *prime* moduli as well as the computation of primitive roots of unity of the desired order in the recursive multiplication levels of Algorithm 22, we have that $\log_2(\text{smallerprime}(p_0))$ is polynomial in $\log_2^{(2)} n$. This is small enough so that simple algorithms are fit for the task of testing smallerprime(p_0) for primality, as well as for finding primitive roots. Thus the complete chain of precomputed triples defined by smallerring in Definition 9.1 can be computed in advance and stored on an auxiliary tape of the Turing machine, as suggested in §9.1.

The complexity of computing n-bit products with Algorithm 23 (MulZ), which we denote by $\mathsf{M}_{\text{new}}(n)$, can be expressed as follows. The equation below is naturally very similar to Equa-

tion (7.2).

$$\mathsf{M}_{\text{new}}(n) = N(3\lceil \log_{2^{\lambda+1}} N\rceil + 1)\cdot \mathsf{M}_{\mathcal{R}_0} + O(N\log N\cdot 2^{\lambda}\log n);$$

9.5 Solution of the recursive complexity equations

9.5.1 Summary of the recursive complexity equations

In Algorithm 22 (MulR), multiplication in $\mathcal{R}$ uses $(\mathcal{R}', N', \omega') = \text{smallerring}(\mathcal{R})$. In turn, multiplication in $\mathcal{R}'$ may use $(\mathcal{R}'', N'', \omega'') = \text{smallerring}(\mathcal{R}')$ if recursion is used again. We define $(\mathcal{R}_i)_{i\geq 0}$ as well as $(N_i)_{i\geq 1}$ and $(\omega_i)_{i\geq 1}$ by:

$$\begin{aligned}\mathcal{R}_0 &= \text{as in Algorithm 23,}\\ (\mathcal{R}_{i+1}, N_{i+1}, \omega_{i+1}) &= \text{smallerring}(\mathcal{R}_i) \text{ for } i\geq 0.\end{aligned}$$

Likewise, we let p_i be such that $\mathcal{R}_i = \mathbb{Z}/p_i\mathbb{Z}$, for $i \geq 0$. Of course, since Definition 9.1 as well as Algorithms 21 and 22 are only valid for $\lambda_i \geq 4$, only a finite number of terms of the above sequences are defined for a given input size n. Part of the work towards determining our final complexity will be to determine this number of terms (the recursion depth). We briefly recall the key equations for the complexity analysis:

$$\begin{aligned}\mathsf{M}'_{\mathcal{R}_i} &\leq 2N_{i+1}\cdot(3+\log_{2^{\lambda_{i+1}+1}} N_{i+1})\cdot \mathsf{M}'_{\mathcal{R}_{i+1}}\\ &\quad + O(N_{i+1}\cdot\lambda_{i+1}\cdot \mathsf{M}(\log p_{i+1}))\\ &\quad + O(N_{i+1}\cdot\log N_{i+1}\cdot\log p_{i+1})\\ &\quad + O(\log p_i).\\ \mathsf{M}_{\mathcal{R}_i} &\leq \frac{3}{2}\mathsf{M}'_{\mathcal{R}_i}.\\ \mathsf{M}_{\text{new}}(n) &= N(3\lceil\log_{2^{\lambda_0+1}} N\rceil+1)\cdot\mathsf{M}_{\mathcal{R}_0} + O(N\log N\cdot 2^{\lambda_0}\log n).\end{aligned}$$

We first prove the following lemma that bounds the transform length N'.

Lemma 9.5. *Using the notations as above, we have*

$$N_{i+1} \leq \min\left(2\left(1+\frac{4\log_2\lambda_{i+1}}{\lambda_{i+1}-1}\right), 7\right)\cdot\frac{\log_2 p_i}{\log_2 p_{i+1}}.$$

Proof. Let $\beta = \text{batchsize}(p_i)$. We have $2^{\lambda_i}\log_2 r_i \leq \log_2 p_i$, therefore

$$N_{i+1} = \frac{2^{\lambda_i}}{\beta} \leq \frac{\log_2 p_i}{\beta\log_2 r_i} \leq 2\frac{\log_2 p_i}{\log_2 p_{i+1}}\frac{\log_2 p_{i+1}}{2\beta\log_2 r_i}.$$

Then (i) in Lemma 8.9 allows to conclude. □

The following result plays a central role in the asymptotic analysis.

Proposition 9.6. *We keep the above notations. Let $i \geq 0$ be such that p_i is admissible. Let*

$\epsilon_{0,i} = \frac{4\log_2 \lambda_{i+1}}{\lambda_{i+1}}$, $\epsilon_{1,i} = \frac{8\log_2 \lambda_i}{\lambda_i}$, *and* $\epsilon_{2,i} = \frac{2+\log_2 \lambda_{i+1}}{\lambda_{i+1}}$. *Let* $m_i = \frac{\mathsf{M}'_{\mathcal{R}_i}}{\log_2 p_i \cdot \log_2^{(2)} p_i}$. *We have*

$$m_i \leq 4 \cdot (1+\epsilon_{0,i}) \cdot (1+\epsilon_{1,i}) \cdot (1+\epsilon_{2,i}) \cdot m_{i+1} + O(1).$$

Proof. We first bound the second and third lines in the equation for $\mathsf{M}'_{\mathcal{R}_i}$, and compare them to $\log p_i \cdot \log^{(2)} p_i$. The third line uses Lemma 9.5. We have

$$N_{i+1}/\log_2 p_i \leq 7/\log_2 p_{i+1} = O(1)$$

which obviously also implies $(\log_2 N_{i+1})/(\log_2^{(2)} p_i) = O(1)$. Then

$$\frac{N_{i+1}\log_2 N_{i+1}\log_2 p_{i+1}}{\log_2 p_i \log_2^{(2)} p_i} \leq 7\frac{\log_2 N_{i+1}}{\log_2^{(2)} p_i} = O(1).$$

For the second line, it suffices to assume that $\mathsf{M}(\log p_{i+1})$ is bounded by the complexity of the Schönhage-Strassen algorithm. We have

$$\frac{N_{i+1}\lambda_{i+1}\mathsf{M}(\log p_{i+1})}{\log_2 p_i \log_2^{(2)} p_i} \leq 7\frac{\lambda_{i+1}\log_2^{(2)} p_{i+1}\log_2^{(3)} p_{i+1}}{\log_2^{(2)} p_i} = O(1).$$

In the expression above, we obtain the upper bound by bounding the numerator by a polynomial in λ_{i+1} (because p_{i+1} is admissible), while the denominator is exponential in λ_{i+1}.

The most important calculation for the analysis is the comparison of the first term of $\mathsf{M}'_{\mathcal{R}_i}$ with $\log p_i \cdot \log^{(2)} p_i$. Lemma 9.5 gives the bound $N_{i+1} \leq 2(1+\epsilon_{0,i})\frac{\log_2 p_i}{\log_2 p_{i+1}}$, and we also have the coarse bound $\log_2 N_{i+1} = \log_2(2^{\lambda_i}/\beta_i) \leq \lambda_i$. This implies

$$\begin{aligned} m_i &\leq 4(1+\epsilon_{0,i})\frac{\log_2 p_i}{\log_2 p_{i+1}} \cdot \left(3 + \frac{\lambda_i}{\lambda_{i+1}+1}\right) \cdot m_{i+1}\frac{\log_2 p_{i+1}\log_2^{(2)} p_{i+1}}{\log_2 p_i \log_2^{(2)} p_i} + O(1) \\ &\leq 4(1+\epsilon_{0,i})\left(3 + \frac{\lambda_i}{\lambda_{i+1}+1}\right)\frac{\log_2^{(2)} p_{i+1}}{\log_2^{(2)} p_i} m_{i+1} + O(1) \end{aligned}$$

By Lemma 8.8 we have $\left(3 + \frac{\lambda_i}{\lambda_{i+1}+1}\right) \leq \frac{\lambda_i + 9\log_2 \lambda_i}{\lambda_{i+1}+1} \leq \frac{\lambda_i + 9\log_2 \lambda_i}{\lambda_{i+1}}$. Furthermore by statement (ii) from Lemma 8.9 for $i > 0$, we have $\log_2^{(2)} p_i \geq \lambda_i + \log_2 \lambda_i$, so that

$$\begin{aligned} m_i &\leq 4 \cdot (1+\epsilon_{0,i}) \cdot \frac{\lambda_i + 9\log_2 \lambda_i}{\lambda_i + \log_2 \lambda_i} \cdot \frac{\log_2^{(2)} p_{i+1}}{\lambda_{i+1}} \cdot m_{i+1} + O(1) \\ &\leq 4 \cdot (1+\epsilon_{0,i}) \cdot (1+\epsilon_{1,i}) \cdot (1+\epsilon_{2,i}) \cdot m_{i+1} + O(1). \end{aligned}$$

where we used again Lemma 8.9 to bound $\log_2^{(2)} p_{i+1}$. This proves our claim. □

It is easy to convince oneself that the three quantities $\epsilon_{0,i}$, $\epsilon_{1,i}$, and $\epsilon_{2,i}$ all tend to zero as λ_i grows (that is, as we deal with larger and larger input numbers). The final asymptotic formula needs the following stronger result, however.

Lemma 9.7. *Let λ_0 be an arbitrarily large integer. Let K be the first integer such that $\lambda_K < 4$. We have $K = \log^* \lambda_0 + O(1)$. Furthermore, for $j = 0, 1, 2$:*

$$\prod_{i=0}^{K-1} (1+\epsilon_{j,i}) < \infty \quad \textit{(independently of } K\textit{)}$$

Proof. The expression of K follows from the inequality $\lambda' < 3\log\lambda - 1$ proved in Lemma 8.8. To see that, let $\Phi(\lambda) = 3\log_2\lambda - 1$, defined for $\lambda \geq 4$. Let $\Phi^*(x)$ be the function defined similarly to $\log^*$, by $\Phi^*(x) = 0$ for $x < 4$, and $\Phi^*(x) = 1 + \Phi^*(\Phi(x))$ otherwise. It is clear that $K \leq \Phi^*(\lambda_0)$. Now using the terminology defined in [32, §5], the function Φ^* is an iterator for the logarithmically slow function Φ. As such, it satisfies $\Phi^*(x) = \log^* x + O(1)$, which corresponds to our claim.

To bound the product, it suffices to bound $\sum_i |\epsilon_{j,i}|$. Let $f_0(x) = \frac{4\log_2 x}{x}$, $f_1(x) = \frac{8\log_2 x}{x}$, and $f_2(x) = \frac{2+\log_2 x}{x}$, so that $\epsilon_{0,i} = f_0(\lambda_{i+1})$, $\epsilon_{1,i} = f_1(\lambda_i)$, $\epsilon_{2,i} = f_2(\lambda_{i+1})$. The functions f_j are decreasing for $x \geq \exp(1)$. In particular, we have $\epsilon_{1,i} \leq f_1(\lambda_{i+1})$. Consider the sequence of real numbers defined by $u_0 = 3$, $u_1 = 4$, $u_2 = 6$, $u_3 = 27$, and $u_{k+1} = 2^{u_k/3}$ for $k \geq 3$. This sequences diverges to infinity. Independently of the starting value λ_0, we have

$$\begin{aligned}
\lambda_K &\geq 3 = u_0,\\
\lambda_{K-1} &\geq 4 = u_1,\\
\lambda_{K-2} &\geq 6 = u_2 \text{ by observing Table 8.3,}\\
\lambda_{K-3} &\geq 27 = u_3 \text{ again by Table 8.3,}\\
\lambda_{K-4} &\geq 2^{\lambda_{K-3}/3} \geq u_4 \text{ by Lemma 8.8,}\\
\lambda_{K-k} &\geq u_k \text{ for all } k \leq K.
\end{aligned}$$

This yields

$$\begin{aligned}
\sum_{i=0}^{K-1}\sum_{j=0}^{2}|\epsilon_{j,i}| = \sum_{k=0}^{K-1}\sum_{j=0}^{2}|\epsilon_{j,K-1-k}| &\leq \sum_{k=0}^{K-1}(f_0(\lambda_{K-k}) + f_1(\lambda_{K-1-k}) + f_2(\lambda_{K-k}))\\
&\leq \sum_{k=0}^{K-1}\sum_{j=0}^{2} f_j(\lambda_{K-k}) \leq \sum_{k=0}^{K-1}\sum_{j=0}^{2} f_j(u_k) \leq \sum_{k=0}^{\infty}\sum_{j=0}^{2} f_j(u_k).
\end{aligned}$$

The latter sum converges to an absolute constant. □

9.5.2 Complexity of integer multiplication

Theorem 9.8. *The complexity $\mathsf{M}_{new}(n)$ of the algorithm presented in §9.4 to multiply n-bit integers is*

$$\mathsf{M}_{new}(n) = O(n \cdot \log n \cdot 4^{\log^* n}).$$

Proof. This theorem is a consequence of the results obtained thus far. Recall that in Algorithm 18 (FurerComplexMul), we have $N = O(n/(\log_2 n)^2)$ and $2^\lambda = O(\log_2 n)$. The input size of Algorithm 23 (MulZ) is $n_0 = \Theta((\log_2 n)^2)$ bits. We have

$$\begin{aligned}
O(N\log N \cdot 2^\lambda \log n) &= O((n/(\log_2 n)^2)(\log_2 n)^3) = O(n\log n).\\
N(3\lceil\log_{2^{\lambda+1}} N\rceil + 1)\cdot \mathsf{M}_{\mathcal{R}_0} &\leq O\left(\frac{n}{(\log_2 n)^2}\right)\cdot O\left(\frac{\log_2 n}{\log_2^{(2)} n}\right)\mathsf{M}_{\mathcal{R}_0}\\
&\leq O(n\log n)\cdot\frac{\mathsf{M}_{\mathcal{R}_0}}{n_0(\log_2 n_0)}.
\end{aligned}$$

We thus have, using $\log p_0 = \Theta(n_0)$:

$$\frac{\mathsf{M}_{\text{new}}(n)}{n\log n} = O(1) + O\left(\frac{\mathsf{M}_{\mathcal{R}_0}}{\log_2 p_0 \log_2^{(2)} p_0}\right) = O(1) + m_0$$

using the notation of Proposition 9.6. Let now A be a constant bounding the $O(1)$ in Proposition 9.6, let $C = A/3$, and let $e(i) = \prod_{0\leq j\leq 2}(1+\epsilon_{j,i})$. We have $4e(i) - 1 \geq 3$ so that $A \leq (4e(i)-1)C$. Proposition 9.6 implies

$$\begin{aligned} m_i &\leq 4e(i)m_{i+1} + (4e(i)-1)C \\ (m_i + C) &\leq 4e(i)(m_{i+1}+C), \end{aligned}$$

so that we get $m_0 = O(4^{\log^* n})$ by Lemma 9.7. Finally, this gives

$$\frac{\mathsf{M}_{\text{new}}(n)}{n\log n} = O(4^{\log^* n}).$$

□

Chapter 10

Practical considerations

While our algorithm is mostly of theoretical interest, several points are worth mentioning, as an answer to the natural question of its practicality. Despite the title of this chapter, we are not reporting data on an actual implementation of our algorithm, but rather measurements that shed some light on its practical value.

10.1 Adaptation of the asymptotically fast algorithm to practical sizes

At the beginning of §9.4, we briefly alluded to a way to multiply two n-bit integers: pick a generalized Fermat number (not a priori prime) of the form $p_0 = P(2^{\lambda_0}, \lambda_0)$, for λ_0 such that $\lambda_0 2^{\lambda_0} \geq 2n$. Then use Algorithm 22 (MulR). This does not work asymptotically because computing roots of unity modulo $p_1 = \text{smallerprime}(p_0)$ cannot be done deterministically with good complexity. However, in practice, for say $n \leq 2^{64}$, Table 8.3 and Lemma 8.9 imply that p_1 would then be at most a 2048-bit prime, for which both the primality proof and the computation of roots can reasonably be assumed to be done once and for all. Therefore, the stumbling blocks that are relevant for the asymptotic analysis need not be considered as such for a practical implementation. This implies in particular that resorting to Algorithm 18 (FurerComplexMul), as we do in Algorithm 23 (MulZ) for asymptotic reasons, is not needed in practice.

Going further in this direction, we may in fact consider as a practical instance of our algorithm the more general procedure that follows Algorithm 14 with $\mathcal{R}_1 = \mathbb{Z}/p_1\mathbb{Z}$ as a base ring, where p_1 is a generalized Fermat prime. The aforementioned strategy can be regarded as Algorithm 14 with $\eta = 2^{\lambda_0}$, $N = 2^{\lambda_0}$ (still with $\lambda_0 2^{\lambda_0} \geq 2n$), at least in the case where $\beta = \text{batchsize}(p_0) = 1$.

Another alteration that we wish to make in practice is that our top-level multiplication need not use a negacyclic transform: whether we compute a product modulo $2^{2n}+1$ or $2^{2n}-1$ makes no difference when both inputs are less than or equal to $2^n - 1$, which is the case on the top-level. On the other hand, a "full" DFT of length N instead of a Half-DFT saves $3N$ multiplications in the base ring, which is not entirely negligible. Since a product modulo $2^{2n}+1$ is done with a Half-DFT, we prefer to compute the product modulo $2^{2n}-1$ on the top-level.

Finally, we note that for all sizes of practical interest, arithmetic in $\mathcal{R}_1 = \mathbb{Z}/p_1\mathbb{Z}$ will not be done with a Fourier-transform-based algorithm, because p_1 is only of very moderate size.

Taking into account all the remarks above, the only link that remains between the practical procedure that we envision and the algorithms (in particular, Algorithm 22 (MulR)) described in the second part of this thesis is that p_1 is a generalized Fermat prime. The developments in this

part of the thesis show that computing with generalized Fermat prime is asymptotically feasible, and yields a good complexity.

10.2 Parameter choices for various input sizes

In this section, we consider various input sizes n, and various candidate generalized Fermat primes $p_1 = {r_1}^{2^{\lambda_1}} + 1$. For combinations of these, we find values η and N (both powers of two) such that Algorithm 14 works. Let us briefly recall its structure: we write both n-bit integer inputs a and b in radix η, or equivalently as the evaluations at η of two polynomials of degree less than $N/2$. We multiply these polynomials in $\mathcal{R}_1[x]$. For this, we compute full N-point DFTs, then a point-wise product, and finally an inverse DFT. Arithmetic in $\mathcal{R}_1$, as in §9, uses representation in radix r_1. For this procedure to correctly compute the integer product $a \cdot b$, the following conditions must hold:

$$\begin{cases} N\eta^2 \leq p_1 & \text{(no overflow occurs in } \mathcal{R}_1\text{)}, \\ 2^{2n} \leq \eta^N - 1 & \text{(correct computation of the product of two } n\text{-bit integers)}, \\ N \mid p_1 - 1 & \text{(a principal } N\text{-th root of unity exists in } \mathcal{R}_1\text{)}. \end{cases}$$

In particular, N is the smallest power of two above $2n/\log_2 \eta$. When choosing η and N subject to the conditions above, we have some freedom. Ultimately, we wish to minimize the number of multiplications in $\mathcal{R}_1$, because we expect those to form the largest part of the computation time. More precisely, we wish to minimize the overall cost $(3E(N)+N)\mathsf{M}_{\mathcal{R}_1}$ of *expensive* multiplications as introduced in §7 ($\mathsf{M}_{\mathcal{R}_1}$ denotes the cost of one expensive multiplication in $\mathcal{R}_1$; we add N because of the point-wise products, and not $4N$ since here we do not use a half-DFT). Using the expression of $E(N)$, a rough estimate of the quantity to minimize is $\frac{n}{\log_2 \eta} \frac{\log_2 n}{\lambda_1+1} \mathsf{M}_{\mathcal{R}_1}$, therefore for n constant we try to minimize

$$Q \approx \frac{\mathsf{M}_{\mathcal{R}_1}}{\lambda_1 \log_2 \eta}.$$

Thus, there is a trade-off to determine: when η grows, larger primes p_1 have to be used: $\mathsf{M}_{\mathcal{R}_1}$ increases, while $\frac{1}{\log_2 \eta}$ decreases. Since the cost $\mathsf{M}_{\mathcal{R}_1}$ is given by the bit length of the prime p_1, the η that we choose should be the largest η for which p_1 is valid (as per the first of the three conditions above). The number of expensive multiplications for various input sizes n and primes p_1 is reported in Table 10.1. We added in Table 10.1 the additional constraint that $\log_2 \eta$ be a multiple of the machine word size, to the extent possible (since N must be a power of two anyway, this constraint has no impact).

10.3 Cost of multiplications in the underlying ring

We now turn to the two last columns of Table 10.1. Our goal is to obtain a coarse lower bound on the time we expect our algorithm to take. Arithmetic in $\mathcal{R}_1$, and in particular multiplication, is our main focus. Elements of $\mathcal{R}_1$ are represented in radix r_1. We avoid the conversion between radix r_1 and binary representation by using Kronecker substitution: an element of $\mathcal{R}_1$, represented as a 2^{λ_1}-uple of integers in $[0, r_1)$, is transformed into an integer of bit length

$$k = (2\lceil \log_2 r_1 \rceil + \lambda_1) \cdot 2^{\lambda_1}.$$

Multiplication in $\mathcal{R}_1$ is then done by multiplying these integers modulo $2^k + 1$ (we deal with signs in the same way as in Algorithm 22 (MulR)). We ignore the cost of converting this product

bit length of both operands: 2^{30}					
p_1	η	N	$3E(N)+N$	bit length of K.S.	lower bound
$\mathbf{984^{16}+1}$	$\mathbf{2^{64}}$	$\mathbf{2^{25}}$	$\mathbf{2^{25}\cdot(16=3\cdot 5+1)}$	$\mathbf{(2\cdot 10+4)\cdot 16=384}$	$\mathbf{2.68\cdot 10^{1}}$ s
$1984^{16}+1$	2^{64}	2^{25}	$2^{25}\cdot(16=3\cdot 5+1)$	$(2\cdot 11+4)\cdot 16=416$	$3.44\cdot 10^{1}$ s
$4016^{16}+1$	2^{64}	2^{25}	$2^{25}\cdot(16=3\cdot 5+1)$	$(2\cdot 12+4)\cdot 16=448$	$3.44\cdot 10^{1}$ s
$448^{32}+1$	2^{128}	2^{24}	$2^{24}\cdot(13=3\cdot 4+1)$	$(2\cdot 9+5)\cdot 32=736$	$3.82\cdot 10^{1}$ s
$884^{32}+1$	2^{128}	2^{24}	$2^{24}\cdot(13=3\cdot 4+1)$	$(2\cdot 10+5)\cdot 32=800$	$4.45\cdot 10^{1}$ s
$412^{64}+1$	2^{256}	2^{23}	$2^{23}\cdot(13=3\cdot 4+1)$	$(2\cdot 9+6)\cdot 64=1536$	$7.57\cdot 10^{1}$ s
$506^{128}+1$	2^{512}	2^{22}	$2^{22}\cdot(10=3\cdot 3+1)$	$(2\cdot 9+7)\cdot 128=3200$	$9.94\cdot 10^{1}$ s
bit length of both operands: 2^{40}					
p_1	η	N	$3E(N)+N$	bit length of K.S.	lower bound
$984^{16}+1$	2^{32}	2^{36}	$2^{36}\cdot(25=3\cdot 8+1)$	$(2\cdot 10+4)\cdot 16=384$	$8.57\cdot 10^{4}$ s
$\mathbf{1984^{16}+1}$	$\mathbf{2^{64}}$	$\mathbf{2^{35}}$	$\mathbf{2^{35}\cdot(22=3\cdot 7+1)}$	$\mathbf{(2\cdot 11+4)\cdot 16=416}$	$\mathbf{4.84\cdot 10^{4}}$ s
$4016^{16}+1$	2^{64}	2^{35}	$2^{35}\cdot(22=3\cdot 7+1)$	$(2\cdot 12+4)\cdot 16=448$	$4.84\cdot 10^{4}$ s
$448^{32}+1$	2^{64}	2^{35}	$2^{35}\cdot(19=3\cdot 6+1)$	$(2\cdot 9+5)\cdot 32=736$	$1.14\cdot 10^{5}$ s
$884^{32}+1$	2^{128}	2^{34}	$2^{34}\cdot(19=3\cdot 6+1)$	$(2\cdot 10+5)\cdot 32=800$	$6.66\cdot 10^{4}$ s
$412^{64}+1$	2^{256}	2^{33}	$2^{33}\cdot(16=3\cdot 5+1)$	$(2\cdot 9+6)\cdot 64=1536$	$9.54\cdot 10^{4}$ s
$506^{128}+1$	2^{512}	2^{32}	$2^{32}\cdot(13=3\cdot 4+1)$	$(2\cdot 9+7)\cdot 128=3200$	$1.32\cdot 10^{5}$ s
bit length of both operands: 2^{50}					
p_1	η	N	$3E(N)+N$	bit length of K.S.	lower bound
$984^{16}+1$	2^{32}	2^{46}	$2^{46}\cdot(31=3\cdot 10+1)$	$(2\cdot 10+4)\cdot 16=384$	$1.09\cdot 10^{8}$ s
$\mathbf{1984^{16}+1}$	$\mathbf{2^{64}}$	$\mathbf{2^{45}}$	$\mathbf{2^{45}\cdot(28=3\cdot 9+1)}$	$\mathbf{(2\cdot 11+4)\cdot 16=416}$	$\mathbf{6.31\cdot 10^{7}}$ s
$4016^{16}+1$	2^{64}	2^{45}	$2^{45}\cdot(28=3\cdot 9+1)$	$(2\cdot 12+4)\cdot 16=448$	$6.31\cdot 10^{7}$ s
$448^{32}+1$	2^{64}	2^{45}	$2^{45}\cdot(25=3\cdot 8+1)$	$(2\cdot 9+5)\cdot 32=736$	$1.54\cdot 10^{8}$ s
$884^{32}+1$	2^{128}	2^{44}	$2^{44}\cdot(25=3\cdot 8+1)$	$(2\cdot 10+5)\cdot 32=800$	$8.97\cdot 10^{7}$ s
$412^{64}+1$	2^{256}	2^{43}	$2^{43}\cdot(22=3\cdot 7+1)$	$(2\cdot 9+6)\cdot 64=1536$	$1.34\cdot 10^{8}$ s
$506^{128}+1$	2^{512}	2^{42}	$2^{42}\cdot(19=3\cdot 6+1)$	$(2\cdot 9+7)\cdot 128=3200$	$1.98\cdot 10^{8}$ s

Table 10.1: Estimated lower bound for the total cost of expensive multiplications in our algorithm depending on the prime used. Timings are based on the multiplication count and the measured time for the Kronecker-Schönhage bit length in the fifth column, on an Intel Xeon E7-4850v3 CPU (2.20GHz).

back to radix r_1. This is likely to be at the very least a significant source of inaccuracy in our lower bounds.

The fifth column of Table 10.1 reports the bit length k introduced above, for the various generalized Fermat primes chosen. Based on this bit length, we determined experimentally on a target machine (Intel Xeon E7-4850v3 CPU clocked at 2.20GHz) the time taken by the function `mpn_mul` in the GMP library [28], thereby giving a lower bound on the multiplication time in $\mathcal{R}_1$. We multiplied this lower bound by the number of expensive multiplication reported on the fourth column of Table 10.1, from which we deduced a lower bound on the multiplication time for n-bit integers using our algorithm.

The determination of the bit length above led us to restrict the set of generalized Fermat primes to consider: two such primes that lead to identical bit length lead to an identical time for internal multiplications. Therefore, we favor the largest generalized Fermat prime for each value of the Kronecker-Schönhage bit length k above. In our choice, we also favored primes such that r_1 has largest 2-valuation among the candidate values (e.g. both $984^{16}+1$ and $1018^{16}+1$ are primes, but we experimented with the former because the latter only allows a maximum transform length of 2^{16}).

We deduce from Table 10.1 that for realistic sizes, choosing the prime p_1 appropriately can lead to a speed-up of the order of 2 to 4, with all the necessary words of caution: as mentioned above, we deliberately omitted some conversion costs that are unlikely to be negligible in practice, and also our measurements are done with all operands in cache memory, which is quite probably optimistic.

10.4 Comparison with Schönhage-Strassen

Let us compare approximatively the cost of Schönhage-Strassen's algorithm to our algorithm. We can do two things. At least up to some size, we can run GMP's implementation of the Schönhage-Strassen algorithm, and obtain an actual computation time. Or we can do as we did in Table 10.1: count the number of small multiplications involved, and measure their cost. We did both, because the latter approach, which inherently gives a lower bound, is a fairer comparison given that a lower bound is all that we have in Table 10.1.

Roughly speaking, a Schönhage-Strassen multiplication of two 2^n-bit integers involves $2^{\lfloor (n+1)/2 \rfloor}$ multiplications of $2^{1+\lceil (n+1)/2 \rceil}$-bit modular integers. In truth, a well-tuned implementation of the Schönhage-Strassen algorithm uses all sorts of optimizations that are well outside the scope of this thesis (see e.g. [27]), so that this is a crude estimate.

In Table 10.2, we report how our lower bound compares to the lower bound that we obtain in this way on the running time of the Schönhage-Strassen algorithm. As we did in Table 10.1, the fourth column is computed by determining experimentally the individual cost of each of the underlying multiplications. For this, we timed GMP's internal routine `mpn_mul_fft`, as it is called in the implementation. The fifth column of Table 10.2 indicates the real computation time, measured experimentally (we modified GMP's internal `mp_size_t` type to go beyond 31 bits). Our measurements were limited by core memory, since the product of two 2^{40}-bit integers took 1.3TB of RAM. The comparison with the previous column shows that our lower bound on the Schönhage-Strassen time is within a factor of two of the real computation time, which is acceptable.

We conclude from Table 10.2 that an implementation of our algorithm will unlikely beat an implementation of the Schönhage-Strassen algorithm for sizes below 2^{40}. Above 2^{40}, the ratio of our lower bounds is only slightly more than two. We may speculate that an optimized implementation could compensate this gap: by improving the arithmetic modulo a generalized

	Schönhage-Strassen algorithm				§10.1			
bit length	#internal products	internal bit length	lower bound	real time	#internal products	prime	internal bit length	lower bound
2^{30}	2^{15}	$\approx 2^{17}$	$9.73 \cdot 10^{0}$ s	$1.50 \cdot 10^{1}$ s	$2^{25} \cdot 16$	$984^{16}+1$	384	$2.68 \cdot 10^{1}$ s
2^{35}	2^{18}	$\approx 2^{19}$	$3.70 \cdot 10^{2}$ s	$6.03 \cdot 10^{2}$ s	$2^{30} \cdot 19$	$984^{16}+1$	384	$1.02 \cdot 10^{3}$ s
2^{40}	2^{20}	$\approx 2^{22}$	$1.63 \cdot 10^{4}$ s	$3.04 \cdot 10^{4}$ s	$2^{35} \cdot 22$	$1984^{16}+1$	416	$4.84 \cdot 10^{4}$ s
2^{45}	2^{23}	$\approx 2^{24}$	$7.90 \cdot 10^{5}$ s	—	$2^{40} \cdot 25$	$1984^{16}+1$	416	$1.76 \cdot 10^{6}$ s
2^{50}	2^{25}	$\approx 2^{27}$	$2.88 \cdot 10^{7}$ s	—	$2^{45} \cdot 28$	$1984^{16}+1$	416	$6.31 \cdot 10^{7}$ s
2^{55}	2^{28}	$\approx 2^{29}$	$1.05 \cdot 10^{9}$ s	—	$2^{50} \cdot 31$	$4016^{16}+1$	448	$2.23 \cdot 10^{9}$ s
2^{60}	2^{30}	$\approx 2^{32}$	$3.44 \cdot 10^{10}$ s	—	$2^{55} \cdot 34$	$4016^{16}+1$	448	$7.84 \cdot 10^{10}$ s

Table 10.2: Comparison of lower bounds on the running time of the Schönhage-Strassen algorithm and the practical algorithm described in 10.1. The right half is from Table 10.1. Timings measured on an Intel Xeon E7-4850v3 CPU (2.20GHz).

Fermat prime for example.

A direction to consider for optimization can be to improve on the time needed for internal multiplications. Considering that multiplication in $\mathcal{R}_1$ can be though as multiplications of polynomials modulo $X^{2^{\lambda_1}}+1$, the idea is to use efficient algorithms to multiply polynomials modulo X^P+1 where P is less than 2^{λ_1}. For example, we may represent elements of $\mathcal{R}_1$ in radix r_1^2 instead of r_1. For $\lambda_1 = 2$, we multiply

$$a_0 + a_1 r_1 + a_2 r_1^2 + a_3 r_1^3 \text{ and } b_0 + b_1 r_1 + b_2 r_1^2 + b_3 r_1^3$$

modulo r_1^4+1. The conversion in radix r_1^2 leads to the computation and the multiplication of

$$(a_0 + a_1 r_1) + (a_2 + a_3 r_1) r_1^2 \text{ and } (b_0 + b_1 r_1) + (b_2 + b_3 r_1) r_1^2.$$

In some cases, it might lead to a smaller bit length, at the expense of some extra conversion costs. For example for $p_1 = 1984^{16}+1$, working in radix 1984^2 leads to polynomials of length 8, and a bit length of $8 \cdot 48 = 384$ bits, instead of 416 bits (see Table 10.1). Another possibility is to use the multipoint Kronecker substitution proposed by Harvey in [29]. For this same example, evaluating at $+2^{24}$ and -2^{24}, we can compute the product via two multiplications of two 192-bit integers, which might be faster. For both ideas however, we have not taken into account the conversion costs, and it seems difficult to be very confident about the induced benefit.

Bibliography

[1] S. Ballet and J. Pieltant. On the tensor rank of multiplication in any extension of $\mathbb{F}_2$. *Journal of Complexity*, 27(2):230–245, 2011. doi:10.1016/j.jco.2011.01.008.

[2] R. Barbulescu, J. Detrey, N. Estibals, and P. Zimmermann. Finding optimal formulae for bilinear maps. *Arithmetic of finite fields: 4th International Workshop, WAIFI 2012, Bochum, Germany, July 16-19, 2012. Proceedings*, pages 168–186, 2012. doi:10.1007/978-3-642-31662-3_12.

[3] R. Barbulescu, J. Detrey, N. Estibals, and P. Zimmermann. Finding optimal formulae for bilinear maps. AriC Seminar, Mar. 2012. URL: https://hal.inria.fr/hal-01413162.

[4] P. T. Bateman and R. A. Horn. A heuristic asymptotic formula concerning the distribution of prime numbers. *Math. Comput.*, 16(79):pp. 363–367, 1962. doi:10.1090/S0025-5718-1962-0148632-7.

[5] A. Bernardi, J. Brachat, P. Comon, and B. Mourrain. General tensor decomposition, moment matrices and applications. *Journal of Symbolic Computation*, 52:51–71, 2013. International Symposium on Symbolic and Algebraic Computation. doi:10.1016/j.jsc.2012.05.012.

[6] D. J. Bernstein. Multidigit multiplication for mathematicians, 2001. http://cr.yp.to/papers.html#m3.

[7] L. I. Bluestein. A linear filtering approach to the computation of discrete Fourier transform. *Audio and Electroacoustics, IEEE Transactions on*, 18(4):451–455, Dec 1970. doi:10.1109/TAU.1970.1162132.

[8] M. Bläser. On the complexity of the multiplication of matrices of small formats. *Journal of Complexity*, 19(1):43–60, 2003. doi:10.1016/S0885-064X(02)00007-9.

[9] W. Bosma, J. Cannon, and C. Playoust. The Magma algebra system. I. The user language. *J. Symbolic Comput.*, 24(3-4):235–265, 1997. Computational algebra and number theory (London, 1993). doi:10.1006/jsco.1996.0125.

[10] R. P. Brent and P. Zimmerman. *Modern computer arithmetic*. Cambridge University Press, 2010.

[11] R. W. Brockett and D. Dobkin. On the optimal evaluation of a set of bilinear forms. *Linear Algebra and its Applications*, 19(3):207–235, 1978. doi:10.1016/0024-3795(78)90012-5.

[12] P. Bürgisser, M. Clausen, and M. A. Shokrollahi. *Algebraic Complexity Theory*. Springer, 1st edition, 2010.

[13] V. P. Burichenko. On symmetries of the Strassen algorithm. *CoRR*, abs/1408.6273, 2014. arXiv:1408.6273.

[14] D. Chudnovsky and G. Chudnovsky. Algebraic complexities and algebraic curves over finite fields. *Journal of Complexity*, 4(4):285–316, 1988. doi:10.1016/0885-064X(88)90012-X.

[15] H. Cohn, R. Kleinberg, B. Szegedy, and C. Umans. Group-theoretic algorithms for matrix multiplication. In *Foundations of Computer Science, 2005. FOCS 2005. 46th Annual IEEE Symposium*, pages 379–388. IEEE, Oct 2005. doi:10.1109/SFCS.2005.39.

[16] H. Cohn and C. Umans. A group-theoretic approach to fast matrix multiplication. In *Foundations of Computer Science, 2003. Proceedings. 44th Annual IEEE Symposium*, pages 438–449. IEEE, Oct 2003. doi:10.1109/SFCS.2003.1238217.

[17] J. W. Cooley and J. W. Tukey. An algorithm for the machine calculation of complex Fourier series. *Math. Comput.*, 19:297–301, 1965. doi:10.1090/S0025-5718-1965-0178586-1.

[18] D. Coppersmith and S. Winograd. Computational algebraic complexity editorial matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9(3):251–280, 1990. doi:10.1016/S0747-7171(08)80013-2.

[19] S. Covanov. Improved method to find optimal formulae for bilinear maps. Preprint, May 2017. URL: https://hal.inria.fr/hal-01519408.

[20] S. Covanov and E. Thomé. Fast integer multiplication using generalized Fermat primes. Preprint submitted to Mathematics of Computation, to appear, Jan. 2016. URL: https://hal.inria.fr/hal-01108166.

[21] A. De, P. P. Kurur, C. Saha, and R. Saptharishi. Fast integer multiplication using modular arithmetic. *SIAM J. Comput.*, 42(2):685–699, 2013. doi:10.1137/100811167.

[22] H. F. de Groote. *Lectures on the Complexity of Bilinear Problems*. Springer-Verlag, 1987.

[23] H. Dubner and Y. Gallot. Distribution of generalized Fermat prime numbers. *Math. Comput.*, 71(238):825–832, 2002. doi:10.1090/S0025-5718-01-01350-3.

[24] P. Elliott. Primes in progressions to moduli with a large power factor. *The Ramanujan Journal*, 13(1-3):241–251, 2007. doi:10.1007/s11139-006-0250-4.

[25] M. Fürer. On the complexity of integer multiplication (extended abstract). Technical Report CS-89-17, Pennsylvania State University, 1989.

[26] M. Fürer. Faster integer multiplication. *SIAM J. Comput.*, 39(3):979–1005, 2009. doi:10.1137/070711761.

[27] P. Gaudry, A. Kruppa, and P. Zimmermann. A GMP-based implementation of Schönhage-Strassen's large integer multiplication algorithm. In *Proceedings of the 2007 International Symposium on Symbolic and Algebraic Computation*, ISSAC '07, pages 167–174, New York, NY, USA, 2007. ACM. doi:10.1145/1277548.1277572.

[28] T. Granlund and the GMP development team. *GNU MP: The GNU Multiple Precision Arithmetic Library*, 2016. version 6.1.0, http://gmplib.org/.

[29] D. Harvey. Faster polynomial multiplication via multipoint kronecker substitution. *Journal of Symbolic Computation*, 44(10):1502 – 1510, 2009. doi:10.1016/j.jsc.2009.05.004.

[30] D. Harvey and J. van der Hoeven. Faster integer multiplication using plain vanilla FFT primes, 2016. arXiv:1611.07144.

[31] D. Harvey and J. van der Hoeven. Faster integer multiplication using short lattice vectors. *ArXiv e-prints*, Feb. 2018. arXiv:1802.07932.

[32] D. Harvey, J. van der Hoeven, and G. Lecerf. Even faster integer multiplication. *J. Complexity*, 36:1–30, 2016. doi:10.1016/j.jco.2016.03.001.

[33] D. F. Holt, B. Eick, and E. A. O'Brien. *Handbook of computational group theory*. Discrete mathematics and its applications. Chapman & Hall/CRC, Boca Raton, 2005. URL: http://opac.inria.fr/record=b1102239.

[34] J. E. Hopcroft and L. R. Kerr. On minimizing the number of multiplications necessary for matrix multiplication. *SIAM Journal on Applied Mathematics*, 20(1):30–36, 1971. doi:10.1137/0120004.

[35] J. Håstad. Tensor rank is NP-complete. *Journal of Algorithms*, 11(4):644–654, 1990. doi:10.1016/0196-6774(90)90014-6.

[36] J. JáJá. Optimal evaluation of pairs of bilinear forms. *SIAM Journal on Computing*, 8(3):443–462, 1979. doi:10.1137/0208037.

[37] A. Karatsuba and Y. Ofman. Multiplication of multidigit numbers on automata. *Soviet Physics-Doklady*, 7:595–596, 1963. (English translation).

[38] D. E. Knuth. *The art of computer programming, volume 2 (3rd ed.): seminumerical algorithms*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997.

[39] L. Kronecker. *Grundzüge einer arithmetischen theorie der algebraischen grössen*. G. Reimer, 1882.

[40] J. D. Laderman. A noncommutative algorithm for multiplying 3×3 matrices using 23 multiplications. *Bull. Amer. Math. Soc.*, 82(1):126–128, 1976.

[41] J. M. Landsberg. New lower bounds for the rank of matrix multiplication. *CoRR*, abs/1206.1530, 2012. arXiv:1206.1530.

[42] J. M. Landsberg. An introduction to geometric complexity theory. *CoRR*, abs/1509.02503, 2015. arXiv:1509.02503.

[43] F. Le Gall. Powers of tensors and fast matrix multiplication. In *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation*, ISSAC '14, pages 296–303. ACM, 2014. doi:10.1145/2608628.2608664.

[44] U. V. Linnik. On the least prime in an arithmetic progression. I. The basic theorem. *Rec. Math. [Mat. Sbornik] N.S.*, 15(57):139–178, 1944.

[45] P. Montgomery. Five, six, and seven-term Karatsuba-like formulae. *IEEE Transactions on Computers*, 54(3):362–369, 2005. doi:10.1109/TC.2005.49.

[46] I. Oseledets. Optimal Karatsuba-like formulae for certain bilinear forms in GF(2). *Linear Algebra and its Applications*, 429(8–9):2052–2066, 2008. doi:10.1016/j.laa.2008.06.004.

[47] C. M. Papadimitriou. *Computational complexity*. Addison-Wesley, Reading, Massachusetts, 1994.

[48] J. Pieltant and H. Randriam. New uniform and asymptotic upper bounds on the tensor rank of multiplication in extensions of finite fields. *Math. Comp.*, 84(294):2023–2045, 2015. doi:10.1090/S0025-5718-2015-02921-4.

[49] C. Pomerance. On the distribution of amicable numbers. *J. Reine Angew. Math.*, pages 217–222, 1977.

[50] H. Randriambololona. Bilinear complexity of algebras and the Chudnovsky–Chudnovsky interpolation method. *Journal of Complexity*, 28(4):489–517, 2012. doi:10.1016/j.jco.2012.02.005.

[51] A. Schönhage. Asymptotically fast algorithms for the numerical multiplication and division of polynomials with complex coeficients. In J. Calmet, editor, *Computer Algebra, EUROCAM '82, European Computer Algebra Conference, Marseille, France, 5-7 April, 1982, Proceedings*, volume 144 of *lncs*, pages 3–15. Springer, 1982. URL: https://doi.org/10.1007/3-540-11607-9_1, doi:10.1007/3-540-11607-9_1.

[52] A. Schönhage and V. Strassen. Schnelle Multiplikation großer Zahlen. *Computing*, 7(3-4):281–292, 1971. doi:10.1007/BF02242355.

[53] I. Shparlinski. On finding primitive roots in finite fields. *Theor. Comput. Sci.*, 157(2):273–275, May 1996. doi:10.1016/0304-3975(95)00164-6.

[54] A. V. Smirnov. The bilinear complexity and practical algorithms for matrix multiplication. *Computational Mathematics and Mathematical Physics*, 53(12):1781–1795, 2013. doi:10.1134/S0965542513120129.

[55] A. Stothers. *On the complexity of matrix multiplication.* PhD thesis, University of Edinburgh, 2010.

[56] V. Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13(4):354–356, 1969. doi:10.1007/BF02165411.

[57] A. L. Toom. The complexity of a scheme of functional elements realizing the multiplication of integers. *Soviet Mathematics Doklady*, 3:714–716, 1963. (English translation).

[58] J. van der Hoeven. The truncated fourier transform and applications. In *Proceedings of the 2004 International Symposium on Symbolic and Algebraic Computation*, ISSAC '04, pages 290–296, New York, NY, USA, 2004. ACM. doi:10.1145/1005285.1005327.

[59] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra.* Cambridge University Press, New York, NY, USA, 1999.

[60] A. Waksman. On winograd's algorithm for inner products. *IEEE Transactions on Computers*, 19(4):360–361, Apr. 1970. doi:10.1109/T-C.1970.222926.

[61] V. V. Williams. Multiplying matrices faster than Coppersmith-Winograd. In *Proceedings of the Forty-fourth Annual ACM Symposium on Theory of Computing*, STOC '12, pages 887–898. ACM, 2012. doi:10.1145/2213977.2214056.

Résumé

Depuis 1960 et le résultat fondateur de Karatsuba, on sait que la complexité de la multiplication (d'entiers ou de polynômes) est sous-quadratique : étant donné un anneau $\mathcal{R}$ quelconque, le produit sur $\mathcal{R}[X]$ des polynômes $a_0 + a_1X$ et $b_0 + b_1X$, pour tous a_0, a_1, b_0 et b_1 dans $\mathcal{R}$, peut être calculé en seulement trois et non pas quatre multiplications sur $\mathcal{R}$: $(a_0 + a_1X)(b_0 + b_1X) = m_0 + (m_2 - m_0 - m_1)X + m_1X^2$, avec les trois produits $m_0 = a_0b_0$, $m_1 = a_1b_1$ et $m_2 = (a_0 + a_1)(b_0 + b_1)$. De la même manière, l'algorithme de Strassen permet de multiplier deux matrices $2n \times 2n$ en seulement sept produits de matrices $n \times n$.

Les deux exemples précédents tombent dans la catégorie des applications bilinéaires : des fonctions de la forme $\Phi : K^m \times K^n \to K^\ell$, pour un corps donné K, linéaires en chacune des deux variables. Parmi les applications bilinéaires les plus classiques, on trouve ainsi la multiplication de polynômes, de matrices, ou encore d'éléments d'extensions algébriques de corps finis. Étant donnée une application bilinéaire Φ, calculer le nombre minimal de multiplications nécessaires au calcul de cette application est un problème NP-difficile. L'objectif de cette thèse est de proposer des algorithmes minimisant ce nombre de multiplications. Deux angles d'attaques ont été suivis.

Un premier aspect de cette thèse est l'étude du problème du calcul de la complexité bilinéaire sous l'angle de la reformulation de ce problème en termes de recherche de sous-espaces vectoriels de matrices de rang donné. Ce travail a donné lieu à un algorithme tenant compte de propriétés intrinsèques aux produits considérés tels que les produits matriciels ou polynomiaux sur des corps finis. Cet algorithme a permis de trouver toutes les décompositions possibles, sur $\mathbb{F}_2$, pour le produit de polynômes modulo X^5 et le produit de matrices 3×2 par 2×3.

Un autre aspect de ma thèse est celui du développement d'algorithmes asymptotiquement rapides pour la multiplication entière. Une famille particulière d'algorithmes récents ont été proposés suite à un article de Fürer publié en 2007, qui proposait un premier algorithme, reposant sur la transformée de Fourier rapide (FFT) permettant de multiplier des entiers de n bits en $O\left(n \log n \cdot 2^{O(\log^* n)}\right)$, où $\log^*$ est la fonction logarithme itéré. Dans cette thèse, un algorithme dont la complexité dépend d'une conjecture de théorie des nombres est proposé, reposant sur la FFT et l'utilisation de premiers généralisés de Fermat. Une analyse de complexité permet d'obtenir une estimation en $O(n \log n \cdot 4^{\log^* n})$.

Mots-clés: multiplication, algorithme, complexité, rang, bilinéaire, entiers, matrices, polynômes, FFT

Abstract

Since 1960 and the result of Karatsuba, we know that the complexity of the multiplication (of integers or polynomials) is sub-quadratic: given a ring $\mathcal{R}$, the product in $\mathcal{R}[X]$ of polynomials $a_0 + a_1X$ and $b_0 + b_1X$, for any a_0, a_1, b_0 and b_1 in $\mathcal{R}$, can be computed with three and not four multiplications over $\mathcal{R}$: $(a_0 + a_1X)(b_0 + b_1X) = m_0 + (m_2 - m_0 - m_1)X + m_1X^2$, with the three multiplications $m_0 = a_0b_0$, $m_1 = a_1b_1$ et $m_2 = (a_0 + a_1)(b_0 + b_1)$. In the same manner, Strassen's algorithm allows one to multiply two matrices $2n \times 2n$ with only seven products of matrices $n \times n$.

The two previous examples fall in the category of bilinear maps: these are functions of the form $\Phi : K^m \times K^n \to K^\ell$, given a field K, linear in each variable. Among the most classical bilinear maps, we have the multiplication of polynomials, matrices, or even elements of algebraic extension of finite fields. Given a bilinear map Φ, computing the minimal number of multiplications necessary to the evaluation of this map is a NP-hard problem. The purpose of this thesis is to propose algorithms minimizing this number of multiplications. Two angles of attack have been studied.

The first aspect of this thesis is to study the problem of the computation of the bilinear complexity under the angle of the reformulation of this problem in terms of research of matrix subspaces of a given rank. This work led to an algorithm taking into account intrinsic properties of the considered products such as matrix or polynomial products over finite fields. This algorithm allows one to find all the possible decompositions, over $\mathbb{F}_2$, for the product of polynomials modulo X^5 and the product of matrices 3×2 by 2×3.

Another aspect of this thesis was the development of fast asymptotic methods for the integer multiplication. There is a particular family of algorithms that has been proposed after an article by Fürer published in 2007. This article proposed a first algorithm, relying on fast Fourier transform (FFT), allowing one to multiply n-bit integers in $O\left(n \log n \cdot 2^{O(\log^* n)}\right)$, where $\log^*$ is the iterated logarithm function. In this thesis, an algorithm, relying on a number theoretical conjecture, has been proposed, involving the use of FFT and generalized Fermat primes. With a careful complexity analysis of this algorithm, we obtain a complexity in $O(n \log n \cdot 4^{\log^* n})$.

Keywords: multiplication, algorithm, complexity, rank, bilinear, integers, matrix, polynomials, FFT