Reconstruction d'un schéma de codage

Mathieu Cluzeau

Thèse de Doctorat

présentée à
L'ÉCOLE POLYTECHNIQUE

pour obtenir le titre de
DOCTEUR EN SCIENCES

Spécialité **Informatique**

soutenue le 28 novembre 2006 par

Mathieu CLUZEAU

Reconnaissance d'un schéma de codage

Jury

Rapporteurs

Thierry BERGER	Université de Limoges
Vladimir SIDORENKO	University of Ulm (Germany)

Directeurs de Thèse

Anne CANTEAUT	INRIA-Rocquencourt projet CODES
Nicolas SENDRIER	INRIA-Rocquencourt projet CODES

Examinateurs

Matthew Geoffrey PARKER	University of Bergen (Norway)
Jean-Marc STEYAERT	École polytechnique
Georges TANTOT	DGA
Jean-Pierre TILLICH	INRIA-Rocquencourt projet CODES
Gilles ZÉMOR	Université de Bordeaux 1

Remerciements

Cette thèse a été effectuée au projet CODES de l'INRIA Rocquencourt entre 2003 et 2006 et a été financée par la Délégation Générale de l'Armement.

Je vais tenter de remercier ici toutes les personnes qui ont contribué de manière plus ou moins directe à la réussite de cette thèse.

Tout d'abord, je souhaite remercier toutes les personnes qui m'ont conduit à faire une thèse à l'INRIA : je souhaite donc remercier Mme Davalant, mon professeur de mathématiques en classe préparatoire qui m'a donné le goût des mathématiques et l'envie de poursuivre mes études dans cette voie. Ensuite, je tiens surtout à remercier François Arnaud, Thierry Berger, Anne Canteaut et Philippe Gaborit qui m'ont donné envie, lors de mon DEA à l'université de Limoges, de faire une thèse dans le domaine de la cryptographie et des codes correcteurs d'erreurs. Cela a été possible en particulier grâce à Philippe Gaborit qui m'a permis d'effectuer mon stage de DEA au sein du projet CODES, je lui en suis très reconnaissant.

Je souhaite remercier Nicolas Sendrier pour m'avoir proposé un sujet ouvert et passionnant ainsi que pour m'avoir aidé et encouragé tout au long de ces trois années de thèse. Malgré son cruel manque de temps, il s'est toujours rendu disponible lorsque j'avais besoin de son aide.

J'ai eu la chance en plus d'un directeur de thèse d'avoir une directrice de thèse en la personne d'Anne Canteaut. Ce manuscrit ne serait certainement pas ce qu'il est sans son aide incessante. Je tiens à m'excuser de lui avoir un peu (enfin j'espère) gâché ses vacances d'été pendant lesquelles elle a relu le premier jet de ce manuscrit. Je me rassure en pensant que la défaite de l'équipe de France en finale du mondial a dû rendre ces vacances en Italie beaucoup plus difficiles... En tout cas, mon document est revenu avec beaucoup de couleurs mais pas dûes au bronzage... Je la remercie donc tout particulièrement pour les heures passées à travailler avec moi à l'amélioration de ce document. Qu'elle accepte toute l'expression de ma gratitude même si ces quelques lignes ne suffisent pas pour lui témoigner toute mon admiration et ma reconnaissance.

Je tiens à remercier Thierry Berger et Vladimir Sidorenko pour m'avoir fait l'honneur d'être rapporteurs de ma thèse. La lecture attentive dont a bénéficié ce document m'a beaucoup touché. Je tiens aussi à remercier Vladimir Sidorenko pour son accueil à Ulm et l'intérêt qu'il a manifesté pour mes travaux.

Je souhaite aussi remercier tous les membres de mon jury : je réitère mes remerciements à mon directeur et à ma directrice de thèse ainsi qu'à mes rapporteurs et je remercie tous

les autre membres de mon jury : Matthew Geoffrey Parker, Jean-Marc Steyaert, Georges Tantot, Jean-Pierre Tillich et Gilles Zémor.

Je remercie Matthew Parker d'être venu de Bergen pour assister à ma soutenance et je le remercie de l'intérêt qu'il a manifesté pour mon travail.

Je tiens aussi à remercier Jean-Pierre Tillich pour toute l'aide qu'il m'a apporté au cours de ces trois années de thèse et pour toutes les réponses qu'il m'a apporté aux questions (parfois naïves) concernant le décodage itératif.

Je remercie Georges Tantot d'avoir accepté de faire partie de mon jury de thèse. À travers lui, je souhaite aussi remercier la DGA pour les trois années d'allocation de recherche dont j'ai bénéficié pour mener cette thèse.

Je remercie Gilles Zémor de m'avoir fait l'honneur de participer à mon jury.

C'est un immense honneur pour moi que Jean-Marc Steyaert ait accepté d'être dans ce jury et qu'il ait ainsi montré son intérêt pour mon travail. Je lui en suis très reconnaissant.

Je tiens à remercier tous les membres du projet CODES. Merci à tous les permanents : Daniel Augot, Anne Canteaut, Pascale Charpin, Jean-Pierre Tillich et Nicolas Sendrier, pour leur disponibilité et leur aide permanente sur tout type de sujet. Un merci particulier à Daniel Augot et à sa famille pour le traditionnel barbecue d'été souvent bien arrosé (dans tous les sens du terme...). Un grand merci à tous les membres extérieurs, stagiaires, thésards : Grisha, Claude, Françoise, Pierre, Ludovic, Bassem, Racem, Caroline, Ayoub, Fabien, Marine, Carmen, Matthieu, Harold, Raghav, Cédric, Scarab, Yann, Fred, Thomas, Manu, Andrea, Stéphane, Christophe, Michael, Maria, Domitille, Marion, Raphael, Iryna, Bhaskar. Merci à tous pour les bons moments partagés à la salle à café, même si je me sens souvent seul le lundi matin lors des défaites de l'OM, ah la la ces gens qui ne savent pas apprécier le football ;-)

Je souhaite remercier tout particulièrement Scarab, Fred et Yann pour toutes les soirées passées ensemble et pour leur aide de tous les instants au projet (et oui des fois on travaille...) mais pas seulement. Je me dois aussi de remercier Carlos (désolé mais Geneviève a insisté) pour ses scripts qui m'ont permis de mettre de belles figures colorées qui égayent mon manuscrit.

J'adresse des remerciements tout particuliers à Christelle, notre assistante, grâce à qui je n'ai pas eu trop à me soucier de tout ce qui est administratif. Je tiens aussi à la remercier pour sa bonne humeur et pour les longues discussions que nous avons eu ensemble.

Durant ces trois années j'ai eu la chance de toujours pouvoir compter sur ma famille et mes amis. Je remercie donc chaleureusement mes parents sans qui je ne serais pas arrivé jusqu'ici. Un grand merci à Igor, Ruby et Pétrole pour les bonnes journées passées ensemble au milieu des bois. Bien sûr je n'oublie pas toutes les personnes sur qui j'ai toujours pu compter, un grand merci à mon frère, à la famille Bigas, à Benoit et Caro, et aux familles Aubreton et Auroux.

Les dernières lignes de ces remerciements seront naturellement pour ma femme. Je la remercie de m'avoir forcé à me lever (presque) tous les matins et pour la motivation qu'elle a réussi à me donner même lorsque j'aurais préféré rester dormir... Je pense que sans elle, ces trois années auraient été beaucoup plus difficiles et je la remercie donc pour son soutien permanent.

Overview

Data confidentiality is achieved by using some cryptographic techniques. Since the famous paper by Kerckoffs *La Cryptographie militaire* in 1883, we know that, the security of a cipher must rely on the secret of a short parameter, named the key, and not on the secret of the method used to encipher data.

To evaluate the quality of a cryptographic algorithm, it is then always assumed that its specifications are public: in accordance with the previous principle, it would actually be dangerous that the security relies (even partly) on the fact that the attacker does not know the specifications. However, this fundamental rule does not necessarily imply that the specifications of a system are always known in practice. When this is not the case, any attack needs a preliminary reverse-engineering step to reconstruct the system. This operation often involves some information leak on the specifications of the system or a precise analysis of a hardware implementation. Famous examples are the reconstruction of Enigma during Word War II, the publication on Internet of the specifications of the RC4 algorithm used in SSL or of those of A5/1 used in the GSM standard.

But this reverse-engineering step is often hard even when the specifications are not intentionally dissimulated, but just unknown to the attacker, and when the system does not involve any cryptographic functions. This situation occurs for instance when an attacker wants to eavesdrop unencrypted communications. Actually, once a communication has been eavesdropped, the transmitted information can be recovered only « disassembling » the transmission system which has transformed the original message. A transmission system usually includes many elements which have different functionalities but there often exists a large variety of possibilities for each element.

The work which has been done during this thesis deals with this problem: our goal is to find the specifications of a transmission system from the knowledge of an eavesdropped communication which will often be corrupted by some transmission errors. Before this thesis, there were only a few publications on this subject, except the works by Bart Rice and by Eric Filiol on convolutionnal codes [Ric95, Fil97, Fil00b], by Johann Barbier for turbo-codes [Bar05], and the thesis of Antoine Valembois [Val00] on the reconstruction of block codes.

Here, we focus on the reconstruction of two major elements in a classical transmission scheme, the linear block code and the scrambler. The two main parts of this document will be successively devoted to these two elements in the system, in the order in which they have to be treated by the eavesdropper: first the error correcting code and then the scrambler.

Before these two parts, an introductory chapter briefly describes the different elements in a transmission scheme and it especially recalls some classical properties of scramblers and of linear codes.

Our work on the reconstruction of linear block codes is first based on some results due to Valembois: he has shown that this problem belongs to the class of NP-hard problems, and he has also presented an algorithm for solving it. This algorithm is presented in Chapter 2. It consists in finding words of low Hamming weight in a code constructed from the eavesdropped noisy codewords. We precisely analyze the statistical test proposed by Valembois. This careful analysis is very important because determines the quantity of information we need to eavesdrop, the time complexity of the reconstruction and especially its success rate. This analysis and our simulations show that the statistical test proposed by Valembois is not very relevant. Therefore, we propose a new statistical test which seems to be more suitable for our problem.

Then, Chapter 3 presents an algorithm inspired by classical iterative decoding algorithms which aims at exploiting the previously found low-weighted parity check equations in order to correct some errors. This chapter actually describes two algorithms: a first attempt which appears to us natural at first glance. This first algorithm has been presented at the 2006 IEEE International Symposium on Information Theory [Clu06a]. But some additional simulations have pointed out that this first algorithm was not efficient anymore in the presence of false equations, whereas it was especially conceived to decode in this context. Thus, we have modified a formula to update the probability that an equation is true which was responsible for the misbehavior. We then present an iterative algorithm which can correct some errors even if some equations are not parity checks for the code. Finally, some simulations for different types of codes are reported and analyzed.

The second part of the thesis investigates the problem of the reconstruction of a linear scrambler. When the linear block code has been recovered, this second element in the communication system must be found. We first show that the fact that the used encoder cannot be found in practice, even if the used error correcting code has been recovered, has an impact on the reconstruction of the scrambler: in general, the output of the scrambler is not known but only its image by a linear transformation per block. Moreover, in all cases, some assumptions on the input stream must be made. We have considered two different hypotheses:

- some bits of the input stream are known ;
- the input stream is unknown but produced by a biased binary source.

In Chapter 5, it is assumed that the exact output of the scrambler is known. We present different techniques for reconstructing the two kinds of scramblers: synchronous scramblers and self-synchronized scramblers. Most results in this chapter have been presented at Chicago in July 2004 at the IEEE International Symposium on Information Theory [Clu04]. In the case where some bits of the input stream are assumed to be known, the different techniques of reconstruction are essentially based on the Berlekamp-Massey algorithm and on solving a linear system. When the input is unknown but biased, the algorithm for finding the feedback polynomial is inspired by an algorithm proposed by Anne Canteaut and

Eric Filiol for the reconstruction of some pseudo-random generators. But this technique relies on a wrong assumption; we have then modified it to avoid this wrong assumption. The second step of the reconstruction, *i.e.* the reconstruction of the initial state of the register is performed by a classical cryptographic technique of correlation attack but with a description closer to iterative decoding.

Finally, Chapter 6 focuses on the reconstruction of the scrambler in the general context, that means its output is known up to a linear transformation only. More precisely, we investigate the reconstruction of a synchronous scrambler when some input bits are known but when only the image of the output by a linear transformation per block is known. We present a purely algebraic method of reconstruction which can solve this problem efficiently.

Summary of the main contributions

The manuscript is naturally split into two parts devoted to both elements of the transmission scheme. Here, we briefly summarize our main contributions.

Chapter 1 This chapter is an introduction to the reconstruction of a classical communication scheme. We present the different elements involved in a transmission scheme and we show that it could often be modelized as shown on Figure 1.5. Then we focus on the two elements that we want to recover: the scrambler and the error correcting code. We also detail the behavior of both types of linear scramblers: synchronous scramblers (Section 1.2.3) and self-synchronized scramblers (Section 1.2.4). We recall some results on Linear Feedback Shift Registers ; most properties we present here can be found in Chapter 8 of [LN83] or in [Til05]. We also present the well-known Berlekamp-Massey algorithm. Then we give some basic definitions and properties of linear error-correcting codes (see e.g. [MS77] or [LC83]).

Part I: reconstruction of a block code in the presence of errors

The first part of this thesis is devoted to the reconstruction of a linear block code when some noisy codewords are known. Some results in Chapters 2 and 3 are presented in [Clu06a], but these chapters also contain new results found later, after we have performed some additional simulations.

Chapter 2 This chapter deals with the problem of finding parity check relations for an unknown code from the knowledge of noisy codewords. We are also interested in finding an algorithm for distinguishing parity checks from random relations. Table 2.1 presents the main notation which will be used in this part. We first recall one of the main results due to A. Valembois [Val00]: the problem of recovering a linear block code belongs to the class of NP-complete problems. We give the basic principle of a reconstruction algorithm, which is presented in Corollary 2.3. Let $\widetilde{m}$ be a noisy codeword corresponding to the result of the transmission of an element of $\mathcal{C}$ through a binary symmetric channel with cross-over

probability τ. If $h \in \mathcal{C}^\perp$, then we have

$$\Pr[h\widetilde{m}^T = 0] = \frac{1 + (1-2\tau)^{wt_H(h)}}{2}.$$

And, if $h \notin \mathcal{C}^\perp$, this probability is equal to $\frac{1}{2}$.
Therefore, among all n-bit words, the elements of $\mathcal{C}^\perp$ can be distinguished from the others by considering the $n \times M$ matrix R^T composed of M noisy words of $\mathcal{C}$. Actually, the previous result implies that the elements of $\mathcal{C}^\perp$ are expected to correspond to small values of $wt_H(hR^T)$.

Based on this principle, Valembois proposed the statistical test on $wt_H(hR^T)$ which is presented in Proposition 2.4. But, we point out on several examples that this test is not very consistent. Then, we present the following new statistical analysis: the relevant quantity is the value of

$$\alpha = \frac{\sharp\left\{h \notin \mathcal{C}^\perp \text{ such that } wt_H(hR^T) \leq T\right\}}{\sharp\left\{h \in \mathcal{C}^\perp \text{ such that } wt_H(hR^T) \leq T\right\}}.$$

Then, Theorem 2.7 shows that the statistical test which decides that $h \in \mathcal{C}^\perp$ where

$$M = \left(\frac{b\sqrt{1-x^2}+a}{x}\right)^2,$$

$$T = \frac{1}{2}\left(M - a\sqrt{M}\right)$$

with $a = -\phi^{-1}\left(\alpha\frac{2^{n-k}}{2^n-2^{n-k}}(1-\beta)\right)$, $b = \phi^{-1}(1-\beta)$, satisfies:

$$\frac{\sharp\left\{h \notin \mathcal{C}^\perp \text{ such that } wt_H(hR^T) \leq T\right\}}{\sharp\left\{h \in \mathcal{C}^\perp \text{ such that } wt_H(hR^T) \leq T\right\}} = \alpha,$$

$$\text{and } \Pr\left[wt_H(hR^T) \geq T | h \in \mathcal{C}^\perp\right] = \beta.$$

We then analyze the values of M and T as a function of the different parameters (Figures 2.10 to 2.17). Most notably, we show that the threshold T for $wt_H(hR^T)$ is very close to the value of the Gilbert-Varshamov bound (see Figures 2.18 to 2.23). We finally present in section 2.5 the method due to Canteaut and Chabaud [CC98] that we used for finding those h for which the weight of hR^T is less than the previously determined threshold T.

Chapter 3 In the previous chapter, we have described an improvement of the algorithms suggested by Valembois. But, all these techniques consist in some statistical tests which choose whether a word belongs to $\mathcal{C}^\perp$ or not. In Chapter 3, we present a new iterative decoding algorithm which can correct some errors using the previously found parity check equations, even if some of them are wrong, *i.e.* if some of them are not really parity check equations for the code. Section 3.2 first makes some recalls on LDPC codes and especially

on Gallager algorithm. Table 3.1 provides the notation used for presenting iterative decoding algorithms. Section 3.3 explains how a first natural idea has led us to the algorithm presented at ISIT 2006 [Clu06a]. It uses the expression of Proposition 3.6 to compute the probability that a given word h belongs to $\mathcal{C}^\perp$. The decoding algorithm is presented in Table 3.4. We then give the results of some simulations of the algorithm described in table 3.6, which can be described as follows:.

- Search for words of low weight in the code $\mathcal{D}$ spanned by R^T. Return words which are candidates to be in $\mathcal{C}^\perp$;
- eliminate words h which creates short cycles in the graph associated to the code;
- use these parity check equations and the probabilities that they belong to $\mathcal{C}^\perp$ in order to correct some errors in the M noisy codewords using the algorithm described in Table 3.4.

We give simulations for LDPC codes of length 100 and 1000 and also for a random code of length 100 and dimension 50.

However, we have tried to observe what happened when false equations have been artificially added. Then, we have noticed that the algorithm in Table 3.4 does not decode anymore and, on the contrary that it adds errors. So, it clearly appears that the previous algorithm does not make what it was expected to. We then describe the origin of the problem. It comes from the formula we use to update the probability that an equation holds:

$$p_{e,i} = \mathrm{Est_e} \otimes \left[\bigotimes_{j \neq i} \widetilde{p}_{e,j} \right].$$

The two key-points to understand why this formula causes a misbehavior are the following:

- the first point is that when we combine two probabilities using the law $\otimes$, if one is close to 0 and the other close to 1, the result is close to $\frac{1}{2}$ which is the neutral element for the law $\otimes$;
- the second point is that, when we consider a random equation, there is a probability $\frac{1}{2}$ that this equation holds for a given word. So when we update the probability $p_{e,i}$ with our formula, it will be close to 1 as soon as the equation e is true for a little more than half of the words.

We want the probability $p_{e,i}$ to be close to $\frac{1}{2}$ if equation e is false in order to have $\mathrm{Extr}_e(c_t)$ close to $\frac{1}{2}$ for all t since this equation does not give us any information. On the contrary, if the equation e is true, we want to have $p_{e,i}$ close to 1 in order to recover the formula for $\mathrm{Extr}_e(c_t)$ in the original Gallager algorithm. So we only have to change this formula in our decoding algorithm: we now compute

$$p_{e,i} = \frac{\sum_{j \neq i} \widetilde{p}_{e,j}}{M-1}.$$

So, this decoding algorithm leads us to a new reconstruction algorithm described in Table 3.11. We then present some simulations on a [15,7,5] BCH code. We show that our algorithm is efficient even when there are false equations. Moreover, when there are false equations, the

algorithm manages to distinguish good ones from false ones as the values of p_e are higher when e is a correct parity check equation. We then make tests for LDPC codes. We show that for such codes, the value of M and T given by Theorem 2.7 are not optimal since the ratio

$$\alpha = \frac{\sharp\left\{h \notin \mathcal{C}^\perp \text{ such that } wt_H(hR^T) \leq T\right\}}{\sharp\left\{h \in \mathcal{C}^\perp \text{ such that } wt_H(hR^T) \leq T\right\}}$$

is adapted to our problem when the weight distribution of $\mathcal{C}^\perp$ is close to the one of a random code which is not the case for LDPC codes. In the situation of an LDPC code, all equations found by the statistical test presented in Chapter 2 are always correct, implying that the decoding algorithm does not lead to any improvement. However, we show that it nevertheless enables us to correct all errors in R even a basis of $\mathcal{C}^\perp$ has not be found by the statistical test. For this reason, in this case, it may be faster to decode all words of R than to construct a parity check matrix for our code. Finally, we have made simulations for random codes of length 100 and dimension 50. The results are presented in Tables 3.17 and 3.18.

Part II: reconstruction of a linear scrambler

This second part is devoted to the reconstruction of a linear scrambler. The main results can be found in [Clu04, Clu].

Chapter 4 Here, we present the problem and the two kinds of scramblers. We show that, in the general case, the exact output of the scrambler is not known but only its image by a linear transformation per block (Figure 4.1). Theorem 4.1 shows that a scrambler is a bijection in the sense that any sequence can be the output of any scrambler. Thus we will have to make some assumptions on the input stream in order to reconstruct the scrambler.

Chapter 5 In this chapter, we assume that the output sequence of the scrambler is known. We present different techniques for reconstructing a synchronous scrambler or a self-synchronized scrambler depending on the assumptions we make on the input sequence. First, we focus on a synchronous scrambler. We show that when the input sequence is known and if the known bits are consecutive, Berlekamp-Massey algorithm allows to reconstruct a scrambler of length L in

$$3\sum_{i=0}^{2L-1} i + L(M-2L) \simeq 4L^2 + LM \text{ operations,}$$

with $M = 2L_{\max}$ where $L_{\max}$ is the maximum possible length for the scrambler—since L is not known a priori. And if the input bits are not consecutive, we have to solve a linear system. In this case, the complexity would be in

$$\mathcal{O}\left(L_{\max}^3\right).$$

When the input stream is unknown but produced by a biased binary source, we present an algorithm inspired by [CF00]. The main idea introduced in [CF00] is that the statistical bias of the output stream can be exploited to detect any multiple of the feedback polynomial. However, in [CF00], all calculations are based on the assumption that the output of the LFSR $(z_t)_{t\in\mathbb{N}}$ corresponds to a sequence of independent random variables, which is obviously not the case. Then, we correct this wrong assumption. Namely Lemma 5.1 considers the case where $(z_t)_{t\in\mathbb{N}}$ is such that z_t and $z_{t'}$ are independent when $|t'-t| > d$ for a given integer d. Let $S = \sum_{t=0}^{N} z_t$, and let M and V denote the mean value of S and its variance. Then, when N grows to infinity, $\frac{S-M}{\sqrt{V}}$ can be approximated by a standard normal law. By applying this Lemma, Theorem 5.2 gives the mean value and the variance of the random variable

$$Z_N = \sum_{t=i_{d-1}}^{N-1} (-1)^{z_t},$$

with

$$z_t = y_t + \sum_{j=1}^{d-1} y_{t-i_j}, \forall t \geq i_{d-1}$$

when

$$Q(X) = 1 + \sum_{j=1}^{d-1} X^{i_j}, \text{ avec } i_1 < i_2 < \cdots < i_{d-1}.$$

is a multiple of the feedback polynomial P of the constituent LFSR.
So this theorem provides a method for distinguishing multiples of P from other polynomials. And, as the involved bias is larger when the weight of the polynomial is low, we will consider polynomials of low weight (in practice trinomials or polynomials of weight 4). In Table 5.1, we present our algorithm for recovering the feedback polynomial of the scrambler. It relies on a statistical test on the mean value of the random variable Z_N. Then we analyze its complexity and the optimal values for the differents parameters. For a scrambler of length L whose input satisfies $\Pr[x_t = 0] = \frac{1}{2} + \varepsilon$, we show that when multiples of weight d are considered, we need

$$N_P \lesssim (d-1)!^{\frac{1}{d-1}} 2^{\frac{L}{d-1}} + \frac{(a+b\delta)^2}{(2\varepsilon)^{2d}} \tag{1}$$

bits of the input stream. Then, the time complexity of the algorithm is:

$$\begin{aligned} W_P &= \frac{D^{d-1}}{(d-1)!} d(N-D) \\ &\lesssim d2^L \frac{(a+b\delta)^2}{(2\varepsilon)^{2d}}, \end{aligned}$$

with $a = \phi^{-1}\left(1 - \frac{P_f}{2}\right)$, $b = -\phi^{-1}(P_n)$ and $\delta = \sqrt{1 + 2d(d-1)}$ when P_f is the false-alarm probability and P_n is the non-detection probability.

Simulation results are presented in Tables 5.6 and 5.7.

Once the feedback polynomial of the scrambler has been found, we have to reconstruct the initial state of the register. The reconstruction of the initial state of the register is performed by a classical cryptographic technique of correlation attack based on low-weight parity-checks [CC98] but we present it in Table 5.9 with a description closer to iterative decoding.

For self-synchronized scramblers, when we assume that the input stream is known, the scrambler is recovered by solving a linear system, so in complexity

$$\mathcal{O}(L_{\max}^3).$$

When the input is unknown but biased, the technique is very similar to the one used for synchronous scramblers: we can detect the feedback polynomial as pointed out in Theorem 5.8. The algorithm we use is described in Table 5.12. The main difference with the synchronous case is that we can only detect the feedback polynomial of the register and not any multiple of it. The number of bits required by this algorithm is:

$$N_{\min} \simeq L_{\max} - b(a+b) + \frac{(a+b)^2}{(2\varepsilon)^2} \cdot \tag{2}$$

And the number of operations performed is roughly

$$(N_{\min} - L_{\max}) \sum_{d=2}^{w} d \binom{d-1}{L_{\max}} \simeq \frac{w L_{\max}^{w-1}}{(w-1)!} \cdot \frac{(a+b)^2}{(2\varepsilon)^2},$$

where w is the number of terms of the feedback polynomial.
Simulation results are presented in Tables 5.13 and 5.14.

Chapter 6 This chapter finally focuses on the general case, where the output of the scrambler is known only up to a linear transformation per block of length k, where k is the dimension of the error-correcting code we have previously recovered. We focus on synchronous scramblers. Let A denote the transition matrix of an LFSR. Theorem 6.2 first shows that, when the input vanishes, we can find an annihilator polynomial for A^k by finding a linear combination of the $\varphi(S_t)$ which vanishes, where S_t denote the t-th block of length k of the output of the scrambler as described on Figure 6.1.
Then, Theorem 6.3 shows that, when the input stream is known but does not vanish, an annihilator polynomial for A^k can also be found from the knowledge $L+k+1$ blocks of the input stream. Then, the aim is to find the characteristic polynomial of A, since it corresponds to the characteristic polynomial of the scrambler. The algorithm described in Table 6.1 shows how we can find the $\gcd(k,2^L-1)$ minimal polynomials of all k-th roots of the element α^k associated to the matrix A^k.
Theorem 6.6 then proves that the set of sequences generated by the $\gcd(2^L-1,k)$ polynomials corresponding to primitive roots of α^k are exactly the same. Thus, finding which

polynomial has been used during the transmission requires some information on the input stream of the scrambler. Actually equations

$$Y_t = (X_t \oplus S_t)B,$$

lead to a quadratic system with $(k^2 + L)$ unknowns corresponding to the coefficients of the linear transformation B and to the initial state of the LFSR. But, it has at most k^2L quadratic terms. Therefore, it can be solved by linearization in

$$\mathcal{O}\left(k^6L^3\right) \text{ operations.}$$

Introduction

Pour protéger la confidentialité des données, on utilise des procédés cryptographiques dits de chiffrement. On sait, depuis le célèbre article de Kerckhoffs *La Cryptographie militaire* de 1883 que, pour offrir une sécurité raisonnable, une technique de chiffrement doit reposer sur le secret d'une quantité courte, la clé, et non sur le secret du procédé employé. Ce principe fondamental a été énoncé par Kerckhoffs : « Il faut que [le système] n'exige pas le secret [...] Et ici j'entends par secret, non la clef proprement dite, mais ce qui constitue la partie matérielle du système : tableaux, dictionnaires ou appareils mécaniques quelconques qui doivent en permettre l'application. »

Ainsi, pour évaluer la qualité d'un algorithme cryptographique, on suppose toujours que ses spécifications sont publiques puisque, conformément au principe précédent, il serait dangereux de vouloir faire reposer la sécurité, ne serait-ce qu'en partie, sur le fait que les spécifications du système ne sont pas connues de l'attaquant. Toutefois, cette règle fondamentale en cryptographie ne signifie pas pour autant que les spécifications du système utilisé sont publiques en pratique. Quand cela n'est pas le cas, toute attaque nécessite au préalable une phase de rétro-ingénierie permettant de reconstituer le système employé. Pour les procédés de chiffrement, cette opération implique souvent la fuite d'informations sur les spécifications du système ou l'analyse précise du matériel le mettant en œuvre. La reconstitution de la machine Enigma pendant la deuxième guerre mondiale, la divulgation sur Internet des spécifications de l'algorithme RC4 utilisé par SSL ou de celles du chiffrement A5/1 employé dans le standard GSM, en sont de quelques exemples.

Mais cette phase de rétro-ingénierie reste souvent délicate même lorsque les spécifications ne sont pas dissimulées à dessein, mais simplement inconnues de l'attaquant, et que le système n'inclut aucune fonction cryptographique. On est face à ce type de situation par exemple quand l'on veut écouter une communication non chiffrée. En effet, même lorsque l'on a intercepté la communication, il est nécessaire pour accéder à l'information transmise de « désassembler » le système de transmission qui a transformé le message d'origine en un signal en général non-intelligible si l'on ne sait pas comment il a été modifié. Un système de transmission est usuellement formé d'une succession d'éléments ayant chacun une raison d'être précise, mais il existe souvent une grande variété de possibilités pour chacun de ces éléments. Ainsi, pour ne mentionner qu'un seul standard, les dispositifs utilisés pour corriger les erreurs de transmission dans le protocole Ethernet décrit par la norme IEEE 802.3 varient avec le médium utilisé (paire torsadée, coaxial, fibre optique...) et avec le débit de transmission (10 Mbits/s pour la définition d'origine, puis 100 Mbits/s, 1 Gbit/s et

maintenant 10 Gbits/s). La dernière version de IEEE 802.3, baptisée 10GBASE-T adoptée le 17 juillet 2006, utilise ainsi pour la première fois des codes à matrice de parité creuse alors que les versions précédentes comme la 1000Base-T (IEEE 802.3ab) employaient un code convolutif. On voit donc très clairement, même si on ne considère qu'un seul standard de communication, que l'opération de rétro-ingénierie permettant d'identifier les caractéristiques du système de transmission est cruciale pour pouvoir reconstituer l'information correspondant à une communication interceptée.

Les travaux présentés dans cette thèse sont consacrés à ce problème : notre objectif est de retrouver les spécifications d'un système de transmission à partir de la connaissance d'une communication interceptée, laquelle sera, en plus, généralement bruitée du simple fait des erreurs de transmission. Il existait jusqu'à très récemment relativement peu de travaux publiés sur ce sujet, mis à part les travaux de Bart Rice et Éric Filiol sur la reconstruction d'un code convolutif [Ric95, Fil97, Fil00b] poursuivis récemment par Johann Barbier pour les turbo-codes [Bar05], et la thèse d'Antoine Valembois [Val00] pour la reconstruction des codes en blocs.

Nous allons ici nous focaliser sur la reconstruction des deux éléments importants d'un système de transmission classique, le code linéaire en blocs et le brasseur. Les deux grandes parties de ce document s'intéresseront donc successivement à ces deux maillons de la chaîne, dans l'ordre dans lequel ils doivent être traités par l'attaquant au moment de l'interception, c'est-à-dire d'abord le code en bloc puis le brasseur. Ces deux parties sont précédées d'un chapitre introductif qui décrit les différents éléments d'une chaîne de communication et qui détaille en particulier les propriétés classiques des deux éléments qui nous allons ensuite chercher à retrouver.

Notre travail sur la reconstruction des codes linéaires en blocs s'est tout d'abord appuyé sur les résultats d'Antoine Valembois qui a montré dans sa thèse que ce problème était dans la classe des problèmes NP-durs, et qui a décrit un algorithme pour le résoudre. Cet algorithme, qui consiste en fait à rechercher des mots de poids faible dans un certain code construit à partir des mots interceptés, est donc présenté au chapitre 2. Nous analysons en particulier précisément le test statistique proposé par Valembois. Cette analyse est en effet très importante car elle permet de déterminer la quantité d'information qu'il va falloir intercepter, la complexité en temps de la reconstruction, et surtout son taux de succès. Cette analyse et les simulations effectuées nous ont permis de montrer que le test statistique de Valembois n'était pas très pertinent. Nous proposons donc un nouveau test statistique qui paraît plus adapté à notre problème.

Ensuite, nous présentons dans le chapitre 3 un algorithme inspiré des algorithmes classiques de décodage itératif afin de tenter d'utiliser les équations de parité de poids faible, obtenues lors de la phase de recherche de mots de poids faible dans le code construit à partir des mots interceptés, pour tenter de corriger quelques erreurs. Nous présentons en fait dans ce chapitre deux algorithmes : une première tentative qui nous a paru naturelle et qui a été présenté en juillet dernier à Seattle lors du colloque ISIT (IEEE International Symposium on Information Theory) [Clu06a]. Nous verrons ensuite que la poursuite des simulations a montré que cet algorithme se comportait mal lorsqu'il y avait des équations

de parité fausses, c'est-à-dire des équations qui n'étaient pas des équations de parité pour le code utilisé lors de la transmission, alors que, justement, cet algorithme avait été conçu pour permettre de décoder même si certaines équations étaient erronées. Nous avons donc modifié une formule de remise à jour de la probabilité des équations de parité qui était à l'origine de ce dysfonctionnement. Nous présentons alors un algorithme itératif qui permet de corriger quelques erreurs même si certaines équations ne sont pas des équations de parité pour notre code. Cet algorithme est suivi de l'analyse de simulations pour différents types de codes.

La deuxième grande partie de cette thèse traite du problème de la reconstruction d'un brasseur linéaire. Une fois que nous avons retrouvé le code linéaire en bloc utilisé, il faut retrouver ce deuxième maillon du système de communication. Nous montrons d'abord que le fait qu'on ne puisse jamais retrouver le codeur utilisé dans la pratique et ce, même si l'on a retrouvé le code utilisé, a forcément un impact sur la reconstruction du brasseur : dans le cas général, on ne connaîtra donc pas la sortie du brasseur mais seulement l'image de celle-ci par une transformation linéaire par bloc. Dans tous les cas, le brasseur étant une bijection, il nous faudra faire des hypothèses sur la suite en entrée du système de communication. Nous avons envisagé deux hypothèses principales :

- la connaissance de certains bits de la suite entrante ;
- la suite en entrée est inconnue mais est produite par une source binaire biaisée.

Dans le chapitre 5, nous supposons que nous connaissons la sortie exacte du brasseur. Nous présentons alors différentes techniques pour reconstruire les deux types de brasseurs linéaires : les brasseurs synchrones et les brasseurs auto-synchronisants. La majeure partie des résultats de ce chapitre ont été présentés à Chicago en juillet 2004 lors du colloque ISIT [Clu04]. Dans le cas où l'on suppose que certains bits de la suite en entrée du système sont connues, les différentes techniques de reconstruction reposent principalement sur l'algorithme de Berlekamp-Massey et sur la résolution de systèmes linéaires. Dans le cas où l'entrée du brasseur est inconnue mais produite par une source binaire biaisée, la recherche du polynôme de rétroaction du registre s'inspire d'un algorithme proposé par Anne Canteaut et Éric Filiol pour la reconstruction de certains générateurs pseudo-aléatoires. Nous montrons par ailleurs que le théorème sur lequel est fondée cette technique repose sur une hypothèse erronée ; nous l'avons donc modifié pour qu'il ne prenne plus en compte cette hypothèse. Pour la reconstruction de l'état initial du registre, nous utilisons les techniques classiques de cryptanalyse par corrélation utilisées en cryptographie mais avec une description plus axée vers les algorithmes de décodage itératif.

Enfin, dans le chapitre 6, nous abordons le problème général de la reconstruction du brasseur lorsqu'on ne connaît sa sortie qu'à une transformation linéaire près. Plus précisément, on se place dans le contexte d'un brasseur synchrone dont on connaît l'entrée mais dont la sortie n'est disponible qu'à équivalence près. Nous présentons une méthode de reconstruction totalement algébrique et permettant de traiter efficacement ce cas.

Chapitre 1

Introduction à la reconstruction d'une chaîne de transmission

L'objectif de cette thèse est d'analyser et de proposer des techniques pour reconstruire un système de communication à partir d'un signal intercepté lors de la transmission. On nous fournit une suite binaire provenant de l'écoute d'une transmission ; on sait que cette transmission a été effectuée à l'aide d'un système de communication mais on ignore ses caractéristiques et le but de mes travaux est de retrouver les spécifications du système. On va donc tout d'abord rappeler rapidement les différents éléments d'un système de communication pour en donner une modélisation classique et faire ressortir les différents problèmes que nous allons ensuite traiter dans ce manuscrit.

1.1 Système de communication

Par définition, une transmission numérique permet d'acheminer des messages de nature numérique entre une source et un destinataire. Cette source délivre une suite de symboles qui prennent leurs valeurs dans un ensemble fini appelé alphabet. Les transmissions numérique ont fait l'objet de nombreuses études depuis la publication en 1948 des travaux de Shannon [Sha48]. Le schéma de base d'une chaîne de transmission numérique est représenté par la figure 1.1. La source produit le message à transmettre et le destinataire traite le message numérique reçu. L'élément central de cette chaîne de transmission est le canal de transmission qui constitue le lien entre émetteur et récepteur. L'émetteur fournit le signal porteur du message qui doit transiter par le canal de transmission et le récepteur effectue l'opération inverse et délivre le message au destinataire.

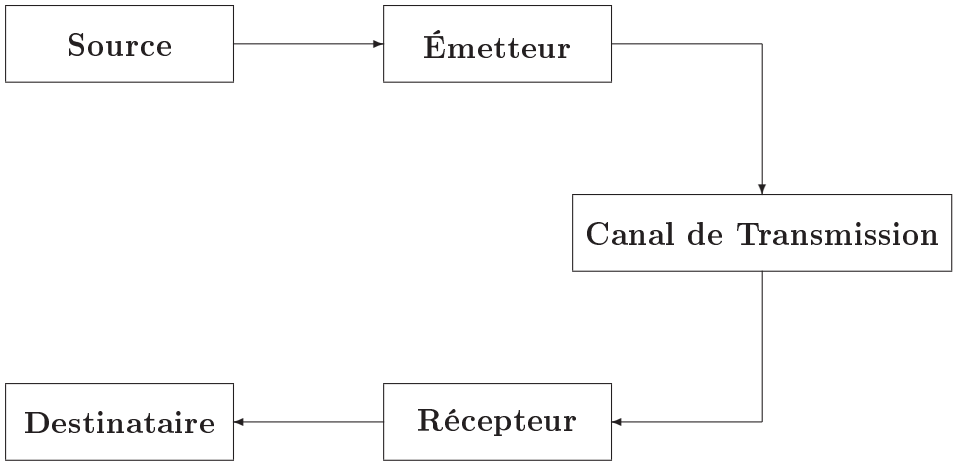

FIG. 1.1 – *Schéma de base d'une chaîne de transmission numérique.*

1.1.1 La source d'information

Le modèle général est une source numérique qui délivre des symboles prenant leurs valeurs dans l'alphabet. L'alphabet contient souvent seulement deux éléments, notés 0 et 1 — on parle alors de source binaire ; mais dans le cas général il comporte q éléments et on parle de source q-aire. La suite des symboles produits par la source constitue le message numérique à transmettre. La source est caractérisée par un débit alphabétique exprimé en nombre de symboles par seconde. On définit également un débit d'information exprimé en shannons par seconde ; cette notion relève de la théorie de l'information. Le lien entre le débit alphabétique et le débit d'information dépend des propriétés statistiques de la source. En général le modèle de source que l'on considère est une source idéale stationnaire qui délivre des symboles indépendants dont les valeurs possibles sont équiprobables. Cette source est la plus efficace puisque la quantité d'information portée en moyenne par chaque symbole est maximale [Rou70]. Pour une source binaire idéale, le débit d'information est égal au débit alphabétique. Lorsque l'on souhaite transmettre un message quelconque, on doit d'abord le transformer en une suite de symboles appartenant à l'alphabet souhaité en sortie de la source. Par exemple, si le message à envoyer est un message analogique, on doit d'abord le transformer en un message numérique. Ensuite, si les caractéristiques du message sont trop éloignées du modèle de la source parfaite, on a intérêt à traiter le message numérique afin de lui ôter un maximum de redondance. Cette opération est appelée codage de source, on peut d'ailleurs modéliser la source par une source d'information et un codeur de source. On peut parallèlement modéliser le « destinataire » en un décodeur de source et un destinataire qui reçoit des messages et non plus des éléments de l'alphabet. Le schéma devient alors le suivant :

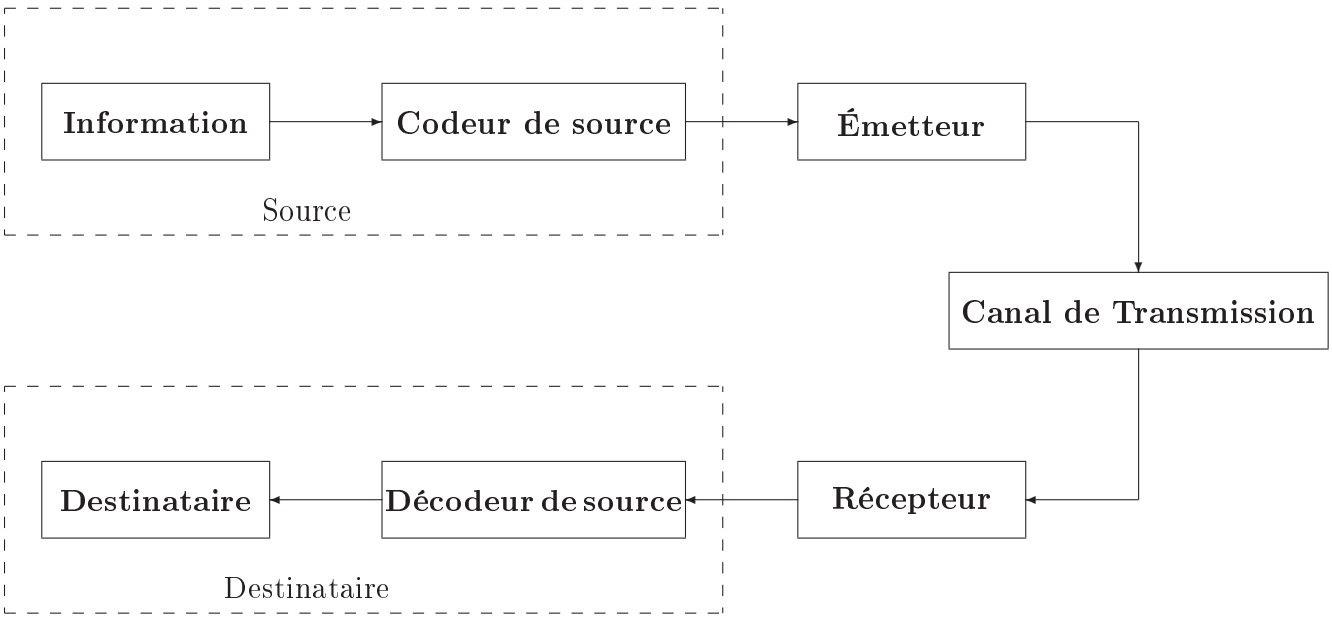

FIG. 1.2 – *Schéma d'une chaîne de transmission en intégrant le codage/décodage de source.*

La source idéale étant un modèle théorique, on doit se demander si la réalité de la source considérée coïncide avec ce modèle. Dans certains cas, le message à transmettre n'est pas aléatoire. Il arrive alors fréquemment qu'une modification notable des caractéristiques de la source réelle se produise (par exemple lorsque la source cesse de fonctionner). Comme les changements de propriétés statistiques risquent d'avoir des conséquences néfastes sur le fonctionnement du système de transmission, on utilise souvent une astuce qui consiste à brasser/débrasser le message numérique. Le rôle du brasseur est de réduire les modifications des caractéristiques du message numérique émis. Pour des raisons de commodité, l'opération de brassage est souvent effectuée par l'émetteur et le débrassage s'effectue dans le récepteur.

1.1.2 Le destinataire

Le destinataire reçoit une suite de symboles à partir de laquelle il reconstitue le message. Les opérations inverses de celles effectuées côté source doivent donc être effectuées par le destinataire. Il doit ainsi faire un décodage de source si un codage de source a été effectué et éventuellement convertir le message numérique en un message analogique. Le problème de la qualité du message reconstitué vient naturellement à l'esprit. Deux questions essentielles se posent : la suite de symboles délivrée au destinataire ne diffère-t-elle pas trop de celle fournie par la source ? Le destinataire a-t-il reçu le message dans un délai convenable ?

La première question pose le problème de la qualité de la transmission numérique proprement dite. En raison des perturbations présentes sur le canal de transmission (voir le paragraphe suivant), les symboles reçus par le destinataire ne sont pas tous identiques

aux symboles fournis par la source : il y a apparition d'erreurs de transmission. On définit le taux d'erreur sur les symboles (TES) comme le rapport du nombre de symboles erronés sur le nombre de symboles transmis. On définit de manière semblable le taux d'erreur sur les bits (TEB) lorsque les symboles sont des bits ou des mots binaires. La qualité de transmission est d'autant meilleure que le taux d'erreur est faible. L'idéal est d'obtenir un taux d'erreur nul, mais cet idéal ne peut être qu'approché au prix de dépenses diverses [Sha48]. Selon la nature du message, on fixe un taux d'erreur à ne pas dépasser de façon à assurer au destinataire une qualité minimale du message utile.

La seconde question pose le problème du temps écoulé entre l'instant où le message est envoyé et l'instant où il est exploité par l'utilisateur. Ce délai est à mettre au compte de la propagation dans le milieu physique utilisé, et également au compte de tous les traitements subis par le message au cours de la transmission. Selon la nature du message, le délai est un paramètre plus ou moins important de la transmission. Par exemple, une communication téléphonique est désagréable si ce délai dépasse quelques dixièmes de seconde.

1.1.3 Le canal de transmission

Le propre d'une communication étant de se faire à distance, il faut utiliser un milieu physique pour assurer le lien entre la source et le destinataire. On doit aussi choisir un signal approprié à ce milieu physique. Citons par exemple, un signal électrique dans un milieu conducteur ou un signal électromagnétique en espace libre. Ici, on considère le canal de transmission au sens de la théorie de l'information. Il reçoit en entrée des symboles appartenant à un alphabet d'entrée et délivre en sortie des symboles appartenant à un alphabet de sortie, les deux alphabets pouvant être identiques ou différents. Le canal de transmission inclut donc le milieu physique mais aussi des organes d'émission et de réception comme la transformation de la suite de symboles en un signal pouvant transiter dans le milieu physique considéré. On peut définir la modulation comme le processus par lequel le message est transformé de sa forme originale en une forme adaptée à la transmission. L'opération inverse est appelée démodulation. On peut donc modéliser le canal de transmission de la manière suivante :

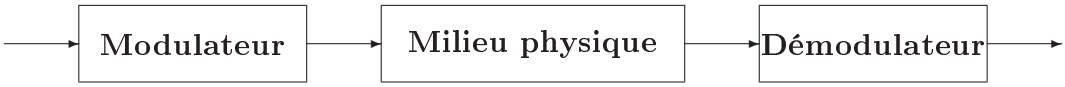

FIG. 1.3 – *Canal de transmission.*

On n'abordera pas ici tous les principes de modulation et de démodulation qui ne seront pas utiles pour ce manuscrit — le lecteur intéressé pourra se référer par exemple à [LM92]. On va maintenant s'intéresser à deux exemples classiques de canaux de transmission : le canal binaire symétrique et le canal à bruit blanc additif gaussien.

1. **Le canal binaire symétrique.**
   Les symboles considérés ici sont des bits. La modulation va donc se charger de

transformer une suite de bits en un signal électrique (ou électromagnétique) et on va transmettre ce signal dans un milieu conducteur (ou en espace libre). Ensuite, la démodulation va permettre de repasser du signal électrique (ou électromagnétique) à une suite binaire. On va modéliser tout le canal de transmission de la manière suivante.
La plupart du temps lorsque l'on émet un bit qui vaut 0, on reçoit 0, mais occasionnellement il se peut que l'on reçoive un 1 alors qu'un 0 a été envoyé ou 0 alors qu'on a envoyé un 1. Disons qu'en moyenne cela arrive une fois sur 100, c'est-à-dire que, pour tout symbole binaire transmis, on a une probabilité $\frac{1}{100}$ que le canal produise une erreur. Ce type de canal s'appelle un canal binaire symétrique avec probabilité d'erreur $\tau = \frac{1}{100}$.

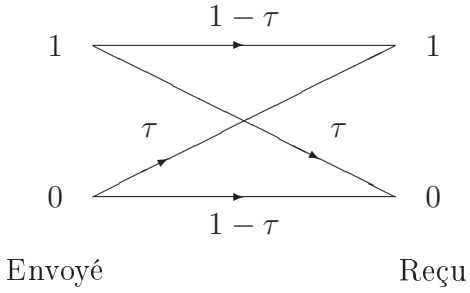

FIG. 1.4 – *Canal binaire symétrique de probabilité d'erreur τ.*

2. **Le canal à bruit blanc additif gaussien.**
C'est un modèle fréquemment utilisé car il décrit souvent bien la transmission dans un milieu physique. Après modulation, on a un signal en entrée du canal ; en sortie du canal, le signal reçu résulte de l'addition du signal émis et d'un bruit. Pour ce canal, le bruit est modélisé par une variable aléatoire complexe de loi normale $\mathcal{N}(0,N_0)$. Sa projection sur un axe réel est donc une variable aléatoire réelle de loi normale $\mathcal{N}\left(0,\frac{N_0}{2}\right)$. Puisque l'on s'intéresse souvent aux distorsions subies dans une direction donnée, on invoque alors la densité mono-latérale de bruit $\frac{N_0}{2}$. On peut montrer que sous certaines modulations et démodulations, pour un alphabet binaire, le canal de transmission composé du modulateur, du canal à bruit blanc gaussien et du démodulateur peut être assimilé à un canal binaire symétrique de probabilité d'erreur

$$\tau = \frac{1}{\sqrt{2\pi}} \int_{\sqrt{\frac{2}{N_0}}}^{\infty} \exp\left(\frac{-t^2}{2}\right) dt.$$

1.1.4 Émetteur-Récepteur

Pour l'instant, nous avons vu apparaître deux grands problèmes :
– la source ne peut pas toujours être considérée comme une source parfaite. Pour re-

médier à ce problème, nous avons vu qu'une solution courante était d'intégrer un brasseur à l'émetteur (et un débrasseur au récepteur).

– lors de la transmission, des erreurs se produisent.

Il nous reste ce second problème à résoudre. Si l'on souhaite transmettre des messages importants sur ce canal et que l'on veut s'assurer que ces messages soient bien reçus sans erreur, on va les *encoder*[1] au moyen d'un code correcteur pour les protéger contre les erreurs du canal. Par exemple, les aviateurs vont transmettre à la tour de contrôle « Alpha Tango Charlie » dans le seul but de transmettre correctement « ATC » à travers leur radio. La séquence « Alpha Tango Charlie », même déformée par la friture, sera bien plus reconnaissable pour l'oreille humaine qu'un « ATC » déformé.

On considère ici des données numériques binaires ; il existe deux grandes familles de codes : les codes convolutifs et les codes en bloc. Les codes convolutifs ne nécessitent pas de découper le message en blocs de taille fixe : ils agissent directement sur les bits en les codant un à un. Les codes en blocs découpent le message en entrée en blocs de taille fixe. On considère donc un bloc de message de k bits $m = m_1 m_2 \dots m_k$ ($m_i = 0$ ou 1) qui va être codé par un mot de code $c = c_1 c_2 \dots c_n$ ($c_i = 0$ ou 1) avec $n \geq k$. Ces mots de code forment un sous-ensemble de $\mathbf{F}_2^n$ qui sera appelé code correcteur d'erreur. On peut donc modéliser notre système de communication en intégrant le brasseur et le code correcteur d'erreur de la manière suivante :

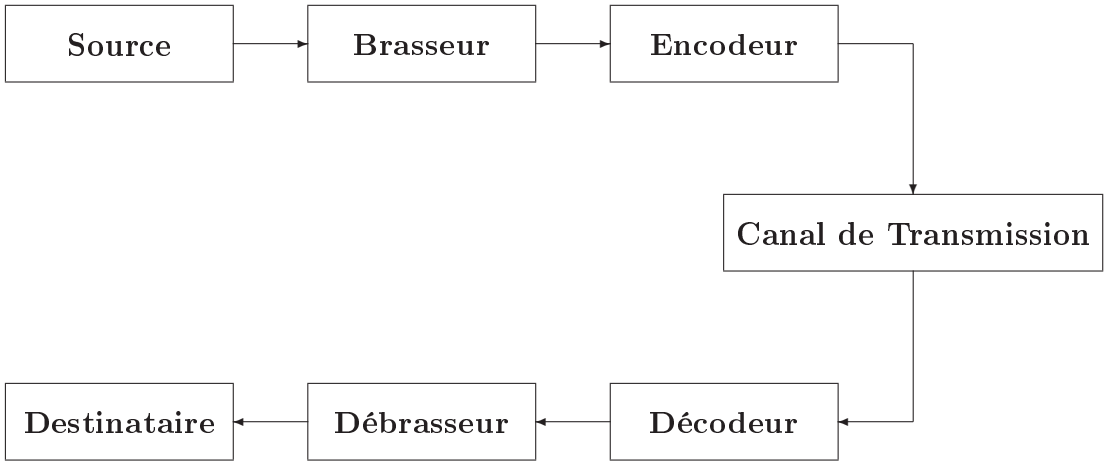

FIG. 1.5 – *Schéma d'une chaîne de transmission.*

L'objectif de cette thèse est le suivant : on a intercepté sur le canal un signal provenant d'un système de communication et on veut essayer de retrouver les éléments du système de communication, c'est-à-dire le brasseur et le code correcteur qui ont été utilisés. Notons que le message intercepté est généralement entaché d'erreurs puisqu'il a été enregistré au

1. On utilisera ici l'anglicisme *encoder* qui met mieux en évidence la nuance entre le code et la fonction de codage, distinction qui apparaîtra de manière fondamentale par la suite, au cours du processus de reconstruction de la chaîne de transmission.

niveau du canal de transmission. On ne s'intéresse pas ici aux problèmes de démodulations : on suppose que l'on connaît la suite binaire provenant de la transmission. Nous pouvons désormais décomposer le problème à résoudre en deux sous-problèmes :

- retrouver le code correcteur qui a été utilisé lors de la transmission. Il faudra par la suite (ou en même temps) décoder, c'est-à-dire enlever les erreurs de transmission du message intercepté). Nous supposerons que le code utilisé appartient à la famille des codes en bloc ; en effet le cas des codes convolutifs a déjà été traité par B. Rice puis de manière plus approfondie par E. Filiol et il existe des algorithmes efficaces pour les reconstruire [Ric95, Fil01, Fil97, Fil00b]. Le cas plus complexe des turbos-codes a lui été abordé récemment par J. Barbier [Bar05] ;
- retrouver le brasseur du système.

Commençons donc d'abord par décrire plus précisément les éléments que l'on souhaite retrouver : un brasseur et un code correcteur d'erreur afin de savoir quelles sont les spécifications à déterminer.

1.2 Les brasseurs

Nous nous intéresserons ici aux brasseurs et plus particulièrement aux brasseurs linéaires qui sont plus faciles à mettre en oeuvre en hardware puisqu'ils nécessitent uniquement un registre à décalage à rétroaction linéaire. En pratique, ce sont d'ailleurs les seuls brasseurs utilisés. On distingue deux types de brasseurs linéaires, les brasseurs synchrones et les brasseurs autosynchronisants, en fonction de la technique utilisée pour combiner le train binaire de départ et la suite pseudo-aléatoire générée par un registre à décalage à rétroaction linéaire (c.f. figure 1.6).

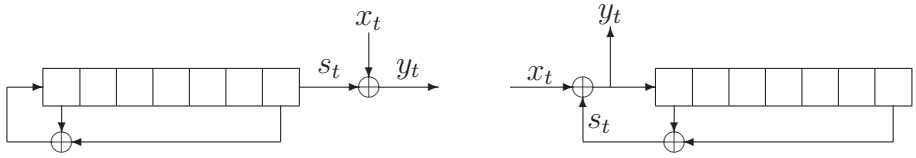

FIG. 1.6 – *Brasseur linéaire synchrone (à gauche) et autosynchronisant (à droite)*

Dans un système de communication, le rôle du brasseur est de diminuer la longueur des paquets de bits identiques qui pourraient occasionner des problèmes de synchronisation lors de la phase de codage. Comme l'élément principal d'un brasseur est un registre à décalage à rétroaction linéaire, nous allons faire ici quelques rappels sur ce type d'objet.

1.2.1 Les registres à décalage à rétroaction linéaire

Définition 1.1 *Un* registre à décalage à rétroaction linéaire *(LFSR pour Linear Feedback Shift Register) binaire de longueur L est un dispositif composé d'un registre à décalage contenant L bits $(s_t, \ldots, s_{t+L-1})$ et d'une fonction de rétroaction linéaire.*

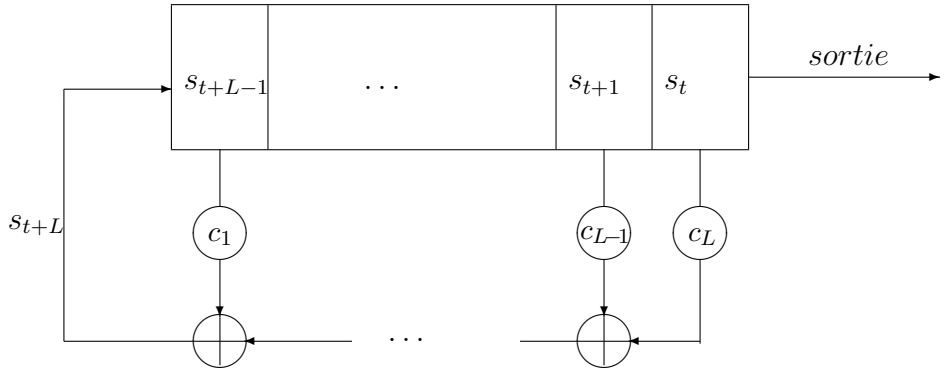

FIG. 1.7 – *Fonctionnement d'un LFSR*

A chaque top d'horloge, le bit de droite s_t constitue la sortie du LFSR et les autres bits sont décalés d'une position vers la droite. Le nouveau bit s_{t+L} placé dans la cellule de gauche du registre est donné par une fonction linéaire des bits $(s_t, \ldots, s_{t+L-1})$:

$$s_{t+L} = c_1 s_{t+L-1} + c_2 s_{t+L-2} + \cdots + c_L s_t \tag{1.1}$$

où les coefficients de rétroaction $(c_i)_{1 \leq i \leq L}$ sont des éléments de $\mathbf{F}_2$.

Définition 1.2 *Les bits $(s_0, \ldots, s_{L-1})$, qui déterminent entièrement la suite, constituent* l'état initial *du registre.*

La plupart des propriétés des LFSRs que nous donnons ici sont bien connues et figurent par exemple dans le chapitre 8 de [LN83] ou dans [Til05].

La suite $(s_t)_{t \geq 0}$ produite par un LFSR de longueur L est donc une suite régie par une récurrence linéaire homogène d'ordre L. Inversement, ce type de suite peut toujours être produit par un LFSR. On peut remarquer qu'une telle suite $(s_t)_{t \geq 0}$ est ultimement périodique, *i.e.* qu'il existe un certain entier t_0 à partir duquel la suite $(s_t)_{t \geq t_0}$ est périodique.

Proposition 1.3 *Toute suite binaire produite par le LFSR de la figure 1.7 de longueur L est ultimement périodique, et sa plus petite période T est inférieure ou égale à $2^L - 1$. De plus, si le coefficient c_L est non nul alors la suite est périodique.*

On peut voir un LFSR de manière plus algébrique. En effet on peut utiliser des séries pour décrire une suite à récurrence linéaire. On associe naturellement à la suite $(s_t)_{t \geq 0}$ la série formelle $s(X) = \sum_{t \geq 0} s_t X^t$. On introduit alors un polynôme appelé *polynôme de rétroaction* du LFSR de la manière suivante :

Définition 1.4 *Soit un LFSR dont la fonction de rétroaction est donnée par la relation (1.1). Son polynôme de rétroaction P est le polynôme de $\mathbf{F}_2[X]$:*

$$P(X) = 1 + c_1 X + c_2 X^2 + \cdots + c_L X^L.$$

Il arrive aussi souvent qu'on ait besoin du polynôme réciproque du polynôme de rétroaction.

Définition 1.5 *Soit un LFSR dont la fonction de rétroaction est donnée par la relation (1.1). Son* polynôme caractéristique *est le polynôme réciproque du polynôme de rétroaction ; c'est donc le polynôme de* $\mathbf{F}_2[X]$

$$P_c(X) = X^L + c_1 X^{L-1} + c_2 X^{L-2} + \cdots + c_L.$$

On peut maintenant écrire le développement en série formelle de la suite $(s_t)_{t\geq 0}$ sous forme d'une fraction rationnelle.

Proposition 1.6 *La suite* $(s_t)_{t\geq 0}$ *est produite par un LFSR de polynôme de rétroaction* P *si et seulement si son développement en série s'écrit*

$$s(X) = \frac{Q(X)}{P(X)}$$

où Q *est un polynôme de* $\mathbf{F}_2[X]$ *tel que* $\deg(Q) < \deg(P)$. *De plus,* Q *ne dépend que de l'initialisation du registre et des coefficients de rétroaction :*

$$Q(X) = \sum_{i=0}^{L-1} X^i \sum_{j=0}^{i} c_{i-j} s_j.$$

Une conséquence assez directe de cette proposition est le résultat suivant.

Proposition 1.7 *Notons* $S(P)$ *l'ensemble des suites produites par un LFSR de polynôme de rétroaction* P. *Soient* P *et* Q *deux polynômes non constants sur* $\mathbf{F}_2$. *Alors* $S(P)$ *est inclus dans* $S(Q)$ si et seulement si P *divise* Q.

Cette proposition entraîne que si une suite $(s_t)_{t\geq 0}$ est générée par un LFSR de polynôme de rétroaction P, alors elle vérifie les relations de récurrence correspondant à tout multiple de P, *i.e.* à PQ pour tout $Q \in \mathbf{F}_2[X]$.

Pour obtenir une forme canonique de la série génératrice associée à la suite $(s_t)_{t\geq 0}$, on définit le polynôme de plus bas degré P_0 parmi les polynômes de rétroaction de tous les LFSRs qui génèrent la suite $(s_t)_{t\geq 0}$. Il s'agit donc d'un diviseur de P qui permet notamment de définir le polynôme minimal de la suite.

Définition 1.8 *Soit* $(s_t)_{t\geq 0}$ *une suite binaire régie par une récurrence linéaire. Il existe un unique polynôme unitaire* P_0 *tel que le développement en série formelle associé est donné par*

$$s(X) = \frac{Q_0(X)}{P_0},$$

où pgcd$(Q_0,P_0) = 1$. *Ainsi le plus petit LFSR permettant d'engendrer* $(s_t)_{t\geq 0}$ *a pour longueur*

$$L_0 = \max(\deg(P_0), \deg(Q_0) + 1),$$

et son polynôme de rétroaction est égal à P_0. *Le polynôme réciproque de* P_0, $X^{L_0}P_0\left(\frac{1}{X}\right)$, *est donc le polynôme caractéristique du plus petit LFSR produisant* $(s_t)_{t\geq 0}$. *Il est appelé* polynôme minimal *de la suite. Le degré* L_0 *du polynôme minimal de la suite s'appelle la* complexité linéaire *de la suite* $(s_t)_{t\geq 0}$, *notée* $\Lambda(s)$.

Exemple 1.1 *Soit* $(s_t)_{t\geq 0}$ *la suite produite par le LFSR de longueur 10, de polynôme de rétroaction*

$$P(X) = X^{10} + X^8 + X^7 + X^6 + X^5 + 1$$

et qui commence par $s_0 s_1 \dots s_9 = 1001001001$. *Soit* Q *le polynôme déterminé par l'état initial du registre :*

$$Q(X) = \sum_{n=0}^{9} X^i \sum_{j=0}^{n} c_{n-j} s_j = 1 + X^3 + X^5 + X^7.$$

La série génératrice de $(s_t)_{t\geq 0}$ *est :*

$$s(X) = \sum_{t=0}^{\infty} s_t X^t = \frac{Q(X)}{P(X)} = \frac{1 + X^3 + X^5 + X^7}{X^{10} + X^8 + X^7 + X^6 + X^5 + 1} = \frac{1}{X^3 + 1}.$$

On a donc $P_0(X) = X^3 + 1$, *ce qui implique que* $(s_t)_{t\geq 0}$ *peut être générée par un LFSR de longueur 3. La complexité linéaire de la suite* $(s_t)_{t\geq 0}$ *est donc 3.*

Il est clair avec ce qui précède que si l'on a pris soin d'utiliser un polynôme de rétroaction irréductible, alors la fraction ne pourra être réduite, et l'on est alors certain de ne pas pouvoir générer la même suite avec un registre plus court. Un autre paramètre essentiel pour une suite produite par un LFSR est sa période. Celle-ci est déterminée par l'ordre du polynôme de rétroaction du LFSR.

Définition 1.9 *Soit* P *un polynôme de* $\mathbf{F}_2[X]$. *Son ordre, noté ord*(P), *est le plus petit entier* $t > 0$ *tel que :*

$$X^t \equiv 1 \mod P(X)$$

Proposition 1.10 *Soit* $(s_t)_{t\geq 0}$ *une suite binaire à récurrence linéaire de polynôme de rétroaction minimal* P_0, *et dont l'état initial est non nul. Alors sa plus petite période est égale à l'ordre de* P_0.

On peut donc contrôler la période de la suite produite en contrôlant l'ordre du polynôme de rétroaction.

Définition 1.11 *Soit* P *un polynôme irréductible de* $\mathbf{F}_2[X]$, *de degré* L. *Il est dit* primitif *s'il est d'ordre* $2^L - 1$.

Ainsi, si on veut utiliser un LFSR de longueur L optimal (au regard de la période de la suite produite), on doit alors s'assurer que le polynôme de rétroaction de degré L choisi est primitif. Un autre intérêt des polynômes de rétroaction primitifs est la qualité statistique des suites générées. Ces dernières vérifient par exemple les propriétés suivantes.

Proposition 1.12 *Considérons la suite produite par un LFSR de longueur L. Si le polynôme de rétroaction P est primitif, et si l'état initial est non nul, alors :*

- *toutes les suites binaires de longueur $\ell < L$ se retrouvent avec la même fréquence dans la suite produite : les sous-suites non nulles apparaissent chacune $(2^{L-\ell} - 1)$ fois, et les sous-suites nulles apparaissent chacune $2^{L-\ell}$ fois ;*
- *toutes les suites binaires non nulles de longueur L apparaissent également avec la même fréquence.*

C'est pourquoi les suites produites par des LFSRs avec un polynôme de rétroaction primitif sont particulièrement intéressantes et qu'en pratique on choisit toujours un polynôme de rétroaction de ce type.

1.2.2 L'algorithme de Berlekamp-Massey

En 1968, E. Berlekamp a présenté dans son livre [Ber68] un algorithme de décodage des codes BCH. Un an plus tard, J.L. Massey a montré dans [Mas69] que cet algorithme permettait également de trouver le plus petit LFSR générant une suite $(s_t)_{t\geq 0}$ à partir uniquement de ses $2\Lambda(s)$ premiers bits. Cet algorithme, connu désormais sous le nom de Berlekamp-Massey, consiste à construire pour les valeurs successives de N un LFSR de longueur minimale L_N et de polynôme de rétroaction f_N qui génère les N premiers bits de la suite $(s_t)_{t\geq 0}$. Par convention, on suppose que la suite nulle est générée par le LFSR de longueur 0 et de polynôme de rétroaction $P(X) = 1$. L'algorithme repose sur le lemme suivant.

Lemme 1.13 *Soit L_N la longueur minimale d'un LFSR qui génère les bits $s_0,s_1,\ldots,s_{N-1}$ mais qui ne génère plus $s_0,s_1,\ldots,s_{N-1},s_N$. Alors la longueur minimale L_{N+1} d'un LFSR générant $s_0,s_1,\ldots,s_{N-1},s_N$ vérifie :*

$$L_{N+1} \geq \max(N + 1 - L_N, L_N).$$

Proposition 1.14 *Soit $(s_t)_{t\geq 0}$ une suite binaire à récurrence linéaire. Considérons un LFSR de longueur minimale L_N et de polynôme de rétroaction $P_N(X) = 1 + \sum_{i=1}^{L_N} c_i^N X^i$ qui génère les N premiers bits de la suite $(s_t)_{t\geq 0}$. On pose $L_0 = 0$ et $f_0(X) = 1$. Soit*

$$d_N = s_N + \sum_{i=1}^{L_N} c_i^N s_{N-i} \mod 2.$$

Alors on a

- *Si $d_N = 0$, $L_{N+1} = L_N$.*

– *Si* $d_N = 1$, $L_{N+1} = \max(L_N, N+1-L_N)$.

On a en plus

– *Si* $d_N = 0$,

$$P_{N+1}(X) = P_N(X).$$

– *Si* $d_N = 1$,

$$P_{N+1}(X) = P_N(X) + P_m(X)X^{N-m}$$

où m *est le plus grand entier inférieur à* N *tel que* $L_m < L_N$.

Pour les preuves du lemme et de cette proposition, on pourra se référer à [Mas69]. On déduit de cette proposition un algorithme permettant de trouver un LFSR de taille minimale générant une suite donnée.

Algorithme.
Entrée : $s_0, s_1, \ldots, s_{n-1}$ une suite de longueur n.
Initialisation

$$P(X) = 1, \quad L = 0, \quad m = -1, \quad g(X) = 1.$$

Pour N **variant de** 0 **à** $n-1$
1. Calculer $d = s_N + \sum_{i=1}^{L} c_i s_{N-i} \ mod\ 2$.
2. Si $d = 1$ alors
   – $t(X) = P(X)$ et $P(X) = P(X) + g(X)X^{N-m}$.
   – Si $2L \leq N$ alors $L = N+1-L$, $m = N$, $g(X) = t(X)$.

TAB. 1.1 – *Algorithme de Berlekamp-Massey.*

Exemple 1.2 *Application de l'algorithme de Berlekamp-Massey à la suite binaire de lon-*

gueur 12, $(s_t)_{t\geq 0}$ = 001011101011.

N	s_N	d	L	$P(X)$	m	$g(X)$
			0	1	-1	1
0	0	0	0	1	-1	1
1	0	0	0	1	-1	1
2	1	1	3	X^3+1	2	1
3	0	0	3	X^3+1	2	1
4	1	1	3	X^3+X^2+1	2	1
5	1	0	3	X^3+X^2+1	2	1
6	1	0	3	X^3+X^2+1	2	1
7	0	0	3	X^3+X^2+1	2	1
8	1	1	6	$X^6+X^3+X^2+1$	8	X^3+X^2+1
9	0	1	6	$X^6+X^4+X^2+X+1$	8	X^3+X^2+1
10	1	0	6	$X^6+X^4+X^2+X+1$	8	X^3+X^2+1
11	1	1	6	$X^5+X^4+X^3+X^2+X+1$	8	X^3+X^2+1

1.2.3 Brasseur synchrone

Le brasseur synchrone réalise l'addition bit à bit, modulo 2, du train binaire représentant le message $(x_t)_{t\geq 0}$ et d'une séquence pseudo-aléatoire $(s_t)_{t\geq 0}$. Les éléments binaires brassés sont donc :

$$y_t = s_t + x_t.$$

Le débrasseur est identique au brasseur, il réalise l'addition bit-à-bit du train numérique démodulé $(\widetilde{y}_t)_{t\geq 0}$ et d'une séquence pseudo-aléatoire $(s_t)_{t\geq 0}$ identique à celle utilisée lors de l'émission.

Les générateurs pseudo-aléatoires du brasseur et du débrasseur doivent donc être identiques et délivrer les mêmes suites, avec la même phase. Il est donc nécessaire de prévoir un dispositif de synchronisation afin de pouvoir les faire fonctionner en phase.

Le débrasseur, lorsqu'il est synchronisé avec le brasseur, réalise l'opération :

$$\widetilde{x}_t = s_t + \widetilde{y}_t.$$

On a donc

$$x_t + \widetilde{x}_t = y_t + \widetilde{y}_t.$$

Une erreur sur un élément binaire de la séquence démodulée se répercute directement sur l'élément correspondant après débrassage. Le taux d'erreur n'est pas modifié par l'opération de brassage/débrassage, à condition que la synchronisation soit parfaite. On a ainsi représenté ci-dessous, l'exemple d'un brasseur/débrasseur synchrone de polynôme de rétroaction X^6+X+1.

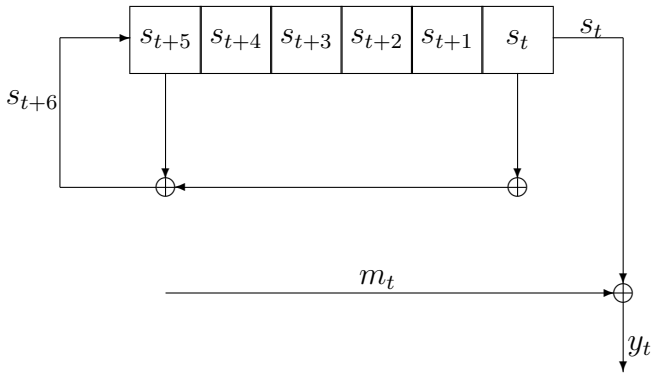

FIG. 1.8 – *Brasseur synchrone de polynôme de rétroaction* $X^6 + X + 1$.

1.2.4 Brasseur autosynchronisant

Le schéma de base d'un brasseur autosynchronisant est dérivé de celui d'un générateur de suites pseudo-aléatoires. Le brasseur est constitué d'un registre à décalage défini par un polynôme de rétroaction et lors de la remise à jour, le bit de sortie est le même que le bit qui entre dans le registre. Il est obtenu en calculant la rétroaction associé au polynôme de rétroaction et en ajoutant à ce résultat le bit d'entrée. On représente ainsi ci-dessous un brasseur/débrasseur autosynchronisant de polynôme de rétroaction $1+X^3+X^4$. L'équation

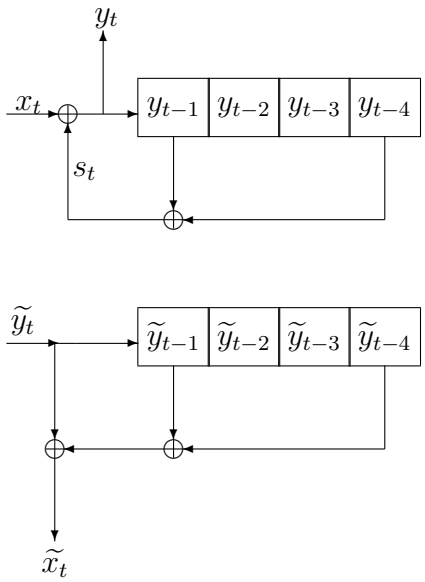

FIG. 1.9 – *Brasseur (en haut) et débrasseur (en bas) autosynchronisant de polynôme de rétroaction* $1 + X^3 + X^4$.

du brasseur est donc :

$$y_t = x_t + \sum_{i=0}^{L-1} c_i y_{t-(l-i)}$$

c'est-à-dire

$$x_t = \sum_{i=0}^{L} c_i y_{t-(L-i)}.$$

À l'inverse, en entrée du débrasseur, on a $\widetilde{y}_t = y_t + e_t$ avec $e_t = 0$ ou 1, alors, on obtient en sortie :

$$\widetilde{x}_t = \sum_{i=0}^{L} c_i y_{t-(L-i)} + \sum_{i=0}^{L} c_i e_{t-(L-i)},$$

où les c_i sont les coefficients du polynôme de rétroaction du registre. En l'absence d'erreur de transmission, le débrassage permet de reconstituer les éléments binaires x_t du message. L'inconvénient majeur de l'opération de brassage/débrassage autosynchronisant est la multiplication des erreurs. En effet, une erreur sur un bit donné va se répercuter sur tous les bits qui font directement ou indirectement intervenir ce bit, c'est-à-dire que le bit x_t sera affecté par toute erreur sur l'un des bits $y_{t-(L-i)}$ avec $0 \leq i \leq L$ et $c_i = 1$; qu'il s'agisse d'une erreur de transmission sur l'un des bits $y_{t-(L-i)}$ ou d'une erreur provenant du débrassage, c'est-à-dire d'une erreur de transmission sur l'un des bits dont dépend $y_{t-(L-i)}$. Ce type de brasseur a par contre le gros avantage de permettre la synchronisation. Au début de la transmission, ou à la suite d'un paquet d'erreurs, il est possible que les registres du brasseur et du débrasseur ne soient pas dans le même état. Le débrasseur n'étant pas bouclé, son registre sera dans le même état que celui du brasseur après la transmission sans erreurs de L bits. La synchronisation sera alors établie.

1.3 Les codes correcteurs d'erreurs

L'autre élément de la chaîne de transmission à retrouver est le code correcteur d'erreur qui a servi lors de la transmission. On va donc tout d'abord rappeler quelques propriétés classiques des codes correcteurs en bloc. La plupart des résultats présentés ici sont des résultats de base qui se trouvent dans de nombreux ouvrages, par exemple dans [MS77] ou [LC83].

1.3.1 Définitions et premières propriétés

On considère l'espace ambiant $\mathbf{F}_2^n$.

Définition 1.15 *Un* code linéaire de longueur n *est un sous-espace vectoriel de* $\mathbf{F}_2^n$.

On définit le support d'un mot de code $c = (c_1, \ldots, c_n)$ par :

$$Supp(c) = \{i \in \{1, \ldots, n\} | c_i \neq 0\}.$$

On peut alors parler du *poids de Hamming* d'un mot c, noté $wt_H(c)$; il est défini comme étant le nombre de positions non nulles de c :

$$wt_H(c) = \sharp Supp(c),$$

où $\sharp E$ désigne le cardinal de l'ensemble E.

On définit alors la distance de Hamming entre deux mots de code c et c', notée $d_H(c,c')$ par :

$$d_H(c,c') = wt_H(c - c').$$

Cette quantité représente le nombre de symboles différents entre c et c'. On montre facilement que la distance de Hamming est bien une distance.

On peut alors parler de la distance minimale d'un code $\mathcal{C}$ définie naturellement de la manière suivante :

$$d_{\min}(\mathcal{C}) = \min_{c,c' \in \mathcal{C}, c \neq c'} d(c,c').$$

On peut noter que, dans le cas d'un code linéaire, on a

$$d_{\min}(\mathcal{C}) = \min_{c \in \mathcal{C}, c \neq 0} wt_H(c).$$

Proposition 1.16 *Considérons un canal binaire symétrique de probabilité d'erreur $p < \frac{1}{2}$. Supposons que la loi d'émission du code $\mathcal{C}$ est uniforme. Connaissant le mot reçu $\widetilde{c}$, le mot de code c le plus probablement émis est le mot de code le plus proche de $\widetilde{c}$ pour la distance de Hamming.*

Théorème 1.17 *Si un code $\mathcal{C}$ a une distance minimale $d_{\min}$, alors il permet de corriger toute erreur de poids inférieur ou égale à $t = \lfloor \frac{d_{\min}-1}{2} \rfloor$. Cet entier t s'appelle la* capacité de correction *du code $\mathcal{C}$.*

1.3.2 Un peu plus sur les codes linéaires

Soit $\mathcal{C}$ un code linéaire sur $\mathbf{F}_2$, *i.e.* un sous-espace vectoriel de $\mathbf{F}_2^n$. Les trois paramètres importants d'un code linéaire sont sa longueur n, sa dimension qu'on notera k (il s'agit de sa dimension en tant que sous-espace vectoriel de $\mathbf{F}_2^n$) et sa distance minimale $d_{\min}$. On parlera alors d'un code $[n,k,d_{\min}]$ (ou simplement $[n,k]$ si la distance minimale est omise). Le rapport $\frac{k}{n}$ s'appelle le rendement du code.

Définition 1.18 *Une* matrice génératrice *du code $\mathcal{C}$ est une matrice de taille $k \times n$ dont les lignes sont k générateurs indépendants du sous-espace vectoriel,* i.e. *dont les lignes forment une base de $\mathcal{C}$.*

Proposition 1.19 *Deux matrices G_1 et G_2 de taille $k \times n$ engendrent le même code si et seulement s'il existe une matrice inversible S de taille $k \times k$ telle que*

$$G_1 = SG_2.$$

Pour un code $\mathcal{C}$, un *encodeur* (ou une fonction de codage) est une application injective $f : \mathbf{F}_2^k \to \mathbf{F}_2^n$ telle que $f(\mathbf{F}_2^k) = \mathcal{C}$. Dans le cas d'un code linéaire $\mathcal{C}$, un encodeur est défini de manière unique par une matrice de taille $k \times n$ qui est une matrice génératrice du code $\mathcal{C}$.

Si

$$G = (I_k | B_{k,n-k}),$$

on dit que G est la matrice génératrice de $\mathcal{C}$ sous forme systématique. En effet, si on encode un message en utilisant une telle matrice génératrice, alors les k premiers symboles du mot de code correspondent exactement à l'information. Notons qu'il existe aussi des encodages systématiques pour certains codes non-linéaires [MS77, Page 302].

On note X^T la matrice transposée d'une matrice X. On munit $\mathbf{F}_2^n$ du produit scalaire usuel.

$$cc'^T = \sum_{i=1}^{n} c_i c'_i \in \mathbf{F}_2.$$

On peut noter que tous les mots de code de poids pair sont isotropes c'est-à-dire qu'ils vérifient $cc^T = 0$.

Le *dual* d'un code $\mathcal{C}$ est l'orthogonal de $\mathcal{C}$ pour le produit scalaire usuel :

$$\mathcal{C}^\perp = \{h \in \mathbf{F}_2^n | \forall c \in \mathcal{C},\ hc^T = 0\}.$$

On a $\dim(\mathcal{C}^\perp) = n - \dim(\mathcal{C})$.

Une *matrice de contrôle* (ou *matrice de parité*) d'un code $\mathcal{C}$ est une matrice génératrice du code dual.

Proposition 1.20 *Soit G une matrice génératrice du code $\mathcal{C}$. Une matrice H de rang $n-k$ et de taille $(n-k) \times n$ est une matrice de contrôle pour $\mathcal{C}$ si et seulement si*

$$GH^T = 0.$$

Corollaire 1.21 *Soit H une matrice de contrôle du code $\mathcal{C}$. Un mot c appartient à $\mathcal{C}$ si et seulement si*

$$Hc^T = 0.$$

Proposition 1.22 *Si un code $\mathcal{C}$ admet une matrice génératrice sous forme systématique*

$$G = (I_k | B_{k,n-k}),$$

alors la matrice

$$H = \left(-B_{k,n-k}^T | I_{n-k}\right)$$

est une matrice de parité de $\mathcal{C}$.

Cette condition s'exprime sous forme de $(n-k)$ équations linéaires indépendantes que doivent vérifier les symboles du mot de code c. Chacune des $(2^{n-k} - 1)$ équations non nulles correspondant aux différentes lignes de la matrice H et à leurs combinaisons linéaires

est appelée *une équation de parité* du code $\mathcal{C}$. Les équations de parité d'un code $\mathcal{C}$ sont naturellement en bijection avec les mots du code dual $\mathcal{C}^\perp$.

Exemple 1.3 *Les codes de Hamming forment une famille classique importante de codes en bloc qui sont faciles à encoder et à décoder. Le code de Hamming binaire $\mathcal{H}_r$ de longueur $n = 2^r - 1$ (avec $r \geq 2$) admet une matrice de parité H dont les colonnes sont tous les vecteurs non nuls de longueur r, chacun étant utilisé une seule fois. Le code $\mathcal{H}_r$ est un code $[n = 2^r - 1,\ k = 2^r - 1 - r,\ d_{\min} = 3]$. On voit que la capacité de correction d'un tel code est égale à 1 ; on ne peut donc corriger qu'une seule erreur.*

Ainsi pour $r = 3$, on a un code [7,4,3] *; la matrice de parité de $\mathcal{H}_3$ est :*

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

Pour mettre H sous forme systématique, il suffit de prendre ses colonnes dans un ordre différent :

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

On peut alors en déduire facilement une matrice génératrice du code $\mathcal{H}_3$:

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

Si on veut encoder le mot $m = 0110$, on calcule alors $c = mG = 0110011$. Supposons que lors de la transmission, il se produise une erreur et que l'on reçoive $\widetilde{c} = 1110011$. Pour décoder le mot reçu $\widetilde{c}$, on calcule son syndrome, c'est-à-dire la quantité

$$H'\widetilde{c}^T = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}.$$

Si on écrit l'erreur sous forme d'un vecteur $e \in \mathbf{F}_2^7$, i.e. *$\widetilde{c} = c + e$, on a en effet*

$$H'\widetilde{c}^T = H'c^T + H'e^T = H'e^T$$

car $Hc^T = 0$. Comme notre erreur est ici de poids 1, on retrouve alors simplement la position de l'erreur puisqu'elle correspond à la position de la colonne de H' égale au syndrome de $\widetilde{c}$; les colonnes de la matrice H' décrivent en effet l'ensemble des mots de $\mathbf{F}_2^3$. On décode donc aisément $\widetilde{c}$ et on trouve $c = 0110011$ ce qui permet d'en déduire facilement le message $m = 0110$, d'autant plus facilement que G est sous forme systématique.

1.3.3 Distribution des poids d'un code

Au-delà de sa distance minimale, paramètre fondamental puisqu'elle détermine la capacité de correction d'un code, l'ensemble de sa distribution des poids est importante. Soit $\mathcal{C}$ un code binaire. Pour i variant de 0 à n, on note A_i le nombre de mots de $\mathcal{C}$ de poids i. On a : $A_0 = 1$, $A_j = 0$ pour $0 < j < d_{\min}$ et $A_{d_{\min}} \neq 0$. La donnée des A_i est appelée la distribution des poids du code $\mathcal{C}$.

Définition 1.23 *Le* polynôme énumérateur des poids *d'un code est le polynôme*

$$W_{\mathcal{C}}(X,Y) = \sum_{i=0}^{n} A_i X^{n-i} Y^i,$$

c'est-à-dire

$$W_{\mathcal{C}}(X,Y) = \sum_{c \in \mathcal{C}} X^{n-wt_H(c)} Y^{wt_H(c)}.$$

Le théorème suivant, appelé théorème de MacWilliams établit le lien entre la distribution des poids d'un code binaire et celle de son dual. On parle aussi souvent de l'identité de MacWilliams.

Théorème 1.24 *Si $\mathcal{C}$ est un code linéaire binaire de paramètres $[n,k]$, alors*

$$W_{\mathcal{C}^\perp}(X,Y) = \frac{1}{2^k} W_{\mathcal{C}}(X+Y, X-Y).$$

Il existe aussi une version q-aire de ce théorème ainsi qu'une version pour les codes non-linéaires. Le lecteur pourra se référer par exemple à [MS77].

Exemple 1.4

1. *Considérons tout d'abord un exemple très simple, celui du code $\mathcal{C} = \{000,011,101,110\}$. L'énumérateur des poids de ce code est :*

$$W_{\mathcal{C}}(X,Y) = X^3 + 3XY^2.$$

On peut alors utiliser l'identité de MacWilliams et calculer

$$W_{\mathcal{C}^\perp}(X,Y) = \frac{1}{2^k} W_{\mathcal{C}}(X+Y, X-Y) = \frac{1}{2^2}(X+Y)^3 + 3(X+Y)(X-Y)^2 = X^3 + Y^3.$$

On peut aussi voir simplement que $\mathcal{C}^\perp = \{000,111\}$ et vérifier que

$$W_{\mathcal{C}^\perp}(X,Y) = X^3 + Y^3.$$

2. *Reprenons maintenant l'exemple du code de Hamming $\mathcal{H}_3$. En regardant attentivement la matrice de parité, on s'aperçoit que la somme de deux de ses lignes produit toujours un mot de poids 4 et que la somme de toutes ses lignes produit également un mot de poids 4. Le code $\mathcal{H}_3^\perp$, engendré par la matrice H contient donc un mot de poids 0 et 7 mots de poids 4. On a par conséquent :*

$$W_{\mathcal{H}_3^\perp}(X,Y) = X^7 + 7X^3Y^4.$$

On peut alors calculer le polynôme énumérateur des poids de $\mathcal{H}_3$. On a

$$\begin{aligned} W_{\mathcal{H}_3}(X,Y) &= \frac{1}{2^3}(X+Y)^7 + 7(X+Y)^3(X-Y)^4 \\ &= \frac{1}{8}(8X^7 + 56X^4Y^3 + 56X^3Y^4 + 8Y^7) \\ &= X^7 + 7X^4Y^3 + 7X^3Y^4 + Y^7. \end{aligned}$$

On peut aussi vérifier, en regardant la matrice génératrice de $\mathcal{H}_3$ qu'elle engendre un mot de poids 0, 7 mots de poids 4, 7 mots de poids 3 et un mot de poids 7. On retrouve donc bien que :

$$W_{\mathcal{H}_3}(X,Y) = X^7 + 7X^4Y^3 + 7X^3Y^4 + Y^7.$$

3. *On peut aussi regarder ce qui se passe dans le cas d'un code de Hamming quelconque $\mathcal{H}_r$. On rappelle que les paramètres de ce code sont : $[n = 2^r - 1,\ k = n - r,\ d_{\min} = 3]$. On va s'intéresser à la distribution des poids de son dual appelé code simplex de longueur $(2^r - 1)$. On sait que les colonnes de la matrice de parité de $\mathcal{H}_r$ sont tous les vecteurs de longueur r. On peut donc montrer que tous les générateurs sont de poids $2^{r-1} = \frac{n+1}{2}$ ainsi que toutes leurs combinaisons linéaires non nulles. On a donc dans le dual le mot de poids 0 et $(2^r - 1)$ mots de poids 2^{r-1}. D'où :*

$$W_{\mathcal{H}_r^\perp}(X,Y) = X^n + nX^{\frac{n-1}{2}}Y^{\frac{n+1}{2}}.$$

La formule de MacWilliams nous permet donc d'en déduire que le polynôme énumérateur des poids de $\mathcal{H}_r$ est :

$$W_{\mathcal{H}_r}(X,Y) = \frac{1}{n+1}\left((X+Y)^n + n(X+Y)^{\frac{n-1}{2}}(X-Y)^{\frac{n+1}{2}}\right).$$

Après ces rappels préliminaires sur les principaux maillons d'une chaîne de transmission, nous allons maintenant nous focaliser sur leur reconstruction. Nous avons vu que le processus de reconstruction dans le modèle que nous venons de décrire comportait essentiellement deux étapes : la reconstruction du code correcteur en bloc (qui inclut en particulier la correction des erreurs de transmission) et celle du brasseur. Ce sont ces deux étapes que nous allons maintenant aborder dans les deux parties suivantes.

Première partie

Reconnaissance d'un code en bloc en présence d'erreurs

Chapitre 2

Recherche d'équations de parité de poids faible probables

On s'intéresse dans cette partie à la reconstruction du code correcteur d'erreur en blocs qui a été utilisé lors de la transmission. On a intercepté un signal et on sait que ce signal provient d'un système de communication classique qu'on peut donc modéliser par un brasseur et un code linéaire en blocs comme montré dans le chapitre introductif, figure 1.5 page 10. Dans cette thèse, on ne s'intéresse pas au problème de modulation/démodulation ; on considère donc que l'on reçoit une suite numérique provenant d'un système de communication. Cependant, les techniques classiques de démodulation permettent souvent de connaître le type de canal utilisé mais aussi d'avoir des renseignements sur le code ; en effet, il a fallu détecter les trames de synchronisation lors de la démodulation et on a donc souvent une bonne idée des paramètres du code utilisé. On supposera donc dans cette partie que la longueur et la dimension du code correcteur sont connues. De plus, on suppose aussi que l'on connaît la synchronisation, *i.e.* que l'on sait où se trouve le début des mots. Avant d'entrer dans les détails techniques, nous allons fixer des notations que nous utiliserons dans toute cette partie.

$\mathcal{C}$	:	le code correcteur utilisé lors de la transmission
n	:	la longueur du code $\mathcal{C}$
k	:	la dimension du code $\mathcal{C}$
$\widetilde{m}$	:	un mot du code $\mathcal{C}$ bruité, *i.e.* après transmission
M	:	le nombre de mots de code bruités utilisés
R	:	la matrice $M \times n$ dont les lignes sont les mots de code bruités
$\mathcal{D}$	:	le code engendré par les lignes de la matrice R^T

TAB. 2.1 – *Notations utilisées.*

2.1 Présentation du problème

Cette partie est donc consacrée au problème de la reconnaissance de code. On supposera ici que le code utilisé est un code en bloc linéaire. On s'intéresse à la suite binaire de sortie du codeur qui a été obtenue après démodulation du signal intercepté comme nous l'avons expliqué au chapitre précédent.

Le problème qu'on se pose ici est donc le suivant : on dispose d'une suite observée en sortie du canal, qui est la concaténation de mots de code bruités ; le problème est de retrouver, à partir de cette suite, quel code linéaire a été utilisé lors de la transmission. On cherche à construire une base de ce code, c'est-à-dire une matrice génératrice. On notera bien ici l'emploi de l'article indéfini « une », en effet, même si on est en mesure de retrouver à partir du signal intercepté une base du code utilisé, on ne saura pas quel codeur (c'est-à-dire quelle matrice génératrice) a été utilisé lors de la transmission. Pour pouvoir déterminer le codeur utilisé, il faudrait, en plus de la sortie, connaître une partie correspondante de l'entrée du codeur. Plus précisément, il faudrait connaître k mots linéairement indépendants pour pouvoir reconstruire l'encodeur utilisé. Dans notre contexte, cela s'avérera impossible puisque pour connaître l'entrée du codeur, il faudrait connaître le message en entrée du système et le brasseur utilisé, alors que la reconstruction du brasseur ne pourra se faire que lorsque le code correcteur aura été reconstruit. Dans ce contexte on ne saura donc jamais quel encodeur a été utilisé lors de la transmission. Le problème auquel on s'intéresse ici est de retrouver le code en tant qu'espace vectoriel et non pas en tant que structure algébrique. On souhaite retrouver une matrice génératrice du code ce qui est un problème totalement différent de celui de retrouver la structure algébrique du code, *i.e.* de déterminer à quelle famille le code utilisé appartient.

Bien qu'il s'agisse d'un problème de télécommunications qui se pose réellement, sa nature a sans doute dissuadé de rendre publiques les études qui lui ont été consacrées. On ne trouvera donc que peu de littérature sur le sujet hormis les travaux d'Antoine Valembois [Val00, Val01].

Dans ce chapitre, nous allons d'abord montrer que le problème que nous traitons se trouve dans la classe problèmes NP-complets. Puis nos présenterons le principe général de l'algorithme que présente A. Valembois dans sa thèse ; cet algorithme repose sur un test statistique que nous analyserons et nous montrerons qu'il n'est pas forcément adapté au problème. Enfin nous présenterons l'algorithme Canteaut Chabaud [CC98] qui nous permet d'obtenir des candidats à tester.

2.2 Complexité théorique

Un résultat important de la thèse d'A. Valembois est que le problème auquel nous nous intéressons est difficile : il est dans la classe des problèmes NP-complets. Pour montrer ce résultat, il faut tout d'abord reformuler plus théoriquement le problème.

La longueur n du code utilisé étant connue, on va découper le train binaire reçu en M mots de longueur n que l'on dispose dans une matrice R à M lignes et n colonnes. On

notera rg(R) le rang de la matrice R. S'il n'y a pas d'erreurs, la matrice R est donc une matrice de rang k où k est la dimension du code utilisé (il s'agit d'un code de longueur n et de dimension k). Mais des erreurs se sont produites pendant la transmission : les M mots reçus ont été bruités et on ne dispose pas de mots de code mais de mots "proches" de mots de code. On va toutefois utiliser la matrice R formée précédemment. On sait que cette matrice correspond à une matrice dont les lignes sont des mots de code auxquels on a ajouté un certain nombre d'erreurs. On peut donc écrire R sous la forme :

$$R = C + E,$$

où C est une matrice formée de mots de code et E est la matrice des erreurs. Le problème de reconnaissance de code peut donc se formuler de la manière suivante : à partir de la matrice reçue R, retrouver la matrice C des mots de code. Plus précisément :

Entrée :	Une matrice R binaire de taille $M \times n$ et un entier k.
Problème RP(R,k) :	Trouver une $M \times n$ matrice binaire E de poids minimum, et vérifiant : $\mathrm{rg}(R+E) \leq k$.

TAB. 2.2 – *Problème de reconstruction.*

Si w est un entier, le problème de décision associé est le suivant :

Entrée :	Une matrice R binaire de taille $M \times n$, un entier k et un entier w.
Problème DRP(R,k,w) **:**	Existe-t-il une matrice binaire E de taille $M \times n$ vérifiant : $\mathrm{rg}(R+E) \leq k$ et $wt(E) \leq w$?

TAB. 2.3 – *Problème de décision associé,* **DRP**(R,k,w).

A. Valembois a montré dans sa thèse que ce problème est NP-complet.

Théorème 2.1 *[Val00, p105] Soit R une matrice binaire de taille $M \times n$, k et w deux entiers. Le problème décisionnel* DRP(R,k,w) *est NP-complet.*

La preuve se trouve dans [Val00], nous allons simplement présenter ici les principaux points de la démonstration. On sait [Var97] que le problème décisionnel suivant est NP-complet :

Entrée :	Une matrice G binaire de taille $k \times n$ et un entier w.
Problème DDMP(G,w) **:**	Existe-t-il un mot non nul de poids inférieur à w qui appartient au sous-espace vectoriel engendré par les lignes de G ?

TAB. 2.4 – *Problème de la distance minimale.*

On peut alors montrer que pour trois entiers k, n et w, si G est une matrice binaire de taille $k \times n$ et de rang k, alors

$$\mathrm{DDMP}(G,w) = \mathrm{DRP}(G,k-1,w),$$

ce qui implique que notre problème se trouve bien dans la classe des problèmes NP-complets.

Mais, ce résultat signifie uniquement qu'il ne sera pas possible d'en résoudre facilement toutes les instances : il n'a aucune conséquence sur la difficulté de résoudre une instance précise et il peut naturellement exister des instances faciles. En particulier, si la longueur du code n'est pas trop importante et/ou si le taux d'erreur est faible, il paraît envisageable qu'on se trouve en présence d'instances faciles. Il est donc indispensable d'estimer les valeurs des différents paramètres (taille du code, taux d'erreur ...) pour lesquelles la reconstruction peut être menée en pratique. Par ailleurs, la NP-complétude du problème n'exclut pas les possibilités d'améliorations algorithmiques.

2.3 Principe de l'algorithme de Valembois

Dans sa thèse, Valembois suggère un algorithme pour reconstruire un code correcteur en bloc. Dans cette section, nous allons présenter clairement cet algorithme et faire l'analyse de ses performances en fonction des différents paramètres ce qui n'était pas fait dans la thèse de Valembois.

On suppose qu'on connaît la longueur et la synchronisation ; on peut donc former la matrice R.

$$R = \begin{pmatrix} \widetilde{m}_1 \\ \widetilde{m}_2 \\ \vdots \\ \widetilde{m}_M \end{pmatrix}$$

où les $\widetilde{m}_i$ sont les mots de longueur n reçus, c'est-à-dire des mots du code $\mathcal{C}$ bruités. L'objectif de l'algorithme que nous allons présenter est de retrouver des mots de $\mathcal{C}^\perp$ à partir de cette matrice R. Ces mots nous permettront alors de reconstruire une base de $\mathcal{C}^\perp$, le dual du code utilisé lors de la transmission.
On va montrer ici que de bons candidats pour les mots de $\mathcal{C}^\perp$ sont les mots correspondant à des mots de poids faible du code $\mathcal{D}$ engendré par R^T.

Si C est le code utilisé, on rappelle que le code dual noté $\mathcal{C}^\perp$ est défini de la manière suivante :

$$\mathcal{C}^\perp = \{h \in \mathbf{F}_2^n | \forall c \in \mathcal{C},\ hc^T = 0\}$$

où hc^T est le produit scalaire entre h et c sur $\mathbf{F}_2^n$.

Donc, s'il n'y avait pas d'erreur, on aurait :

$$\forall h \in \mathcal{C}^\perp,\ hR^T = 0.$$

Dans notre cas, il y a des erreurs. Il va donc y avoir des mots bruités $\widetilde{m}$ qui vont vérifier $h\widetilde{m}^T = 1$ pour h dans le code dual ; cependant cela arrive seulement s'il y a un nombre impair d'erreurs sur les positions du support de h [où $h_i = 1$ (avec la notation $h = (h_1, \ldots, h_n)$)]. Pour des taux d'erreurs relativement faibles, cela arrivera seulement dans un petit nombre de cas et on aura donc hR^T de poids faible. Précisons un peu les choses.

Proposition 2.2 *Si $\widetilde{m}$ est le mot reçu après transmission à travers un canal binaire symétrique de taux d'erreur τ, alors la probabilité que le nombre de positions erronées soit pair vaut :*

$$\frac{1 + (1-2\tau)^n}{2}.$$

Preuve.
Soit p la probabilité cherchée. On a :

$$p = \sum_{i=0}^{\lfloor \frac{n}{2} \rfloor} \binom{2i}{n} \tau^{2i} (1-\tau)^{n-2i}.$$

Or, on sait que

$$(1-2\tau)^n = ((1-\tau)-\tau)^n = \sum_{i=0}^{n} \binom{i}{n} (-1)^i \tau^i (1-\tau)^{n-i}.$$

De même

$$1 = ((1-\tau)+\tau)^n = \sum_{i=0}^{n} \binom{i}{n} \tau^i (1-\tau)^{n-i}.$$

Donc

$$\frac{1 + (1-2\tau)^n}{2} = \sum_{i=0}^{\lfloor \frac{n}{2} \rfloor} \binom{2i}{n} \tau^{2i} (1-\tau)^{n-2i},$$

d'où l'égalité annoncée. ◇

On peut donc estimer la probabilité que le mot bruité $\widetilde{m}$ vérifie l'équation de parité h en fonction du poids de Hamming de cette équation.

Corollaire 2.3 *Soient $h \in \mathcal{C}^\perp$ et $\widetilde{m}$ un mot de code bruité par un canal binaire symétrique de probabilité d'erreur τ. Alors :*

$$\Pr[h\widetilde{m}^T = 0] = \frac{1 + (1-2\tau)^{wt_H(h)}}{2},$$

et

$$\Pr[h\widetilde{m}^T = 1] = \frac{1 - (1-2\tau)^{wt_H(h)}}{2},$$

où $wt_H(h)$ est le poids de Hamming de h, i.e. le nombre de coordonnées non nulles de h.

Dans la suite de cette partie, nous noterons $x = (1-2\tau)^{wt_H(h)}$.

Ainsi pour un mot $h \in \mathcal{C}^\perp$, le poids du mot hR^T sera en moyenne

$$wt_H(hR^T) \simeq \frac{M}{2}(1-x),$$

et si $h \notin \mathcal{C}^\perp$, alors le poids de hR^T sera en moyenne

$$wt_H(hR^T) \simeq \frac{M}{2}.$$

Cela nous donne un moyen pour distinguer les mots du dual des autres et donc un point de départ pour reconstruire une matrice de parité du code cherché. Mieux encore, cela nous donne une méthode pour trouver des mots du dual : les mots h de $\mathcal{C}^\perp$ correspondent à des combinaisons des lignes de R^T de poids faible. Les rechercher est donc équivalent à chercher des vecteurs de poids faible dans $\mathcal{D}$, le code de longueur M engendré par R^T. Nous évoquerons le problème de la dimension de ce code plus tard, au début de la section 2.5.1. Cette recherche peut donc être menée en utilisant un algorithme qui, à partir d'une matrice génératrice, fournit des mots de poids faible dans le code. Ce problème a été beaucoup étudié et on dispose de plusieurs algorithmes efficaces pour résoudre ce problème [Leo88, LB88, Ste89, Can96, CC98].

2.4 Test statistique

Supposons que l'on choisisse l'un de ces algorithmes qui permettent de trouver des mots de poids faible dans un code linéaire et qu'on l'applique au code $\mathcal{D}$. Nous en présenterons un en détail dans la section 2.5.2. L'algorithme nous fournit donc des mots h tels que le poids de Hamming de hR^T est petit. Comme nous l'avons vu dans la section précédente, ce mot h est un bon candidat pour être un mot du code dual. Nous allons donc obtenir des candidats h, reste à les tester pour écarter les mauvais candidats et retenir les mots du dual.

Nous allons donc tout d'abord étudier en détail le test statistique proposé par A. Valembois dans sa thèse. Nous verrons que ce test est simple mais qu'il ne répond pas de manière pertinente au problème, car il ne fournit pas une mesure appropriée. Nous présenterons ensuite un nouveau test beaucoup plus pertinent que nous illustrerons par des simulations.

2.4.1 Le test de Valembois

Le test statistique proposé dans la thèse de Valembois pour décider si un mot h donné appartient ou non à $\mathcal{C}^\perp$ est le suivant : si le rapport

$$\frac{\Pr[R|h \in \mathcal{C}^\perp]}{\Pr[R]}$$

est supérieur à un seuil T fixé, alors on décidera que h appartient à $\mathcal{C}^\perp$. La valeur de ce rapport est donnée par la proposition suivante.

Proposition 2.4 *[Val00, p122] On a :*

$$\frac{\Pr[R|h \in \mathcal{C}^\perp]}{\Pr[R]} = \left(1 + (1-2\tau)^{wt_H(h)}\right)^M \times \left(\frac{1-(1-2\tau)^{wt_H(h)}}{1+(1-2\tau)^{wt_H(h)}}\right)^{wt_H(hR^T)}.$$

Preuve.
Pour un mot du code $\mathcal{C}$ bruité donné $\widetilde{m}_i$, on a :

$$\Pr[h\widetilde{m}_i^T = 0|h \in \mathcal{C}^\perp] = \frac{1+(1-2\tau)^{wt_H(h)}}{2},$$

$$\Pr[h\widetilde{m}_i^T = 1|h \in \mathcal{C}^\perp] = \frac{1-(1-2\tau)^{wt_H(h)}}{2},$$

alors que, bien sûr, en l'absence d'hypothèse sur h,

$$\Pr[h\widetilde{m}_i^T = 0] = \frac{1}{2}.$$

On veut calculer

$$\frac{\Pr[R|h \in \mathcal{C}^\perp]}{\Pr[R]}.$$

On a

$$\frac{\Pr[R|h \in \mathcal{C}^\perp]}{\Pr[R]} = \prod_{i=1}^{M} \frac{\Pr[h\widetilde{m}_i^T|h \in \mathcal{C}^\perp]}{\Pr[h\widetilde{m}_i^T]}.$$

Or, le nombre de $\widetilde{m}_i$ tels que $h\widetilde{m}_i^T = 1$ est par définition de R égal à $wt_H(hR^T)$. On a donc :

$$\frac{\Pr[R|h \in \mathcal{C}^\perp]}{\Pr[R]} = (1-(1-2\tau)^{wt_H(h)})^{wt_H(hR^T)} \times (1+(1-2\tau)^{wt_H(h)})^{M-wt_H(hR^T)},$$

d'où le résultat. ◇

Le test d'hypothèses va donc consister à décider, pour chaque mot h, si les valeurs de $wt_H(h)$ et $wt_H(hR^T)$ correspondent à un rapport $\frac{\Pr[R|h\in\mathcal{C}^\perp]}{\Pr[R]}$ supérieur au seuil ou non. La valeur du seuil T dépendra bien sûr des valeurs des probabilités d'erreurs de première et de seconde espèce (fausse alarme et non détection) souhaitées.

L'algorithme proposé par Valembois dans sa thèse [Val00] est alors le suivant :

- générer des couples (h,hR^T) avec hR^T de poids faible ;
- tester les candidats potentiels h de $\mathcal{C}^\perp$;
- écarter les candidats linéairement dépendants des éléments déjà retenus de façon à former une base de $\mathcal{C}^\perp$.

L'algorithme que nous venons de décrire est celui qu'Antoine Valembois appelle dans sa thèse « algorithme de premier ordre ». Il présente aussi des algorithmes d'ordre supérieur, la grande différence étant qu'un algorithme d'ordre r, au lieu de tester les candidats de $\mathcal{C}^\perp$ un par un, teste directement si un espace vectoriel de dimension r est inclus ou non dans $\mathcal{C}^\perp$. Ce test repose alors sur un résultat similaire à celui de la proposition 2.4 dont la preuve est donnée aux pages 142-144 de [Val00].

Proposition 2.5 *[Val00, p142] Soit $\mathcal{H}_r$ l'événement : l'espace linéaire engendré par $h_1,\ldots,h_r$ est inclus dans $\mathcal{C}^\perp$. Soit H la matrice $n \times r$ donnée par $H = (h_1,\ldots,h_r)$. Alors*

$$\frac{\Pr[R|\mathcal{H}_r]}{\Pr[R]} = \prod_{i=1}^{M} \sum_{B \in \mathbf{F}_2^r} (-1)^{B(m_i H)^T} (1-2\tau)^{wt_H(BH^T)}.$$

Lorsqu'on teste si le sous-espace engendré par r mots de poids faible est inclus dans $\mathcal{C}^\perp$, on calcule le rapport précédent et on le compare à un seuil. Dans chaque cas, algorithme du premier ordre ou algorithmes d'ordre supérieur, on peut résumer les algorithmes de reconstruction par les deux étapes suivantes :

- générer des mots de poids h tels que hR^T soit de poids faible ;
- tester s'ils appartiennent à $\mathcal{C}^\perp$ et, si c'est le cas, éliminer ceux qui appartiennent à l'espace linéaire engendré par les mots retenus pour construire une base de $\mathcal{C}^\perp$.

En fait, ces algorithmes gardent en mémoire un espace vectoriel correspondant à $\mathcal{C}^\perp$. Il est initialisé par l'espace vectoriel de dimension nulle, puis on essaye d'incrémenter sa dimension jusqu'à avoir retrouvé $\mathcal{C}^\perp$.

Exemple 2.1 *Illustrons le principe de base de cet algorithme par un cas très simple afin simplement de montrer comment la recherche de mots de poids faible permet de reconstituer le code utilisé, dans le cas d'un code en bloc de longueur 7 et de dimension 4. Le code $\mathcal{C}$ effectivement utilisé lors de la communication est le code de Hamming* [7,4] *de matrice génératrice*

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

La seule information dont nous disposons sur le code $\mathcal{C}$, hormis sa longueur et sa dimension, est fournie par l'observation du message suivant, composé de 9 mots de code bruités par la transmission à travers un canal binaire symétrique de probabilité d'erreur $\tau = \frac{1}{100}$ (les erreurs de transmission sont en gras) :
1100100 0011001 1010101 1000011 1001100 0010110 0110011 0100101 11**0**0000

On met ces mots dans une matrice R :

$$R^T = \begin{pmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & \mathbf{0} \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}.$$

On cherche maintenant des mots h susceptibles d'être dans le dual. Donc on cherche h tel que hR^T soit de poids faible.

On trouve 3 mots tels que hR^T soit de poids 0 :
1100110 1101001 0001111

et 4 mots tels que hR^T soit de poids 1 :
0111100 1011010 1010101 0110011

Prenons donc les deux premiers mots trouvés et n'importe quel autre mot linéairement indépendant, par exemple :

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

On est obligé de prendre un mot tel que hR^T soit de poids 1 car l'espace vectoriel engendré par les mots h tels que hR^T est de poids nul est de dimension 2 et ne couvre donc pas entièrement $\mathcal{C}^\perp$. On peut vérifier que la matrice H ainsi formée est une matrice de parité du code de Hamming ce qui signifie que l'on a bien retrouvé le code linéaire utilisé.

Dans l'exemple précédent, on avait $wt_H(h) = 4$ pour tout $h \in \mathcal{C}^\perp$ puisque le dual du code de Hamming [7,4], le code simplex, a un mot de poids 0 et uniquement des mots de poids 4. Nous allons donc maintenant prendre un autre exemple où cette fois-ci la distribution des poids du dual est mieux répartie. Nous allons tout d'abord montrer, à travers cet exemple, que le test statistique effectué par Valembois, que nous venons de décrire, n'est pas le test pertinent pour détecter les mots du dual.

Exemple 2.2 *On considère que le code utilisé lors de la transmission est le code BCH* [15,7,5] *de matrice génératrice :*

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

Supposons qu'on ait 17 mots de code bruité par une transmission sur un canal binaire symétrique de probabilité d'erreur $\tau = 0.01$ *correspondant à la matrice* R^T *suivante :*

$$R^T = \begin{pmatrix}
1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\
0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\
1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\
1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \\
1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\
1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\
1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1
\end{pmatrix}.$$

La figure 2.1 représente les valeurs du rapport $\frac{\Pr[R|h\in\mathcal{C}^\perp]}{\Pr[R]}$ *en fonction de* $wt_H(h)$ *et* $wt_H(hR^T)$*, conformément à la formule obtenue à la proposition 2.5*[1]*. On voit alors que la valeur de ce rapport ne semble pas corrélée au fait que* h *appartienne ou non à* $\mathcal{C}^\perp$*. En effet, pour des valeurs de* $wt_H(hR^T)$ *égales à 1 ou 2, les valeurs du rapport testé sont maximales (pour* $wt_H(hR^T)$ *fixé) lorsque le poids de* h *est proche de* $\frac{n}{2}$*.*

1. Nous avons exclu de cette figure les mots h tels que $wt_H(hR^T) = 0$ pour des raisons de lisibilité.

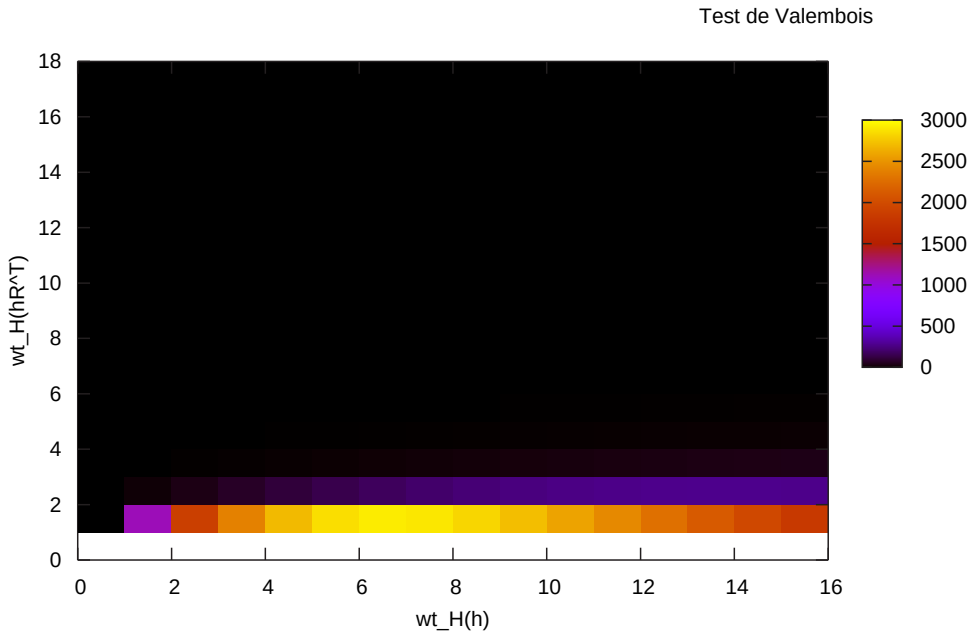

FIG. 2.1 – *Valeur du rapport pour le test statistique proposé par Valembois.*

Pourtant, si l'on calcule le rapport théorique $\alpha = \frac{\sharp\{h \notin \mathcal{C}^\perp \text{ tels que } wt_H(hR^T) \leq 3\}}{\sharp\{h \in \mathcal{C}^\perp \text{ tels que } wt_H(hR^T) \leq 3\}}$ *en fonction de* $wt_H(h)$*, on s'aperçoit au contraire que parmi les mots* h *vérifiant* $wt_H(hR^T) = 2$ *ce ne sont pas ceux dont le poids est proche de* $\frac{n}{2}$ *qui sont les plus susceptibles d'appartenir à* $\mathcal{C}^\perp$*. En effet, il faut accorder plus de crédit aux mots tels que* $wt_H(h)$ *est faible comme le montre la figure 2.2.*

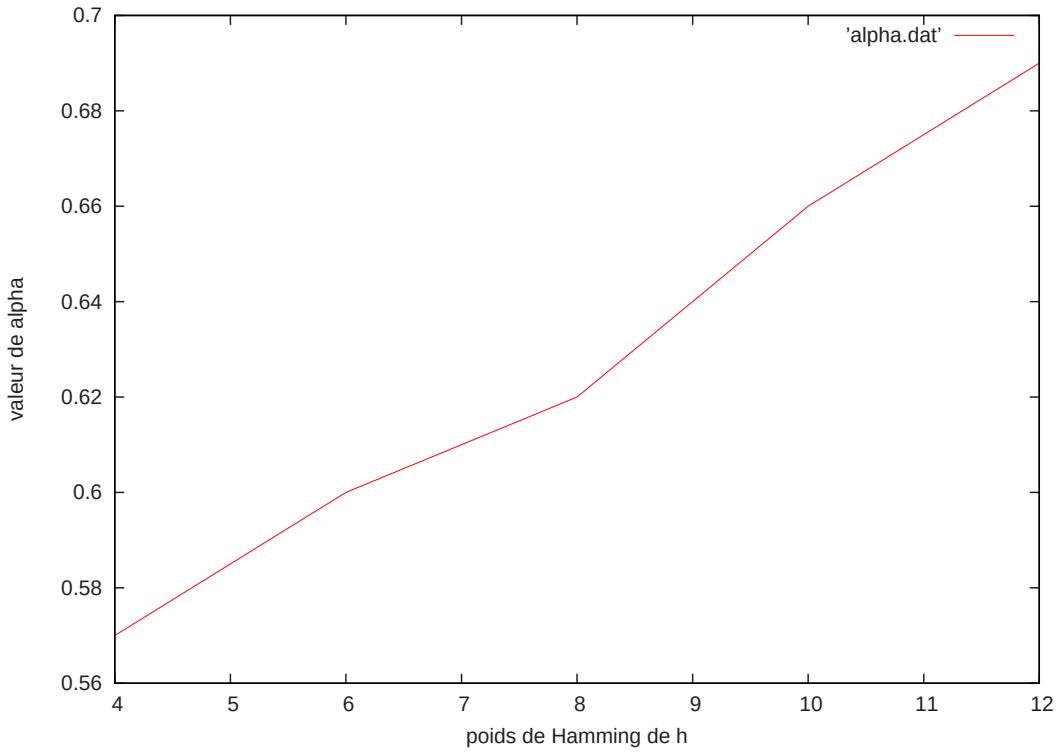

FIG. 2.2 – *Évolution du rapport α en fonction du poids de Hamming de h pour les h tels que $wt_H(hR^T) \leq 3$.*

Le test proposé par Valembois ne semble donc pas être parfaitement en cohérence avec le problème que l'on souhaite traiter. Par conséquent, nous allons maintenant tenter de présenter un test mieux adapté à cette situation.

2.4.2 Un nouveau test statistique

On va s'intéresser tout d'abord au nombre de mots, noté M, que l'on va prendre. On veut pouvoir distinguer un mot du dual d'un mot aléatoire. Si $h \in \mathcal{C}^\perp$, on a :

$$\Pr[h\widetilde{m}^T = 1] = \frac{1-(1-2\tau)^{wt_H(h)}}{2} = \frac{1}{2} - \frac{x}{2},$$

alors que si h est aléatoire, on a

$$\Pr[h\widetilde{m}^T = 1] = \frac{1}{2}.$$

On veut donc avoir un nombre M de mots $\widetilde{m}$ suffisant pour pouvoir distinguer deux distributions binomiales de probabilité $\frac{1}{2}$ et $\frac{1}{2} - \frac{x}{2}$. Ensuite, pour le test statistique, on utilisera un seuil de décision T pour faire le choix entre les deux hypothèses ($h \in \mathcal{C}^\perp$ ou $h \notin \mathcal{C}^\perp$). Le

nombre minimum de mots M suffisant pour pouvoir effectuer ce test dépend du nombre d'erreurs de décision que l'on s'autorise.

En fait dans notre cas, on veut que la probabilité de non détection soit assez faible pour pouvoir trouver suffisamment de mots du dual et le reconstituer entièrement. Par contre, la probabilité de fausse-alarme n'est pas le critère pertinent ici, on va plutôt s'intéresser au rapport

$$\frac{\sharp\left\{h \notin \mathcal{C}^{\perp} \text{ tels que } wt_H(hR^T) \leq T\right\}}{\sharp\left\{h \in \mathcal{C}^{\perp} \text{ tels que } wt_H(hR^T) \leq T\right\}}.$$

En effet, nous sommes dans une situation où le nombre de mots de $\mathcal{C}^{\perp}$ est très petit devant le nombre de mots qui ne lui appartiennent pas.

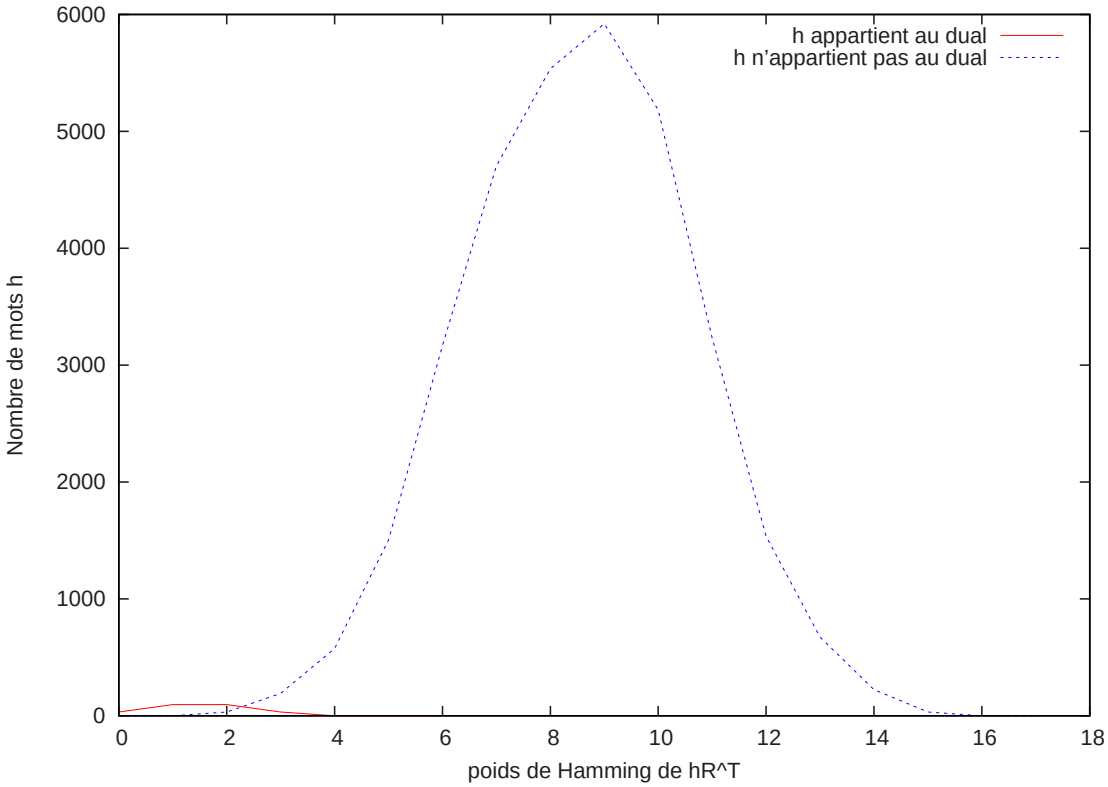

FIG. 2.3 – *Valeurs de* $wt_H(hR^T)$ *quand* $h \in \mathcal{C}^{\perp}$ *et quand* $h \notin \mathcal{C}^{\perp}$ *pour* $M = 17$, $\tau = 0.01$ *quand* $\mathcal{C}$ *est un BCH[15,7,5] correspondant à l'exemple 2.2.*

On voit clairement sur la figure 2.3 que les mots h pour lesquels $wt_H(hR^T) = 3$ se répartissent en parts à peu près égales entre $\mathcal{C}^{\perp}$ et son complémentaire. La valeur du seuil T à prendre pour le test statistique sera donc inférieure à 3. Pourtant, la probabilité :

$$\Pr[wt_H(hR^T) \leq 3 | h \notin \mathcal{C}^{\perp}] = 0.006$$

est très faible, et ne constitue évidemment pas une mesure pertinente pour le test statistique qui nous intéresse ici.

Nous allons donc chercher les valeurs de M et T telles que :

$$\frac{\sharp\left\{h \notin \mathcal{C}^{\perp} \text{ tels que } wt_H(hR^T) \leq T\right\}}{\sharp\left\{h \in \mathcal{C}^{\perp} \text{ tels que } wt_H(hR^T) \leq T\right\}} = \alpha, \tag{2.1}$$

$$\Pr\left[wt_H(hR^T) \geq T | h \in \mathcal{C}^{\perp}\right] = \beta \tag{2.2}$$

pour des valeurs de α et β données.

Proposition 2.6 *Soit $h \in \mathcal{C}^{\perp}$ et soit $x = (1-2\tau)^{wt_H(h)}$ où τ est la probabilité d'erreur du canal binaire symétrique considéré. Soit R la matrice formée par M mots bruités du code $\mathcal{C}$.*
Notons $\mu_0 = \frac{M}{2}(1-x)$ et $\sigma_0^2 = \frac{M}{4}(1-x^2)$. Alors, pour M tendant vers l'infini

$$\frac{wt_H(hR^T) - \mu_0}{\sigma_0}$$

tend vers une loi normale centrée réduite.

Preuve.
Notons z_i la variable aléatoire $z_i = h\widetilde{m_i}^T$ où les $\widetilde{m_i}$ sont les mots de code bruités, *i.e.* les lignes de R. Les z_i sont des variables aléatoires indépendantes et identiquement distribuées. D'après le théorème central limite, pour des grandes valeurs de M, $\frac{\sum_{i=1}^{M}(h\widetilde{m_i}^T) - \mu_0}{\sigma_0}$ tend vers une loi normale centrée réduite avec $\mu_0 = M\Pr[z_i = 1]$ et $\sigma_0^2 = M\Pr[z_i = 1]\Pr[z_i = 0]$. On a donc

$$\mu_0 = \frac{M}{2}\left(1 - (1-2\tau)^{wt_H(h)}\right),$$

$$\sigma_0^2 = \frac{M}{4}\left(1 - (1-2\tau)^{2wt_H(h)}\right).$$

◇

Soit ϕ la fonction de répartition associée à la densité de la loi normale :

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

Théorème 2.7 *Le test statistique consistant à décider que $h \in \mathcal{C}^{\perp}$ si et seulement si $wt_H(hR^T) \leq T$ à partir d'une matrice R formée par M mots bruités où*

$$M = \left(\frac{b\sqrt{1-x^2} + a}{x}\right)^2,$$

$$T = \frac{1}{2}\left(M - a\sqrt{M}\right)$$

avec $a = -\phi^{-1}\left(\alpha\frac{2^{n-k}}{2^n-2^{n-k}}(1-\beta)\right)$, $b = \phi^{-1}(1-\beta)$, *vérifie :*

$$\frac{\sharp\left\{h \notin \mathcal{C}^{\perp} \text{ tels que } wt_H(hR^T) \leq T\right\}}{\sharp\left\{h \in \mathcal{C}^{\perp} \text{ tels que } wt_H(hR^T) \leq T\right\}} = \alpha,$$

$$\Pr\left[wt_H(hR^T) \geq T | h \in \mathcal{C}^{\perp}\right] = \beta.$$

Preuve.
D'après la proposition précédente, on sait que si $h \in \mathcal{C}^{\perp}$, alors pour des grandes valeurs de M, on peut supposer que $\frac{wt_H(hR^T)-\mu_0}{\sigma_0}$ suit une loi normale centrée réduite. Si $h \notin \mathcal{C}^{\perp}$, alors, naturellement, $\frac{wt_H(hR^T)-\frac{M}{2}}{\sqrt{\frac{M}{4}}}$ suit une loi normale centrée réduite. Notons (μ_0,σ_0^2) et (μ_1,σ_1^2) les moyennes et variances correspondant aux deux hypothèses, $h \in \mathcal{C}^{\perp}$ et $h \notin \mathcal{C}^{\perp}$, *i.e.*

$$\begin{aligned}
\mu_0 &= \frac{M}{2}\left(1-(1-2\tau)^{wt_H(h)}\right), \\
\sigma_0^2 &= \frac{M}{4}\left(1-(1-2\tau)^{2wt_H(h)}\right), \\
\mu_1 &= \frac{M}{2}, \\
\sigma_1^2 &= \frac{M}{4}.
\end{aligned}$$

L'équation (2.1) correspond à :

$$\frac{\Pr[wt_H(hR^T) \leq T | h \notin \mathcal{C}^{\perp}]\left(\sharp\{h \notin \mathcal{C}^{\perp}\}\right)}{\Pr[wt_H(hR^T) \leq T | h \in \mathcal{C}^{\perp}]\left(\sharp\{h \in \mathcal{C}^{\perp}\}\right)} = \alpha,$$

ce qui revient à

$$\frac{\Pr[wt_H(hR^T) \leq T | wt_H(hR^T) \text{ suit une loi } \mathcal{N}(\mu_1,\sigma_1)]\left(\sharp\{h \notin \mathcal{C}^{\perp}\}\right)}{\Pr[wt_H(hR^T) \leq T | wt_H(hR^T) \text{ suit une loi } \mathcal{N}(\mu_0,\sigma_0)]\left(\sharp\{h \in \mathcal{C}^{\perp}\}\right)} = \alpha$$

On va noter N_1 la quantité $\sharp\{h \notin \mathcal{C}^{\perp}\} = (2^n - 2^{n-k})$ et $N_0 = \sharp\{h \in \mathcal{C}^{\perp}\} = 2^{n-k}$. On a donc :

$$\frac{\phi\left(\frac{T-\mu_1}{\sigma_1}\right)}{\phi\left(\frac{T-\mu_0}{\sigma_0}\right)} = \alpha\frac{N_0}{N_1}. \tag{2.3}$$

De l'équation (2.2), il s'ensuit :

$$\phi\left(\frac{T-\mu_0}{\sigma_0}\right) = 1-\beta. \tag{2.4}$$

On cherche donc T et M tels que :

$$\begin{aligned} \phi\left(\frac{T-\mu_0}{\sigma_0}\right) &= 1-\beta, \\ \text{et } \phi\left(\frac{T-\mu_1}{\sigma_1}\right) &= \alpha\frac{N_0}{N_1}(1-\beta). \end{aligned}$$

Notons $a = -\phi^{-1}\left(\alpha\frac{N_0}{N_1}(1-\beta)\right)$ et $b = \phi^{-1}(1-\beta)$.
On doit donc résoudre le système :

$$\begin{cases} -a &= \dfrac{T-\mu_1}{\sigma_1} \\ b &= \dfrac{T-\mu_0}{\sigma_0} \end{cases}$$

Cela nous conduit à :

$$T = \frac{1}{2}\left(M - a\sqrt{M}\right),$$

$$\text{avec } M = \left(\frac{a+b\sqrt{1-x^2}}{x}\right)^2.$$

◇

Ce théorème (et sa preuve) nous donne donc, pour une valeur de $wt_H(h)$ donnée, une borne inférieure sur le nombre d'échantillons M nécessaires au test statistique et une valeur maximale T pour $wt_H(hR^T)$ qui nous permet de distinguer les mots h du dual $\mathcal{C}^\perp$ de ceux qui n'y sont pas.

La question que nous nous sommes alors posée est de savoir comment évoluait la valeur de α en fonction du poids de Hamming de h : si on choisit une valeur pour $wt_H(h)$ et les valeurs de M et T associées, que se passe-t-il pour un $wt_H(h)$ différent ? En particulier si le poids de Hamming de h est plus faible est-ce que le test va encore fonctionner ?

Exemple 2.3 *Reprenons l'exemple précédent où le code utilisé lors de la transmission est un code BCH* [15,7,5]. *L'information dont nous disposons sur le code $\mathcal{C}$ est fournie par l'observation d'un message composé de mots de code bruités par la transmission à travers un canal binaire symétrique de probabilité d'erreur $\tau = \frac{1}{100}$. On choisit $\alpha = 0.01$ et $\beta = 0.1$, pour $wt_H(h) = 12$. Cela conduit d'après le théorème 2.7 à choisir :*

$$M = 32 \text{ et } T = 5.$$

Sur le schéma de la figure 2.4, on regarde la répartition des éléments h de $\mathbf{F}_2^{15}$ en fonction des valeurs des couples $(wt_H(h), wt_H(hR^T))$. Sur cette figure comme sur les suivantes, on se focalise sur les petites valeurs de $wt_H(hR^T)$ afin de mettre en valeur la cision correspondant à la « frontière » entre les deux distributions.

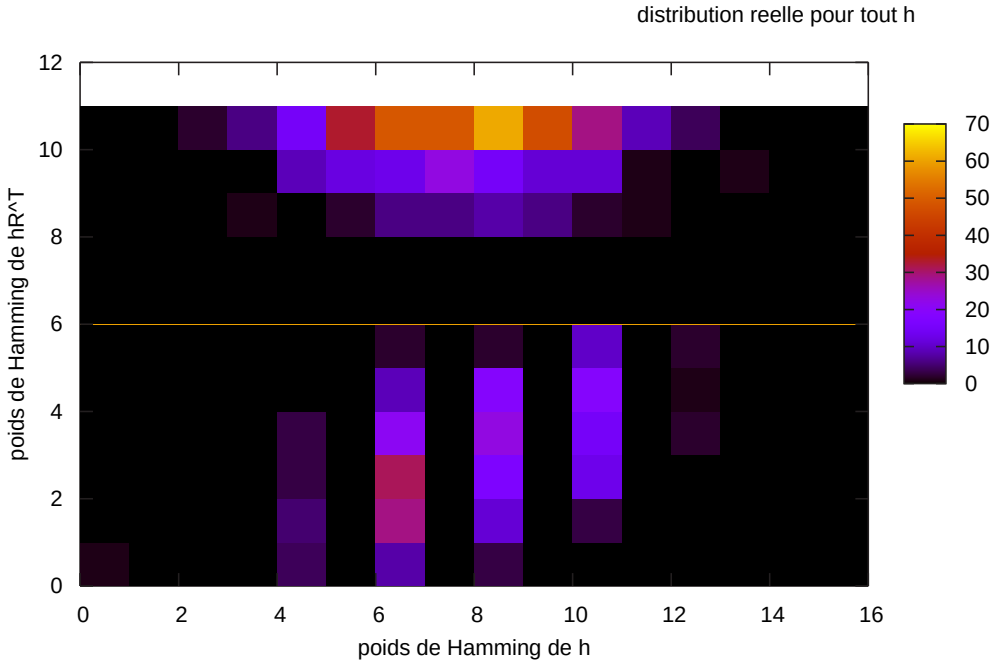

FIG. 2.4 – *Répartition des mots* $h \in \mathbf{F}_2^{15}$ *suivant* $wt_H(h)$ *et* $wt_H(hR^T)$ *pour* $M = 32$, $\tau = 0.01$ *quand* $\mathcal{C}$ *est un BCH [15,7,5].*

On voit que ce test permet bien de distinguer parfaitement les mots du code dual des autres puisque, pour tous les $h \in \mathcal{C}^\perp$, *on a* $wt_H(hR^T) \leq 5$ *alors que pour* $h \notin \mathcal{C}^\perp$, *on a* $wt_H(hR^T) \geq 8$ *et ceci quel que soit le poids de* h.

Nous allons maintenant regarder ce qu'il se passe lorsque l'on prend les mêmes paramètres mais que l'on modifie la valeur de α *pour choisir* $\alpha = 0.7$. *Cela nous amène à choisir* $M = 17$ *et* $T = 3$. *Les résultats des simulations sont présentés sur la figure 2.5 et les deux tables qui suivent.*

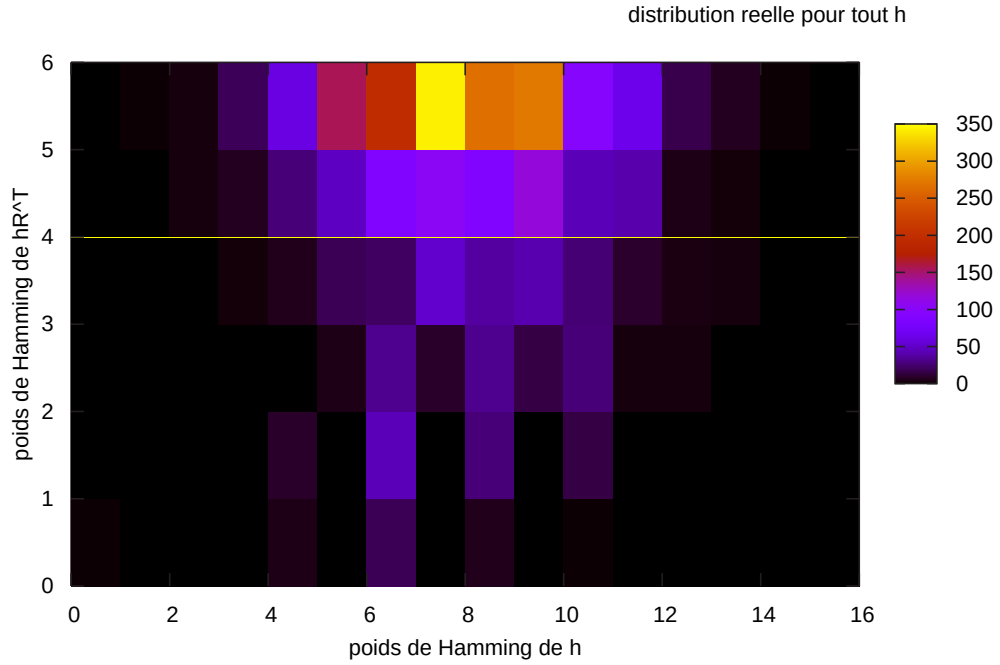

FIG. 2.5 – *Répartition des mots* $h \in \mathbf{F}_2^{15}$ *suivant* $wt_H(h)$ *et* $wt_H(hR^T)$ *pour* $M = 17$, $\tau = 0.01$ *quand* $\mathcal{C}$ *est un BCH* [15,7,5].

$wt_H(hR^T)$	$wt_H(h)$																
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	total
0	1	0	0	0	5	0	19	0	6	0	1	0	0	0	0	0	32
1	0	0	0	0	9	0	45	0	27	0	15	0	0	0	0	0	96
2	0	0	0	0	0	0	33	0	33	0	27	0	3	0	0	0	96
3	0	0	0	0	1	0	3	0	9	0	17	0	2	0	0	0	32
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TAB. 2.5 – *Distribution des couples* ($wt_H(h)$,$wt_H(hR^T)$) *quand* $h \in \mathcal{C}^\perp$ *pour* $M = 17$, $\tau = 0.01$, $\mathcal{C}$ *étant un code BCH [15,7,5].*

$wt_H(hR^T)$	$wt_H(h)$																
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	total
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	5	0	9	0	15	0	3	0	0	0	0	32
3	0	0	0	2	5	19	19	52	29	42	9	10	2	3	0	0	192
4	0	0	3	7	27	48	88	104	88	118	45	41	5	2	0	0	576
5	0	1	3	20	61	156	195	343	266	272	97	65	17	7	1	0	1504
6	0	0	9	66	109	363	421	704	569	498	254	118	46	11	0	0	3168
7	0	4	17	70	198	424	694	948	934	684	455	198	68	8	2	0	4704
8	0	1	17	62	231	426	890	1014	1175	815	566	220	94	22	3	0	5536
9	0	4	16	75	255	526	906	1102	1213	880	559	271	88	22	3	0	5920
10	0	2	19	75	228	488	825	1034	977	829	422	214	55	13	2	1	5184
11	0	0	14	48	148	318	515	650	620	458	300	118	32	8	3	0	3232
12	0	2	5	23	58	149	236	274	326	204	175	47	32	5	0	0	1536
13	0	1	0	5	23	56	93	131	142	122	50	36	11	1	1	0	672
14	0	0	2	2	7	22	23	56	21	56	11	22	0	2	0	0	224
15	0	0	0	0	0	3	0	14	0	12	0	2	0	1	0	0	32
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TAB. 2.6 – *Distribution des couples* $(wt_H(h), wt_H(hR^T))$ *quand* $h \notin \mathcal{C}^\perp$ *pour* $M = 17$, $\tau = 0.01$, $\mathcal{C}$ *étant un code BCH [15,7,5].*

Nous voyons dans ces tables et sur la figure que la distinction entre les mots du dual et les mots n'appartenant pas au dual est beaucoup moins claire surtout lorsque $wt_H(hR^T) = 2$ *ou* 3. *Nous avons regardé comment variait la valeur du rapport* α *défini par*

$$\alpha = \frac{\sharp\left\{h \notin \mathcal{C}^\perp \text{ tels que } wt_H(hR^T) \leq T\right\}}{\sharp\left\{h \in \mathcal{C}^\perp \text{ tels que } wt_H(hR^T) \leq T\right\}}$$

suivant le poids de h. *Les résultats sont récapitulés dans le tableau suivant :*

$wt_H(h)$	Valeur théorique de α	Valeur observée de α
4	0.57	0.33
6	0.60	0.19
8	0.62	0.38
10	0.66	0.15
12	0.69	0.4

TAB. 2.7 – *Influence de* $wt_H(h)$ *sur la valeur de* α.

On remarque qu'il existe une grande différence entre la valeur théorique et la valeur observée de α. *Une des explications est la répartition des mots du code dual ; en effet le*

code dual n'a que des mots de poids pair et cela joue forcément un rôle important puisque pour toutes les valeurs impaires de $wt_H(h)$*, on a* $\alpha = \infty$*. L'autre remarque importante est que la valeur théorique de* α *augmente avec le poids de Hamming de* h *et que donc, si notre test permet de distinguer les mots du dual pour tout mot* h *de poids de Hamming fixé, il sera aussi efficace pour tous les mots de poids de Hamming inférieur à cette valeur.*

Nous avons ensuite augmenté la valeur de τ *pour voir ce qu'il se passait. On a choisi une probabilité d'erreur sur le canal de* $\tau = 0.05$*. Pour* $\alpha = 0.01$*,* $\beta = 0.1$ *et* $wt_H(h) = 12$*, on a* $M = 298$ *et* $T = 117$*. Les simulations rapportées à la figure 2.6 montrent qu'il y a un mot de* $\mathcal{C}^\perp$ *qu'on ne pourra pas détecter puisque pour ce mot, on a* $wt_H(hR^T) = 131 > T$*. Et, on aura 14 mots tels que* $wt_H(hR^T) < T$ *avec* $h \notin \mathcal{C}^\perp$*. On aura donc en réalité* $\alpha \simeq 5\%$ *et* $\beta = 0.4\%$*.*

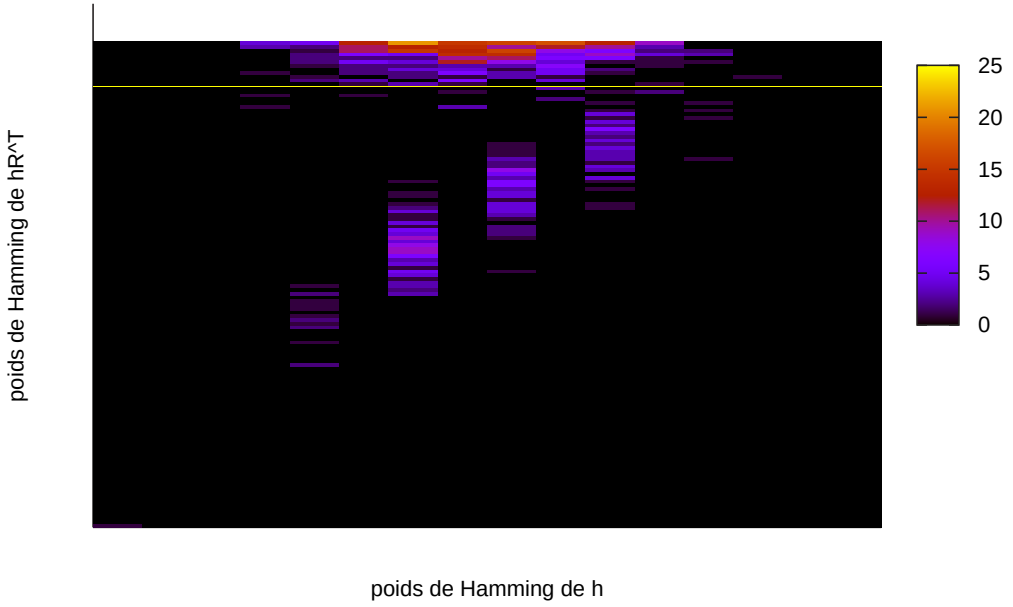

FIG. 2.6 – *Distribution des couples* $(wt_H(h), wt_H(hR^T))$ *pour* $h \in \mathbf{F}_2^{15}$ *pour* $M = 298$*,* $\tau = 0.05$*.*

On a ensuite testé ce qu'il se passait pour une valeur de α *plus grande :* $\alpha = 1.14$*. Pour cette valeur, on a* $M = 170$ *et* $T = 69$*. Les résultats montrent qu'il y a toujours un seul mot que l'on ne détecte pas ; par contre on a maintenant 223 mots tels que* $wt_H(hR^T) < T$ *alors que* h *n'appartient pas au code dual. Le* α *réel vaut* $\alpha = 0.87$*. On a représenté les répartitions des poids sur les graphiques des figures 2.7, 2.8 et 2.9 :*

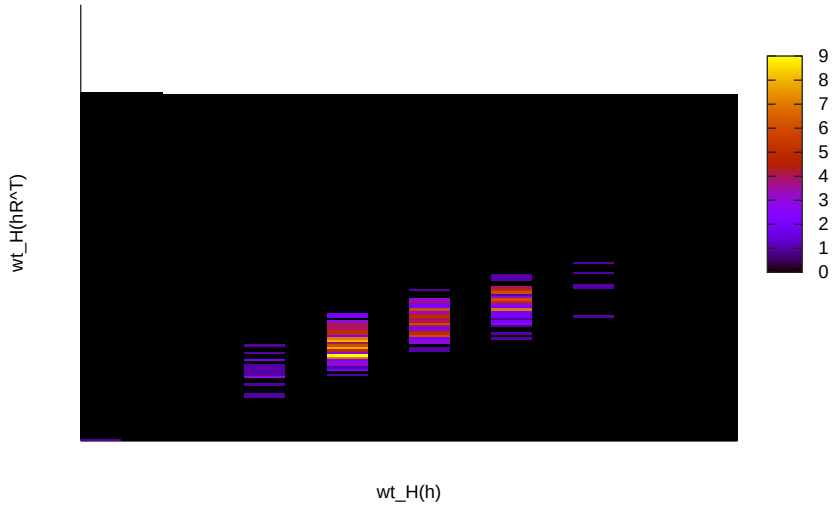

FIG. 2.7 – *Distribution des couples* $(wt_H(h), wt_H(hR^T))$ *quand* $h \in \mathcal{C}^\perp$ *pour* $M = 170$, $\tau = 0.05$.

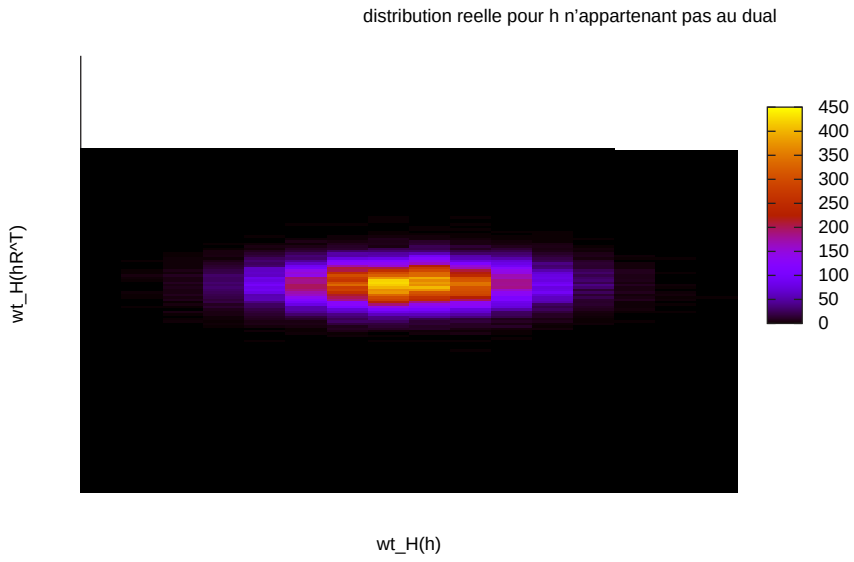

FIG. 2.8 – *Distribution des couples* $(wt_H(h), wt_H(hR^T))$ *quand* $h \notin \mathcal{C}^\perp$ *pour* $M = 170$, $\tau = 0.05$.

FIG. 2.9 – *Distribution des couples* $(wt_H(h),wt_H(hR^T))$ *pour* $h \in \mathbf{F}_2^{15}$ *pour* $M = 170$, $\tau = 0.05$.

Au delà de cet exemple, nous avons voulu regarder plus généralement comment évoluaient les valeurs de M et du seuil T en fonction des paramètres du code correcteur utilisé lors de la transmission. Nous avons fixé $wt_H(h) = \frac{n}{4}$, $\beta = 0.1$ et $\alpha = 0.01$ pour toutes ces simulations.

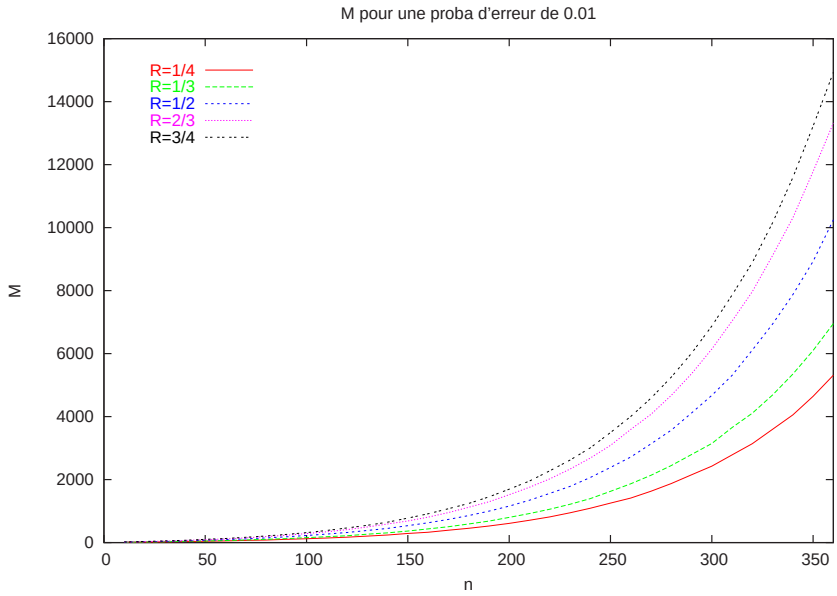

FIG. 2.10 – *Valeurs de M en fonction de n pour des codes $\mathcal{C}$ de différents rendements pour un taux d'erreur $\tau = 0.01$.*

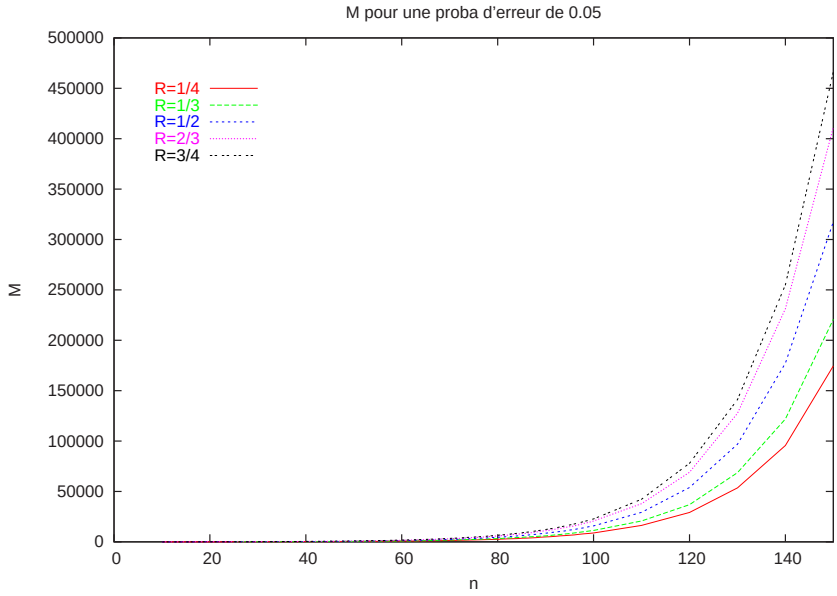

FIG. 2.11 – *Valeurs de M en fonction de n pour des codes $\mathcal{C}$ de différents rendements pour un taux d'erreur $\tau = 0.05$.*

On voit donc que le nombre de mots M à considérer semble augmenter exponentiellement avec la longueur du code. Pour y voir plus clair, nous avons donc tracé M en fonction de la longueur n du code $\mathcal{C}$ en utilisant une échelle logarithmique.

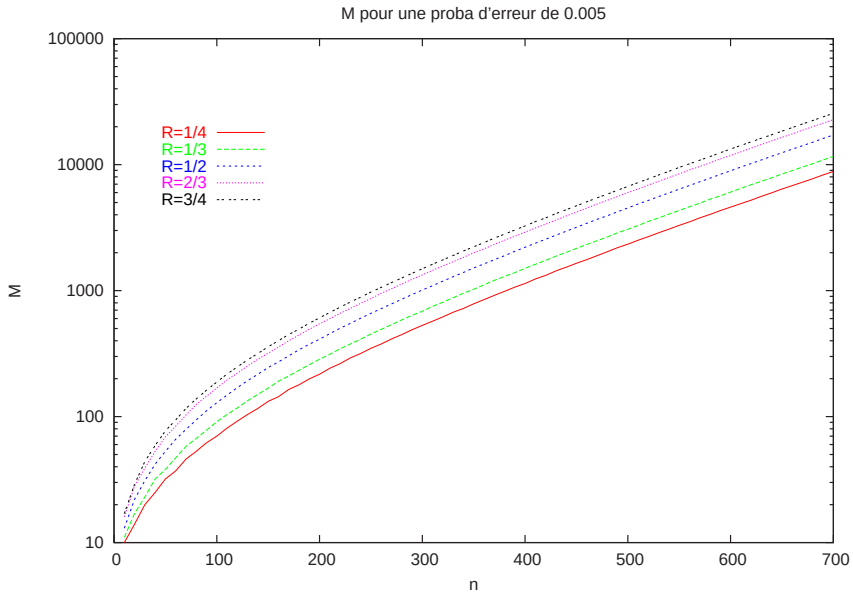

FIG. 2.12 – *Valeurs de M en fonction de n pour des codes $\mathcal{C}$ de différents rendements pour un taux d'erreur $\tau = 0.005$.*

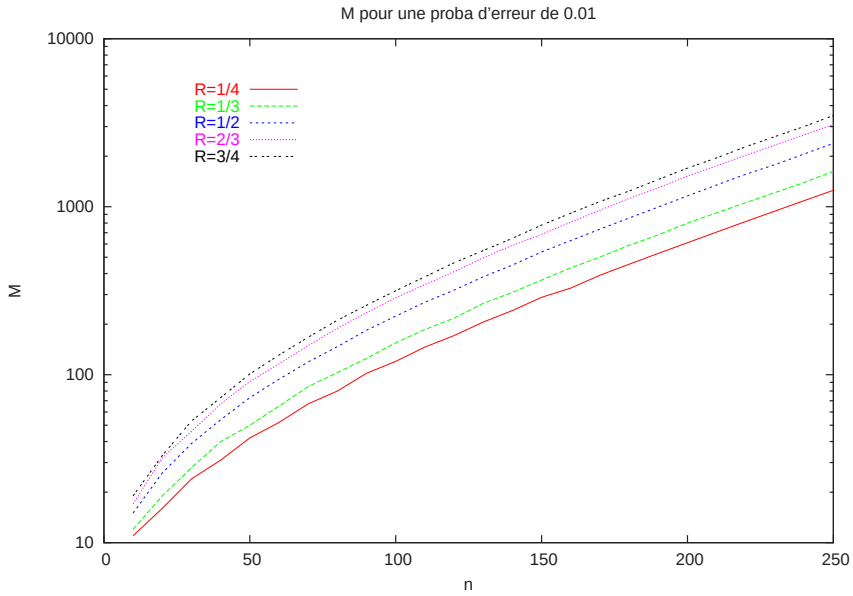

FIG. 2.13 – *Valeurs de M en fonction de n pour des codes $\mathcal{C}$ de différents rendements pour un taux d'erreur $\tau = 0.01$.*

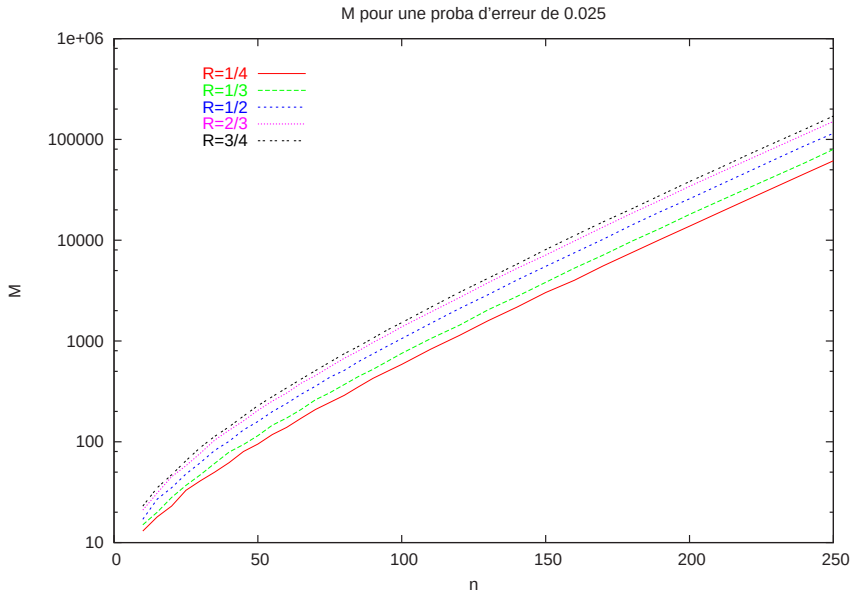

FIG. 2.14 – *Valeurs de M en fonction de n pour des codes $\mathcal{C}$ de différents rendements pour un taux d'erreur $\tau = 0.025$.*

On s'aperçoit ici que, dès que τ devient élevé, il sera très difficile de trouver des mots dans le code $\mathcal{D}$ puisque la longueur M de ce code va très vite devenir trop élevée pour espérer réussir à trouver des mots de poids faible dans ce code en utilisant un algorithme classique. Nous constatons aussi que la valeur de M augmente légèrement avec le rendement du code. On a ensuite représenté l'évolution de M en fonction de n et du taux d'erreur τ, pour certains rendements fixés du code $\mathcal{C}$. Si on suppose par exemple que l'algorithme de recherche de mots de poids faible plus efficace dès que $M \geq 2500$ (ce qui est à peu près le cas en pratique car la complexité devient trop élevée), on voit que pour une probabilité d'erreur de $\tau = 0.025$, on pourra simplement reconstruire un code de longueur inférieure à $n = 120$. Si la probabilité d'erreur est maintenant seulement de $\tau = 0.005$, on pourra alors reconstruire un code de longueur inférieure à $n = 400$. Notons tout de même que ces courbes ont été tracées pour une valeur $\alpha = 0.01$, cela signifie que ces valeurs sont les limites pour n d'un algorithme de reconstruction qui n'accepte quasiment aucune équation de parité fausse. Par contre, si on se permet d'augmenter la valeur de α, alors la valeur de M diminue et on peut alors tenter de reconstruire des codes de longueur plus importante grâce à l'algorithme de décodage que nous présentons au chapitre suivant.

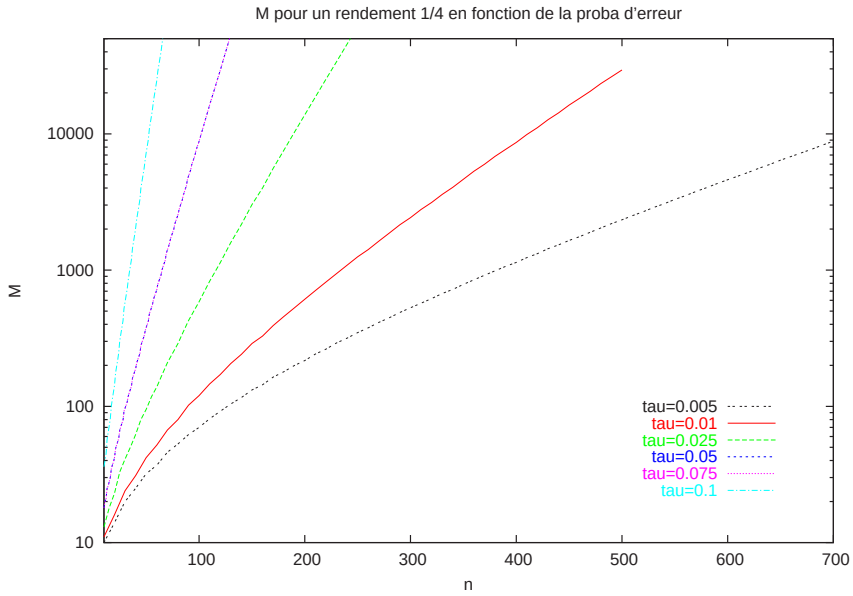

FIG. 2.15 – *Valeurs de M en fonction de n pour différentes valeurs de τ avec un code $\mathcal{C}$ de rendement $\frac{1}{4}$.*

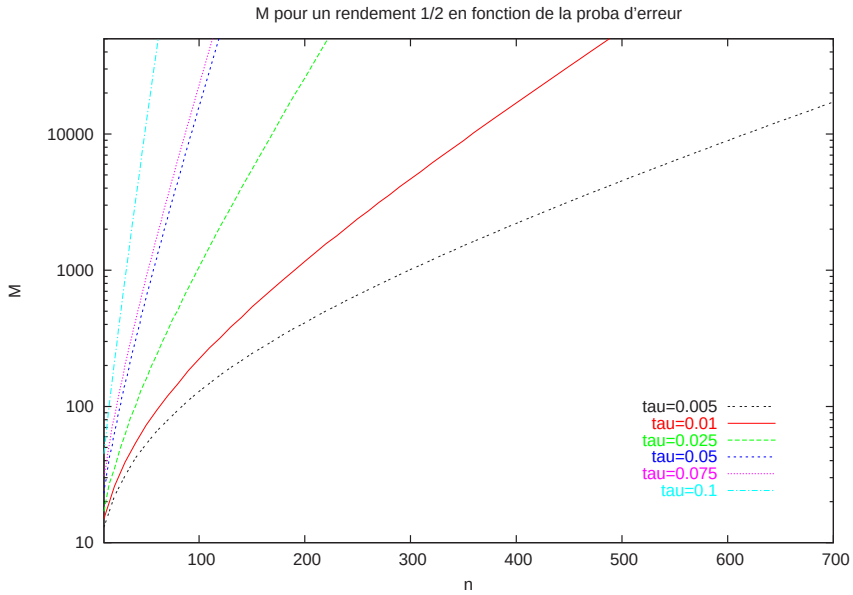

FIG. 2.16 – *Valeurs de M en fonction de n pour différentes valeurs de τ avec un code $\mathcal{C}$ de rendement $\frac{1}{2}$.*

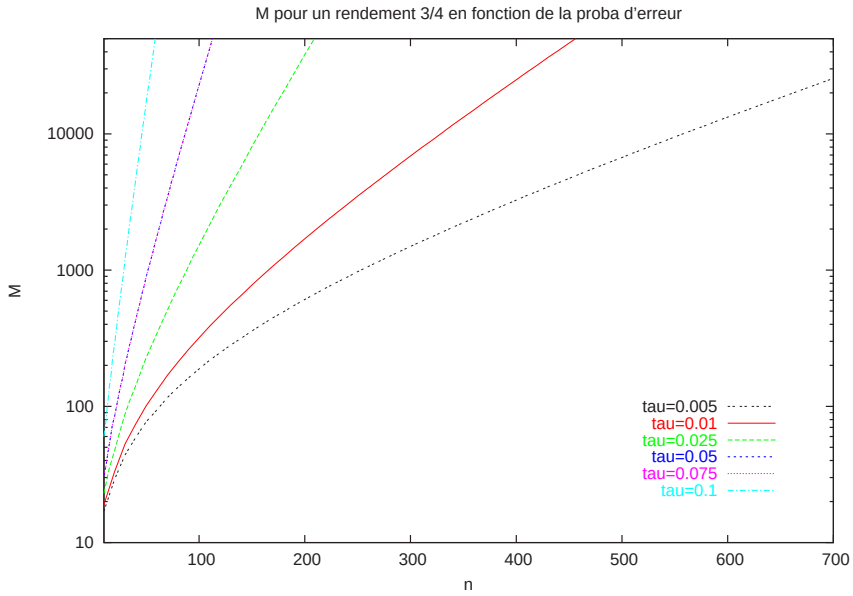

FIG. 2.17 – *Valeurs de M en fonction de n pour différentes valeurs de τ avec un code $\mathcal{C}$ de rendement $\frac{3}{4}$.*

Ces différentes courbes nous montrent que le taux d'erreur est un paramètre essentiel, plus important que le rendement, puisque la valeur de M augmente de manière significative avec τ.

Ensuite, nous nous sommes intéressés à la valeur du seuil T et nous l'avons comparée à la borne de Gilbert-Varshamov.

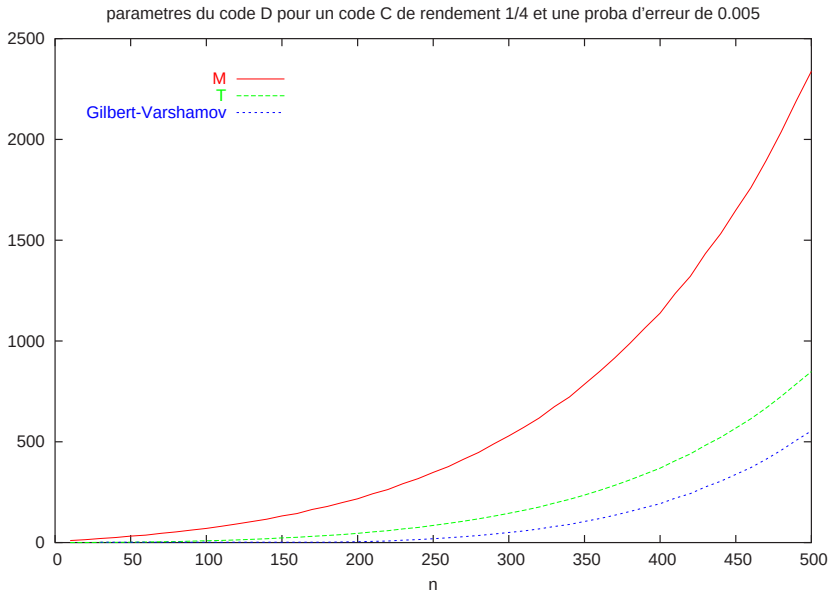

FIG. 2.18 – *Valeurs de M et T en fonction de n pour un code $\mathcal{C}$ de rendement $\frac{1}{4}$ et pour une probabilité d'erreur $\tau = 0.005$.*

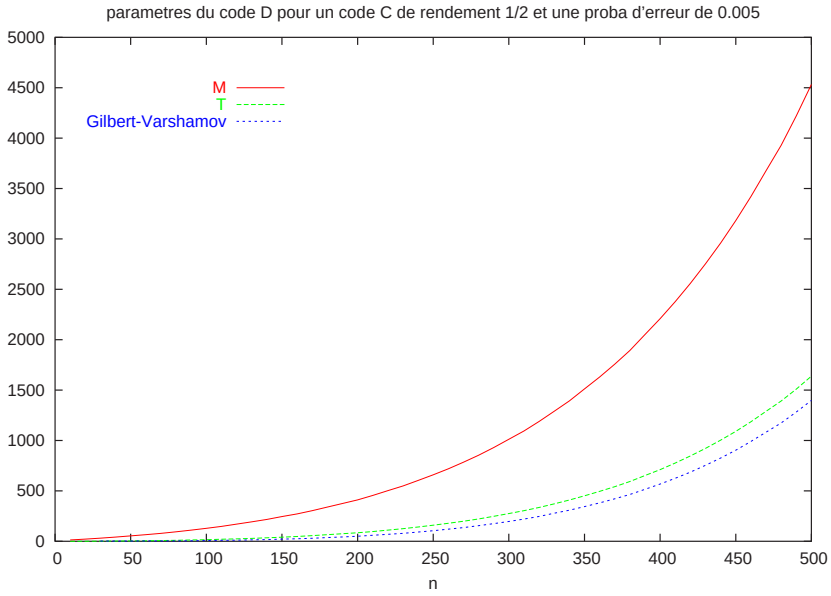

FIG. 2.19 – *Valeurs de M et T en fonction de n pour un code $\mathcal{C}$ de rendement $\frac{1}{2}$ et pour une probabilité d'erreur $\tau = 0.005$.*

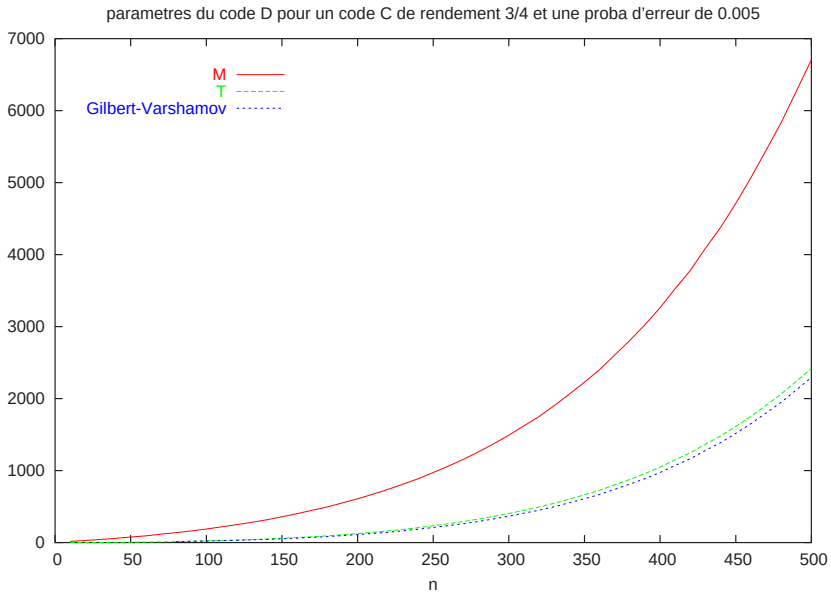

FIG. 2.20 – *Valeurs de M et T en fonction de n pour un code $\mathcal{C}$ de rendement $\frac{3}{4}$ et pour une probabilité d'erreur $\tau = 0.005$.*

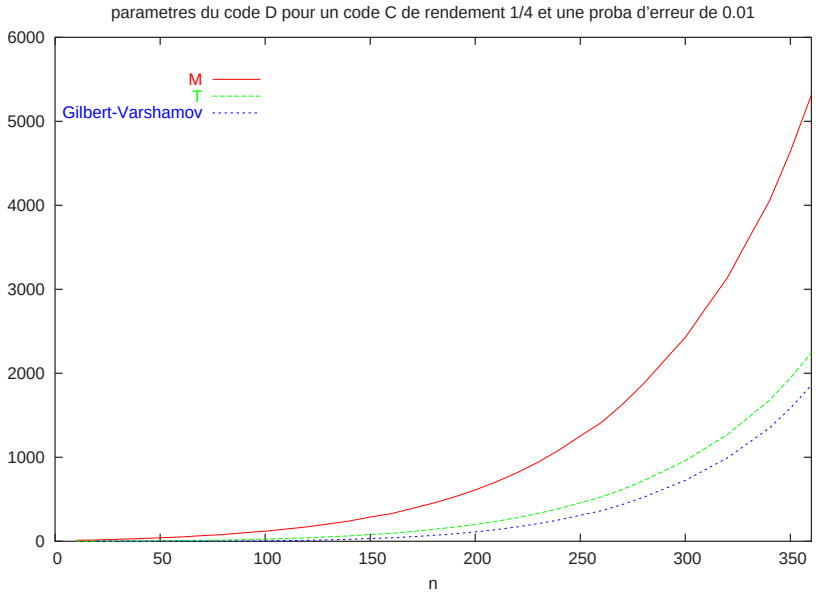

FIG. 2.21 – *Valeurs de M et T en fonction de n pour un code $\mathcal{C}$ de rendement $\frac{1}{4}$ et pour une probabilité d'erreur $\tau = 0.01$.*

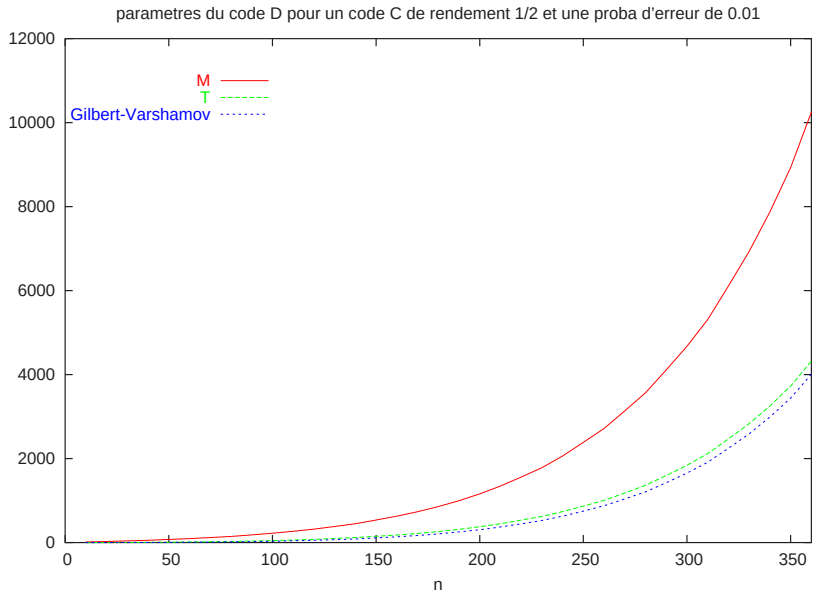

FIG. 2.22 – *Valeurs de M et T en fonction de n pour un code $\mathcal{C}$ de rendement $\frac{1}{2}$ et pour une probabilité d'erreur $\tau = 0.01$.*

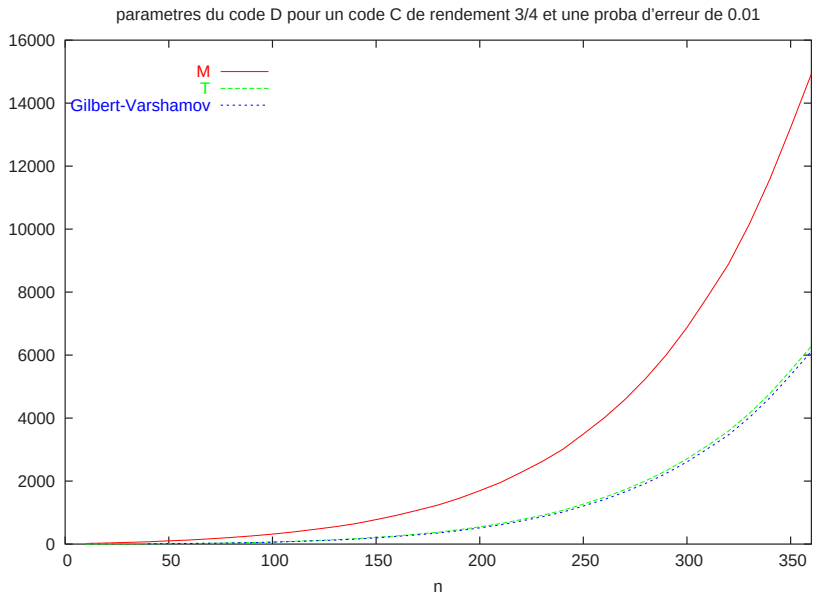

FIG. 2.23 – *Valeurs de M et T en fonction de n pour un code $\mathcal{C}$ de rendement $\frac{3}{4}$ et pour une probabilité d'erreur $\tau = 0.01$.*

On voit donc sur ces six figures, que la valeur de T s'approche de la borne de Gilbert-Varshamov lorsque τ augmente et surtout lorsque le rendement du code augmente. Cela

paraît assez naturel puisque quand le rendement du code augmente, le nombre de mots de $\mathcal{C}^\perp$ diminue. Donc, pour que la valeur de α soit très faible, il faut que la valeur du seuil T soit très proche de la borne de Gilbert-Varshamov.

2.5 Recherche de mots de poids faible

Nous venons de détailler un test statistique permettant de distinguer les mots du code dual $\mathcal{C}^\perp$ des autres : ce test consiste à sélectionner les mots h dont l'image hR^T dans le code $\mathcal{D}$, a un poids inférieur à un certain seuil T. Nous allons donc maintenant nous intéresser à la recherche de mots de poids inférieur à T dans le code $\mathcal{D}$.

2.5.1 Dimension du code $\mathcal{D}$

Tous les algorithmes de recherche de mots de poids faible dans un code linéaire manipulent une matrice génératrice du code. Ainsi, nous n'allons évidemment pas appliquer ces algorithmes sur la matrice R^T mais sur une matrice génératrice de $\mathcal{D}$, ce qui nous amène à nous intéresser au rang de la matrice R^T, *i.e.* à la dimension du code $\mathcal{D}$. Cette valeur dépend bien évidemment des valeurs de M et τ. Si la matrice R^T est de rang plein, on peut utiliser les algorithmes classiques de recherche de mots de poids faible dans un code. Lorsque ce n'est pas le cas, on va d'abord effectuer un pivot de Gauss sur la matrice R^T en gardant en mémoire les opérations sur les lignes. Si la matrice R^T est de rang r, on la met donc sous la forme

$$\left(\begin{array}{c|c} I_r & A \\ \hline 0 & 0 \end{array}\right)$$

Les $n-r$ dernières lignes de la matrice signifient donc qu'il existe $n-r$ combinaisons linéaires nulles indépendantes des lignes de R^T. Ces combinaisons linéaires ont donc de fortes chance de correspondre à des mots de $\mathcal{C}^\perp$. Plus précisément, avec les notations précédentes,

$$\sharp\{h \in \mathcal{C}^\perp \text{ tels que } wt_H(hR^T) = 0\} = 2^{n-k} \times \left(\frac{1}{2} + \frac{x}{2}\right)^M,$$

avec $x = (1-2\tau)^{wt_H(h)}$.

Remarquons que, si l'on trouve des mots h tels que $wt_H(hR^T) = 0$, alors ces mots ont très peu de chance de ne pas appartenir à $\mathcal{C}^\perp$ puisque

$$\sharp\{h \notin \mathcal{C}^\perp \text{ tels que } wt_H(hR^T) = 0\} = \frac{2^n - 2^{n-k}}{2^M} < 2^{n-M},$$

ce qui implique que ce nombre est strictement inférieur à 1 dès que $M \geq n$.

Exemple 2.4 *Pour le code BCH [15,7,5] décrit précédemment, nous déterminons la dimension de l'espace formé par les mots h pour lesquels $wt_H(hR^T) = 0$, que nous avons obtenue dans les simulations pour différentes probabilités d'erreur τ et des longueurs M du*

code $\mathcal{D}$ variant entre $(n+1)$ et la valeur M déterminée au théorème 2.7. Ce nombre est comparé avec le nombre de mots n'appartenant pas à $\mathcal{C}^\perp$ pour lesquels $wt_H(hR^T) = 0$.

$\tau = 0.01$				
M	*16*	*17*	*18*	*32*
$\dim\big(\mathrm{Ker}(R^T)\big)$	*5*	*5*	*4*	*4*
nb de $h \notin \mathcal{C}^\perp$ *dans* $\mathrm{Ker}(R^T)$	*0*	*0*	*0*	*0*

$\tau = 0.05$			
M	*16*	*20*	*298*
$\dim\big(\mathrm{Ker}(R^T)\big)$	*1*	*0*	*0*
nb de $h \notin \mathcal{C}^\perp$ *dans* $\mathrm{Ker}(R^T)$	*0*	*0*	*0*

Lorsque τ est faible, on voit que, en prenant $M = n+1$ et en faisant simplement un pivot de Gauss, on peut réussir à trouver des mots de $\mathcal{C}^\perp$. Par contre, lorsque τ est plus élevé, les mots h tels que $wt_H(hR^T) = 0$ sont beaucoup plus rares. Pour chercher des mots de poids faible dans le code $\mathcal{D}$, on va donc procéder de la manière suivante :

- on va d'abord prendre $M = n+1$ et chercher des mots h tels que $wt_H(hR^T) = 0$;
- puis prendre la valeur M déterminée grâce au théorème 2.7 et chercher des mots de poids faible dans la sous-matrice de R^T de rang $\mathrm{rg}(R^T)$.

Pour chercher ces autres candidats de $\mathcal{C}^\perp$, on va utiliser les techniques classiques de recherche de mots de poids faible dans un code.

2.5.2 Algorithme

La plupart de ces algorithmes ont été développés pour trouver la distance minimale d'un code ou pour décoder dans le contexte de la cryptanalyse du système de McEliece [McE78] ; c'est aussi le cas de l'algorithme que nous avons choisi de présenter ici puisque c'est celui que l'on a utilisé lors de nos simulations [CC98]. Le but de cet algorithme, appelé par la suite algorithme CC, est de trouver des mots de poids faible dans un code $[n,k]$ engendré par une matrice génératrice G (dans notre cas, il s'agit du code de longueur M engendré par R^T). À partir de cette matrice génératrice G, on va chercher des mots du code dont le poids de Hamming est inférieur à un certain seuil T. Remarquons ici que le seuil T, que nous avons déterminé au théorème 2.7, correspond effectivement à un poids de Hamming faible au regard de la longueur M du code $\mathcal{D}$ et très proche de la borne de Gilbert-Varshamov comme nous l'avons vu à la fin de la section précédente page 54.

Notre algorithme fait intervenir deux paramètres, s et p qui seront choisis de manière à optimiser la complexité. L'algorithme se déroule alors de la manière suivante : dans un premier temps, on choisit aléatoirement un ensemble d'information I pour le code $[n,k]$. On met G sous forme systématique de sorte que les k premières colonnes de G correspondent à l'ensemble d'information I.

$$P^{-1}G = G_I = (I_k|Z)$$

où P est une matrice $k \times k$ inversible. On sépare alors l'ensemble d'information en deux parties I_1 et I_2 de même taille et on choisit aléatoirement un ensemble J de s positions en dehors de l'ensemble d'information. On recherche alors tous les mots m qui vérifient :

$$wt_H(m_{|I_1}) = wt_H(m_{|I_2}) = p \text{ et } wt_H(m_{|J}) = 0.$$

Les mots m vérifiant cette propriété peuvent être facilement isolés. Ils correspondent en effet à la somme d'une combinaison linéaire de p lignes de G prises dans I_1 et d'une combinaison linéaire de p lignes prises dans I_2 qui prennent la même valeur sur J.

Un tel mot m a des chances d'avoir un poids de Hamming faible puisque il a un poids $2p$ sur $k+s$ positions et en pratique on choisira p petit, typiquement $p = 1$ ou $p = 2$. On calcule alors son poids de Hamming complet et on teste si ce dernier est plus faible que le seuil T, c'est-à-dire si $wt_H(m|_I Z) \leq T - 2p$ où $m|_I$ correspond aux k premières coordonnées de m. Dans ce cas, on retourne les coordonnées originales du mot trouvé, *i.e.* avant permutation des colonnes de G ; c'est-à-dire $m|_I P^{-1}$ car on a $m = m|_I G_I = m|_I P^{-1} G$.

Après avoir effectué cette procédure pour un ensemble d'information, on recommence avec un autre jusqu'à ce qu'on ait trouvé suffisamment de mots de code de petit poids. Pour accélérer le processus et pour éviter des éliminations de Gauss dont la complexité est le facteur limitant car elle est supérieure à celle du reste de l'algorithme, on ne change qu'une seule position de I à chaque étape comme dans [CC98].

Les paramètres s et p peuvent être optimisés [Can96]. Dans notre contexte, leurs valeurs dépendent fortement de la valeur de T déterminée au théorème 2.7.

Algorithme.
Entrée : Une matrice génératrice G, trois paramètres s, p et T.
Sortie : Des mots h tels que $wt_H(hG) < T$.

Choisir aléatoirement un ensemble d'information I :

$$P^{-1}G = G_I = \left(\; I_k \;\middle|\; Z \; \right).$$

Itérer en changeant une seule position dans l'ensemble d'information :

- Séparer I en deux parties de même taille et choisir aléatoirement un ensemble J de s positions en dehors de I.
- Trouver des mots m tels que $wt_H(m|_{I_1}) = wt_H(m|_{I_2}) = p$ et $wt_H(m|_J) = 0$.
- Si $wt_H(m|_I Z) \leq T - 2p$, retourner $m|_I P^{-1}$.

TAB. 2.8 – *Algorithme de recherche de mots de poids faible dans un code linéaire.*

Dans les algorithmes que nous venons de présenter, nous effectuons simplement des tests statistiques pour savoir si un mot appartenait ou non au code dual. Notre idée est maintenant d'essayer d'améliorer cette décision et en particulier d'affiner la probabilité que

nous avons affectée à chacun des mots h, candidats pour être des éléments de $\mathcal{C}^{\perp}$. Pour cela, nous allons essayer d'exploiter ces mots, en tant qu'équations de parité, dans un algorithme de décodage itératif que nous décrivons au chapitre suivant.

Chapitre 3

Décodage itératif fondé sur des équations de parité probables

3.1 Motivations

Dans le chapitre précédent, nous avons décrit précisément les algorithmes suggérés par Antoine Valembois dans sa thèse et analysé l'influence de leurs paramètres. Ces algorithmes effectuent des tests statistiques pour déterminer si un mot appartient ou non au code dual. Ils sont efficaces seulement si la probabilité d'erreur du canal est relativement faible et s'il est possible de trouver des mots de poids faible dans le code $\mathcal{D}$ grâce à l'algorithme Canteaut-Chabaud. Pour les améliorer et notamment pouvoir traiter des taux d'erreur plus élevés, notre idée principale est alors d'essayer de décoder en utilisant les équations de parité (mots du dual) que nous a fournies l'étape précédente, afin à la fois de décoder les mots reçus et d'en déduire une information supplémentaire sur la fiabilité des équations de parité. Un des objectifs est de faciliter la recherche de mots de poids faible dans $\mathcal{D}$. En effet, il devient possible de considérer des mots du code $\mathcal{D}$ de poids plus élevé même s'ils sont moins fiables, dans la mesure où l'algorithme de décodage permet ensuite de les valider ou de les écarter. De plus, même si l'on ne retrouve qu'une partie du code dual, c'est-à-dire un nombre insuffisant d'équations de parité, on peut toutefois espérer décoder partiellement ce qui permet d'éliminer une partie des erreurs dans la matrice R. On peut ensuite itérer le processus car, comme le taux d'erreur sera plus faible, il devrait être plus facile de reconstruire le code dual. Il faut noter que les mots h tels que $wt_H(hR^T) = 0$ ne nous serviront pas pour décoder puisque ces relations de parité sont déjà vérifiées pour tous les mots reçus. Par contre il faut les inclure dans notre algorithme de décodage car elles confortent la décision sur les bits qui y interviennent. Plus précisément, une fois que nous avons trouvé des mots pour le dual, notre idée est de leur associer la probabilité appartiennent effectivement au dual et d'utiliser ces équations de pondérées par leurs probabilités pour décoder. Nous allons donc modifier l'algorithme de décodage de Gallager pour qu'il prenne en compte la fiabilité des différentes équations de parité. La section suivante a pour but de rappeler quelques généralités sur les codes LDPC, mais

surtout de présenter l'algorithme de Gallager.

3.2 Les codes LDPC et l'algorithme de Gallager

Les codes LDPC (Low Density Parity Check codes) ont été inventés par R. Gallager lors de sa thèse [Gal63]. Peu après leur conception, ils ont été oubliés et redécouverts de nombreuses fois lors de ces trente dernières années.

3.2.1 Présentation des codes LDPC

Les codes LDPC forment une famille de codes linéaires obtenus à partir de graphes biparties creux. Soit $\mathcal{G}$ un graphe avec n nœuds gauches et r nœuds 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 nœuds gauches. Les mots de code sont alors les vecteurs $(c_1,\ldots,c_n)$ tels que la somme de tous les nœuds gauches reliés à chacun des nœuds droits vaut zéro. Cette représentation sous forme de graphe est analogue à une représentation sous forme matricielle en considérant la matrice d'adjacence du graphe. Soit H une matrice binaire $r \times n$ dans laquelle $H[i,j]$ vaut 1 si et seulement si le i-ème nœud droit du graphe $\mathcal{G}$ est connecté au j-ème nœud gauche. Alors, le code LDPC associé au graphe $\mathcal{G}$ est le code dont une matrice de parité est la matrice H, matrice d'adjacence de $\mathcal{G}$. 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 a exactement i éléments égaux à 1 ;
- toute ligne a j éléments égaux à 1.

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

$$i \times n = j \times (n-k).$$

Exemple 3.1 *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.*

C'est justement le fait que la matrice de parité soit creuse qui permet d'avoir des algorithmes de décodage qui sont très performants.

3.2.2 Algorithme de décodage classique des codes LDPC

Dans [Gal63], R. Gallager introduit un algorithme de décodage des codes LDPC. Cet algorithme est un algorithme itératif et probabiliste dans lequel on associe à chaque symbole

la probabilité qu'il vaille 1. À chaque itération, on met à jour ces probabilités en utilisant les équations de parité (creuses) que vérifient les mots de code. Dans toute la suite de ce paragraphe, nous décrivons l'algorithme de Gallager dans le cas d'une transmission sur un canal binaire symétrique de probabilité d'erreur τ. On notera x l'événement $x = 1$ et $\overline{x}$ l'événement contraire ($x = 0$). On notera $\Pr[c_t]$ la probabilité de l'événement c_t, *i.e.* la probabilité que le bit c_t soit égal à 1. L'information dont on dispose lorsqu'on essaye de décoder un mot est celle qui résulte de l'observation, c'est-à-dire du mot reçu. Si on reçoit un bit qui vaut 1, alors la probabilité que la variable aléatoire associée à ce bit vaille 1 est égale à $1 - \tau$, alors que si ce bit vaut zéro la probabilité qu'il vaille 1 est égale à τ.

On associe naturellement à tout mot du code dual du code LDPC une équation de parité vérifiée par les mots de code. En effet, si $h = (h_1, \ldots, h_n) \in \mathcal{C}^\perp$, alors $\forall c \in \mathcal{C}$, $ch^T = 0$. On a donc :

$$\sum_{i \in Supp(h)} c_i = 0.$$

Une équation de poids d

$$EQ : \quad c_{t_0} + \sum_{j=1}^{d-1} c_{t_j} = 0,$$

permet alors de calculer une nouvelle probabilité pour le bit c_{t_0} en utilisant les autres bits c_{t_j}. Supposons que l'on connaisse $\Pr[c_{t_j}]$ pour chaque t_j, et que l'on souhaite calculer $\Pr[c_{t_0}|EQ]$. On a donc

$$\Pr[c_{t_0}|EQ] = \Pr\left[\sum_{j=1}^{d-1} c_{t_j}\right].$$

Lemme 3.1 *Considérons n bits indépendants qui ont chacun une probabilité p_i de valoir 1. Alors, la probabilité qu'un nombre impair de ces bits valent 1 est égale à :*

$$\frac{1 - \prod_{i=1}^{n}(1 - 2p_i)}{2}.$$

Preuve.
Considérons le polynôme en la variable t :

$$\prod_{i=1}^{n}(1 - p_i + tp_i).$$

Si on développe ce polynôme, on se rend compte que le coefficient de t^k correspond à la probabilité qu'on ait exactement k bits parmi les n qui soient égaux à 1. Si on considère maintenant le polynôme $\prod_{i=1}^{n}(1-p_i-tp_i)$, on a exactement le même résultat hormis le fait que le coefficient de t^k est précédé d'un facteur $(-1)^k$. Donc, la probabilité qu'un nombre impair parmi les n bits considérés valent 1 est obtenue en faisant la différence de ces deux polynômes, en substituant 1 à t et en divisant par deux. On obtient donc que la probabilité cherchée est égale à :

$$\frac{\prod_{i=1}^{n}(1 - p_i + p_i) - \prod_{i=1}^{n}(1 - p_i - p_i)}{2}.$$

◇

Ce lemme nous permet donc de déterminer la probabilité que $c_{t_0} = 1$ sachant que l'équation de parité EQ est satisfaite :

$$\Pr[c_{t_0}|EQ] = \frac{1 - \prod_{i=1}^{d-1}(1 - 2\Pr[c_{t_j}])}{2}.$$

Maintenant, pour mettre à jour la probabilité pour un certain bit c_t, il faut combiner les diverses informations dont on dispose : il s'agit des informations provenant de toutes les équations dans lesquelles ce bit apparaît ainsi que de l'information résultant de l'observation. La combinaison de toutes ces informations se fait au moyen d'une loi de composition dont la définition repose sur le lemme suivant :

Lemme 3.2 *On suppose que les bits en sortie du canal sont équiprobables, c'est-à-dire que* $\Pr[x] = \Pr[\overline{x}]$ *et que le canal est sans mémoire ce qui signifie que* $\Pr[x_1,x_2|y] = \Pr[x_1|y]\Pr[x_2|y]$. *On a alors:*

$$\Pr[x|y_1,y_2] = \frac{\Pr[x|y_1]\Pr[x|y_2]}{\Pr[x|y_1]\Pr[x|y_2] + \Pr[\overline{x}|y_1]\Pr[\overline{x}|y_2]}.$$

Preuve.
Notons $p = \Pr[x|y_1,y_2]$. En utilisant la formule de Bayes on a :

$$p = \frac{\Pr[y_1,y_2|x]\Pr[x]}{\Pr[y_1,y_2]}.$$

On sait que le canal est sans mémoire, donc :

$$p = \frac{\Pr[y_1|x]\Pr[y_2|x]\Pr[x]}{\Pr[y_1,y_2]}.$$

En utilisant à nouveau la formule de Bayes, on trouve :

$$p = \frac{\Pr[x|y_1]\Pr[x|y_2]}{\frac{\Pr[x]\Pr[y_1,y_2]}{\Pr[y_1]\Pr[y_2]}}.$$

Le même calcul mené en remplaçant x par $\overline{x}$, conduit à :

$$1 - p = \frac{\Pr[\overline{x}|y_1]\Pr[\overline{x}|y_2]}{\frac{\Pr[\overline{x}]\Pr[y_1,y_2]}{\Pr[y_1]\Pr[y_2]}}.$$

Mais on sait que $\Pr[x] = \Pr[\overline{x}]$ donc, en faisant la somme, on en déduit que

$$\frac{\Pr[x]\Pr[y_1,y_2]}{\Pr[y_1]\Pr[y_2]} = \Pr[x|y1]\Pr[x|y_2] + \Pr[\overline{x}|y_1]\Pr[\overline{x}|y_2].$$

On obtient ainsi le résultat souhaité. ◇

Définition 3.3 *Soient u et v deux réels de l'intervalle* [0,1] *avec* $(u,v) \neq (0,1)$ *et* $(u,v) \neq (1,0)$. *La loi* $\otimes$ *est définie de la manière suivante :*

$$u \otimes v = \frac{uv}{uv + (1-u)(1-v)}.$$

On remarque donc que

$$\Pr[x|y_1,y_2] = \Pr[x|y_1] \otimes \Pr[x|y_2].$$

Cette loi $\otimes$ va alors intervenir pour calculer la probabilité pour un bit c_t en combinant toutes les informations dont on dispose. Remarquons par ailleurs que cette loi possède les propriétés suivantes qui pourront être utiles par la suite.

Proposition 3.4

- *La loi* $\otimes$ *est associative et commutative ;*
- $\frac{1}{2}$ *est un élément neutre pour la loi* $\otimes$ *;*
- 0 *et* 1 *sont des éléments absorbants pour cette loi,* i.e.

$$\forall u \neq 1, u \otimes 0 = 0 \text{ et } \forall u \neq 0,\ u \otimes 1 = 1.$$

Preuve.

- On a :

$$\begin{aligned} u \otimes (v \otimes w) &= \frac{u\frac{vw}{vw+(1-v)(1-w)}}{u\frac{vw}{vw+(1-v)(1-w)}+(1-u)(1-\frac{vw}{vw+(1-v)(1-w)})} \\ &= \frac{uvw}{uvw+(1-u)(1-v)(1-w)} \\ &= \frac{\frac{uv}{uv+(1-u)(1-v)}w}{\frac{uv}{uv+(1-u)(1-v)}w+(1-\frac{uv}{uv+(1-u)(1-v)})(1-w)} \\ &= (u \otimes v) \otimes w. \end{aligned}$$

  La loi $\otimes$ est donc bien associative.

- Les autres propriétés sont triviales.

◇

On peut aussi remarquer que $\Pr[x|y_1] \otimes \Pr[x|y_2]$ est proportionnel au produit $\Pr[x|y_1] \times \Pr[x|y_2]$ et que le facteur de proportionnalité est égal à

$$\frac{1}{\Pr[x|y_1]\Pr[x|y_2] + \Pr[\overline{x}|y_1]\Pr[\overline{x}|y_2]}.$$

De même $\Pr[\overline{x}|y_1] \otimes \Pr[\overline{x}|y_2]$ est proportionnel à $\Pr[\overline{x}|y_1] \times \Pr[\overline{x}|y_2]$ avec le même coefficient de proportionnalité. C'est pour cela que, dans l'algorithme de Gallager, on peut soit choisir de mener les calculs avec la loi $\otimes$, soit se contenter du produit, mais dans ce dernier cas, on

doit aussi garder en mémoire les probabilités des événements $\overline{x}$ afin de pouvoir déterminer le coefficient de proportionnalité.

La description détaillée de l'algorithme de décodage nécessite maintenant un certain nombre de notations regroupées à la table 3.1.

$\mathrm{Obs}(c_t)$	:	la probabilité de c_t provenant de l'observation, c'est-à-dire du mot reçu
$\mathrm{Extr}_{\mathrm{e}}(c_t)$	:	la probabilité de c_t qui vient de l'équation d'indice e
$\mathrm{A}_{\mathrm{e}}(c_t)$	:	l'APP (la probabilité a posteriori) partielle pour c_t c'est-à-dire la probabilité calculée en utilisant toutes les informations sauf celle qui provient de l'équation e
g	:	la fonction log-vraisemblance, *i.e.* $g(x) = \log\left(\frac{x}{1-x}\right)$

TAB. 3.1 – *Notations utilisées pour les algorithmes de décodage itératif.*

L'algorithme de Gallager se décompose en trois phases ; il est décrit dans la table 3.2.

Algorithme.
Entrée : Un mot de code bruité $(c_1,\ldots,c_n)$.
Sortie : Ce mot de code décodé.

- **Initialisation :**
  Pour tout $t \in \{1,\ldots,n\}$,
  - Calculer $\mathrm{Obs}(c_t) = \begin{cases} 1-\tau & \text{si l'observation est un 1.} \\ \tau & \text{si l'observation est un 0.} \end{cases}$
  - $\forall e$, $\mathrm{Extr_e}(c_t) = \frac{1}{2}$.
- **Itérer :**
  Pour tout $t \in \{1,\ldots,n\}$,
  - Calculer :
$$\forall e,\ \mathrm{A_e}(c_t) = \mathrm{Obs}(c_t) \otimes \left[\bigotimes_{i\neq e} \mathrm{Extr_i}(c_t)\right].$$
  - Calculer :
$$\forall e,\ \mathrm{Extr_e}(c_t) = \frac{1}{2}\left[1 - \prod_{i\in I_e\setminus\{t\}} (1-2\mathrm{A_e}(c_i))\right],$$
    avec $I_e = \{i | a_i$ appartient à l'équation $e\}$.
- **Terminaison :**
  Pour tout $t \in \{1,\ldots,n\}$,
  - calculer
$$\mathrm{APP}(c_t) = \mathrm{Obs}(c_t) \otimes \left[\bigotimes_{i} \mathrm{Extr_i}(c_t)\right].$$
  - Si $\mathrm{APP}(c_t) > \frac{1}{2}$ alors $c_t = 1$, sinon $c_t = 0$.

TAB. 3.2 – *Algorithme de Gallager.*

Proposition 3.5 *Soit g l'application définie sur $]0,1[$ par :*

$$g(x) = \log\left(\frac{x}{1-x}\right).$$

Cette application g est une bijection et sa bijection réciproque est la fonction g^{-1} définie sur $\mathbb{R}$ par

$$g^{-1}(x) = \frac{\exp(x)}{1+\exp(x)}.$$

De plus, on a

$$g(u \otimes v) = g(u) + g(v).$$

Preuve.
Commençons par montrer que g est bijective et admet pour bijection réciproque g^{-1}.

$$\begin{aligned}
(g \circ g^{-1})(x) &= g\left[g^{-1}(x)\right] \\
&= g\left(\frac{\exp(x)}{1+\exp(x)}\right) \\
&= \log\left[\frac{\frac{\exp(x)}{1+\exp(x)}}{1-\frac{\exp(x)}{1+\exp(x)}}\right] \\
&= \log(\exp(x)) \\
&= x.
\end{aligned}$$

De même,

$$\begin{aligned}
(g^{-1} \circ g)(x) &= g^{-1}\left[g(x)\right] \\
&= g^{-1}\left(\log\left(\frac{x}{1-x}\right)\right) \\
&= \frac{\frac{x}{1-x}}{1+\frac{x}{1-x}} \\
&= x.
\end{aligned}$$

Il ne nous reste plus qu'à montrer que $g(u \otimes v) = g(u) + g(v)$.

$$\begin{aligned}
g(u \otimes v) &= \log\left(\frac{\frac{uv}{uv+(1-u)(1-v)}}{1-\frac{uv}{uv+(1-u)(1-v)}}\right) \\
&= \log\left(\frac{uv}{(1-u)(1-v)}\right) \\
&= g(u) + g(v).
\end{aligned}$$

◇

Pour implanter notre algorithme nous avons tenu compte de cette propriété puisqu'il est alors plus rapide, pour déterminer la valeur de

$$\mathrm{Obs}(c_t) \otimes \left[\bigotimes_i \mathrm{Extr_i}(c_t)\right],$$

de calculer

$$g^{-1}\left(g\left(\mathrm{Obs}(c_t)\right) + \left[\sum_i g\left(\mathrm{Extr_i}(c_t)\right)\right]\right).$$

De plus, en pratique, on précalcule des tables pour g et g^{-1} et on quantifie nos probabilités, ce qui permet d'accélérer considérablement les calculs.

Variante avec les log-vraisemblances. Pour éviter des calculs de la loi ⊗coûteuse, on utilise aussi souvent une autre forme de l'algorithme de Gallager qui utilise tout le temps la fonction g, qui est appelé "log-vraisemblance". Au lieu de stocker et de mettre à jour les probabilités des bits c_t, on manipule les vraisemblances de ces bits. Cet algorithme est décrit à la table 3.3.

Algorithme.
Entrée : Un mot de code bruité $(c_1, \ldots, c_n)$.
Sortie : Ce mot de code décodé.

- **Initialisation :**
  Pour tout $t \in \{1, \ldots, n\}$,
  - Calculer $g\left(\mathrm{Obs}(c_t)\right) = \begin{cases} \log\left(\frac{1-\tau}{\tau}\right) & \text{si l'observation est un 1.} \\ \log\left(\frac{\tau}{1-\tau}\right) & \text{si l'observation est un 0.} \end{cases}$
  - $\forall e,\ g\left(\mathrm{Extr_e}(c_t)\right) = 0.$
- **Itérer :**
  Pour tout $t \in \{1, \ldots, n\}$,
  - Calculer :

$$\forall e,\ g\left(\mathrm{A_e}(c_t)\right) = g\left(\mathrm{Obs}(c_t)\right) + \left[\sum_{i \neq e} g\left(\mathrm{Extr_i}(c_t)\right)\right].$$

  - Calculer :

$$\forall e,\ g\left(\mathrm{Extr_e}(c_t)\right) = \log\left(\frac{1 - \prod_{i \in I_e \setminus \{t\}} \frac{1-\exp(g(A_e(c_i)))}{1+\exp(g(A_e(c_i)))}}{1 + \prod_{i \in I_e \setminus \{t\}} \frac{1-\exp(g(A_e(c_i)))}{1+\exp(g(A_e(c_i)))}}\right),$$

    avec $I_e = \{i | a_i \text{ appartient à l'équation } e\}$.
- **Terminaison :**
  Pour tout $t \in \{1, \ldots, n\}$,
  - calculer

$$g\left(\mathrm{APP}(c_t)\right) = g\left(\mathrm{Obs}(c_t)\right) + \left[\sum_{i} g\left(\mathrm{Extr_i}(c_t)\right)\right].$$

  - Si $g\left(\mathrm{APP}(c_t)\right) > 0$ alors $c_t = 1$, sinon $c_t = 0$.

TAB. 3.3 – *Algorithme de Gallager avec les log-vraisemblances.*

Pour démontrer la validité de la variante avec les log-vraisemblances, avec ce que nous avons vu auparavant, la seule chose qui reste à montrer est la formule de remise à jour de

$g\left(\mathrm{Extr_e}(c_t)\right)$. Par définition de g, on a

$$\begin{aligned} g\left(\mathrm{Extr_e}(c_t)\right) &= \log\left(\frac{\frac{1}{2}\left[1-\prod_{i\in I_e\setminus\{t\}}(1-2\mathrm{A_e}(c_i))\right]}{1-\frac{1}{2}\left[1-\prod_{i\in I_e\setminus\{t\}}(1-2\mathrm{A_e}(c_i))\right]}\right) \\ &= \log\left(\frac{1-\prod_{i\in I_e\setminus\{t\}}(1-2\mathrm{A_e}(c_i))}{1+\prod_{i\in I_e\setminus\{t\}}(1-2\mathrm{A_e}(c_i))}\right). \end{aligned}$$

Comme on connaît seulement les $g(\mathrm{A_e}(c_i)$, on peut écrire :

$$g\left(\mathrm{Extr_e}(c_t)\right) = \log\left(\frac{1-\prod_{i\in I_e\setminus\{t\}}(1-2g^{-1}\left[g\left(\mathrm{A_e}(c_i)\right)\right])}{1+\prod_{i\in I_e\setminus\{t\}}(1-2g^{-1}\left[g\left(\mathrm{A_e}(c_i)\right)\right])}\right),$$

mais par définition de g^{-1} :

$$\begin{aligned} 1-2g^{-1}\left[g\left(\mathrm{A_e}(c_i)\right)\right]) &= 1-2\frac{\exp\left[g\left(\mathrm{A_e}(c_i)\right)\right]}{1+\exp\left[g\left(\mathrm{A_e}(c_i)\right)\right]} \\ &= \frac{1-\exp\left[g\left(\mathrm{A_e}(c_i)\right)\right]}{1+\exp\left[g\left(\mathrm{A_e}(c_i)\right)\right]}. \end{aligned}$$

Analyse. En mémoire, il faut stocker les n observations et $2\times n\times \mathrm{nb_{eq}}$ probabilités où $\mathrm{nb_{eq}}$ est le nombre moyen d'équations par bit. Le calcul d'une APP partielle nécessite $\mathrm{nb_{eq}}$ opérations, celui d'une extrinsèque environ d opérations où d est le poids de l'équation de parité. Il y a $\mathrm{nb_{iter}}$ itérations d'où une complexité de la phase d'itération d'environ

$$\mathrm{nb_{iter}}\times n\times \mathrm{nb_{eq}}(\mathrm{nb_{eq}}+d).$$

Le calcul de l'APP finale requiert lui $n\times \mathrm{nb_{eq}}$ opérations. D'où une complexité en calculs totale de :

$$\mathcal{O}\left(\mathrm{nb_{iter}} n \mathrm{nb_{eq}}(\mathrm{nb_{eq}}+d)\right).$$

Si, à chaque itération de l'algorithme de décodage, les messages entrant sont statistiquement indépendants alors la mise à jour des probabilités est exacte. Cependant, cette condition est vérifiée pour les ℓ premières itérations de l'algorithme seulement si, pour un nœud du message, les voisins de profondeur inférieure à ℓ forment un arbre. Ceci explique qu'on cherche toujours à éviter qu'il y ait des cycles courts dans le graphe de Tanner associé au code LDPC.

L'analyse de la convergence des algorithmes de décodage itératif est un problème très complexe qui a été traité pour la première fois dans [LMS01]. Cette analyse a été généralisée dans [RU01] pour de nombreux canaux de communications.

En pratique, on se limitera à un nombre d'itérations donné qui sera sauf indications contraires de 10 itérations.

3.3 Une première tentative

Dans cette section, nous présentons un premier algorithme de reconstruction d'un code en bloc fondé sur des techniques de décodage itératif. Nous exposons d'abord la première idée qui nous a paru naturelle pour modifier l'algorithme de Gallager afin qu'il prenne en compte les probabilités de chacune des équations de parité et qu'il les remette à jour. Dans cet algorithme, on remet à jour les probabilités des équations en essayant de faire le même type de calcul que pour les remises à jour des probabilités a posteriori dans l'algorithme de Gallager. Nous devons tout d'abord calculer une première estimation de la probabilité qu'une équation de parité e soit vérifiée, notée $\mathrm{Est_e}$. Cette probabilité jouera en quelque sorte le rôle de l'observation, mais pour les équations de parité. Ce travail a été présenté récemment à Seattle [Clu06a]. Cependant, la poursuite des simulations a montré qu'il avait de mauvaises performances ce qui nous a amené à le corriger. Nous allons tout de même commencer par présenter cette version qui marche mal puisqu'elle résulte d'une idée qui nous a paru naturelle et qu'il est instructif d'analyser les raisons des mauvaises performances.

3.3.1 Calcul de $\mathrm{Est_e}$

Les équations de parité h dont nous disposons proviennent de l'algorithme de recherche de mots de poids faible. La probabilité que h soit une équation de parité valide est donc égale à la probabilité que $h \in \mathcal{C}^\perp$, connaissant le poids de Hamming de hR^T :

$$\Pr[h \in \mathcal{C}^\perp | wt_H(hR^T) = w].$$

Proposition 3.6 *On a :*

$$\Pr[h \in \mathcal{C}^\perp | wt_H(hR^T) = w] = \frac{\Pr[h \in \mathcal{C}^\perp]\binom{M}{w}(\frac{1}{2}-x)^w(\frac{1}{2}+x)^{M-w}}{\Pr[h \in \mathcal{C}^\perp]\binom{M}{w}(\frac{1}{2}-x)^w(\frac{1}{2}+x)^{M-w} + \Pr[h \notin \mathcal{C}^\perp]\frac{\binom{M}{w}}{2^M}}.$$

Preuve.
On sait simplement calculer $\Pr[wt_H(hR^T) = w | h \in \mathcal{C}^\perp]$ et $\Pr[wt_H(hR^T) = w | h \notin \mathcal{C}^\perp]$. On rappelle qu'on note m_i un mot de code et $\widetilde{m}_i$ le mot de code bruité associé, on a :

$$R^T = (\widetilde{m}_1 \widetilde{m}_2 \cdots \widetilde{m}_M).$$

Notons p la probabilité cherchée. On va aussi noter $P_0 = \Pr[wt_H(hR^T) = w | h \in \mathcal{C}^\perp]$ et $P_1 = \Pr[wt_H(hR^T) = w | h \notin \mathcal{C}^\perp]$. On peut alors calculer ces probabilités en utilisant les formules de Bayes :

$$P_0 = \frac{p\Pr[wt_H(hR^T) = w]}{\Pr[h \in \mathcal{C}^\perp]},$$

$$P_1 = \frac{(1-p)\Pr[wt_H(hR^T) = w]}{\Pr[h \notin \mathcal{C}^\perp]}.$$

La probabilité p cherchée est donc égale à :

$$\frac{\Pr[h \in \mathcal{C}^\perp]P_0}{\Pr[h \in \mathcal{C}^\perp]P_0 + \Pr[h \notin \mathcal{C}^\perp]P_1}.$$

Si $h \in \mathcal{C}^\perp$, alors $wt_H(hR^T)$ suit une distribution binomiale de probabilité $\frac{1}{2} - x$, on a donc :

$$P_0 = \binom{M}{w} \left(\frac{1}{2} - x\right)^w \left(\frac{1}{2} + x\right)^{M-w}.$$

Calculons maintenant P_1. Pour des grandes valeurs de M, on peut supposer que les $\widetilde{m}_i$ sont aléatoires et indépendants et alors on a :

$$\Pr[wt_H(hR^T) = w | h \notin \mathcal{C}^\perp] = \frac{\binom{M}{w}}{2^M},$$

d'où le résultat. ◇

Finalement, on peut écrire cette probabilité comme une fonction de $\Pr[h \in \mathcal{C}^\perp]$. Malheureusement, on ne peut pas calculer cette probabilité, en effet $\Pr[h \in \mathcal{C}^\perp]$ dépend de la distribution des poids de $\mathcal{C}^\perp$ et bien sûr, on ne peut pas connaître cette distribution puisque l'on est en train de chercher le code $\mathcal{C}$ qui a été utilisé. Toutefois, en pratique, pour des schémas de communication comme ceux que l'on envisage, on utilise seulement certains codes bien connus. Il arrive alors que l'on connaisse leur distribution des poids, au moins pour les poids faibles (et c'est exactement ce qui nous intéresse ici). Donc si l'on suppose que le code utilisé appartient à une certaine famille de codes pour lesquels on peut estimer le nombre de mots de poids faible, on est capable d'estimer la probabilité cherchée et par conséquent de calculer une valeur de $\mathrm{Est_e}$ pour chaque équation de parité e.

3.3.2 Principe de l'algorithme

Dans le cadre classique de l'algorithme de Gallager, on rappelle que pour chaque itération, on a deux opérations de remise à jour des probabilités. D'abord, on remet à jour les probabilités a posteriori partielles $\mathrm{A_e}(c_t)$ en combinant par la loi $\otimes$ toutes les informations dont on dispose sur le bit auquel on s'intéresse à l'exception de celle venant de l'équation e. L'autre opération consiste à remettre à jour la probabilité d'un bit suivant une équation donnée, $\mathrm{Extr_e}(c_t)$, en fonction des probabilités des autres bits intervenant dans cette équation. Dans cette étape on se sert du fait que, si l'on a une équation de poids d :

$$EQ \ : \ c_{t_0} + \sum_{j=1}^{d-1} c_{t_j} = 0,$$

alors on peut calculer :

$$\Pr[c_{t_0} | EQ] = \Pr\left[\sum_{j=1}^{d-1} c_{t_j}\right] = \frac{1 - \prod_{i=1}^{d-1}(1 - 2\Pr[c_{t_j}])}{2},$$

où c_t est l'événement « le bit c_t vaut 1 », Nous allons maintenant regarder comment cette procédure doit être modifiée lorsque cette équation est vérifiée avec une certaine probabilité seulement, et non pas avec probabilité 1 comme dans le cas de l'algorithme de Gallager. Supposons donc maintenant que la même équation

$$EQ \ : \quad c_{t_0} + \sum_{j=1}^{d-1} c_{t_j} = 0,$$

soit vraie avec une probabilité p. On souhaite alors remettre à jour la probabilité du bit c_{t_0} en utilisant la probabilité des autres bits intervenant dans l'équation et le fait que cette équation est vérifiée avec probabilité p. On a a alors :

$$\Pr\left[c_{t_0} | \Pr[EQ] = p\right] = p\Pr\left[\sum_{j=1}^{d-1} c_{t_j} = 1\right] + (1-p)\Pr\left[\sum_{j=1}^{d-1} c_{t_j} = 0\right].$$

La probabilité cherchée est donc :

$$p.\frac{1 - \prod_{i=1}^{d-1}(1 - 2\Pr[c_{t_j}])}{2} + (1-p)\frac{1 + \prod_{i=1}^{d-1}(1 - 2\Pr[c_{t_j}])}{2}.$$

Pour la phase où l'on utilise toutes les informations que l'on a sur un bit, c'est-à-dire les informations venant des équations et de l'observation, la formule reste inchangée : il faut simplement faire le produit $\otimes$ de toutes ces probabilités. Par contre, il va falloir introduire une nouvelle phase pour remettre à jour la probabilité de chacune des équations à partir des probabilités des bits qui y interviennent. On pourra avoir une telle probabilité pour chaque mot que l'on décode ; on va donc décoder tous nos mots simultanément pour avoir le plus d'informations possibles. On notera $\widetilde{p}_{e,i}$ la probabilité que l'équation e soit vraie en utilisant les probabilités des autres bits intervenant dans l'équation e lorsqu'on décode le mot $\widetilde{m}_i$. On rappelle que I_e désigne l'ensemble des indices des bits qui apparaissent dans l'équation e. On a :

$$\widetilde{p}_{e,i} = \frac{1 + \prod_{t \in Ie}(1 - 2\mathrm{A_e}(c_t))}{2}.$$

Pour l'équation e, on dispose donc de plusieurs informations : on a une première estimation de cette probabilité notée $\mathrm{Est_e}$ puis, pour chaque mot $\widetilde{m}_i$ que l'on décode, on calcule $\widetilde{p}_{e,i}$. On remettra donc à jour la probabilité que l'équation e soit vraie lorsqu'on décode le mot i, $p_{e,i}$, en combinant toutes ces informations.

$$p_{e,i} = \mathrm{Est_e} \otimes \left(\bigotimes_{j \neq i} \widetilde{p}_{e,j} \right).$$

3.3.3 Algorithme de décodage

Nous avons maintenant décrit tout ce dont nous avions besoin pour présenter un algorithme de décodage itératif fondé sur des équations de parité pondérées. On utilisera

naturellement les mêmes notations que lors de la présentation de l'algorithme de Gallager, voir le tableau 3.1 page 68. En particulier, $\mathrm{Obs}(c_t)$ est la probabilité que c_t vaille 1 à partir de l'observation du bit reçu, $\mathrm{Extr}_{\mathrm{e}}(c_t)$ est la probabilité calculée en utilisant l'équation e, $\mathrm{A}_{\mathrm{e}}(c_t)$ est la probabilité a posteriori partielle que c_t vaille 1 calculée à partir de toutes les informations dont on dispose sur c_t exceptée celle qui nous vient de l'équation e. $\mathrm{Est}_{\mathrm{e}}$ est l'estimation de la probabilité que l'équation e soit vérifiée ; $\widetilde{p}_{e,i}$ est la probabilité que l'équation e soit vérifiée en utilisant le mot $\widetilde{m}_i$, $p_{e,i}$ est la probabilité que l'équation e soit vraie en utilisant toutes les informations dont on dispose sur cette équation exceptée celle venant du mot $\widetilde{m}_i$ que l'on est en train de décoder et enfin p_e est la probabilité a posteriori que l'équation e soit vérifiée.

Algorithme.
Entrée : des mots de code bruités de la forme $(c_1^{(i)},\ldots,c_n^{(i)})$.
Sortie : Ces mots de code décodés.

- **Initialisation :**
  Pour toutes les équations e, calculer $\mathrm{Est_e}$ avec (3.6).
  Pour tous les mots $\widetilde{m}_i$,
  $\widetilde{p}_{e,i} = \frac{1}{2}$.
  Pour tout $t \in \{1,\ldots,n\}$,
  - Calculer $\mathrm{Obs}(c_t^{(i)}) = \begin{cases} 1-\tau & \text{si l'observation est un 1.} \\ \tau & \text{si l'observation est un 0.} \end{cases}$
  - $\mathrm{Extr_e}(c_t^{(i)}) = \frac{1}{2}$.
- **Itérer :** Pour tous les mots $\widetilde{m}_i$ et pour toute équation e,
  - Calculer :

$$\forall t \in \{1,\ldots,n\},\ \mathrm{A_e}(c_t^{(i)}) = \mathrm{Obs}(c_t^{(i)}) \otimes \left[\bigotimes_{j\neq e} \mathrm{Extr_j}(c_t^{(i)})\right].$$

  - Calculer

$$\begin{aligned} \widetilde{p}_{e,i} &= \frac{1+\prod_{t\in I_e}(1-2\mathrm{A_e}(c_t^{(i)}))}{2}. \\ p_{e,i} &= \mathrm{Est_e} \otimes \left[\bigotimes_{j\neq i} \widetilde{p}_{e,j}\right]. \end{aligned}$$

  - Calculer :

$$\forall t \in \{1,\ldots,n\},\ \mathrm{Extr_e}(c_t^{(i)}) = \left(\frac{1}{2} - p_{e,i}\right) \prod_{j\in I_e\setminus\{t\}} (1-2\mathrm{A_e}(c_j^{(i)})) + \frac{1}{2}.$$

- **Terminaison :**
  - Pour toute équation e, calculer :

$$p_e = \mathrm{Est_e} \otimes \left[\bigotimes_{i} \widetilde{p}_{e,i}\right].$$

  - Pour tout mot $\widetilde{m}_i$ et pour tout $t \in \{1,\ldots,n\}$, calculer

$$\mathrm{APP}(c_t^{(i)}) = \mathrm{Obs}(c_t^{(i)}) \otimes \left[\bigotimes_{j} \mathrm{Extr_j}(c_t^{(i)})\right].$$

  - Si $\mathrm{APP}(c_t^{(i)}) > \frac{1}{2}$ alors $c_t^{(i)} = 1$, sinon $c_t^{(i)} = 0$

TAB. 3.4 – *Algorithme de décodage itératif fondé sur des équations de parité pondérées (première tentative).*

La complexité en mémoire de cet algorithme est de

$$M\left(n + 3\mathrm{nb}_{\mathrm{eq}} + n\mathrm{nb}_{\mathrm{eq}}\right) = \mathcal{O}\left(nM\mathrm{nb}_{\mathrm{eq}}\right).$$

Dans la phase d'itération, on fait $2M\mathrm{nb}_{\mathrm{eq}}$ calculs de loi $\otimes$ et autant de produits. Et dans la terminaison, on fait $2M\mathrm{nb}_{\mathrm{eq}}$ calculs de loi $\otimes$. La complexité générale est donc de

$$2M\mathrm{nb}_{\mathrm{eq}}(1 + \mathrm{nb}_{\mathrm{iter}})$$

calculs de la loi $\otimes$ et de $2M\mathrm{nb}_{\mathrm{eq}}$ produits.
On a donc une complexité en loi $\otimes$ de l'ordre de

$$\mathcal{O}\left(\mathrm{nb}_{\mathrm{iter}}M\mathrm{nb}_{\mathrm{eq}}\right).$$

La détermination des conditions de convergence de cet algorithme semble être un problème complexe que nous n'avons donc pas été traité de manière théorique. Par contre, nous avons effectué de nombreuses simulations. Il est rapidement apparu qu'il était gênant d'avoir des équations de parité qui faisaient apparaître des cycles courts dans le graphe de Tanner associé au code considéré. En effet lorsqu'on a effectué des tests de décodage en mettant deux fois certaines équations, alors l'algorithme ne convergeait plus. C'est pour cela qu'avant d'utiliser cet algorithme de décodage, nous prendrons soin d'éliminer les équations de parité qui font apparaître des cycles de petite longueur (en pratique en éliminera tous les mots qui font intervenir des cycles de longueur 4 ou moins) dans le graphe associé au code. Pour ce faire, nous construirons les arêtes du graphe en ajoutant les équations une à une et si l'ajout d'une équation fait apparaître un cycle de longueur 4 ou moins, nous choisirons d'enlever cette équation.

3.3.4 Simulations du processus de reconstruction

On a donc construit tous les blocs nécessaires pour notre nouvel algorithme de reconnaissance de code. Il fonctionnera de la manière suivante :

Algorithme.

- Itérer :
  - Chercher des mots de poids faible dans le code engendré par R^T (avec M correctement choisi). Retourner les mots h qui seront des candidats pour être des mots du code dual.
  - Supprimer les équations de parité qui font apparaître des cycles de petite longueur (en pratique on éliminera tous les mots qui font intervenir des cycles de longueur 4 ou moins). Pour les autres h, calculer une estimation de la probabilité qu'ils appartiennent au code $\mathcal{C}^{\perp}$.
  - Exploiter ces équations de parité et leurs probabilités pour corriger des erreurs en utilisant l'algorithme présenté à la table 3.4.
- Calculer alors la probabilité que chaque mot appartienne au code dual et retenir les $n-k$ mots linéairement indépendants de probabilité maximum s'il en existe suffisamment de probabilité supérieure à $\frac{1}{2}$.

TAB. 3.5 – *Algorithme de reconstruction d'un code en bloc utilisant l'algorithme de décodage de la table 3.4.*

Nous avons effectué des simulations pour cet algorithme, tout d'abord en nous intéressant seulement à l'algorithme interne à la phase d'itération, c'est-à-dire que l'on effectue des simulations sur l'algorithme suivant :

Algorithme.

- Chercher des mots de poids faible dans le code engendré par R^T (avec M correctement choisi). Retourner les mots h qui seront des candidats pour être des mots du code dual.
- Supprimer les équations de parité qui font apparaître des cycles de courte longueur (en pratique on éliminera tous les mots qui font intervenir des cycles de longueur 4 ou moins). Pour les autres h, calculer une estimation de la probabilité qu'ils appartiennent au code $\mathcal{C}^{\perp}$.
- Utiliser ces équations de parité et leurs probabilités pour corriger des erreurs en utilisant l'algorithme présenté à la table 3.4.

TAB. 3.6 – *Algorithme sur lequel nous allons faire des simulations.*

Reconstruction d'un code LDPC (3,6)

Puisque notre algorithme repose sur l'existence de mots de poids faible dans le dual du code utilisé lors de la transmission, un premier exemple naturel est la reconstruction d'un code LDPC, *i.e.* on suppose que le code utilisé lors de la transmission est un code LDPC. Ici on considère un code LDPC régulier (3,6), c'est-à-dire que chaque colonne de la matrice de parité du code a exactement trois «1» et que chaque ligne a six «1». Ce code est donc un code de rendement $\frac{1}{2}$. On doit d'abord calculer $\Pr[h \in \mathcal{C}^{\perp}]$. Ce calcul nécessite une approximation de la distribution des poids du code dual au moins pour les poids faibles. On fait ici quelques approximations brutales sur le nombre de mots de poids plus petit que 24 dans le code dual en utilisant le fait qu'il y a environ k mots de poids 6. Par exemple, la plupart des mots de poids 10 sont obtenus par somme de deux mots de poids 6 qui ont seulement une position en commun, cette approximation nous a amener à approximer le nombre de mots de poids 10 par $2k$ puisque chaque position intervient dans exactement trois équations. On a fait le même type de raisonnement pour les autres poids. On est alors capable d'approximer la probabilité

$$\Pr[h \in \mathcal{C}^{\perp}] = \frac{\#\{a \in \mathcal{C}^{\perp} | wt(a) = wt(h)\}}{\binom{n}{wt(h)}}.$$

Donc, on sait aussi approximer $\Pr[h \in \mathcal{C}^{\perp} | wt(hX^T) = w]$ pour tout h de poids plus petit que 24. En pratique, on ne trouve que très rarement des mots de poids plus grand que 24 et on a donc choisi de ne pas les prendre en compte. Pour les simulations, on a séparé l'algorithme pour trouver des mots de poids faible (algo CC) de l'algorithme de décodage dans lequel on a choisi après quelques simulations de faire 30 itérations. On itère l'algorithme CC un temps donné. On rappelle que τ est la probabilité d'erreur du canal binaire symétrique et on va noter p_{err} le taux d'erreur par bit après décodage avec notre algorithme. Pour un code LDPC (3,6) de longueur 100, les simulations sur un Pentium 4 à 3.2GHz donnent les résultats présentés à la table 3.7.

τ	temps de l'algo CC	p_{err}	temps de décodage
0.01	1 s	0	1 s
0.02	1 s	0.003	0.7 s
0.02	5 s	0.00017	0.8 s
0.02	10 s	0	0.8 s
0.03	10 s	0.027	0.5 s
0.03	30 s	0.022	0.74 s
0.03	1 mn	0.011	1.12 s
0.03	5 mn	0.002	1.43 s
0.03	10 mn	0.0018	1.39 s
0.03	1 h	0.0012	1.37 s

TAB. 3.7 – *Performance de l'algorithme de la table 3.6 pour un code LDPC (3,6) de longueur 100.*

On peut remarquer que, si nous devons retrouver notre code lorsque la probabilité d'erreur du canal est 0.03, on peut faire mieux qu'utiliser notre algorithme pour trouver des mots de poids faible pendant 20 minutes puis décoder. En effet, on peut utiliser l'algorithme CC pendant 1 minute puis décoder. On a alors réduit notre taux d'erreur à 0.01 et on peut maintenant réutiliser notre algorithme pour trouver des mots de poids faible dans le code engendré par une nouvelle matrice R^T construite à partir des mots corrigés par la phase précédente. Avec une telle probabilité d'erreur, une seule seconde suffit pour trouver des mots de poids faible et corriger toutes les erreurs. En pratique, il faut un peu plus de temps puisque la plupart des mots de poids faible «faciles» à trouver seront linéairement dépendants de ceux utilisés lors de la première phase de décodage.

Regardons maintenant ce qui se passe dans le cas plus réaliste d'un code LDPC de longueur 1000.

τ	temps de l'algo CC	p_{err}	temps de décodage
0.001	1 mn	0.0001	53 s
0.001	2 mn	0.000006	56 s
0.001	3 mn	0.000008	55 s
0.001	4 mn	0	57 s
0.001	5 mn	0	57 s
0.002	1 mn	0.0019	12 s
0.002	5 mn	0.0017	16 s
0.002	10 mn	0.0014	25 s
0.002	20 mn	0.0011	30 s
0.002	30 mn	0.0007	36 s
0.002	1 h	0.0005	43 s

TAB. 3.8 – *Performance de l'algorithme de la Table 3.6 pour un code LDPC (3,6) de longueur 1000.*

Les résultats montrent, que pour un code de longueur 1000, on réussit à corriger la plupart des erreurs seulement si la probabilité d'erreur du canal est très faible ce qui semble limiter considérablement la portée de l'algorithme. Cependant, en pratique, la démodulation fournit une information souple et dès que la quantité de données est suffisante, on pourra choisir les mots de code les plus sûrs. Dans ce cas, des taux d'erreur de l'ordre de 0.001 ne sont pas aberrants.

Reconstruction d'un code aléatoire de longueur 100 et de dimension 50

Nous venons de voir que notre algorithme est efficace dans le cas d'un code LDPC régulier (3,6) et maintenant nous décrivons ce qu'il se passe pour le cas général d'un code aléatoire. Comme nous avons pris pour le code $\mathcal{C}$ un code aléatoire, $\mathcal{C}^{\perp}$ est aussi un code aléatoire ; on sait donc que sa distribution des poids suit une loi binomiale. Nous avons donc :

$$\Pr[h \in \mathcal{C}^{\perp}] = \frac{1}{2^k}.$$

On peut alors approximer $\Pr[h \in \mathcal{C}^{\perp} | wt(hX^T) = w]$ pour tout h. Pour un code aléatoire de longueur 100 et de dimension 50, les simulations sur un Pentium 4 à 3.2GHz donnent les résultats suivants.

τ	temps de l'algo CC	p_{err}	temps de décodage
0.005	10 s	0.003	1.22 s
0.005	30 s	0.003	1.18 s
0.005	1 mn	0.003	1.20 s
0.01	10 s	0.006	1.18 s
0.01	30 s	0.006	1.19 s
0.01	1 mn	0.006	1.18 s
0.01	10 mn	0.006	1.19 s
0.02	30 s	0.02	0 s
0.02	5 mn	0.02	0 s
0.02	1 h	0.02	0 s

TAB. 3.9 – *Performance de l'algorithme de la Table 3.6 pour un code aléatoire de longueur 100 et de dimension 50.*

Pour les codes aléatoires, notre algorithme est moins efficace que pour les codes LDPC puisqu'il nous faut plus de temps pour trouver des mots de poids faible et pour décoder. Il sera donc plus difficile de retrouver le code utilisé. Cependant, on sait que les codes aléatoires correspondent généralement au cas le plus défavorable en décodage, et qu'il s'agit naturellement du cas le plus éloigné de notre situation car en pratique, le code utilisé lors de la transmission est structuré puisqu'il a pour but de corriger les erreurs de transmission. De plus, on arrive tout de même à diminuer le taux d'erreur en quelques minutes lorsque la probabilité d'erreur est très faible.

3.4 Un nouvel algorithme de reconstruction

3.4.1 Problème rencontré...

Toutefois, nous avons également voulu regarder ce qu'il se passait lorsque l'on ajoutait artificiellement des équations de parité fausses aux mots retournés par l'algorithme de Canteaut-Chabaud. On a donc ajouté une équation fausse à la liste de nos équations et lancé notre algorithme de décodage. Malheureusement il ne décodait plus et, au contraire, rajoutait des erreurs, ce qui nous a clairement indiqué que cet algorithme ne faisait pas ce que nous voulions. La question qui se posait était donc d'identifier l'origine de ce phénomène et le corriger. Le problème provient en fait de la remise à jour des probabilités des équations de parité. On remet à jour avec la formule suivante :

$$p_{e,i} = \mathrm{Est_e} \otimes \left[\bigotimes_{j \neq i} \widetilde{p}_{e,j} \right].$$

Pour comprendre pourquoi cette formule est à l'origine du dysfonctionnement que nous avons décrit, il faut rappeler deux points importants. Tout d'abord, la probabilité $\widetilde{p}_{e,j}$ est

la probabilité que la relation de parité associée à l'équation e soit vraie pour le mot $\widetilde{m_j}$. Si c'est une équation de parité du code alors cette probabilité sera proche de 1 pour tout j ; mais si ce n'est pas une équation de parité du code, elle sera quand même vérifiée pour environ un mot sur deux. Par ailleurs, la mise à jour de ces probabilités sur les équations de parité est effectuée en utilisant la loi $\otimes$. Or, le résultat de cette mise à jour conduit à affecter à $p_{e,i}$ une valeur proche de 1 (resp. de 0) dès que $\widetilde{p}_{e,j}$ est proche de 1 (resp. de 0) pour un peu plus de la moitié des valeurs de j. En effet, lorsque l'on combine avec la loi $\otimes$ une probabilité proche de 1 avec une probabilité proche de zéro, on a un résultat proche de $\frac{1}{2}$ et la valeur $\frac{1}{2}$ est un élément neutre pour la loi $\otimes$. Par conséquent, si on a un peu plus de la moitié des mots qui vérifient une équation, le résultat va être proche de 1 alors que si cette équation n'est vraie que pour un tout petit peu plus de la moitié des mots, il y a de grandes chances que ce ne soit pas une équation de parité du code. En effet, on souhaite que si l'équation est vraie pour environ la moitié des mots, elle n'influence pas la remise à jour des bits, autrement dit que la valeur de $\mathrm{Extr}_{\mathrm{e}}(c_t)$ soit proche de $\frac{1}{2}$ pour tout t, ce qui correspond à une valeur de $p_{e,i}$ proche de $\frac{1}{2}$.

3.4.2 Nouvel algorithme de décodage

Nous avons donc corriger l'erreur que nous venons de décrire en changeant simplement dans l'algorithme 3.4 la formule de remise à jour de $p_{e,i}$. Vu que l'on souhaite que cette probabilité soit proche de $\frac{1}{2}$ lorsque l'équation est vérifiée pour environ la moitié des mots et qu'elle soit proche de 1 lorsque l'équation est vérifiée pour la grande majorité des mots, on a décidé de faire la moyenne des $\widetilde{p}_{e,j}$. De ce fait, le rôle joué par l'estimation $\mathrm{Est}_{\mathrm{e}}$ est maintenant faible au point que nous allons supprimer cette probabilité. Avec cette modification, notre algorithme devient celui décrit dans la table 3.10.

Algorithme.
Entrée : M mots de code bruités de la forme $(c_1^{(i)},\ldots,c_n^{(i)})$.
Sortie : Ces mots de code décodés.

- **Initialisation :**
  Pour tous les mots $\widetilde{m}_i$, pour tout $t \in \{1,\ldots,n\}$,
  - Calculer $\mathrm{Obs}(c_t^{(i)}) = \begin{cases} 1-\tau & \text{si l'observation est un 1.} \\ \tau & \text{si l'observation est un 0.} \end{cases}$
  - $\forall e$, $\mathrm{Extr_e}(c_t^{(i)}) = \frac{1}{2}$.

  Pour toute équation e : $\widetilde{p}_{e,i} = \frac{1}{2}$.
- **Itérer :**
  Pour tous les mots $\widetilde{m}_i$, pour toute équation e :
  - Calculer :

$$\forall t \in \{1,\ldots,n\},\ \mathrm{A_e}(c_t^{(i)}) = \mathrm{Obs}(c_t^{(i)}) \otimes \left[\bigotimes_{j\neq e} \mathrm{Extr_j}(c_t^{(i)})\right].$$

  - Calculer :

$$\begin{aligned} \widetilde{p}_{e,i} &= \frac{1+\prod_{t\in I_e}(1-2\mathrm{A_e}(c_t^{(i)}))}{2}\cdot \\ p_{e,i} &= \frac{\sum_{j\neq i}\widetilde{p}_{e,j}}{M-1}. \end{aligned}$$

  - Calculer :

$$\forall t \in \{1,\ldots,n\},\ \mathrm{Extr_e}(c_t^{(i)}) = \left(\frac{1}{2} - p_{e,i}\right) \prod_{j\in I_e\setminus\{t\}} (1-2\mathrm{A_e}(c_j^{(i)})) + \frac{1}{2}.$$

- **Terminaison :**
  - Pour toute équation e, calculer :

$$p_e = \frac{\sum_{j=1}^{M}\widetilde{p}_{e,j}}{M}.$$

  - Pour tout mot $\widetilde{m}_i$ et pour tout $t \in \{1,\ldots,n\}$, calculer

$$\mathrm{APP}(c_t^{(i)}) = \mathrm{Obs}(c_t^{(i)}) \otimes \left[\bigotimes_{j} \mathrm{Extr_j}(c_t^{(i)})\right].$$

  - Si $\mathrm{APP}(c_t^{(i)}) > \frac{1}{2}$ alors $c_t^{(i)} = 1$, sinon $c_t^{(i)} = 0$

TAB. 3.10 – *Algorithme de décodage itératif fondé sur des équations de parités pondérées (algorithme corrigé).*

Nous avons ensuite effectué des simulations pour l'algorithme suivant :

Algorithme.

- Chercher des mots de poids faible dans le code engendré par R^T (avec M correctement choisi). Retourner les mots h qui seront des candidats pour être des mots du code dual.
- Supprimer les équations de parité qui font apparaître des cycles de petite longueur (en pratique on éliminera tous les mots qui font intervenir des cycles de longueur 4 ou moins). Pour les autres h, calculer une estimation de la probabilité qu'ils appartiennent au code $\mathcal{C}^{\perp}$.
- Exploiter ces équations de parité et leurs probabilités pour corriger des erreurs en utilisant l'algorithme présenté à la table 3.10.
- Calculer alors la probabilité que chaque mot appartienne au code dual et retenir les $n-k$ mots linéairement indépendants de probabilité maximum s'il en existe suffisamment de probabilité supérieure à $\frac{1}{2}$.

TAB. 3.11 – *Algorithme de reconstruction d'un code en bloc utilisant l'algorithme de décodage de la table 3.10.*

3.5 Simulations

Toutes les simulations présentées dans la suite ont été réalisées sur un Pentium 4 à 3.2GHz.

3.5.1 Reconstruction d'un code BCH [15,7,5]

On reprend ici notre exemple du code BCH [15,7,5]. On commence par prendre $\tau = 0.01$, $wt_H(h) = 12$, $\alpha = 0.01$ et $\beta = 0.1$. On a déjà vu que ces paramètres nous conduisaient à prendre $M = 32$, $T = 5$. On calcule la distribution des poids du code engendré par hR^T et on sélectionne tous les mots h tels que $wt_H(hR^T) \leq 5$, ici nous n'utilisons pas l'algorithme CC, nous nous intéressons simplement à notre algorithme de décodage. Nous obtenons ainsi 255 équations qui sont en fait tous les mots du dual excepté le mot nul (nous avons donc déjà reconstruit notre code dual). La suppression des équations de parité faisant apparaître des cycles de longueur 4 ou moins nous réduit le nombre d'équations à considérer lors du décodage à seulement 9 équations. On décode alors et le décodage ne laisse aucune erreur, nous avons donc corrigé toutes les erreurs et le code dual reconstruit

est donné par la matrice suivante :

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

On peut vérifier que cette matrice est bien une matrice de parité du code BCH considéré.

Nous souhaitons maintenant regarder l'impact d'équations fausses sur notre algorithme. Nous nous plaçons donc dans un cas où α est plus grand, nous allons reconsidérer l'exemple où $\alpha = 1$ avec tous les autre paramètres inchangés. Cela conduit, comme nous l'avons déjà vu à prendre $M = 17$, $T = 3$. On obtient alors 479 mots tels que $wt(hR^T) \leq 3$. La suppression des équations de parité faisant apparaître des cycles de longueur 4 ou moins nous réduit le nombre d'équations à considérer lors du décodage à seulement 16 équations dont deux sont fausses. On décode alors et le décodage ne laisse aucune erreur. Après décodage, les 255 équations qui sont des équations de parité ont une valeur de p_e qui vaut 1 alors que pour les 224 autres mots n'appartenant pas au dual, la valeur de p_e est entre 0.70 et 0.76. Notre algorithme a donc fait ressortir les équations de parité du code et le code dual reconstruit est donné par la matrice suivante :

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

On prend désormais $\tau = 0.05$ et $\alpha = 0.1$, les autres paramètres restant inchangés. Cela conduit à choisir $M = 298$ et $T = 117$. Cette fois, nous utilisons l'algorithme CC pour chercher des mots de poids faible dans le code $\mathcal{D}$. On trouve ainsi 248 équations qui sont toutes des équations de parité pour le code BCH de départ. La suppression des équations de parité faisant apparaître des cycles de longueur 4 ou moins nous réduit le nombre d'équations à considérer lors du décodage à seulement 15 équations. Le décodage ne supprime pas toutes les erreurs : après décodage, la probabilité d'erreur par bit est de 0.01. Par contre, on a réussi à reconstruire le dual du code BCH et la matrice de parité

trouvée est :

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

Dans ce cas, nous avons aussi regardé ce qu'il se passait lorsqu'on tentait de décoder avec toutes les équations, sans enlever les cycles de petite taille. Dans ce cas, l'algorithme ne décode pas, la probabilité d'erreur après décodage vaut 0.05 et la fiabilité que l'on a sur les équations est beaucoup plus faible : on a des valeurs de p_e entre 0.5 et 0.7.

Ensuite, nous avons examiné le cas où l'on prend maintenant $\alpha = 1$ et tous les autres paramètres restent inchangés. Cela conduit à choisir $M = 170$ et $T = 69$. L'algorithme CC nous fournit 387 équations dont 132 sont fausses. La suppression des équations de parité faisant apparaître des cycles de longueur 4 ou moins nous réduit le nombre d'équations à considérer lors du décodage à seulement 13 équations. Après décodage, la probabilité d'erreur reste de 0.027. Par contre, nous avons réussi à séparer les vraies équations des fausses. Pour des équations fausses, la valeur de p_e est comprise entre 0.5 et 0.62 alors que pour les vraies équations, cette valeur est comprise entre 0.80 et 0.94. Nous avons encore réussi à reconstruire une matrice de parité du code BCH :

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

3.5.2 Reconstruction d'un code LDPC (3,6)

Revenons à notre test statistique et plus précisément au rapport

$$\alpha = \frac{\sharp\left\{h \notin \mathcal{C}^{\perp} \text{ tels que } wt_H(hR^T) \leq T\right\}}{\sharp\left\{h \in \mathcal{C}^{\perp} \text{ tels que } wt_H(hR^T) \leq T\right\}}$$

que nous avons étudié au chapitre précédent. En fait la quantité qui nous intéresse vraiment est celle-ci mais pour $wt_H(h)$ plus petit qu'un certain seuil. Si la distribution des poids de $\mathcal{C}^{\perp}$ est proche de celle d'un code aléatoire, alors la valeur de α correspond bien à celle donnée page 40. Cependant pour les codes LDPC, on sait que le code $\mathcal{C}^{\perp}$ contient beaucoup de mots de poids faible, ce qui n'est pas le cas pour son complémentaire. Autrement dit, la

valeur réelle du rapport $\frac{\sharp\left\{h\notin\mathcal{C}^\perp \text{ tels que } wt_H(h)\leq w\right\}}{\sharp\left\{h\in\mathcal{C}^\perp \text{ tels que } wt_H(h)\leq w\right\}}$ est beaucoup plus petite que 2^k-1 quand w est petit. La valeur réelle de α va donc être beaucoup plus faible qu'en théorie et on va donc pouvoir se permettre pour ces des valeurs de α très grande.

Commençons tout de même par prendre une valeur faible de α pour bien séparer les deux distributions. Pour un code de longueur $n = 100$, on prend $wt_H(h) = 25$, pour $\tau = 0.01$ cela conduit à prendre $M = 223$ et $T = 51$. Pour $\tau = 0.02$, on a $M = 650$ et $T = 223$; pour $\tau = 0.03$, cela conduit à $M = 1887$ et $T = 769$. Notons que les mots retrouvés sont dans tous les cas des mots appartenant à $\mathcal{C}^\perp$.

τ	temps de l'algo CC	p_{err}	Dimension du code dual reconstruit
0.01	1 s	0	50
0.02	1 s	0.02	2
0.02	5 s	0.013	35
0.02	10 s	0.001	46
0.02	30 s	0	50
0.03	10 s	0.03	9
0.03	30 s	0.03	15
0.03	1 mn	0.023	37
0.03	5 mn	0.008	47
0.03	6 mn	0.0009	50

TAB. 3.12 – *Performance de l'algorithme de la table 3.11 pour un code LDPC (3,6) de longueur 100.*

Dans le cas d'un code de longueur 100, on voit donc que notre algorithme est très efficace puisqu'on arrive à reconstruire le code en quelques minutes pour des taux d'erreurs allant jusqu'à 0.03.

Prenons maintenant un code de longueur 250, on prend $wt_H(h) = 63$, $\alpha = 0.01$, $\beta = 0.1$. Pour $\tau = 0.05$, cela conduit à prendre $M = 703$ et $T = 179$.

τ	temps de l'algo CC	p_{err}	Dimension du code dual reconstruit
0.005	10 s	0.003	63
0.005	30 s	0.003	74
0.005	1 mn	0.001	105
0.005	5 mn	0.0003	115
0.005	10 mn	0.0001	119
0.005	30 mn	0	125

TAB. 3.13 – *Performance de l'algorithme de la table 3.11 pour un code LDPC (3,6) de longueur 250.*

En fait, pour un code LDPC, il y a de nombreux mots de $\mathcal{C}^{\perp}$ de poids très faible donc on peut prendre un seuil T plus faible pour espérer détecter ces mots (dans notre cas de poids 6) plus facilement. Nous avons donc fait un test pour $M = 518$ et $T = 30$, les résultats sont résumés à la table 3.14. La valeur du seuil a été déterminée en calculant l'espérance du poids de hR^T pour un mot h de poids 12 : $\frac{M}{2}\left(1 + (1-2\tau)^{12}\right)$.

τ	temps de l'algo CC	p_{err}	Dimension du code dual reconstruit
0.005	30 s	0.0008	103
0.005	2 mn	0	119
0.005	3 mn	0	121
0.005	10 mn	0	125

TAB. 3.14 – *Performance de l'algorithme de la table 3.11 pour un code LDPC (3,6) de longueur 250.*

On s'est ensuite demandé ce qu'il se passait si on diminuait encore la valeur du seuil T. On a donc refait des simulations avec $M = 518$ et $T = 15$. Les résultats se trouvent à la table 3.15. La valeur $T = 15$ correspond cette fois à l'espérance du poids de hR^T pour un mot de poids 6 : $\frac{M}{2}\left(1 + (1-2\tau)^{6}\right)$.

τ	temps de l'algo CC	p_{err}	Dimension du code dual reconstruit
0.005	1 mn	0.004	45
0.005	5 mn	0.002	79
0.005	10 mn	0.002	63

TAB. 3.15 – *Performance de l'algorithme de la table 3.11 pour un code LDPC (3,6) de longueur 250.*

On voit donc ici que lorsque l'on diminue trop la valeur du seuil, il devient difficile de retrouver des mots par l'algorithme CC ; tous les mots retournés dans ce cas étaient d'ailleurs des mots h de poids 6, les plus faciles à retrouver.

Regardons maintenant ce qu'il se passe pour un code de longueur 1000. Pour $\tau = 0.001$, on a $M = 1986$ et $T = 413$, et pour $\tau = 0.002$, on a $M = 5485$ pour $wt_H(h) = 250$, cette valeur étant beaucoup trop élevée pour espérer une bonne performance de l'algorithme CC, nous avons choisi de nous limiter à $wt_H(h) = 100$. Dans ce cas, on a $M = 1619$ et $T = 286$.

τ	temps de l'algo CC	p_{err}	Dimension du code dual reconstruit
0.001	1 mn	0.0004	315
0.001	2 mn	0.0002	383
0.001	5 mn	0.00008	441
0.001	10 mn	0.000021	475
0.001	30 mn	0	491
0.001	1h	0	500
0.002	10 mn	0.002	4
0.002	20 mn	0.002	10
0.002	1h	0.002	12
0.002	12h	0.0019	48

TAB. 3.16 – *Performance de l'algorithme de la table 3.11 pour un code LDPC (3,6) de longueur 1000.*

On voit sur cette dernière table que l'on réussi à reconstruire notre code simplement lorsque le taux d'erreur est très faible : $\tau = 0.001$. Les simulations effectuées sur les LDPC ont permis de montrer que pour ce type de code les valeurs de M et le T données par le théorème 2.7 ne sont certainement pas optimales. De plus, comme il semble que la séparation entre les mots du code dual et des mots aléatoires soit très marquée, l'algorithme CC nous fournira des équations de parité toutes exactes et notre algorithme de décodage ne permettra pas forcément d'améliorer les performances d'un algorithme qui se contenterait de chercher une base des mots retournés par l'algorithme CC. Cependant, on voit que l'on arrive souvent à décoder toutes les erreurs même lorsque l'on ne dispose pas du code dual en entier et dans ce cas il peut donc être plus rapide de décoder tous les mots de code pour reconstruire une matrice génératrice du code que de reconstruire une matrice de parité du code, voir par exemple la table 3.14.

3.5.3 Reconstruction d'un code aléatoire de longueur 100 et de dimension 50

Nous examinons maintenant le comportement de notre algorithme pour un code aléatoire de longueur 100 et de dimension 50. Nous prenons dans toutes les simulations qui suivent $wt_H(h) = 25$ et $\beta = 0.1$. Nous commençons par regarder le cas où $\alpha = 0.01$. Cela nous amène à choisir $M = 158$, $T = 22$ pour $\tau = 0.005$, $M = 274$, $T = 62$ pour $\tau = 0.01$ et $M = 797$, $T = 271$ pour $\tau = 0.02$. Les résultats sont résumés dans le tableau 3.17, il faut noter que dans tous les cas toutes les équations retournées par l'algorithme CC sont effectivement des équations de parité pour le code utilisé lors de la transmission.

τ	temps de l'algo CC	p_{err}	Dimension du code dual reconstruit
0.005	1 s	0.0047	50
0.01	1 s	0.008	50
0.02	30 s	-	0
0.02	5 mn	0.02	28
0.02	10 mn	0.019	50

TAB. 3.17 – *Performance de l'algorithme de la table 3.11 pour un code aléatoire de longueur 100 et de rendement* $\frac{1}{2}$.

Nous avons ensuite regardé ce qu'il se passe lorsqu'on augmente la valeur de α : on a pris $\alpha = 1$. Les simulations ont montré qu'on ne trouvait encore quasiment que des équations qui étaient vraiment des équations de parité du dual, excepté pour $\tau = 0.005$, où nous avons trouvé une équation fausse parmi les 178, mais celle-ci avait une faible valeur de $p_{e,i}$ par rapport aux autres équations (0.8 contre environ 0.95 pour toutes les autres) et n'a donc pas été retenue lors de la reconstruction d'une matrice du code dual. Cette faible valeur s'explique car notre α correspondrait au rapport $\frac{\sharp\left\{h\notin\mathcal{C}^\perp \text{ tels que } wt_H(hR^T)\leq T\right\}}{\sharp\left\{h\in\mathcal{C}^\perp \text{ tels que } wt_H(hR^T)\leq T\right\}}$ mais pour les mots h de poids 25 (la plupart des mots retournés sont de poids plus petit que 25). Or, comme le montre la courbe 2.2 page 38 la valeur de α augmente avec le poids de Hamming de h et donc pour des mots de poids inférieur la valeur de α est beaucoup plus faible. On a aussi noté que le mot qui n'appartenait pas au code dual était d'ailleurs de poids 25. Les résultats des simulations se trouvent dans le tableau 3.18. Pour $\tau = 0.005$, on a $M = 127$, $T = 18$; pour $\tau = 0.1$, $M = 222$, $T = 51$ et pour $\tau = 0.02$, on a $M = 648$ et $T = 222$

τ	temps de l'algo CC	p_{err}	Dimension du code dual reconstruit
0.005	1 s	0.0044	50
0.01	1 s	0.006	50
0.02	30 s	0.02	2
0.02	5 mn	0.02	4
0.02	10 mn	0.02	12
0.02	30 mn	0.02	27
0.02	1 h	0.02	45
0.02	2 h	0.02	50

TAB. 3.18 – *Performance de l'algorithme de la table 3.11 pour un code aléatoire de longueur 100 et de rendement* $\frac{1}{2}$.

On ne sait pas totalement expliquer pourquoi dans ce cas il nous faut plus de temps pour reconstruire le code dual du code aléatoire dans le cas $\tau = 0.02$ que pour les simulations effectuées avec $\alpha = 0.01$. Pourtant la valeur de M est plus faible. Un élément de réponse est qu'il ne s'agit pas du même code aléatoire mais ce n'est sans doute pas la seule explication.

Il est important de noter que même lorsque la probabilité d'erreur τ est relativement élevée (0.02), on ne réussit généralement pas à corriger toutes les erreurs, mais cependant nous arrivons à reconstruire le code assez facilement. Enfin, il faudrait poursuivre les simulations sur des codes de plus grande longueur pour mesurer l'impact de notre algorithme de décodage puisque pour les longueurs considérées nous arrivons souvent à reconstruire le code en utilisant simplement l'algorithme CC, autrement dit toutes les équations retrouvées sont exactes. Mais l'exemple du code BCH pour $M = 117$ et $T = 69$ montre bien que notre algorithme de décodage nous permet de différencier les bonnes des fausses équations et qu'il peut donc s'avérer très utile si l'on doit reconstruire un code de longueur élevée. Si pour une valeur très faible de α, chercher des mots de poids faible dans le code $\mathcal{D}$ s'avère impossible à cause de sa longueur M trop élevée, nous sommes alors obligés d'augmenter la valeur de α pour diminuer la valeur de M. Il y a alors de fortes chances pour que nous ayons des équations fausses et c'est dans ce cas que notre algorithme est très utile.

Deuxième partie

Reconnaissance d'un brasseur linéaire

Chapitre 4

Présentation du problème

4.1 Reconstruction d'un brasseur

Nous nous intéresserons ici aux brasseurs et plus particulièrement aux brasseurs linéaires. Rappelons que l'on distingue deux types de brasseurs linéaires, les brasseurs synchrones et les brasseurs auto-synchronisants, en fonction de la technique utilisée pour combiner le train binaire de départ et la suite pseudo-aléatoire générée par un LFSR (c.f. figure 1.6 page 11).

Dans un système de communication, le rôle du brasseur est de diminuer la longueur des paquets de bits identiques qui pourraient occasionner des problèmes de synchronisation lors de la phase de codage. On a vu que lorsque le polynôme de rétroaction du LFSR est primitif alors la suite pseudo-aléatoire générée par le LFSR a de bonnes propriétés statistiques, c'est pourquoi les brasseurs synchrones utilisés en pratique ont des polynômes de rétroaction qui sont primitifs. Pour le cas auto-synchronisant, on n'a pas forcément besoin d'un polynôme de rétroaction primitif puisque les bits de la suite d'entrée $(x_t)_{t\geq 0}$ se propagent dans la suite de sortie $(y_t)_{t\geq 0}$. Afin d'assurer une opération synchrone lors de la transmission et de la réception avec un brasseur synchrone, une trame de synchronisation doit être utilisée. Une trame de synchronisation est un mot donné qui est placé dans la suite à transmettre à intervalles réguliers. Celui qui reçoit la suite cherche alors les trames adjacentes et peut alors déterminer où le LFSR doit être réinitialisé avec un état initial prédéfini. Le débrasseur synchrone est simplement le même que le brasseur synchrone. Un brasseur/débrasseur synchrone est défini par le polynôme de rétroaction du LFSR et son état initial.

Contrairement aux brasseurs synchrones, les brasseurs auto-synchronisants ne nécessitent pas l'envoi de trames de synchronisation. Malheureusement, les brasseurs auto-synchronisants ont des inconvénients significatifs : leurs suites de sortie ne sont pas aussi aléatoires que celles des brasseurs synchrones, et ils propagent les erreurs comme nous l'avons expliqué au chapitre 1 (*i.e.* une erreur à l'entrée du débrasseur entraînera plusieurs erreurs sur la sortie). Un brasseur/débrasseur auto-synchronisant est lui entièrement déterminé par le polynôme de rétroaction du LFSR.

Dans cette partie, notre but est de retrouver quel brasseur a été utilisé lors de la communication. Nous avons vu dans la partie précédente que lorsqu'on est capable de retrouver le code en bloc qui a été employé, on sait retrouver l'espace vectoriel caractérisant notre code mais nous ne pourrons jamais connaître l'encodeur utilisé. A ce stade de notre reconnaissance du schéma de transmission, deux possibilités s'offrent à nous :

- nous pouvons supposer que l'encodeur utilisé est l'encodeur systématique du code en bloc ou bien que, pour le code utilisé, il existe un encodeur naturel (pas forcément systématique) ce qui implique que l'on connaît désormais la sortie du brasseur ;
- sinon, sans faire aucune hypothèse sur l'encodeur utilisé, on ne connaît plus la sortie exacte du brasseur mais seulement l'image de cette suite binaire par une transformation linéaire par bloc φ dont la taille correspond à la dimension du code correcteur utilisé. Cela signifie que l'on souhaite dans ce cas retrouver quel brasseur a été utilisé à partir de la connaissance de la suite de sortie $(\varphi(Y_t))_{t\geq 0}$.

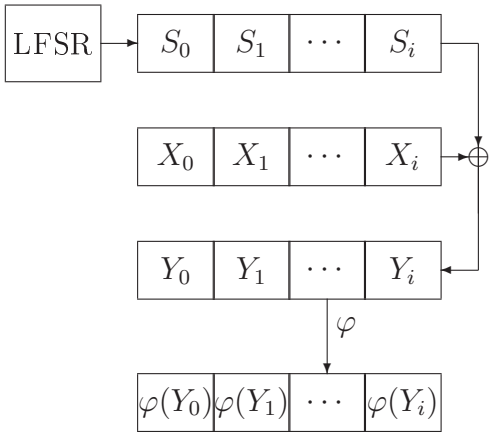

FIG. 4.1 – *Action de la transformation linéaire sur la suite de sortie du brasseur pour une entrée $(x_t)_{t\geq 0}$ donnée.*

Que signifie exactement retrouver le brasseur utilisé ? On a vu que dans les deux cas, synchrone et auto-synchronisant, l'élément indéterminé du brasseur est le LFSR. Donc, retrouver le brasseur utilisé signifie déterminer quel LFSR a été employé, c'est-à-dire déterminer son polynôme de rétroaction et, dans le cas d'un brasseur synchrone, son état initial.

4.2 Hypothèses sur la suite entrante

Nous allons maintenant voir qu'il est indispensable de faire des hypothèses sur la suite binaire entrant dans le brasseur, même dans le cas où l'on connaît sa sortie exacte. En effet, tout brasseur est une bijection, c'est-à-dire que n'importe quelle suite binaire est susceptible d'être obtenue en sortie d'un brasseur donné.

Théorème 4.1 *Toute suite binaire est engendrée par une infinité de brasseurs linéaires (synchrones ou auto-synchronisants).*

Preuve.
Soit $(x_t)_{t\geq 0}$ la suite considérée. Cette suite est trivialement produite par tout brasseur synchrone à partir de l'entrée $(x_t \oplus s_t)_{t\geq 0}$ où $(s_t)_{t\geq 0}$ est la suite produite par le LFSR. Pour le cas du brasseur auto-synchronisant, c'est la même chose, mais il ne faut plus raisonner en matière de suite binaire mais bit à bit. Supposons que nous avons un brasseur auto-synchronisant de polynôme de rétroaction donné. Pour chaque bit de sortie, on peut alors calculer le bit de rétroaction correspondant et on reconstruit ainsi bit à bit une suite d'entrée qui donnera la sortie voulue. ◇

Donc, pour identifier un brasseur à partir de la connaissance de certains bits de sa sortie, il est indispensable de faire des hypothèses sur son entrée. On va donc devoir faire des hypothèses sur la suite binaire d'entrée pour pouvoir reconstruire le brasseur qui a été utilisé lors de la transmission. Nous allons notamment tenter d'apporter différentes méthodes de reconstruction en fonction du type d'hypothèses envisagées sur l'entrée du brasseur. Nous avons considéré trois hypothèses différentes que nous allons présenter ici, de la moins à la plus restrictive. Il s'agit d'hypothèses similaires à celles que l'on fait en cryptanalyse (attaques à chiffré seul, à clair connu, à clair choisi ...).

- L'hypothèse qui semble la plus raisonnable est l'existence d'un biais dans le message transmis. En effet, cette supposition se justifie puisque tous les schémas de codage utilisés en pratique (ASCII, MURRAY, CCITTx, ...) satisfont cette hypothèse. Donc si la suite d'entrée est issue d'un schéma de codage classique alors elle présentera un biais statistique.
- Une hypothèse plus forte que nous avons été amené à faire dans certains cas consiste à supposer que l'on connaît une partie du message transmis. Cette hypothèse peut être réaliste dans la mesure où on peut très bien imaginer qu'il existe une séquence de contrôle connue émise au début de chaque transmission. En effet, pour de nombreuses communications, un en-tête est envoyé au début du message. Si le brasseur utilisé est un brasseur synchrone, on a vu qu'on transmettait alors des trames de synchronisation qui peuvent être connues.
- La dernière hypothèse envisagée est encore plus restrictive puisqu'il s'agit de supposer qu'une portion du message en entrée du brasseur est identiquement nulle. Cette hypothèse est légitimée par le fait que la finalité d'un brasseur est de casser de longues plages de bits identiques.

Les deux chapitres suivants vont maintenant décrire des techniques de reconstruction d'un brasseur synchrone ou d'un brasseur auto-synchronisant pour ces trois hypothèses ; d'abord à partir de la connaissance de la sortie du brasseur, puis à partir de l'image de cette suite par une transformation linéaire par bloc de taille connue.

Chapitre 5

Reconstruction d'un brasseur linéaire lorsque l'on connaît la sortie

Dans ce chapitre, on suppose que l'on connaît la sortie exacte d'un brasseur. Dans le contexte de la reconnaissance d'une chaîne de transmission complète, cette hypothèse implique que l'on a précédemment réussi à retrouver non seulement le code utilisé en aval mais aussi l'encodeur qui a servi lors de la transmission. L'objectif est maintenant de déterminer le brasseur qui a été utilisé. On a vu que pour reconstruire le brasseur, on devait nécessairement faire des hypothèses sur le train binaire d'entrée. Nous allons donc présenter différentes techniques qui peuvent être employées selon l'hypothèse envisagée sur l'entrée, d'abord dans le cas d'un brasseur synchrone, puis dans celui d'un brasseur autosynchronisant.

5.1 Brasseur synchrone

Cette section est consacrée à la reconstruction de brasseurs linéaires synchrones. Nous avons vu que l'élément principal d'un brasseur est un registre à décalage à rétroaction linéaire. Dans le cas du brasseur synchrone, la sortie du brasseur est simplement obtenue en additionnant modulo 2 chaque bit de la suite pseudo-aléatoire générée par le registre avec chaque bit du message entrant.

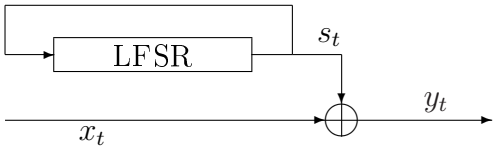

FIG. 5.1 – *Brasseur linéaire synchrone.*

On notera $(x_t)_{t\geq 0}$ la suite binaire en entrée du brasseur (le message), $(s_t)_{t\geq 0}$ le train binaire pseudo-aléatoire générée par le LFSR et $(y_t)_{t\geq 0}$ la suite binaire de sortie. On rappelle

que, dans le cas d'un brasseur synchrone, retrouver le brasseur signifie retrouver le polynôme de rétroaction du registre ainsi que son état initial. Dans toute la suite du manuscrit, on notera L la longueur du brasseur. Celle-ci est généralement inconnue, on supposera toutefois qu'elle ne dépasse pas une certaine valeur fixée $L_{\max}$. En particulier, la valeur $L_{\max} = 100$ permet de couvrir la majorité des applications. On notera P le polynôme de rétroaction du registre à décalage. Ainsi, si $P(X) = 1 + c_1 X + \cdots + c_L X^L$, la suite $(s_t)_{t\geq 0}$ produite par le LFSR vérifie l'équation de récurrence suivante :

$$\forall t,\ s_{t+L} = c_1 s_{t+L-1} + c_2 s_{t+L-2} + \cdots + c_L s_t. \tag{5.1}$$

En pratique, pour les raisons déjà abordées à la section 1.2.1 page 15, P est toujours un polynôme primitif pour assurer de bonnes propriétés statistiques aux suites pseudo-aléatoires générées par le LFSR.

5.1.1 Entrée connue

Dans un premier temps, nous faisons une hypothèse forte : on suppose que l'on connaît une plage de M bits consécutifs du message d'entrée. À partir de ces bits connus et des M bits correspondants de la suite de sortie, on est alors capable de calculer les M bits de la suite $(s_t)_{t\geq 0}$. En effet, on a pour tout bit t :

$$y_t = s_t + x_t.$$

On connaît donc une portion de M bits consécutifs du train binaire pseudo-aléatoire générée par le LFSR. On va alors utiliser l'algorithme de Berlekamp-Massey présenté page 15. Comme l'algorithme requiert la connaissance de $2\Lambda(s) = 2L$ bits de la suite $(s_t)_{t\geq 0}$ (car le polynôme est primitif), le brasseur pourra être reconstruit si on connaît $M \geq 2L$ bits de la suite d'entrée. On peut noter que la complexité de cet algorithme de reconstruction dépend fortement de la longueur exacte du LFSR même si cette valeur n'est pas connue a priori. En effet, la i-ème itération dans l'algorithme de Berlekamp-Massey détermine un LFSR de longueur minimale qui génère les i premiers bits de la suite pseudo-aléatoire. Donc, pour $i \geq 2L$, la i-ème itération vérifie simplement que s_i correspond au i-ème bit généré par le LFSR obtenu à l'étape précédente. La i-ème itération nécessite donc seulement L opérations pour $i \geq 2L$ et environ $3i$ opérations lorsque $i < 2L$. Le nombre total d'opérations nécessaires pour la reconstruction d'un brasseur de longueur L à partir d'une suite de M bits est donc approximativement :

$$3\sum_{i=0}^{2L-1} i + L(M-2L) \simeq 4L^2 + LM,$$

avec $M = 2L_{\max}$.

Dans le cas où les M bits connus ne sont pas consécutifs, on ne peut plus appliquer l'algorithme de Berlekamp-Massey. Toutefois, on peut réussir à reconstruire le polynôme de rétroaction et l'état initial en résolvant un système d'équations linéaires. Mais, dans ce cas, la complexité sera en

$$\mathcal{O}\left(L_{\max}^3\right).$$

5.1.2 Entrée produite par une source biaisée

On suppose désormais que le message d'entrée est inconnu mais qu'il est produit par une source binaire sans mémoire biaisée, c'est-à-dire qu'on a :

$$\Pr[x_t = 0] = \frac{1}{2} + \varepsilon,$$

avec $\varepsilon \neq 0$. On a vu que ce cas apparaissait dans de nombreuses situations pratiques, par exemple lorsque le train binaire d'entrée est produit par un schéma classique de codage comme l'ASCII. Dans ce cas, chacun des 128 caractères sont codés sur un octet et le bit de poids fort vaut donc toujours 0 ce qui fait que la proportion de 0 est en moyenne 1/2+1/16 ; le biais vaut donc 0.0625. Il faut noter qu'en pratique, les 32 premiers caractères du code ne sont pas imprimables et donc très peu utilisés dans des messages usuels, le biais véritable est donc souvent un peu plus faible. Pour les schémas classiques, les valeurs de biais sont de l'ordre de $\varepsilon = 0.1$ ou $\varepsilon = 0.05$.

Principe de l'algorithme de reconstruction du polynôme de rétroaction

L'algorithme pour reconstruire le polynôme de rétroaction du brasseur est assez similaire à la technique présentée dans [CF00] dans un contexte cryptographique qui visait à retrouver les spécifications d'un générateur pseudo-aléatoire par combinaison de LFSRs dans le cadre d'un algorithme de chiffrement à flot. L'idée principale est que la suite de sortie présente des biais statistiques qui vont nous permettre de détecter n'importe quel multiple du polynôme de rétroaction, détection qui sera d'autant plus performante que le multiple sera creux. Dans [CF00], le théorème principal qui fournit la distribution statistique de la quantité utilisée pour cette détection repose sur l'hypothèse que la suite est produite par le LFSR est constituée de variables aléatoires indépendante, hypothèses qui n'est pourtant pas vérifiée. Nous démontrons donc ici un théorème assez similaire sans cette hypothèse erronée.

Lemme 5.1 *Soient $(z_t)_{t\in\mathbb{N}}$ des variables aléatoires telles que z_t et $z_{t'}$ sont indépendantes dès que $|t'-t| > d$ pour un certain entier d fixé. Considérons alors la variable aléatoire $S = \sum_{t=0}^{N} z_t$. On note M sa moyenne et V sa variance. Alors $\frac{S-M}{\sqrt{V}}$ tend vers une loi normale centrée réduite lorsque N tend vers l'infini.*

Preuve.
On suppose dans cette preuve qu'il existe $n \in \mathbb{N}$ tel que $N = (2^n - 1)d - 1$. En effet cela évitera de compliquer les notations et on peut facilement adapter cette preuve si ce n'est pas le cas.
Notons

$$S_i = \{(2^i - 1)d, \ldots, (2^{i+1} - 1)d - 1\}.$$

En décomposant l'ensemble $\{0, \ldots, N\}$ en $\cup_{i=0}^{n-1} S_I$, on peut alors écrire :

$$S = \sum_{i=0}^{n-1} \left(\sum_{t\in S_i} z_t \right).$$

On décompose maintenant chacun des ensembles S_i en deux parties U_i et D_i définies de la manière suivante : $D_0 = \emptyset$ et

$$\forall i > 0,\ D_i = \{(2^i - 1)d, \ldots, 2^i d - 1\} \text{ et } U_i = S_i \setminus D_i.$$

On a alors :

$$S = \sum_{i=0}^{n-1} \left(\left(\sum_{t \in D_i} z_t \right) + \left(\sum_{t \in U_i} z_t \right) \right).$$

Remarquons que, par choix des ensembles D_i, si i est différent de i', alors $\sum_{t \in U_i} z_t$ et $\sum_{t \in U_{i'}} z_t$ sont indépendants. En effet, pour $i < i'$, l'ensemble U_i contient des indices inférieurs ou égaux à $(2^{i+1} - 1)d - 1$ et l'ensemble $U_{i'}$ contient des indices supérieurs strictement à $2^{i'} d - 1$. Le choix des ensembles S_i impose que nous avons un nombre logarithmique d'ensembles : $n - 1 \simeq \log(N)$. La taille d'un ensemble D_i est toujours plus petite que d alors que la taille des ensembles U_i tend vers l'infini quand N tend vers l'infini. C'est pourquoi, on a :

$$V = \mathrm{Var}\left(\sum_{i=0}^{n-1} \left(\sum_{t \in U_i} z_t \right) \right) + \mathcal{O}(n).$$

Maintenant, écrivons :

$$\frac{S - M}{\sqrt{V}} = \frac{\sum_{i=0}^{n-1} \left(\sum_{t \in U_i} z_t - E(\sum_{t \in U_i} z_t) \right)}{\sqrt{\mathrm{Var}\left(\sum_{i=0}^{n-1} (\sum_{t \in U_i} z_t) \right) + \mathcal{O}(n)}} + \frac{\sum_{i=0}^{n-1} \left(\sum_{t \in D_i} z_t - E(\sum_{t \in D_i} z_t) \right)}{\sqrt{V}}.$$

Le second terme devient négligeable lorsque N tend vers l'infini car son dénominateur contient 2^{n-1} termes alors que son numérateur en compte au plus $(n-1)d$. Par ailleurs, le premier terme tend vers

$$\frac{\sum_{i=0}^{n-1} \left(\sum_{t \in U_i} z_t - E(\sum_{t \in U_i} z_t) \right)}{\sqrt{\mathrm{Var}\left(\sum_{i=0}^{n-1} (\sum_{t \in U_i} z_t) \right)}}.$$

Comme les ensembles U_i ont été choisis de sorte que $\forall i \neq i',\ \sum_{t \in U_i} z_t$ et $\sum_{t \in U_{i'}} z_t$ soient indépendants, on peut appliquer le théorème central limite à

$$\frac{\sum_{i=0}^{n-1} \left(\sum_{t \in U_i} z_t - E(\sum_{t \in U_i} z_t) \right)}{\sqrt{\mathrm{Var}\left(\sum_{i=0}^{n-1} (\sum_{t \in U_i} z_t) \right)}}.$$

On en déduit que $\frac{S - E(S)}{\sqrt{V}}$ tend vers une loi normale centrée réduite lorsque N tend vers l'infini. ◇

Théorème 5.2 *Soit $(y_t)_{t \geq 0}$ la sortie d'un brasseur synchrone dont l'entrée $(x_t)_{t \geq 0}$ vérifie*

$$\forall t \geq 0,\ \Pr[x_t = 0] = \frac{1}{2} + \varepsilon.$$

Soit Q un polynôme de $\mathbf{F}_2[X]$ possédant d termes, i.e.

$$Q(X) = 1 + \sum_{j=1}^{d-1} X^{i_j}, \text{ avec } i_1 < i_2 < \cdots < i_{d-1}.$$

Soit

$$z_t = y_t + \sum_{j=1}^{d-1} y_{t-i_j}, \forall t \geq i_{d-1}.$$

Notons

$$Z_N = \sum_{t=i_{d-1}}^{N-1} (-1)^{z_t},$$

$$\mu = (N - i_{d-1})(2\varepsilon)^d$$

et

$$\sigma^2 = (N - i_{d-1})\left(1 - (2\varepsilon)^{2d}\right) + \sum_{u=1}^{d-1} N_u \left((2\varepsilon)^{2(d-u)} - (2\varepsilon)^{2d}\right)$$

où N_u désigne le nombre de monômes communs entre $Q(X)$ et $X^u Q(X)$.
Si Q est un multiple du polynôme de rétroaction du LFSR, alors la variable aléatoire

$$Z = \frac{Z_N - \mu}{\sigma}$$

tend vers une loi normale centrée réduite lorsque N tend vers l'infini.

Preuve.
On a :

$$\begin{aligned} \Pr[z_t = 1] &= \Pr[y_t + \textstyle\sum_{j=1}^{d-1} y_{t-i_j} = 1] \\ &= \Pr[x_t + \textstyle\sum_{j=1}^{d-1} x_{t-i_j} = 1], \end{aligned}$$

puisque $s_t + \sum_{j=1}^{d-1} s_{t-i_j} = 0$ lorsque $1 + \sum_{j=1}^{d-1} X^{i_j}$ est un multiple de $P(X)$.

Les x_i sont des variables indépendantes vérifiant $\Pr[x_i = 0] = \frac{1}{2} + \varepsilon$. On a alors la formule suivante :

$$\begin{aligned} \Pr[z_t = 1] &= \sum_{\substack{i=0 \\ i \ odd}}^{d} \binom{d}{i} \left(\frac{1}{2} - \varepsilon\right)^i \left(\frac{1}{2} + \varepsilon\right)^{d-i} \\ &= \frac{1}{2}\left[\sum_{i=0}^{d} \binom{d}{i} \left(\frac{1}{2} - \varepsilon\right)^i \left(\frac{1}{2} + \varepsilon\right)^{d-i} - \sum_{i=0}^{d} \binom{d}{i} \left(\varepsilon - \frac{1}{2}\right)^i \left(\frac{1}{2} + \varepsilon\right)^{d-i}\right] \\ &= \frac{1}{2}\left[1 - (2\varepsilon)^d\right]. \end{aligned}$$

Toutes les variables aléatoires z_t sont identiquement distribuées mais elles ne sont pas nécessairement indépendantes puisqu'elles ont des termes en commun. Cependant, deux

variables aléatoires z_t et $z_{t'}$ ne peuvent être dépendantes que pour des indices t et t' vérifiant $|t'-t| \leq i_{d-1}$. Les hypothèses du lemme 5.1 sont donc vérifiées ce qui implique que

$$\frac{\sum_{t=i_{d-1}}^{N-1} z_t - M}{V}$$

tend vers une loi normale centrée réduite, où M et V sont respectivement la moyenne et la variance de $\sum_{t=i_{d-1}}^{N-1} z_t$.

Calculons alors les valeurs de M et V. La moyenne M se calcule facilement :

$$\begin{aligned} M &= \text{Moyenne}\left(\sum_{t=i_{d-1}}^{N-1} z_t\right) \\ &= \sum_{t=i_{d-1}}^{N-1} \text{Moyenne}(z_t) \\ &= \frac{(N-i_{d-1})}{2}(1-(2\varepsilon)^d). \end{aligned}$$

Le calcul de la variance V est différent car les z_t peuvent être dépendants. On a alors :

$$\begin{aligned} V &= \text{Var}\left(\sum_{t=i_{d-1}}^{N-1} z_t\right) \\ &= \sum_{t=i_{d-1}}^{N-1} \text{Var}(z_t) + \sum_{\substack{t,t'=i_{d-1} \\ t\neq t'}}^{N-1} \text{Cov}(z_t,z_{t'}), \end{aligned}$$

où $\text{Cov}(x,y)$ désigne la covariance entre x et y.

Il est important de remarquer que la valeur de $\text{Cov}(z_t,z_{t'})$ dépend seulement du nombre de termes y_i que z_t et $z_{t'}$ ont en commun. Notons U l'ensemble des indices i tels que y_i apparaît à la fois dans l'écriture de z_t et dans celle de $z_{t'}$ et notons u le cardinal de cet ensemble, *i.e.* le nombre de termes communs à z_t et $z_{t'}$. On peut alors écrire $z_t = c + a$ et $z_{t'} = c + a'$ où a et a' sont indépendants et $c = \sum_{i\in U} y_i$. On a :

$$\text{Cov}(z_t,z_{t'}) = E(z_t z_{t'}) - E(z_t)E(z_{t'}).$$

$$\begin{aligned} \Pr[z_t z_{t'} = 1] &= \Pr[z_t z_{t'} = 1 | c = 0]\Pr[c = 0] + \Pr[z_t z_{t'} = 1 | c = 1]\Pr[c = 1] \\ &= \Pr[a = 1, a' = 1]\Pr[c = 0] + \Pr[a = 0, a' = 0]\Pr[c = 1]. \end{aligned}$$

Mais a et a' sont indépendants, c'est pourquoi :

$$\begin{aligned}\Pr[z_t z_{t'}=1] &= \Pr[a=1]\Pr[a'=1]\Pr[c=0]+\Pr[a=0]\Pr[a'=0]\Pr[c=1]\\ &= \left(\frac{1}{2}(1-(2\varepsilon)^{d-u})\right)^2\times\frac{1}{2}(1+(2\varepsilon)^u)+\left(\frac{1}{2}(1+(2\varepsilon)^{d-u})\right)^2\times\frac{1}{2}(1-(2\varepsilon)^u),\end{aligned}$$

ce qui conduit à

$$\Pr[z_t z_{t'}=1]=\frac{1}{4}\left[1+(2\varepsilon)^{2(d-u)}-2(2\varepsilon)^d\right].$$

D'où,

$$\begin{aligned}\mathrm{Cov}(z_t,z_{t'}) &= \frac{1}{4}\left[1+(2\varepsilon)^{2(d-u)}-2(2\varepsilon)^d\right]-\frac{1}{4}\left[1-(2\varepsilon)^d\right]^2\\ &= \frac{1}{4}\left((2\varepsilon)^{2(d-u)}-(2\varepsilon)^{2d}\right).\end{aligned}$$

Soit N_u le nombre de couples $(z_t,z_{t'})$ qui possèdent exactement u termes y_i en commun. Alors :

$$V=\frac{(N-i_{d-1})}{4}\left(1-(2\varepsilon)^{2d}\right)+\sum_{u=1}^{d-1}\frac{N_u}{4}\left((2\varepsilon)^{2(d-u)}-(2\varepsilon)^{2d}\right).$$

Finalement, si l'on considère la variable aléatoire définie dans l'énoncé par

$$Z_N=\sum_{t=i_{d-1}}^{N-1}(-1)^{z_t}=(N-i_{d-1})-2\sum_{t=i_{d-1}}^{N-1}z_t,$$

il en découle que, pour de grandes valeurs de $N-i_{d-1}$, on peut supposer que $\frac{Z_N-\mu}{\sigma}$ suit une loi normale centrée réduite avec

$$\mu=(N-i_{d-1})-2M$$

et

$$\sigma^2=4V.$$

◇

Par ailleurs, on peut faire une approximation sur la variance de Z afin de rendre la formule du théorème précédent plus maniable.

Corollaire 5.3 *Avec les mêmes notations que dans le théorème précédent, on a la borne supérieure suivante :*

$$\frac{\sigma^2}{(N-i_{d-1})}\leq[1+2d(d-1)](1-(2\varepsilon)^{2d}).$$

Preuve.
On a

$$\sum_{u=1}^{d-1} N_u \leq (N - i_{d-1})2d(d-1).$$

En effet chacun des $(N - i_{d-1})$ z_t a au pire $2d(d-1)$ $z_{t'}$ qui peuvent avoir un y_i en commun avec lui. Donc, dans le pire des cas, on aura $N_1 = N_2 = \cdots = N_{d-2} = 0$ et $N_{d-1} = (N - i_{d-1})2d(d-1)$ car le terme $\left((2\varepsilon)^{2(d-u)} - (2\varepsilon)^{2d}\right)$ qui intervient dans la somme du calcul de la variance est une fonction croissante en la variable u. On a alors :

$$\frac{\sigma^2}{(N - i_{d-1})} \leq \left[1 + 2d(d-1)(2\varepsilon)^2 - (2\varepsilon)^{2d}(1 + 2d(d-1))\right].$$

On peut en déduire la borne supérieure annoncée.

◇

Dans toute la suite de ce manuscrit nous utiliserons cette borne supérieure sur la variance.

Le théorème précédent va nous permettre de détecter les polynômes de poids faible multiples du polynôme de rétroaction P. En effet, si Q est un polynôme non multiple de P alors on a $\Pr[z_t = 1] = \frac{1}{2}$ ce qui implique que la variable aléatoire Z_N suit une loi normale centrée de variance $N - i_{d-1}$. On va donc faire un test d'hypothèse sur la moyenne de la variable aléatoire Z_N. On veut distinguer les deux hypothèses suivantes :

- (H_0) : $|Z_N|$ suit une loi normale de moyenne 0 et de variance $N - i_{d-1}$. Cette hypothèse correspond au cas où le polynôme $Q(X) = 1 + \sum_{j=1}^{d-1} X^{i_j}$ n'est pas un multiple de P.
- (H_1) : $|Z_N|$ suit une loi normale de moyenne μ et de variance σ^2. Cette hypothèse correspond alors au cas où Q est un multiple du polynôme P.

On va utiliser un seuil de décision T, $T > 0$, pour faire un choix entre ces deux hypothèses. Si $|Z_N| < T$ alors on accepte l'hypothèse (H_0), sinon on accepte (H_1). Le nombre minimum de bits de la suite $(y_t)_{t \geq 0}$ requis dépend du nombre d'erreurs de décision que l'on s'autorise, c'est-à-dire des probabilités de fausse alarme et de non-détection qui correspondent aux 2 types d'erreurs décrits au tableau suivant.

décision / réalité	(H_0) acceptée	(H_1) acceptée
(H_0) vraie	OK	fausse alarme
(H_1) vraie	non-détection	OK

Soit ϕ la fonction de répartition associée à la densité de la loi normale :

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

Calculons maintenant les probabilités de fausse alarme P_f et de non-détection P_n. On a :

$$\begin{aligned} P_f &= Pr\left[|Z_N| \geq T \mid (H_0) \text{ vraie}\right] \\ &= 2Pr\left[Z_N \leq -T \mid Z_N \in N(0, \sqrt{N - i_{d-1}})\right] \\ &= 2\phi\left(\frac{-T}{\sqrt{N - i_{d-1}}}\right). \end{aligned}$$

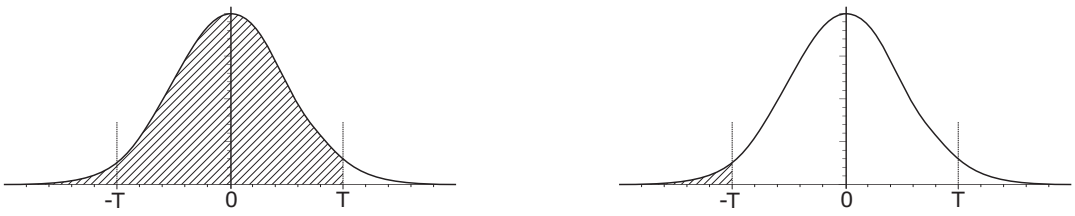

FIG. 5.2 – *Loi normale.*

De même, on peut calculer :

$$\begin{aligned} P_n &= Pr\left[|Z_N| \leq T \mid (H_1) \text{ vraie}\right] \\ &= \frac{1}{\sqrt{2\pi}} \int_{\frac{-T-\mu}{\sigma}}^{\frac{T-\mu}{\sigma}} \exp\left(-\frac{x^2}{2}\right) dx \\ &= \phi\left(\frac{T-\mu}{\sigma}\right) - \phi\left(\frac{-T-\mu}{\sigma}\right). \end{aligned}$$

On approximera P_n par :

$$P_n \simeq \phi\left(\frac{T - |\mu|}{\sigma}\right)$$

car dans la plupart des cas $\phi(\frac{-T-|\mu|}{\sigma})$ est négligeable devant $\phi(\frac{T-|\mu|}{\sigma})$.

On a donc

$$\phi^{-1}\left(P_n\right) = \frac{T-\mu}{\sigma}, \tag{5.2}$$

$$\phi^{-1}\left(1 - \frac{P_f}{2}\right) = \frac{T}{\sqrt{N - i_{d-1}}}. \tag{5.3}$$

Notons $a = \phi^{-1}\left(1 - \frac{P_f}{2}\right)$ et $b = -\phi^{-1}\left(P_n\right)$, a et b étant strictement positifs.
En utilisant les équations (5.2), (5.3) et la borne sur la variance σ^2 détaillée au corollaire 5.3, on en déduit que :

$$b \geq \frac{\left(\frac{T}{a}\right)^2 (2\varepsilon)^d - T}{\left(\frac{T}{a}\right)\sqrt{(1 + 2d(d-1))(1 - (2\varepsilon)^{2d})}}.$$

D'où

$$T \leq \frac{1 + \frac{b}{a}\sqrt{(1 + 2d(d-1))(1 - (2\varepsilon)^{2d})}}{\frac{(2\varepsilon)^d}{a^2}}.$$

Pour ε petit, on a donc

$$\begin{aligned} T &\leq \frac{a^2 + ab\sqrt{(1+2d(d-1))(1-(2\varepsilon)^{2d})}}{(2|\varepsilon|)^d} \\ &\lesssim \frac{a\left(a + b\sqrt{1+2d(d-1)}\right)}{(2|\varepsilon|)^d}. \end{aligned} \tag{5.4}$$

Posons $\delta = \sqrt{1+2d(d-1)}$, on déduit alors de l'équation (5.3) que :

$$N - i_{d-1} = \left(\frac{T}{a}\right)^2.$$

Et alors :

$$\begin{aligned} N &\leq i_{d-1} + \frac{\left(a + b\delta\sqrt{1-(2\varepsilon)^{2d}}\right)^2}{(2\varepsilon)^{2d}} \\ &\leq i_{d-1} + \frac{a^2 + 2ab\delta\sqrt{1-(2\varepsilon)^{2d}} + b^2\delta^2\left(1-(2\varepsilon)^{2d}\right)}{(2\varepsilon)^{2d}}. \end{aligned}$$

Pour ε petit, en faisant un développement limité de $\sqrt{1-(2\varepsilon)^{2d}}$ à l'ordre 1 au voisinage de zéro, on obtient :

$$N \lesssim i_{d-1} - b\delta\left(a + b\delta\right) + \frac{\left(a+b\delta\right)^2}{(2\varepsilon)^{2d}}\ . \tag{5.5}$$

Il apparaît alors clairement que le nombre N de bits de la suite de sortie $(y_t)_{t\geq 0}$ nécessaires pour reconstruire le polynôme de rétroaction augmente avec le poids d (*i.e.* avec le nombre de termes) du polynôme Q. Pour cette raison, on cherchera seulement des multiples de P de poids faible. L'algorithme de reconstruction du polynôme de rétroaction consiste alors à déterminer si Q est un multiple de P pour tous les Q de poids d et de degré plus petit qu'une certaine borne D (les valeurs typiques de d sont 3, 4 ou 5). Son déroulement est détaillé à la table 5.1.

Algorithme.
Entrée :

- N premiers bits de la sortie du brasseur, $y_0, \ldots, y_{N-1}$.
- $a = \phi^{-1}\left(1 - \frac{P_f}{2}\right)$ et $b = -\phi^{-1}(P_n)$.
- ε tel que l'entrée du brasseur $(x_t)_{t\geq 0}$ vérifie : $\Pr[x_t = 0] = \frac{1}{2} + \varepsilon$.
- $d =$ poids des multiples.

Sortie : le polynôme de rétroaction du LFSR (ou un multiple).

Déroulement :
Calculer T avec (5.4)
Pour tout $(i_1, \ldots, i_{d-1})$ tel que $0 < i_1 < \cdots < i_{d-1} \leq D$
  Calculer N en utilisant (5.5)
  $Z_N \leftarrow 0$
  Pour t de i_{d-1} à N
    $z \leftarrow y_t + \sum_{j=1}^{d-1} y_{t-i_j}$
    $Z_N \leftarrow Z_N + (-1)^z$
  Si $|Z_N| > T$
    Stocker $Q = 1 + \sum_{j=1}^{d-1} X^{i_j}$ dans une liste
    Pour $Q' \neq Q$ dans la liste
      Calculer pgcd(Q,Q')
      Si pgcd(Q,Q') $\neq 1$
        Retourner pgcd(Q,Q').

TAB. 5.1 – *Algorithme de reconstruction du polynôme de rétroaction.*

Analyse de complexité et optimisation des paramètres

Maintenant, nous allons voir comment régler les différents paramètres de notre algorithme. Quand P est un polynôme primitif aléatoire de degré L, le nombre moyen $m(d)$ de multiples de P qui ont un poids d et un degré plus petit que D peut être approximé par [CT00] :

$$m(d) = \frac{D^{d-1}}{(d-1)!2^L}. \tag{5.6}$$

Dans la table 5.2, on a testé cette approximation dans le cas de trinômes : on compare donc le nombre de trinômes trouvés effectivement à $m(3) = \frac{D^2}{2^{L+1}}$.

deg(P)	D	Nb de trinômes générés	Nb théorique de trinômes (formule (5.6))
26	10000	3	0.7
	100000	102	74.5
	1000000	7481	7450.6
27	10000	2	0.4
	100000	37	37.2
	1000000	3687	3725.3
28	10000	0	0.18
	100000	23	18.6
	1000000	1818	1862.6
30	10000	0	0.04
	100000	4	4.7
	1000000	469	465.7
60	10000	12	0
	100000	21	0
	1000000	33	0

TAB. 5.2 – *Pertinence de la formule* $m(d) = \frac{D^{d-1}}{(d-1)!2^L}$.

Les polynômes que nous avons pris pour faire nos tests sont les suivants :

$$P_{26}(x) = x^{26} + x^{14} + x^{10} + x^8 + x^7 + x^6 + x^4 + x + 1$$

$$P_{27}(x) = x^{27} + x^{12} + x^{10} + x^9 + x^7 + x^5 + x^3 + x^2 + 1$$

$$P_{28}(x) = x^{28} + x^{13} + x^7 + x^6 + x^5 + x^2 + 1$$

$$P_{30}(x) = x^{30} + x^{17} + x^{16} + x^{13} + x^{11} + x^7 + x^5 + x^3 + x^2 + x + 1$$

$$P_{60}(x) = x^{60} + x + 1$$

Pour le cas du degré 60 la différence entre le nombre de trinômes trouvés et le nombre de trinômes théoriques vient du fait que le polynôme de degré 60 était un trinôme et dans le cas d'un trinôme, pour tout i, $P(X)^{2^i}$ est aussi un trinôme. De plus la formule n'est valide que si D est suffisamment grand.

Pour un polynôme de rétroaction de degré L notre algorithme trouvera un multiple de degré inférieur ou égal à D dès que $m(d) = 2$, donc après 2^{L+1} essais pour $(i_1, \ldots, i_{d-1})$. En effet, nous avons choisi de chercher deux multiples du polynôme de rétroaction afin de réduire la probabilité de fausse alarme ; si on choisissait de renvoyer le premier multiple trouvé alors la moindre erreur entraînerait une erreur pour le polynôme de rétroaction. Au contraire, avec l'option que nous avons choisie, le polynôme de rétroaction retourné ne sera pas correct uniquement si on trouve deux polynômes détectés comme multiples, alors qu'ils ne le sont pas et dont le pgcd est non trivial.

Algorithme.
Entrée :

- N premiers bits de la sortie du brasseur, $y_0, \ldots, y_{N-1}$.
- $a = \phi^{-1}\left(1 - \frac{P_f}{2}\right)$ et $b = -\phi^{-1}(P_n)$.
- ε tel que l'entrée du brasseur $(x_t)_{t\geq 0}$ vérifie : $\Pr[x_t = 0] = \frac{1}{2} + \varepsilon$.
- $d =$ poids des multiples.

Sortie : le polynôme de rétroaction du LFSR (ou un multiple)

Déroulement :
Calculer T avec (5.4)
Pour tout $(i_1, \ldots, i_{d-1})$ tel que $0 < i_1 < \cdots < i_{d-1} \leq D$
  Calculer N en utilisant (5.5)
  $Z_N \leftarrow 0$
  Pour t de i_{d-1} à N
    $z \leftarrow y_t + \sum_{j=1}^{d-1} y_{t-i_j}$
    $Z_N \leftarrow Z_N + (-1)^z$
  Si $|Z_N| > T$
    Stocker $Q = 1 + \sum_{j=1}^{d-1} X^{i_j}$ dans une liste
    Pour $Q', Q'' \neq Q$ dans la liste
      Calculer pgcd(Q,Q',Q'')
      Si pgcd(Q,Q',Q'') $\neq 1$
        Retourner pgcd(Q,Q',Q'').

TAB. 5.3 – *Variante de notre algorithme de reconstruction du polynôme de rétroaction.*

Notre algorithme retrouvera bien le bon polynôme de rétroaction si la probabilité de fausse alarme choisie est très faible (par exemple $P_f = 2.10^{-10}$). Pour des valeurs plus élevées de P_f, le nombre de multiples à trouver augmente car certains des polynômes détectés ne sont pas divisibles par P. Pour cette raison, nous avons comparé deux algorithmes : l'algorithme initial qui retourne le premier pgcd non trivial de deux polynômes détectés et une variante de cet algorithme, présentée dans la table 5.3, qui, elle, retourne le premier pgcd non trivial de trois polynômes détectés.

On a effectué des tests sur 100 polynômes primitifs de degré 15 afin d'examiner l'impact de P_f sur les deux algorithmes. Les résultats sont comparés dans les tables 5.4 et 5.5. Ces simulations ont été effectuées sur un Pentium 4 à 3.2GHz avec $d = 3$ et $P_n = 10^{-5}$. Comme la valeur de ε n'est pas essentielle pour déterminer l'influence de P_f, nous ne présentons que les simulations effectuées pour $\varepsilon = 0.2$.

P_f	succès %	temps de calcul moyen (en s)	valeur moyenne de N
2.10^{-4}	16	6.78	36614
7.10^{-5}	50	13.71	38291
2.10^{-5}	79	21.91	40147
7.10^{-6}	95	24.47	41602
2.10^{-6}	100	26.95	44196
2.10^{-7}	100	28.56	46239

TAB. 5.4 – *Performance de l'algorithme initial (tests sur* 100 *brasseurs de longueur* 15*).*

P_f	succès %	temps de calcul moyen (en s)	valeur moyenne de N
2.10^{-4}	16	9.90	36671
7.10^{-5}	51	21.53	38373
2.10^{-5}	81	23.60	40179
7.10^{-6}	95	33.77	41696
2.10^{-6}	100	36.54	44283
2.10^{-7}	100	37.76	46334

TAB. 5.5 – *Performance de la variante décrite à la table notre algorithme qui retourne le premier pgcd non trivial de 3 polynômes détectés (tests sur* 100 *brasseurs de longueur* 15*).*

Il apparaît assez nettement que, le temps de calcul moyen de la variante décrite à la table 5.3 est plus élevé. Par exemple, si l'on veut reconstruire un brasseur de longueur 15 avec une probabilité de succès de l'ordre de 0.5, notre algorithme original prend environ 14 secondes pour $P_f = 7.10^{-5}$. On voit que pour la même valeur de P_f, la variante qui retourne le pgcd de 3 polynômes détectés a pratiquement le même taux de succès mais il prend en moyenne plus de temps (environ 22 secondes). Par ailleurs, la variante ne permet pas d'augmenter significativement la probabilité de succès pour des valeurs de P_f élevées, et a donc relativement peu d'intérêt.

C'est pourquoi nous allons maintenant nous intéresser seulement à notre algorithme original puisqu'il donne de meilleurs résultats en terme de complexité. Nos simulations mettent aussi en évidence le fait que P_f doit être choisi dans l'intervalle $[10^{-7}; 5.10^{-6}]$ pour atteindre un taux de succès supérieur à 90%. Des valeurs plus faibles de P_f ne donnent pas de meilleurs résultats et font augmenter sensiblement le temps de calcul car elles nécessitent plusieurs calculs de pgcd pour obtenir une probabilité de succès raisonnable comme nous venons de le montrer.

Si P est un polynôme primitif aléatoire de degré L, la longueur minimale de la suite de sortie nécessaire pour notre algorithme de reconstruction est :

$$N_P \lesssim (d-1)!^{\frac{1}{d-1}} 2^{\frac{L}{d-1}} + \frac{(a+b\delta)^2}{(2\varepsilon)^{2d}}. \tag{5.7}$$

Les paramètres d, P_f et P_n doivent donc être choisis de telle sorte que le nombre de bits de sortie effectivement disponibles soit supérieur à N_P. Le nombre d'opérations effectuées par notre algorithme est :

$$\begin{aligned} W_P &= \frac{D^{d-1}}{(d-1)!} d(N-D) \\ &\lesssim d2^L \frac{(a+b\delta)^2}{(2\varepsilon)^{2d}}. \end{aligned}$$

Dès que le nombre de bits disponibles est supérieur à N_P (ce qui est toujours le cas dans nos applications) le choix optimal pour minimiser le nombre d'opérations est $d = 3$.

Ces résultats doivent être comparés à une autre stratégie pour retrouver le polynôme de rétroaction du LFSR qui consiste à le rechercher parmi tous les polynômes de degré plus petit que $L_{\max}$. On doit alors choisir $D = L_{\max}$ et d doit prendre toutes les valeurs impaires inférieures ou égales au poids w de P. On se contente en effet des valeurs impaires car P est primitif et ne possède donc qu'un nombre impair de termes. Pour cet autre algorithme, on trouve :

$$N'_P \simeq L_{\max} + \frac{(a+b)^2}{(2\varepsilon)^{2w}},$$

$$W'_P \simeq \frac{wL_{\max}^{w-1}}{(w-1)!} \cdot \frac{(a+b)^2}{(2\varepsilon)^{2w}}.$$

Comparons ces résultats pour P primitif aléatoire de degré L, ce qui conduit à choisir $w = \frac{L}{2}$. Dans ce cas, on voit donc que notre algorithme requiert moins de bits de sortie et est beaucoup plus efficace que l'énumération de tous les polynômes de degré plus petit que $L_{\max}$. En effet, le terme dominant dans l'expression du nombre de bits requis est $(d-1)!^{\frac{1}{d-1}} 2^{\frac{L}{d-1}}$ pour notre algorithme alors que c'est $\frac{1}{(2\varepsilon)^{2w}}$ dans le cas de l'énumération de tous les polynômes possibles. Pour le nombre d'opérations, on a fait des simulations avec $L_{\max} = 100$, $P_f = 2.10^{-7}$ et $P_n = 10^{-5}$. Les résultats sont comparés sur les figures 5.3 et 5.4.

Les résultats montrent donc que notre algorithme avec $d = 3$ sera plus efficace pour un polynôme primitif dès que $L \geq 6$ ce qui sera toujours le cas en pratique. Pour $d = 5$, notre algorithme sera plus efficace dès que $L \geq 8$ ce qui sera aussi généralement le cas en pratique.

Le tableau 5.6 donne le temps de calcul qu'il nous a fallu pour retrouver différents polynômes de rétroaction sur un Pentium 4 à 3.2GHz avec comme paramètres $d = 3$, $\varepsilon = 0.1$, $P_f = 2.10^{-7}$ et $P_n = 10^{-5}$. Pour la table 5.7, on prend les mêmes valeurs de paramètres sauf que le biais est 0.05 au lieu de 0.1.

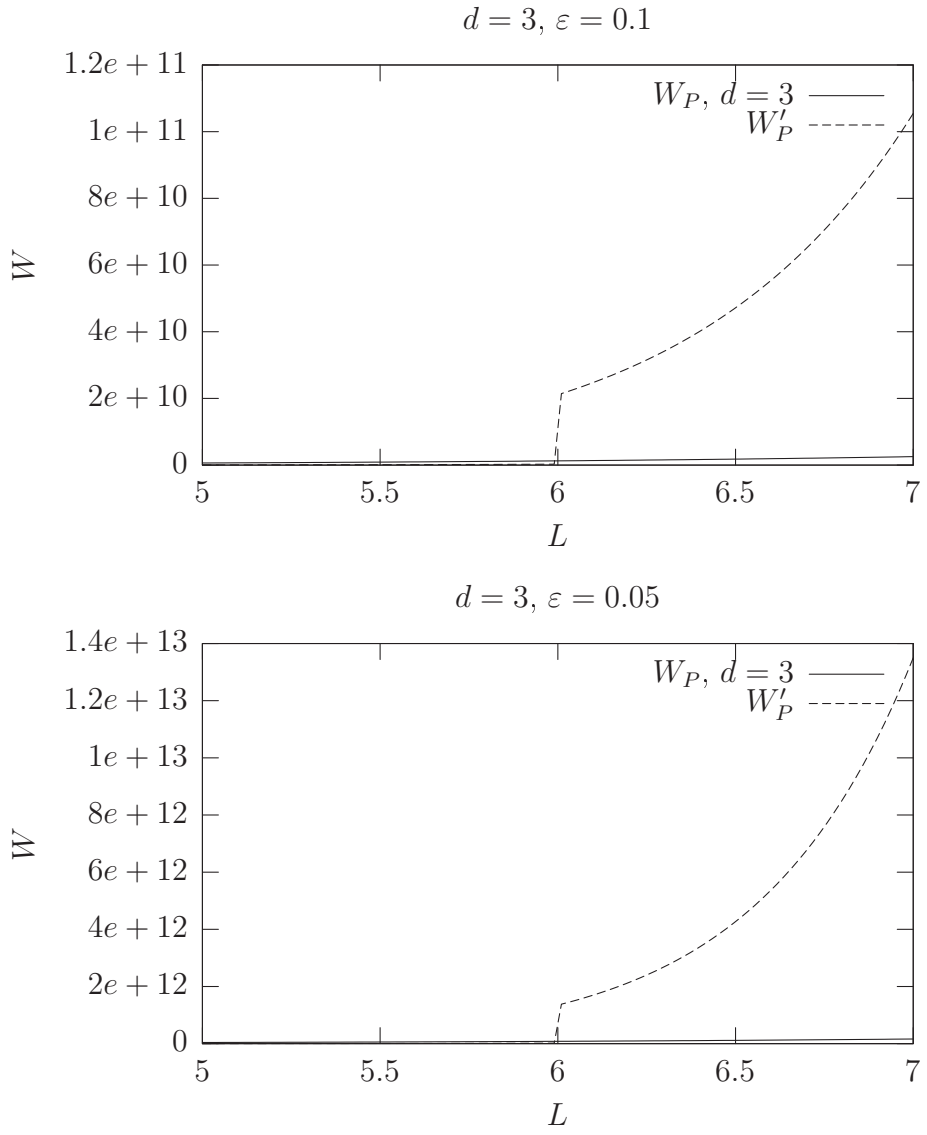

FIG. 5.3 – *Comparaison des deux algorithmes pour* $d = 3$ *(*$\varepsilon = 0.1$ *en haut et* $\varepsilon = 0.05$ *en bas).*

Polynôme de rétroaction	Polynôme retrouvé	temps de calcul	Valeur de N
$x^8 + x^4 + x^3 + x^2 + 1$	$x^8 + x^4 + x^3 + x^2 + 1$	2.5 s	1399696
$x^{10} + x^6 + x^5 + x^3 + x^2 + x + 1$	$x^{23} + x^7 + 1$	8.9 s	1399717
$x^{15} + x^5 + x^4 + x^2 + 1$	$x^{15} + x^5 + x^4 + x^2 + 1$	20 mn 32 s	1400202
$x^{21} + x^6 + x^5 + x^2 + 1$	$x^{21} + x^6 + x^5 + x^2 + 1$	3 h 42	1401427
$x^{29} + x^2 + 1$	$x^{29} + x^2 + 1$	13.8 s	1399729

TAB. 5.6 – *Performances de notre algorithme pour* $\varepsilon = 0.1$.

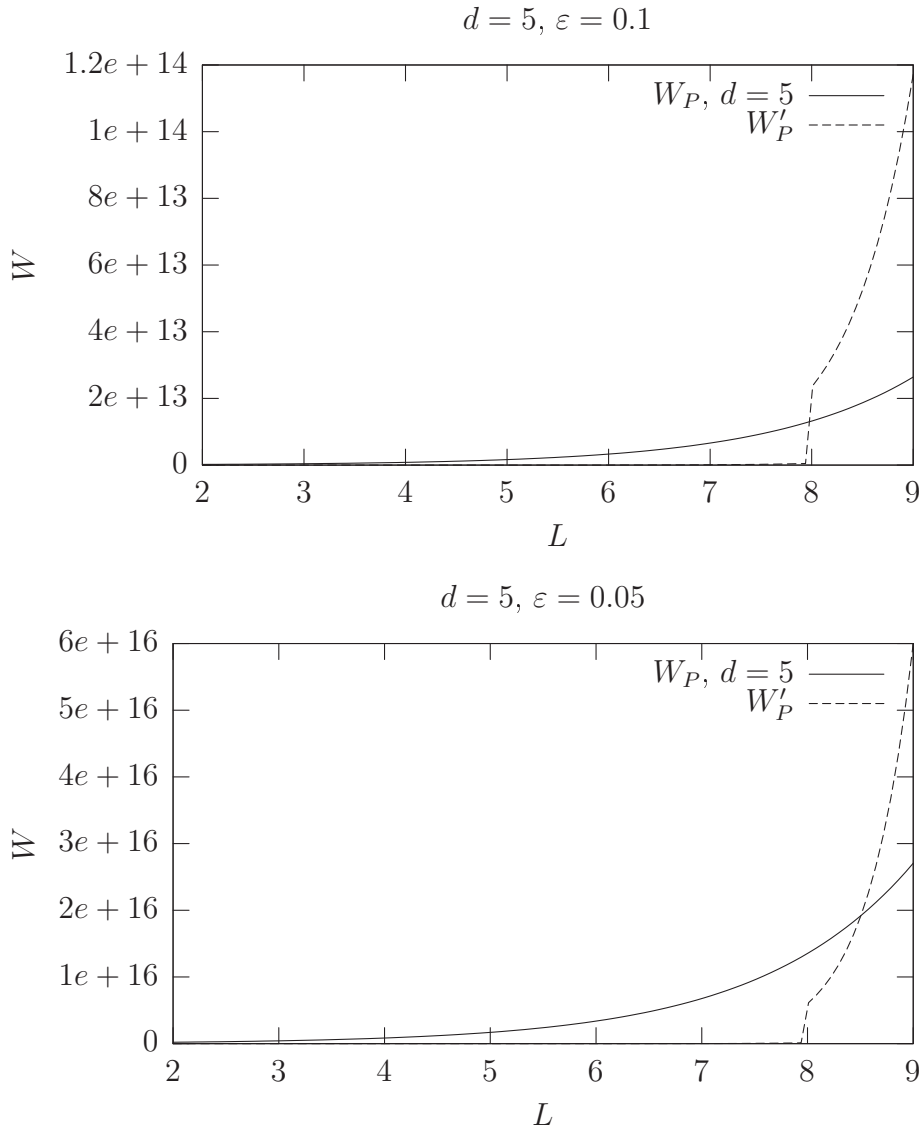

FIG. 5.4 – *Comparaison des deux algorithmes pour* $d = 5$ *(*$\varepsilon = 0.1$ *en haut et* $\varepsilon = 0.05$ *en bas).*

Polynôme de rétroaction	Polynôme retrouvé	temps de calcul	Valeur de N
$x^8 + x^4 + x^3 + x^2 + 1$	$x^8 + x^4 + x^3 + x^2 + 1$	2 mn 58 s	89582609
$x^{10} + x^6 + x^5 + x^3 + x^2 + x + 1$	$x^{23} + x^7 + 1$	10 mn 38 s	89582630
$x^{15} + x^5 + x^4 + x^2 + 1$	$x^{15} + x^5 + x^4 + x^2 + 1$	24 h 55 mn	89583115
$x^{21} + x^6 + x^5 + x^2 + 1$	$x^{21} + x^6 + x^5 + x^2 + 1$	11 jours 4 h	89584340
$x^{29} + x^2 + 1$	$x^{29} + x^2 + 1$	16 mn 36 s	89582642

TAB. 5.7 – *Performances de notre algorithme pour* $\varepsilon = 0.05$.

Il est important de noter que notre algorithme est très efficace lorsque le polynôme de rétroaction du brasseur est un trinôme, ce qui est le cas dans la plupart des systèmes de communications réels pour des raisons d'implémentation et de rapidité.

Reconstruction de l'état initial du registre

Une fois le polynôme de rétroaction P du LFSR retrouvé, la reconstruction complète d'un brasseur synchrone nécessite de retrouver l'état initial du registre utilisé.

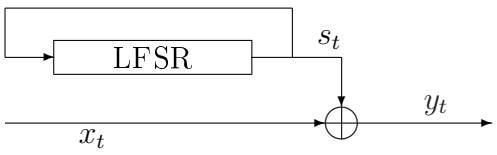

FIG. 5.5 – *Problème étudié.*

Comme précédemment, nous avons fait l'hypothèse que le message $(x_t)_{t\geq 0}$ en entrée du brasseur était biaisé, ce qui implique que $(y_t)_{0\leq t\leq N-1}$ fournit un fragment de la sortie du LFSR avec une probabilité d'erreur de $\frac{1}{2} - \varepsilon$.

Principe La détermination de l'état initial d'un LFSR, dont le polynôme de rétroaction est connu, à partir de la connaissance d'un fragment bruité de sa sortie est un problème classique en cryptanalyse. Il est résolu par les attaques dites par corrélation rapides introduites par Meier et Staffelback [MS89], et qui ont été depuis améliorées par de nombreuses variantes [CT00, CJS00, JJ00]. Le principe essentiel de ces attaques est de modéliser le problème par un problème de correction d'erreurs. On assimile en effet la sortie du brasseur au résultat de la transmission de la suite produite par le LFSR à travers un canal bruité. Les erreurs qui surviennent au cours de la transmission proviennent en réalité du message. Comme la suite engendrée par un LFSR est fortement redondante puisqu'il s'agit d'une suite engendrée par une relation de récurrence linéaire, on va pouvoir la corriger en utilisant un algorithme de décodage approprié (voir Fig 5.6).

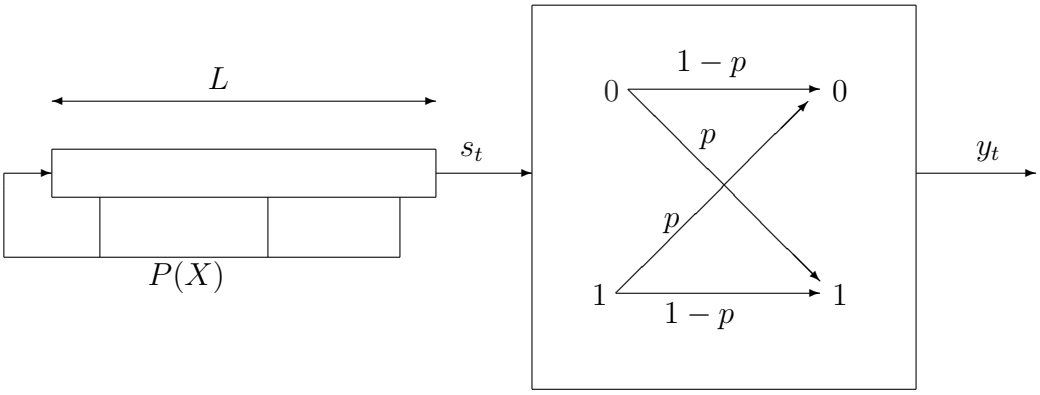

FIG. 5.6 – *Modèle d'une attaque par corrélation rapide ou de façon équivalente, du problème de reconstruction de l'état initial d'un brasseur synchrone.*

La suite $(s_t)_{t\geq 0}$, par définition, vérifie l'équation de récurrence linéaire définie par le polynôme de rétroaction P. Le mot de N bits $(s_0,\ldots,s_{N-1})$, est donc un mot du code de longueur N et de dimension L, que nous allons chercher à décoder. Il existe plusieurs techniques de décodage. L'idée d'utiliser le décodage itératif a été introduite dans [MS89] et développé par exemple dans [CT00]. Il existe d'autres techniques au moins aussi performantes, comme le décodage à maximum de vraisemblance d'un code de dimension réduite, développé dans [CJS00]. Ici, on choisit d'utiliser la technique du décodage itératif car notre algorithme de reconstruction du polynôme fournit déjà des équations de parité de poids faible. On utilise donc ici l'algorithme dû à Gallager présenté dans la section 3.2.2, page 64, qui exploite l'existence d'équations de parité creuses pour le code.

L'attaque se divise donc en deux parties bien distinctes :

- *une phase de précalcul*, qui ne dépend que du polynôme de rétroaction P. Cette phase consiste à déterminer des équations de parité de poids d' pour la suite $(s_t)_{t\geq 0}$. Dans la suite, nous prendrons $d'=3$ car cette valeur conduit à l'algorithme le plus performant pour le nombre de bits de sortie N imposé par la phase de reconstruction du polynôme, comme nous le verrons par la suite, page 127. Notons que le fait que d' soit optimal est spécifique au contexte de la reconstruction d'un brasseur et n'est généralement pas vrai dans celui plus classique de la cryptanalyse ;
- *une phase de décodage*, qui consiste à décoder la suite $(y_t)_{t<N}$ afin de retrouver $(s_t)_{t<N}$ dont les premiers bits correspondent à l'initialisation du brasseur.

Dans un contexte cryptographique, $(y_t)_{t\geq 0}$ correspond à la suite chiffrée et $(s_t)_{t\geq 0}$ est la sortie d'un LFSR corrélée à la suite chiffrée. Dans ce contexte, on différencie calcul/précalcul car le précalcul se fait indépendamment de la connaissance de la sortie. Ce n'est pas le cas ici puisque lorsqu'on cherche à reconstruire l'état initial du registre, on doit déjà connaître la sortie car on en a eu besoin pour reconstruire le polynôme de rétroaction du registre. Nous avons quand même choisi de garder cette terminologie pour distinguer les deux phases de l'algorithme.

Phase de précalcul. Dans cette phase, on recherche toutes les équations de parité faisant intervenir exactement trois bits de la suite $(s_t)_{t<N}$, il s'agit des équations de la forme

$$s_t + s_{t-i} + s_{t-j} = 0$$

satisfaites pour tout t. De telles équations correspondent exactement aux trinômes multiples de P de la forme

$$1 + X^i + X^j.$$

L'étape de précalcul consiste donc à rechercher tous ces trinômes. Pour cela, on utilisera l'algorithme suivant :

Algorithme.
Entrée : le polynôme de rétroaction du LFSR : P
Sortie : les trinômes multiples de P de degré plus petit que D

- *Calculer tous les restes* $q_i(X) = X^i \ mod \ P(X)$ *pour* $1 \leq i < D$ *et les stocker dans un tableau* T *défini par :*

$$\forall a,\ 0 \leq a < 2^L,\ \ T[a] = \{i, q_i(X) = a\}.$$

- *Pour tout* $i \in \{1, \ldots, D\}$,
  calculer $A = 1 + q_i(X)$
  Pour $j \in T[A]$, $1 + X^i + X^j$ *est un multiple de* $P(X)$ *ayant la forme souhaitée.*

TAB. 5.8 – *Algorithme pour trouver des trinômes multiples du polynôme de rétroaction.*

En fait, dans la phase de reconstruction du polynôme de rétroaction, on a déjà trouvé les trinômes de degré plus petit qu'un certain D_r multiples du polynôme de rétroaction, donc il suffit de faire la recherche sur les trinômes de degré supérieur à D_r.

On s'intéresse au nombre de trinômes multiples de degré inférieur ou égal à D d'un polynôme primitif de degré L. On a déjà vu que leur nombre moyen est de l'ordre de :

$$m(3) \simeq \frac{D^2}{2^{L+1}}.$$

Les résultats obtenus à partir de l'algorithme de la phase de précalcul montre que la formule est une bonne approximation du nombre de trinômes trouvés (*cf.* table 5.2).

La complexité de cette phase de précalcul est de l'ordre de :

$$\mathcal{O}(D) = \mathcal{O}\left(\sqrt{m_{\text{requis}}(3)}\, 2^{\frac{L}{2}}\right),$$

où $m_{\text{requis}}(3)$ est le nombre de trinômes requis par l'algorithme de la phase de décodage.

Remarque 5.4 *Pour* $d' \neq 3$, *la recherche de multiples du polynôme de rétroaction de poids* d' *ne s'effectue pas exactement de la même manière : elle repose sur un compromis*

mémoire/temps [CT00]. Pour $d' \geq 5$, il existe également une amélioration plus poussée de cet algorithme décrit dans [CJM02].

Phase de décodage On va maintenant utiliser les équations de parité de poids faible obtenues lors de la phase de précalcul pour décoder le mot $(y_0, \ldots, y_{N-1})$. L'algorithme de décodage que l'on va employer est l'algorithme itératif décrit dans la partie 3.2.2 page 64. On utilise ici la version avec les log-vraisemblances décrite à la table 3.3. Dans ce contexte une version approchée, moins coûteuse, de l'algorithme de Gallager sous forme log-vraisemblances est suffisante. Cette approximation utilise la propriété suivante.

Proposition 5.5 *[HOP96] On peut faire l'approximation suivante :*

$$\log\left(\frac{\prod_{j=1}^{J}(\exp(x_j)+1)+\prod_{j=1}^{J}(\exp(x_j)-1)}{\prod_{j=1}^{J}(\exp(x_j)+1)-\prod_{j=1}^{J}(\exp(x_j)-1)}\right) \simeq \mathrm{Sign}(\prod_{j=1}^{J} x_j).\min_{j=1\ldots J}|x_j|.$$

Corollaire 5.6 *Soit I_e l'ensemble des indices intervenant dans l'équation de parité e. On peut remplacer dans l'algorithme de Gallager*

$$\forall e,\ g\left(\mathrm{Extr_e}(y_t)\right) = \log\left(\frac{1-\prod_{i\in I_e\setminus\{t\}}\frac{1-\exp(g(A_e(y_i)))}{1+\exp(g(A_e(y_i)))}}{1+\prod_{i\in I_e\setminus\{t\}}\frac{1-\exp(g(A_e(y_i)))}{1+\exp(g(A_e(y_i)))}}\right),$$

par

$$\forall e,\ g\left(\mathrm{Extr_e}(y_t)\right) = (-1)^{\sharp I_e}\mathrm{Sign}\left(\prod_{i\in I_e\setminus\{t\}} g\left(A_e(y_i)\right)\right).\min_{i\in I_e\setminus\{t\}}|g\left(A_e(y_i)\right)|.$$

Preuve.

Notons $x_i = g\left(A_e(y_i)\right)$, alors :

$$\begin{aligned}
g\left(\mathrm{Extr_e}(y_t)\right) &= \log\left(\frac{1-\prod_{i\in I_e\setminus\{t\}}\frac{1-\exp(x_i)}{1+\exp(x_i)}}{1+\prod_{i\in I_e\setminus\{t\}}\frac{1-\exp(x_i)}{1+\exp(x_i)}}\right)\\
&= \log\left(\frac{\prod_{i\in I_e\setminus\{t\}}(1+\exp(x_i))-\prod_{i\in I_e\setminus\{t\}}(1-\exp(x_i))}{\prod_{i\in I_e\setminus\{t\}}(1+\exp(x_i))+\prod_{i\in I_e\setminus\{t\}}(1-\exp(x_i))}\right)\\
&= \log\left(\frac{\prod_{i\in I_e\setminus\{t\}}(1+\exp(x_i))-(-1)^{\sharp I_e-1}\prod_{i\in I_e\setminus\{t\}}(\exp(x_i)-1)}{\prod_{i\in I_e\setminus\{t\}}(1+\exp(x_i))+(-1)^{\sharp I_e-1}\prod_{i\in I_e\setminus\{t\}}(\exp(x_i)-1)}\right).
\end{aligned}$$

En distinguant les deux cas, $\sharp I_e - 1$ pair ou impair et en utilisant l'approximation de la propriété précédente, on trouve le résultat annoncé. ◇

Dans notre cas ici, on ne s'intéresse qu'à des équations de parité de poids 3 donc on aura $\sharp I_e$ impair.

On fait également une deuxième approximation qui consiste à remplacer les probabilités a posteriori partielles par les APP. En effet, du fait de la valeur élevée de n et du grand nombre m d'équations de parité, la complexité en mémoire correspondant au stockage des APP partielles, $\mathcal{O}(mn)$, rend leur utilisation impossible en pratique dans notre contexte (ou dans celui des attaques par corrélation rapides). On obtient alors la version approchée de l'algorithme de Gallager décrite à la table 5.9. Cette version est plus naturelle que la version approchée décrite dans [CT00] comme montré dans [Lev04]. Elle corrige également l'erreur présente dans l'algorithme décrit à la page 87 de la thèse de S. Leveiller ; en effet dans la formule $g(\mathrm{Extr_e}(y_t))$ (noté $\mathcal{L}(\mathrm{Extr_e}(y_t))$ dans le manuscrit de S. Leveiller), il y a bien un $(-1)^{\sharp I_e}$ contrairement à ce qui est écrit.

Algorithme.
Entrée : Un mot de code bruité $(y_1, \ldots, y_n)$.
Sortie : Ce mot de code décodé.

- **Initialisation :** Pour tout $t \in \{1, \ldots, n\}$
  - Calculer $g(\mathrm{Obs}(y_t)) = \begin{cases} \log\left(\frac{1-\tau}{\tau}\right) & \text{si l'observation est un 1.} \\ \log\left(\frac{\tau}{1-\tau}\right) & \text{si l'observation est un 0.} \end{cases}$
  - $g(\mathrm{Extr_e}(y_t)) = 0$.
- **Itérer :** Pour tout $t \in \{1, \ldots, n\}$,
  - Calculer :

$$g(\mathrm{APP}(y_t)) = g(\mathrm{Obs}(y_t)) + \left[\sum_i g(\mathrm{Extr_i}(y_t))\right].$$

  - Calculer :

$$\forall e, g(\mathrm{Extr_e}(y_t)) = (-1)^{\sharp I_e} \mathrm{Sign}\left(\prod_{i \in I_e \setminus \{t\}} g(\mathrm{APP}(y_i))\right) . \min_{i \in I_e \setminus \{t\}} |g(\mathrm{APP}(y_i))|,$$

    avec $I_e = \{i | y_i \text{ appartient à l'équation } e\}$.
- **Terminaison :**
  Pour tout $t \in \{1, \ldots, n\}$,
  - calculer

$$g(\mathrm{APP}(y_t)) = g(\mathrm{Obs}(y_t)) + \left[\sum_i g(\mathrm{Extr_i}(y_t))\right].$$

  - Si $g(\mathrm{APP}(y_t)) > 0$ alors $y_t = 1$, sinon $y_t = 0$.

TAB. 5.9 – *Algorithme de Gallager approché.*

Bien sûr, en pratique, on ne stocke pas les $g\left(\mathrm{Extr_e}(y_t)\right)$ ni les $g\left(\mathrm{Obs}(y_t)\right)$, mais nous les avons laissées dans la description de l'algorithme simplement pour faire le parallèle avec les autres versions de l'algorithme de Gallager et simplifier aussi la lecture. En fait, les deux opérations itérées au cœur de l'algorithme sont remplacées par une simple mise à jour des APP par :

$$\forall e,\, g\left(\mathrm{APP}(y_t)\right) = g\left(\mathrm{Obs}(y_t)\right) + \left[\sum_{i\neq e}(-1)^{\sharp I_e}\mathrm{Sign}\left(\prod_{i\in I_e\setminus\{t\}} g\left(\mathrm{APP}(y_i)\right)\right).\min_{i\in I_e\setminus\{t\}}\left|g\left(\mathrm{APP}(y_i)\right)\right|\right].$$

Les équations de parité faisant intervenir y_t sont obtenues de la façon suivante : chaque trinôme $1 + X^i + X^j$ avec $i < j$ multiple de $P(X)$ fournit les équations suivantes :

$$\begin{array}{rcll} y_t & = & y_{t-i} + y_{t-j} & si\ t-j \geq 0 \\ y_t & = & y_{t+i} + y_{t-j+i} & si\ t+i < N\ et\ t-j+i \geq 0 \\ y_t & = & y_{t+j} + y_{t+j-i} & si\ t+j < N \end{array}$$

L'algorithme converge quand toutes les valeurs de $g\left(\mathrm{APP}(y_t)\right)$ sont grandes, autrement dit quand la suite $(y_t)_{t\geq 0}$ n'est plus modifiée par de nouvelles itérations. Dans la pratique le nombre d'itérations est de l'ordre de 10.
Pour que l'algorithme converge, il faut avoir un nombre suffisant d'équations de parité. Pour évaluer les performances de l'algorithme, nous avons effectué des simulations pour un LFSR de longueur $L = 21$, de polynôme de rétroaction $P(X) = X^{21} + X^{14} + X^{12} + X^{11} + X^{10} + X^5 + X^3 + X^2 + 1$. Dans toutes ces simulations, nous avons utilisé un segment de longueur $N = 100\ 000$ bits de suite chiffrante.

Tout d'abord, la figure 5.7, qui représente le taux de succès de l'algorithme de décodage décrit à la table 5.9 en fonction du nombre de trinômes utilisés, met en évidence un effet de seuil. Autrement dit, il existe clairement un nombre de trinômes nécessaires au-delà duquel l'algorithme de décodage produit pratiquement toujours le bon résultat, et en deçà duquel il ne converge presque jamais. Au vu de cette propriété, nous avons considéré que le nombre de trinômes nécessaires au décodage était le nombre pour lequel le décodage produisait le bon résultat dans 90 % des cas.

Dans ce contexte, la figure 5.8 donne le nombre de trinômes nécessaires pour décoder avec l'algorithme itératif présenté à la table 5.9. On constate alors qu'avec notre algorithme, il faut de l'ordre de

$$m \gtrsim \frac{1}{3C(p_0)}$$

trinômes, où $C(p)$ est la capacité du canal binaire symétrique de probabilité d'erreur p, c'est-à-dire $C(p) = 1 + p\log_2(p) + (1-p)\log_2(1-p)$. Si $\varepsilon = p_0 - \frac{1}{2}$ est petit, on peut approximer la capacité $C(p_0)$ par

$$C(p_0) \simeq \frac{2\varepsilon^2}{\ln(2)}.$$

En effet

$$\begin{aligned} C(p_0) &= 1 + (\tfrac{1}{2}+\varepsilon)\log_2(\tfrac{1}{2}+\varepsilon) + (\tfrac{1}{2}-\varepsilon)\log_2(\tfrac{1}{2}-\varepsilon) \\ &= (\tfrac{1}{2}+\varepsilon)\log_2(1+2\varepsilon) + (\tfrac{1}{2}-\varepsilon)\log_2(1-2\varepsilon) \\ &= \tfrac{1}{\ln(2)}\left((\tfrac{1}{2}+\varepsilon)(2\varepsilon - 2\varepsilon^2 + o(\varepsilon^2)) + (\tfrac{1}{2}-\varepsilon)(-2\varepsilon - 2\varepsilon^2 + o(\varepsilon^2))\right) \\ &= \tfrac{1}{\ln(2)}(2\varepsilon^2 + o(\varepsilon^2)). \end{aligned}$$

Notons tout d'abord que ces simulations sont différentes de celles décrites dans [CT00] puisque nous nous sommes ici placés à N fixé, et nous avons fait varier le nombre de trinômes dont nous disposions. Au contraire, dans [CT00], le nombre de trinômes est entièrement déterminé par la valeur de N puisqu'il correspond au nombre de trinômes de degré au plus N multiples du polynôme de rétroaction. Ainsi, dans [CT00], le nombre d'équations par bit de suite chiffrante était égal au nombre de trinômes considérés, ce qui n'est pas le cas pour nos simulations.

Nous avons donc comparé l'algorithme de décodage classique décrit à la table 5.9 et la version « approximative » présentée dans [CT00] dans laquelle la remise à jour de la probabilité de chacun des bits y_t est effectuée en utilisant, non pas l'observation $\mathrm{Obs}(y_t)$, mais la valeur de l'APP de y_t à l'itération précédente. Contrairement aux algorithmes itératifs classiques, la version présentée dans [CT00] applique en quelque sorte un principe « d'auto-satisfaction » : chaque itération conforte la décision prise à l'itération précédente. Les résultats décrits à la figure 5.8 semblent montrer que, grâce à cette remise à jour optimiste, le décodage avec un taux de succès proche de 100 % nécessite moins de trinômes qu'avec une remise à jour classique. Par contre, nos simulations montrent également que cette procédure de remise à jour conduit à des valeurs aberrantes pour les APP quand l'algorithme de décodage ne converge pas, au contraire des décodages itératifs classiques. Identifier plus précisément l'origine des performances de l'algorithme dans [CT00] (au regard des algorithmes classiques) reste encore un problème ouvert.

On déduit alors de l'analyse précédente que le nombre N_I de bits de la sortie du brasseur nécessaires pour retrouver l'état initial est de l'ordre de

$$N_I \simeq \frac{\sqrt{\ln(2)}}{2\varepsilon\sqrt{6}} 2^{\frac{L}{2}}. \tag{5.8}$$

Il s'ensuit également que la complexité de la phase de précalcul est de l'ordre de

$$\mathcal{O}\left(\left(\frac{1}{2\varepsilon\sqrt{6}}\right) 2^{\frac{L}{2}}\right).$$

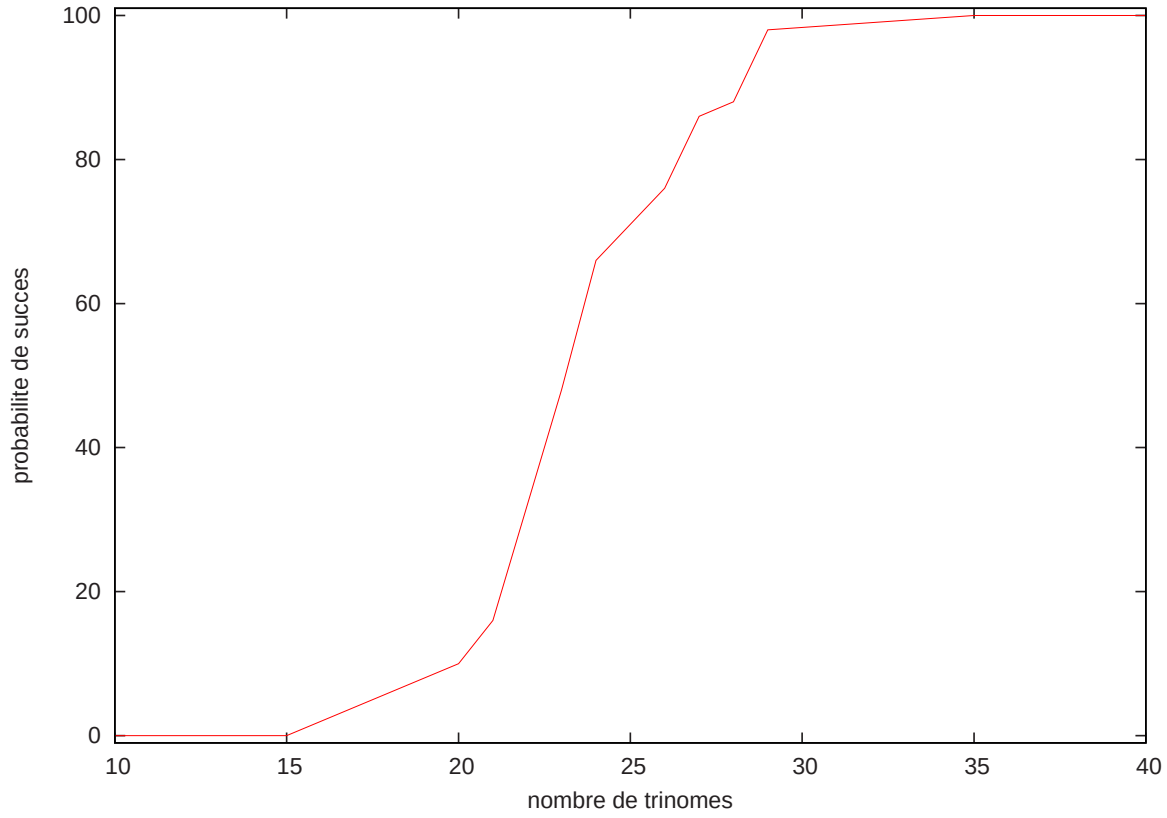

FIG. 5.7 – *Pourcentage de décodages réussis en fonction du nombre de trinômes considérés pour le polynôme de rétroaction* $X^{21} + X^{14} + X^{12} + X^{11} + X^{10} + X^5 + X^3 + X^2 + 1$.

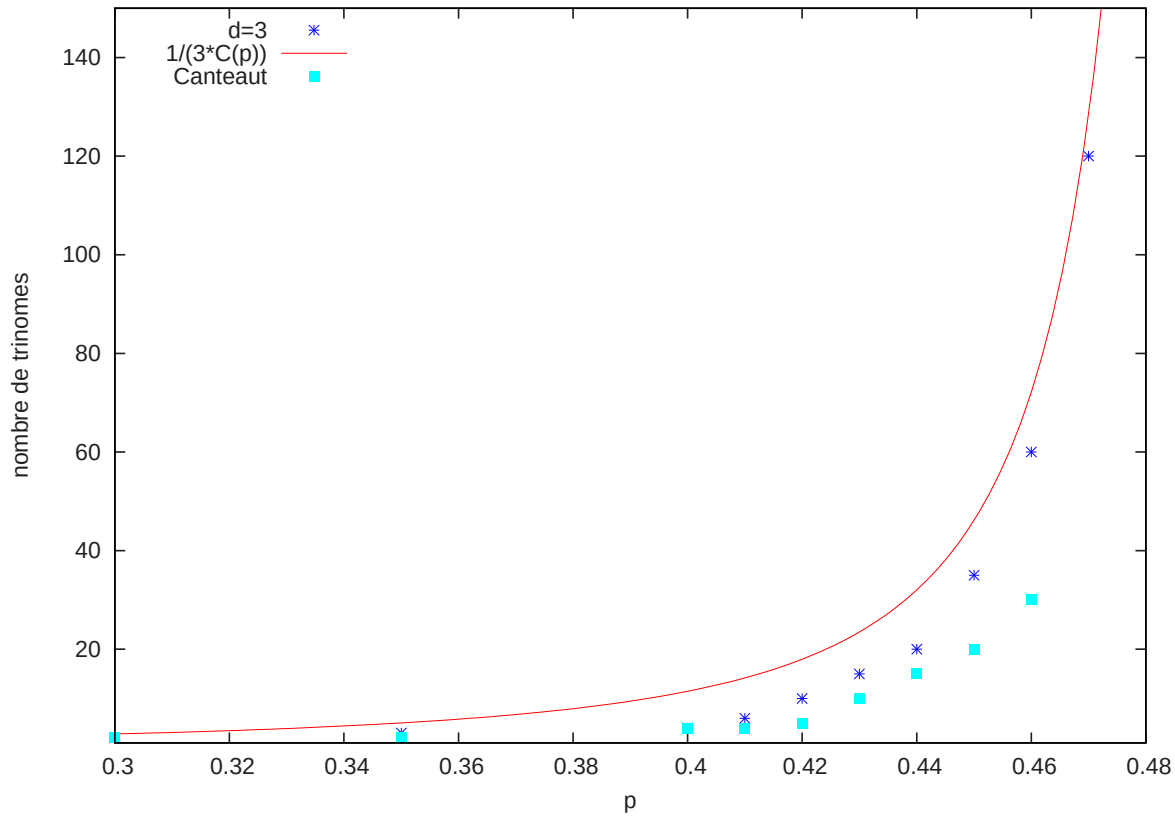

FIG. 5.8 – *Nombre de trinômes nécessaires pour reconstruire l'état initial d'un LFSR de longueur 21.*

ε	L	N_I	N_P
0.1	10	376	6 616 300
	20	12 020	6 617 700
	30	385 000	6 662 600
	40	12 500 000	8 099 160
	50	400 000 000	54 069 383
0.2	10	186	103 424
	20	5 950	104 827
	30	190 215	149 720
	40	6 100 000	1 586 289
	50	200 000 000	47 556 512
0.3	10	122	9 121
	20	3 900	10 524
	30	125 000	55 417
	40	4 000 000	1 491 986
	50	128 000 000	47 462 209

TAB. 5.10 – *Nombre de bits nécessaires pour reconstruire l'état initial (N_I) et le polynôme (N_P) en fonction de L et de ε.*

La complexité de cette phase de décodage est de l'ordre de

$$\mathcal{O}\left(\left(\frac{1}{2\varepsilon\sqrt{6}}\right)^3 2^{\frac{L}{2}}\right).$$

Sur la table ci-dessous, nous donnons les résultats de simulations effectués avec l'algorithme décrit dans [CT00] puisqu'il nécessite moins de trinômes. Dans tous les cas, nous avons retrouvé le bon état initial en à peine une seconde, nous donnons donc dans ce tableau le nombre de bits nécessaire N_I et le nombre de trinômes obtenu pour ce N_I en fonction du polynôme de rétroaction du registre.

Polynôme de rétroaction	N_I	Nombre de trinômes
$x^8 + x^4 + x^3 + x^2 + 1$	200	79
$x^{10} + x^6 + x^5 + x^3 + x^2 + x + 1$	400	81
$x^{15} + x^5 + x^4 + x^2 + 1$	2000	63
$x^{21} + x^6 + x^5 + x^2 + 1$	7000	10
$x^{29} + x^2 + 1$	5000	9

TAB. 5.11 – *Nombre de bits nécessaires pour reconstruire l'état initial (N_I) pour $\varepsilon = 0.1$.*

Remarque 5.7 *On peut généraliser toute cette phase en utilisant des polynômes multiples de P de petit poids d' à la place des trinômes comme dans [CT00], mais cela permet*

simplement de réduire le nombre de bits de sortie nécessaires ce qui n'est pas capital ici contrairement au contexte cryptographique. En effet, pour les petites valeurs de L utilisées dans la pratique ($L \leq 35$) le nombre de bits de $(y_t)_{t\geq 0}$ requis pour retrouver le polynôme de rétroaction donné par la formule (5.7) page 114 est supérieur à celui nécessité par la phase de recherche de l'état initial avec $d' = 3$ (formule (5.8) page 124).
Cette généralisation pourrait être utile dans le cas où la longueur du registre est grande (supérieure à 35). Cependant, on dispose souvent de suffisamment de bits, aussi cette généralisation n'a pas été utilisée dans le contexte de la reconstruction d'un brasseur. Nous donnons les résultats généraux de complexité dans le cas où l'on utilise des polynômes de poids d' multiples de P. La reconstruction de l'état initial du registre requiert :

$$N_I \simeq (2(d'-1)!\ln 2)^{\frac{1}{d'-1}} \left(\frac{1}{2\varepsilon}\right)^{\frac{2(d'-2)}{d'-1}} 2^{\frac{L}{d'-1}}$$

bits de la suite de sortie $(y_t)_{t\geq 0}$.
Les complexités de la phase de recherche de polynômes multiples de poids d' et de la phase de décodage sont respectivement :

$$W_I^1 = \mathcal{O}\left(\left(\frac{1}{2\varepsilon}\right)^{\frac{2(d'-2)^2}{d'-1}} 2^{\frac{(d'-2)L}{d'-1}}\right)$$

et

$$W_I^2 = \mathcal{O}\left(\left(\frac{1}{2\varepsilon}\right)^{\frac{2(d'-2)d'}{d'-1}} 2^{\frac{L}{d'-1}}\right) .$$

5.2 Brasseur autosynchronisant

Dans cette section, nous nous intéressons maintenant aux brasseurs linéaires autosynchronisants. Nous avons vu que l'élément principal d'un brasseur est un registre à décalage à rétroaction linéaire. Dans le cas du brasseur autosynchronisant, à chaque itération on ajoute modulo 2 un bit de la suite d'entrée lors du calcul de la rétroaction. Ce bit constitue la sortie à cette itération. Autrement dit, le bit de sortie du brasseur est identique au bit qui va entrer dans la première case du registre pour l'itération suivante.

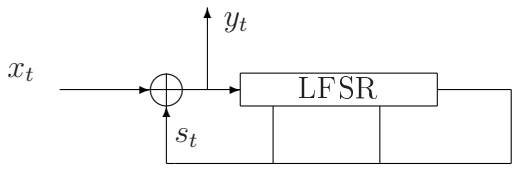

FIG. 5.9 – *Brasseur linéaire autosynchronisant.*

On notera $(x_t)_{t\geq 0}$ le train binaire en entrée du brasseur (le message), $(s_t)_{t\geq 0}$ la suite pseudo-aléatoire en sortie du LFSR et $(y_t)_{t\geq 0}$ le train binaire de sortie du brasseur. Rappelons que l'on note L la longueur du brasseur qui est généralement inconnue, on sait toutefois que, dans la majorité des applications, L ne dépasse pas $L_{\max} = 100$. Dans la suite, nous allons décrire différentes techniques pour retrouver le brasseur autosynchronisant utilisé lors de la transmission. Les diverses situations correspondent à diverses hypothèses sur la suite en entrée du brasseur. Dans le cas d'un brasseur autosynchronisant, il suffit de retrouver le polynôme de rétroaction du registre ; en effet un brasseur autosynchronisant est généralement initialisé par l'état nul. De plus, la connaissance de L bits consécutifs de $(y_t)_{t\geq 0}$ suffit à déterminer un état interne complet du LFSR. Comme dans la section précédente, on suppose ici que l'on connaît la sortie exacte d'un brasseur. Cela signifie que l'on a précédemment réussi à retrouver non seulement le code en bloc utilisé mais aussi l'encodeur qui a servi lors de la transmission. Nous allons donc montrer que la reconstruction d'un brasseur autosynchronisant dans ce contexte requiert des techniques différentes de celles décrites dans le cas synchrone.

5.2.1 Entrée connue

Nous supposons dans cette partie que l'on connaît une portion de la suite binaire d'entrée $(x_t)_{t\geq 0}$. Nous allons représenter ici les suites par des séries formelles : à la suite $(x_t)_{t\geq 0}$, on associe la série formelle :

$$X(D) = \sum_{t=0}^{+\infty} x_t D^t.$$

Si le train binaire en entrée $(x_t)_{t\geq 0}$ était identiquement nul, la sortie du brasseur correspondrait exactement à la suite produite par le LFSR. On aurait alors, en utilisant la proposition 1.6 :

$$Y(D) = \frac{Q(D)}{P(D)}$$

où P est le polynôme de rétroaction du registre et Q un polynôme de degré inférieur à celui de P et qui ne dépend que de l'état initial du registre.
Si la suite d'entrée n'est pas nulle, la relation devient alors :

$$Y(D) = \frac{Q(D)}{P(D)} + \frac{X(D)}{P(D)}.$$

On peut remarquer ici que l'algorithme de Berlekamp-Massey peut être utilisé pour retrouver P si une portion de $2L$ bits de la suite d'entrée est nulle.
Comme $\deg(Q) < L$, on en déduit que

$$\forall t \geq L,\ x_t + y_t = \sum_{i=1}^{L} c_t y_{t-i},$$

avec $P(X) = 1 + \sum_{i=1}^{L} c_i X^i$. On obtient donc un système linéaire à L inconnues qui sont les coefficients de P. On peut donc retrouver P dès que l'on connaît L bits d'entrée x_t correspondant à des instants $t \geq L$. Par ailleurs, on peut aussi remarquer que, dans la pratique, on peut ignorer la condition $t \geq L$ si le registre est initialisé à zéro. En effet, dans ce cas, la relation

$$x_t + y_t = \sum_{i=1}^{L} c_t y_{t-i}$$

est vérifiée pour tout $t \geq 0$ (avec la convention $y_t = 0$ pour $t < 0$) puisqu'on a l'égalité :

$$Y(D) = \frac{X(D)}{P(D)}.$$

Cependant, on ne connaît pas a priori la valeur de L. On doit donc résoudre un système de taille $L_{\max}$ correspondant à une borne sur la longueur du registre. La résolution de ce système pour retrouver le polynôme de rétroaction P requiert donc

$$\mathcal{O}(L_{\max}^3)$$

opérations, quelle que soit la longueur effective L du registre. Ceci ne nous semble toutefois pas être une solution optimale et nous pensons qu'il est probablement possible d'améliorer cet algorithme en détectant dans la matrice de taille $L_{\max}$, la sous-matrice de taille L.

5.2.2 Entrée produite par une source biaisée

On suppose désormais que le message d'entrée est inconnu mais statistiquement biaisé au sens déjà utilisé à la section 5.1.2, c'est-à-dire qu'on a $\Pr[x_t = 0] = \frac{1}{2} + \varepsilon$ avec $\varepsilon \neq 0$. Nous avons vu que ce cas apparaissait dans de nombreuses situations pratiques, par exemple lorsque le train binaire en entrée résulte d'un schéma classique de codage comme l'ASCII. Dans ces situations, les valeurs usuelles de biais sont de l'ordre de $\varepsilon = 0.1$ ou $\varepsilon = 0.05$. Dans le cas synchrone, nous avons développé un algorithme de reconstruction qui consiste à chercher un multiple de petit poids du polynôme de rétroaction du LFSR. Cet algorithme utilise essentiellement la possibilité de distinguer un polynôme multiple du polynôme de rétroaction d'un polynôme aléatoire grâce au biais du message d'entrée. En effet lorsqu'on considère une équation de parité associée à un multiple du polynôme de rétroaction alors cette équation est vérifiée par la suite $(s_t)_{t\geq 0}$. Elle sera donc vérifiée avec un biais pour la suite $(y_t)_{t\geq 0}$ puisqu'on a pour tout t, $y_t = x_t + s_t$, et que la suite $(x_t)_{t\geq 0}$ est biaisée. Malheureusement nous ne pouvons pas avoir la même approche dans le cas autosynchronisant : en effet la relation liant y_t à s_t fait intervenir à la fois x_t mais aussi d'autres x_i avec $i < t$ puisque l'état interne du LFSR est modifié en tenant compte de la suite $(x_t)_{t\geq 0}$. Il sera donc impossible de détecter un biais sur une équation de parité associée à un multiple du polynôme de rétroaction du registre. Cependant, nous allons voir comment cette idée peut être adaptée dans le cas d'un brasseur autosynchronisant. Au lieu

de détecter un multiple de P, on va directement chercher à étudier ce qu'il se passe pour P lui-même.

Théorème 5.8 *Soit $(y_t)_{t\geq 0}$ la sortie d'un brasseur synchrone dont la suite d'entrée $(x_t)_{t\geq 0}$ vérifie*

$$\forall t \geq 0,\ \Pr[x_t = 0] = \frac{1}{2} + \varepsilon.$$

Soit $1 + \sum_{j=1}^{d-1} X^{i_j}$ $(i_1 < \cdots < i_{d-1} = L)$ le polynôme de rétroaction du registre et soit

$$z_t = y_t + \sum_{j=1}^{d-1} y_{t-i_j},\ \ t \geq L.$$

Notons $Z_N = \sum_{t=L}^{N-1} (-1)^{z_t}$,

$$\mu = (N-L)(2\varepsilon)$$

et

$$\sigma^2 = (N-L)(1-(2\varepsilon)^2).$$

Alors la variable aléatoire

$$Z = \frac{Z_N - \mu}{\sigma}$$

tend vers une loi normale centrée réduite lorsque N tend vers l'infini.

Preuve.
On a : $\Pr[z_t = 0] = p_0 = \frac{1}{2} + \varepsilon$ car $z_t = x_t$. Toutes les variables z_t sont donc des variables aléatoires indépendantes et identiquement distribuées. D'après le théorème central limite, pour des grandes valeurs de N, on peut donc supposer que la variable aléatoire $\frac{\sum_{i=L}^{N-1} z_i - M}{V}$ suit une loi normale centrée réduite. On a :

$$M = (N-L)\Pr[z_t = 1]$$

et

$$V = (N-L)\Pr[z_t = 1]\Pr[z_t = 0].$$

Par ailleurs, on peut exprimer la variable aléatoire Z_N de la manière suivante :

$$Z_N = (N-L) - 2\sum_{i=L}^{N-1} z_i.$$

Il en découle :

$$\mu = (N-L) - (N-L)[1-2\varepsilon] = (N-L)(2\varepsilon),$$

$$\sigma^2 = 4(N-L)\left(\frac{1}{2}+\varepsilon\right)\left(\frac{1}{2}-\varepsilon\right) = (N-L)(1-4\varepsilon^2).$$

◇

Nous allons ensuite, comme dans le cas synchrone, utiliser un test d'hypothèse avec un seuil T. Il est important de remarquer que le biais à observer est ici plus important. En effet, dans le cas synchrone on devait distinguer un écart entre les deux moyennes en $(2\varepsilon)^d$ où d était le nombre de termes du multiple de P cherché alors qu'ici l'écart est en (2ε). Le test statistique nécessitera donc moins de bits de sortie puisque, naturellement, plus le biais est élevé, plus il est facile de déterminer dans quel cas on se trouve. Par contre, nous devrons tester tous les polynômes et pas seulement ceux de poids faible. Pour minimiser le temps de calcul lorsque le polynôme de rétroaction est de poids faible, on a décidé de tester les polynômes de degré inférieur à $L_{\max}$ en faisant croître leur poids. Des équations (5.2) et (5.3) de la section 5.1.2, on déduit que la valeur du seuil est :

$$T = \frac{a\left(a + b\sqrt{1-(2\varepsilon)^2}\right)}{2|\varepsilon|},$$

où $a = \phi^{-1}(1 - \frac{P_f}{2})$ et $b = -\phi^{-1}(P_n)$. Le seuil est donné par :

$$T \simeq \frac{a(a+b)}{2|\varepsilon|}. \tag{5.9}$$

Et le nombre de bits nécessaires pour la reconstruction est environ :

$$N_{\min} = L_{\max} + \frac{\left(a + b\sqrt{1-(2\varepsilon)^2}\right)^2}{(2\varepsilon)^2}.$$

$$N_{\min} \simeq L_{\max} - b(a+b) + \frac{(a+b)^2}{(2\varepsilon)^2}\ . \tag{5.10}$$

Algorithme.
Calculer $N_{\min}$ avec (5.10) et T avec (5.9)
$d = 2$
Pour tout $(d-1)$-tuplet $(i_1, \ldots, i_{d-1})$ tel que $0 < i_1 < \cdots < i_{d-1} < L_{\max}$
    $Z \leftarrow 0$
    Pour t de $L_{\max}$ à $N_{\min}$:
        $Z \leftarrow Z + (-1)^z$ avec $z = y_t + \sum_{j=1}^{d-1} y_{t-i_j}$
    Si $|Z| > T$, retourner $1 + \sum_{j=1}^{d-1} X^{i_j}$
$d \leftarrow d + 1$

TAB. 5.12 – *Algorithme de reconstruction d'un brasseur autosynchronisant.*

Les tables 5.13 et 5.14 nous donnent le temps de calcul nécessaire sur un Pentium 4 à 3.2GHz pour retrouver différents brasseurs autosynchronisants avec $L_{\max} = 100$, $P_f = 2.10^{-7}$, $P_n = 10^{-5}$ pour $\varepsilon = 0.1$ et $\varepsilon = 0.05$ respectivement.

Polynôme de rétroaction	Polynôme détecté	temps de calcul
$x^8 + x^4 + x^3 + x^2 + 1$	$x^8 + x^4 + x^3 + x^2 + 1$	13.02 s
$x^{10} + x^6 + x^5 + x^3 + x^2 + x + 1$	$x^{10} + x^6 + x^5 + x^3 + x^2 + x + 1$	5 mn 18 s
$x^{15} + x^5 + x^4 + x^2 + 1$	$x^{15} + x^5 + x^4 + x^2 + 1$	13.11 s
$x^{21} + x^6 + x^5 + x^2 + 1$	$x^{21} + x^6 + x^5 + x^2 + 1$	14.01 s
$x^{29} + x^2 + 1$	$x^{29} + x^2 + 1$	$\ll 1$ s
$x^{43} + 1$	$x^{43} + 1$	$\ll 1$ s

TAB. 5.13 – *Performance de l'algorithme de reconstruction d'un brasseur autosynchronisant pour* $\varepsilon = 0.1$.

Polynôme de rétroaction	Polynôme détecté	temps de calcul
$x^8 + x^4 + x^3 + x^2 + 1$	$x^8 + x^4 + x^3 + x^2 + 1$	47.26 s
$x^{10} + x^6 + x^5 + x^3 + x^2 + x + 1$	$x^{10} + x^6 + x^5 + x^3 + x^2 + x + 1$	19 mn 16 s
$x^{15} + x^5 + x^4 + x^2 + 1$	$x^{15} + x^5 + x^4 + x^2 + 1$	49 s
$x^{21} + x^6 + x^5 + x^2 + 1$	$x^{21} + x^6 + x^5 + x^2 + 1$	50 s
$x^{29} + x^2 + 1$	$x^{29} + x^2 + 1$	$\ll 1$ s
$x^{43} + 1$	$x^{43} + 1$	$\ll 1$ s

TAB. 5.14 – *Performance de l'algorithme de reconstruction d'un brasseur autosynchronisant pour* $\varepsilon = 0.05$.

Le nombre d'opérations effectuées par notre algorithme est égal à :

$$(N_{\min} - L_{\max}) \sum_{d=2}^{w} d \binom{d-1}{L_{\max}} \simeq \frac{w L_{\max}^{w-1}}{(w-1)!} \cdot \frac{(a+b)^2}{(2\varepsilon)^2},$$

où w est le poids du polynôme de rétroaction. Les deux tables précédentes confirment le comportement en $\mathcal{O}(\frac{1}{\varepsilon^2})$: en effet on voit qu'il faut environ 4 fois plus de temps pour retrouver un même brasseur autosynchronisant lorsque la valeur ε est divisée par 2. Cet algorithme nous permet de reconstruire les brasseurs autosynchronisants dont le polynôme de rétroaction est de poids relativement faible. Le temps de calcul augmente exponentiellement avec le poids du polynôme de rétroaction. Trouver un algorithme de reconstruction dont la complexité n'est pas directement reliée au poids du polynôme de rétroaction du LFSR reste un problème ouvert. Par ailleurs, il est important de remarquer que la technique présentée fournit directement un distingueur permettant de tester si un brasseur donné a été utilisé : si on dispose d'une liste de brasseurs autosynchronisants souvent utilisés dans

la pratique, on peut donc rapidement les tester avec le critère employé par l'algorithme que nous venons de présenter.

Chapitre 6

Reconnaissance d'un brasseur linéaire dans le contexte général

Nous allons maintenant nous intéresser au cas général qui correspond à la situation où, durant la phase de reconstruction du code en bloc linéaire, le code utilisé a été retrouvé mais on ne sait pas quel codeur a été employé lors de la transmission. Autrement dit, au lieu de connaître la sortie exacte du brasseur, on connaît seulement l'image de cette sortie par une transformation linéaire inversible par bloc dont la taille est égale à la dimension du code correcteur. Dans ce chapitre, nous allons tout d'abord montrer que l'on peut reconstruire un brasseur synchrone par des techniques algébriques si l'entrée du brasseur est nulle, puis nous allons généraliser cette méthode au cas où l'entrée du brasseur est quelconque mais connue.

6.1 Lorsque l'entrée du brasseur est nulle

Nous supposons dans cette section que l'entrée du brasseur synchrone est une suite binaire identiquement nulle. Dans ce cas, la sortie du brasseur est exactement celle du LFSR. On doit reconstruire le polynôme de rétroaction de notre LFSR en connaissant l'image de sa sortie par une transformation linéaire inversible qui agit sur des blocs de taille k, où k est la dimension du code correcteur que l'on a préalablement reconnu. Soit φ cette transformation linéaire inversible sur $\mathbf{F}_2^k$. Notons S_t le t^e bloc de k bits de la suite $(s_t)_{t\geq 0}$ produite par le LFSR, *i.e.* $S_t = (s_{tk}, s_{tk+1}, \ldots, s_{(t+1)k-1})$. Alors, φ agit sur le LFSR de la manière décrite à la figure 6.1.

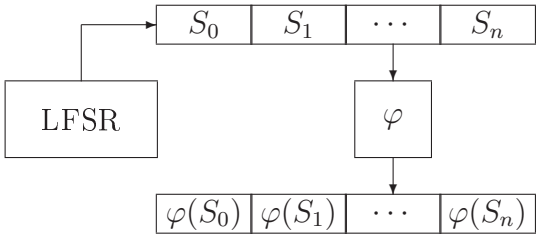

FIG. 6.1 – *Action de la transformation linéaire sur la sortie du lorsque* $(x_t)_{t\geq 0} = 0$.

Dans la suite, on supposera que $k > L$, c'est-à-dire que la longueur du brasseur est inférieure à la dimension du code correcteur qui a été utilisé lors de la transmission. On note alors u_t le mot formé par les L premiers bits d'un bloc S_t.

Soit A la matrice de transition du LFSR :

$$A = \begin{pmatrix} 0 & . & . & . & . & 0 & c_L \\ 1 & . & & & & . & c_{L-1} \\ 0 & 1 & . & & & . & . \\ . & & . & . & & . & . \\ . & & & . & . & & . \\ . & & & & . & 0 & . \\ 0 & . & . & . & 0 & 1 & c_1 \end{pmatrix}. \tag{6.1}$$

Lemme 6.1 *Soit A la matrice associée à un LFSR dont le polynôme de rétroaction est irréductible. Alors, pour tout polynôme Q :*

$$Ker(Q(A)) \neq \{0_L\} \text{ si et seulement si } Q(A) = 0,$$

où $Ker(A)$ est le noyau de la matrice A.

Preuve.
On sait que le polynôme caractéristique de la matrice A est $X^L + c_1X^{L-1} + \cdots + c_L$. Ce polynôme est le polynôme réciproque du polynôme de rétroaction du LFSR. Il est d'ailleurs souvent appelé polynôme caractéristique du LFSR.

Soit α une racine de $X^L + c_1X^{L-1} + \cdots + c_L$. Alors, comme P est irréductible, il existe une matrice C inversible telle que

$$C^{-1}AC = \begin{pmatrix} \alpha & & & & & \\ & \alpha^2 & & & 0 & \\ & & \alpha^4 & & & \\ & & & . & & \\ & & & & . & \\ & & 0 & & & \\ & & & & & \alpha^{2^{L-1}} \end{pmatrix}.$$

Il s'ensuit que:

$$C^{-1}Q(A)C = \begin{pmatrix} Q(\alpha) & & & & \\ & \ddots & & 0 & \\ & & \ddots & & \\ & & & \ddots & \\ & 0 & & \ddots & \\ & & & & Q(\alpha^{2^{L-1}}) \end{pmatrix}.$$

Si $Ker(Q(A)) \neq \{0_L\}$ alors une des valeurs propres de $Q(A)$ est nulle. Comme toutes les valeurs propres sont des conjuguées de α, cela est possible seulement si $Q(A) = 0_{L\times L}$, ce qui implique que $Ker(Q(A)) = \mathbf{F}_2^L$. ◇

Théorème 6.2 *Toute combinaison linéaire nulle des $(\varphi(S_t))_{t\geq 0}$ correspond à un polynôme annulateur de la matrice A^k. Autrement dit, pour un ensemble d'indices I,*

$$\sum_{t\in I} \varphi(S_t) = 0_k \text{ si et seulement si } \sum_{t\in I} (A^k)^t = 0.$$

Preuve.
La matrice A est clairement reliée au LFSR : si le registre est dans un état donné $x \in \mathbf{F}_2^L$, à l'étape suivante, il sera dans l'état xA. On a donc :

$$u_t = u_{t-1}A^k = u_0(A^k)^t.$$

Chaque bloc de bits S_t est une fonction linéaire des L premiers bits u_t (car le LFSR est de longueur L). Donc, il existe une matrice M telle que :

$$S_t = u_t\,(I_L|M) = u_0(A^k)^t\,(I_L|M)\,.$$

La fonction φ peut être représentée par une matrice non singulière B de taille $k \times k$. On a :

$$\varphi(S_t) = S_tB = u_0(A^k)^t\,(I_L|M)\,B.$$

On va noter 0_k l'élément nul de l'espace vectoriel $\mathbf{F}_2^k$.
Supposons qu'un ensemble de blocs, $\{S_t, t \in I\}$, satisfasse

$$\sum_{t\in I} \varphi(S_t) = 0_k.$$

Alors, on a :

$$\sum_{t\in I} u_0(A^k)^t\,(I_L|M)\,B = (0_L|0_{k-L})\,.$$

Comme B est une matrice inversible, on a aussi

$$\sum_{t\in I} u_0(A^k)^t\,(I_L|M) = (0_L|0_{k-L})\,.$$

Cela implique donc que :

$$\sum_{t\in I} u_0(A^k)^t = u_0\left(\sum_{t\in I}(A^k)^t\right) = 0_L.$$

Soit $Q(A) = \sum_{t\in I} A^t$. L'équation précédente signifie alors que

$$u_0 \in Ker(Q(A^k)).$$

Le vecteur u_0 correspond à l'état initial du LFSR et donc $u_0 \neq 0$. Ceci implique d'après le lemme 6.1 que $Q(A^k) = 0$ ou de manière équivalente que $Q(\alpha^k) = 0$ où α est une racine du polynôme de rétroaction du LFSR. Nous avons donc ainsi trouvé un polynôme annulateur de la matrice A^k. ◇

6.2 Lorsque le train binaire en entrée est connue

Maintenant, nous allons simplement montrer comment il est possible de trouver un polynôme annulateur de A^k dans le cas où la suite d'entrée du braasseur $(x_t)_{t\geq 0}$ est connue. Notons X_t le t^e bloc de k bits de la suite $(x_t)_{t\geq 0}$. En plus de la suite d'entrée, on connaît la suite $(\varphi(Y_0),\ldots,\varphi(Y_n))$ obtenue de la manière décrite à la figure 6.2.

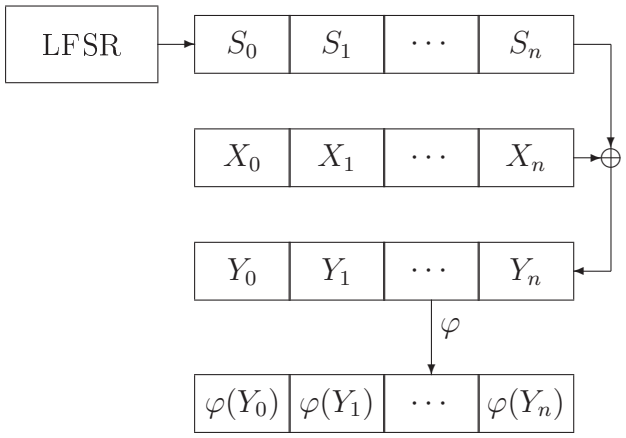

FIG. 6.2 – *Action de la transformation linéaire sur la suite de sortie du brasseur pour une entrée $(x_t)_{t\geq 0}$ donnée.*

Théorème 6.3 *La connaissance de $L+k+1$ blocs d'entrée de taille k, X_0,$X_1,\ldots,X_{L+k}$, permet de trouver un polynôme annulateur de la matrice A^k.*

Preuve.
Si on connaît $L+k+1$ blocs de taille k de la suite d'entrée X_0,$X_2,\ldots,X_{L+k}$, on peut

trouver $L+1$ relations linéairement indépendantes entre ces blocs, *i.e.* $(L+1)$ relations de la forme :

$$\sum_{\lambda \in I_j} X_\lambda = 0$$

où $I_j \subset \{0,\ldots,L+k\}$ pour $0 \leq j \leq L$. En effet, les $X_0,\ldots,X_{L+k}$ sont $L+k+1$ éléments d'un espace vectoriel de dimension k. Ces relations correspondent à $L+1$ relations :

$$\sum_{\lambda \in I_j} Y_\lambda = \sum_{\lambda \in I_j} S_\lambda.$$

Mais, on peut toujours écrire $S_t = u_t\,(I_L|M)$ où u_t désigne comme précédemment les L premiers bits de S_t. Par conséquent :

$$\begin{aligned} \sum_{\lambda \in I_j} Y_\lambda &= \sum_{\lambda \in I_j} u_0 (A^k)^\lambda\,(I_L|M) \\ &= u_0 Q_j(A^k)\,(I_L|M) \end{aligned}$$

avec $Q_j(X) = \sum_{\lambda \in I_j} X^\lambda$. Il en découle que :

$$\sum_{\lambda \in I_j} \varphi(Y_\lambda) = u_0 Q_j(A^k)\,(I_L|M)\,B.$$

On obtient donc $L+1$ éléments de $\mathbf{F}_2^L$ de la forme $u_0 Q_j(A^k)$, $0 \leq j \leq L$. Ces $L+1$ éléments sont donc obligatoirement linéairement dépendants. D'où

$$\exists (c_0,\ldots,c_L) \in \mathbf{F}_2^{L+1} \qquad \text{tel que}$$

$$c_0 u_0 Q_0(A^k) + \cdots + c_L u_0 Q_L(A^k) = 0.$$

En notant

$$Q(X) = \sum_{i=0}^{L} c_i Q_i(X),$$

on obtient :

$$u_0 Q(A^k)\,(I_L|M)\,B = 0.$$

On vient donc de montrer que connaissant les $\varphi(Y_t)$, on peut toujours trouver un polynôme Q qui vérifie la relation précédente. Exactement comme dans le premier cas où l'entrée était une suite identiquement nulle, on a donc :

$$u_0 Q(A^k) = 0.$$

Et, comme $u_0 \neq 0$, on a trouvé un polynôme annulateur de A^k. $\diamond$

6.3 Du polynôme minimal de A^k à celui de A

Par les techniques décrites précédemment, on a obtenu un polynôme Q tel que

$$Q(\alpha^k) = 0$$

pour toute racine α du polynôme caractéristique du LFSR : $X^L + c_1 X^{L-1} + \cdots + c_L$. Notre objectif est donc maintenant de trouver le polynôme minimal de α. Si Q est un polynôme irréductible alors Q correspond au polynôme minimal de α^k, M_{α^k}. Sinon, M_{α^k} est un diviseur du polynôme Q. Il nous est donc fourni en factorisant Q dans $\mathbf{F}_2[X]$; on ne conserve que les facteurs « plausibles », notamment en terme de longueur, de Q. De cette manière, on détermine donc le polynôme minimal de α^k.
Avant de rechercher le polynôme minimal de α, il faut tout d'abord retrouver la valeur de L, longueur du LFSR utilisé, car elle correspond au degré d'extension du corps dont α est un élément primitif. On peut alors calculer l'ordre de α^k, on sait que

$$ord(\alpha^k) = \frac{2^L - 1}{\text{pgcd}(2^L - 1,k)}.$$

On en déduit donc la propriété suivante :

Proposition 6.4 *La longueur L du LFSR est telle que $(2^L - 1)$ est un multiple de $ord(\alpha^k)\text{pgcd}(k,ord(\alpha^k))$ et un diviseur de $ord(\alpha^k)k$.*

Considérons le polynôme $M_{\alpha^k}(X) \circ X^k$ où $\circ$ désigne la composition. Ce polynôme est un polynôme annulateur de α et donc, le polynôme minimal de α est un diviseur de ce polynôme. On factorise donc ce polynôme dans $\mathbf{F}_2[X]/(X^{2^L} - X)$. Pour chaque facteur primitif de degré L, on construit le corps $\mathbf{F}_2^L$ et on vérifie si dans ce corps le polynôme minimal de la racine primitive puissance k est bien le polynôme minimal M_{α^k}. On va donc trouver de cette manière les $\text{pgcd}(2^L - 1,k)$ racines primitives de α^k. Dans le cas où $\text{pgcd}(2^L - 1,k) = 1$, alors on est certain que α^k a une racine k-ième unique α. Une autre solution dans ce cas consiste à construire une suite générée par le LFSR de polynôme caractéristique α^k et à la décimer [Fil00a] par l'inverse de k (k est bien inversible dans $\mathbf{F}_2^L$ puisque $\text{pgcd}(2^L - 1,k) = 1$). On applique alors l'algorithme de Berlekamp-Massey (voir section 1.2.2) à la suite décimée et on retrouve le polynôme de rétroaction du LFSR qui a été utilisé. L'algorithme permettant de retrouver les polynômes minimaux des $\text{pgcd}\left(2^L - 1,k\right)$ racines primitives de α^k est résumé à la table 6.1.

Algorithme.
Entrée : un polynôme annulateur de α^k : Q(X)
Sortie : les polynômes minimaux des racines de α^k

- Trouver $M_{\alpha^k}(X)$: si $Q(X)$ est irréductible alors $M_{\alpha^k}(X) = Q(X)$, sinon on trouve $M_{\alpha^k}(X)$ en factorisant dans $\mathbf{F}_2[X]$.
- Calculer L à l'aide de la proposition 6.4.
- Calculer les racines k-ièmes de α^k en factorisant le polynôme $M_{\alpha^k}(X) \circ X^k$, il y en a pgcd($k$,$2^L - 1$). Si pgcd($k$,$2^L - 1$) $= 1$, on peut aussi décimer par l'inverse de k dans $\mathbf{F}_2^L$ pour trouver le polynôme de rétroaction du LFSR utilisé.

TAB. 6.1 – *Algorithme pour trouver les polynômes minimaux des racines de α^k.*

Nos techniques nous permettent donc de trouver les polynômes minimaux des racines k-ièmes de α^k si l'on dispose de $(L + k + 1) \times k$ bits consécutifs de la suite entrante ou si la suite d'entrée a un segment de longueur $(k + 1) \times k$ identiquement nul.

Il reste maintenant à déterminer, dans le cas où pgcd(2^L-1,k) > 1, à laquelle des racines k-ièmes de α^k correspond au polynôme minimal utilisé par le brasseur. Tout d'abord, nous allons démontrer que les ensembles des suites engendrées par les pgcd($2^L - 1$,k) polynômes correspondant aux racines primitives de α^k sont identiques. Pour cela, nous aurons notamment besoin du lemme suivant dont la preuve [Her75] repose sur la mise sous forme normale de Frobenius des matrices A et B.

Lemme 6.5 [Her75, p 312] *Supposons que deux matrices A et B à coefficients dans un corps K soient semblables dans une extension de K. Alors elles sont déjà semblables dans K.*

Théorème 6.6 *Soient k et L deux entiers.*
Soient P_1 et P_2 deux polynômes primitifs qui sont les polynômes minimaux de deux racines k-ièmes d'un certain élément α^k avec α primitif dans $\mathbf{F}_2^L$. Notons $s_i(u_0)$ la suite engendrée par le LFSR de polynôme caractéristique P_i et d'état initial u_0. Alors, il existe une transformation linéaire inversible par bloc de taille k qui permet de passer de $s_1(u_0)$ à $s_2(v_0)$, i.e. *si l'on note $S_t^{(i)}(u_0)$ le t-ième bloc de taille k de la suite $s_i(u_0)$ alors $\exists \varphi : \mathbf{F}_2^k \to \mathbf{F}_2^k$ linéaire inversible telle que*

$$\left\{ \varphi \left(S_t^{(1)}(u_0) \right) | u_0 \in \mathbf{F}_2^L \right\} = \left\{ S_t^{(2)}(v_0) | v_0 \in \mathbf{F}_2^L \right\}.$$

De plus, il existe une matrice P de taille L telle que :
- *$v_0 = u_0 P$ et*
- *la matrice B de taille k associée à φ est*

$$B = \left(\begin{array}{c|c} P & M_1 + PM_2 \\ \hline 0 & I_{k-L} \end{array} \right),$$

où M_i est une matrice qui ne dépend que du polynôme P_i.

Preuve.
Prenons deux racines k-ièmes de α^k, et notons P_1 et P_2 leurs polynômes minimaux. On notera $S_t^{(i)}$ le t-ième bloc de k bits de la suite produite par le LFSR de polynôme de rétroaction le polynôme réciproque de P_i. On note comme précédemment A_i la matrice caractéristique du LFSR de polynôme caractéristique P_i. P_1 et P_2 étant deux polynômes irréductibles, on peut diagonaliser A_1^k et A_2^k. Ces deux matrices A_1^k et A_2^k ont mêmes valeurs propres, elles sont donc semblables sur $\mathbf{F}_2^L$. D'après le lemme 6.5, elles sont alors semblables sur $\mathbf{F}_2$: il existe une matrice inversible P à coefficients dans $\mathbf{F}_2$ telle que

$$A_2^k = P^{-1} A_1^k P.$$

Notons $\widetilde{M_i}$ la matrice définie par $S_t^{(i)} = u_t^{(i)} \widetilde{M_i}$ où $u_t^{(i)}$ désigne les L premiers bits du bloc $S_t^{(i)}$:

$$\widetilde{M_i} = \left(\begin{array}{c|c} I_L & M_i \end{array} \right).$$

Considérons alors la matrice inversible

$$B = \left(\begin{array}{c|c} P & M_1 + PM_2 \\ \hline 0 & I_{k-L} \end{array} \right).$$

On a alors :

$$P^{-1}\widetilde{M_1}B = \left(\begin{array}{c|c} P^{-1} & P^{-1}M_1 \end{array} \right) \left(\begin{array}{c|c} P & M_1 + PM_2 \\ \hline 0 & I_{k-L} \end{array} \right) = \widetilde{M_2}.$$

Intéressons-nous maintenant à un bloc de taille k de la suite produite par le LFSR de polynôme caractéristique P_1 à partir de l'état initial $u_0 \in \mathbf{F}_2^L$. On a :

$$\varphi\left(S_t^{(1)}(u_0)\right) = u_t^{(1)} \widetilde{M_1} B = u_0 (A_1^k)^t \widetilde{M_1} B = u_0 P (A_2^k)^t P^{-1} \widetilde{M_1} B$$

et donc

$$\varphi\left(S_t^{(1)}(u_0)\right) = S_t^{(2)}(u_0 P).$$

La transformation linéaire de $\mathbf{F}_2^k$ associée à la matrice B nous permet donc de passer de $\left(s_t^{(1)}(u_0)\right)_{t\geq 0}$ à $\left(s_t^{(2)}(v_0)\right)_{t\geq 0}$.

◇

Ce théorème nous montre donc que, pour chaque racine k-ième primitive β α^k, il existe un état initial tel que le LFSR de polynôme caractéristique M_β produit la même suite pseudo-aléatoire que le LFSR de caractéristique M_α et d'état initial fixé.

Pour retrouver le LFSR qui a été utilisé lors de la transmission, c'est-à-dire pour déterminer lequel des polynômes M_β a été effectivement employé, il est donc nécessaire d'utiliser une information sur l'entrée du brasseur.

On va utiliser la relation suivante :

$$Y_t = (X_t \oplus S_t)B. \tag{6.2}$$

On se fixe un polynôme de rétroaction qui est le polynôme réciproque du polynôme minimal M_β d'une racine k-ième de α^k. Pour déterminer l'état initial du LFSR et la matrice B, on peut employer deux méthodes différentes :

1. On fait une recherche exhaustive sur les $(2^L - 1)$ états initiaux possibles du LFSR. Pour chacun d'entre eux, on doit alors résoudre un système d'équations linéaires à k^2 inconnues qui sont les coefficients de la matrice B. Chaque couple (X_t,Y_t) nous fournit k équations (6.2). On peut donc résoudre ce problème lorsqu'on connaît k couples (X_t,Y_t) distincts et la résolution du système s'effectue en $\mathcal{O}(k^6)$ opérations. La complexité moyenne pour retrouver l'état initial du registre et la matrice B est donc de l'ordre de $\mathcal{O}\left(k^6 2^L\right)$.
2. On considère directement le système associé à (6.2). C'est un système quadratique avec au plus k^2L inconnues de degré 2 qui résultent du produit X_tB. En linéarisant, on se ramène donc à un système d'équations à au plus $k^2L + k^2 + L$ inconnues. On peut donc résoudre ce système lorsqu'on connaît $kL+k+1$ couples (X_t,Y_t) distincts. La complexité moyenne pour retrouver l'état initial et la matrice B est donc de l'ordre de $\mathcal{O}(k^6L^3)$.

En effectuant ceci pour chacun des $\text{pgcd}(k,2^L-1)$ polynômes de rétroaction possibles, on arrive donc à retrouver le brasseur synchrone utilisé en connaissant $kL+k+1$ couples (X_t,Y_t) avec une complexité moyenne de

$$\mathcal{O}\left(\text{pgcd}(2^L-1,k)k^6L^3\right) \text{ opérations.}$$

Algorithme.
Entrée : un polynôme annulateur de α^k : Q(X)
Sortie : le polynôme de rétroaction du LFSR utilisé

- trouver les polynômes minimaux des racines de α^k en utilisant l'algorithme décrit dans la table 6.1. Si $\text{pgcd}(k,2^L-1) = 1$, on peut aussi retrouver le polynôme de rétroaction du LFSR par décimation
- si $\text{pgcd}(2^L-1,k) > 1$, pour chaque polynôme de rétroaction possible, on résout le système associé à (6.2).

TAB. 6.2 – *Algorithme pour retrouver le polynôme de rétroaction du LFSR.*

6.4 Conclusion

Les techniques présentées dans les sections précédentes permettent donc de retrouver le polynôme de rétroaction utilisé et la transformation linéaire à partir de la connaissance

de $k^2L + k^2 + L$ bits de la suite d'entrée. Lorsqu'on suppose que la suite d'entrée est inconnue mais biaisée, nous n'avons pas trouvé de techniques permettant de reconstruire un brasseur synchrone. Dans cette situation, la conception d'un distingueur pour tester si un brasseur donné a été utilisé lors de la transmission reste même un problème ouvert. Dans le cas d'un brasseur autosynchronisant, si une large portion s'annule alors on peut par la technique présentée dans la section 6.1 page 135 retrouver un polynôme annulateur de α^k. Si $\text{pgcd}(k,2^L-1) = 1$, alors on peut retrouver le polynôme de rétroaction utilisé ; sinon on retrouve les différents polynômes de rétroaction possibles mais retrouver le polynôme de rétroaction qui a été effectivement utilisé reste un problème ouvert. De plus, si l'on connaît simplement des bits d'entrée sans que ceux-ci soient nuls, on ne connaît pas de technique pour retrouver le polynôme de rétroaction utilisé ; on ne peut pas utiliser de techniques analogues à celles employées dans le cas synchrone car il est difficile d'écrire des relations entre les blocs de sortie du brasseur autosynchronisant puisqu'un bit de sortie du brasseur dépend de tous les précédents. Lorsqu'on suppose que la suite d'entrée est inconnue mais biaisée, nous n'avons pas trouvé la moindre technique permettant de reconstruire un brasseur autosynchronisant. Dans cette situation, comme dans le cas synchrone, la conception d'un distingueur reste même un problème ouvert.

Conclusions et perspectives

Dans cette thèse, nous avons abordé le problème général de la reconnaissance des spécifications d'un système de communication classique. Les deux grandes parties portent donc sur la reconnaissance des deux principaux maillons d'une chaîne de communication : le code correcteur en blocs et le brasseur linéaire. Nous avons présenté différents algorithmes permettant de traiter de nombreux cas. Cependant il reste encore des situations dans lesquelles nous ne pouvons pas encore apporter de solutions satisfaisantes mais uniquement quelques pistes de recherche que nous esquissons ici.

Reconstruction d'un code linéaire en blocs

La première partie a abordé le problème de la reconnaissance d'un code en blocs. Nous avons présenté l'algorithme proposé par Valembois pour résoudre ce problème, puis nous avons montré que le test statistique pour distinguer des mots du code dual de mots aléatoires n'était pas totalement approprié au contexte. Nous avons alors proposé un nouveau test statistique et mis en évidence, par des simulations, l'influence des différents paramètres. L'algorithme utilisant simplement un test statistique s'avérant inefficace dès que la longueur devient trop importante et/ou le taux d'erreur trop élevé, nous avons présenté un algorithme inspiré des techniques classiques de décodage itératif. Cet algorithme, complètement nouveau, permet de décoder à partir d'équations dont certaines ne sont pas forcément des équations de parité du code. Nous en avons déduit une nouvelle méthode de reconstruction d'un code linéaire en blocs fondé sur cet algorithme de décodage. Les simulations ont montré que cet algorithme était particulièrement efficace pour les codes LDPC.

Notre nouvel algorithme de décodage, qui permet de décoder même si certaines équations ne sont pas des équations de parité du code, a certainement d'autres applications possibles dans d'autres contextes, notamment en cryptanalyse. Une autre perspective importante dans le contexte de la reconnaissance de code est de tenter d'adapter notre algorithme au cas où l'on dispose d'information souple. En effet, nous nous sommes limités dans ce manuscrit au cas du canal binaire symétrique, pourtant la démodulation du signal intercepté fournit souvent une information souple et il serait donc très intéressant d'adapter notre algorithme à ce cas, ce qui semble tout à fait envisageable. Un autre problème ouvert est de construire un distingueur qui permettrait de tester si un code donné a utilisé lors de la communication. Au premier abord, nous aurions envie tester les mots du dual

de ce code à l'aide de notre test statistique, peut-être existe-t-il d'autres techniques plus efficaces.

Reconstruction d'un brasseur linéaire

Dans la seconde partie de cette thèse, nous avons abordé le problème de la reconstruction d'un brasseur linéaire. Dans un premier temps, nous avons présenté différentes techniques de reconstruction selon le type de brasseur (synchrone ou auto-synchronisant) et le type d'hypothèses envisagées sur la suite en entrée du système de communication lorsque l'on connaît la sortie exacte du brasseur. Nous avons proposé dans tous les cas des algorithmes permettant de reconstruire le brasseur linéaire. Dans le cas synchrone, lorsque l'on suppose que la suite d'entrée du système est inconnue mais produite par une source binaire biaisée, nous avons présenté un algorithme de reconstruction du polynôme de rétroaction efficace. Cet algorithme repose sur un test statistique permettant de détecter des multiples creux du polynôme de rétroaction du registre, principe déjà employé par A. Canteaut et E. Filiol dans un autre contexte [CF00]. Toutefois nous avons donné ici une analyse exacte de la distribution statistique intervenant dans ce test, qui ne nécessite pas l'hypothèse erronée d'indépendance des variables aléatoires qui composent la sortie du LFSR. Nous pensons d'ailleurs que le même type d'analyse pourrait être mené dans le contexte de la cryptanalyse de chiffrement à flot pour éviter cette hypothèse d'indépendance dans certaines attaques par distingueur [MH04, Gol96, EJ05, LZGB03, Can06].

Pour la reconstruction de l'état initial du registre, nous avons mis en évidence les différences de performances entre un algorithme de décodage itératif classique et celui présenté dans [CT00] pour les attaques par corrélation rapide. Les simulations ont montré que l'algorithme utilisé dans [CT00] nécessitait moins de trinômes qu'un algorithme classique pour converger. Cependant, lorsque l'algorithme proposé dans [CT00] ne converge pas, les valeurs des fiabilités sur les bits sont inutilisables ce qui n'est sans doute pas le cas pour l'algorithme de décodage itératif classique et il pourrait dons être intéressant de voir si ces valeurs ne peuvent pas être exploitées par un autre algorithme de décodage dans le cas où l'algorithme n'a pas convergé. De plus, il semble que lorsqu'on approxime les APP partielles par les APP globales dans l'approximation de l'algorithme de Gallager, on fasse aussi une sorte d'« auto-satisfaction » et qu'on puisse donc diminuer ce phénomène en ajoutant un facteur inférieur à 1 lorsqu'on calcule l'approximation des APP partielles. Dans le cas auto-synchronisant, lorsqu'on fait la même hypothèse sur la suite entrante, nous avons présenté un algorithme qui nous permet de reconstruire efficacement les brasseurs auto-synchronisants dont le polynôme de rétroaction est de poids relativement faible (ce qui est souvent le cas dans les applications pratiques). Mais, le temps de calcul de cet algorithme augmente exponentiellement avec le poids du polynôme de rétroaction. Trouver un algorithme de reconstruction dont la complexité n'est pas directement reliée au poids du polynôme de rétroaction du LFSR reste donc un problème ouvert.

Ensuite, nous nous sommes intéressés au cas général. Pour un brasseur linéaire synchrone, dans le cas où l'on connaît une portion de la suite en entrée, nous avons présenté

un algorithme de reconstruction très efficace. Lorsqu'on suppose que la suite d'entrée est inconnue mais biaisée, nous n'avons pas trouvé de techniques de reconstruction. Dans cette situation, la conception d'un distingueur pour tester si un brasseur donné a été utilisé lors de la transmission reste même un problème ouvert. Dans le cas d'un brasseur autosynchronisant, si une large portion de la suite en entrée s'annule, alors on a montré qu'il n'y a que quelques possibilités pour le polynôme de rétroaction, mais retrouver celui qui a été effectivement utilisé lors de la transmission reste un problème ouvert. De plus, le problème semble encore plus difficile si l'on connaît simplement des bits d'entrée sans que ceux-ci soient nuls, ou si l'on suppose que la suite d'entrée est inconnue mais biaisée. Dans cette situation, comme dans le cas synchrone, la conception d'un distingueur reste également un problème ouvert.

Bibliographie

[Bar05] J. Barbier. « Reconstruction of turbo-code encoders ». Dans *SPIE Defense and Security Symposium, Space communications technologies conference*, Orlando, 2005.

[Ber68] E.R. Berlekamp. *Algebraic coding theory.* McGraw-Hill, 1968.

[Can96] A. Canteaut. « *Attaques de cryptosystèmes à mots de poids faible et construction de fonctions t-résilientes* ». Thèse de doctorat, Université Paris-VI, 1996.

[Can06] A. Canteaut. « *Analyse et conception de chiffrements à clef secrète* ». Mémoire d'habilitation à diriger des recherches, Université Paris 6, 2006.

[CC98] A. Canteaut et F. Chabaud. « A new algorithm for finding minimum-weight words in a linear code: application to primitive narrow-sense BCH codes of length 511 ». *IEEE Transaction on Information Theory*, 44(1):367–378, janvier 1998.

[CF00] A. Canteaut et E. Filiol. « Ciphertext Only Reconstruction of Stream Ciphers based on Combination Generators ». Dans *Fast Software Encryption 2000*, volume 1978 de *Lecture Notes in Computer Science*, pages 165–180. Springer-Verlag, 2000.

[CJM02] P. Chose, A. Joux et M. Mitton. « Fast correlation attacks: an algorithmic point of view ». Dans *Advances in Cryptology - EUROCRYPT 2002*, volume 2332 de *Lecture Notes in Computer Science*, pages 209–221. Springer-Verlag, 2002.

[CJS00] V. Chepyshov, T. Johansson et B. Smeets. « A simple algorithm for fast correlation attacks on stream ciphers ». Dans *Fast Software Encryption 2000*, volume 1978 de *Lecture Notes in Computer Science*, pages 181–195. Springer-Verlag, 2000.

[Clu] M. Cluzeau. « Reconstruction of a Linear Scrambler ». *IEEE Transactions on Computers.* soumis.

[Clu03] M. Cluzeau. « Reconstruction d'un brasseur linéaire ». Rapport de stage de DEA, Université de Limoges, 2003.

[Clu04] M. Cluzeau. « Reconstruction of a linear scrambler ». Dans *Proceedings of the 2004 IEEE International Symposium on Information Theory - ISIT 2004*, page 230, Chicago, USA, 2004.

[Clu05] M. Cluzeau. « Reconstruction d'un brasseur linéaire ». Dans *Ecole de Jeunes Chercheurs en Algorithmique et Calcul Formel - EJC 2005*, Montpellier, avril 2005.

[Clu06a] M. Cluzeau. « Block code reconstruction using iterative decoding techniques ». Dans *Proceedings of the 2006 IEEE International Symposium on Information Theory - ISIT 2006*, pages 2269–2273, Seattle, USA, 2006.

[Clu06b] M. Cluzeau. « Reconnaissance d'un code linéaire en bloc en utilisant un algorithme de décodage itératif ». Dans *Journées Codage et Cryptographie*, Eymoutiers, octobre 2006.

[CS04a] M. Cluzeau et N. Sendrier. « Reconstruction d'un brasseur linéaire ». Rapport technique d'avancement - convention DGA 02 60 65 095 450 75 01, février 2004. 75 pages.

[CS04b] M. Cluzeau et N. Sendrier. « Reconstruction d'un schéma de codage ». Rapport technique final - convention DGA 02 60 65 095 450 75 01, octobre 2004. 102 pages.

[CT00] A. Canteaut et M. Trabbia. « Improved fast correlation attacks using parity-check equations of weight 4 and 5 ». Dans *Advances in Cryptology - EUROCRYPT 2000*, volume 1807 de *Lecture Notes in Computer Science*, pages 573–588. Springer-Verlag, 2000.

[EJ05] H. Englund et T. Johansson. « A new simple technique to attack filter generators and related ciphers ». Dans *Selected Areas in Cryptography - SAC 2004*, volume 3357 de *Lecture Notes in Computer Science*, pages 39–53. Springer-Verlag, 2005.

[Fil97] É. Filiol. « Reconstruction of Convolutional Encoders over $GF(q)$ ». Dans *IMA Conference on Cryptography and Coding*, volume 1355 de *Lecture Notes in Computer Science*, pages 100–110. Springer-Verlag, 1997.

[Fil00a] É. Filiol. « Decimation attack of stream ciphers ». Dans *INDOCRYPT 2000*, volume 1977 de *Lecture Notes in Computer Science*, pages 31–42. Springer-Verlag, 2000.

[Fil00b] É. Filiol. « Reconstruction of Punctured Convolutional Encoders ». Dans *International Symposium on Information Theory and Applications*, pages 4–7. SITA and IEICE, 2000.

[Fil01] É. Filiol. « *Techniques de reconstruction en cryptologie et théorie des codes* ». Thèse de doctorat, École Polytechnique, 2001.

[Gal63] R.G. Gallager. *Low Density Parity Check Codes*. MIT Press, 1963.

[Gol96] J. Golic. « On the Security of Nonlinear Filter Generators ». Dans *Fast Software Encryption 1996*, volume 1039 de *Lecture Notes in Computer Science*, pages 173–188. Springer-Verlag, 1996.

[Her75] I.N. Herstein. *Topics in Algebra*. John Wiley & Sons, New York, 2ème édition, 1975.

[HOP96] J. Hagenauer, E. Offer et L. Papke. « Iterative decoding of binary block and convolutional codes ». *IEEE Transactions on Information Theory*, 42(2):429–445, 1996.

[JJ99a] T. Johansson et F. Jönsson. « Fast correlation attacks based on turbo code techniques ». Dans *Advances in Cryptology - CRYPTO'99*, volume 1666 de *Lecture Notes in Computer Science*, pages 181–197. Springer-Verlag, 1999.

[JJ99b] T. Johansson et F. Jönsson. « Improved fast correlation attack on stream ciphers via convolutional codes ». Dans *Advances in Cryptology - EUROCRYPT'99*, volume 1592 de *Lecture Notes in Computer Science*, pages 347–362. Springer-Verlag, 1999.

[JJ00] T. Johansson et F. Jönsson. « Fast correlation attacks through reconstruction of linear polynomials ». Dans *Advances in Cryptology - CRYPTO 2000*, volume 1880 de *Lecture Notes in Computer Science*, pages 300–315. Springer-Verlag, 2000.

[Jön02] F. Jönsson. « *Some results on fast correlation attacks* ». Thèse de doctorat, Lund University, Sweden, 2002.

[LB88] P.J. Lee et E.F. Brickell. « An observation on the security of McEliece's public-key cryptosystem ». Dans *Advances in Cryptology - EUROCRYPT'88*, volume 330 de *Lecture Notes in Computer Science*, pages 275–280. Springer-Verlag, 1988.

[LC83] S. Lin et D. J. Costello Jr.. *Error Control Coding.* Pearson Prentice Hall, 2ème édition, 1983.

[Leo88] J.S. Leon. « A probabilistic algorithm for computing minimum weights of large error-correcting codes ». *IEEE Transaction on Information Theory*, 34(5):1354–1359, 1988.

[Lev04] S. Leveiller. « *Quelques algorithmes de cryptanalyse du registre filtré* ». Thèse de doctorat, École Nationale Supérieure des Télécommunications, 2004.

[LM92] E. A. Lee et D. G. Messerschmitt. *Digital Communication.* Kluwer Academic Publishers, Boston, 4ème édition, 1992.

[LMS01] M.G. Luby, M. Mitzenmacher et M.A. Shokrollahi. « Analysis of Random Processes via And-Or Tree Evaluation ». Dans *SODA: ACM-SIAM Symposium on Discrete Algorithms*, pages 364–373, 2001.

[LN83] R. Lidl et H. Niederreiter. *Finite fields.* Cambridge University Press, 1983.

[LZGB03] S. Leveiller, G. Zémor, P. Guillot et J. Boutros. « A new cryptanalytic attack for PN-generators filtered by a Boolean function ». Dans *Selected areas in cryptography - SAC 2002*, volume 2595 de *Lecture Notes in Computer Science*, pages 232–249. Springer-Verlag, 2003.

[Mas69] J.L. Massey. « Shift-register synthesis and BCH decoding ». *IEEE Transactions on Information Theory*, 15:122–127, 1969.

[McE78] R.J. McEliece. « A public-key cryptosystem based on algebraic coding theory ». *JPL DSN Progress Report*, pages 114–116, 1978.

[MH04] H. Molland et T. Helleseth. « An improved correlation attack against irregular clocked and filtered generator ». Dans *Advances in Cryptology - CRYPTO 2004*, volume 3152 de *Lecture Notes in Computer Science*, pages 373–389. Springer-Verlag, 2004.

[MS77] F.J. MacWilliams et N.J.A. Sloane. *The theory of error-correcting codes.* North-Holland, 1977.

[MS89] W. Meier et O. Staffelbach. « Fast correlation attack on certain stream ciphers ». *Journal of Cryptology*, 1(3):159–176, 1989.

[Ric95] B. Rice. « Determining the parameters of a rate $\frac{1}{n}$ convolutional encoder over $GF(q)$ ». Dans *Third International Conference on Finite Fields and Applications*, Glasgow, 1995.

[Rou70] E. Roubine. *Introduction à la théorie de la communication.* Masson, Paris, 3ème édition, 1970.

[RU01] T. Richardson et R. Urbanke. « The Capacity of Low-Density Parity-Check Codes Under Message-Passing Decoding ». *IEEE Transactions on Information Theory*, 47(2):599–618, 2001.

[Sha48] C. E. Shannon. « A Mathematical Theory of Communication ». *Bell System Technical Journal*, 27:379–423, 623–656, 1948.

[Ste89] J. Stern. « A method for finding codewords of small weight ». Dans *Coding Theory and Applications*, volume 388 de *Lecture Notes in Computer Science*, pages 106–113. Springer-Verlag, 1989.

[Til05] H. C. A. Van Tilborg, éditeur. *Encyclopedia of cryptography and security.* Springer, 2005.

[Val00] A. Valembois. « *Décodage, Détection et Reconnaissance des Codes Linéaires Binaires* ». Thèse de doctorat, Université de Limoges, 2000.

[Val01] A. Valembois. « Detection and recognition of a binary linear code ». *Discrete Applied Mathematics*, 111:199–218, 2001.

[Var97] A. Vardy. « Algorithmic complexity in coding theory and the minimum distance problem ». Dans *STOC '97: Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 92–109, New York, NY, USA, 1997. ACM Press.

Index

Liste des tableaux

Table des figures

Table des matières