Reconnaissance de codes correcteurs d'erreurs

Maxime Côte

Thèse de Doctorat

présentée à
l'École Polytechnique

pour obtenir le titre de
Docteur de l'École Polytechnique en Informatique

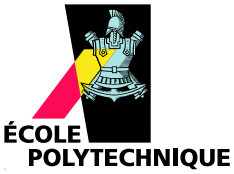
par
Maxime Côte

sous la direction de
Nicolas Sendrier
et de
Jean-Pierre Tillich

Reconnaissance de codes correcteurs d'erreurs

Soutenue le 22 mars 2010

Rapporteurs :	Thierry Berger
	Pierre Loidreau
Directeurs :	Nicolas Sendrier
	JeanPierre Tillich
Jury :	Daniel Augot
	Gilles Burel
	Claude Berrou
	Denys Boiteau



INRIA

ipsis
INGÉNIERIE POUR SIGNAUX ET SYSTÈMES

Table des matières

Table des figures

Remerciements

Dans cette partie, je tiens à remercier toutes les personnes qui m'ont permises d'effectuer cette thèse dans les meilleures conditions.

Tout d'abord, je remercie Nicolas Sendrier et Jean-Pierre Tillich mes deux co-directeurs. Le premier pour sa disponibilité, toujours prêt à conseiller, donner de nouvelles pistes. Le deuxième pour sa maîtrise des statistiques et la rigueur d'écriture qu'il m'a inculqué lors de la relecture de ma thèse. Leurs idées débordantes m'ont été fortes utiles à l'accomplissement de cette thèse. Aujourd'hui je ne peux qu'être fier d'avoir eu des directeurs comme eux.

Je tiens à remercier Mathieu Cluzeau et Matthieu Finiasz pour avoir pris énormément de leur temps à m'expliquer les différentes méthodes de reconnaissance de codes afin d'améliorer ma compréhension du domaine.

Un grand « MERCI » à tout le projet SECRET de l'INRIA Rocquencourt, Pascal Charpin, Anne Canteaut mais aussi Christelle, Cédric, Yann, Frédéric, Maria, Andrea, les deux Stéphane, Baskahar, mon voisin de bureau Vincent et tous les autres membres du projet SECRET, pour les longues discussions et débats dans la salle à café. Ces échanges m'ont permis d'en apprendre plus sur la cryptographie et les codes ainsi que sur les axes de recherches de mes collègues thésards. Grâce à ces personnes, le projet SECRET est un endroit où il fait bon travailler.

Cette thèse n'aurait pu voir le jour sans la participation de la société IPSIS, qui a financé ces trois années. Je remercie tout particulièrement Denys Boiteau qui représente cette entreprise en faisant partie de mon jury de thèse. Il m'a apporté un point de vue industriel lors de ce projet et m'a permis de réaliser une véritable thèse en acceptant la dissociation entre le travail de recherche et celui d'ingénieur.

Je tiens à remercier Thierry Berger et Pierre Loidreau mes deux rapporteurs qui ont autorisé la soutenance de cette thèse, ainsi que Daniel Augot, Claude Berrou et Gilles Burel pour avoir accepté de faire partie de mon jury.

Un doctorat est souvent vu comme un parcours scolaire long et compliqué. Je remercie mes parents qui m'ont laissé la liberté de mes choix dans mon orientation et m'ont soutenu tout au long de ce parcours. Je ne peux pas penser à mes parents sans penser à toute ma famille notamment à mon frère et ma soeur.

Enfin, ma dernière pensée va à mon amie Amélie avec qui je partage ma vie et qui m'a toujours soutenue pendant ces trois années de thèse. Je lui souhaite de réussir dans ses études tout autant que moi.

Introduction

Ce manuscrit représente le fruit de trois années de recherche en partenariat avec la société Ipsis [1] et le projet SECRET de l'INRIA de Paris-Rocquencourt [2]. Ces travaux auront permis la publication de l'article [CS09] et la soumission (encore en cours) d'un autre article [CS] à ISIT 2010. Les travaux les plus récents sur les turbo-codes font encore l'objet de recherches afin de valider leur théorie et fonctionnement.

Mais, au-delà de la présentation de mes travaux, j'espère, à l'aide de ce document, faire naître en vous un intérêt pour cette discipline peu connue et pourtant si captivante qu'est la reconnaissance de codes. Alliant un soupçon de cryptographie par sa complexité et ses méthodes d'approche du problème ainsi qu'une bonne dose de codes et de théorie de l'information utile au maniement d'outils et à la compréhension de familles de codes qui, dans la plus grande discrétion, sont utilisés de tous depuis plus d'un demi-siècle. Ce manuscrit s'intéresse particulièrement à la famille des codes convolutifs, inventés en 1955. Ces codes sont encore utilisés aujourd'hui, notamment dans les turbo-codes. Nous avons choisi ces deux types de construction dans notre thèse car leur utilisation très répandue de nos jours en font des cibles de choix dans un contexte de reconnaissance. Ce choix aurait aussi pu être fait à cause de leur remarquable construction due à l'ingéniosité des personnes qui les ont créés afin de défier les limites de théorie de l'information établîes par Shannon.

Ce mémoire est structuré en quatre parties. Dans la première partie, nous introduirons la théorie des codes et les éléments nécessaires à la compréhension de ce manuscrit. Nous décrirons les premiers codes créés jusqu'à ceux utilisés quotidiennement de nos jours en passant par certains dont les capacités de correction ont été dépassées, mais qui à leur apogée, ont connu de grands succès comme celui de la conquête spatiale.

La deuxième partie permettra de lever le voile sur la problématique de cette thèse. Nous donnerons un état de l'art des méthodes plus ou moins performantes utilisées en reconnaissance. Ces algorithmes qui sont souvent originaires du domaine de la cryptographie et adaptés à la reconnaissance de codes. En effet, la recherche de mots de poids faible peut à la fois servir d'outil de recherche de relations de parité dans un cryptosystème que d'algorithme de décodage d'un code, et peut être apparentée à un problème de recherche de code dual dans un milieu bruité.

La troisième partie présentera mes travaux sur les codes convolutifs. Nous commencerons par rappeler en détail les propriétés des codes convolutifs. Nous

1. Ingénierie Pour SIgnaux et Systemes, 3 square du Chêne Germain, 35510 Cesson-Sévigné
2. Institut Natiaonal pour la Recherche en Informatique et en Automatique, Domaine de Voluceau-Rocquencourt, B.P 105-78153 Le Chesnay Cedex France

définirons les notions essentielles à la résolution du problème. Nous présenterons deux résultats algorithmiques : une méthode utilisant les polynômes et la forme de Smith et une autre méthode algébrique se passant de la contrainte des calculs polynomiaux pour se contenter de calculs d'algèbre linéaire. Nous donnerons les résultats des simulations menées avec notre algorithme pour en déterminer les performances et les limites. Enfin, nous étudierons le problème de la reconnaissance des codes poinçonnés. Ce problème consiste à retrouver à partir d'un code (n', k'), le code parent(non poinçonné) (n, k) et le motif de poinçonnage permettant de simplifier le décodage de Viterbi.

Enfin dans une dernière partie, nous étendrons le problème aux turbo-codes parallèles. Nous donnerons une méthode de résolution fondée sur une reconnaissance à la fois des polynômes et de l'entrelaceur au moyen de certaines statistiques. Sous l'hypothèse que le rapport des polynômes générateurs possède un terme constant nous serons capable de reconstruire notre codeur. Nous analyserons les limites et performances de notre méthode et la comparerons à deux autres encore expérimentales. Ces idées mettent en avant des techniques très différentes qui peuvent naître dans la résolution d'un problème.

Chapitre 1

Codage canal et codes correcteurs d'erreurs

Dans ce manuscrit nous allons étudier les codes correcteurs dans un contexte de rétro-ingénierie. Pour cela, Nous présentons dans ce chapitre à la fois des généralités sur une chaîne de communication et des éléments de la théorie des codes correcteurs d'erreurs.

1.1 Schéma de transmission

Un système de communication permet à une source d'envoyer un message à un destinataire capable de comprendre le message. L'information numérique envoyée est formée d'une suite de symboles que l'on appellera message ou séquence d'information. Le schéma de transmission de la figure 1.1 permet de mieux comprendre le cheminement de l'information de la source au destinataire. Nous parlerons de message binaire lorsque la séquence d'information sera constituée de $\ll 0 \gg$ ou de $\ll 1 \gg$ et de séquence q-aire lorsque les symboles prendront leurs valeurs dans un alphabet $\mathcal{A}$ fini à q éléments, souvent nous aurons $\mathcal{A} = \{0, 1\}$.

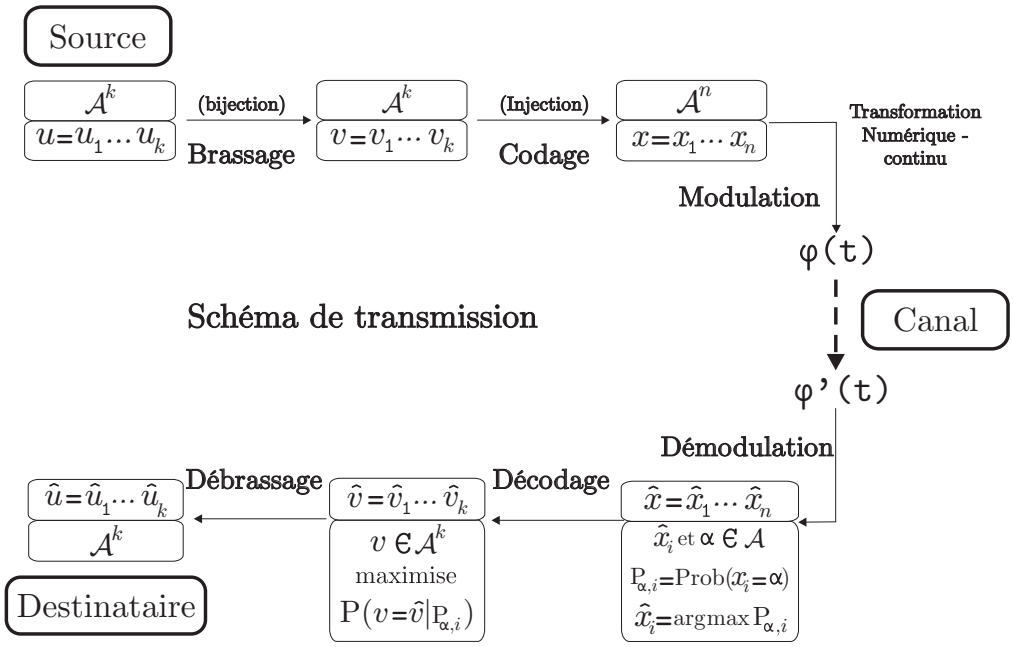


FIGURE 1.1 – Schéma d'une chaîne de transmission

Nous utiliserons les notations de ce schéma dans la suite de ce document. Les notations avec accents circonflexes correspondent aux données estimées. La valeur de n est toujours supérieur à k.

La séquence d'information est envoyée par la source, celle-ci subit divers traitements comme le brassage ou le codage puis est envoyée au travers d'un canal. A la réception du signal, les étapes sont effectuées dans l'ordre inverse pour que le destinataire retrouve la séquence d'information envoyée. Le canal constitue le point névralgique du système, il est décrit figure 1.2. Il est constitué en entrée de la séquence codée que l'on module afin de convertir le signal numérique en signal continu, ce signal est envoyé au travers d'un canal de communication (air, fibre, ...) et est démodulé en réception afin de retrouver notre séquence codée.

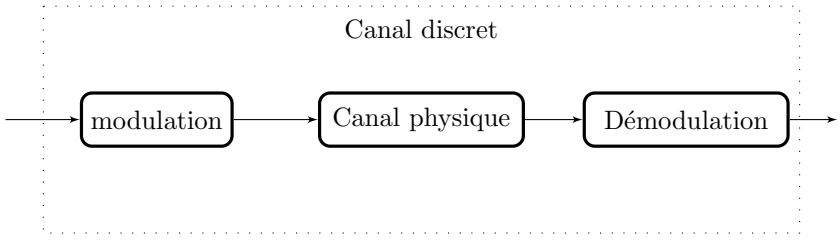


FIGURE 1.2 – Schéma du canal discret

Ensuite le codage comportera un code correcteur d'erreurs (convolutif, en bloc, concaténé...). Nous négligeons la possibilité d'un entrelacement ou d'une modulation codée selon le canal de communication qui ne sont pas traités dans ce document. Nous ne l'avons pas représenté sur le schéma mais la séquence

codée peut préalablement subir un traitement cryptographique inutile à l'étude de notre problème.

Enfin, le brassage est une opération réversible, ayant pour objectif d'éliminer les longues séquences de symboles identiques susceptibles de nuire au bon fonctionnement de la démodulation. Il est facilement compréhensible que de très longues séquences de « 1 » ou de « 0 » produites par une source imparfaite soient difficilement modulables à l'émission et démodulables à la réception. L'objectif est d'éviter ce phénomène, pour cela on applique une fonction linéaire, qui va mélanger de façon « aléatoire » les bits sur une grande séquence de code afin d'empêcher les longues séries de bits identiques. L'opération inverse est réalisée lors de la réception : le signal continu bruité est démodulé, cela donne en pratiques des probabilités sur chaque symboles de la séquence codée. Puis on utilise le modèle disponible du canal de communication pour essayer de reconstituer le mot de code transmis. Enfin le mot de code ainsi reconstitué est « débrassé » pour obtenir le mot émis par la source d'origine.

Nous introduisons dans ce paragraphe une notion de distance de Hamming. Soient $\mathcal{A}^n$ l'ensemble des mots (élément de $\mathcal{A}$) de longueur n sur $\mathcal{A}$. $u = (u_1, u_2, \ldots, u_n) \in \mathcal{A}^n$ et $v = (v_1, v_2, \ldots, v_n) \in \mathcal{A}^n$.

– On définit la distance de Hamming entre u et v comme

$$d_H(u, v) = |\{i | 1 \leq i \leq n, u_i \neq v_i\}|$$

$d_H(u, v)$ est une métrique, on appelle alors espace de Hamming sur $\mathcal{A}$ l'ensemble $\mathcal{A}^n$ muni de la métrique $d_H(u, v)$

– Si $\mathcal{A}$ est un groupe, on définit le poids de Hamming $|u|_H$ d'un mot $x \in \mathcal{A}^n$ par le nombre de ses coordonnées non nulles. Ainsi

$$|u|_H = d_H(u, 0)$$

où 0 est le mot de $\mathcal{A}^n$ ayant toutes ses coordonnées égales à l'élément neutre de $\mathcal{A}$.

1.2 Théorie de l'information

1.2.1 Modèles de canaux

Un code correcteur est l'ensemble des mots de code ($\in \mathcal{A}^n$) obtenus après le codage. Les systèmes de communication utilisent un support de transmission, aussi appelé « canal de transmission » , afin d'échanger de l'information d'un point à un autre. Le canal peut être de différents types : câble, fibre optique, canal radioélectrique... N'étant pas parfait, il introduit des perturbations, affaiblissement, écho, bruit qui détériorent l'information émise et créent des erreurs dans le message. Afin de fiabiliser le message (empêcher la perte de données due aux perturbations, bruit), les systèmes intègrent un processus de protection du message émis. Le principe général de cette protection est l'ajout de redondance, c'est-à-dire d'information supplémentaire, la plus optimale possible en terme de coût, de volume et de contraintes qui dépendent largement du canal.

Décrivons dans un premier temps les modèles simplifiés de canaux de communication les plus communs.

1.2.1.1 Le canal binaire symétrique

C'est un canal binaire caractérisé par la probabilité d'erreur p qu'au cours de la transmission un bit (0 ou 1) soit modifié en son opposé. Ces modifications se produisent indépendamment sur chacun des bits transmis On représente ce canal par le schéma 1.3.

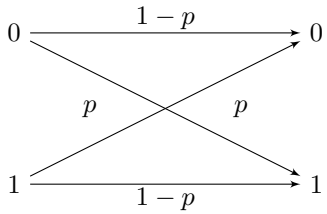


FIGURE 1.3 – Canal binaire symétrique de probabilité d'erreur p

1.2.1.2 Le canal Gaussien

Soit $x_i \in \mathcal{A} = \{0, 1\}$ la séquence d'entrée discrète du canal gaussien. On appelle canal gaussien, le canal liant le signal d'entré au signal de sortie selon la loi.

$$y_i = (-1)^{x_i} + N_i$$

Où N_i est une variable aléatoire suivant une loi normale $\mathcal{N}(0, \sigma^2)$. La sortie y_i est une variable continue. Il est possible de définir une loi de décodage au maximum de vraisemblance (cf. section 1.5). $\widehat{x}_1, \ldots, \widehat{x}_n = \arg\min_x d(x, y)$, où chaque $\widehat{x}_i$ suit la loi

$$\begin{cases} \widehat{x}_i = 0 \text{ si } y_i > 0 \\ \widehat{x}_i = 1 \text{ si } y_i < 0 \end{cases}$$

Si la probabilité que 0 soit envoyé est la même que la probabilité que 1 soit envoyé. La probabilité que la quantification soit erronée ne dépend pas du bit transmis. Appelons cette probabilité P_{err}. On vérifie immédiatement que

$$P_{err} = P(N_i > 1) = \int_1^\infty \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-t^2}{2\sigma^2}} dt$$

1.3 Définition générale d'un code

Définition 1.1 *Un code C sur $\mathcal{A}$ est une partie non vide de l'espace de Hamming $\mathcal{A}^n$, n est appelé longueur du code, les éléments de C sont appelés mots de code.*

Définition 1.2 *On appelle distance minimale d'un code C sur $\mathcal{A}$ l'entier*

$$d = \min \{d_H(u, v) | u, v \in \mathcal{A}, u \neq v\}$$

Si $\mathcal{A}$ est un groupe on appelle poids minimal d'un code C l'entier

$$\min \{|u|_H | u \in C, u \neq 0\}$$

Dans le cas où $\mathcal{C}$ est un sous groupe de $\mathcal{A}^n$ la distance minimale et le poids minimal sont égaux

Pour un mot de code x donné, on peut définir sa région de décodage comme l'ensemble des mots $\mathcal{A}^n$ se décodant en x. Quand l'ensemble de décodage consiste à chercher le mot le plus proche au sens de la distance de Hamming, on parle parfois de décodage « au plus proche » , il est clair que la région de décodage d'un mot de code donné x contient toujours la boule de rayon $\lfloor \frac{d-1}{2} \rfloor$ centrée autour de x. On vérifie en effet dans ce cas tout mot de la boule est nécessairement plus proche de x que de tout autre mot de code. Un code parfait est un code où toutes les régions de décodage sont nécessairement ces boules.

1.4 Définition d'un code linéaire

L'alphabet $\mathcal{A}$ (avec $|\mathcal{A}| = p^n$ où p est un nombre premier) peut être muni d'une structure de corps, on appelle $\mathbb{F}_q$ le corps à q élément.

Rappel : Un corps est un ensemble K muni de deux lois internes notées en général $+$ et $\times$ vérifiant
- $(K, +)$ forme un groupe commutatif dont l'élément neutre est noté 0.
- $(K \setminus \{0\}, \times)$ forme un groupe multiplicatif.
- la multiplication est distributive pour l'addition (à gauche comme à droite).

On parle de corps $(K, +, \times)$.

$\mathbb{F}_q[x]$ l'anneau des polynômes sur $\mathbb{F}_q$. Enfin, on appelle $\mathbb{F}_q(x)$ le corps des fonctions rationnels sur $\mathbb{F}_q$.

Un code d'alphabet $\{0, 1\}$ est appelé code binaire, on le note $\mathbb{F}_2$.

Nous nous intéresserons plus particulièrement dans ce manuscrit aux codes linéaires.

Définition 1.3 *Une matrice génératrice d'un code C est une matrice $k \times n$ dont les lignes forment une base de C.*

Pour un même code, il peut exister plusieurs matrices génératrices.

Corollaire 1.4 *Soit G une matrice génératrice $k \times n$ d'un code C et P une matrice inversible $k \times k$. Alors la matrice $G' = PG$ est aussi une matrice génératrice du code C*

Définition 1.5 *Une matrice génératrice polynomiale est une matrice génératrice $k \times n$ à coefficients dans $\mathbb{F}_q[D]$*

Certaines matrices génératrices ont une forme particulière car elles laissent l'information en clair dans le mot de code. Celles-ci sont les matrices systématiques.

Définition 1.6 *Tout code linéaire C possède une matrice génératrice systématique de la forme.*

$$G_{sys} = \left(\begin{array}{c|c} I_k & P \end{array} \right)$$

à une permutation des colonnes près.

La plupart des codes sont définis par leur matrice génératrice, cependant certains codes comme les codes, de Hamming, sont définis par leur matrice de parité.

Définition 1.7 *Soit* $\dim C = k$*, on définit la matrice génératrice tel que*

$$Code = \{xG,\ x \in \mathbb{F}_k^n\}$$

avec G *une matrice de taille* $k \times n$*.*

On peut alors définir la matrice de parité du code comme la matrice de taille $r \times n$ *avec* $r \geq n - k$ *tel que :*

$$Code = \{x \in \mathbb{F}_q^n \mid \mathcal{H}x^\top = 0\}$$

Dans $\mathbb{F}_2^n$, les lignes de $\mathcal{H}$ (de rang $n-k$ plein) forment une base de l'orthogonal de C noté $C^\perp$ et appelé code dual de C.

Définition 1.8 *Soit* $< x, y >= \sum_{i=1}^{n} x_i y_i$ *pour* $x = (x_i),\ 1 \leq i \leq n$ *et* $y = (y_i),\ 1 \leq i \leq n$ *On définit le code dual* $C^\perp$ *:*

$$C^\perp = \{x \in \mathcal{F}_q^n | < x, y >= 0\ \forall y \in \mathcal{C}\}$$

Définition 1.9 *Pour un choix donné de matrice de parité* $\mathcal{H}$*, on définit le syndrome* S *de* $u \in \mathbb{F}_2^n$ *comme le vecteur de* $\mathbb{F}_2^n$ *définit par*

$$S(u) = \mathcal{H}u^\top$$

l'application $S : \mathbb{F}_2^n \to \mathbb{F}_2^n$ *ainsi définie est* $\mathbb{F}_2$*-linéaire.*

1.5 Généralités du décodage

Plusieurs méthodes de décodage existent, le décodage au maximum de vraisemblance est la méthode la plus souhaitable mais n'est pas toujours réalisable en pratique, sauf pour les codes convolutifs. Elle consiste à partir d'un modèle probabiliste du canal à calculer le mot le plus probable compatible avec le mot reçu. Il est facile de montrer que le décodage au maximum de vraisemblance consiste à retrouver le mot de code le plus proche (au sens de Hamming) du mot reçu.

Le décodage en revanche va utiliser la structure algébrique du code ou de son dual. La connaissance de la seule matrice génératrice ne suffit en général pas à décoder de manière efficace, et la structure algébrique n'est pas nécessairement facile à obtenir à partir de la matrice génératrice (c'est même l'un des fondements de la sécurité de certains cryptosystèmes tels que celui de McEliece).

Le décodage d'un code en bloc est un problème difficile (NP-dur). On ne connaît pas d'autre algorithme pour décoder les codes linéaires, que des techniques au coût exponentiel dans le nombre d'erreurs corrigées par bloc.

Pour de petits codes en blocs binaires, ce coût n'est pas un problème et des solutions à base de mise en table donnent des résultats satisfaisants. Dans les autres cas, la connaissance du code (par exemple par la donnée d'une matrice génératrice) sera insuffisante et il faudra retrouver la structure algébrique. Des solutions existent pour les codes GRS [1] pour les codes cycliques dans certains cas favorables, mais pas, par exemple, pour les codes de Goppa ou les codes géométriques.

1. generalised Reed-Solomon = Reed-Solomon généralisés

1.6 Différents codes correcteurs

Dans ce mémoire nous regroupons les codes selon deux classes qu'ils soient en blocs ou convolutifs. A l'intérieur d'une classe nous distinguerons plusieurs familles de codes, chaque famille se distinguant des autres par ses propriétés structurelles (construction, algorithme de décodage). Nous donnons quelques exemples de familles de codes.

1.6.1 Codes en blocs

Nous donnons ci-dessous une liste non exhaustive de familles de code en bloc linéaires. Les codes que nous considérons sont binaires ou sont définis sur une extension du corps à deux éléments. Nous noterons $[n, k, d]$ un code linéaire défini sur un alphabet à q éléments de longueur n, de dimension k et de distance minimale d. Nous avons choisi de ne lister ici que des familles de codes ayant un intérêt pratique (s'ils sont utilisés, typiquement Reed-Solomon ou BCH, ou susceptibles de l'être, typiquement codes géométriques) ou théorique. Certains ouvrages présentent ces codes de manière détaillée, ainsi les documents [LC04, Bos99, Wic03, ML85, Bla03, MS77] introduisent la théories des codes correcteurs d'erreurs et décrivent les codes correcteurs d'erreurs en bloc les plus utilisés.

1.6.1.1 Code à répétition

Le code à répétition consiste à répéter n fois le bit d'information. C'est un code $[n, 1, n]$, de dimension 1, de longueur n et de distance n. Pour n impair, ce code est parfait et optimal lorsque l'on protège un seul bit de donnée.

En théorie de l'information pour un modèle de bruit donné (canaux sans mémoire) et à rendement constant on peut avoir une probabilité d'erreur qui tend vers 0. Pour un code à répétition le rendement décroît pour une probabilité d'erreur qui tend vers 0. Ce code ne tire malheureusement pas parti du faite qu'il est plus efficace de protéger simultanément plusieurs bits d'information.

Le décodage s'effectue par vote majoritaire et la capacité de correction est donnée pas $\frac{n-1}{2}$

Ce code est surtout utilisé en télécommunication pour reconnaître une modulation de signal.

1.6.1.2 Codes de Hamming

Le code de Hamming se définit simplement au moyen d'une matrice de parité $\mathcal{H}$. De dimension $r \times n'$ avec $n' = 2^r - 1$ la matrice de parité possède toutes ses colonnes distinctes non nulles. Le code de Hamming correspond au $2^{k'} - 1 - r$ mots de code de longueur $2^r - 1$ contenus dans le noyau de $\mathcal{H}$. Par exemple, pour $r = 3$ une des matrices de parité s'écrie.

$$\mathcal{H} = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix}$$

Les colonnes de la matrice de parité étant toutes distinctes, la distance minimale du code est 3. Plus généralement, les codes de Hamming sont des codes $[2^r - 1, 2^r - r - 1, 3]$.

Le décodage de ce code s'effectue par syndrome en utilisant la matrice de parité. Soit m le mot de code utilisé, $\tilde{m}$ le message reçu tel que $\tilde{m} = m + e$ avec e l'erreur. Alors le syndrome est $\mathcal{S} = \mathcal{H}\tilde{m}^\top = \mathcal{H}m^\top + \mathcal{H}e^\top = \mathcal{H}e^\top$ avec $m \in \mathcal{H}^\top$. Soit e une erreur de poids 1, alors le syndrome $\mathcal{S}$ nous renseigne sur la colonne où se situe l'erreur. Les colonnes de $\mathcal{H}$ étant toutes distinctes, il est possible de retrouver e et donc m.

Cette méthode permet de corriger une unique erreur, si deux erreurs se produisent, il est incapable de les corriger et de surplus en ajoute une troisième. Le code de Hamming généralisé est un code $[2^r, 2^r - r, 3]$ où le 2^r bit est un bit de parité contrôlant les $2^r - 1$ bits précédents. Ce nouveau code permet de détecter une deuxième erreurs mais sans pouvoir la corriger.

Le code de Hamming pour $r = 1$ contient que des mots nuls, pour $r = 2$ nous avons un code à répétition où $n = 3$. Le code de Hamming est un code parfait pour tout r.

1.6.1.3 Codes cycliques

Les codes cycliques sont des codes linéaires stables par permutation circulaire des coordonnées. Ils ont une forte structure mathématique : si on les voit comme des espaces vectoriels de polynômes (les coordonnées des mots deviennent les coefficients des polynômes). Un code linéaire C est dit cyclique s'il est stable par permutation circulaire des coordonnées, i.e

$$(v_0, v_1, \ldots, v_{n-1}) \in C \Rightarrow (v_{n-1}, v_0, \ldots, v_{n-2}) \in C$$

Si l'on représente le vecteur ci-dessus par le polynôme $v(x) = v_0 + v_1 x + \cdots + v_{n-1}x^{n-1}$, alors la permutation circulaire d'une unité à droite correspond à la multiplication par x modulo $x^n - 1$. Un code cyclique q-aire est un idéal de l'anneau $\mathbb{F}_n = \frac{\mathbb{F}_q[x]}{x^n-1}$ des polynômes à coefficients dans le corps à q éléments modulo $x^n - 1$, et sera de la forme $C = G(x)$, c'est-à-dire que tout élément de C est un multiple d'un certain polynôme $G(x)$ à coefficients q-aires. Lorsqu'il est de degré minimal et unitaire, ce polynôme est unique, nous l'appellerons polynôme générateur du code.

Théorème 1.10 *Soit C un code cyclique binaire de longueur n , on :*

1. *Il existe un unique polynôme unitaire $g(x)$ de degré minimal dans C.*
2. *C est l'idéal engendré par $g(x)$, $g(x)$ est appelé polynôme générateur de C.*
3. *$g(x)$ est un diviseur de $x^n - 1$.*
4. *Tout $c(x) \in C$ s'écrit de façon unique $c(x) = f(x)g(x)$ dans $\mathbb{F}_q[x]$. La dimension de C est $n - r$ où $r = \deg g(x)$*

Codes de Reed-Solomon

Les codes de Reed-Solomon sont des codes cycliques q-aires de longueur $n = q - 1$. Le polynôme générateur d'un code de Reed-Solomon est de la forme

$$g(x) = \prod_{i=1}^{d-1}(x - \alpha^i)$$

où α est un élément primitif du corps fini à q éléments. Il a pour degré $d-1$. Le code correspondant au polynôme générateur ci-dessus a pour longueur $n = q-1$, pour distance minimale d et pour dimension $k = n - d + 1$. Nous avons donc affaire à des codes $[n, k, n - k + 1]$.

Ces codes sont efficaces dans la corrections d'erreurs par paquet du fait de leur structure. Ils ont été employé dans les communications spatiales et restent encore utilisées dans les codes concaténés (cf. 1.6.1.5).

Un code de Reed-Solomon est un code BCH de longueur $q - 1$ sur $\mathbb{F}_q$

Les codes BCH (Bose-Chaudhuri-Hocquenghem)

Les codes BCH binaires sont des codes de Reed-Solomon binaires. Lorsque le cardinal de l'alphabet sur lequel un code de Reed-Solomon est défini est une puissance de 2, alors il est possible de définir son sous-code binaire (les mots de code ayant des coefficients 0 ou 1). Ce sous-code est stable par addition (la somme de deux mots binaires est un mot binaire) et donc linéaire. Un code de Reed-Solomon $[n, k, d]$ avec $q = 2m$ fournira un code BCH binaire $[n, k', d']$. La longueur est conservée, en revanche la distance minimale peut augmenter, et la dimension diminue fortement. Les valeurs exactes de d' et k' sont dépendantes de la structure du corps fini et ne peuvent être décrites par des formules closes simples. Si $g(x)$ est le polynôme générateur du Reed-Solomon, alors, le code BCH aura pour générateur le polynôme binaire de plus bas degré multiple de $g(x)$. Le décodage d'un code BCH sera très similaire au décodage du code de Reed-Solomon dont il est issu.

1.6.1.4 Codes alternants

Les codes BCH et de Reed-Solomon se décodent à l'aide de l'algorithme de Berlekamp-Massey. Après l'invention et surtout la compréhension de cet algorithme, il est apparu que l'on pouvait définir d'autres familles de codes, proches de ces derniers, dans lesquels on retrouvait la même structure algébrique et auxquels on pouvait ainsi appliquer les mêmes procédures de décodage.

Codes de Reed-Solomon généralisés

Les codes de Reed-Solomon généralisés, appelés GRS, sont très voisins des codes de Reed-Solomon. Ils sont obtenus en multipliant chaque coordonnée par une constante non nulle dans le corps à q éléments (les constantes peuvent être différentes pour chaque coordonnée). Les paramètres sont ceux des Reed-Solomon.

Codes de Goppa

Les codes de Goppa binaires sont issus des codes de Reed-Solomon généralisés et permettent d'obtenir une bonne distance minimale.

Si le code GRS d'origine est $[n, k = n - d + 1, d]$, sur un alphabet $q = 2m$ alors le code de Goppa binaire correspondant a généralement pour paramètres $[n, n - md, 2d + 1]$. La distance minimale est le double de ce que l'on attendrait normalement d'un sous-code binaire. Ces codes sont utilisés en cryptologie (cryptosystème de McEliece [KT91, Loi97, LB88, KI01, Gib91]) mais pas pour le codage de canal.

Décodage des codes alternants

Les codes alternants regroupent pratiquement l'ensemble des codes en bloc pour lesquels on dispose d'un décodage algébrique (code de Reed-Solomon, codes BCH, codes de Goppa). Tous ces codes sont des codes GRS (Reed-Solomon généralisés), ou sont définis à partir d'un code GRS (codes BCH ou de Goppa). Nous parlerons du code GRS sous-jacent.

Le décodage des codes alternants s'effectue à l'aide de l'algorithme de Berlekamp-Massey(cf.[Bos99]) ou l'un de ses équivalents (algorithme d'Euclide étendu). Cet algorithme utilise l'arithmétique du corps fini sur lequel le GRS sous-jacent est défini (en général une extension du corps à 2 éléments, c'est-à-dire un corps dont le cardinal est une puissance de 2). Les calculs sont des opérations sur des polynômes à coefficients dans ce corps. Le coût total du décodage est de l'ordre de tn opération dans le corps, où n est la longueur du code et t le nombre d'erreurs corrigées.

1.6.1.5 Codes produits, codes concaténés

Codes produits

Le produit de deux codes linéaires en bloc sur un même alphabet q-aire est défini comme l'ensemble des matrices génératrices dont toutes les lignes sont dans un code et toutes les colonnes dans un autre code. Il s'agit de l'une des premières constructions de codes disposant d'un algorithme de décodage itératif.

Codes concaténés

Les codes concaténés ont constitué, et constituent toujours, l'une des constructions les plus utilisées pour obtenir une protection contre des niveaux de bruit importants. Ils risquent d'être supplantés dans les années à venir par les turbo-codes et les codes LDPC.

Dans un schéma concaténé, l'information est codée deux fois séquentiellement. Tels qu'ils ont été décrits initialement par Forney, ils utilisent deux codes en bloc. Le premier, le code externe, est défini sur un alphabet de grande taille q et le second, le code interne, binaire en général, codera chaque symbole q-aire afin de fournir une protection supplémentaire. Par la suite, le code interne a été remplacé par un code convolutif. Ce schéma concaténé avec un code convolutif interne et un code de Reed-Solomon externe a été standardisé pour les communications spatiales.

Si l'on se place à la sortie du canal, tout se passe comme si la séquence codée provenait du code interne seul. Ce codeur permet une forte résistance au bruit. L'ajout de redondance sur un code en possédant déjà avec le premier codeur offre la possibilité de retrouver l'information là où un seul codeur n'aurait pas suffit pour la retrouver.

FIGURE 1.4 – Schéma de codage concaténé

Comme le montre le schéma de codage d'un code concaténé (cf. 1.4) le décodage s'effectue dans l'ordre inverse du codage. Chacun des codes utilisera un

algorithme ad hoc. La construction est efficace car les codes sont complémentaires. Schématiquement, le code interne va corriger les erreurs isolées et le code externe les rafales. De plus, dans les décodeurs les plus élaborés, le résultat du décodage interne pourra être assorti d'une information de fiabilité qui améliore (marginalement) les performances du décodeur externe (plus le décodeur interne à corrigé d'erreurs, moins l'information fournie est fiable).

1.6.1.6 Codes LDPC

Bien qu'étant une technique ancienne (ils ont été inventés par Gallager en 1962), les codes LDPC (Low Density Parity Check) ,[LC04, Bos99], n'ont été sérieusement envisagés pour les applications en télécommunications qu'à partir de l'avènement du décodage itératif dans les années 90. Ils sont d'ailleurs prévus en option dans les normes IEEE802.11n (WiFi), 802.16e (WiMax), 802.20 (MBWA) et 802.22 (WRAN) ainsi que dans les futures normes de téléphonie 4G.

Les codes LDPC forment une famille de codes linéaires obtenus à partir de matrices de parité creuses. Bien que leur définition ne l'impose nullement, les codes LDPC utilisés (et les seuls que nous considérerons pour le moment) sont binaires.

Les codes LDPC ont par définition une matrice de parité creuse (le poids de chaque ligne vaut quelques unités, alors que la longueur peut atteindre plusieurs milliers). La performance de l'algorithme de décodage tient essentiellement à la faible densité de la matrice.

Les codes LDPC peuvent être obtenus à partir de graphes bipartis creux appelés graphes de Tanner. Soit G un graphe avec n noeuds gauches et r noeuds droits. Ce graphe nous permet de construire un code linéaire binaire de longueur n et de dimension au moins $n-r$ de la manière suivante : les n coordonnées des mots de code sont associées aux n noeuds gauches. Les r noeuds de droite sont associés à des équations de parité. Les mots de code sont alors les vecteurs tels que la somme de tous les noeuds gauches reliés à chacun des noeuds droits vaut zéro. Soit H la matrice binaire $r \times n$ dans laquelle $H[i,j]$ vaut 1 si et seulement si le i-ème noeud droit du graphe G est connecté au j-ème noeud gauche. Alors, le code LDPC associé au graphe G est le code dont une matrice de parité est la matrice H. Lorsque les degrés des noeuds gauches sont identiques à ceux des noeuds droits (i.e. le nombre de 1 dans chaque ligne et chaque colonne de H est constant) on parle de code LDPC régulier. Les codes LDPC réguliers forment une sous famille très étudiée des codes LDPC. Dans ce cas, on parle souvent d'un code LDPC (i,j) : il s'agit d'un code dont la matrice de parité vérifie les conditions suivantes :

- toute colonne de la matrice de parité possède exactement i éléments égaux à 1 ;
- toute ligne de la matrice de parité possède exactement j éléments égaux à 1.

Ce type de code à un rendement $\geq 1 - \frac{i}{j}$. En effet, en comptant les 1 de la matrice de parité, on a

$$in = j(n-k)$$

Exemple : Considérons un code LDPC $(3,6)$. Il s'agit d'un code de rendement $\frac{1}{2}$. Au sein de cette famille, un code de longueur 1000, par exemple, aura

donc 500 équations de parité. Chacun des 1000 bits d'un mot de code devra vérifier 3 équations de parité. Chaque équation de parité fera intervenir exactement 6 bits. La matrice de parité de taille 500×1000 contient donc seulement 3000 coefficients égaux à 1. C'est pour cela qu'on parle de code à matrice de parité creuse.

Gallager a proposé avec les codes LDPC un algorithme de décodage itératif (décrit dans [Clu06, LC04, Wic03]) qui fait l'intérêt de cette famille. Il s'agit d'un algorithme qui propage les probabilités (de valoir 0 par exemple) entre symboles et équations dans le graphe de Tanner code (qui se trouve être précisément le graphe biparti défini plus haut). Si le graphe ne comporte pas de cycles courts, le vecteur de probabilité des symboles converge rapidement vers celui d'un mot de code. L'algorithme est bien adapté au décodage souple, ce qui renforce son intérêt.

La seule donnée nécessaire au décodeur est la matrice de parité creuse. La complexité du décodage est quasiment linéaire (en fixant le nombre d'itérations). Cette faible complexité autorise des blocs de très grande taille (plusieurs dizaines de milliers), ce qui est un facteur supplémentaire de performance et de rapidité de traitement.

1.6.2 Codes convolutifs et Turbo-codes

1.6.2.1 Codes convolutifs

Les codes convolutifs ont été introduits en 1955 par P. Elias comme une alternative aux codes en blocs. Les codes convolutifs traitent l'information non plus par morceaux mais par flux. Ils sont souvent traités comme des blocs linéaires « en flux » sur des corps finis. Nous nous limiterons ici aux cas binaire (le corps à 2 éléments).De nombreux ouvrages traitent des codes convolutifs notamment le livre [Pir88] entièrement consacré aux codes convolutifs et à la manière de les construire. D'autres ouvrages décrivent les codes convolutifs mis en références [LC04, Bos99, Wic03, ML85, Bla03] .

Un codeur convolutif binaire est défini à l'aide de trois paramètres, n le nombre de sorties, k le nombre d'entrées et m la taille de la mémoire. À chaque unité de temps, le codeur lit k bits d'information et produit n bits codés (nous avons $n > k$). La sortie au temps t dépend des entrées aux temps $t, t-1, \ldots, t-m$. Autrement dit le codeur se « souvient » des m derniers blocs d'entrée. D'une certaine manière un codeur en bloc sera un codeur convolutif avec une mémoire nulle.

La définition usuelle d'un codeur convolutif se fait à l'aide de sa description en tant que circuit. Nous parlerons également de codeur convolutif (n, k) lorsque la mémoire n'est pas précisée. Un autre paramètre définissant un codeur est son rendement ou taux, il correspond au rapport entre le nombre de bits en entrée et le nombre de bits en sortie, soit $\frac{k}{n}$.

Définition 1.11 (Première définition d'un codeur convolutif)
Un codeur convolutif (polynomial) binaire (n, k, m) est un circuit linéaire sans rétroaction, d'ordre m, à k entrées et n sorties sur $\mathbb{F}_2$, et invariant dans le temps.

Nous définissons ensuite un code convolutif relativement à un codeur

Définition 1.12 *Un* code convolutif *est l'ensemble des séquences produites par un codeur convolutif.*

$\mathbb{F}_q((x))$ le corps des séries de Laurent sur $\mathbb{F}_q$. Une série de Laurent est définie $\sum_i = r^{\infty} a_i x^i$ avec $r \in \mathbb{Z}$.

Registre à décalage

Un registre à décalage est un outil électronique ou informatique de taille fixe dans lequel les bits sont décalés à chaque coup d'horloge t du système. Chaque registre est formé de bascule qui permettent de stocker un bit $\{0, 1\}$ et qui a chaque temps d'horloge fait sortir l'information qu'elle contient. A chaque temps d'horloge un bit rentre et un bit modifié sort du registre. Au cours du registre à décalage, il est possible d'effectuer une série d'opérations linéaires et de transformations comme un « xor » sur les bits ou toutes autre opérations électroniques, ainsi chaque bit de sortie dépend des m autre bits du registre, on parle de mémoire du registre ou complexité. Certains registres réinsèrent la donnée en sortie dans le registre, on parle alors de registre à décalage avec feedback [2], on les appelle des LFSR [3]

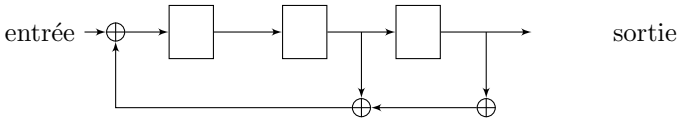


FIGURE 1.5 – Exemple de LFSR

On définit la complexité linéaire d'une suite (finie) comme la longueur minimale d'un LFSR qui génère cette suite.

Le LFSR produit une séquence de sortie $V(x)$ à partir de la séquence d'entrée $U(x) = \sum_{i=0}^{\infty} u_i x^i$ et du polynôme générateur caractérisant le LFSR $P(x) = a_0 + a_1 x + \cdots + a_{m-1} x^{m-1}$ tel que :

$$V(x) = P(x).U(x)$$

Les LFSR sont très utilisé en cryptographie pour les implémentations matérielles de certains algorithmes de chiffrement de flot, ou de fonction aléatoire. Il sont utilisés en communication pour l'implémentation de code convolutifs sous forme systématique (cf 1.6.2.1) lorsque le nombre de sorties du LFSR est supérieure au nombre d'entrées.

Dans un premier temps nous nous intéresserons sur un exemple (figure 1.6) au cas des codes convolutifs de taux $\frac{1}{2}$, qui est aussi le cas le plus fréquent.

2. rétroaction
3. Linear Feedback Shift Register, registre à décalage avec rétroaction

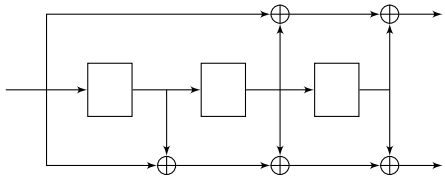


FIGURE 1.6 – Schéma d'un codeur convolutif $(2, 1, 3)$

Le message (train binaire à coder) entre à gauche et circule dans le registre, ici de taille (ou mémoire) 3, de gauche à droite par décalage successif. Le nombre de sorties indique le nombre d'équations linéaires que possède le codeur, elles calculent à chaque unité de temps t les bits de sorties en fonction du registre et de l'entrée (dans le cas ci-dessus le nombre de bits calculé est 2). Dans le registre, la séquence est décalée d'un bit à chaque temps pour le calcul des nouveaux bits de sorties et ainsi de suite. Il y a bien « deux fois plus de bits » après le codage qu'avant.

Il existe plusieurs algorithmes de décodage pour les codes convolutifs, l'algorithme utilisé en pratique est l'algorithme de Viterbi qui est un algorithme de décodage au maximum de vraisemblance sur tout canal sans mémoire (les algorithmes de décodages des codes convolutifs sont décrits [JZ99, Pir88, LC04, Bos99])

Cet algorithme permet à partir de n'importe quelle séquence binaire (tronquée) de déterminer la séquence d'information produisant la séquence codée qui lui est la plus probable. Cette propriété d'optimalité est vraie même lorsque la séquence fournie au codeur est accompagnée d'une information de fiabilité (décodage souple). En particulier, pour le décodage d'un code convolutif poinçonné, les symboles décimés sont réintroduits dans le décodeur, avec une valeur quelconque et une fiabilité nulle. Le décodeur prendra automatiquement en compte, et de façon optimale, cette information.

La complexité par bit d'information du décodeur de Viterbi est proportionnelle à 2^{k+m}. Ce caractère exponentiel est la raison principale pour laquelle les codes retenus pour les applications ont souvent une seule entrée (k=1) et une longueur de contrainte faible ($m < 10$).

L'algorithme de décodage de Viterbi consiste à trouver le chemin minimum dans un treillis de codage (graphe orienté sans cycle).

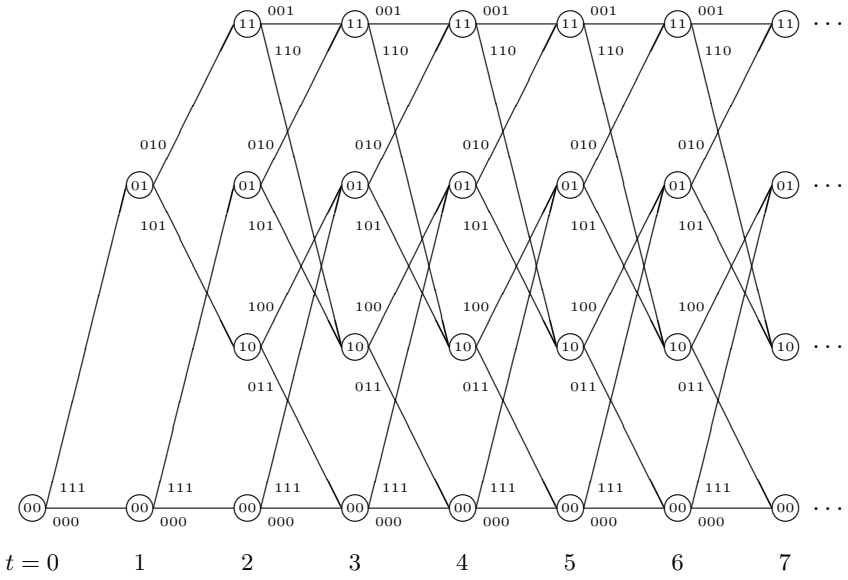


FIGURE 1.7 – Treillis de codage d'un code (3,1,2)

Chaque sommet correspond à un état du codeur, chaque arête correspond à une transition et a pour étiquette la sortie correspondante du codeur. Tout chemin correspond à une séquence codée, obtenue en concaténant les étiquettes.

Nous parcourons le graphe de gauche à droite à partir de la séquence d'initialisation 00. Les poids de chaque branche sont représentés par la distance de Hamming entre le mot de la branche et le mot de code reçu. Le mot retenu correspond à la branche choisie, il permet de choisir le mot de code le plus proche du mot reçu. Plus la distance du code est grande plus la différenciation des mots sera efficace. Par exemple nous pouvons voir le chemin résultant pour un message reçu donné Figure 1.8.

Supposons la séquence émise :

111 010 001 110 100 101 011

La séquence reçue est :

111 110001 110 111 101 011

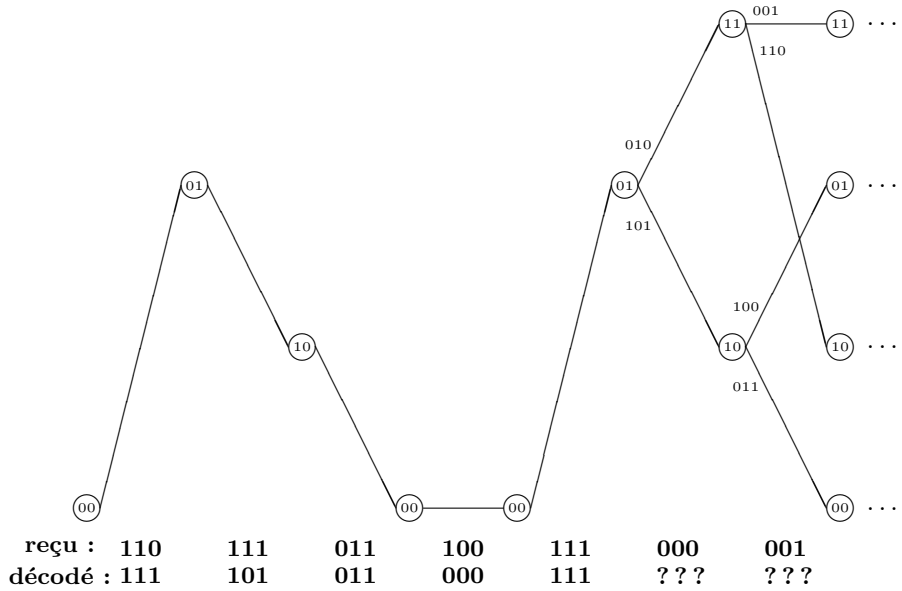


FIGURE 1.8 – Mot codé

Plusieurs chemins peuvent être mémorisés par l'algorithme au cours du décodage. Si, pour un instant de décodage, un chemin possède une métrique inférieure aux autres alors il sera considéré comme le meilleur chemin. Cependant, il est probable que plusieurs chemins soient minimaux (cf. figure 1.8), dans cette circonstance l'état final du codeur permet de choisir le chemin. En effet on peut supposer que l'état initial et final du codeur sont à zéro. Nous supposons, dans le cas d'une interception, que nous ne connaissons pas l'état initial et final du codeur. Le chemin choisi, s'il en demeure plusieurs à la fin de notre algorithme, sera choisit de manière aléatoire.

Cette méthode « hard input hard output (HIHO)» ne permet pas de gérer les effacements qui apparaissent dans un code poinçonné. La métrique utilisée permet fortement le passage à l'étape d'après et ne convient plus si nous introduisons des symboles « inconnus » ou effacements. Afin de palier à ce problème sans changer notre algorithme, nous introduisons des probabilités et travaillons de façon « SIHO (soft input hard output ». A chaque état nous calculons le bit le plus vraisemblable, c'est à dira la probabilité qu'un bit vaille 1 ou 0 par aller-retour du graphe. A l'inverse de la méthode « HIHO » qui calcul le mot le plus vraisemblable par simple aller. Ces techniques de Viterbi sont rapides mais ont une convergence légèrement moins bonne que pour un algorithme Viterbi de « forward backward ». Nous n'effectuons plus une mais deux passes sur le treillis qui ont pour but d'augmenter la fiabilité du chemin choisi. Cet algorithme fonctionne en « SISO (soft in soft out)». Nous ne cherchons plus le « meilleur » chemin mais à optimiser la meilleure probabilité pour tout t (cf. Figure 5) de façon locale. Si une erreur se produit à un niveau elle ne se répercute pas ailleurs offrant un gain de convergence, ce qui n'était pas le cas précédemment où une erreur se répercutait sur plusieurs noeuds du treillis. Cependant nous sommes obligés dans ce cas , une fois le chemin parcouru de gauche à droite, d'effectuer la même opération de droite à gauche du treillis. La moyenne des deux

passes nous permet de retenir le chemin optimum. En pratique, cette dernière méthode est souvent retenue car elle offre une bonne efficacité aussi bien pour un code convolutif normal et poinçonné. Néanmoins la première méthode reste très efficace est suffisante pour décoder un code convolutif.

1.6.2.2 Turbo-codes

Les turbo-codes (cf. [BGT93, LC04, Wic03, Bla03]) sont nés au sein de l'école nationale des télécommunications de Bretagne (ENST Bretagne), en 1993. En raison de ses performances, le turbo code a été adopté par plusieurs organismes pour être intégré dans leurs standards. C'est ainsi que la NASA a décidé d'utiliser les turbo codes pour toutes ses sondes spatiales construites à partir de 2003. De son côté, l'agence spatiale européenne (ESA) a été la première agence spatiale à utiliser le turbo code, avec sa sonde lunaire smart 1.

Plus près de nous, le turbo code est utilisé dans la norme UMTS (téléphonie 3G) et l'ADSL2.

Plusieurs types de turbo-codes ont été inventés. Le plus utilisé (cf. 1.9) est le codeur en parallèle constitué de codeurs convolutifs. Une variante est le turbo codeur duo-binaire, son utilité est liée à l'architecture matérielle. Deux bits sont traités à la fois générant 6 bits de sorties (moins si l'on poinçonne). L'intérêt d'une telle structure repose sur le décodage qui traite 2 bits simultanément, cependant des schémas de 3 bits n'ont pas d'utilité car ils n'apportent que peu de performance par rapport à la complexité. On peut construire tout type de turbo-code, notamment en changeant les codeurs le caractérisant. Par exemple en remplaçant les codeurs convolutifs par des codeurs de Hamming ou des codes BCH. Les codes BCH sont utilisés sous forme de code produit, le premier crée les lignes, le second les colonnes. Ces codeurs existent à petites échelles et possèdent de bonnes propriétés. Une autre façon de voir les turbo-codes est de les représenter sous forme ligne (série) et non parallèle (cf. figure 1.10).

Un turbo codeur classique résulte de l'association de deux codeurs. Il s'agit souvent de codeurs convolutifs systématiques de taux $\frac{1}{2}$ (noté CCS) en parallèle. Comme le montre la figure 1.9, le turbo codeur produira typiquement trois sorties à envoyer sur le canal de transmission (après modulation éventuelle) :

- u_0 un bloc d'information binaire qui est également l'entrée
- u_1 qui a été modifié par le codeur convolutif
- u_2 qui est précédé d'une permutation alpha.

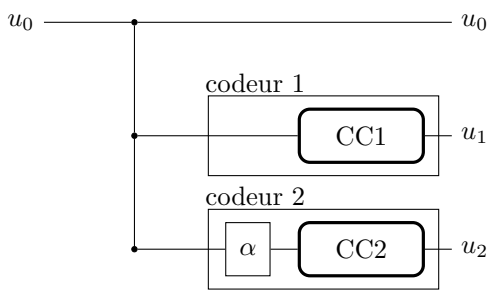


FIGURE 1.9 – Schéma d'un turbo-code parallèle

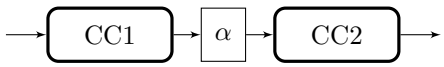


FIGURE 1.10 – Schéma d'un turbo-code série

Ce codage permet donc de répartir l'information apportée par un bit de la trame u sur ses voisins (avec le codage de parité 1) et même sur toute la longueur de la trame transmise (avec le codage de parité 2). Ainsi, si une partie du message est fortement abîmée pendant la transmission, l'information peut encore se retrouver ailleurs. Au total, ce codeur a un taux de transmission de $\frac{1}{3}$. Le bloc (u_0, u_1, u_2) de taille $3T$ constitue le mot de code qui sera transmis à travers le canal de communication. D'autres taux sont évidemment possibles (avec des codes convolutifs poinçonnés par exemple).

Les turbo-codes sont des codes en bloc, mais ses mots de codes sont constitués de séquences provenant de codeurs convolutifs. Les codes convolutifs utilisés auront généralement une faible longueur de contrainte (3 ou 4) mais la taille des blocs T sera grande (plusieurs milliers de bits). La permutation n'est pas choisie aléatoirement, car elle a un impact important sur les performances. En revanche, les ≪ bonnes ≫ permutations (celles qui sont utilisées) sont souvent déterminées par tâtonnement (elles évitent les cycles de petite taille dans le graphe de Tanner associé au code), et n'ont à priori pas de propriétés particulières qui permettraient de les retrouver plus facilement. Nous les considérerons comme aléatoires.

Bien que le codage (cf. [Wic03, Bla03]) soit relativement simple car composé de seulement deux codeurs convolutifs et d'un entrelaceur, le décodage est plus complexe.

Le principe de décodage est présenté sur la Figure 1.11. Les permutations utilisées sont les mêmes que celles du codeur. Pour améliorer les performances la séquence u0 reçue et entrelacée est prise en compte dans le décodeur 2 et dans l'organe de décision. De plus un rebouclage est fait de la sortie du décodeur 2, vers l'entrée du décodeur 1. Il faut noter que pendant la phase de décodage, le décodeur connaît les probabilités calculées à l'étape précédente. L'algorithme de décodage est itératif à la manière des LDPC.

Le nom donné aux turbo-codes vient du rebouclage qui permet d'augmenter les performances globales du décodeur et qui fait penser au rebouclage des gaz d'échappement utilisé dans les moteurs ≪ turbo ≫ afin d'accroître les performances.

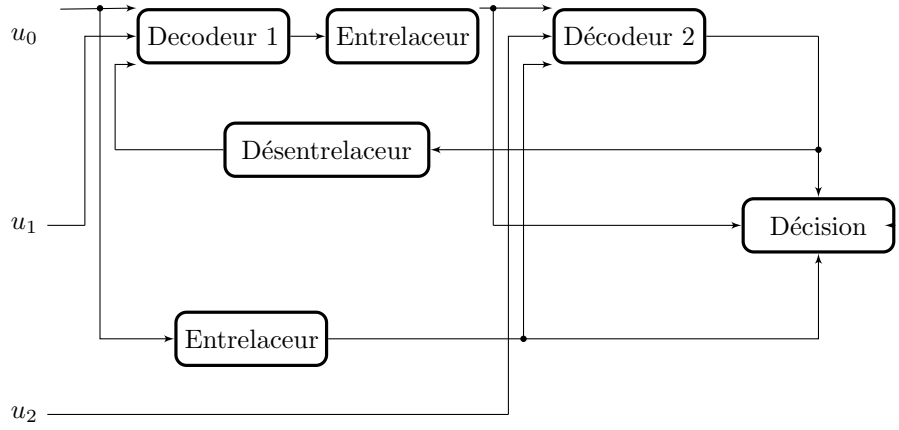


FIGURE 1.11 – Schéma d'un turbo décodeur

Chapitre 2

Reconnaissance des familles de codes

Dans ce chapitre, nous dressons un état de l'art des outils existants pour la reconnaissance de codes linéaires. Ces outils utilisent en général des particularités de la famille à laquelle appartient le code que l'on cherche à reconnaître (code linéaire, code convolutif, code LDPC, etc ...).

2.1 La problématique

Notre problème s'inscrit dans le cadre de la rétro-ingénierie d'une chaîne de transmission. Pour une personne qui intercepte un signal de communication, ce problème consiste à retrouver la chaîne de transmission (codes correcteurs, modulation choisie, brasseur utilisé, etc...) à partir de l'unique connaissance de la séquence interceptée. Nous allons ici nous intéresser à la reconnaissance du code correcteur utilisé.

Pour notre étude, nous nous plaçons dans le cas des codes binaires. Nous possédons une séquence interceptée $\hat{x}_1, \ldots, \hat{x}_T$, que nous noterons $\hat{x}$ de longueur T à la suite de la reconnaissance de la démodulation, où $\hat{x}_i$ représente la probabilité que le symbole observé soit $\ll\ 1 \gg$.

Nous produisons à partir de cette suite $\hat{x}$ la suite binaire $y_1, \ldots, y_T$ au moyen de la règle suivante :

$$0,5 < \hat{x}_i < 1, \text{ alors } y_i = 1$$

$$0 \leq \hat{x}_i \leq 0,5, \text{ alors } y_i = 0$$

La suite binaire observée est définie par :

$$Y = y_1, \ldots, y_T \ \in \mathbb{F}_2^T$$

FIGURE 2.1 – Schéma d'interception d'une transmission électronique

Dans notre séquence interceptée, nous cherchons à identifier un mot de code ou une séquence de mots suivant le type de code utilisé. On différencie les codes selon deux catégories, les codes en blocs et les codes convolutifs ou en flux dans lesquels l'information sur le code n'est pas agencée de la même façon.

- Les codes en blocs :
  Dans le cas des codes en blocs, les mots de longueur n sont concaténés pour former la séquence codée. Le premier et le dernier mot de notre séquence interceptée peuvent être tronqués. Ces mots de code tronqués n'apportent qu'une partie de l'information nécessaire à la reconnaissance du code. Il est donc essentiel d'identifier de manière exacte le début de ces mots. Soit $Y = y_1, \ldots, y_T \ \in \mathbb{F}_2^T$ la suite interceptée.
  Les mots de code bruités sont définis par :

$$\tilde{m}_i = y_{s+in}, \ldots, y_{s+(i+1)n-1}, \text{ avec } s + (i+1)n - 1 < T$$

  où s est l'indice de synchronisation et n la longueur d'un mot de code. On appelle m_i le mot de code non bruité dont m_i est issu.
- les codes convolutifs :
  Dans ce cas spécifique, les mots de code ne sont plus concaténés, on ne parle pas de mots de code tronqués en début et fin de séquence. La synchronisation prend un sens légèrement différent. Pour les codes convolutifs (n, k), k bits en entrée du registre à décalage utilisé pour le codage forment n bits en sortie. Ces n bits ne forment pas un mot de code mais un paquet de bits provenant tous des mêmes k bits de départ. On appel $\ll$ synchronisation $\gg$ le début d'un paquet de n bits.
  Cette structure caractéristique des codes convolutifs sera étudiée de manière approfondie dans la partie 3.

Si pour une bonne hypothèse de synchronisation et de longueur, nous sommes capables d'identifier l'utilisation d'un code linéaire, par recherche exhaustive de la synchronisation ($\in [1, n]$) et la longueur ($n \in [1, n_{max}]$) nous serons capables d'identifier à la fois la synchronisation, la reconnaissance de la longueur et d'identifier le code.

En effet, sans connaissance sur l'information émise nous ne pouvons pas identifier le codeur à partir de son code. Nous verrons par la suite (cf. 3) la cause de cette indétermination pour les codes convolutifs.

Bien que ce sujet constitue une problématique en télécommunications qui se pose naturellement dans un contexte militaire, peu d'études ont été publiées à ce sujet. La littérature commence à apparaître et débute avec les travaux de B. Rice [Ric95] sur la mise en équation des codes convolutifs qui ont permis de poser les bases dans ce domaine. E. Filiol [Fil97a, Fil97b, Fil00] en s'inspirant de ces travaux à créé une méthode de reconnaissance des codes convolutifs $(2, 1)$, il a étendu cette recherche en décrivant les outils nécessaires pour les codes (n, k). Ces travaux ont été poursuivis par J. Barbier [Bar07] qui a permis de donner un algorithme de reconnaissance des codes (n, k) dans un milieu bruité. Il a aussi abordé le problème des turbo-codes parallèles. Pour les codes en blocs, les travaux initiaux sont dus à G. Planquette [Pla96]. Plus tard arrivent ceux de A. Valembois [Val00, Val01] qui décrit un algorithme, robuste au bruit, de reconstruction du code dual. Ses travaux ont été poursuivis et sa méthode améliorée par M. Cluzeau [Clu06] en établissant une probabilité pour les équations de parité identifiées.

2.2 Techniques générales

Dans cette section, nous rappelons les principales méthodes de reconnaissance de codes. Afin de faciliter leur compréhension nous pouvons supposer que certains paramètres du code ont pu être reconnu lors de la démodulation (synchronisation, taille des mots de codes).

2.2.1 Probabilité des équations de parité

Nous notons N le nombre de mots reçus. Nous considérons la matrice M de taille $N \times n$ de mots de code non bruités

$$M = \begin{pmatrix} m_1 \\ m_2 \\ \vdots \\ m_N \end{pmatrix}$$

et la matrice $\tilde{M}$ de taille $N \times n$ de mots de code bruités

$$\tilde{M} = \begin{pmatrix} \tilde{m}_1 \\ \tilde{m}_2 \\ \vdots \\ \tilde{m}_N \end{pmatrix}$$

La reconstruction s'appuie sur la constatation que pour tout mot h dans le dual du code que nous cherchons à reconnaître :

$$hM^T = 0$$

Dans un contexte bruité, cette équation n'est plus vraie. Nous avons cependant le lemme suivant :

Lemme 2.1 *Soit m un mot binaire et soit $\tilde{m}$ un mot bruité correspondant au mot m après transmission à travers un canal binaire symétrique de probabilité d'erreur p, alors*

$$P(h\tilde{m}^T = hm^T) \neq \frac{1 - (1-2p)^{|h|_H}}{2}$$

Par la suite nous noterons $q_h \stackrel{\text{def}}{=} \frac{1-(1-2p)^{|h|_H}}{2}$.

La preuve peut être trouvée dans la thèse de A. Valembois [Val00] ou celle de M. Cluzeau [Clu06]. Nous pouvons ainsi en déduire un distingueur des mots appartenant au dual. Dans le cas d'un vecteur h n'appartenant pas au dual le produit hm_i^T vaut 0 ou 1 avec la même probabilité. Par conséquence l'espérance du poids de $h\tilde{M}^T$ vaudra :

$$\mathbb{E}(|(h\tilde{M}^T)|_H) = \frac{N}{2}$$

Alors que si h appartient au code dual nous aurons :

$$\mathbb{E}(|(h\tilde{M}^T)|_H) = q_h N$$

puisque $hM^T = 0$.

Il nous est alors facile de distinguer un vecteur appartenant au dual en calculant le poids de Hamming $|h\tilde{M}^T|_H$.

Approche :
Nous considérons la matrice $\tilde{M}^T$ de mots de code bruités et nous recherchons des mots de poids faibles tel que $h\tilde{M}^T$ soit inférieur à un seuil fixé. Si h vérifie cette équation alors nous le considérons comme un élément du code dual.

2.2.2 Méthode de Cluzeau-Finiasz

Cette méthode provient de leur travaux de recherche des mots de poids faible pour les codes LDPC. Quand on cherche un mot du dual de poids $2P$, nous devons donc trouver une combinaison linéaire de $2P$ lignes telle que le produit $h\tilde{M}^T$ soit de petit poids. La recherche exhaustive de $2P$ lignes parmi n peut s'avérer coûteuse en termes de temps de calcul. Une des idées de cette méthode réside dans la simplification de cette recherche. Faisons l'hypothèse que le vecteur du dual h soit un vecteur de poids $2P$ dont la première moitié h_1 du vecteur possède un poids P et la seconde moitié h_2 aussi. La matrice $\tilde{M}^T$ est séparée dans sa longueur en deux parties respectivement de taille $\lceil \frac{n}{2} \rceil$ et $\lfloor \frac{n}{2} \rfloor$. Nous choisissons ensuite une plage de L bits, telle que $2^L \approx \binom{\frac{n}{2}}{P}$, sur laquelle nous voulons que les combinaisons linéaires des $2P$ lignes s'annulent et notons R_1 et R_2 les deux parties de la matrice $\tilde{M}^T$ de longueur L, voir figure 2.2. Ceci implique donc que l'on veuille que $h_1R_1 = h_2R_2$. La forme particulière choisie pour le vecteur h implique une diminution du nombre de mots de poids faible trouvés, cependant cela permet typiquement d'identifier qu'une fonction constante de mots du dual de poids P suffisant pour reconstruire entièrement la matrice génératrice du code dual. L'intérêt de cette méthode réside dans l'amélioration de la complexité algorithmique. En effet, dans le cas de la recherche exhaustive, nous recherchons une combinaison de poids $2P$ parmi n éléments. Dans notre cas, nous avons une complexité en temps et en mémoire de l'ordre de $\mathcal{O}\binom{\frac{N}{2}}{p}$ en utilisant une table de hachage. Nous stockons en effet tous les candidats pour h_1 dans une table de hachage (en mettant h_1 à l'adresse h_1R_1), et nous testons tous les candidats h_2 en regardant si des éléments h_1 sont stockés à l'adresse h_2R_2.Nous choisissons L de manière à ce qu'une entrée de la table de hachage contient en moyenne un nombre constant de candidat.

Théoriquement un vecteur de poids faible possède un grand nombre de positions nulles (en fonction de la taille du vecteur et du taux d'erreur du canal). Il est possible qu'aucune combinaison ne s'annule sur la fenêtre de L bits choisie, auquel cas nous recommençons avec une nouvelle fenêtre différent de la précédente.

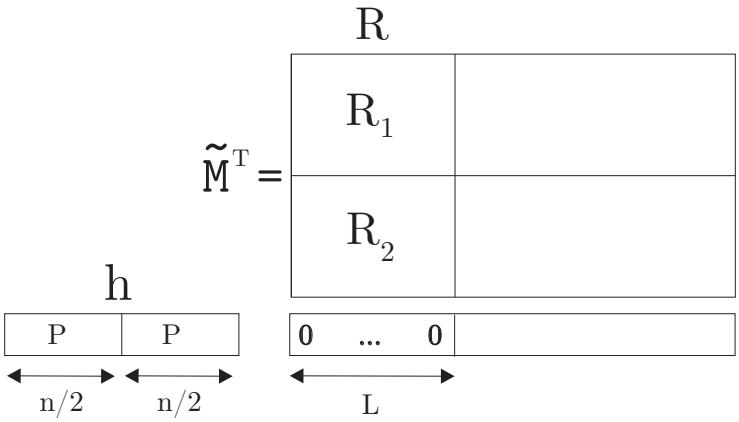


FIGURE 2.2 – Méthode Cluzeau-Finiasz de recherche de mots de poids faible.

La figure ci-dessus (cf figure 2.2) illustre l'algorithme de Cluzeau-Finiasz. Nous pouvons le décrire selon les étapes suivantes :

Algorithm 1 ALGORITHME DE CLUZEAU-FINIASZ

Entrée : Une matrice $\tilde{M}$ de taille $n \times N$ avec $N > n$.
Un seuil *Seuil* de fausse alarme
Sortie : Un vecteur h appartenant au code dual.

1: Générer tous les h_1 pour $|h_1|_H = P$.
2: Stocker h_1 à l'adresse $h_1 R_1$
3: Générer tous les h_2 pour $|h_2|_H = P$.
4: Pour tout h_2 de poids P
5: **if** Il y a une entrée h_1 à l'adresse $h_2 R_2$ **then**
6: On calcule $(h_1||h_2)\tilde{M}^T$.
7: **if** $|(h_1||h_2)\tilde{M}^T|_H < Seuil$ **then**
8: on conserve le mot dans la table de hachage.
9: **end if**
10: **end if**

Nous allons étudier une autre méthode de recherche de mots de poids faible (méthode de Canteaut-Chabaud) et nous verrons que les méthodes ont de nombreuses similitudes.

2.2.3 Recherche de mots de petit poids

La recherche de mots de poids faible fût principalement utilisée comme algorithme pour attaquer le cryptosystème de R. McEliece basé sur les codes. Cependant décoder un code au plus proche voisin est conceptuellement proche de rechercher des mots de poids faible. Plusieurs méthodes existent, le premier algorithme de R. McEliece en 1978 (cf. [McE78]) fut amélioré par la suite par P. Lee et E. Brickell en 1988 (cf. [LB88]). De nouveaux algorithmes sont apparus par J. Leon en 1988 (cf. [Leo88]) et J. Stern en 1989 (cf. [Ste89]). Ces différentes méthodes sont analysées et comparées dans la thèse de A. Canteaut

(cf. [Can96]), elle améliore l'algorithme de J. Stern qui offrait les meilleures performances avec la participation de F. Chabaud. La méthode la plus récente de recherche de mots de poids faible qui s'impose en tant que référence est décrite par Bernstein [BLP08].

Méthode de Stern

Nous présentons cet algorithme. Pour cela considérons la matrice I sous la forme systématique

$$G = \left(\ I_n \ \middle| \ R \ \right)$$

Dans un contexte de cryptographie nous nous intéressons aux matrices génératrices et nous présenterons notre algorithme comme tel. La matrice G représente dans le cas de la reconnaissance de code la matrice $\tilde{M}^T$ mise sous forme systématique (cf figure 2.3).

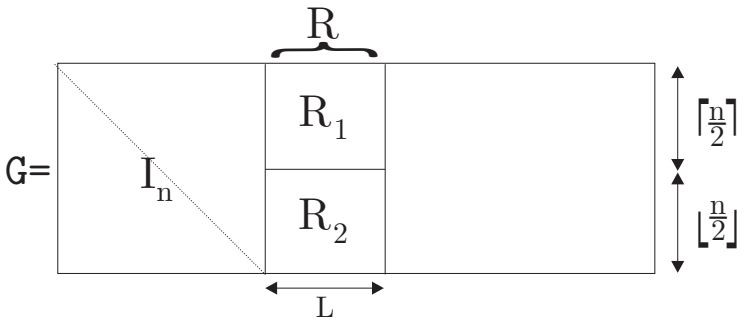


FIGURE 2.3 – Passage de la matrice $\tilde{M}$ en la matrice G

La matrice R est séparée en deux parties R_1 et R_2 (cf figure 2.4),respectivement de taille $\lceil \frac{n}{2} \rceil \times L$ et $\lfloor \frac{n}{2} \rfloor \times L$. On définit un sous ensemble de R de longueur L. Nous souhaitons rechercher des mots h tels que hG soit de poids faible. Pour cela nous découpons h en deux parties h_1 et h_2 de longueur $\lceil \frac{n}{2} \rceil$ et $\lfloor \frac{n}{2} \rfloor$ respectivement (ici $h = (h_1||h_2)$) et cherchons de tels vecteurs h de la forme suivante :

$$|h_1|_H = |h_2|_H = P \text{ et } h_1 R_1 = h_2 R_2 \text{ où } P \text{ est une certaine constante} \quad (2.1)$$

Si w est le poids de hG alors hG à la forme donnée par la figure 2.4.

L'élément novateur par rapport à la méthode précédente est l'utilisation de la forme systématique de la matrice que nous ne prenions pas en compte précédemment. L'ajout de cette information influe directement sur le vecteur de poids faible recherché. Soit le vecteur h est de petit poids, soit la partie des $N - n - L$ derniers bits de R est de petit poids, sinon il faut chercher un compromis entre les deux pour obtenir le meilleur résultat possible.

Les mots formés ainsi sont de bons candidats pour être des mots de poids faible et sont facilement constructibles. Ainsi, pour tout mot de code vérifiant l'équation (2.1), leur construction s'effectue de la manière suivante :

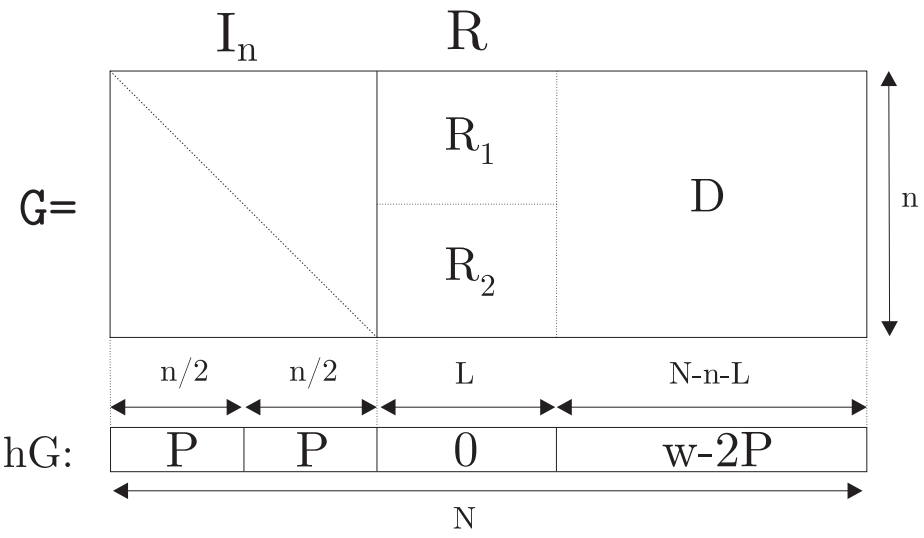


FIGURE 2.4 – Recherche de mots de poids faible pour l'algorithme de Stern

Algorithm 2 ALGORITHME DE STERN

Entrée : Une matrice $\tilde{M}$ de taille $n \times \sigma$ avec $\sigma > n$.
Un seuil *Seuil* de fausse alarme
Sortie : Un vecteur h appartenant au code dual.

1: Mise de la matrice sous forme systématique
2: Générer tous les h_1 pour $|h_1|_H = P$.
3: Stocker h_1 à l'adresse $h_1 R_1$.
4: Générer tous les h_2 pour $|h_2|_H = P$.
5: Pour tout h_2 de poids P
6: **if** il y a une entrée $h_1 R_1$ à l'adresse $h_2 R_2$ **then**
7: On calcule $(h_1||h_2)D$.
8: **if** $|(h_1||h_2)D|_H < Seuil$ **then**
9: on conserve le mot dans la table de hachage.
10: **end if**
11: **end if**

L'article de F. Chabaud ([Cha92]) renseigne sur la meilleure valeur de P à utiliser selon la mémoire ($P = 2$ ou 3), et la longueur L est choisie de manière à ce que $2^L \approx \binom{\frac{n}{2}}{P}$. On a donc $L = \mathcal{O}(\log_2 n)$

A chaque itération l'algorithme renvoie un mot de petit poids de hG s'il existe, sinon nous changeons complètement l'ensemble d'information pour recommencer. A chaque étape nous réitérons un pivot de Gauss complet sur I_n, faisant croître très vite la complexité.

Méthode itérative (Canteaut-Chabaud)
La méthode de Canteaut-Chabaud est une amélioration de la méthode de Stern. Cette méthode permet de minimiser l'effet d'un pivot de Gauss coûteux à chaque itération de l'algorithme.

Définition 2.2 ***(Ensembles d'informations voisins)*** *Deux ensembles d'informations I et I' sont voisins si et seulement si*

$$\forall \alpha \in I, \forall \beta \in \{1, \ldots, n\} \setminus I, \text{ tels que } I' = (I \setminus \{\alpha\}) \cup \{\beta\})$$

Pour que cette définition soit vraie les variables α et β doivent vérifier certaines conditions.

Proposition 2.3 *Soit I un ensemble d'information pour le code C et soit $G_{syst} = (I_n, R)$ la matrice systématique associée à la matrice génératrice G. Soit $\alpha \in I$ et $\beta \in J$ où J est le complémentaire de I. Alors, l'ensemble $I' = (I \setminus \{\alpha\}) \cup \{\beta\}$ est un ensemble d'information pour le code C si et seulement si le coefficient de la ligne α et de la colonne β, $p_{\alpha,\beta}$ est égale à 1.*

La preuve peut facilement être comprise en considérant les dépendances linéaires entre les colonnes de G

$$\forall j \in J, \; G_j = \sum_{i \in I} r_{i,j} G_i$$

il est ainsi facile de déterminer le lien entre les colonnes α et β de la matrice G. L'équation suivante est donc vérifiée si et seulement si la condition $p_{\alpha,\beta} = 1$ est vérifiée.

$$G_\beta = r_{\alpha,\beta} + \sum_{i \in I \setminus \{\beta\}} R_{i,\beta} G_i$$

Deux matrices G_{sys} et G'_{sys} d'un même code C ont un ensemble d'information différent mais voisin simplement en changeant une colonne de l'ensemble d'information de G avec sa partie redondance (cf 2.5). Dans le cadre de la recherche de mots de poids faible, cette méthode permet à chaque itération de fournir une dont l'information est différente mais voisine de celle précédente tout en ayant seulement à réaliser un pivot seulement sur quelques valeurs de la colonne $\beta \in I$. Pour cela, nous ajoutons la ligne d'indice α à toutes les lignes possédant un $\ll 1 \gg$ sur la colonne β (privé de l'indice α). Nous économisons un pivot de Gauss complet coûteux pour de grandes tailles de matrices. Les variables α et β sont choisies de manière aléatoire tout en vérifiant la condition $r_{\alpha,\beta} = 1$.

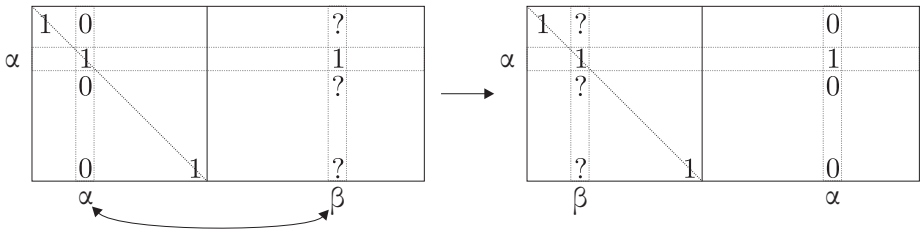


FIGURE 2.5 – Remise à jour de la matrice systématique pour des ensembles d'informations voisins

2.2.3.1 Techniques d'amélioration

Nous avons vu précédemment que l'utilisation de la forme systématique permettait d'améliorer les performances de l'algorithme, en contrepartie on ajoute un pivot de Gauss en début d'algorithme et un pivot « rapide » à chaque étape. Nous voulons tirer partie des avantages des deux méthodes en limitant les effets négatifs qui augmentent le temps de calcul. En effet, la méthode de Cluzeau-Finiasz est meilleure que la méthode de Canteaut-Chabaud lorsque l'on cherche des mots h de poids faible. Cette seconde méthode ne prend pas en compte le poids de h. Par ailleurs, le nombre de mots de codes de $\tilde{C}$ de poids faible recherchés ont typiquement un poids de nq_H quand on se restreint à la fenêtre d'information. La méthode de Canteaut-Chabaud va les retrouver lorsque $nq_H \leq 2P$.Lorsque n est grand, la méthode de Canteaut-Chabaud n'est plus efficace alors que celle de Cluzeau-Finiasz (cf. [CF09b]) le reste toujours. Cependant, ces deux algorithmes qui paraissent distincts peuvent être considérés comme deux instances d'un même algorithme. La méthode que l'on propose consiste à ne plus considérer la matrice $\tilde{M}^T$ mais la matrice R comme la concaténation de la matrice $\tilde{M}^T$ et de la matrice identité (cf figure 2.6) qui est :

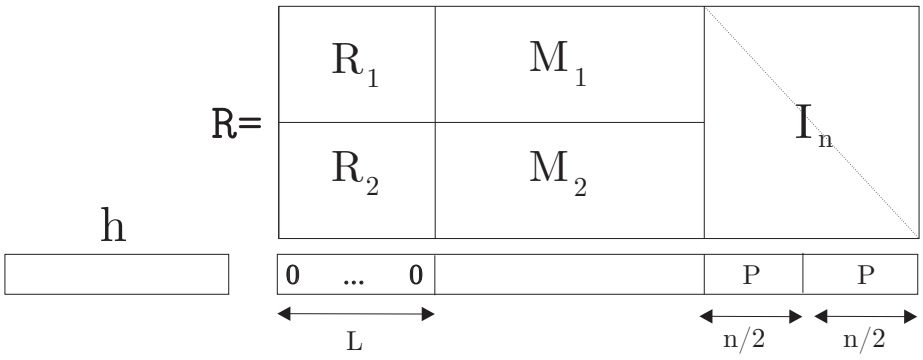


FIGURE 2.6 – Méthode d'amélioration possible de recherche de poids faible

Sur cette partie identité nous voulons comme pour l'algorithme de Stern obtenir un poids P sur la première partie et un poids P sur la seconde moitié. Cette étape est inéluctable à la simplification de la recherche. Toutefois, l'étape de Gauss n'est plus nécessaire ni la permutation de colonne dans le but d'obtenir une information voisine. Privé de ces étapes, l'algorithme reste le même que celui de Stern. L'ajout de la partie information permet de contrôler le poids du mot h et le poids de hR restreint aux colonnes de M_1 (ou de M_2). La recherche de mots de poids faible est un ingrédient important pour la reconstruction d'une base du dual. Nous allons voir dans la partie suivante l'utilisation de ces méthodes dans un contexte de reconnaissance de code.

2.2.4 Reconstruction du code dual

2.2.4.1 Méthode de Gauss

Cette méthode s'inspire d'une opération naturelle que l'on pourrait effectuer pour reconnaître un code linéaire dans le cas sans erreur.

Il n'est pas à écarter la possibilité de trouver une séquence non bruitée de longueur nécessaire à l'accomplissement de nos méthodes sans bruit dans une séquence interceptée bruitée, selon la probabilité d'erreur p du canal. La recherche d'une suite de bits sans bruit et l'exécution de la méthode dans ce cas est alors moins complexe et plus rapide qu'une méthode de traitement spécifique du bruit.

La méthode de Gauss est très utilisée en algèbre linéaire pour résoudre un système d'équations, calculer le rang d'une matrice ou pour calculer l'inverse d'une matrice carrée inversible.
Rappelons que le principe de fonctionnement de l'algorithme consiste à exprimer la première inconnue de la matrice en fonction des autres et de l'éliminer dans toutes les équations suivantes. L'opération est réitérée sur la sous-matrice jusqu'à obtention d'une sous-matrice de taille 1×1. La complexité de l'algorithme pour rendre cette matrice triangulaire supérieur est de $0(n^3)$, elle peut être améliorée en utilisant un algorithme de Strassen dont la complexité est $\approx 0(n^{2,807})$.

Que cela soit pour les codes en blocs ou les codes convolutifs nous pouvons appliquer la méthode de Gauss. L'utilisation diverge selon le type de code utilisé. Pour les codes convolutifs nous cherchons à partir d'une matrice M de mots de code non bruités m_i le code dual de notre code. Pour obtenir la matrice génératrice nous devons calculer le dual du code dual ainsi il est nécessaire d'effectuer deux fois l'algorithme de Gauss. Les codes en blocs permettent directement de reconstruire une matrice génératrice sans passer par le code dual.

Il est essentiel que l'entrée de l'algorithme soit la matrice M de mots de code non bruités. Cela suppose connu la synchronisation s et la taille de bloc n. La complexité de l'algorithme pour une matrice A de taille $a \times b$ est $0(\max(a,b)^3)$. Dans un contexte de rétro-ingénierie ces données sont inconnues de l'intercepteur mais peuvent être retrouvés par un critère du rang.

Si la synchronisation n'est pas connue, on peut néanmoins adapter la méthode en testant différentes synchronisations. Soit s_e la valeur de synchronisation testée. Si $s_e \neq s$ alors la matrice $M_{(n,s_e)}$ formée contiendra des mots de code répartis sur deux lignes.

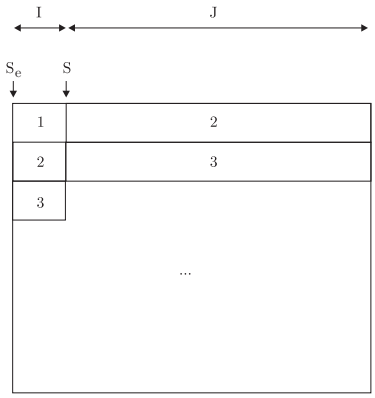


FIGURE 2.7 – Système crée avec la bonne taille de bloc et une mauvaise synchronisation

Chaque mot de la matrice 2.7 est coupé en deux parties du à l'erreur de synchronisation. Les colonnes de la matrice comprises entre s_e et s (ensemble des colonnes I) et celles sur la deuxième partie de la matrice (ensemble des colonnes J) ne sont plus liées, nous perdons donc les relations de parité faisant intervenir les deux blocs disjoints d'un même mot et seule les relations de parité internes à chacun de ces blocs distincts sont vérifiées. La désynchronisation a pour effet de diminuer la taille des relations de parité observables. Seul les relations de longueur $n - s_e < n$ sont identifiables par la méthode de Gauss. Cette conséquence affecte le rang de la matrice $M_{(n,s_e)}$. En considérant le rang normalisé nous avons les relations suivantes :

$$\frac{\mathrm{rg}(M_{n_e,s_e})}{n_e} \approx \frac{k}{n}$$

quand $n_e = n$ et $s_e = s$ sinon (avec forte probabilité)

$$\frac{\mathrm{rg}(M_{n_e,s_e})}{n_e} > \frac{k}{n}$$

Ceci fournit un critère simple permettant de retrouver la bonne synchronisation.

2.2.4.2 La méthode de Valembois

Ce que nous appelons méthode de Valembois consiste à utiliser l'algorithme de Canteaut-Chabaud pour trouver des h tels que $h\tilde{M}^T$ soit de poids faible et enfin proposer un seuil T pour lequel à $|h\tilde{M}^T|_H \leq T$ alors h a de bonnes chances d'appartenir au dual, alors que si $|h\tilde{M}^T|_H > T$ ce dernier n'a que peu de chance s d'appartenir au dual.

Test statistique

On recherche T et N tel que $|hR^T|_H \leq T$

- si le test est vrai alors on décide que $h \in C^\perp$
- si le test est faux alors on décide que $h \notin C^\perp$

Nous définissons la probabilité de non détection comme la probabilité qu'un mot h dans le dual ne soit pas trouvé définit par :

$$P(|h\tilde{M}^T|_H \geq T | h \in C^\perp)$$

La probabilité de fausse alarme est définie par la probabilité de considérer que h appartient au dual alors qu'il n'y appartient pas. Elles est définie par :

$$P(|h\tilde{M}^T|_H < T | h \notin C^\perp)$$

A. Valembois recherche des mots de poids faible pour une probabilité de fausse alarme et une probabilité de non détection bornées. Ainsi nous avons des quantités α et β telles que :

$$P(|h\tilde{M}^T|_H \geq T | h \in C^\perp) \leq \alpha$$

$$P(|h\tilde{M}^T|_H < T | h \notin C^\perp) \leq \beta$$

A partir de ces probabilités, du poids $|h|_H$ et $|h\tilde{M}^T|_H$ de chaque candidat h nous sélectionnons ceux dont le rapport $\frac{P(\tilde{M}|h\in C^\perp)}{P(\tilde{M})}$ (constituant le test statistique de Valembois cf. [Val00]) est supérieur au seuil fixé. L'algorithme de Valembois se déroule selon les étapes suivantes.

- On sélectionne les mots h candidats tels que $h\tilde{M}$ soit de poids faible. Cela est réalisé par l'algorithme de Canteaut-Chabaud.
- On sélectionne les bons candidats h appartenant à $C^\perp$ par test statistique.
- On regarde si l'élément choisi n'est pas combinaison linéaire des éléments déjà sélectionné afin de créer une base de $C^\perp$.

Il s'avère que le test statistique utilisé par A. Valembois dans son algorithme n'est pas optimal et une modification a été présentée par M. Cluzeau dans sa thèse ([Clu06]). Il propose un test qui a une probabilité de non-détéction plus petite que le test proposé par Valembois. Ceci est important pour trouver rapidement une base du dual.

Là où Valembois cherche à trouver le plus petit N tel que la probabilité de fausse alarme soit bornée en fonction de T, M. Cluzeau a, dans sa thèse, présenté un nouveau test permettant d'améliorer le choix des paramètres et ainsi la probabilité de fausse alarme et celle de non détection. L'idée étant de ne pas fixer la probabilité de fausse alarme mais de fixer une statistique a priori sur la probabilité de fausse alarme.

Cette statistique est définie par :

$$\frac{P(|(h\tilde{M}^T)|_H < T | h \notin C^\perp)(\text{Card}(h \notin C^\perp))}{P(|(h\tilde{M}^T)|_H < T | h \in C^\perp)(\text{Card}(h \in C^\perp))}$$

Cette statistique est fixée à une valeur α (typiquement un pour cent) et la probabilité de non détection est quant à elle fixée à β. Pour effectuer le test de M. Cluzeau, nous devons choisir T et N tel que

$$\frac{P(|(h\tilde{M}^T)|_H < T | h \notin C^\perp)(\text{Card}(h \notin C^\perp))}{P(|(h\tilde{M}^T)|_H < T | h \in C^\perp)(\text{Card}(h \in C^\perp))} = \beta$$

$$P(|h\tilde{M}^T|_H \geq T | h \in C^\perp) \leq \alpha$$

.

Ainsi nous pouvons donner les valeurs de N et T correspondant par le théorème suivant.

Théorème 2.4 *On note $x = (1-2p)^{|h|_H}$. Soit $\tilde{M}$ la matrice composé de N mots. Soit*

$$\begin{aligned} N &= \left(\frac{b\sqrt{1-x^2}+a}{x}\right)^2, \\ T &= \frac{1}{2}(N-\alpha\sqrt{N}) \end{aligned}$$

avec $a = -\phi^{-1}(\alpha\frac{2^{n-k}}{2^n-2^{n-k}}(1-\beta)$ et $b = \phi^{-1}(1-\beta)$, où ϕ est la fonction de répartition associé a la densité de la loi normale

$$\phi(x) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{x}\exp\left(-\frac{t^2}{2}\right)$$

alors

$$\frac{P(|h\tilde{M}^T|_H \leq T | h \notin C^\perp)(Card(h \notin C^\perp))}{P(|h\tilde{M}^T|_H \leq T | h \in C^\perp)(Card(h \in C^\perp))} \leq \beta$$

$$P(|h\tilde{M}^T|_H \geq T | h \in C^\perp) \leq \alpha$$

Pour des valeurs α et β données.

Cette méthode permet d'obtenir une borne inférieure pour N et une borne supérieure pour T pour une valeur de $|h|_H$ et de $|h\tilde{M}^T|_H$.

2.2.4.3 Gauss randomisé

L'algorithme du Gauss randomisé a été décrit la première fois par Sicot-Houcke en 2005 [SH05] sous le nom d'algorithme de Sicot-Houcke. Il est décrit et réutilisé par J. Barbier [Bar07] en reconnaissance de code convolutif sur un train binaire bruité. La méthode se décompose en deux étapes, une recherche d'un vecteur du code dual et une validation. On suppose la longueur de bloc n et la synchronisation s connus. L'algorithme prend une matrice $\tilde{M}$ de hauteur N de mots de code bruités. Nous supposons $N > n(m+1)$ où m est le degré du code.

Nous notons GR la décomposition de $\tilde{M}$ sous la forme d'une partie information I et d'une partie redondance J :

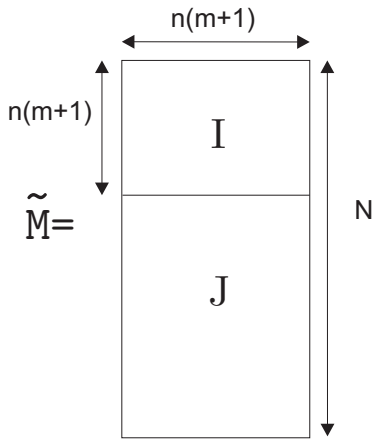


FIGURE 2.8 – Composition de la matrice $\tilde{M}$ pour l'algorithme de Gauss randomisé

Algorithm 3 ALGORITHME DE GAUSS RANDOMISÉ

Entrée : Une matrice $\tilde{M}$ de taille $N \times n(m+1)$.
T,
$l \in \mathbb{N}*$.
Sortie : Une matrice Ker contenant les vecteurs du code dual de C en fonction des paramètres (l, γ).

1: $Ker \leftarrow \{0\}$
2: **for** $i = 1$ from 1 to l **do**
3: Générer une nouvelle matrice $\tilde{M}_i$ par permutation aléatoire des lignes de $\tilde{M}$.
4: Appliquer l'algorithme de Gauss aux matrices I_i et évaluer les vecteurs candidats du dual h correspondant.
5: **for** Pour tout h candidat du dual **do**
6: Si $|J_i h|_H < T$ alors $Ker \leftarrow Ker \cup \{h\}$
7: **end for**
8: **end for**
9: On renvoie la matrice de vecteurs Ker.

Cette algorithme possède une complexité en $0(n^3)$.

Il est possible grâce a cette méthode de rechercher les paramètres n et s de notre code en utilisant une technique du critère du rang (paragraphe 2.2.4.1). Dans ce cas la complexité de notre algorithme est en $0(n^5)$.

L'algorithme de Gauss randomisé est à la fois une amélioration de l'algorithme de Gauss étendu au cas bruité et à la fois un sous algorithme de Valembois. Il convient que cet algorithme fonctionne si le taux d'erreur est faible dans la matrice I ce qui n'est pas un facteur limitant de l'algorithme de Valembois. L'approche utilisé par Valembois de recherche de mots de poids faible selon un certain « poinçonnage » permet une plus grande robustesse aux erreurs. De plus, la conception poussée de cette algorithme ne nécessite pas de calculer plusieurs matrices et de réitérer un pivot complet à chaque étape comme cela est

le cas pour la méthode du Gauss randomisé.

Chapitre 3

Reconnaissance des codes convolutifs en milieu bruité

3.1 Théorie algébrique des codes convolutifs

Une majorité des éléments de nos travaux sont inspirés des travaux de R. J. McEliece sur les codes convolutifs (cf. [McE98]). Nous en reprenons les notations afin de définir ces codeurs et leur fonctionnement qui seront la base de l'élaboration de notre méthode de reconnaissance des codes convolutifs.

3.1.1 Codeurs et codes convolutifs

Dans cette partie nous introduisons les notations et notions nécessaires à la compréhension mathématique du problème et à sa reconnaissance. Comme dans de nombreux livres nous noterons la variable inconnue D comme l'opérateur « delay »[1]. Soit $\mathbf{F}_2$ le corps a deux éléments, binaire. Notons $\mathbf{F}_2[D]$ l'anneau des polynômes sur $\mathbf{F}_2$, $\mathbf{F}_2((D))$ le corps des séries de Laurent (séries entières avec des monômes de degré négatif) sur $\mathbf{F}_2$ et $\mathbf{F}_2(D)$ le corps des fonctions rationnelles sur $\mathbf{F}_2$. Il est possible de décrire les codes convolutifs comme des codes en blocs linéaires. L'alphabet de ce code est le corps des séries de Laurent. Le corps des fractions rationnelles étant un sous-corps des séries de Laurent, le produit d'un vecteur de k séries de Laurent (les k entrées) par la matrice génératrice produit un vecteur de n séries de Laurent (les n sorties).
Nous allons donner une description algébrique d'un codeur convolutif :

- $\vec{X}(D) = (X_1(D), \ldots, X_k(D))$ le vecteur d'entrée, à chacune des k entrées correspond une suite infinie que l'on écrit comme une série en l'indéterminée D : $X_i(D) = \sum_{j=t_0}^{\infty} x_{ij} D^j$, $1 \leq i \leq k$
- $\vec{Y}(D) = (Y_1(D), \ldots, Y_n(D))$ le vecteur de sortie, à chacune des n sorties correspond une suite infinie que l'on écrit également comme une série en D : $Y_i(D) = \sum_{j=t_0}^{\infty} y_{ij} D^j$, $1 \leq i \leq n$

1. délai

– On a la relation

$$\vec{Y}(D) = \vec{X}(D) \cdot G$$

où G est la matrice de transfert, $G = (g_{ij}(D))_{1 \leq i \leq k, 1 \leq j \leq n}$ avec $g_{ij}(D) \in \mathbf{F}_2[D]$. Dans l'équation ci dessus, nous faisons le produit du vecteur $\vec{X}(D)$ par la matrice G, tous les calculs ont lieu dans l'anneau des séries entières en l'indéterminée D.

La version algébrique de la Définition 1.11 consiste ainsi à définir un codeur (n, k, m) comme étant la donnée d'une matrice $k \times n$ de polynômes de $\mathbf{F}_2[D]$ et m le degré du code.

Exemple : Le codeur binaire $(2, 1, 3)$ de matrice de transfert

$$G = \begin{pmatrix} 1 + D + D^3 & 1 + D^2 \end{pmatrix}$$

peut être représenté par le circuit donné en figure 3.1. Ce circuit est bien

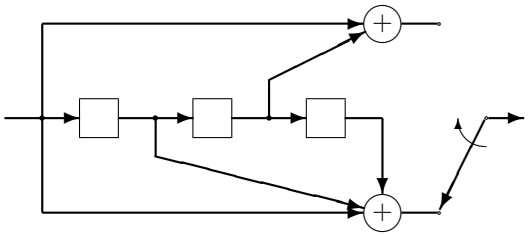


FIGURE 3.1 – Codeur convolutif $(2, 1, 3)$

linéaire, d'ordre 3 (une entrée donnée cesse d'affecter la sortie après 3 unités de temps), à 1 entrée et 2 sorties.

Le codeur binaire $(3, 2, 1)$ de matrice de transfert

$$G = \begin{pmatrix} 1 + D & D & 1 + D \\ D & 1 & 1 \end{pmatrix}$$

peut être représenté par le circuit donné en figure 3.2.

Définition 3.1 *Un code convolutif (n, k) est un sous espace de dimension k sur $\mathbf{F}_2(D)^n$*

Un ensemble $\mathcal{E} \subset \mathbf{F}_2(D)^n$ est un ensemble générateur d'un code convolutif $\mathcal{C}$ si un élément de $\mathcal{C}$ peut être écrit (pas nécessairement de façon unique) comme combinaison linéaire à coefficients dans $\mathbf{F}_2(D)$ d'éléments de $\mathcal{E}$. Inversement, pour tout ensemble $\mathcal{E} \subset \mathbf{F}_2(D)^n$, nous notons $\langle \mathcal{E} \rangle$ le code convolutif généré par $\mathcal{E}$, qui est le plus petit code convolutif de longueur n contenant $\mathcal{E}$. Par extension, pour toute matrice G à coefficients dans $\mathbf{F}_2(D)$, nous noterons $\langle G \rangle$ le code convolutif généré par les lignes de G (nous avons $\mathcal{C} = \langle G \rangle$ pour toute matrice génératrice G de $\mathcal{C}$

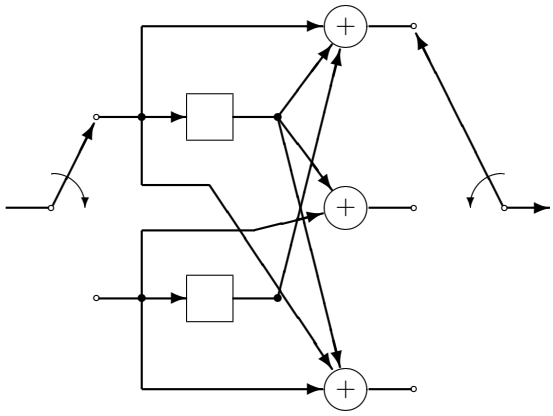


FIGURE 3.2 – Codeur convolutif $(3, 2, 1)$

3.1.2 Matrice génératrice polynomiale

Soit G une matrice polynomiale $k \times n$ à coefficients dans $\mathbf{F}_2[D]$

$$G = \begin{pmatrix} g_{1,1}(D) & \cdots & g_{1,n}(D) \\ \vdots & & \vdots \\ g_{k,1}(D) & \cdots & g_{k,n}(D) \end{pmatrix}.$$

Nous notons $\mathbf{g}_i = (g_{i,1}(D), \ldots, g_{i,n}(D))$ la i-ème ligne de G et $\deg(\mathbf{g}_i)$ le degré de g_i ou plus particulièrement de tout vecteur de polynômes, est définie comme $\deg(\mathbf{g}_i) = \max_{1 \leq j \leq n} \deg(g_{i,j}(D))$. Le *degré interne* et le *degré externe* de G sont définis par :

- $\text{intdeg}(G) \stackrel{\text{def}}{=} \max \{\deg(\Delta) \mid \Delta \text{ un mineur } k \times k \text{ de } G\}$,
- $\text{extdeg}(G) \stackrel{\text{def}}{=} \sum_{i=1}^{k} \deg(\mathbf{g}_i)$.

(un mineur $k \times k$ est le déterminant d'une sous matrice $k \times k$ de G).

Les notions de degré interne et degré externe permettent d'introduire de nombreuses propriétés des matrices génératrices polynomiales (notées MGP) que nous aborderons dans la section 3.1.4.

3.1.3 Codeurs convolutifs systématiques

Les codeurs utilisés en pratique ne se limitent pas à ceux de la Définition 1.11. Cela est vrai en particulier si l'on souhaite définir des codeurs systématiques. On utilise pour cela le corps $\mathbf{F}_2(D)$ des fractions rationnelles, défini comme l'ensemble des $P(D)/Q(D)$ où $P(D)$ et $Q(D)$ sont des polynômes de $\mathbf{F}_2[D]$ (avec $Q(D)$ non nul). Ceci nous permet de donner une seconde définition, plus générale, d'un codeur convolutif.

Définition 3.2 (Seconde définition d'un codeur convolutif) *Un codeur convolutif binaire (n, k) est une matrice $k \times n$, appelée matrice de transfert*

$$G = \begin{pmatrix} f_{1,1}(D) & \cdots & f_{1,n}(D) \\ \vdots & \ddots & \vdots \\ f_{k,1}(D) & \cdots & f_{k,n}(D) \end{pmatrix}$$

à coefficients dans $\mathbf{F}_2[D]$.

Pour définir un code convolutif à partir de cette définition de codeur, nous devons utiliser les séries de Laurent, notées $\mathbf{F}_2((D))$ plutôt que les séries entières, une série de Laurent est de la forme

$$\sum_{i=l}^{\infty} s_i D^i$$

Où les s_i sont dans $\mathbf{F}_2$ et $l \in \mathbf{Z}$ est un entier éventuellement négatif.

Définition 3.3 *Le code convolutif associé à un codeur convolutif* (n, k) *est l'ensemble des éléments* $(Y_1(D), \ldots, Y_n(D))$ *de* $\mathbf{F}_2((D))^n$ *de la forme*

$$(Y_1(D), \ldots, Y_n(D)) = (X_1(D), \ldots, X_k(D)) \begin{pmatrix} f_{1,1}(D) & \cdots & f_{1,n}(D) \\ \vdots & \ddots & \vdots \\ f_{k,1}(D) & \cdots & f_{k,n}(D) \end{pmatrix}$$

où $(X_1(D), \ldots, X_k(D)) \in \mathbf{F}_2((D))^n$ *et* $f_{i,j}(D) \in \mathbf{F}_2[D] \forall i \in [0, k], j \in [0, n]$.

Il est nécessaire d'utiliser des séries de Laurent dans cette définition car l'un des $f_{i,j}(D)$ peut avoir un facteur de la forme D^l dans son dénominateur. En pratique, la variable D représente le temps, et donc le terme de degré l de $Y_i(D)$ est le i-ème bit de sortie (parmi n) au temps l. Autoriser les degrés négatifs consiste donc à changer l'origine du temps, qui n'est plus 0, mais un nombre négatif, qui dépendra du codeur et de l'origine du temps de la séquence d'entrée.

Exemple : Le codeur binaire de matrice de transfert

$$G = \begin{pmatrix} 1 + D + D^3 & 1 + D^2 \end{pmatrix}$$

coïncide avec celui défini plus haut. En revanche on peut en donner une version systématique

$$G = \begin{pmatrix} 1 & \dfrac{1 + D^2}{1 + D + D^3} \end{pmatrix}$$

qui peut se représenter (cf. figure 3.3) sous la forme d'un circuit avec

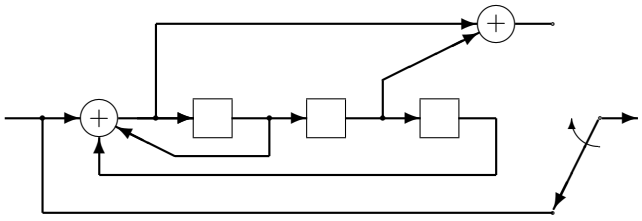


FIGURE 3.3 – Codeur convolutif $(2, 1)$ systématique

rétroaction. Le même code peut également être rendu systématique par rapport à la seconde sortie, dans ce cas nous avons

$$G = \begin{pmatrix} D + \dfrac{1}{1 + D^2} & 1 \end{pmatrix}$$

qui correspondrait à un circuit avec rétroaction d'une complexité similaire.

3.1.4 Propriétés des matrices génératrices polynomiales canoniques

La compréhension des outils mathématiques passe aussi par celui des matrices utilisées. Les matrices génératrices polynomiales possèdent de nombreuses propriétés qui découlent des notions de degré interne et degré externe définies précédemment (cf. section 3.1.2).

Définition 3.4 *Une matrice polynomiale G de taille $k \times n$ est basique si son degré interne est minimal parmi toutes les matrices $T \cdot G$ où T est inversible sur $\mathbf{F}_2(D)$.*

Définition 3.5 *Une matrice polynomiale G de taille $k \times n$ est réduite si son degré externe est minimal parmi toutes les matrice $T \cdot G$ où T est unimodulaire (coefficients polynomiaux et de déterminant égal à 1).*

Puisque toute matrice unimodulaire est le produit de matrices élémentaires, alors cette définition revient à définir une matrice dont le degré externe ne peut pas être réduit par une suite d'opération élémentaire sur les lignes.

A partir de ces définitions, nous pouvons introduire le théorème suivant :

Théorème 3.6 *Soit G une matrice polynomiale $k \times n$*

- *Si T est une matrice polynomiale de déterminant non nul $k \times k$ inversible alors,*
$$\text{intdeg}(TG) = \text{intdeg}(G) + \deg\det(T)$$
En particulier $\text{intdeg}(TG) \geq \text{intdeg}(G)$ *avec égalité si et seulement si T est unimodulaire.*
- $\text{intdeg}(G) \leq \text{extdeg}(G)$

Une preuve de ce théorème est donnée par R. J. McEliece dans [McE98].

Les propriétés basiques et réduites des matrices génératrices polynomiales imposent de nombreuses contraintes sur les matrices. Celles-ci peuvent être définies dans les deux théorèmes suivants (3.9 et 3.10) qui seront parmi les éléments fondamentaux de notre méthode de reconnaissance. Pour cela nous devons d'abord introduire la notion de facteur invariant :

Définition 3.7 *Soit G une matrice polynomiale $k \times n$. Pour tout $i \in \{1, \ldots, k\}$ le i-ème* facteur invariant *de G est égal à $\Delta_i(D)/\Delta_{i-1}(D)$ où $\Delta_i(D)$ est le pgcd de tous les mineurs $i \times i$ de G (Par convention $\Delta_0(D) = 1$).*

Définition 3.8 *Le rang d'une matrice polynomiale G est la taille du plus grand mineur non nul de G.*

Notons que pour tout $i > \text{rg}(G)$ (nous noterons dans ce manuscrit $\text{rg}(G)$ le rang d'une matrice G) le i-ème facteur invariant est nul, et pour tout $i > 0$, nous avons $\gamma_i(D)$ divise $\gamma_{i+1}(D)$. De même, nous noterons I_k la matrice $k \times k$ de $\mathbf{F}_2[D]$ comme la matrice de polynômes constant sur la diagonale et nuls ailleurs.

Théorème 3.9 *Une matrice polynomiale G de taille $k \times n$ est basique si et seulement si l'une de ces conditions est satisfaite :*

(i) Les k premiers facteurs invariants de G sont égaux à 1.

(ii) Le pgcd des mineurs $k \times k$ de G est égale à 1.

(iii) G possède un inverse à droite $\mathbf{F}_2[D]$, i.e. il existe une matrice polynomiale H de taille $n \times k$ tel que $GH = I_k$.

(iv) Pour tout $\mathbf{u} \in \mathbf{F}_2(D)^k$, $(\mathbf{u} \cdot G \in \mathbf{F}_2[D]^n) \Rightarrow \left(\mathbf{u} \in \mathbf{F}_2[D]^k\right)$

Il existe d'autres propriétés à ce théorème, cependant, dans le but d'expliquer notre méthode, nous décrivons les propriétés qui nous seront utiles par la suite dans ce manuscrit notamment pour leur capacité à être retranscrites de manière algorithmique. Ainsi, les facteurs invariants peuvent être considérés comme une notion vague, elle est souvent plus utilisée dans un contexte de calcul formel et apparaît clairement dans la méthode de la forme de Smith décrit par le théorème 3.15.

Théorème 3.10 *Soit G une matrice polynomiale $k \times n$ de coefficients $g_{i,j}(D)$. Nous notons g_i la i-ème ligne et $[G]$ la matrice binaire $k \times n$ dont le terme en position (i, j) est le coefficient de degré $\deg(\mathbf{g}_i)$ de $g_{i,j}(D)$. la matrice G est réduite si et seulement si elle vérifie*

(i) la matrice binaire $[G]$ est de rang k.

(ii) Pour tout $\mathbf{u} \in \mathbf{F}_2[D]^k$,

$$\deg(\mathbf{u} \cdot G) = \max_{1 \leq i \leq k} (\deg(u_i(D)) + \deg(\mathbf{g}_i))$$

Corollaire 3.11 *Le code convolutif généré par une matrice réduite $k \times n$ est de dimension k.*

Sinon nous pourrions trouver $\mathbf{u} \in \mathbf{F}_2(D)$ tel que $\mathbf{u} \cdot G = 0$ et donc $\mathbf{u}' \in \mathbf{F}_2[D]$ tel quel $\mathbf{u}' \cdot G = 0$ ($\mathbf{u}'$ est obtenu en multipliant $\mathbf{u}$ par le plus petit multiple commun à tous les dénominateurs des coefficients). Ce qui contredirait le théorème 3.10-(ii).

Tout code convolutif admet une matrice génératrice polynomiale (MGP). La plus petite complexité de l'encodeur issue d'une MGP est obtenu lorsque le degré externe est minimal. Nous allons dans le contexte de reconnaissance des codes convolutifs manipuler et générer des matrices canoniques. La canonicité d'une matrice génératrice polynomiale la caractérise par la matrice génératrice polynomiale « minimale ». Il est alors indispensable que notre algorithme retourne les matrices génératrices sous cette forme. Cette définition intuitive donne une idée de la canonicité. Ces matrices possèdent un grand nombre de propriétés utiles à l'élaboration d'une méthode de reconnaissance des codes convolutifs. Nous en donnons deux définitions équivalentes.

Définition 3.12 *Une MGP d'un code convolutif $\mathcal{C}$ est canonique si son degré externe est minimum parmi toutes les MGP de $\mathcal{C}$. Nous l'appelons degré (ou mémoire) de $\mathcal{C}$, noté* $\deg(\mathcal{C})$.

Théorème 3.13 *Une MGP G d'un code convolutif $\mathcal{C}$ est canonique si et seulement si elle est à la fois réduite et basique. Nous avons donc* $\text{extdeg}(G) = \text{intdeg}(G) = \deg(\mathcal{C})$. *De plus, l'ensemble des degrés des lignes est le même pour toutes MGP canonique de $\mathcal{C}$ (on appelle ces entiers les indices de Forney). Nous noterons $d_{\min}(\mathcal{C})$ et $d_{\max}(\mathcal{C})$ le plus petit et le plus grand indice de Forney de $\mathcal{C}$.*

3.1.5 Implémentation des matrices canoniques

Les propriétés, définies précédemment, permettent de nous amener aux notions de "réduit" et "basique" caractérisant les matrices "minimales" d'un code convolutif. Dans cette partie, nous cherchons à résoudre le problème de création des matrices canoniques d'un code $\mathcal{C}$ à partir d'une matrice génératrice G de $\mathcal{C}$. Les éléments de cette partie sont tirés de [JZ99]

3.1.5.1 Matrice basique

Définition 3.14 *Une matrice $k \times n$ polynomiale G est basique si son degré interne est minimal.*

Un algorithme est disponible dans [JZ99] page 39. Celui-ci utilise la forme de Smith des matrices que nous allons citer.

Théorème 3.15 *Soit G une matrice polynomiale binaire $k \times n$ avec $k \leq n$ (i.e, $G = g_{ij}(D)$, avec $g_{ij}(D) \in \mathbf{F}_2[D], 1 \leq i \leq k, 1 \leq j \leq n$) de rang r. Alors G peut être écrite de la manière suivante :*

$$G = A\Gamma B \ \in \mathbf{F}_2[D]$$

où A et B sont des matrices polynomiales binaires de taille respectivement $k \times k$ et $n \times n$, de déterminant unimodulaire, et où Γ est la matrice $k \times n$ définie par :

$$\Gamma = \begin{pmatrix} \gamma_1(D) & & & & & & & & \\ & \gamma_2(D) & & & & & & & \\ & & \ddots & & & & & & \\ & & & \gamma_r(D) & & & & & \\ & & & & 0 & & & & \\ & & & & & \ddots & & & \\ & & & & & & 0 & \dots & 0 \end{pmatrix}$$

Γ est appelée forme de Smith de G, où les éléments $\gamma_i(D)$ non nuls appartenant à $\mathbf{F}_2[D], 1 \leq i \leq r$, sont appelés les facteurs invariants de G, sont les uniques polynômes qui satisfont :

$$\gamma_i(D) | \gamma_{i+1}(D), i = 1, \dots, r-1$$

De plus, soit $\Delta_i(D) \in \mathbf{F}_2[D]$ le pgcd des mineurs d'ordre i de G alors :

$$\gamma_i(D) = \frac{\Delta_i(D)}{\Delta_{i-1}(D)}$$

avec $\Delta_0(D) = 1$ par convention $i = 1, 2, \dots, r$.

Rappel : La multiplication à gauche d'une matrice polynomiale A par une matrice de déterminant unimodulaire B modifie A par combinaisons linéaires de ses lignes.

La multiplication à droite d'une matrice polynomiale A par une matrice de déterminant unimodulaire B modifie A par combinaisons linéaires de ses colonnes.

Preuve : (Théorème 3.15)

On suppose que $G \neq 0$ sinon il n'y a rien à prouver.

Étape 1 :

Premièrement, montrons que par opérations élémentaires, nous pouvons obtenir une matrice dans laquelle l'élément supérieur gauche n'est pas nul et à le plus petit degré de tous les éléments non nuls de la matrice :

Par permutation des lignes et des colonnes on amène le polynôme de plus petit degré dans le coin supérieur gauche. Toutes les opérations modifient soit la matrice A soit la matrice B cependant nous nous intéressons seulement à la matrice Γ.

On note $\alpha_{ij}(D)$, $1 \leq i \leq k$, $1 \leq j \leq n$ les éléments de la nouvelle matrice. Désormais, nous allons chercher à annuler tous les termes sur la ligne et la colonne 1. Si l'on regarde sur la première ligne, nous avons $\alpha_{1j}(D) = \alpha_{11}\beta_j(D) + \beta_{1j}(D)$ avec $deg(\beta_{1j}(D)) < deg(\alpha_{11}(D))$. A la deuxième colonne nous ajoutons $\beta_j(D)$ fois la première. Cette opération élémentaire remplace les $\alpha_{1j}(D)$ par des $\beta_{1j}(D)$. Ainsi si $\beta_{1j}(D)$ est non nul alors nous avons réduit le degré de tous les polynômes sur la ligne. On effectue de même pour les lignes de la matrice.

Nous répétons l'opération sur la nouvelle matrice. De même que précédemment, les polynômes sur la première ligne et la première colonne sont de degré plus petit. A chaque étape nous réduisons le degré des polynômes jusqu'à obtenir un $\beta_{11}(D)$ qui divise tous les $\beta_{1j}(D)$ et les $\beta_{i1}(D)$. L'étape suivante nous fournit la matrice suivante :

$$G' = \begin{pmatrix} \beta_{11}(D) & 0 & \dots & 0 \\ 0 & & & \\ \vdots & & G" & \\ 0 & & & \end{pmatrix}$$

On note $\beta_{11}(D) = \gamma_1(D)$

Étape 2 :

Maintenant, nous voulons prouver que $\beta_{11}(D)$ divise tous les éléments de $G" = \delta_{ij}(D)$. Si nous sommes dans le cas contraire $\beta_{11}(D)$ ne divise pas les $\delta_{ij}(D)$, alors nous pouvons ajouter la i-ème ligne à la première et recommencer l'étape 1. On obtient ainsi un élément de degré plus petit que $deg(\beta_{11}(D))$, ce qui est une contradiction. On peut ainsi écrire la matrice sous la forme suivante :

$$G' = \begin{pmatrix} \gamma_1(D) & 0 & \dots & 0 \\ 0 & & & \\ \vdots & & \gamma_1(D)G^* & \\ 0 & & & \end{pmatrix}$$

Nous répétons l'opération pour G^* et ainsi de suite ce qui prouve la première partie du théorème.

Étape 3 :

Soit $\Delta_i(D)$ le pgcd des mineurs d'ordre i de la matrice G. Ces mineurs ne varient pas suivant les modifications dues aux opérations élémentaires. On note $A = a_{ij}(D)$ la matrice de permutation $k \times k$ avec $a_{ij} \in \mathbf{F}_2[D]$. L'élément (i, j) de AG est $\sum_k a_{ik}(D)g_{kj}(D)$. Le produit AG est une combinaison linéaire des lignes de G. Donc les mineurs d'ordre i de AG sont des combinaisons linéaires des mineurs d'ordre i de G. Ainsi, le pgcd des mineurs d'ordre i de G est un diviseur

de tous les mineurs d'ordre i de AG. Comme A est unimodulaire, A^{-1} est aussi une matrice polynomiale. En suivant le même raisonnement que précédemment, on montre que le pgcd des mineurs d'ordre i de AG est un diviseur de tous les mineurs d'ordre i de $A^{-1}(AG) = G$. En conclusion, les mineurs d'ordre i de G et de AG sont les mêmes. Les $\Delta_i(D)$ ne sont donc pas affectés par les opérations élémentaires sur les lignes. De la même façon, nous pouvons montrer que les $\Delta_i(D)$ sont invariants par opérations sur les colonnes.

Étape 4 :

Nous cherchons, dans cette dernière partie, à prouver que nous pouvons identifier les $\Delta_i(D) =$ pgcd de tous les mineurs d'ordre i de Γ.

La forme de Γ fait que nous avons :

$$\begin{aligned}
\Delta_1(D) &= \gamma_1(D) \\
\Delta_2(D) &= \gamma_1(D)\gamma_2(D) \\
&\vdots \\
\Delta_r(D) &= \gamma_1(D)\gamma_2(D)\dots\gamma_r(D)
\end{aligned}$$

Si $\Delta_0(D) = 1$ alors :

$$\gamma_i(D) = \frac{\Delta_i(D)}{\delta_{i-1}(D)},\ i = 1, 2, \dots, r$$

De plus, l'unicité des $\Delta_i(D),\ i = 1, 2, \dots, r$ implique l'unicité des $\gamma_i(D)$.

□

Nous cherchons à l'aide de cette décomposition de Smith à retrouver notre matrice G sous forme basique.

$$G = A \begin{pmatrix} \gamma_1(D) & & & & & & \\ & \gamma_2(D) & & & & & \\ & & \ddots & & & & \\ & & & \gamma_r(D) & 0 & \dots & 0 \end{pmatrix} B$$

Comme la partie droite de la matrice Γ est nulle, on définit la matrice génératrice G' $r \times n$ formée à partir de la matrice B dont on a supprimé les $n - b$ dernières lignes.

$$G = A \begin{pmatrix} \gamma_1(D) & & & \\ & \gamma_2(D) & & \\ & & \ddots & \\ & & & \gamma_r(D) \end{pmatrix} G'$$

Comme A et

$$\begin{pmatrix} \gamma_1(D) & & & \\ & \gamma_2(D) & & \\ & & \ddots & \\ & & & \gamma_r(D) \end{pmatrix}$$

Sont des matrices inversibles sur $\mathbf{F}_2(D)$. D'après la définition 3.19 d'équivalence des codeurs convolutifs G' est équivalent à G. Or G' est polynomiale et B possède

Algorithme 4 BASIQUE

Entrée : une matrice polynomiale G $k \times n$.
Sortie : une matrice polynomiale G, $k' \times n$ avec $k' < k$, correspondant à une partie de la matrice B décrite dans la forme de Smith.

1: **for** $i = 1$ to k **do**
2: **while** $\beta_{ij}(D) \neq 0,\ j = i+1, \ldots, n$ ou $\beta_{li}(D) \neq 0,\ l = i+1, \ldots, n$, ≪ On continue tant que seul l'élément commun de la i-ème ligne et colonne n'est pas nul ≫ **do**
3: On amène l'élément de plus petit degré en position (i, i).
4: On répercute les opérations dans la matrice B.
5: **for** $j = i + 1$ to n, ≪ on parcours les colonnes ≫ **do**
6: On multiplie la i-ème colonne par $\beta_j(D)$ et on ajoute a la j-ème colonne.
7: On répercute l'opération sur la matrice B.
8: **end for**
9: **for** $l = i + 1$ to k, ≪ on parcours les lignes ≫ **do**
10: On multiplie la i-ème ligne par $\beta^l(D)$ et on ajoute à la l-ème ligne.(la matrice B n'est pas modifiée par des opérations sur les lignes).
11: **end for**
12: **end while**
13: **end for**
14: On calcule le rang r de G.
15: On sélectionne les r premières colonnes de B que l'on transpose. On note G cette nouvelle matrice.

un inverse polynomial, donc G' a un inverse polynomial à droite, formé par les r colonnes de B^{-1}. G' est alors une matrice génératrice basique.

Exemple : Considérons la matrice génératrice polynomiale 2×3.

$$G = \begin{pmatrix} 1+D^2 & D & 1 \\ D+D^5 & D+D^2+D^4 & 1+D^2+D^3 \end{pmatrix}$$

On décompose la matrice G de la façon suivante :

$$G = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & D & 1+D^2 \\ 1+D^2+D^3 & D+D^2+D^4 & D+D^5 \end{pmatrix} \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

L'opération suivante va annuler les éléments sur la première ligne

$$\begin{pmatrix} 1 & D & 1+D^2 \\ 1+D^2+D^3 & D+D^2+D^4 & D+D^5 \end{pmatrix} \begin{pmatrix} 1 & D & 1+D^2 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 0 & 0 \\ 1+D^2+D^3 & D^2+D^3 & 1+D+D^3+D^4 \end{pmatrix}$$

Dorénavant nous allons annuler les éléments sur la première colonne

$$\begin{pmatrix} 1 & 0 \\ 1+D^2+D^3 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 1+D^2+D^3 & D^2+D^3 & 1+D+D^3+D^4 \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 0 & 0 \\ 0 & D^2+D^3 & 1+D+D^3+D^4 \end{pmatrix}$$

Tous les éléments de la ligne et de la colonne ont été annulé, nous avons mis l'élément de plus petit degré en position $(1,1)$. L'étape suivante va consister à mettre l'élément de plus petit degré en haut à gauche de la sous-matrice 1×2. Comme celui-ci est déjà à sa place nous allons annuler les termes de la ligne.

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & D^2+D^3 & 1+D+D^3+D^4 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & D \\ 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 0 & 0 \\ 0 & D^2+D^3 & 1+D \end{pmatrix}$$

Nous n'avons pas annuler la ligne, on place le terme de plus petit degré à la position $(2,2)$.

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & D^2+D^3 & 1+D \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1+D & D^2+D^3 \end{pmatrix}$$

On réitère l'opération de suppression sur la ligne jusqu'à annulation du dernier terme.

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1+D & D^2+D^3 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & D^2 \\ 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1+D & 0 \end{pmatrix}$$

La décomposition sous forme de Smith est donc

$$G = A\Gamma B$$

$$\begin{pmatrix} 1 & 0 \\ 1+D^2+D^3 & 1 \end{pmatrix} \Gamma \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & D^2 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

$$\times \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & D \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & D & 1+D^2 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

Nous en concluons que

$$A = \begin{pmatrix} 1 & 0 \\ 1+D^2+D^3 & 1 \end{pmatrix}$$

et

$$B = \begin{pmatrix} 1+D^2 & D & 1 \\ 1+D^3 & D^2 & 0 \\ D & 1 & 0 \end{pmatrix}$$

Nous avons donc la décomposition suivante sous forme de Smith

$$G = \begin{pmatrix} 1 & 0 \\ 1+D^2+D^3 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1+D^2 & D & 1 \\ 1+D^3 & D^2 & 0 \\ D & 1 & 0 \end{pmatrix}$$

Le rang r de Γ est 2, la matrice génératrice basique G_{basic} de taille 2×3 correspond au r premières lignes de la matrice B.

$$G_{basic} = \begin{pmatrix} 1+D^2 & D & 1 \\ 1+D^3 & D^2 & 0 \end{pmatrix}$$

3.1.5.2 Matrice réduite

Définition 3.16 *Une matrice $k \times n$ polynomiale G est réduite si son degré externe ne peut être réduit par une suite d'opérations élémentaires sur les lignes (additions, multiplications par un scalaire).*

On note $[G]$ de coefficients dans $\{0, 1\}$ avec un 1 à la position (i, j) si le degré du polynôme $G_{ij}(D) = v_i$ et 0 sinon (avec $v_i = \max \deg g_i$).

$$G = \begin{pmatrix} 1+D+D^3 & 1+D^3 & 1 \\ D+D^2 & 1+D^2+D^3 & 1+D \end{pmatrix}, \; [G] = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

D'après le théorème 3.10 une matrice est réduite si et seulement si la matrice $[G]$ est de rang plein. De même que pour basique, nous allons explicité l'algorithme utilisé dans [JZ99] page 58. Dans cette algorithme nous considérons que la matrice G de départ est déjà basique.

Exemple : Prenons une matrices génératrice polynomiale « aléatoire » G afin de comprendre l'algorithme précédent :

$$G = \begin{pmatrix} 1+D & D & 1 \\ 1+D^2+D^3 & 1+D+D^2+D^3 & 0 \end{pmatrix}$$

La matrice $[G]$ correspondante est donc :

$$[G] = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \end{pmatrix}$$

Cette matrice n'est pas de rang plein, et peut donc être réduite. En appliquant l'algorithme nous obtenons la matrice G :

$$G = \begin{pmatrix} 1+D & D & 1 \\ 1 & 1+D+D^2 & D^2 \end{pmatrix}$$

La matrice $[G]$ correspondante est donc :

$$[G] = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}$$

La matrice $[G]$ est de rang plein. Nous avons obtenus notre matrice réduite et basique.

Désormais nous disposons d'une méthode générique pour former une matrice génératrice polynomiale canonique à partir d'une matrice génératrice polynomiale quelconque. Ainsi nous séparons les problèmes de reconnaissance du problème de « construction » d'une matrice génératrice canonique.

Algorithme 5 FORME RÉDUITE

Entrée : Une matrice polynomiale (peut être basique) (base d'un code convolutif C) $G = G_{i,j}(D), i \in [1,k], j \in [1,n]$ avec $G_i = G_{i,1}(D), \ldots, G_{i,n}(D)$
Sortie : Une matrice polynomiale réduite du même code (si la matrice est basique en entrée alors la matrice obtenue est canonique).

1: **loop**
2: On calcul les indices de Forney de chaque ligne de la matrice G. $d_i = \deg(\mathbf{g}_i)$; Soit μ l'indice tel que $d_\mu = \max_{i=1\ldots k} d_i$.
3: Calculer la matrice binaire $[G]$ définie par $[G]_{i,j} = 1$ si $\deg(g_{i,j}(D)) = d_i$ et 0 sinon.
4: **if** $\text{rang}([G]) \neq 0$ **then**
5: **STOP** La base est réduite.
6: **else**
7: Soient $h_1, ..., h_l$ un ensemble de l lignes de $[G]$ (la ligne d'indice μ est incluse dans cet ensemble) tel que $\sum_{j=0}^{l} h_j = 0$
8: Mettre à jour la base avec :

$$G_\mu \leftarrow G_\mu + \sum_{j=1}^{l-1} h_{i_j} \times D^{d_{i_l} - d_{i_j}}.$$

9: **end if**
10: **end loop**

3.1.6 Code convolutif dual

Le code dual $\mathcal{C}^\perp$ d'un code convolutif $\mathcal{C}(n,k)$ est un code convolutif $(n, n-k)$. Considérons une matrice génératrice de $\mathcal{C}^\perp$ (ou une matrice de parité de $\mathcal{C}$)

$$H = \begin{pmatrix} h_{1,1}(D) & \cdots & h_{1,n}(D) \\ \vdots & & \vdots \\ h_{n-k,1}(D) & \cdots & h_{n-k,n}(D) \end{pmatrix}.$$

Le n-uplet $(S_1(D), \ldots, S_n(D)) \in \mathbf{F}_2(D)^n$ appartient à $\mathcal{C}$ si et seulement si

$$\begin{cases} h_{1,1}(D)S_1(D) + \cdots + h_{1,n}(D)S_n(D) &= 0 \\ & \vdots \\ h_{n-k,1}(D)S_1(D) + \cdots + h_{n-k,n}(D)S_n(D) &= 0 \end{cases}$$

Propriété : Le degré du dual $\mathcal{C}^\perp$ d'un code convolutif $\mathcal{C}$ est égal au degré de $\mathcal{C}$

3.1.7 Encodeur catastrophique

Tout d'abord, les polynômes doivent vérifier une propriété pour que le codeur soit non catastrophique. C'est-à-dire vérifient la propriété suivante :

Propriété 3.17 *Une séquence d'information de poids fini est codée en une séquence de poids fini.*

Le poids d'une série est le nombre de coefficients non nuls de la série. Contre exemple, un codeur catastrophique : $u = 110\ldots.$ et $G = (1 + D\ 1 + D^2)$ La séquence codée sera $v = (110100\ldots)$. Si au cours de la transmission les 2 premiers bits sont modifiés, un décodeur à maximum de vraisemblance décodera la séquence en $u = (00....)$, donc commettra un nombre infini d'erreurs (dans l'information).

Définition 3.18 *Une matrice génératrice G est dite non catastrophique si et seulement si le pgcd (plus grand dénominateur commun) de ses mineurs d'ordre k est une puissance de D.*

3.1.8 Non unicité des matrices génératrices d'un code convolutif

Nous examinons les problèmes liés à l'identification d'un code et d'un codeur convolutif. Reconnaître un code et éliminer les erreurs est un problème connu. Au delà de cette résolution, il subsiste, comme pour les codes en bloc une indétermination sur le codeur particulier utilisé dans le système. Dans un contexte où l'on souhaite reconstituer un système de communication, la nature de cette indétermination est évidemment essentielle, par exemple pour reconstruire le brasseur utilisé dont la sortie sera l'entrée du codeur. Nous examinons ici les définitions de codeurs convolutifs ayant un sens en pratique (c'est-à-dire les codeurs ayant une chance d'être implantés dans des systèmes de communication) et en déduisons une notion d'équivalence de codeurs.

Codeur convolutifs et codeurs systématique

Si l'on s'en tient à la Définition 1.11, dans le premier exemple, les seuls codeurs permettant d'obtenir les mêmes séquences codées sont obtenues en multipliant les deux polynômes de la matrice de transfert par un même polynôme. Ces codeurs équivalents ne sont cependant pas gênants car à partir de l'un d'entre eux on arrivera toujours à retrouver (par un calcul de pgcd) la version « minimale » donnée dans l'exemple. On imagine mal pourquoi le codeur utilisé dans un système de communication ne serait pas cette version minimale.

Le second exemple pose plus de problèmes, en effet, le codeur obtenu en multipliant la matrice de transfert à gauche par une matrice 2×2 de polynômes dont le déterminant est non nul, fournit les même séquence codées. Par exemple le codeur défini par la matrice ci-dessous

$$\begin{pmatrix} 1+D & 1 \\ D & 1 \end{pmatrix} \begin{pmatrix} 1+D & D & 1+D \\ D & 1 & 1 \end{pmatrix}$$
$$= \begin{pmatrix} 1+D+D^2 & 1+D+D^2 & D^2 \\ D^2 & 1+D^2 & 1+D+D^2 \end{pmatrix}$$

produit les même séquences codées que le codeur $(3, 2, 1)$ de l'exemple. Dans ce cas l'ordre du codeur a augmenté, mais ce n'est forcément le cas :

$$\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1+D & D & 1+D \\ D & 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 1+D & D \\ D & 1 & 1 \end{pmatrix}$$

Ce codeur a le même ordre, produit les mêmes séquences, mais la séquence d'entrée correspondant à une séquence codée ne sera pas la même qu'avec le codeur initial.

D'une façon générale, deux codeurs (n, k) de matrice de transfert G_1 et G_2 vont produire les mêmes séquences codées si et seulement si il existe une matrice U inversible de taille $k \times k$ à coefficients dans $\mathbf{F}_2(D)$ telle que $G_2 = UG_1$.

Une notion d'équivalence

La définition opérationnelle (et aussi la plus simple) de l'équivalence de deux codeurs convolutifs est la suivante :

Définition 3.19 *Deux codeurs convolutifs sont* équivalents *s'ils produisent le même code convolutif.*

Dans cette définition, la définition de *codeur convolutif* doit être la seconde (Définition 3.2). La première définition, ce que nous appellerons les codeurs convolutifs polynomiaux, est suffisante pour décrire tous les *codes* convolutifs, mais la seconde est nécessaire pour décrire toutes les réalisations matérielles de ces codes (c'est-à-dire tous les codeurs, et en particulier les codeurs systématiques).

Le code pourra être défini de façon unique à l'aide de la séquence observée. Le codeur particulier utilisé ne dépend pas de l'observation, et ne peut donc être retrouvé qu'à l'aide d'une information additionnelle soit sur la séquence d'entrée, soit sur le type de codeur.

Dans la pratique, les codeurs convolutifs utilisés sont $(2, 1)$ (une entrée, deux sorties, éventuellement poinçonné). Dans ce cas il n'existe qu'un codeur polynomial, et donc le codeur convolutif est défini *à une multiplication par une fraction rationnelle de* $\mathbf{F}_2(D)$ *près.* Un codeur convolutif $(2, 1)$ sera donc toujours composé d'un multiplicateur par une fraction rationnelle (c'est-à-dire en pratique un brasseur) suivi d'un codeur polynomial uniquement défini.

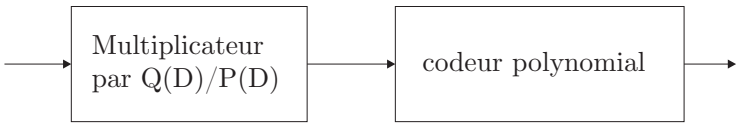


FIGURE 3.4 – Codeur convolutif $(1, 2)$

Impact sur l'identification d'un codeur.

Le brasseur défini par la multiplication par une fraction rationnelle est très proche d'un brasseur auto-synchronisant (qui est une division par un polynôme $P(D)$), la résolution du problème va donc poser la même difficulté : il faut faire une hypothèse sur la séquence d'entrée. En revanche un paramètre sera considérablement plus simple : le degré du registre sera inférieur à l'ordre du codeur convolutif, et donc ne dépassera pas 8 ou 10 en pratique.

3.1.9 les codes convolutifs poinçonnés

En poinçonnant le code nous cherchons à augmenter le taux de transmission tout en conservant le plus possible les propriétés de correction du code. L'opération de poinçonnage des codes convolutifs consiste à supprimer des colonnes de la matrice du code regroupé tout en conservant la longueur de contrainte m. On sélectionne N' bits à transmettre parmi $N = Mn$ bits (où M est la profondeur de regroupement). Cela revient à appliquer un masque de longueur N et de poids N' sur la sortie de notre codeur.

3.1.9.1 Les codes regroupés

On peut voir un code convolutif $(n, 1, m)$ comme un code dans lequel on observe non plus une entrée et n sorties par unité de temps mais M bits d'entrée et Mn bits de sortie. On parlera de code regroupé M fois noté $G^{[M]}$. Si $G = [P_1(D), \ldots, P_n(D)]$ est une matrice génératrice de $\mathcal{C}$ alors $G^{[M]}$ est une matrice de taille $M \times (Mn)$ composée de blocs de taille $1 \times n$. Chaque bloc contient n polynômes déduits des polynômes $P_1(D), \ldots, P_n(D)$ en ne conservant que les termes de degrés égaux à un certain α modulo M(dépend du bloc que l'on regarde). Par exemple, en regroupant 3 fois un code convolutif $(2, 1)$ de matrice génératrice $G = [P_1(D), P_2(D)]$ on obtient une matrice de la forme :

$$G^{[3]} = \left(\begin{array}{cc|cc|cc} A_1(D) & A_2(D) & B_1(D) & B_2(D) & C_1(D) & C_2(D) \\ DC_1(D) & DC_2(D) & A_1(D) & A_2(D) & B_1(D) & B_2(D) \\ DB_1(D) & DB_2(D) & DC_1(D) & DC_2(D) & A_1(D) & A_2(D) \end{array}\right)$$

où, pour tout $i \in \{1, 2\}$, $P_i(D) = A_i(D^3) + DB_i(D^3) + D^2C_i(D^3)$.

Théorème 3.20 *Si G est une matrice génératrice canonique d'un code $\mathcal{C}$, alors $G^{[M]}$ est une matrice génératrice pour le code regroupé $\mathcal{C}^{[M]}$. Le degré (longueur de contrainte) du code regroupé $\mathcal{C}^{[M]}$ est égal au degré de $\mathcal{C}$.*

Exemple : Considérons un code convolutif $(2, 1)$ de longueur de contrainte 4 avec $G = (1 + D + D^2 + D^4 \;\; 1 + D^3 + D^4)$. Nous pouvons écrire G comme $G = (1,1) + (1,0)D + (1,0)D^2 + (0,1)D^3 + (1,1)D^4$. Soit $G'(D)$ la décomposition polynomiale de G :

$$G'(D) = \begin{pmatrix} (1,1) & (1,0)D & (1,0)D^2 & (0,1)D^3 & (1,1)D^4 & & \\ & (1,1)D & (1,0)D^2 & (1,0)D^3 & (0,1)D^4 & (1,1)D^5 & \\ & & (1,1)D^2 & (1,0)D^3 & (1,0)D^4 & (0,1)D^5 & (1,1)D^6 \\ & \vdots & & & & \ddots & \end{pmatrix}$$

Le regroupement consiste à regrouper dans la matrice G' les colonnes par groupe de M, nous choisissons $M = 3$. On utilisera le changement de variable : $d^i = D^j$ modulo D^M

$$G'^{[3]} = \left(\begin{array}{ccc|ccc|c} (1,1) & (1,0)D & (1,0)D^2 & (0,1)D^3 & (1,1)D^4 & & \\ & (1,1)D & (1,0)D^2 & (1,0)D^3 & (0,1)D^4 & (1,1)D^5 & \\ & & (1,1)D^2 & (1,0)D^3 & (1,0)D^4 & (0,1)D^5 & (1,1)D^6 \end{array}\right)$$

$$\begin{aligned} G'^{[3]} &= \begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix} + \begin{pmatrix} 0 & d & d & d & 0 & 0 \\ d & 0 & 0 & d & d & d \\ d & 0 & d & 0 & 0 & d \end{pmatrix} \\ &\quad + \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ d^2 & d^2 & 0 & 0 & 0 & 0 \end{pmatrix} \\ &= \begin{pmatrix} 1 & 1+d & 1+d & d & 1 & 0 \\ d & 0 & 1 & 1+d & 1+d & d \\ d+d^2 & d^2 & d & 0 & 1 & 1+d \end{pmatrix} \end{aligned}$$

Le degré du code (somme du max des degrés des lignes d'une matrice canonique) est bien égal à 4.

3.1.9.2 Le poinçonnage

Le code convolutif parent (n, k) est transformé en un code convolutif $(N', K = M)$ de taux de transmission supérieur au code parent. Cette transformation consiste en une étape de regroupement M fois et en une suppression des colonnes ayant donc pour but d'augmenter le nombre de sorties (et d'entrées) afin d'augmenter le taux de transmission tout en gardant les propriétés du code. Prenons une MGP $(2, 1)$ de polynômes générateurs $(P_1(D)\ P_2(D))$ que nous poinçonnons avec un masque de longueur $l = 6$ [110110]. Procédons à l'étape de regroupement pour $M = \frac{l}{n} = 3$. Nous obtenons la matrice 3×6 suivante :

$$\left(\begin{array}{cc|cc|cc} A_1(D) & A_2(D) & A_2(D) & B_2(D) & C_1(D) & C_2(D) \\ DC_1(D) & DC_2(D) & A_1(D) & A_2(D) & B_1(D) & B_2(D) \\ DB_1(D) & DB_2(D) & DC_1(D) & DC_2(D) & A_1(D) & A_2(D) \end{array}\right)$$

Pour la deuxième étape nous supprimons les colonnes 3 et 6 correspondant aux $\ll 0 \gg$ dans le masque de poinçonnage. Nous obtenons la MGP du code poinçonné de taille 3×4 suivante :

$$\left(\begin{array}{cccc} A_1(D) & A_2(D) & B_2(D) & C_1(D) \\ DC_1(D) & DC_2(D) & A_2(D) & B_1(D) \\ DB_1(D) & DB_2(D) & DC_2(D) & A_1(D) \end{array}\right)$$

Nous appelons code parent le code $(2, 1)$ avant poinçonnage et le code fils le code poinçonné.

Il convient que le code ainsi poinçonné ne génère pas le même ensemble de mots que le code parent. L'algorithme de reconnaissance des codes convolutifs identifiera alors le code (N', K) et une base associée. Il est possible, supposant connue la véritable matrice génératrice du code, de décoder à partir du code (N', K'). Dans ce cas il convient que nous avons deux possibilités, soit retrouver le code parent généré par la matrice $(P_1(D)\ P_2(D))$ soit décoder directement à partir de la matrice du code poinçonné. L'algorithme de décodage de Viterbi possède une complexité de 2^{m+k} (avec m le degré du code et k le nombre d'entrées de celui-ci). La longueur de contrainte étant invariante, nous avons $2^{m+k} < 2^{m+K}$, selon les codes, il devient non négligeable d'un point vue complexité de décoder le code (n, k) plutôt que le code (N', K'). A cela s'ajoute que pour retrouver l'information nous devons disposer de la matrice génératrice. Il convient que retrouver (par recherche exhaustive) toutes les matrices canoniques d'un même code est plus rapide pour un code de petits paramètres que pour son code poinçonné. Ces deux avantages indéniables de reconnaissance du code parent pose le problème des codes poinçonnés. Le problème qui se pose est donc le suivant : comment, à partir d'une base (matrice) basique réduite, retrouver une autre base (matrice) basique réduite possédant de plus la structure forte d'un code regroupé ?

3.1.9.3 Une méthode de poinçonnage complémentaire

Une autre façon de considérer les blocs regroupés et le poinçonnage est décrite par R.J. McEliece (cf.[McE98]). Celle-ci permet de formaliser la méthode précédente en utilisant la décomposition polyphase des matrices

Définition 3.21 *$f(D) = a_0 + a_1 D + a_2 D^2 + \ldots$ est une série entière polynomiale d'indéterminé D. Pour tout entier $M \geq 1$, on appelle la M-ème décomposition polyphase de $f(D)$ les M séries entières défies :*

$$(f_{0,M}(D), f_{1,M}(D), \ldots, f_{M-1,M}(D)) \stackrel{def}{=} \left(\sum_{k\geq 0} a_{kM} D^k, \sum_{k\geq 0} a_{kM+1} D^k, \ldots, \sum_{k\geq 0} a_{kM+(M-1)} D^k \right)$$

Remarque : Nous avons utilisé cette décomposition de manière cachée lors du changement de variable de l'explication sur les codes regroupés vu précédemment.

La décomposition polyphase nous permet d'introduire la notion de matrice polycyclique pseudo-circulante associée à $f(D)$. Pour $M \geq 1$, si nous prenons $(f_1(D), f_2(D), \ldots, f_{M-1}(D))$ la M-ème décomposition polyphase de $f(D)$, alors la m-ème matrice polycyclique pseudo-circulante associée à $f(D)$ est la matrice polynomiale $M \times M$ noté $\overline{F}^M$ définie par :

$$\overline{F}^{[M]} = \begin{pmatrix} f_0 & f_1 & \cdots & f_{M-1} \\ Df_{M-1} & f_0 & \cdots & f_{M-2} \\ \vdots & & & \vdots \\ Df_1 & Df_2 & \cdots & f_0 \end{pmatrix}$$

Ainsi si nous reprenons l'exemple de la matrice génératrice $G = (1 + D + D^2 + D^4 \; 1 + D^3 + D^4)$, la décomposition sous forme de matrice polycyclique pseudo-circulante pour $M = 3$ donne pour le premier polynôme :

$$\overline{g}_1^{[M]} = \begin{pmatrix} 1 & 1+d & 1 \\ d & 1 & 1+d \\ d+d^2 & d^2 & 1 \end{pmatrix}$$

pour le second polynôme nous obtenons :

$$\overline{g}_2^{[M]} = \begin{pmatrix} 1+d & d & 0 \\ 0 & 1+d & d \\ d^2 & 0 & 1+d \end{pmatrix}$$

la matrice du code regroupé est la matrice formée de la concaténation des matrices polycycliques pseudo-circulantes, $\overline{G}^{[M]} = (\overline{g}_1^{[M]}, \overline{g}_2^{[M]})$:

$$\overline{G}^{[M]} = \begin{pmatrix} 1 & 1+d & 1 & 1+d & d & 0 \\ d & 1 & 1+d & 0 & 1+d & d \\ d+d^2 & d^2 & 1 & d^2 & 0 & 1+d \end{pmatrix}$$

Cette matrice est la matrice $G'^{[M]}$ trouvée dans le cas précédent à une permutation des colonnes près. Cette observation permet d'introduire le théorème suivant.

Théorème 3.22 *Si $\mathcal{C}$ est un code convolutif (n, k), le code bloc de taille M de $\mathcal{C}$ noté $\mathcal{C}^{[M]}$, est un (nM, kM) code convolutif. Si $G = (g_{i,j}(D)$ est une matrice génératrice polynomiale du code $\mathcal{C}$ de taille $k \times n$, alors une matrice génératrice*

de $\mathcal{C}^{[M]}$ appelé $G^{[M]}$ peut être obtenue à partir de G en remplaçant chaque entrée de $g_{i,j}(D)$ de G par la matrice polycyclique pseudo-circulante de taille M et en permutant les colonnes de profondeur M.

Ainsi si l'on inverse les colonnes de profondeur 3 dans notre exemple nous avons la matrice :

$$G'^{[M]} = \begin{pmatrix} 1 & 1+d & 1+d & d & 1 & 0 \\ d & 0 & 1 & 1+d & 1+d & d \\ d+d^2 & d^2 & d & 0 & 1 & 1+d \end{pmatrix}$$

Ces matrices regroupées gardent certaines propriétés de leur matrices parents, notamment la canonicité.

Théorème 3.23 *Soit G une MGP de $\mathcal{C}$, alors $G^{[M]}$ est une matrice génératrice pour $\mathcal{C}^{[M]}$. Donc le degré du code bloc $\mathcal{C}^{[M]}$ est le même que celui du code original.*

Ce théorème 3.23 permet d'introduire les modifications qui ont lieux sur les coefficients de Forney tel que la matrice reste canonique.

Théorème 3.24 *Le mise en bloc d'un code convolutif (n,k) est un code convolutif (nM,kM) de même longueur de contrainte et de même distance libre. Si les indices de Forney (degré max de chaque ligne de la matrice génératrice) du code d'origine sont $(f_1, f_2, \ldots, f_k)$ alors les indices de Forney pour $C^{[M]}$ sont :*

$$(\lfloor \frac{f_1}{M} \rfloor, \lfloor \frac{f_2+1}{M} \rfloor, \ldots, \lfloor \frac{f_i+M-1}{M} \rfloor) \text{ pour } i = 1 \ldots k.$$

Le passage d'un code regroupé à un code poinçonné s'effectue de la même manière que précédemment par la suppression des colonnes correspondant aux $\ll 0 \gg$ du motif de poinçonnage.

3.2 Méthode de reconnaissance des codes convolutifs

Au cours de la chaîne de télécommunication certaines informations peuvent être retrouvées, c'est le cas de la synchronisation ou du type de canal que l'étape de démodulation peut dans des cas favorables retrouver. Cependant, nous nous plaçons dans le cas où seule la séquence binaire codée est fournie par la démodulation.

3.2.1 État de l'art

E. Filiol a démontré au cours de ses travaux (cf.[Fil97b]) que le problème de reconstruction de code convolutif, bien qu'il ait été étudié en terme de polynômes, peut être exprimé comme un système linéaire sur le corps $\mathbf{F}_2$. Ce système linéaire nécessite seulement un petit nombre de données (quelques dizaine de bits) et retourne une réponse juste (une description du code convolutif, paramètres), si les données ne contiennent pas d'erreur. La probabilité qu'une séquence interceptée contienne suffisamment de données non bruitées est faible mais n'est pas négligeable et seulement un certain nombre d'essais d'une fenêtre

glissante sur nos données est nécessaire. Dans sa thèse ([Bar07]), J. Barbier utilise explicitement le concept de matrice génératrice canonique. En particulier, il convient que le problème de reconstruction des codes convolutifs ne consiste pas seulement à trouver une matrice génératrice mais aussi de trouver une matrice génératrice *canonique*. Celui-ci devient plus difficile lorsque les paramètres du code augmentent.

3.2.2 Algorithme de Filiol-Barbier

Le principe de l'algorithme est de ramener le problème des codeurs (n, k) au traitement des cas des codeurs $(k+1, k)$ plus simple.

Cas des codeur $(2, 1)$
Le cas des codeurs $(2, 1)$ est simple, une matrice génératrice du code dual de $(P_1(D), P_2(D))$ (avec $P_i(D)$ les polynômes générateurs) est $(P_2(D), P_1(D))$. Dans le cas d'un codeur $(2, 1)$, il n'y a pas d'indétermination des polynômes. Les seuls polynômes premiers entre eux annulant $(P_2(D), P_1(D))$ sont $(P_1(D), P_2(D))$. Les polynômes retrouvés correspondent à ceux utilisés dans le codeur, nous pouvons étendre ce résultats au cas des code $n, 1$ pour lesquels il n'y a pas d'indétermination des polynômes générateurs. Cette constatation n'est plus vérifiée pour les codes n, k.

Cas des codeur $(n, 1)$
Le code se présente sous la forme $(P_1(D), P_2(D), P_3(D), \ldots, P_n(D))$. Une matrice génératrice non canonique du code dual est :

$$\begin{pmatrix} P_2(D) & P_1(D) & 0 & 0 & 0 \\ P_3(D) & 0 & P_1(D) & 0 & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ P_n(D) & 0 & \ldots & 0 & P_1(D) \end{pmatrix}$$

En pratique, l'algorithme de E. Filiol et J. Barbier se ramène au cas (2,1) en gardant en commun à chaque étape la sortie 1. Il construit ainsi la matrice ligne par ligne en n étapes. Dans le cas ou tous les polynômes $P_1 \ldots P_n$ sont premiers entre eux alors le polynôme P_1 est le même pour chacune des lignes de la matrice. Cependant, si les polynômes ont un facteur commun, il peut se produire une diminution du degré. Dans ce cas, nous devons vérifier la consistance des lignes. Ceci est expliqué plus précisément dans le paragraphe suivant, de même que le passage d'une matrice génératrice du code dual aux polynômes.

Cas des codeur (n, k)

cas des codeur $(k+1, k)$
Le dual d'un code $(k+1, k)$ est un code $(k+1, 1)$. Si G est une matrice génératrice canonique du code $(k+1, k)$ alors $\Delta = (\Delta_1(D), \Delta_2(D), \ldots, \Delta_{k+1}(D))$, où $\Delta_i(D)$ est le i-ème mineur $k \times k$ de G (i.e. on enlève la i-ème colonne de G), est une matrice génératrice canonique du code dual. L'algorithme de J. Barbier exploite cette propriété et l'étend aux codes (n, k).

Application aux codes (n, k)
Nous nous ramenons au cas des codeurs $(k+1, k)$ en considérant notre codeur (n, k) comme $(n-k)$ codeurs $(k+1, k)$. En pratique nous conservons les

k premières sorties et faisons varier la $k+1$ ème $n-k$ fois. Le mineur d'ordre k pour les k premières sorties est le même (noté g) pour chacun des $n-k$ codeurs $(k+1,k)$. Dans le cas général une matrice génératrice non canonique du code dual est de la forme :

$$\begin{pmatrix} \Delta_{1,1}(D) & \Delta_{1,2}(D) & \dots & \Delta_{1,k}(D) & g(D) & 0 & \dots & \dots \\ \Delta_{2,1}(D) & \Delta_{2,2}(D) & \dots & \Delta_{2,k}(D) & 0 & g(D) & 0 & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \Delta_{n-k,1}(D) & \Delta_{n-k,2}(D) & \dots & \Delta_{n-k,k}(D) & 0 & \dots & \dots & g(D) \end{pmatrix}$$

L'algorithme de J. Barbier crée cette matrice ligne par ligne comme dans le cas $(n,1)$. Il convient que le degré des k premiers $\Delta_{1,1}(D),\dots,\Delta_{1,k}(D)$ peut varier. Ce problème d'augmentation de degré se révèle principalement dans le cas des codeurs (n,k). La difficultés réside dans le contrôle de ces degrés afin qu'ils restent le plus petit possible, assurant ainsi la canonicité du codeur. Si Le degré de g de la première ligne est plus petit que celui de la seconde ligne, on augmente le degré de la première ligne. On introduit alors un terme de consistance dans le but d'ajuster la solution obtenue. Les $\Delta i,j(D)$, pour les $n-k$ codeurs, sont calculés puis ajusté à l'aide du facteur de consistance afin d'obtenir un g identique pour chaque lignes.

Une fois la consistance réalisée, nous voulons retrouver une matrice génératrice canonique de notre code (n,k). L'algorithme, pour chaque ligne de la matrice génératrice du code dual $C^{\perp}$, calcul un nouveau dual. Nous créons pour la première ligne une matrice du code $(k+1,1)$, de dual $(P_1(D),P_2(D),\dots,P_{k+1}(D))$. En réitérant avec la seconde ligne de notre matrice $C^{\perp}$ nous trouvons le polynôme $P_{k+2}(D)$. L'algorithme construit ainsi $n-k$ matrices.

3.2.3 Codes binaires et sous-codes canoniques d'un code convolutif

Le problème de reconnaissance des codes convolutifs contient un sous-problème de construction de matrices canoniques. Dans l'algorithme vu précédemment ces deux problèmes sont résolus simultanément. Nous voulons proposer une approche dans laquelle ces deux problèmes seraient séparés. A partir des algorithmes fournis dans [JZ99], nous créons une méthode générique permettant de passer d'une matrice génératrice à une matrice génératrice canonique. Cette approche permettrait de nous libérer des contraintes polynomiales au cours du calcul pour nous orienter sur un aspect de reconnaissance pure. Pour cela nous devons décrire les notions structurelles des codes convolutifs de manière plus approfondies.

Soit $\mathcal{C}(n,k)$ un code convolutif. Une séquence polynomiale codée f noté $f=(f_1(D),\dots,f_n(D))$ de $\mathcal{C}$ est un élément de $\mathcal{C}\bigcap\mathbf{F}_2[D]^n$.

3.2.3.1 Code binaire associé avec un code convolutif

Pour tout $s\geq 0$, nous considérons les espaces polynomiaux de degré $<s$ suivants :

$$\overline{\mathcal{C}}_s=\{\mathbf{f}\in\mathcal{C}\cap\mathbf{F}_2[D]^n \mid \deg(\mathbf{f})<s\}\,.$$

Cet ensemble est stable par addition et peut être vu comme un sous-espace linéaire du $\mathbf{F}_2$ espace vectoriel linéaire de dimension sn.

$$\{\mathbf{f} \in \mathbf{F}_2[D]^n \mid \deg(\mathbf{f}) < s\} \simeq \mathbf{F}_2^{sn}.$$

Nous appelons $\overline{\mathcal{C}}_s$ le s-ème code binaire associé à $\mathcal{C}$.

Proposition 3.25 *La suite des codes linéaires binaires $(\overline{\mathcal{C}}_s)_{s\geq 0}$ est croissante pour l'inclusion. Leur dimension, en tant qu'espace vectoriel sur $\mathbf{F}_2$ vérifie :*

$$\begin{cases} \dim_{\mathbf{F}_2}(\overline{\mathcal{C}}_s) = 0 & si\ s \leq d_{\min}(\mathcal{C}), \\ \dim_{\mathbf{F}_2}(\overline{\mathcal{C}}_s) = sk - m & si\ s \geq d_{\max}(\mathcal{C}). \end{cases}$$

Où $m = \deg(\mathcal{C})$, $d_{\min}(\mathcal{C})$ et $d_{\max}(\mathcal{C})$ sont respectivement le plus petit et le plus grand indice de Forney de $\mathcal{C}$

Preuve : (Proposition 3.25)

Soit G une matrice génératrice canonique du code $\mathcal{C}(n,k)$. Soit $\mathbf{g}_i$ la i-ème ligne de G et soit e_i son degré. Soit $\mathbf{f}$ une séquence codée de degré $< s$. Comme G est à la fois basique et réduite, à partir du théorème 3.9 et du théorème 3.10 nous en déduisons que toute séquence codée $\mathbf{f}$ de degré $< s$ peut s'écrire de manière unique $\mathbf{u}\cdot G$ avec $\mathbf{u} = (u_1(D), \ldots, u_k(D)) \in \mathcal{F}_2[D]^k$ et $\deg(u_i(D)) < s - e_i$ pour tout $i = 1, \ldots, k$. Si $e_i \geq s$, alors $u_i(D) = 0$, sinon il y a 2^{s-e_i} valeurs possibles pour $u_i(D)$. Il en suit que

$$\dim_{\mathcal{F}_2}(\overline{\mathcal{C}}_s) = \sum_{i=1}^{n} \max(0, s - e_i), \tag{3.1}$$

et si $s \geq d_{\max}(\mathcal{C})$ nous avons une dimension $\sum_{i=1}^{n}(s - e_i) = sn - \deg(\mathcal{C})$.
□

3.2.3.2 Sous-codes canoniques d'un code convolutif

Pour tout entier $s \geq 0$, nous considérons le code convolutif $\overline{\mathcal{C}}_s = \langle \overline{\mathcal{C}}_s \rangle$ engendré par les séquences codées polynomiales de degré $< s$. Nous appelons $\mathcal{C}_s$, le s-ème sous-code canonique de $\mathcal{C}$

Proposition 3.26 *La suite de codes convolutifs $\left(\overline{\mathcal{C}}_s\right)_{s\geq 0}$ est croissante par inclusion. Nous avons,*

$$\{0\} = \mathcal{C}_v \subsetneq \mathcal{C}_{v+1} \subset \cdots \subset \mathcal{C}_d \subsetneq \mathcal{C}_{d+1} = \mathcal{C},$$

associé à l'endroit où $v = d_{\min}(\mathcal{C})$ et $d = d_{\max}(\mathcal{C})$. De plus, pour tout $s > 0$

$$\dim_{\mathbf{F}_2}(\overline{\mathcal{C}}_s) = \dim_{\mathbf{F}_2}(\overline{\mathcal{C}}_{s-1}) + \dim_{\mathbf{F}_2(D)}(\mathcal{C}_s),$$

Où $\dim_{\mathbf{F}_2(D)}(\mathcal{C}_s)$ est la dimension de $\mathcal{C}_s$ comme code convolutif, qui est sa dimension en tant qu'espace vectoriel sur $\mathbf{F}_2(D)$.

Preuve : (Proposition 3.26)

Nous prenons la même notations que dans la preuve que la proposition 3.25. Le code $\mathcal{C}_s$ est généré par les lignes de G de degré $< s$, sinon il y aurait une contradiction avec le théorème 3.10. Cela prouve que les $\mathcal{C}_s$ forment une suite de codes croissant. Soit $v = d_{\min}(\mathcal{C})$ et $d = d_{\max}(\mathcal{C})$. Il n'y a aucune ligne de degré

$< v$ et au moins une ligne de degré v, donc $\mathcal{C}_v = \{0\}$ et $\mathcal{C}_{v+1} \neq \{0\}$. Toutes les lignes sont de degré $< d+1$ et au moins une de degré d, donc $\mathcal{C}_{d+1} = \mathcal{C}$ et $\mathcal{C}_d \neq \mathcal{C}$. D'après l'équation (3.1) nous pouvons facilement déduire

$$\dim_{\mathbf{F}_2}(\overline{\mathcal{C}}_s) - \dim_{\mathbf{F}_2}(\overline{\mathcal{C}}_{s-1}) = \sum_{i=1, e_i<s}^{n} 1$$

et le terme de droite correspond au nombre de lignes de degré $< s$, qui est la dimension du code convolutif $\mathcal{C}_s$.
□

3.2.3.3 Représentation binaire des séquences polynomiales

Au cours de notre résolution nous devons écrire des matrices génératrices binaires. Sur celles-ci, nous appliquons divers algorithmes (comme l'algorithme de Gauss) pour lesquels l'ordre d'écriture des coefficients doit être spécifié. On introduit de ce fait les notations [2] LE et BE les deux morphismes surjectifs d'espace vectoriel sur $\mathbf{F}_2$ par :

$$\begin{array}{rcccl} LE_l & : & \mathbf{F}_2(x)^n & \rightarrow & \mathbf{F}_2^{ln} \\ & & \mathbf{f} & \mapsto & (\mathbf{f}_{\bullet,l-1}, \ldots, \mathbf{f}_{*,1}, \mathbf{f}_{*,0}) \\ BE_l & : & \mathbf{F}_2(x)^n & \rightarrow & \mathbf{F}_2^{ln} \\ & & \mathbf{f} & \mapsto & (\mathbf{f}_{*,0}, \mathbf{f}_{*,1} \ldots, \mathbf{f}_{*,l-1}) \end{array}$$

Où pour tout $\mathbf{f} = (\mathbf{f}_1(x), \ldots, \mathbf{f}_l(x)) \in \mathbf{F}_2(x)^l$, avec $\mathbf{f}_i(x) = \sum_j \mathbf{f}_{i,j} x^j$, nous notons $\mathbf{f}_{*,j} = (\mathbf{f}_{1,j}, \ldots, f_{l,j}) \in \mathbf{F}_2^l$ la séquence des coefficients de degré j. Pour toute séquence binaire $v \in \mathbf{F}_2^{ln}$, nous noterons $BE_s^{-1}(v)$(respectivement $LE_s^{-1}(v)$) l'unique séquence polynomial de degré $< l$ tel que $BE_s^{-1}(\mathbf{f}) = v$ (resp. $LE_s^{-1}(\mathbf{f}) = v$). Nous pouvons étendre cette notations aux matrices et espaces linéaires.

3.2.4 Code dual à partir d'une séquence tronquée

Soit $S = (S_1(D), \ldots, S_n(D))$ une séquence codée. Nous considérerons cette séquence vierge de bruit, nous montrerons par la suite quels outils utiliser afin que notre méthode fonctionne dans le cas bruité. Soit $\mathcal{C}^\perp$ le plus grand code convolutif de longueur n orthogonal à S. Une séquence $(v_1(D), \ldots, v_n(D))$ est dans $\mathcal{C}^\perp$ si et seulement si

$$v_1(D)S_1(D) + \cdots + v_n(D)S_n(D) = 0. \tag{3.2}$$

Si $\mathcal{S}$ est tronqué du degré 0 à $\ell - 1$ (nous connaissons tous les $S_{i,j}$ avec $1 \leq i \leq n$ et $1 \leq j \leq \ell$) et si nous écrivons 3.2 pour des séquences polynomiales de $\mathcal{C}^\perp$ de degré $< s$, nous avons les équations linéaires suivantes sur $\mathbf{F}_2$

$$\sum_{i=1}^{n} \sum_{j=0}^{s-1} v_{i,j} S_{i,d-j} = 0, s-1 \leq d \leq \ell - 1. \tag{3.3}$$

2. little endian LE (petit degré à droite),big endian BE (grand degré à droite)

Nous réécrivons le systèmes 3.3 dont les solutions sont le noyau de la matrice binaire suivante.

$$S = \begin{array}{|c|c|c|c|} \hline \mathbf{S}_{*,0} & \mathbf{S}_{*,1} & \cdots & \mathbf{S}_{*,s-1} \\ \hline \mathbf{S}_{*,1} & \mathbf{S}_{*,2} & \cdots & \mathbf{S}_{*,s} \\ \hline \vdots & \vdots & & \vdots \\ \hline \mathbf{S}_{*,\ell-s} & \mathbf{S}_{*,\ell-s+1} & \cdots & \mathbf{S}_{*,\ell-1} \\ \hline \end{array} . \tag{3.4}$$

En d'autres termes, la i-ème ligne de S, pour $i = 0, \ldots, \ell - 1$ est égale à $BE_s(\frac{1}{D^i}\mathcal{S})$. Le noyau de S est égale au code convolutif binaire $LE_s(\overline{\mathcal{C}^\perp}_s)$ (où $\overline{\mathcal{C}^\perp}_s$ est le s-ème code binaire associé à $\mathcal{C}^\perp$. cela nous fournit un ensemble générateur de $\mathcal{C}^\perp$. Nous remarquons que nous pouvons facilement obtenir la dimension et la longueur de contrainte de $\mathcal{C}$ à ce moment, ainsi à partir de la proposition 3.25, la dimension de $\overline{\mathcal{C}^\perp}_s$ sur $\mathbf{F}_2$ est $s(n-k) - \deg \mathcal{C}^\perp$ et $\deg \mathcal{C} = \deg \mathcal{C}^\perp$

A partir de la séquence $\mathcal{S}$ nous devons construire le système S. Il est bien sûr impossible de connaître les paramètres exactes du code et donc la taille de la matrice. Nous allons procéder par recherche exhaustive jusqu'à obtention d'une solution. Dès qu'une solution à notre algorithme est retournée, celle-ci est testée sur une partie non utilisé de notre train intercepté pour être validée. Si une solution est validée, nous avons trouvé les plus petits paramètres d'un codeur pouvant engendrer le code. Dans l'autre cas nous continuons la recherche jusqu'à une borne fixée par l'utilisateur. Le nombre de bits utiles pour créer notre système dépend des paramètres choisis. Nous feront varier n entre 2 et 8, pour un m environ égal à 10. Dans le cas sans bruit la matrice formée de mots de code reçu est carré de longueur de côté $(m+1)n$.Dans la pratique nous créons le système voulu en créant la matrice dont chaque ligne contient un mot de code. Néanmoins, nous pouvons diminuer le nombre de bits nécessaire en utilisant les propriété des codes convolutifs (cf. figure 3.5). Pour k bits d'entrée nous avons n bits de sortie, ainsi le mot de code garde ses propriétés par décalage de n bits. Ainsi seul la première ligne contient $(m+1)n$ nouveaux bits la où chacun des mots suivants contiendront n nouveaux bits. Dans un milieu bruité, le système créé n'est plus carré mais est une matrice vertical (nombre de lignes plus grand que le nombre de colonne) dans le but d'ajouter de la redondance et ainsi de permettre aux algorithmes de recherche de mots de poids faible de fonctionner, souvent nous prendrons un nombre de ligne égale à trois fois le nombre de colonne.

Cas pratique

Prenons le cas de $n = 2$ et $m < s = 10$ (où s est une borne supérieur de m), si nous considérons le cas sans bruit la matrice formée contient alors $2(10 + 1) = 22$ colonnes et lignes soit, 22 bits pour la première lignes et 21.2 nouveaux bits pour les lignes suivantes. Pour former les système nous avons alors besoin de $22 + 21.2 = 64$ bits consécutifs de donnée. Dans le cas bruité, si nous considérons que la hauteur du système est égal à trois fois la largeur nous avons donc un nombre de colonne inchangée $2(10 + 1) = 22$ et une hauteur de $3(2(10+1)) = 66$. Le nombre de bits nécessaire pour former le système est alors de $22 + 65 * 2 = 152$ bits consécutifs.

Prenons le cas extrême de $n = 8$ et $m < s = 10$. Dans le cas sans bruit nous avons $8(10 + 1) = 88$ colonnes pour autant de lignes, soit un système de $88 + 87 * 8 = 784$ bits. Dans le cas bruité le nombre de ligne est $3(88) = 264$,

pour créer ce système de taille 264×88 nous avons besoin de $88+(263)8 = 2192$ bits consécutifs de donnée.

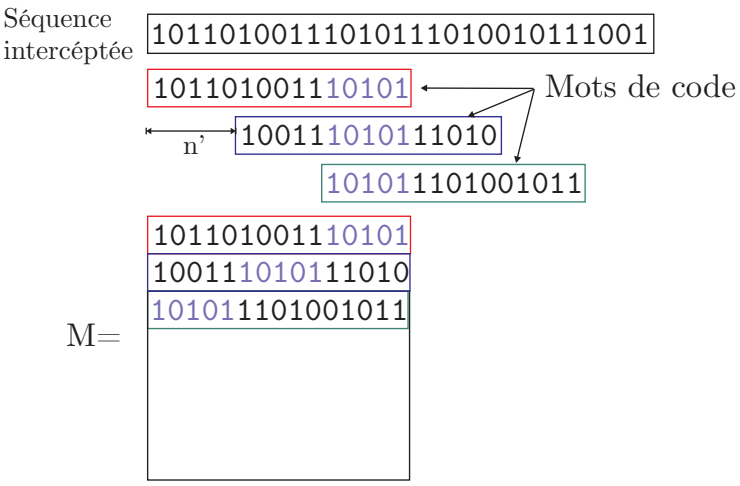


FIGURE 3.5 – Construction du système à partir de la séquence interceptée

On voit sur la figure 3.5 qu'une séquence de $n' = n$ bits peut appartenir à plusieurs mots de code. La taille du système formée est $N_{bsysteme} = n's + (n's - 1)n' = n'(s + n's - 1)$ bits.

Dû au choix de $s \geq m$, le code dual souhaité est inclus dans le code dual engendré par H. A ce stade nous pouvons connaître d'après la proposition 3.25 les valeurs de m et de k.

Remarque : Le pivot est toujours réaliser des degré les plus grand vers les plus petits. L'ordre du pivot dépend donc grandement de l'ordenancement des données sur la matrice de départ, à partir duquel l'ordre de chaque pivot est inversé à chaque nouvelle recherche de dual.

Si l'on procède à un pivot de Gauss nous obtenons une matrice dites "en escalier" de la forme suivante :

$$H_s = \begin{array}{|c|c|c|c|} \hline U_s & & & \\ \hline & U_{s-1} & & \\ \hline & \ddots & & \\ \hline 0 & & U_2 & \\ \hline & & & U_1 \\ \hline \end{array} \tag{3.5}$$

Chaque bloc est ordonné de la façon suivante, le bloc U_s est le bloc supérieur, dont toutes les lignes ne sont pas nulles sur les n dernières positions à droite, chaque séquence U_s est donc une séquence binaire de degré $s-1$. Le bloc U_{s-1} contient toutes les lignes non nulles sauf sur les n dernières positions à droite, ce sont les séquences binaires de degré $s-2$, le bloc U_i contient alors toutes les lignes nulles sur les $s-i$ fois n dernière positions à droite et non nulles ailleurs, comprenant toutes les séquences binaires de degré $s-i-1$.

La valeur k correspond au nombre de lignes de U_s. En effet, les blocs U_i ont un nombre de ligne croissant jusqu'au bloc j système de dimension $n-k$ à

partir duquel tous les blocs U_i pour $i = 1, \ldots, s$ sont composés de $n - k$ lignes. Le bloc U_j est donc le bloc de plus petit degré capable d'engendrer le code dual de $\mathcal{C}$. A l'aide le la proposition 3.25 et du bloc U_j, nous en déduisons la valeur de m.

Le bloc U_j est suffisant pour retrouver une matrice génératrice du code $\mathcal{C}$, la sélection de ce système permet non seulement de retrouver la valeur de m mais aussi de simplifier les calculs qui vont suivre dû à la création de systèmes de taille minimale.

3.2.5 Code convolutif à partir de son dual

Dans cette section nous nous intéressons à partir du code dual observé à retrouver notre code convolutif.

Soit $\mathbf{h}_i = (h_{i,1}(D), \ldots, h_{i,n}(D)), i = 1, \ldots, r$ un ensemble de r générateurs de $\mathcal{C}^{\perp}$, le code convolutif $\mathcal{C}$ est l'ensemble de toutes les séquences $(u_1(D), \ldots, u_n(D)) \in \mathbf{F}_2(D)^n$ telles que :

$$\begin{cases} h_{1,1}(D)u_1(D) + \cdots + h_{1,n}(D)u_n(D) &= 0 \\ & \vdots \quad . \\ h_{r,1}(D)u_1(D) + \cdots + h_{r,n}(D)u_n(D) &= 0 \end{cases} \tag{3.6}$$

Nous écrivons 3.6 pour des séquences polynomiales de $\mathcal{C}$ de degré $< s$, nous obtenons le système linaire sur $\mathbf{F}_2$ suivant

$$\sum_{j=1}^{n} \sum_{\ell=0}^{s-1} u_{j,\ell} h_{i,j,d-\ell} = 0, \begin{cases} 1 \leq i \leq r, \\ 0 \leq d < s + t. \end{cases} \tag{3.7}$$

Où $t = \max_{1 \leq i \leq r}(\deg(\mathbf{h}_i))$. Pour tout $j = 0, \ldots, t$, nous notons $H_{*,j}$ les matrices $r \times n$ suivantes

$$H_{*,j} = \begin{pmatrix} \mathbf{h}_{1,*,j} \\ \vdots \\ \mathbf{h}_{r,*,j} \end{pmatrix}$$

Nous réécrivons l'équation (3.7) dont les solutions forment le noyau de la matrice binaire $(rs + rt) \times (sn)$ suivante

$$S = \begin{array}{|cccc|} \hline H_{*,0} & & & \\ H_{*,1} & H_{*,0} & & 0 \\ & \ddots & \ddots & \\ 0 & & H_{*,t} & H_{*,t-1} \\ & & & H_{*,t} \\ \hline \end{array} .$$

Cette matrice représente simplement la réalisation du produit polynomial 3.7 de manière algébrique. Si l'on regarde la forme de la matrice, cela revient à annuler les termes de degré 0 puis les termes de degré 0 et de degré 1 et ainsi de suite.

Le noyau de S est égal au code binaire $LE_s(\overline{C}_s)$ ($\overline{C}_s$ est le s-ème code binaire associé à $\mathcal{C}$). Nous avons donc trouvé un ensemble générateur de $\mathcal{C}$

Calcul d'une matrice génératrice canonique

Cet ensemble générateur $g_i(D)$ de $\mathcal{C}$ est utilisé pour former une matrice génératrice canonique de $\mathcal{C}$ en utilisant les algorithmes de basicité puis de réduction d'une matrice polynomiale.

3.2.6 Amélioration de notre méthode

L'amélioration de notre méthode réside dans l'utilisation exclusive de l'algèbre linéaire $\mathbf{F}_2$. Nous allons plus loin dans la compréhension des matrices que nous manipulons.

Dans nos algorithmes, nous traitons d'ensembles générateurs d'un code convolutif ou de son dual. En pratique, ces ensembles sont les lignes d'une matrice associée à un code binaire et possèdent une taille supérieur à celle des matrices polynomiales canoniques recherchées. Ces ensembles peuvent être réduits jusqu'à obtention d'ensembles de taille minimum. Les améliorations suivantes sont établies pour $\mathcal{C}$ mais peuvent être appliquées aussi bien à $\mathcal{C}^{\perp}$.

Matrices génératrices associées à des codes binaires

Soit $\mathcal{C}$ un code convolutif (n, k). Pour tout $s > 0$, soient $\overline{\mathcal{C}}_s$ et $\mathcal{C}_s$ respectivement le s-ème sous-code canonique de $\mathcal{C}$ et le s-ème code binaire associé à $\mathcal{C}$.

Nous écrivons la matrice génératrice de $\mathcal{LE}_s(\overline{\mathcal{C}}_s)$ (les degrés les plus petits à droite) sous forme de matrice échelonnée par réduction des lignes (ou en escalier).

$$V_s = \begin{array}{|c|}\hline U_s \\ \hline U_{s-1} \\ \hline \vdots \\ \hline U_2 \quad\quad 0 \\ \hline U_1 \\ \hline \end{array} \tag{3.8}$$

Où, pour $i > 0$, U_i est une matrice binaire $k_i \times (in)$ dont les blocs $k_i \times n$ situés tout à gauche de chaque U_i sont de rang k_i (nous pouvons avoir $k_i = 0$). Nous attirons l'attention sur le fait que toute matrice binaire peut être écrite sous forme de lignes échelonnées avec sn colonnes, dans la forme 3.8. Nous notons k_s le rang des n premières colonnes, et k_{s-1} le rang des n colonnes suivantes après avoir supprimé les premières k_s lignes, etc... L'importance n'est pas d'écrire V_s comme au dessus mais bien de comprendre la signification et l'importance des U_i transcrits dans la proposition suivante.

Proposition 3.27 *pour tout $i = 1, \ldots, s$, soit $\mathcal{D}_i$ l'espace de dimension k_i sur $\mathbf{F}_2$ généré par $\mathrm{LE}_s^{-1}(U_i)$.*

(i) $\forall i, 1 \leq i \leq s, \overline{\mathcal{C}}_i = \mathcal{D}_i \oplus \mathcal{D}_{i-1} \oplus \cdots \mathcal{D}_1$.

(ii) $\forall i, 1 \leq i \leq s, k_i = \dim_{\mathbf{F}_2(D)}(\mathcal{C}_i)$ *et* $\mathcal{C}_i = \langle \mathcal{D}_i \rangle$.

(iii) Il existe une matrice génératrice canonique $k_s \times n$ formée de $k_i - k_{i-1}$ lignes de chaque $\mathrm{LE}_s^{-1}(U_i)$ pour $i = 1, \ldots, s$ (avec la convention $k_0 = 0$).

Preuve :

Rappelons que

$$V_s = \begin{array}{|c|c|c|c|}\hline U_s & & & \\ \hline & U_{s-1} & & \\ \hline & \ddots & & \\ \hline & 0 & U_2 & \\ \hline & & & U_1 \\ \hline \end{array},$$

Avec toutes les matrices binaires U_i de taille $k_i \times (in)$ dont le bloc le plus à gauche de taille $k_i \times n$ est de rang plein.

(i) Pour tout $i \leq s$, les éléments non nul de $\mathcal{D}_i \subset \mathbf{F}_2[D]^n$ sont de degré $i-1$ exactement. Nous avons donc $\mathcal{C}_i = \mathcal{D}_i \oplus \cdots \oplus \mathcal{D}_1$.

(ii) En particulier, nous avons $\overline{\mathcal{C}}_i = \mathcal{D}_i \oplus \overline{\mathcal{C}}_{i-1}$, d'après la Proposition 3.26, nous en déduisons que

$$k_i = \dim_{\mathbf{F}_2}(\mathcal{D}_i) = \dim_{\mathbf{F}_2(D)}(\mathcal{C}_i).$$

les lignes de la matrice polynomiale $k_i \times n$ $F_i = \mathrm{LE}_s^{-1}(U_i)$ forment une base de $\mathcal{D}_i$. La matrice binaire $[T_i]$ contient les coefficients de degré $i-1$ des éléments de la matrice F_i, est la matrice binaire $k_i \times n$ formée du bloc, non nul, le plus à gauche de U_i de rang k_i. On en déduit que F_i est réduite (théorème 3.10) et est donc une matrice génératrice du code convolutif $\mathcal{C}_i$ (théorème 3.15). C'est à dire $\mathcal{C}_i = \langle \mathcal{D}_i \rangle$.

(iii) le premier bloc non vide $G_j = \mathrm{LE}_s^{-1}(U_j)$ est une matrice génératrice polynomiale $k_j \times n$ de G_j. Son nombre de ligne dans chaque bloc respecte les contraintes précédentes.

Nous supposons que nous avons une matrice génératrice polynomiale G_{i-1} de $\mathcal{C}_{i-1}$ qui a été extraite de $U_1, \ldots, U_{i-1}$ avec le nombre de ligne prescrites dans chaque G_i. Le bloc suivant $\mathrm{LE}_s^{-1}(U_i)$ contient k_i lignes et génère $\mathcal{C}_i$. Au moins $k_i - k_{i-1}$ lignes de $\mathrm{LE}_s^{-1}(U_i)$ n'appartient pas à $\mathcal{C}_{i-1}$. Nous ajoutons cette ligne à G_{i-1} pour former une matrice génératrice polynomiale G_i de $\mathcal{C}_i$.

Au final, nous obtenons une matrice génératrice polynomiale G_s de $\mathcal{C}_s$ avec $k_i - k_{i-1}$ lignes de degré $i-1$, avec $i = 1, \ldots, s$. A partir de la Proposition 3.26, cet ensemble de degrés de lignes est le même que pour une matrice génératrice canonique. Le degré externe de G_s est minimal et donc G_s est canonique.

□

Dans notre algorithme de reconstruction, nous avons programmé la forme canonique à partir d'un ensemble de générateurs d'un code convolutif. L'algorithme de Smith nous permet de réaliser ceci, ce qui en pratique nous oblige à implémenter les opérations sur les polynômes. La complexité algorithmique n'augmente pas beaucoup avec l'utilisation de polynômes, au contraire la complexité du programme croit beaucoup. Cependant, en utilisant la proposition 3.27, qui dérive de la description structurelle que nous donnons aux codes convolutifs, nous obtenons une matrice génératrice canonique directement à partir de la forme échelonnée associée au code binaire.

Algorithme 6 Amélioration algorithme

Entrée : une base V (matrice génératrice de $\mathcal{C}$ ou de $\mathcal{C}^{\perp}$) **Sortie :** matrice canonique binaire B de $\mathcal{C}$ ou de $\mathcal{C}^{\perp}$.

1: $B = U_1$ et $k = k_1$
2: **while** $k_{temp} < k$ **do**
3: **if** $k_{i-1} - k_i > 0$ **then**
4: On rajoute à B les $k_{i-1} - k_i > 0$ lignes supplémentaires de $\mathrm{LE}_s^{-1}(U_i)$.
5: $k \leftarrow k + k_{i-1} - k_i$
6: **end if**
7: **end while**
8: On retourne B

De manière plus précise, l'ajout des lignes supplémentaires s'effectue de façon a conserver une matrice canonique. Pour cela nous utilisons la notation d'indice de pivot V_i associé à un bloc U_i correspondant aux indices des colonnes utilisés pour réaliser le pivot ou encore aux colonnes possédant un $\ll 1 \gg$ sur la diagonale. Ainsi au cours de l'algorithme de Gauss nous stockons la liste des pivots utilisés afin de séléctionner les lignes permettant de créer une matrice canonique. Soit une matrice X formée de deux sous-matrices Y et Z de la forme

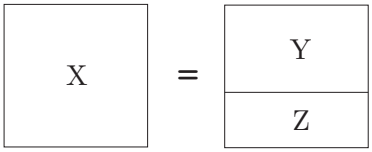


Figure 3.6 – Relation des pivots entre une matrice et ses sous matrices

Alors si les pivots est effectuer dans l'ordre croissants des colonnes, nous admettrons que tous les pivots de la sous matrice Y sont inclus dans les pivots de la matrice X. Un moyen efficace d'ajouter les lignes tout en gardant la canonicité revient à jouter les pivots V_i de U_i non utilisés par les pivot V_{i-1} de U_{i-1} modulo n.

3.2.7 Impact de la synchronisation

Dans toute notre résolution nous avons supposé que la synchronisation était connue. Même si cette information peut être fournie peut dans certains cas être fournie lors de la démodulation ou par l'étude du protocole de communication (paquets), nous nous plaçons dans le cas où seule la séquence codée est connue. Nous devons en premier lieu définir le concept de synchronisation dans le cas des codes convolutifs.

Synchronisation d'un code convolutif

Pour un code convolutif (n, k), nous avons k bits en entrée qui fournissent n bits en sortie. la séquence interceptée se constitue de paquets consécutifs de n bits. Nous appelons donc synchronisation notée Sy la position du premier bit d'un paquet de n. On a donc $Sy = Sy + n$ Tout décalage de n nous redonne une synchronisation.

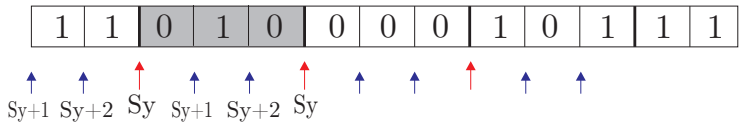


FIGURE 3.7 – Séquence binaire interceptée

La figure 3.7 met en évidence le problème de synchronisation. En gris, un paquet de $n = 3$ bits provenant du (des) même(s) k bits envoyés. Les flèches rouges indiques le débuts de chacun de ces blocs de n bits, les flèches bleues renseignent sur la désynchronisation. Ainsi si l'on choisit une place aléatoire sur le train binaire correspondant à l'interception d'un message en cours, nous avons une probabilité $\frac{1}{n}$ pour que le premier bits de notre séquence soit suivi de n bits provenant des mêmes k bits de codage. La séquence est alors synchronisée.

Impact et détection de la synchronisation

Si $\mathcal{S}$ est la série interceptée, tronqué du degré 0 à $\ell-1$ et est désynchronisée.

Nous réécrivons le systèmes 3.3 dont les solutions sont le noyau de la matrice binaire suivante.

$$S = \begin{array}{|c|c|c|c|} \hline \mathbf{S}_{*,0} & \mathbf{S}_{*,0} & \cdots & \mathbf{S}_{*,s-1} \\ \hline \mathbf{S}_{*,1} & \mathbf{S}_{*,1} & \cdots & \mathbf{S}_{*,s} \\ \hline \vdots & \vdots & & \vdots \\ \hline \mathbf{S}_{*,\ell-s} & \mathbf{S}_{*,\ell-s+1} & \cdots & \mathbf{S}_{*,\ell-1} \\ \hline \end{array} . \tag{3.9}$$

Ce système est toujours vérifié, cependant les $\mathbf{S}_{*,i}$ de taille n sont formés d'une séquence de bits provenant de deux séquences d'entrée différentes. Sy' bits appartiennent à la première séquence d'entrée de k bits et $Sy+n-Sy'$ sont issues de la séquence d'entrée de k bits suivante où Sy est le bit synchronisé et Sy' la désynchronisation.

Ce décalage influe sur la matrice finale qui s'écrit

$$G = \sum_{i=1}^{k} (Dg_{i,n-Sy'}(D), Dg_{i,n-Sy'+1}(D), \ldots, Dg_{i,n}(D), g_{i,0}(D), g_{i,1}(D), \ldots, g_{i,n-Sy'-1}(D))$$

Les polynômes subissent une permutation et une multiplication par des pour les polynômes de $Sy-n \ldots, n$. La multiplication par D peut être une indication permettant de reconnaître la synchronisation car elle a tendance à faire monter la mémoire de contrainte. Cependant, ce n'est pas le cas et illustrons ce fait par un exemple

Exemple : Soit G une matrice génératrice polynomiale canonique d'un code convolutif $(3, 1)$. $G = (1+D^3\ 1+D+D^3\ 1+D+D^2)$ Si nous considérons une séquence codée désynchronisée de 2 bits alors nous identifions la matrice génératrice $G = (D + D^2 + D^3, 1 + D^3\ 1 + D + D^3)$ Cette matrice est toujours canonique mais sa mémoire de contrainte n'a pas augmenté. Nous ne pouvons donc différencier les deux matrices et en déduire la bonne synchronisation.

Nous sommes incapables de différencier la synchronisation par la simple observation des matrices génératrices aléatoires trouvées à partir de différentes synchronisations. De même pour les codes convolutifs (n, k), il est impossible

de trouver la synchronisation avec les matrices génératrices d'autant plus que la multiplication des colonnes par D n'est pas visible sur une matrice $k \times n$.

3.3 Reconnaissance du poinçonnage

Nous avons vu que le poinçonnage peut être considéré comme la succession deux deux étapes, l'étape de regroupement et celle de suppression des colonnes. La forme forme possède une structure forte que nous allons utiliser pour retrouver le poinçonnage. Ce travail a été réalisé en partenariat avec M. Cluzeau et M. Finiasz. Leurs travaux sur les poinçonnages font l'objet d'un article [CF09a].

Dans le but de pouvoir trouver une matrice de passage P de taille $K \times K$ permettant de passer d'une matrice basique réduite quelconque à la matrice du code regroupé, il est nécessaire de mettre en équations la structure que l'on recherche. Notons :

$$M = \begin{bmatrix} A_1(D) & A_2(D) \\ B_1(D) & B_2(D) \\ C_1(D) & C_2(D) \end{bmatrix} \text{ et } Z = \begin{Bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ D & 0 & 0 \end{Bmatrix}$$

Plus généralement, Z est de taille $K \times K$ et se compose d'une sur-diagonale de 1 et d'un D dans le coin inférieur gauche. Alors, la matrice du code regroupé est la concaténation des matrices $Z^2 \times M$, $Z \times M$, M ... Partant d'une matrice basique réduite G, on cherche donc une matrice P telle que :

$$P \times G = \left[\; Z^2 \times M \;\middle|\; Z \times M \;\middle|\; M \; \right]$$

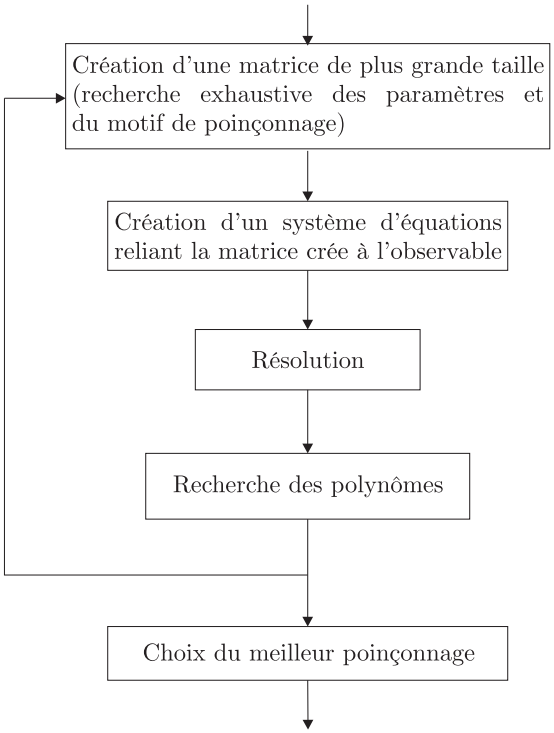
Si l'on note G_i la i-ème colonne de G, on a alors la relation :

$$P \times G_i = Z \times P \times G_{i+n}$$

Comme G et Z sont connus, cette équation correspond en fait à K équations linéaires liant les différents coefficients de P entre eux. Afin de retrouver la matrice de passage P vers la matrice du code regroupé, il suffit donc de combiner les équations linéaires données par l'équation 3.3 pour chaque paire de colonne i et $i+n$ de G. L'ensemble des solutions à ces équations forme un espace vectoriel de matrices de transition P et n'importe quel élément de cet espace nous donne une matrice $P \times G$ qui a la structure d'un code regroupé.

Comment retrouver le poinçonnage d'un code convolutif ?

Nous procédons par recherche exhaustive selon le motif de poinçonnage. Pour tous les motifs de poinçonnages nous cherchons à retrouver la matrice du code regroupées nous permettant d'identifier un code $(n, 1)$ de même longueur de contrainte que le code (N', K). Les étapes suivantes décrivent la manière dont nous procédons.



Création de la matrice

Nous disposons à la fin de l'algorithme de reconnaissance des codes convolutifs, d'une matrice génératrice $K \times N'$. Nous cherchons à trouver une matrice de passage par identification avec la matrice $K \times N (= Mn)$ du code regroupé. Nous procédons par recherche exhaustive sur les paramètres du codeur non poinçonné, $(2,1)$, $(3,1)$,..., $(5,1)$, en faisant varier le n (noté ici n_{test}). Pour chaque codeur, nous effectuons une recherche exhaustive sur le motif de poinçonnage. La longueur maximum du motif de poinçonnage est donné par $L^P_{max} = n_{test}K$.
Le motif de poinçonnage permet d'ajouter des colonnes afin d'obtenir deux matrices de tailles identiques

Exemple : Soit

$$G = \begin{pmatrix} P_{1,1}(D) & P_{1,2}(D) & P_{1,3}(D) \\ P_{2,1}(D) & P_{2,2}(D) & P_{2,3}(D) \end{pmatrix}$$

la matrice génératrice 2×3 observée. Nous considérons un code convolutif $(2,1)$ regroupé deux fois :

$$\overline{G}^{[2]} = \begin{pmatrix} A_1(D) & A_2(D) & B_1(D) & B_2(D) \\ DB_1(D) & DB_2(D) & A_1(D) & A_2(D) \end{pmatrix}$$

Pour un motif de poinçonnage choisi [1011], nous « agrandissons » la matrice G :

$$G_{etendue} = \begin{pmatrix} P_{1,1}(D) & ?? & P_{1,2}(D) & P_{1,3}(D) \\ P_{2,1}(D) & ?? & P_{2,2}(D) & P_{2,3}(D) \end{pmatrix}$$

Les « ? ? » représentent la colonne effacée dont les polynômes nous sont inconnus. Pour un motif de poinçonnage [1110] nous obtenons :

$$G_{etendue} = \begin{pmatrix} P_{1,1}(D) & P_{1,2}(D) & P_{1,3}(D) & ?? \\ P_{2,1}(D) & P_{2,2}(D) & P_{2,3}(D) & ?? \end{pmatrix}$$

Mise en équation Le système d'équations est décrit par l'équation 3.3. Ce système contient M équations pour chaque paire de colonnes liant les coefficients de P matrice de passage. Dans notre exemple précédent, P est une matrice 2×2, $P = \begin{pmatrix} a(D) & b(D) \\ c(D) & d(D) \end{pmatrix}$. D'après l'équation 3.3, nous « voyons » que la première colonne et la troisième on une relation :

$$P \times G_1 = Z \times P \times G_{1+n-3}$$

Nous obtenons deux équations :

$$a(D)P_{1,1}(D) + b(D)P_{2,1}(D) = c(D)P_{1,3}(D) + d(D)P_{2,3}(D)$$

$$c(D)P_{1,1}(D) + d(D)P_{2,1}(D) = Da(D)P_{1,3}(D) + Db(D)P_{2,3}(D)$$

Résolution

Une fois le système précédent écrit, on trouve l'espace des matrices de passage possibles par un simple pivot de Gauss. On conserve une base de dimension k quelconque de cet espace.

Récupération des polynômes

Chaque élément de la base précédemment trouvée est une matrice de passage P tel que $P \times G$ ait la forme d'une matrice regroupée poinçonnée. A partir de ces matrices nous retrouvons pour chaque élément de la base les polynômes associés à la matrice de passage.

On remarque que $Z \times P$ retourne une combinaison linéaire des polynômes. Ainsi, pour les k éléments de la base, nous avons trouvé n polynômes que nous mettons dans une matrice.

$$\begin{pmatrix} g_{1,1}(D) & g_{1,2}(D) \\ g_{2,1}(D) & g_{2,2}(D) \\ \vdots & \\ g_{k,1}(D) & g_{k,2}(D) \end{pmatrix}$$

La matrice ainsi obtenue est rendu canonique. Si pour un poinçonnage la matrice obtenue est telle que $k > 1$ alors plusieurs solutions pour un même poinçonnage sont possibles. Nous retenons les polynômes de longueur de contrainte la plus faible.

Choix du meilleur poinçonnage

Dans le cas d'un code $(n, 1)$, la mémoire de contrainte est le degré du polynôme de plus haut degré. Afin de choisir le meilleur poinçonnage possible pour le décodage, nous devons trouver des polynômes de mémoire de contrainte m la plus petite possible pour obtenir le meilleur décodeur. Dans le cas ou nous savons que la mémoire de contrainte ne varie pas nous pouvons chercher un m

équivalent à la longueur de contrainte du code $K \times N'$ observé. Ces cinq étapes de calcul sont chrono-phages notamment à cause des nombreux pivots de Gauss et formes de Smith à calculer. Pour des poinçonnages de longueur 16 et de poids 9 le temps de calcul est d'environ une minute. Au-delà, la recherche exhaustive montre ses limites sans une puissance de calcul plus importante.

Exemple : Si nous poursuivons l'exemple de reconstruction, nous avons une matrice génératrice 3×4 définie par :

$$\begin{pmatrix} 1 & 1+D+D^2 & D+D^2 & 0 \\ 0 & D+D^2 & D & 1+D+D^2 \\ D+D^2 & D^2 & 1+D^2 & D+D^2 \end{pmatrix}$$

Nous allons donc tester les codes $(2,1),\ldots,(n,1)$ avec $n = 4$, afin de déterminer quel décodeur est le meilleur. Pour chaque décodeur nous testons tous les motifs de poinçonnage, pour un code convolutif (n,k) le nombre de motifs distincts possible est $\binom{K'n}{K'n-N'}$
Pour $n = 2$, $K' = 3$ et $N' = 4$ nous avons donc $\binom{6}{2} = 15$ possibilités de poinçonnage.
Pour $n = 3$, $K' = 3$ et $N' = 4$ nous avons donc $\binom{9}{5} = 126$ possibilités de poinçonnage.
Pour chaque possibilité nous en déduisons les polynômes et conservons seulement ceux de mémoire de contrainte la plus faible. Nous obtenons pour ces codes :

motif de poinçonnage : 011100100
Longueur de contrainte $= 6$
$1+D^2+D^3+D^5+D^6,\ 1+D+D^2+D^4+D^6,\ 1+D+D^2+D^3+D^6$

motif de poinçonnage : 110001001
Longueur de contrainte $= 6$
$1+D^2+D^3+D^5+D^6,\ 1+D+D^2+D^4+D^6,\ 1+D+D^2+D^3+D^6$

Nous obtenons les mêmes polynômes que ceux utiliser pour créer notre séquence codée, cependant 3 motif de poinçonnage sont possibles, parmi eux le bon motif utilisé [110001001].

3.3.1 Exemple complet

Dans cette section, nous allons suivre un exemple réel de déroulement du programme. Pour cela nous allons considérer le codeur convolutif $(3,1,4)$, de polynômes générateurs $1+D+D^3$, $1+D^2+D^3$, $1+D^3$. Nous poinçonnons ce code avec le masque 101010110011 afin d'obtenir un code convolutif poinçonné $(7,4,4)$. Ce code a été choisi de taille moyenne afin de pouvoir faire l'objet d'un exemple, de même pour limiter la taille des matrices, nous choisirons des paramètres légèrement réduits par rapport aux vrais paramètres du code. Notamment $n = 7$, nous avons vérifié qu'aucune fausse solution (code convolutif plus petit) n'a été détectée et $s = 6$ (6 est le degré maximum choisi) là où dans la réalité nous choisissons 15 pour couvrir tous les codes connus dans les normes. Nous considérons un exemple sans erreur, cela ne change en rien l'exemple, la différence minime revient dans la taille de la matrice nécessaire à l'algorithme

de Valembois qui grandit très vite en présence de bruit afin de permettre à l'algorithme de fonctionner (au moins le double en taille).

Voici la séquence de bits interceptés.

```
11010111101110101010001001000101011001101000100100111010110110
00001110110010101100010010110101110011011010001111101010001010
01011001111110101000011101100011111000000001111101100010011 1
10101011100010100000100100100101000000011111011110111100000 0
1
```

FIGURE 3.8 – Séquence de bits interceptée.

Dans la figure 3.8, nous voyons la taille de la séquence nécessaire pour reconnaître le code avec $n = 7$ et $s = 6$ donc $nombre\ bits = n'(s+1) + (n'(s+1) - 1) * n$, nous avons dans une solution lorsque $n' = n$ donc $nombre\ bits = 49 + 48 * 4 = 241$ bits. A partir de cette séquence nous formons une matrice de mots de code.

0100100010110011010100010010001010101110111101011
1010110010010001011001101010001001000101010111011
0001101101011001001000101100110101000100100010101
1101110000110110101100100100010110011010100010010
1010100110111000011011010110010010001011001101010
0010001101010011011100001101101011001001000101100
0101101001000110101001101110000110110101100100100
1100111010110100100011010100110111000011011010110
0010110110011101011010010001101010011011100001101
0111110001011011001110101101001000110101001101110
1000101011111000101101100111010110100100011010100
1101000100010101111100010110110011101101100010001
1111100110100010001010111110001011011001110101101
0001010111110011010001000101011111000101101100111
1101110000101011111001101000100010101111100010110
1111000110111000010101111100110100010001010111110
0000001111100011011100001010111110011010001000101
1111100000000011111000110111000010101111100110100
1000110111110000000011111000110111000010101111100
0111100100011011111000000001111100011011100001010
1110101011110010001101111100000000111110001101110
0101000111010101111001000110111110000000011111000
0010000010100011101010111100100011011111000000001
1001001001000001010001110101011110010001101111100
0000010100100100100000010100011101010111100100011 0
1111100000001010010010010000010100011101010111100
1011110111110000000101001001001000001010001110101
0000111101111011111000000010100100100100000101000
0101100000011110111101111100000001010010010010000
0000011010110000001111011110111111000000001010010 01
1111000000001101011000000111101111011111000000010
0011001111100000000110101100000011110111101111100
0101100001100111100000000110101100000011110111110
1000100010110000110011111000000001101011000000111
0111000100010001011000011001111100000000110101100
0010101011100010001000101100001100111110000000011
1110010001010101110001000100010110000110011111000
1100100111001000101010111000100010001011000011001
0110001110010011100100010101011100010001000101100
0000011011000111001001110010001010101110001000100
1101011000001101100011100100111001000101010111000
1100100110101100000110110001110010011100100010101
1101110110010011010110000011011000111001001110010
1101011110111011001001101011000001101100011100100
0101000110101111011101100100110101100000110110001
0001000010100011010111101110110010011010110000011
0110101000100001010001101011110111011001001101011
1110000011010100010000101000110101111011101100100
0111111111100000110101000100001010001101011110100

FIGURE 3.9 – Matrice de mots de code à partir de la séquence interceptée.

La figure 3.9 montre la matrice M formée de mots de code et met en évidence sa forme caractéristique. Toujours dans un soucis d'écriture de polynômes sous forme binaire, celle-ci est construite de telle sorte que le premier bit de la séquence interceptée se situe en haut à droite et le dernier bit en bas à gauche. Chaque ligne est remplie de droite à gauche pour former un mots. La matrice ainsi formée est une matrice binaire $n(s+1) \times n(s+1) = 49 \times 49$. A partir de cette matrice M, nous cherchons la matrice binaire H tel que $MH = 0$

```
11001000101001000010001010010000100010100111000000
00111010001000010010100010000100101000101100000000
00000110100101000100001001010001000010010100011011
00000001100100010100100001000101001000010010100011
00000000011101000100001001010001000010010100011011
00000000000011010010100010000100101000101100000000
00000000000000110010001010010000100010100111000000
00000000000000001110100010000100101000100001110000
00000000000000000001101001010001000010010100011011
00000000000000000000011001000101001000011111010011
00000000000000000000000111010001000010011001100011
00000000000000000000000011010010100010000111000011
00000000000000000000000000011001000101010101100011
00000000000000000000000000000001110100010110000000
00000000000000000000000000000000011010011001100011
00000000000000000000000000000000000011001001010011
00000000000000000000000000000000000000111010001011
```

FIGURE 3.10 – Matrice génératrice du dual du code $\overline{\mathcal{C}}_s$.

La matrice $\mathcal{H}$ est mise sous forme échelonnée. A partir de ce point nous évaluons k et n par la formule 3.25. Ainsi nous avons une matrice H de taille 18×49, d'où $\dim_{\mathbf{F}_2}(\overline{\mathcal{C}}_s) = 18$. Comme $\dim_{\mathbf{F}_2}(\overline{\mathcal{C}}_s) = sk - m$ si $s \geq d_{\max}(\mathcal{C})$, s est connu égal à 7. Nous cherchons la valeur de m et k. Pour k la matrice nous indique sa valeur, pour cela il suffit de découper la matrice en colonne de n bits. La matrice est ordonnée de façon à lire les degré décroisant de gauche à droite. Ainsi les paquets de n colonnes représentent un degré différent. Nous regroupons les lignes par degré maximum, nous appelons k'_i le nombre de lignes regroupées pour le degré $i \in [0, s-1]$. Lorsque k'_i ne croit plus alors, si $k'_i = k'_{i+1} = k'_{i+j} = k'$ avec $i \geq 0$ et $i+j \leq s-1$ alors $k = k'$. La figure figure 3.11 montre le découpage de la matrice dans le cas présent.

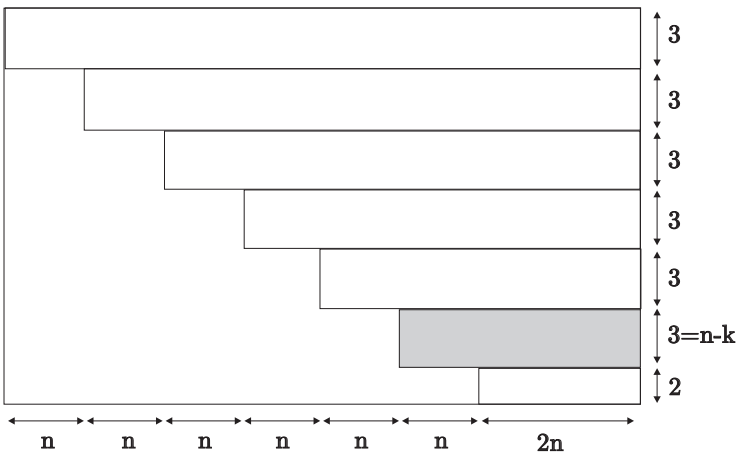


FIGURE 3.11 – Schéma de la forme de la matrice 3.10.

Dans ce cas si aussi nous avons un matrice H de taille 18×49 mais le regroupement est différent. Nous voyons alors bien le changement des k'_i. Comme k vaut 3 nous avons $m = 7.3 - 18 = 3$, les polynômes sont donc de degré 3.

Soit nous considérons tout le code dual observé non optimisé, soit nous prenons un code dual contenant le dual de notre code de plus petite taille dans un souci d'optimisation. Le choix d'une matrice plus courte permet de

générer des systèmes moins complexes, il est néanmoins important de rappeler que l'opération de traitement du bruit à un coût bien supérieur à celui de résolution des systèmes linéaires. Nous considérerons le dual en entier, cependant nous donnons la méthode afin de réduire celui-ci. Si $m = 3$, nous avons besoins des polynômes à $m + 1$ monômes. Si l'on se rapporte au schéma 3.11, ceux-ci sont représentés par le rectangle grisé. Cette matrice binaire (grise sur le schéma) de taille $k \times (m+1)n$ est un générateur de notre code dual noté H'.

Désormais nous recherchons une matrice génératrice du code convolutif à partir de son dual. En choisissant H ou H' nous écrivons le système d'équation définit dans 3.7 en une matrice de la même forme que la matrice 3.2.5. Le dual obtenu, sous forme échelonnée, est donnée par la figure 3.12 ;

```
111001100000000000000000000000000000000000000000000000
110111011000000000000000000000000000000000000000000000
000010000110000000000000000000000000000000000000000000
001101010101000000000000000000000000000000000000000000
110111000100110000000000000000000000000000000000000000
111000010000101100000000000000000000000000000000000000
001101010100000011000000000000000000000000000000000000
000010000000101010100000000000000000000000000000000000
111000010000100010011000000000000000000000000000000000
110111000100001000101100000000000000000000000000000000
000010000000101010000011000000000000000000000000000000
001101010100000000010101010000000000000000000000000000
110111000100001000010001001100000000000000000000000000
111000010000100010000100001011000000000000000000000000
001101010100000000010101000000110000000000000000000000
000010000000101010000000010101010000000000000000000000
111000010000100010000100001000100110000000000000000000
110111000100001000010001000010000101100000000000000000
000010000000101010000000010101000000110000000000000000
001101010100000000010101000000001010101000000000000000
110111000100001000010001000010000100010011000000000000
111000010000100010000100001000100001000010110000000000
001101010100000000010101000000010101000000011000000000
000010000000101010000000010101000000001010101000000000
111000010000100010000100001000100001000010001100100011
```

FIGURE 3.12 – Matrice génératrice binaire

La matrice 3.12 est une matrice binaire H^2, dans l'espace des polynômes, cette matrice est égale à $BE_{m+1}(\frac{1}{D^i})$. Nous appelons G_{gen} la matrice génératrice polynomiale, issue de H^2, de notre code convolutif. Nous allons successivement appliquer l'algorithme de basicité dont le résultat est donné par l'équation 3.3.1 :

$$\begin{pmatrix} 1 & 1+D+D^3+D^5 & 0 & 0 & 1+D^2+D^4+D^6 & D^6 & D^+D^3+D^5+D^6 \\ 0 & D+D^3+D^5+D^7+D^9+D^{11} & 1 & D^{11} & 1+D^2+D^4+D^6+D^8+D^{10}+D^{11} & 0 & 1+D^2+D^4+D^6+D^8+D^{10} \\ 0 & D & 0 & 0 & D^2 & 1+D^2 & 1+D+D^2 \\ 0 & D^2+D^3 & 0 & 1+D & 1+D^2+D^3+D^4 & D^3+D^4 & D+D^2+D^4 \end{pmatrix}$$

puis nous effectuons l'algorithme de réductions sur la matrice $G_{bas}(D)$ et obtenons

$$G = \begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & D & 1 & D & 1+D & 0 & 1 \\ 0 & D & 0 & 1+D & 1 & 1+D & 1+D \\ 0 & 0 & D & 1 & 1 & 0 & 0 \end{pmatrix}$$

La matrice G ainsi formée est une matrice génératrice canonique de taille 4×7 de notre code convolutif poinçonné. Le code reconnu est donc un code convolutif (7,4,4).

Ce procédé nécessite de passer dans le domaine des polynômes pour y faire les calculs de canonicité. L'amélioration de cette méthode permet directement de trouver la matrice canonique à partir de la matrice H^2. Pour cela, nous considérons le même découpage de la matrice H^2 que pour la matrice 3.11. Nous partons du bas et tant que nous n'avons pas atteints un nombre de lignes égal à k nous en ajoutons en remontant. La première ligne est unique 1110011, elle appartient donc à la matrice génératrice.

1110011000000
1101110110000
0000100001100
0011010101010

FIGURE 3.13 – MGP résultat de notre méthode de reconnaissance en utilisant seulement de l'algèbre linéaire

Le deuxième bloc est constitué de quatre lignes, nous ajoutons donc les trois dernières lignes du bloc pour former notre matrice G_{bis}

$$G_{bis} = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1+D & 1+D & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & D & D & 1 & 0 & 0 \\ D & 0 & 1+D & 1 & D & 1 & 0 \end{pmatrix}$$

La matrice G_{bis} est aussi une matrice génératrice polynomiale canonique du code convolutif (7,4,4). Nous faisons remarquer que les matrices G et G_{bis} génèrent le même code et sont différentes ce qui met en évidence la non unicité des matrices génératrices (canoniques) d'un même code.

A partir de cette matrice G (ou G_{bis}) nous allons chercher un poinçonnage. Pour cela nous devons, pour tout motif de poinçonnage, trouver une matrice de passage de notre matrice G (ou G_{bis}) vers une matrice regroupée. Pour cela nous supposons que le code de départ est un code $(n_{bis}, 1)$. La matrice G (ou G_{bis}) est ainsi une matrice génératrice $n \times k$ équivalente à une matrice regroupée $n_{bis}k \times k$ dont on a supprimé des colonnes. Pour un code non poinçonné $(2, 1)$, le masque est un masque de taille 8 et de poids 7. Nous testons ainsi tous les poinçonnages pour des codes $(n', 1)$ avec $n' = 2, \ldots, n'_{max}$. Si pour un motif de

poinçonnage nous obtenons une matrice de passage alors nous avons trouvé un motif de poinçonnage et donc identifié un code poinçonné. Nous retournons le poinçonnage correspondant au code de plus petite longueur de contrainte, qui constitue au décodeur le plus efficace de notre code.

Dans notre exemple nous observons la matrice trouvée

$$G_{nonP} = \begin{pmatrix} 1+D+D^3 & 1+D^2+D^3 & 1+D^3 \end{pmatrix}$$

avec le motif de poinçonnage 101010110011

Nous retrouvons bien le code de départ, de plus la matricée génératrice G_{nonP} possède une indétermination dû à la permutation des polynômes mais sont bien les polynômes utilisés pour former la matrice génératrice canonique du code. Nous avons donc diminué la complexité de décode mais aussi diminué la complexité sur l'indétermination du codeur.

3.4 Simulations et commentaires

Dans ce paragraphe nous mettons en avant les résultats obtenus à l'aide de notre méthode et cherchons à en évaluer les limites et capacités. Premièrement, nous observons tableau 4.2, le résultat de différentes exécutions en milieu bruité. Plusieurs codeurs ont été utilisés pour l'obtention de ces résultats, néanmoins tous possèdent une longueur de contrainte de 9. Le codeur $(2,1)$ est formé des deux polynômes suivants $G = (1+D^2+D^3+D^4+D^8, 1+D+D^2+D^3+D^4+D^7+D^8)$. La matrice du codeur $(4,1)$ est $G = (1+D+D^2+D^3+D^4+D^6+D^8, 1+D+D^3+D^4+D^5+D^8, 1+D^2+D^5+D^7+D^8, 1+D^3+D^4+D^5+D^7+D^8)$. Les autres codeurs sont obtenus par poinçonnage du code $(2,1)$ afin de générer des codes de plus grande taille. Le motif de poinçonnage [111010] permet d'obtenir le code $(4,3)$, alors que pour générer le code $(7,6)$ le motif [1101011001] est utilisé. Nous utilisons dans l'exemple une version de l'algorithme utilisant l'algorithme de la forme de Smith.

	taux d'erreur binaire dans la séquence observée.								
(n,k)	0	0.1%	0.15%	0.3%	0.5%	1%	1.5%	3%	5%
$(2,1)$	0.3	0.3	0.3	0.3	0.4	0.8	2	24	360
$(4,1)$	3	3	3	3	3	3	3	6	240
$(4,3)$	1.4	3	3	6	10	50	270	–	–
$(6,5)$	4	7	10	30	110	920	–	–	–
$(8,7)$	9	15	19	57	340	–	–	–	–

temps moyen d'exécution en millisecondes (processeur Core2 3 Ghz Pentium)

TABLE 3.14 – Performances pour reconstruire des codes (n,k) de degré 8

Un code de petits paramètres permet d'atteindre des taux d'erreur plus élevés qu'un code de paramètres plus grand mais de même longueur de contrainte. Le tableau 4.2 montre la grande rapidité de l'algorithme pour les petits codes, il a été présenté au cours de la conférence d'ISIT 2009 ([CS09]). Au cours de cette présentation seule la méthode de Gauss avait été implémenté. Afin de voir le gain d'une méthode tout algébrique nous reprenons les mêmes tests et regardons le gain de notre nouvelle méthode. Nous mettons ici en avance l'amélioration de

notre méthode par rapport à l'état de l'art aussi bien en améliorant la vitesse d'exécution pour retrouver un code convolutif qu'en permettant de reconnaître un code convolutif pour des taux d'erreur plus élevés.

	taux d'erreur binaire dans la séquence observée.								
(n,k)	0	0.1%	0.15%	0.3%	0.5%	1%	1.5%	3%	5%
$(2,1)$	10%	15%	20%	12%	2%	0%	0%	0%	0%
$(4,1)$	8%	6%	5%	15%	3%	3%	3%	0%	0%

TABLE 3.15 – Amélioration de l'algorithme algébrique par rappor a l'algorithme utilisant la méthode de Smith pour reconstruire des codes (n,k) de degré 8

L'utilisation de la méthode de Gauss plutôt que la version amélioré de l'algorithme n'influence pas les résultats pour un train binaire bruité, en présence d'erreurs, l'algorithme de Valembois permettant de retrouver le dual du code sans erreur possède un temps d'execution nettement supérieur à l'exécution de l'algorithme sans erreur (cf tableau).

3.4.1 Poinçonnage

Cet algorithme a été testé en pratique sur des séquences codées poinçonnées et il en ressort les points suivants :

- Presque tous les poinçonnages donnent une solution. Chaque solution a une longueur de contrainte supérieure (et beaucoup plus élevée) ou égale à la meilleure solution. Pour l'instant il est impossible de prédire si un poinçonnage donnera une bonne solution (longueur de contrainte faible) ou non.
- Sur les exemples testés, il arrive que plusieurs poinçonnages donnent des résultats aussi bons (la même longueur de contrainte). Le plus souvent, deux bons poinçonnages ont des motifs quasi-équivalents (par décalage cyclique d'un ou plusieurs bits du motif).
- En général, notre algorithme se comporte bien et retrouve le code sous-jacent sans difficulté particulière (la bonne solution est dans les solutions possibles).

Chapitre 4

Reconnaissance des turbo-codes en milieu bruité

4.1 Méthode de reconnaissance des turbo-codes

Il existe plusieurs types de turbo-codes, parmi eux les turbo-codes parallèles et les turbo-codes séries représentent les deux grandes constructions de turbo-codes. Chaque turbo-code comprend un entrelaceur et composé de deux codeurs convolutifs en général de petite longueur de contrainte.

Le problème que nous chercherons à résoudre dans cette section consiste à reconnaître un turbo-code parallèle de taux $\frac{1}{3}$, composé de codeurs convolutifs systématiques, à partir de l'observation d'une séquence bruitée. Nous présenterons notre méthode mise au point par N. Sendrier et moi même, soumise à ISIT 2010 [CS], et la confronterons à deux méthodes développées par M. Finiasz, M. Cluzeau et J.P. Tillich.

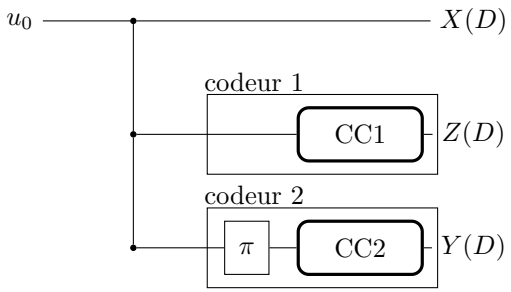


FIGURE 4.1 – Schéma d'un turbo-code parallèle

Dans la section 3, nous nous sommes efforcé de reconnaître tous types de codes convolutifs. Dans ce cas précis, nous possédons une information supplémentaire, la forme systématique du codeur impose la forme de la matrice systématique

(paragraphe 1.6)

$$G_{sys} = \left(\; I_k \; \middle| \; P \; \right)$$

Ainsi, nous ne reconnaissons pas seulement le code convolutif mais aussi le codeur systématique qui lui est associé. Cette méthode n'est pas répétable pour reconnaître le deuxième codeur car l'information $X(D)$ observée ne correspond pas à l'information du deuxième codeur mais à l'information permutée par la permutation π. Les méthodes suivantes permettent de résoudre en partie le problème de reconnaissance du second code convolutif du turbo-code. Pour cela nous faisons certaines hypothèses faibles qui peuvent facilement être vérifiées ou connues à l'aide d'autres étapes de la chaîne de transmission comme la démodulation.

- Les séparations du train binaire formant les séquences $X(D)$, $Z(D)$ et $Y(D)$.
  En effet, il existe dans les normes plusieurs types de « paquets », c'est à dire de façons d'agencer les données afin de les envoyer au destinataire. Certains problèmes existent comme le rajout de « bits de bourrage » permettant de remettre le registre du codeur convolutif dans un état initial, de compléter un paquet de données incomplet ou l'ajout de séquences de synchronisation. Nous supposons connu le protocole utilisé et disposons des séquences $X(D)$, $Z(D)$ et $Y(D)$.
- Le codeur convolutif 1 est retrouvé par la méthode de la section 3.
- La taille n de l'entrelaceur π est supposée connue.

4.1.1 Méthode 1 : méthode statistique et de correction d'erreurs

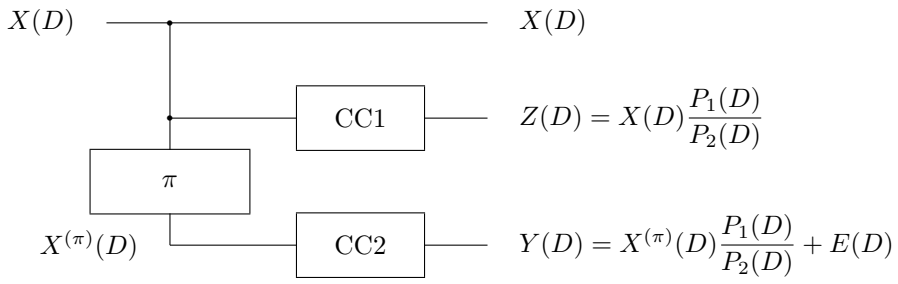
En utilisant la méthode de reconstruction des codes convolutifs décrite dans la section 3, nous faisons l'hypothèse qu'en utilisant la sortie d'information $X(D)$ et la première sortie de redondance $Z(D)$, nous reconnaissons le codeur convolutif systématique $CC1$. Nous allons traiter le problème de reconnaissance des turbo-codes cadre simplifié dans un premier temps, puis nous traiterons le cadre général par la suite.

Hypothèse 4.1

- *H1 : Dans un premier temps, nous supposerons que le codeur convolutif $CC1$ à été reconnu et que toutes les erreurs sur la séquence d'information $X(D)$ et sur la séquence de redondance $Z(D)$ ont été corrigées. De plus, nous ferons l'hypothèse que les deux codeurs convolutifs sont identiques et donc connus.*
- *H2 : Les deux codeurs ne sont pas toujours identiques. Pour cette raison nous devons être capable de reconnaître le deuxième codeur convolutif. Ce problème sera traité dans un deuxième temps.*
- *H3 : Enfin, dans une troisième partie, nous supposerons que les erreurs sur la séquence d'information $X(D)$ n'ont pas toutes été corrigées par le premier codeur convolutif $CC1$.*

4.1.1.1 Contexte où la séquence d'information est dépourvue de bruit

Dans cette partie nous considérons les trois hypothèses H1, H2 et H3. Nous obtenons le schéma suivant :



Nous notons $X^{(\pi)}(D)$ le polynôme défini par :

$$X^{(\pi)}(D) = X_{\pi(0)} + X_{\pi(1)}D + \cdots + X_{\pi(n-1)}D^{n-1}$$

$$Y(D) = X^{(\pi)}(D)\frac{P_1(D)}{P_2(D)} + E(D)$$

Si nous supposons que les polynômes qui engendrent le code convolutif possèdent un monôme de degré zéro non nul (ce qui est le cas en pratique)

$$P_1(D) = 1 + \cdots + P_{1,m}D^m \quad (4.1)$$
$$P_2(D) = 1 + \cdots + P_{2,m}D^m \quad (4.2)$$

on note $g(D)$ la série correspondant à $\frac{P_1(D)}{P_2(D)}$ et on note $g(D) = \sum g_i D^i$ avec $g_0 = 1$ alors la série possède un terme constant non nul.

Donc, il existe une forte corrélation entre les termes de degré zéro de $X^{(\pi)}(D)$ et de $Y(D)$. Notre méthode repose sur l'hypothèse forte mais réaliste que les polynômes $P_1(D)$ et $P_2(D)$ possèdent un monôme de degré zéro. Seules les sorties $X(D)$ et $Y(D)$ sont utilisées pour la reconnaissance du deuxième codeur convolutif et de la permutation.

Calcul de la permutation

Afin de retrouver la permutation nous devons disposer d'un grand nombre de mots de code. Considérons un tel mot de code et les $X(D)$ et $Y(D)$ associés

$$X(D) = X_0 + X_1 D + \cdots + X_n D^{n-1}$$
$$Y(D) = Y_0 + Y_1 D + \cdots + Y_n D^{n-1}$$

Le principe de la méthode est alors d'utiliser l'équation $Y(D) = X^{(\pi)}(D)\frac{P_1(D)}{P_2(D)} + E(D)$ et d'identifier les termes constants de gauche et de droite en remarquant que le terme constant de gauche est égal à Y_0 et le terme constant de droite est égale à $X_{\pi(0)}$ quand E_0 est nul.

Nous décidons par vote majoritaire le choix de $i_0 = \pi(0)$ pour lequel $X_{i_0} = Y_0$ le plus souvent lorsque nous parcourons tous les mots de code observés.

$$\begin{aligned} Y(D) &= X^{(\pi)}(D)\frac{P_1(D)}{P_2(D)} + E(D) \\ Y_0 + Y_1 D + Y_2 D^2 + \cdots + Y_{n-1} D^{n-1} &= (X_{\pi(0)} + X_{\pi(1)} D + \cdots + X_{\pi(n-1)} D^{n-1}) \\ &\quad \times (1 + g_1 D + \cdots + g_{n-1} D^{n-1}) + E(D) \\ Y_0 + D(Y_1 + \cdots + Y_{n-1} D^{n-2}) &= X_{\pi(0)} + D(X_{\pi(1)} + g_1 X_{\pi(0)} \\ &\quad + \cdots + g_1 X_{\pi(n-2)} D^{n-3}) + E(D) \end{aligned}$$

par identification nous avons $X_{\pi(0)} = Y_0$. Pour un nombre de couples assez grand un indice i_0 est largement supérieur aux autres. La corrélation entre les bits est vraie seulement pour le premier terme de $Y(D)$ et de $X_{\pi(0)}$. Pour que la corrélation existe pour le deuxième bit de $Y(D)$, le premier doit être nul. De façon générale, pour que nous puissions avoir une corrélation au rang i entre Y_i et $X_{\pi(i)}$ la séquence de redondance $Y(D)$ doit être nulle jusqu'au rang $i-1$. Nous avons les coefficients de $Y(D)$ de la manière suivante :

$$\begin{aligned} Y_0 &= X_{\pi(0)} + E_0 \\ Y_1 &= X_{\pi(1)} + E_1 + X_{\pi(0)} g_1 \\ &\vdots \\ Y_i &= X_{\pi(i)} + E_i + \sum_{j=0}^{i-1} X_{\pi(j)} g_{i-j} \end{aligned}$$

à la i-ème étape nous avons

$$Y_i - \sum_{j=0}^{i-1} X_{\pi(j)} g_{i-j} = X_{\pi(i)} + E_i \tag{4.3}$$

et donc $X_{\pi(i)}$ est corrélé avec $Y_i - \sum_{j=0}^{i-1} X_{\pi(j)} g_{i-j}$. Dans le contexte décrit, l'erreur se situe seulement sur la sortie du deuxième décodeur. Lors de l'ajustement de chaque couple $(X(D), Y(D))$ si $X_{\pi(r)} \neq Y_r$ nous détectons l'erreur. Celle-ci est située sur $Y(D)$ et Y_r devient égal à $X_{\pi(r)}$.

D'après l'équation 4.3, nous somme capable de calculer la corrélation à l'étape i. Le calcul à chaque étape peut s'avérer plus complexe qu'un ajustement à chaque étape. Nous allons voir ce phénomène dans l'étude du nombre de couples nécessaires pour la reconnaissance du turbo-code.

Statistique du nombre de couples nécessaires

Nous calculons à l'étape une la probabilité que l'indice $i_0 = \pi(0)$ soit bien le bon indice observé et non une erreur due au bruit. Nous calculons la position i_0 par statistiques, en additionnant ou soustrayant le message d'information de longueur M au message de la partie codée. Nous considérons deux variables aléatoires ν et ρ suivantes :

- ν est le nombre de correspondances exactes entre deux bits sur la bonne position : $\nu = \left\{ i : Y_0^{(i)} = X_{\pi(0)}^{(i)} \right\}$

– ρ est le nombre de correspondance exacte entre deux bits sur une position quelconque j : $\rho = \left\{ i : Y_0^{(i)} = X_{\pi(j)}^{(i)}, j \neq 0 \right\}$

Notons que

– ν suit une loi binomiale de paramètres (M, p)
– ρ suit une loi binomiale de paramètres $(M, \frac{1}{2})$

$$\begin{aligned} P(\nu = i) &= \binom{M}{i}(1-p)^i(p)^{M-i} \\ P(\rho = i) &= \frac{1}{2^M}\binom{M}{i} \end{aligned}$$

La probabilité de succes totale correspond à retrouver la bonne permutation à chaque étape et est définie par :

$$P_{all-succes} = \prod_{l=0}^{n-1} P_{succ}(\ell)$$

La probabilité de succes au rang i est calculée en considérant que pour ν est vérifiée pour $Y_i = X_{\pi(i)}$ et ρ est vérifiée pour les $n - i$ positions non retrouvés dans la permutation, sachant que toutes les probabilités de succes au rang $1, \ldots, i$ sont vérifiées.

$$\begin{aligned} & \text{A l'étape 0 nous avons :} \\ P_{succ}(0) &= \sum_{i=0}^{M} P(\nu = i)P(\rho < i)^{n-1} \\ & \text{A l'étape 1 nous avons :} \\ P_{succ}(1) &= \sum_{i=0}^{M} P(\nu = i)P(\rho < i)^{n-2} \\ &\vdots \\ P_{succ}(\ell) &= \sum_{i=0}^{M} P(\nu = i)P(\rho < i)^{n-\ell-1} \\ & \text{La probabilité de succès de l'étape } \ell \end{aligned} \tag{4.4}$$

On a l'expression suivante

$$\begin{aligned}
P_{succ}(k) &= \sum_{i=0}^{M} P(\nu = i) P(\rho < i)^{n-k-1} \\
&= \sum_{i=0}^{M} P(\nu = i) \left(\sum_{j=0}^{i-1} P(\rho = j) \right)^{n-k-1} \\
&= \sum_{i=0}^{M} \binom{Mq}{i} (1-p)^i (p)^{M-i} \left(\sum_{j=0}^{i-1} \binom{M}{j} \left(\frac{1}{2}\right)^i \left(\frac{1}{2}\right)^{M-i} \right)^{n-k-1} \\
&= \sum_{i=0}^{M} \binom{M}{i} \left(\frac{1-p}{p}\right)^i (p)^M \left(\sum_{j=0}^{i-1} \binom{M}{j} \frac{1}{2^M} \right)^{n-k-1} \\
&= \frac{(p)^M}{2^{M(n-1)}} \sum_{i=0}^{M} \binom{M}{i} \left(\frac{1-p}{p}\right)^i \left(\sum_{j=0}^{i-1} \binom{M}{j} \right)^{n-k-1} \qquad (4.5)
\end{aligned}$$

Soit $\mathcal{S}_{n,p}(\epsilon)$ la taille minimum de l'échantillon pour laquelle la probabilité de succès est supèrieure $1 - \epsilon$.

$$M \geq \mathcal{S}_{n,p}(\epsilon) \Rightarrow P_{all-succes} \geq 1 - \epsilon$$

probabilité d'erreur ϵ	taux d'erreur binaire p du canal					
	0.01	0.02	0.05	0.1	0.2	0.3
$\epsilon = 0.5$	30	34	45	65	134	326
$\epsilon = 0.01$	38	43	57	84	175	429

TABLE 4.2 – Taille nécessaire des données $\mathcal{S}_{n,p}(\epsilon)$ pour $n = 5000$

Le tableau 4.2 donne le nombre de données nécessaires pour retrouver la permutation avec une probabilité de succès de 0.5 et de 0.99 pour une séquence d'information non erronée. Le nombre de données nécessaire est faible même pour des taux d'erreurs élevés. Nous n'avons pas besoin d'une séquence beaucoup plus longue pour une probabilité de succès de 99% que pour une probabilité de succès de 50%.

Il y a n étapes, à l'étape l nous devons calculer $n - l$ fois l'expression 4.4, qui a un coût de calcul proportionnel à l, pour chacun des M blocs. Le coût de calcul est alors proportionnel à :

$$\sum_{l=1}^{n} Ml(n-l) \approx \frac{1}{6} Mn^3$$

opérations élémentaires. Nous pouvons réduire ce coût en ajustant a chaque étape les blocs. A la fin de la i-ème étape ($i < l$), nous soustrayons $X_i(D)\frac{P_1(D)}{P_2(D)}$ à $Y(D)$. le coût de calcul devient proportionnel à :

$$\sum_{l=1}^{n} M(n-l) \approx \frac{1}{2} Mn^2$$

Dans le cas où la séquence d'information est aussi bruitée, le nombre de blocs décroît assez rapidement lors de la reconnaissance et les données viennent à manquer pour retrouver notre turbo-code. Deux solutions permettent de palier à cela. La première consiste à prendre plus de données mais le volume devient très vite contraignant pour le stockage et dans le contexte d'interception. Une autre solution est de corriger les erreurs limitant ainsi la perte de données et assurant la faisabilité de notre méthode.

Algorithme 7 Algorithme de reconnaissance de turbo-codes

Entrée : Une séquence binaire non bruitée de mots de code tel que pour le $i$ème mots nous ayons le couple C_i=($X(D)$,$Y(D)$)
Sortie : Un vecteur k contenant la permutation du turbo-code

$X_{corr}(D) = 0$, polynôme à coefficients entiers.
Un vecteur k de taille n initialisé à zéro.
for i de 0 à $n-1$ **do**
  for j de 1 à M **do**
    if $Y_0 = 1$ (correspondant au mot de code C_j) alors **then**
      $X_{corr}(D) = X_{corr}(D) + X(D)$.
    else
      $X_{corr}(D) = X_{corr}(D) - X(D)$.
    end if
  end for
  On détermine l'indice k_i qui est le degré du polynôme X_{corr} possédant le coefficient le plus grand.
  for j de 1 à M **do**
    $Y(D) = Y(D) - X_{k_i}(D)\frac{P_1(D)}{P_2(D)}$
  end for
end for

4.1.1.2 Calcul du développement de $\frac{P_1(D)}{P_2(D)}$

Il est souvent utilisé dans les normes deux codeurs convolutifs identiques, ainsi le développement en série entières de $\frac{P_1(D)}{P_2(D)}$ serait connu à partir du codeur convolutif 1. Cependant, cela n'est pas toujours le cas et il est essentiel de considérer le cas où les deux codeurs sont différents. Dans cette partie nous considérons les hypothèses H1 et H3 seulement. La seule connaissance du développement de $\frac{P_1(D)}{P_2(D)}$ en série entière provient de notre hypothèse selon laquelle le premier terme vaut 1. A chaque étape de la reconstruction de la permutation, nous reconstruisons aussi le développement de $\frac{P_1(D)}{P_2(D)}$. Pour cela, on calcule à chaque étape la valeur de Y'_r, terme de rang r de $Y(D)$ dont les $r-1$ premiers termes ont été mis à zéro. Nous disposons d'un développement de $\frac{P_1(D)}{P_2(D)}$ au degré r.

$$\frac{P_1(D)}{P_2(D)} = g_0 + g_1 D + \cdots + g_{r-1}D^{r-1} + g_r D^r,\ g_0 = 1 \text{ et } g_r \in \{0,1\}$$

et

$$Y'_r \stackrel{\text{def}}{=} Y_r - \sum_{i=0}^{r} g_{r-i} X_{\pi(i)}$$

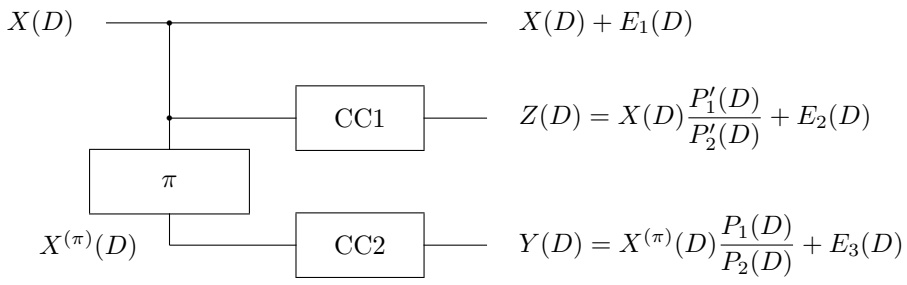
Soit $A_{l,\epsilon} = \left\{ \sharp X_e^{(i)} : Y_1 = X_l + E_1 + X_{\pi(0),\epsilon} \right\}$ avec $\epsilon = 0, 1$. La valeur maximale entre $A_{l,0}$ et A_{l_1} permet de déterminer à chaque étape $\pi(1) = l$ et $g_1 = \epsilon$.

$$\begin{aligned} Y_1 &= X_{\pi(1)} + E_1 + X_{\pi(0)} g_1 \\ &\vdots \\ Y_i - \sum_{j=1}^{i-1} X_{\pi(j)} g_{i-j} &= X_{\pi(i)} + E_i + X_{\pi(0)} g_i \end{aligned}$$

A chaque étape (ci-dessus) nous reconstruisons la permutation mais décidons aussi du terme du développement $(g_1, \ldots, g_i)$ de $\frac{P_1(D)}{P_2(D)}$. Les polynômes $P_1(D)$ et $P_2(D)$ sont retrouvés par algorithme de Berleckamp-Massey à partir de la séquence $\frac{P_1(D)}{P_2(D)}$ obtenue à la fin de notre méthode. Notons m le maximum de degré de $P_1(D)$ et de $P_2(D)$. Si m est le degré du polynôme de plus haut degré, nous pouvons nous arrêter au rang $2(m+1)$. La longueur de la série entière est suffisante pour que l'algorithme de Berleckamp-Massey nous renvoie les polynômes $P_1(D)$ et $P_2(D)$. A partir des deux polynômes, nous pouvons aisément retrouver le développement $\frac{P_1(D)}{P_2(D)}$ jusqu'au rang $2(m+1)$. Connaissant le développement à partir du rang $2(m+1)$, il nous reste juste à retrouver la permutation comme vu précédemment.

4.1.1.3 Contexte dans lequel la séquence d'information contient des erreurs

L'utilisation du turbo-code permet de corriger des taux d'erreur plus élevés qu'un simple code convolutif ne pourrait corriger. Nous considérons maintenant toutes les sorties du turbo-code sont bruitées (hypothèse H1 uniquement). Considérons le schéma de turbo-code suivant. Nous ne considérons plus l'hypothèse H1 dans ce paragraphe.



L'incapacité de reconnaître où se situe l'erreur nous impose de supprimer chaque couple $(X(D), Y(D))$ erroné afin de ne pas fausser les statistiques. La quantité nécessaire à la reconnaissance n'en devient que plus importante. Nous allons étudier le nombre de données nécessaires pour une tel méthode.

Traitement de l'erreur

En présence d'erreurs, la reconnaissance de l'entrelaceur ainsi que des polynômes du codeur convolutif ne change pas. Le principal changement vient de l'incapacité à reconnaître d'où vient l'erreur dans le cas où $X_{\pi(r)} \neq Y'_r$. Il est impossible de corriger l'erreur « à la volée ». Deux solutions sont possibles, soit nous supprimons les données contenant une erreur. Dans ce cas le nombre de données nécessaire augmente de façon exponentielle en fonction du bruit. L'autre solution est de corriger les erreurs dès que possible en utilisant un système de quarantaine. L'idée est alors de corriger l'erreur, lorsque cela est possible, plusieurs étapes après. Cette méthode s'apparente à du décodage non optimum. Pour cela on considère l'équation :

$$\begin{aligned} Y(D) &= G(D)X(D) \\ \sum_{i=0}^{n} y_i D^i &= (\sum_{j=0}^{n} g_j D^j)(\sum_{l=0}^{n} x_l D^l) \end{aligned}$$

Par identification nous pouvons écrire toutes les relations

$$\begin{aligned} y_0 &= x_0 \\ y_1 &= g_1 x_{\pi(0)} + x_{\pi(1)} \\ y_2 &= g_2 x_0 + g_1 x_{\pi(1)} + x_{\pi(2)} \\ &\vdots \\ y_k &= \sum_{i=0}^{k} g_{k-I} x_{\pi(i)},\ g_0 = 1 \end{aligned}$$

La première équation ne peut pas être utilisée car on ne peut pas déterminer lequel des bits est faux. Toutes les autres équations font intervenir un bit de l'équation recherchée $x_{\pi(0)}$ et des bits successifs ($x_{\pi(i)}$ ou y_i avec $i > 0$). Tout couple $(X(D), Y(D))$ contenant une erreur est mis en quarantaine en attendant une étape ultérieure permettant de corriger l'équation. Ainsi nos statistiques se font uniquement sur des couples sans erreur. Nous devons choisir quelles équations utiliser pour corriger notre erreur. Cependant, nous ne pouvons pas toutes les prendre car la quantité de données stockées serait trop importante et que plus l'équation fait intervenir de bits suivant l'indice de notre erreur plus la probabilité de faire intervenir des bits erronés et donc de faussé notre équation augmente. Nous choisissons donc d'utiliser l'équation :

$$y_{j+i} = g_{j+i} x_{\pi(j)} + x_{\pi(j+i)}$$

où

$$\frac{P_1(D)}{P_2(D)} = 1 + D^i + \dots$$

On limite ainsi le nombre de bits nécessaires à la correction pour éviter d'introduire d'autres erreurs.

- Si y_j ou $x_{\pi(j)}$ est erroné et y_{j+i} et $x_{\pi(j+i)}$ sont justes alors nous corrigeons l'erreur. Nous sortons l'équation de la quarantaine, elle est ajusté jusqu'au rang $j + i$ si elle ne contient pas d'autre erreur et sera réutiliser pour le calcul de statistiques.

- Si y_j ou $x_{\pi(j)}$ est erroné et y_{j+i} ou $x_{\pi(j+i)}$ est faux alors nous propageons l'erreur. Nous sommes incapables de la corriger , dans ce cas nous ignorons définitivement l'équation (pour les statistiques et la correction de celle-ci).
- Si y_j et $x_{\pi(j)}$ sont erronés, nous ne détectons pas l'erreur au rang j. celle-ci est détecté à un rang ultérieur mais ne pourra être corriger, nous nous retrouvons dans le cas précédent et ignorons l'équation.

De manière plus précise, nous prenons la bonne décision si $y_{j+i} - x_{\pi(j+i)} = 0$ avec la probabilité $2p - p^2$. Si la décision est fausse alors nous avons un bloc erroné mais pas détecté comme tel. En effet, les blocs faux apparaissent avec la probabilité p^2 à chaque étape. Dès qu'ils se comportent de manière aléatoire, ils doivent rapidement être envoyés en quarantaine. A chaque étape $2p$ blocs sont envoyés en quarantaine et une proportion de $2p$ blocs sont corrigés de la quarantaine et réintroduit dans nos statistiques. Si un bloc se retrouve trop souvent en quarantaine, celui-ci à de forte chance de correspondre à un bloc dont l'erreur est propagée. Nous pouvons par des heuristiques simples les supprimer pour nos calculs. Nous n'avons pas réalisé d'analyse précise de notre algorithme mais les informations précédentes suggèrent que le nombres de blocs va décroître d'un facteur $\frac{1}{(1-4p^2)^n}$ à chaque étape. En pratique cela ne suffit pas à retrouver la permutation pour de large taille de blocs. Par exemple, pour $n = 5000$ et $p = 0.01$ nous avons $\frac{1}{(1-4p^2)^n} \approx 7$. Encore, pour de large taille de bloc, lorsque la probabilité d'erreur du canal augmente, la diminution du nombre de blocs a chaque étape devient inacceptable. Par exemple, pour $n = 5000$ et $p = 0.05$, nous obtenons $\frac{1}{(1-4p^2)^n} \approx 7.10^{21}$

Comme nous l'avons vu pour le cas simple où seule la séquence redondante est bruitée, la complexité est linéaire en la longueur des données et quadratique en la taille de bloc. Dans le cas présent, la taille des données décroît à chaque étape, la taille initiale doit donc être plus grande que nécessaire pour distinguer le plus de positions corrélées. Pour calculer ces corrélations, nous avons seulement besoin d'une petite partie des données, toutefois toute la séquence de données est examinée (et aussi corrigée ou mise en quarantaine) à chaque étape. Si nous ciblons une probabilité d'erreur de ϵ, dépendant de la stratégie utilisée soit mise « en quarantaine » soit « correction », la complexité de calcul est respectivement proportionnelle à :

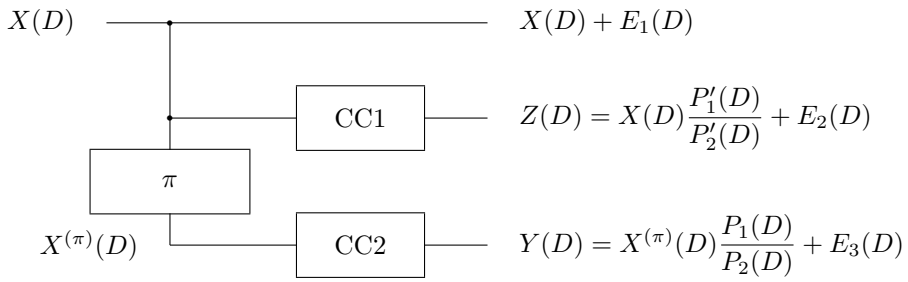
$$\frac{\mathcal{S}_{n,p}(\epsilon)n^2}{(1-2p)^n} \text{ et } \frac{\mathcal{S}_{n,p}(\epsilon)n^2}{(1-4p^2)^n}$$

où $\mathcal{S}_{n,p}(\epsilon)s$ la taille minimum de l'échantillon pour laquelle la probabilité de succès est $\geq 1 - \epsilon$.

Cette méthode basée sur les statistiques est très rapide ce qui en fait son plus grand avantage. Ses deux principaux inconvénients sont les données nécessaires et l'échec de la méthode dans le cas poinçonné. En effet, la perte de donnée dans le pire des cas peut être très rapide car nous rejetons le mot dans le cas d'une impossibilité de corriger. Dans le cas où toutes les séquences sont bruités, nous pouvons exécuter le programme seulement pour des petites tailles de blocs ($n = 100$) et un taux d'erreur binaire du canal petit (1% ou 2%). De plus, dans le cas poinçonné, les bits de redondance et d'information ne sont plus liés.

4.1.2 Méthode 2 : méthode basée sur le décodage

Les méthodes suivantes sont étudiées en parallèles de nos travaux par M. Finiasz, M. Cluzeau et J.P. Tillich [CFT10]. En voici une brève déscription. Reprenons le turbo-code étudié :



Nous avons vu dans la méthode précédente que la présence d'erreur est un facteur limitant de la méthode, en effet nous perdons très vite les blocs de données rendant impossible de retrouver l'entrelaceur. Pour contrer cet effet du aux erreurs nous les corrigeons en partie à l'aide d'équations de parité liant les bits d'informations aux bits de redondance. Cette méthode corrige les erreurs de manière simpliste en s'approchant d'un système de décodage. Dans cette nouvelle méthode, nous allons nous inspirer des algorithmes de décodage des codes convolutifs pour obtenir de meilleurs résultats face au bruit.

Pour chaque mots de code nous constituons un diagramme d'état qui, à un état du registre, associe une probabilité du(des) bit(s) de sortie en fonction de la probabilité d'erreur p du canal et du(des) bit(s) d'entrée. Nous considérerons l'état du registre nul à chaque nouveau mot. De ce faite, lors de l'initialisation de notre vecteur nous considérons la probabilité que le registre soit nul à un et celle des $2^m - 1$ autres états possibles du registre égale à zéro. Pour chaque nouveau bit d'entrée, les probabilités de chaque états évoluent permettant ainsi de décoder notre mot reçu en fonction de la probabilité la plus grande de se situer dans un état du registre. Les paramètres du décodage sont dans un contexte coopératif connu, ainsi connaissant l'entrelaceur, le décodage revient à un calcul de probabilités. Dans notre cas, nous supposerons l'entrelaceur connu [1] et nous devons donc deviner la permutation. Nous reconstruisons la permutation position par position à partir du début. Pour cela, il faut être capable de distinguer à une étape j le bon choix pour cette j-ème position d'un mauvais choix.

Recherche d'un distingueur

plaçons nous à la j-ème étape de la reconstruction. Nous supposons donc connues les $j-1$ premières positions de la permutation et les M diagrammes d'états de notre décodeur après avoir lu les $j-1$ premiers bits d'entrée et les $j-1$ bits de sorties associés. Pour deviner la j-ème position de la permutation nous allons essayer les $n-j+1$ possibilités correspondant chacune à un ensemble de M bits d'entrée pour la j-ème étape. Les j-ème bits de sortie étant connus,

1. Il est facile d'essayer tous les entrelaceurs par recherche exhaustive car la longueur de contrainte des codes convolutifs utilisés dans les turbo-codes est très courte.

nous pouvons pour chaque choix possible déduire la distribution de probabilité du diagramme d'état après j bits d'entrée. Selon que le choix pour cette j-ème position est correct ou erroné l'entropie de ces diagrammes d'état suit une distribution ou une autre (voir FIGURE 4.3). Ces distributions cibles peuvent être calculées par échantillonnage en calculant l'entropie de l'état interne de notre décodeur après j choix corrects ou après $j-1$ choix corrects et un choix incorrect. De façon générale un mauvais choix entraîne une augmentation de l'entropie de l'état interne.

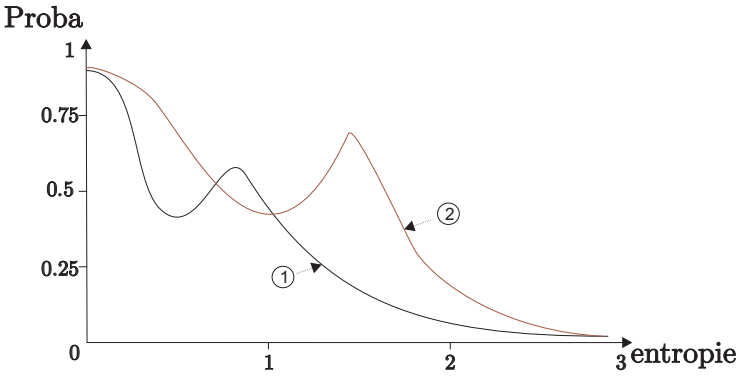


FIGURE 4.3 – Exemple de répartition d'entropie. en noir (1) l'entropie de décodage connaissant la permutation, en rouge (2) l'entropie pour une permutation aléatoire et un registre a 3 états

Le distingueur que nous construisons est un test de Neyman-Pearson pour savoir si la distribution que l'on calcule à partir des M mots interceptés est plus proche de l'une ou l'autre des deux distributions échantillonnées. Ce test nécessite un échantillonnage de nos distributions (cf FIGURE 4.4). Il convient que plus le nombre de mots M est grand plus la représentation de la répartition de l'entropie est précise. En fonction du nombre de faux positifs/faux négatifs que l'on vise et de l'écartement des deux distributions cibles (principalement lié au bruit du canal), il est possible de calculer une borne minimale sur M pour que la reconstruction puisse fonctionner.

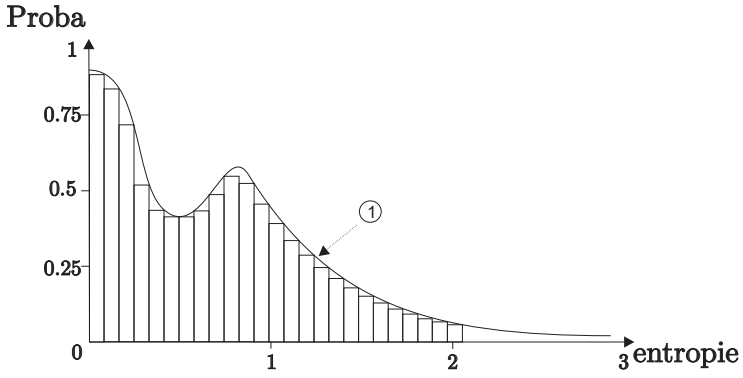


FIGURE 4.4 – Échantillonnage de la fonction de répartition de l'entropie (courbe 1)

Reconstruction de la permutation du Turbo-code

Pour reconstruire la permutation en entier, on procède en n étapes : à la j-ème étape on essaye les $n - j + 1$ valeurs possibles de la permutation et on ne conserve que les candidats qui passent le test décrit précédemment. Si M est suffisamment grand un seul candidat est conservé à chaque étape et la permutation est entièrement reconstruite à la fin. Dans ce cas la complexité de l'algorithme de reconstruction est

$$\sum_{i=0}^{n-1}(n-i)M2^m \approx O(n^2 M2^m).$$

Poinçonnage

Le principe de fonctionnement de cette méthode reste ici inchangé. Nous devons considérer la contrainte supplémentaire qu'à un bit de sortie est associé deux bits d'entrée (pour un poinçonnage $\frac{3}{4}$). Deux bits de la permutation doivent être supposés et devinés à chaque étape. Comme précédemment il existe $\binom{N}{2}$ possibilités de choix des deux éléments de la permutation pour la première étape. Deux possibilités existent.

Soit les deux positions de la permutation à identifier ont bien été trouvées, dans quel cas nous continuons avec $\binom{N-2}{2}$ possibilité de choix de la permutation au pas suivant.

Dans le cas général, en fonction des polynômes du codeur et du poinçonnage, chaque bits de sortie peut être associé à un nombre variable Nb_{entree} de bits d'entrée. La valeur maximale de ce Nb_{entree} définit la complexité de l'algorithme qui devient :

$$\sum_{i=0}^{\frac{n-1}{Nb_{entree}}} \binom{n - iNb_{entree}}{Nb_{entree}} M2^m \approx O(n^{Nb_{entree}+1} M2^m).$$

Conclusion

Basée sur le principe de décodage des turbo-codes, cette méthode est très robuste aux erreurs, ce qui en fait sa principale force. Néanmoins, cette technique nécessite la connaissance des polynômes du codeur $(P_1(D), P_2(D))$: il est

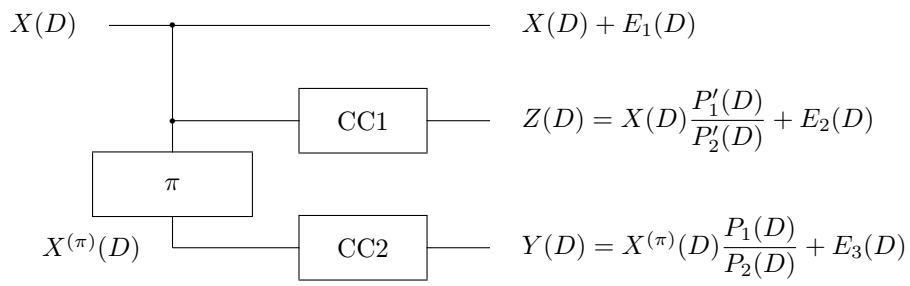
nécessaire de les essayer tous. Un mauvais choix de ce codeur fait que l'algorithme de reconstruction s'arrête au bout de quelques étapes seulement.

n	σ	M	(théorie)	temps d'exécution en secondes
64	0.43	50	(48)	0.2
64	0.6	115	(115)	0.3
64	1	1380	(1380)	12
512	0.6	170	(169)	11
512	0.8	600	(597)	37
512	1	2 800	(2 736)	173
512	1.1	3 840	(3 837)	357
512	1.3	29 500	(29 448)	4 477
10 000	0.43	300	(163)	8 173
10 000	0.6	250	(249)	7 043

TABLE 4.5 – Temps de reconstruction de la permutation complète pour un canal gaussien d'écart type σ et différentes tailles de turbo-code et d'échantillon.

4.1.3 Méthode 3 : méthode basée sur la recherche d'équations de parité de poids faible

Reprenons le turbo-code étudié :



La problématique reste inchangée, l'idée nouvelle de cette méthode n'est de plus s'intéresser au code mais à son dual et plus particulièrement aux équations de parité de poids faible. Considérons le code C' formée de la sortie $X(D)$ et de la sortie $Y(D)$ tel que

$$C' = m'_1, \ldots, m'_k$$

Où

$$m'_i = x_1, y_1, x_2, y_2, \ldots, x_N, y_N$$

Ce code C' possède des équations de parité liant les bits de $X(D)$ et $Y(D)$ qui nous permettent de retrouver la permutation.

Recherche de mots de poids faibles dans le codes

Soit n la taille de la permutation, nous devons donc rechercher toutes les

équations de parité dans le message de longueur $2n$. Sachant que n peut valoir plusieurs milliers de bits, il convient que rechercher des relation de parité de poids quelconques sur un tel intervalle avec l'algorithme de Valembois demande une puissance de calcul très importante. À part pour des niveaux de bruit très faibles, cette solution n'est pas acceptable. Néanmoins, dans ce code il existe des relations de parité de poids faible w plus facile à rechercher et suffisantes pour retrouver la permutation. Pour cela nous utilisons un algorithme de recherche déterministe de mots de poids donné dans un code, basé sur le paradoxe des anniversaires. Il permet pour un même temps d'exécution $(2n)^{\lceil \frac{w}{2} \rceil}$ d'obtenir une taille en mémoire moindre, de l'ordre $(2n)^{\lceil \frac{w}{4} \rceil}$, c'est l'algorithme de P. Chose, A. Joux et M. Mitton [CJM02]. Ainsi nous cherchons des équations de poids w (et inférieur) dans notre code. w est choisi de manière expérimentale en fonction de la machine, cette valeur ne peut pas être trop petite ne fournissant pas assez de données pour retrouver notre permutation, ni trop grande limitant le temps de calcul. Expérimentalement, il convient que $w = 6$ ou $w = 8$ est un bon paramètre pour la reconstruction des turbo-codes.

Retrouver la permutation

Les équation de parité trouvées dans le code sont les mêmes équations que celles fournies par le code, engendré par l'entrelacement des sorties $u_1(D)$ et $u_2(D)$ et visible sur la figure 4.6. On suppose $P_1(D)$ et $P_2(D)$ connu dans un premier temps.

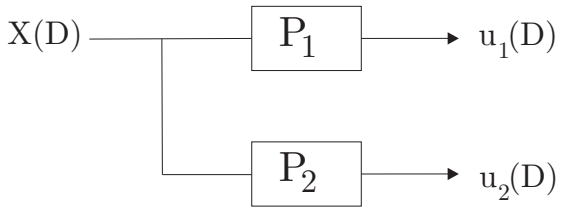


FIGURE 4.6 – Schéma engendrant un système d'équation de parité équivalent aux sortie $X(D)$ et $Y(D)$ du turbo-code

Cette représentation (cf figure 4.6) est équivalente en terme d'équation de parité du code au code précédant. Soit α et β appartenant à $\mathbb{F}_2[D]$, tous les équations de parité vérifient l'équation :

$$\alpha(D)P_1(D) + \beta(D)P_2(D) = 0$$

Pour tout équation de parité de poids w faible, $\alpha(D)$ et $\beta(D)$ vérifient

$$nb_{monomes}(\alpha(D)) + nb_{monomes}(\beta(D)) \leq w$$

Ou encore, si nous représentons ces polynômes sous forme de vecteur binaire α_{BE_d} et β_{BE_d}, de taille d égale au degré du polynômes plus un ,alors

$$|\alpha_{BE_d}|_H + |\beta_{BE_d}|_H \leq w$$

Nous nous ramenons au calcul de

$$\begin{aligned} \alpha(D) &= \lambda(D)P_2(D) \\ \beta(D) &= \lambda(D)P_1(D) \end{aligned}$$

où $\lambda(D) \in \mathbb{F}_2[D]$, dont $|\lambda_{BE_d}|_H \leq w$. Nous obtenons tous les couples $(\alpha(D), \beta(D))$ signature des polynômes $(P_1(D), P_2(D))$.

Désormais nous possédons les équations de parité de poids faible engendrées par nos polynômes, nous allons les relier aux équations de poids faibles identifiées dans le code. Les équations de parité du code font intervenir des bits de redondance (sortie $Y(D)$) qui suivent l'ordre de sortie du codeur convolutif et les bits d'informations (sortie $X(D)$) qui eux, sont rangés dans un ordre donné par la permutation. Soit $\Gamma(D)$ la partie des équations de parités du code faisant intervenir les bits de redondance et $\Delta(D)$ la partie faisant intervenir les bits d'information. Nous recherchons les couples

$$(\alpha_i(D), \beta_i(D)) = (\Gamma_{i,k}(D), \Delta_{i,k}(D))$$

où $i \in [1, nb_{eq}]$, i est l'indice de l'équation de parité et nb_{eq} le nombre d'équations de parité pour le couple $(P_1(D), P_2(D))$. k est l'indice du premier bits de la relation de parité. Soit d_{Δ_i} le degré du polynôme $\Delta_i(D)$ et d_{Γ_i} le degré du polynôme $\Gamma_i(D)$.

$$\begin{aligned} \Gamma_{i,k}(D) &= \gamma_{i,0} + \gamma_{i,1}D + \cdots + \gamma_{i,d_{\Gamma_i}} \\ \Delta_{i,k}(D) &= \delta_{i,0} + \delta_{i,1}D + \cdots + \delta_{i,d_{\Delta_i}} \end{aligned}$$

Le bit de redondance n'étant pas unique (ce qui impliquerait qu'un des polynômes vaut 1), nous avons une indétermination sur la position des bits d'information. L'équation de parité est vraie pour $k = 0$ mais aussi pour $k = d_{\delta_i}$ ainsi nous avons deux équations de parités. Si nous nous intéressons aux bits de $X(D)$ dont on ne connaît pas la position nous avons

$$\begin{aligned} \delta_{i,0} + \delta_{i,1}D + \cdots + \quad \delta_{i,k} &= 0 \\ \delta_{i,k} &+ \delta_{i,k+1}D + \cdots + \delta_{i,k+d_{\Delta_i}} = 0 \end{aligned}$$

En décalant l'équation de $k = d_{\delta_i}$, nous assurons de lever une indétermination, celle de $\delta_{i,k}$ qui est la seule variable inconnue commune aux deux équations. En réitérant l'opération nous pouvons retrouver l'emplacement dans la permutation de tous les bits définis par 0 modulo k. Les équations de parités $(\alpha(D), \beta(D))$ sont aussi vrai par multiplication d'une puissance de D.

$$D^j(\Gamma_{i,k}(D), \Delta_{i,k}(D)) = (\Gamma_{i,k+j}(D), \Delta_{i,k+j}(D)) = 0$$

Cela se répercute sur les bits par un décalage de j bits en fonctions de la multiplication par D^j. Ainsi nous pouvons retrouver les bits de la permutation définis par j modulo k avec $j < k$. Aux extrémités de la permutation apparaît plusieurs équations parasites de petit poids. Nous ne les considérons pas pour nos calcul.

Recherche des polynômes $P_1(D)$ et $P_2(D)$)

Nous avons supposé pour la reconstruction de l'entrelaceur que $P_1(D)$ et $P_2(D)$ était connue, ce qui n'est pas vrai dans le cadre de reconnaissance de code. Pour lever cette indétermination, nous allons créer l'empreinte de chaque couple $(P_1(D), P_2(D))$, nous l'appelons un dictionnaire de chaque couple contenant les $(\alpha_i(D), \beta_i(D))$. Il sera alors possible pour toute les équations de parité de poids faible du code, de discriminer les polynômes dont le dictionnaire ne contient

pas une relation de parité. Nous obtenons ainsi un seul couple $(P_1(D), P_2(D))$ vérifiant toutes les équations de parité et seulement celles-ci.

Cas bruité, poinçonné

Il peut néanmoins arriver que certaines positions de l'entrelaceur ne soient pas identifiable. Notamment les positions qui font intervenir des bits dans les équations de parité au extrémité des données étudiés. En effet, Il se peut que les k premiers bits et les k dernier ne soient reconnus, les autres équations de parité $(\alpha_i(D), \beta_i(D))$ associé aux polynômes $(P_1(D), P_2(D))$ nous permettent de lever cette indétermination. La robustesse face au bruit en est de plus améliorée.

Le bruit aura pour effet de diminuer le nombre d'équations de poids faible que nous trouvons dans le code, ne permettant pas à une unique équation de retrouver la totalité de la permutation. Les équations $(\alpha(D), \beta(D))$ fournies par les polynômes $(P_1(D), P_2(D))$ permettent, jusqu'à un certain niveau de bruit, de retrouver celle-ci.

Quand au poinçonnage, il jouera le même rôle de réducteur d'équation que le bruit en ne faisant plus intervenir certains bits de redondance.

Dans cette méthode, nous utiliserons les équations de parité comme nous l'avons fait pour les codes convolutifs (cf section 3). Les inconvénients sont que la recherche des équations de parité nous oblige à utiliser des mots de poids faible dû à sa complexité algorithmique chronophage. L'utilisation de dictionnaire rend très rapide la reconnaissance des polynômes car il suffit simplement de faire une comparaison. Un des avantages de cette méthode est qu'elle reste inchangée que le code soit poinçonné ou non.

4.1.4 Comparaison des méthodes

Il est difficile de comparer ces méthodes qui sont très différentes, mais nous pouvons déjà dresser les qualités et les défauts de chacune d'elles.

La méthode basée sur les statistiques que j'ai réalisé avec N. Sendrier est fondée sur l'hypothèse que le développement en série entière de $\frac{P_1(D)}{P_2(D)}$ possède un terme constant. Cette hypothèse n'est pas contraignante car la plupart des codeurs possèdent un monôme de degré zéro. Le point fort de cette méthode est sa rapidité, malgré un parcours de toutes les données plusieurs fois, elle réalise seulement des opérations simples (addition, multiplication, xor). Au contraire, cela se produit au détriment de la robustesse aux erreurs qui malgré sa méthode de correction trouvera assez vite ses limites face au bruit. Pour amélioré ce facteur, il serait judicieux de remplacer l'opération de correction par un véritable décodage rendant la méthode un peu plus longue. Cette méthode est intéressante, pour la compréhension du problème et reste fonctionnelle dans beaucoup de cas. Une étude plus approfondie permettra son amélioration et aussi d'en déterminer plus précisément les limites.

La méthode basée sur le décodage a été implémenté par M. Finiasz et M. Cluzeau. L'avantage certain de cette méthode repose sur le fait qu'elle est basée sur le décodage la rendant très robuste aux erreurs. Ainsi, si l'utilisateur est capable de retrouver le message malgré les erreurs, alors nous sommes capables de reconnaître le code considérant le même taux de bruit. Une hypothèse très forte et limitant considère que nous connaissant les polynômes. Cette méthode

est beaucoup plus lente que la précédente du fait de nombreux pré calcul et d'un arbre de recherche potentiellement grand. Il est possible de traiter le cas du poinçonnage à travers cet algorithme, en contrepartie d'un coups de calcul grandement augmenté.

La troisième méthode utilisant les équations de poids faible correspond à un compromis aux deux premières méthodes. Basée sur la recherche d'équations de poids faible (pour un poids fixé) dans le dual rend cette méthode plus robuste en théorie au bruit que la première méthode dans sa proposition actuelle. L'algorithme utilisé pour la recherche des mots d'un poids donné dans le dual étant rapide pour des petits poids, nous obtenons une rapidité accrue par rapport à la seconde méthode. Le temps d'exécution est indépendant du niveau de bruit : pour des taux d'erreurs très faible cette méthode est fonctionnelle mais plus lente que les deux autres, pour des taux d'erreurs élevés le temps d'exécution reste inchangé mais la méthode ne reconstruit qu'une partie de la permutation à la fois.

4.1.5 Conclusion des turbo-codes

Cette section plus théorique a permis de découvrir une méthode que j'ai implémenté (méthode statistique) et deux autres méthodes en cours d'étude théorique et d'implémentation. En amenant de nouvelles idées et en les implémentant, nous allons plus loin que ce qui a été réalisé à ce jour par les travaux de J. Barbier [Bar07]. Ces études offrent de grandes perspectives pour la reconnaissance de turbo-codes. L'implémentation et le test de ces méthodes permettra de déterminer en détail les caractéristiques réelles de ces algorithmes. Malgré ce progrès sur les turbo-codes, il reste de nombreux points non résolus car même si la recherche a grandement avancée dans le domaine des turbo-codes parallèles, il n'en n'est rien des turbo-codes série qui sont une cible intéressante pour la reconnaissance de code.

Conclusions et perspectives

La reconnaissance des codes convolutifs a été étudiée dans le passé au travers des thèses de E. Filiol et J. Barbier. Nous sommes reparti de ces travaux et plus fondamentalement des travaux de R.J. McEliece sur les codes convolutifs pour développer un nouvel algorithme de reconnaissance des codes convolutifs à la fois simple et de faible complexité. Il est le seul algorithme connu de reconnaissance des codes convolutifs totalement automatique.

Cet algorithme utilise entre autres l'algorithme de A. Valembois de recherche de mots du code dual à partir de séquences codées bruitées. La complexité de cet algorithme n'a jamais été étudiée en détail. Nous pensons que son étude et sa compréhension ainsi que son amélioration permettrait de reconnaître les codes jusqu'à des taux d'erreurs d'une dizaine de pour cent.

Dans ce manuscrit, nous avons également abordé le sujet des codes convolutifs poinçonnés. L'intérêt de retrouver le poinçonnage est de diminuer la complexité de décodage. En effet, la complexité de décodage d'un code convolutif en utilisant l'algorithme de Viterbi est $\mathcal{O}(2^{k+m})$, m étant la longueur de contrainte. Nous voyons alors la différence importante de complexité entre le décodage s'un code $(4, 1)$ (complexité 2^{m+1}) et son poinçonné $(13, 8)$ (complexité $\mathcal{O}(2^{8+m})$). Il permet entre autre de diminuer légèrement l'indétermination du codeur qui dans un contexte de reconnaissance est important. Des recherches communes avec M. Cluzeau et M. Finiasz sur le poinçonnage ont permis d'élaborer un algorithme de reconnaissance dans le cas d'un code convolutif poinçonné. Cette méthode se fonde sur l'élaboration d'un algorithme de complexité polynomiale permettant de répondre au problème suivant : Existe-t-il un codeur convolutif C1 pour un (n, k, m) donnée et un motif de poinçonnage tels que la séquence codée par le codeur C1 et poinçonnée coïncide avec une séquence codée par un codeur convolutif C2 non poinçonné. L'algorithme proposé s'effectue en deux temps. Dans un premier temps, nous reconnaissons un code convolutif poinçonné comme un code convolutif de paramètres plus grand (n', k', m). Dans un second temps, nous testons tous les motifs de poinçonnage pour des paramètres qui seraient compatibles avec le code convolutif reconnu. Néanmoins cette recherche exhaustive des motifs est coûteuse en temps et demande énormément de puissance de calcul notamment pour les motifs longs. Il serait souhaitable de trouver une méthode qui permettrait de trouver des critères capables de diminuer l'espace des motifs de poinçonnages rendant ainsi la méthode moins coûteuse dans le cas de très grands motifs de poinçonnage.

De manière pratique, nous avons constaté que pour un bruit assez élevé notre algorithme pouvait ne pas reconnaître le code recherché mais un code identique de paramètres plus grand ((ni, ki), avec $i \in \mathbb{N}$). Certaines heuristiques simples seraient envisageable pour résoudre ce problème. Etant capable de retrouver un

codeur, il suffirait alors de décoder (corriger les erreurs) pour diminuer grandement le taux de bruit de la séquence interceptée. Ainsi si nous effectuons l'algorithme avec la nouvelle séquence corrigée, nous devrions trouver le code (n, k) d'origine (si celui-ci était le bon codeur).

Les turbo-codes convolutifs étant basés sur deux codeurs convolutifs mis en parallèle, notre algorithme de reconnaissance de codes convolutifs précédent a pu être mis à profit pour reconnaître un turbo-code. En effet, le premier codeur convolutif est facilement identifiable par simple application de notre algorithme. Le problème est donc revenu à retrouver le deuxième codeur dont les bits d'entrée ont subi une permutation. La méthode de reconnaissance que nous avons proposée consiste à reconnaître, pour un turbo-code non poinçonné, la permutation pas à pas. Elle est compétitive face aux autres méthodes auxquelles nous l'avons comparée, notamment par sa rapidité. Nous avons considéré deux cas : dans le premier, l'identification du premier codeur convolutif a permis de corriger toutes les erreurs de la séquence d'information. Dans ce cas, nous avons reconstruit l'entrelaceur et une base du code convolutif pour des taux d'erreurs de trente pour cent sur la partie redondance. Dans le second cas, il se peut que le premier codeur n'ait pas totalement corrigé les erreurs sur la séquence d'information. Dans cette situation, nous ne savons pas ou se situe l'erreur soit sur la séquence d'information, soit sur la séquence de redondance. La quantité de blocs nécessaires à la reconnaissance du turbo-code grandit de façon exponentielle plus le taux d'erreur est élevé.

Nous avons considéré la séquence observée comme une séquence codée de bits $\ll 0 \gg$ et$\ll 1 \gg$. En pratique, la séquence observée est une séquence codée dans laquelle chaque symbole a une probabilité de valoir $\ll 0 \gg$ ou $\ll 1 \gg$. Les décodeurs utilisent toujours cette information souple, ce qui leur permet d'utiliser au mieux la capacité de correction du code. En utilisant de manière naturelle ces vraisemblances pour le problème de reconnaissance, il est probable que l'on puisse reconnaître le code pour des niveaux de bruit plus élevés. Cette information souple peut être utilisée de plusieurs façons. La première, naïve, serait d'observer la séquence codée jusqu'à identifier une plage de données peu bruitée, transformer l'information souple en une information dure et d'y appliquer notre algorithme de détection. Cette simple manipulation offrirait de meilleures performances face au bruit dans le cas de la reconnaissance d'un code convolutif par exemple. Une seconde méthode serait de reconsidérer le problème directement à partir de l'information souple et d'y adapter ou développer des algorithmes nouveaux pour la reconnaissance de codes.

Notations

Pour les codes nous utilisons les notations suivantes
- C correspond à un code.
- $\mathcal{A}$: alphabet de q éléments finis, souvent $\mathcal{A} = \{0, 1\}$ pour les codes binaires.
- u est un vecteur ($\in \mathcal{A}$) d'éléments u_i correspondant à l'information que l'on souhaite transmettre.
- $\hat{u}$: correspond au vecteur des $\hat{u}_i$ éléments du message d'information estimé à la réception.
- v est un vecteur ($\in \mathcal{A}$) d'éléments u_i correspondant à notre séquence d'information après brassage.
- $\hat{v}$: correspond au vecteur des $\hat{v}_i$ éléments du message reçu estimé après décodage et avant le débrassage.
- x est un vecteur ($\in \mathcal{A}$) d'éléments u_i correspondant à notre séquence codée que l'on va envoyer.
- $\hat{x}$: correspond au vecteur des $\hat{x}_i$ éléments du message codé estimé reçu après démodulation.
- n correspond à la longueur d'un code C.
- k est la dimension de notre code C.
- s correspond à la synchronisation d'un mot de code. Il faut différencier la synchronisation d'un code en bloc d'un code convolutif.
- G est une matrice génératrice du code C.
- $\mathcal{H}$ est une matrice de parité du code C. Nous la notons aussi $C^{\perp}$
- G_{sys} est la matrice systématique du code.
- t est un opérateur de temps.
- N le nombre de mots de code reçus.
- $\tilde{m}$ est un mot bruité reçu.
- $\tilde{M}$ est la matrice de mots bruités $\tilde{m}_i$ reçu.

Pour les codes convolutifs
- n représente le nombre de sorties (la dimension de l'espace d'arrivée) du code.
- k est la dimension de notre code, pour les codes convolutifs cette dimension correspond au nombre d'entrée du codeur.
- m est la longueur de contrainte. Plusieurs définitions existent, nous utilisons celle donné par McEliece (cf. [McE98])
- $G^{[]}$ est la matrice regroupé du code convolutif.
- D est l'opérateur $\ll$ Delay $\gg$.

Pour les turbo-codes, nous rajoutons des notations à celles des codes convolutifs.
- π l'entrelaceur du turbo-code.
- M est le longueur de la séquence reçue et utilisée.
- n est la longueur de l'entrelaceur π.
- s correspond à la synchronisation d'un mot de code ou bloc de code de longueur n (différent des codes convolutifs).

Annexe A

Berlekamp-Massey

L'algorithme de E. Berlakamp est très utilisé en informatique, il a été présenté en 1968 dans son livre [Ber68] en tant qu'algorithme de décodage des codes BCH. J.M. Massey a démontré un an plus tard que l'intérêt de cet algorithme ne résidait pas dans ce seul fait mais aussi dans le calcul de la complexité linéaire d'une suite. Il devient possible de retrouver le plus petit LFSR pouvant engendrer une suite de bits donnés. Cette méthode provient principalement de la cryptanalyse où les LFSR sont omniprésents. En reconnaissance de code, ils peuvent être utilisés pour retrouver des brasseurs linéaires ou encore des codes convolutifs (n,1).

Soit $y(D)$ la série entière tel que $Y(D) = \frac{P_1(D)}{P_2(D)}U(D)$ avec $P_1(D)$ le polynôme "forward", $P_2(D)$ le polynôme "backward" et $u(D)$ la séquence d'entrée du LFSR. La méthode de Berlekamp-Massey permet de nous renvoyer les polynômes $P_1(D)$ et $P_2(D)$ en fonction de la séquence interceptée $Y(D)$. J.L. Dornstetter [Dor87] à démontré que cette méthode était équivalent à un algorithme d'Euclide. Il possède une complexité quadratique en $0(n^2)$. Il peut être amélioré en utilisant des versions sous-quadratiques dérivées de la version sous-quadratique d'Euclide où la multiplication de polynômes est réaliser par une transformée de Fourrier discret de complexité $0(n \log^2 n)$
Le degré maximum de $(P_1(D), P_2(D))$ est l, la complexité en $0(l)$ est définit par la taille ou la mémoire minimum du LFSR contenant ses polynômes. Pour toute suite en-itère $y(D)$ de complexité linéaire $0(l)$, il est nécessaire de disposer de $2l$ bits consécutifs pour être sur de retrouver le polynôme minimal (développement en série entière de $\frac{P_1(D)}{P_2(D)}$) engendrant cette séquence.

Afin de retrouver les polynôme, l'algorithme parcours notre séquence interceptée et construit le LFSR minimal jusqu'au temps t. L'état initial du registre est donné par les premiers bits de la série, seul le polynôme, série entière tronquée au terme D^{2l}, $S(D) = Y(D)^{(2l)}$ est recherchée. Le passage au temps $t+1$ laisse apparaître deux possibilités, soit le LFSR engendre encore notre suite dans quel cas on ne fait rien soit il ne l'engendre plus, on parle "d'échec". L'algorithme va alors corriger le polynôme $S(D)$ afin qu'il engendre la suite en lui ajoutant avec un décalage indiqué par la variable δ le polynôme $S_t(D)$ précédemment calculé au temps t. De façon plus formel, l'algorithme est le suivant.

Algorithme 8 Algorithme de Berlekamp-Massey

Entrée : La suite Y(D).
un entier d
Sortie : La suite S(D) jusqu'au degré d.

1: [Initialisation]
On commence avec $t \leftarrow 0$, $S_t(D) \leftarrow 0$, $S_{t-1} \leftarrow 0$, le degré $l \leftarrow 0$. $\delta \leftarrow -1$ on aura toujours $\delta + t + 1 = 2l$
2: [fin ?]
Si $t = n$ alors on a fini. Le polynôme qui génère la suite est $S_t(D)$ et le degré de la suite est en l.
3: [Erreur ?]
Si

$$\sum_{i=0}^{l} c_i y_{t+i} = 0$$

avec $S(D) = \sum_{i \geq 0} c_i D^i$. Alors le polynôme $S_t(D)$ génère toujours la suite, aller en 5.
4: [Correction]
- Si $\delta > 0$, alors l ne change pas et $S_t(D) \leftarrow S_{t-1}(D) + D^{\delta} S_{t-1}(D)$
- Sinon $\delta \leftarrow -\delta$. $S_t(D) = S_{t-1}(D) + D^{\delta} S_t(D)$ et S_{t-1} prend l'ancienne valeur de $S_t(D)$. $l \leftarrow l + \delta$

5: [Boucle] faire $t \leftarrow t + 1$, $\delta \leftarrow \delta - 1$ et retourner en 2.

Index

Bibliographie

[Bar07] J. Barbier. *Analyse de canaux de communication dans un contexte non coopératif.* Thèse de doctorat, École Polytechnique, November 2007.

[Ber68] E. R. Berlekamp. *Algebraic Coding Theory.* Aegen Park Press, 1968.

[BGT93] C. Berrou, A. Glavieux, and P. Thitimajshima. Near shannon limit error-correcting coding and decoding : turbo-codes. In *IEEE ICC '93, Geneva*, pages 1064 – 1070, 1993.

[Bla03] R. E. Blahut. *Algebraic Codes for data transmission.* Cambridge university press, 2003.

[BLP08] D. Bernstein, T. Lange, and C. Peters. Attacking and defending the McEliece cryptosystem. In J. Buchmann and J. Ding, editors, *Post-Quantum Cryptography*, number 5299 in LNCS, pages 31–46. Springer-Verlag, 2008.

[Bos99] M. Bossert. *Channel Coding for telecommunications.* John Wiley & sons, 1999.

[Can96] A. Canteaut. *Attaques de cryptosystèmes à mots de poids faible et construction de fonction t-résilientes.* Thèse de doctorat, Université Paris 6, October 1996.

[CF09a] Mathieu Cluzeau and Matthieu Finiasz. Reconstruction of punctured convolutional codes. In *Proceedings of the 2009 IEEE Information Theory Workshop.* IEEE, 2009.

[CF09b] Mathieu Cluzeau and Matthieu Finiasz. Recovering a code's length and synchronization from a noisy intercepted bitstream. In *Proceedings of the 2009 IEEE International Symposium on Information Theory.* IEEE, 2009.

[CFT10] M. Cluzeau, M. Finiasz, and J. P. Tillich. Methods for the reconstruction of parallel turbo codes. In *Proceedings of the 2010 IEEE International Symposium on Information Theory.* IEEE, 2010.

[Cha92] F. Chabaud. Asymtotic analysis of probabilistic algorithms for finding short codewords. In S. Harari P. Camion, P. Charpin, editor, *EUROCODE 92*, pages 175–183, 1992.

[CJM02] P. Chose, A. Joux, and M. Mitton. Fast correlation attacks : an algorithmic point of view. In L.R. Knudsen, editor, *Eurocrypt 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 209–221. Springer, 2002.

[Clu06] M. Cluzeau. *Reconnaissance d'un schéma de codage.* Thèse de doctorat, École Polytechnique, November 2006.

[CS] M. Côte and N. Sendrier. Reconstruction of a turbo-code interleaver from noisy observation. submitted to ISIT 2010.

[CS09] M. Côte and N. Sendrier. Reconstruction of convolutional codes from noisy observation. In *International Symposium on Information Theory(ISIT'09)*, Séoul, June 2009.

[Dor87] J. L. Dornstetter. On the equivalence between berlekamp's and euclid's algorithms. *IEEE Transactions on Information Theory*, 33(3), May 1987.

[Fil97a] É. Filiol. Reconstruction de codeurs convolutifs. Mémoire de DEA, Université de Paris 6, 1997.

[Fil97b] É. Filiol. Reconstruction of convolutionnal encoders over GF(q). In Michael Darnell, editor, *Cryptography and Coding ; proceedings of the 6th IMA conference*, number 1355 in LNCS, pages 101–109. Springer-Verlag, 1997.

[Fil00] É. Filiol. Reconstruction of punctured convolutional encoders. In *International Symposium on Information Theory and its Applications (ISITA'00)*, Hawaii, November 2000.

[Gib91] J. K. Gibson. Equivalent Goppa codes and trapdoors to McEliece's public key cryptosystem. In D. W. Davies, editor, *Advances in Cryptology - EUROCRYPT'91*, number 547 in LNCS, pages 517–521. Springer-Verlag, 1991.

[JZ99] R. Johannesson and K. Zigangirov. *Fundamentals of Convolutional Coding.* John Wiley & Sons, 1999.

[KI01] K. Kobara and H. Imai. Semantically secure McEliece public-key cryptosystems -Conversions for McEliece PKC-. In K. Kim, editor, *PKC'2001*, number 1992 in LNCS, pages 19–35. Springer-Verlag, 2001.

[KT91] V.I. Korzhik and A.I. Turkin. Cryptanalysis of McEliece's public-key cryptosystem. In D. W. Davies, editor, *Advances in Cryptology - EUROCRYPT'91*, number 547 in LNCS, pages 68–70. Springer-Verlag, 1991.

[LB88] P. J. Lee and E. F. Brickell. An observation on the security of McEliece's public-key cryptosystem. In C. G. Günther, editor, *Advances in Cryptology - EUROCRYPT'88*, number 330 in LNCS, pages 275–280. Springer-Verlag, 1988.

[LC04] Shu Lin and Daniel J. Costello. *Error Control Coding Fundamentals and application.* Pearson education Inc, 2004.

[Leo88] J. S. Leon. A probabilistic algorithm for computing minimum weights of large error-correcting codes. *IEEE Transactions on Information Theory*, 34(5) :1354–1359, September 1988.

[Loi97] P. Loidreau. éléments sur les codes de Goppa en relation avec le protocole de McEliece. Rapport de DEA, École Polytechnique, 1997.

[McE78] R. J. McEliece. A public-key cryptosystem based on algebraic coding theory. *DSN Prog. Rep.,* Jet Prop. Lab., California Inst. Technol., Pasadena, CA, pages 114–116, January 1978.

[McE98] R. J. McEliece. The algebraic theory of convolutional codes. In V. S. Pless and W. C. Huffman, editors, *Handbook of Coding theory*, volume I, chapter 12, pages 1065–1138. North-Holland, 1998.

[ML85] A. M. Michelson and A. H. Levesque. « *Error-control techniques for digital communication*. John Wiley & sons, 1985.

[MS77] F. J. MacWilliams and N. J. A. Sloane. *The Theory of Error-Correcting Codes*, chapter 12. Alternant, Goppa and other generalized BCH codes. North-Holland, 1977.

[Pir88] Ph. Piret. *Convolutional codes : an algebraic approach*. MIT Press, Cambridge, MA, 1988.

[Pla96] Guillaume Planquette. *Identification de trains binaires codés*. Thèse de doctorat, Université de Rennes I, December 1996.

[Ric95] B. Rice. Determining the parameters of a $\frac{1}{n}$ convolutional encoder over gf(q). In *Third International Conference on Finite Fields and Applications*, Glasgow, UK, 1995.

[SH05] G. Sicot and S. Houcke. Blind detection of interleaver parameters. In *ICASSP*, philadelphia, USA, 2005.

[Ste89] J. Stern. A method for finding codewords of small weight. In G. Cohen and J. Wolfmann, editors, *Coding theory and applications*, number 388 in LNCS, pages 106–113. Springer-Verlag, 1989.

[Val00] A. Valembois. *Détection Décodage et Reconnaissance des Codes Linéaires Binaires*. Thèse de doctorat, Université de Limoges, October 2000.

[Val01] A. Valembois. Detection and recognition of a binary linear code. *Discrete Applied Mathematics*, 111(1-2) :199–218, July 2001.

[Wic03] Stephen B. Wicker. *Fundamentals of codes, Graphs, and Iterative decoding*. kluwer Academic Publishers, 2003.