



263 avenue du Général Leclerc  
CS 74205  
35042 RENNES CEDEX



Campus de Beaulieu . F  
35 042 Rennes Cedex

---

# Codage de sources distribué : codage symétrique et adaptatif en débit de sources corrélées

---

*Stage effectué par*  
**Velotiaray TOTO-ZARASOA**  
Diplôme d'ingénieur de l'IFSIC  
Master Recherche Signal / TRAMP / Image  
Année 2006-2007

*Sous la responsabilité de*  
**Christine GUILLEMOT** et **Aline ROUMY** (*IRISA*)  
**Gérard FAUCON** (*SPM, IFSIC*)

Septembre 2007



## Remerciements

Avant d'aller plus loin dans la présentation de cette expérience professionnelle dans le domaine de la recherche à l'**IRISA**, il me semble indispensable d'adresser mes plus sincères remerciements aux personnes sans lesquelles je n'aurais pas pu soutenir aujourd'hui.

Je remercie donc en premier lieu mes deux responsables de stage à l'*IRISA* et dans l'équipe *TEMICS*, *Dr Christine Guillemot* et *Dr Aline Roumy*. Christine Guillemot est directeur des recherches de l'équipe, au sein de l'INRIA ; Aline Roumy est chercheur dans le domaine du traitement du signal, de la théorie du codage et de la théorie de l'information. Elles ont eu de la patience et leurs conseils m'ont énormément aidé dans mes travaux.

Merci également à *Dr Claude Labit*, directeur de l'IRISA, qui m'a donné l'opportunité d'effectuer mon stage dans l'enceinte de l'IRISA.

Merci à *Dr Khaled Lajnef*, chercheur de l'équipe qui travaille également dans le domaine du codage distribué. Il m'a beaucoup aidé à comprendre le codage distribué dans ses détails.

Merci enfin à *toute l'équipe TEMICS*, pour leurs réponses à mes questions dans le cadre de mes travaux ainsi que de m'avoir accueilli comme membre de l'équipe.

*Velotiaray Toto-Zaraso.*



## Résumé

La compression sans perte et distribuée d'informations provenant de plusieurs sources corrélées a fait ses débuts en 1973 quand Dr D. Slepian et Dr J. K. Wolf démontrent que la limite d'une compression disjointe, avec restitution conjointe, n'est autre que l'entropie conjointe de ces sources. Ce qui est remarquable est que cette limite est la même que lors d'une compression conjointe. La mise en œuvre théorique passe par un codage aléatoire, ce qui est impossible à implémenter en pratique. On montre que le codage de canal permet d'atteindre cette limite : les données étant corrélées, l'une est une version bruitée de l'autre.

Récemment, Li *et al.* et D. Varodayan *et al.* ont respectivement proposé un codage symétrique, c'est à dire permettant à chacune des sources de transmettre des quantités différentes d'information tant qu'on restitue les sources sans erreur, et un codage adaptatif en débit, laissant au système la possibilité de s'adapter aux différents niveaux de corrélation entre les sources. Nous nous aidons de ces deux travaux pour concevoir notre algorithme.

La combinaison de ces deux méthodes commence par le décodage de la différence entre les deux séquences sources, à partir de leurs versions compressées au niveau des entropies conditionnelles : leurs syndromes. Pour le codage LDPC, un algorithme de A. Liveris *et al.* permet un tel décodage : la propagation de croyance. En ce qui concerne le codage convolutif, un algorithme permettant également de retrouver un mot à partir de sa version bruitée a été mise au point par l'équipe TEMICS. Plus les sources sont corrélées moins les syndromes doivent être grands, d'où l'adaptation en débit.

D'autre part, on transmet au décodeur, ainsi que les syndromes, des parties complémentaires des sources ; recouvrer la différence entre les sources permet donc de compléter ces parties. Enfin, on finalise le décodage en s'aidant de la représentation matricielle des différents codes. La taille des parties systématiques transmises varie d'une source à l'autre pour permettre un codage symétrique.

## Summary

Lossless distributed compression of correlated data from separate sources was first studied in 1973 when Dr D. Slepian and Dr J. K. Wolf stated that, as long as joint decoding is performed, the limit of a disjoint compression is the joint entropy of the sources. What is surprising is that the limit is the same as when joint coding of the sources is executed. Theoretical implementation uses random coding, which is not practically feasible. It has been shown that channel coding is one means to reach that bound : the data being correlated, one of the sources can be viewed as a noise-corrupted version of another.

Lately, Li *et al.* and D. Varodayan *et al.* respectively suggested symmetric coding, that is to let the sources free to send different amount of data as long as error free recovering of the sources can be done, and rate-adaptive coding, which leaves the system the possibility to adjust to different levels of correlation between the sources. We make use of these two works to design our algorithm.

While combining both these frameworks, one must begin with decoding the difference pattern between the sources, using their conditional-entropy-level compressed versions : their syndromes. For LDPC coding, the algorithm which completes such a decoding was discovered by A. Liveris *et al.* : belief propagation. When using convolutional codes, a similar algorithm decodes a word from its erroneous version and was proposed by the TEMICS team. The more the sources are correlated the less the syndromes have to be long, that is how rate adaptive coding is carried out.

Besides the syndromes, some complementary parts of the sources have to be transmitted to the decoder ; then retrieving the difference pattern between the sources returns to filling these parts in. Finally, the decoding ends with the use of the matrix representation of the codes. The size of the transmitted systematic parts differ from one source to another to allow symmetric coding.



# Table des matières

<b>Introduction</b>	<b>1</b>
<b>I État de l'art</b>	<b>3</b>
<b>1 Théorie de l'information</b>	<b>4</b>
1.1 Entropie et information mutuelle . . . . .	4
1.1.1 L'entropie . . . . .	4
1.1.2 L'information mutuelle . . . . .	4
1.1.3 Séquences conjointement typiques : . . . . .	5
1.2 Codage source ( <i>compression de l'information</i> ) . . . . .	5
1.2.1 Les différents types de codes . . . . .	5
1.2.2 Théorème du codage source . . . . .	6
1.2.3 Le codage de Huffman . . . . .	6
1.3 Codage canal ( <i>transmission de l'information</i> ) . . . . .	6
1.3.1 Capacité du canal . . . . .	6
1.3.2 Théorème du codage canal : . . . . .	7
1.4 Codage source-canal conjoint : . . . . .	8
<b>2 Codage LDPC</b>	<b>9</b>
2.1 Représentations d'un code LDPC : . . . . .	9
2.1.1 <i>Représentation matricielle</i> : . . . . .	9
2.1.2 <i>Représentation graphique</i> : . . . . .	9
2.1.3 <i>Ensemble de codes LDPC</i> : . . . . .	10
2.2 Codage et décodage d'un code LDPC : . . . . .	10
2.2.1 <i>Codage</i> : . . . . .	10
2.2.2 <i>Décodage</i> : canal symétrique binaire . . . . .	10
2.2.3 <i>Décodage</i> : canal à effacement . . . . .	12
<b>3 Codage convolutif</b>	<b>13</b>
3.1 Structure des codes convolutifs . . . . .	13
3.1.1 Code convolutif de rendement $1/n$ . . . . .	13
3.1.2 Code convolutif de rendement $k/n$ . . . . .	14
3.2 Représentations d'un code convolutif . . . . .	15
3.2.1 Représentation polynômiale . . . . .	15
3.2.2 Codes convolutifs récurrents systématiques . . . . .	15
3.2.3 Représentations graphiques des codes convolutifs . . . . .	16
3.2.4 Transformations des codes convolutifs . . . . .	19
3.3 Algorithmes de décodage . . . . .	19
3.3.1 Les algorithmes traditionnels . . . . .	19
3.3.2 L'algorithme utilisé lors de la présence d'une information adjacente au niveau du décodeur . . . . .	20

<b>4</b>	<b>Codage de sources distribué</b>	<b>21</b>
4.1	Les différentes configurations . . . . .	22
4.1.1	<i>Cas où une source est entièrement connue au décodage</i> : . . . . .	22
4.1.2	<i>Cas où les deux sources sont codées à des débits complémentaires</i> . . . . .	22
4.2	Exemple de codage de Slepian-Wolf . . . . .	23
4.2.1	Codage aux points anguleux A et B : . . . . .	23
4.2.2	Codage symétrique au point C : . . . . .	24
4.2.3	Généralisation : . . . . .	24
<b>5</b>	<b>Approche de la borne de Slepian-Wolf : Méthode SF-ISF pour un codage symétrique</b>	<b>26</b>
5.1	Principe . . . . .	26
5.2	Mise en oeuvre : . . . . .	27
5.3	Application et résultats . . . . .	28
<b>6</b>	<b>Approche de la borne de Slepian-Wolf : Poinçonnage du syndrome pour une adaptation en débit</b>	<b>30</b>
6.1	Motivations de l'accumulation et principes du poinçonnage . . . . .	30
6.2	Construction du code . . . . .	31
6.3	Application et résultats . . . . .	32
<b>II</b>	<b>Contributions</b>	<b>34</b>
<b>7</b>	<b>Approche de la borne de Slepian-Wolf : codage symétrique et adaptatif en débit</b>	<b>35</b>
7.1	Algorithme développé pour le codage LDPC . . . . .	36
7.1.1	Principe . . . . .	36
7.1.2	Description de l'algorithme : . . . . .	36
7.2	Résultats . . . . .	38
7.3	Exemple d'application . . . . .	41
7.4	Problèmes rencontrés . . . . .	42
7.4.1	Inversibilité de la partie B . . . . .	42
7.4.2	Propagation d'erreurs . . . . .	42
7.5	Solutions envisagées . . . . .	43
7.6	Codage symétrique dans le cadre du codage convolutif . . . . .	43
7.6.1	Cas particulier du codeur convolutif de rapport de compression 1 :2 . . . . .	43
7.6.2	Généralisation à tout code convolutif non récursif . . . . .	45
7.6.3	Codes convolutifs récursifs . . . . .	46
<b>8</b>	<b>Travaux à venir</b>	<b>48</b>
8.1	Décodage itératif par propagation de croyance conjointe . . . . .	48
8.1.1	Présentation de l'algorithme . . . . .	48
8.2	Codage symétrique utilisant les turbo-codes . . . . .	49
	<b>Conclusion</b>	<b>50</b>
	<b>Annexes</b>	<b>52</b>
<b>9</b>	<b>Conception d'un code LDPC aux bons rendements</b>	<b>53</b>
<b>10</b>	<b>Inversion d'une matrice H de taille quelconque par la réduction de Gauss</b>	<b>54</b>
<b>11</b>	<b>Résolution des systèmes d'équations avec le pivot de Gauss</b>	<b>55</b>

<b>12 Le codage de Huffman</b>	<b>56</b>
12.1 Algorithme de Huffman : <i>Codage</i> . . . . .	56
12.2 Algorithme de Huffman : <i>Décodage</i> . . . . .	57
<b>13 Le codage turbo</b>	<b>58</b>
13.1 Codage d'un code turbo . . . . .	58
13.1.1 Choix des codeurs convolutifs . . . . .	58
13.1.2 Choix de l'entrelaceur . . . . .	59
13.1.3 Représentation des codes turbo en tant que codes en blocs . . . . .	60
13.2 Décodages d'un code turbo . . . . .	63
13.2.1 Décodage classique : MAP . . . . .	63
13.2.2 Décodage en tant que code en bloc : Propagation de croyance . . . . .	63
 <b>Bibliographie</b>	 <b>63</b>



## Introduction

Ce stage, effectué du 7 mars 2007 au 7 septembre 2007, se déroule à l'IRISA, centre de recherche membre de l'INRIA, dans l'équipe TEMICS. Il vise à compresser les informations issues de deux sources corrélées aux limites découvertes par Slepian et Wolf en 1973.

### Présentation du centre de recherche :

L'institut de recherche en informatique et systèmes aléatoires (IRISA), est un pôle de recherche public regroupant environ 530 personnes dont 205 chercheurs ou enseignants chercheurs, 175 chercheurs en thèse, 90 ingénieurs, techniciens, administratifs et de nombreux collaborateurs contractuels ou invités internationaux pour des séjours de plus courte durée. L'INRIA, le CNRS, l'Université de Rennes 1 et l'INSA de Rennes sont les partenaires de cette unité mixte de recherche.

L'INRIA, institut national de recherche en informatique et en automatique placé sous la double tutelle des ministères de la recherche et de l'industrie, a pour vocation d'entreprendre des recherches fondamentales et appliquées dans les domaines des sciences et technologies de l'information et de la communication (STIC). L'institut assure également un fort transfert technologique en accordant une grande attention à la formation par la recherche, à la diffusion de l'information scientifique et technique, à la valorisation, à l'expertise et à la participation à des programmes internationaux. Jouant un rôle fédérateur au sein de la communauté scientifique de son domaine et au contact des acteurs industriels, l'INRIA est un acteur majeur dans le développement des STIC en France. L'INRIA accueille dans ses 6 unités de recherche situées à Rocquencourt, Rennes, Sophia Antipolis, Grenoble, Nancy et Bordeaux, Lille, Saclay et sur d'autres sites à Paris, Marseille, Lyon et Metz, 3 600 personnes dont 2 800 scientifiques, issus d'organismes partenaires de l'INRIA (CNRS, universités, grandes écoles) qui travaillent dans plus de 138 projets de recherche communs. Un grand nombre de chercheurs de l'INRIA sont également enseignants et leurs étudiants (environ 1 000) préparent leur thèse dans le cadre des projets de recherche de l'INRIA.

L'équipe de traitement, modélisation d'images et communications (TEMICS) est un projet de l'IRISA qui porte sur le traitement de l'image et de la vidéo. Ses activités de recherches sont principalement l'analyse, le codage et la transmission de la vidéo sur les réseaux. L'équipe travaille sur quatre thèmes majeurs : *“l'analyse et la modélisation de séquences vidéo”*, *“le codage de source-canal conjoint”*, *“la compression, le codage scalaire et le codage de sources distribué”*, et *“le masquage de l'information et le filigranage”*. Mon intervention entre dans le thème du codage de sources distribué

### Le but du stage :

Le codage de sources distribué a été présenté pour la première fois par David Slepian et Jack K. Wolf en 1973. Il s'agit de la transmission de deux sources corrélées ne communiquant pas entre elles sur un canal. La transmission et/ou la compression vidéo sont des utilisations courantes du codage distribué de sources corrélées. À la réception, les deux sources sont décodées conjointement. Dans notre cas, nous voulons utiliser le codage de sources distribué pour compresser deux sources corrélées. Si on appelle  $H$  la grandeur entropie,  $H(X, Y)$  représentant l'entropie conjointe de deux sources  $X$  et  $Y$ , il est connu bien avant Slepian et Wolf qu'un codage conjoint des deux sources permet, dans le cas extrême, de compresser  $X$  à un débit  $H(X)$  et  $Y$  à un débit  $H(Y|X)$  (rappelons que  $H(X, Y) = H(X) + H(Y|X) = H(Y) + H(X|Y)$ ), les valeurs alternatives étant permises. Le résultat intéressant de Slepian-Wolf est qu'un codage disjoint des deux sources, du moment qu'un codage conjoint est entrepris, autorise tout aussi bien les mêmes taux de compression. Cependant, Slepian et Wolf ne fournissent pas de méthodes permettant d'atteindre ces taux de compression, ni de compresser/transmettre aux valeurs des débits alternatifs. Une question de recherche dans ce domaine reste donc la manière de coder aux débits approchant le plus intimement possible la limite de compression de Slepian-Wolf.

Dans un domaine aussi vaste et en constante expansion que le codage de sources distribué, notre première démarche a été de répertorier tout ce qui se fait en la matière dans les publications des chercheurs et des doctorants. C'est pour cela qu'une première partie de ce rapport est assignée à l'état de l'art dans ce domaine. Dans cette partie, le lecteur pourra aussi trouver des précisions théoriques

sur les grandeurs et notions mises en œuvre dans le domaine, qui sont sous le label de la théorie de l'information. Puis nous découvrirons plus en détails ce qu'est la limite de Slepian-Wolf et deux méthodes remarquables permettant d'approcher, à partir d'angles différents, cette limite théorique ; la première approche est le parcours de la borne, et la deuxième approche est l'adaptation en débit vis-à-vis de la borne. Dans cette première partie, un chapitre sera aussi accordé à la présentation de nos outils principaux quant à l'accomplissement de notre mission : les codages LDPC et convolutif.

La deuxième partie de ce rapport portera précisément sur l'utilisation que nous avons faite de tout ce qui a été vu précédemment pour approcher la borne de Slepian-Wolf des deux angles, simultanément. Nous montrerons que l'approche adaptation en débit et parcours de la borne sont compatibles, sous certaines conditions liées aux codes LDPC ou convolutifs utilisés. La fin de la deuxième partie du rapport présente les pistes qu'il nous reste à suivre (parmi d'autres qu'il nous reste à imaginer) afin d'améliorer les performances des codes mis en œuvre ou de se passer des conditions imposées par ces mêmes codes.

Nous espérons que vous prendrez plaisir à découvrir les fruits de notre travail lors de la lecture des pages à venir.

Première partie

État de l'art

# Chapitre 1

## Théorie de l'information

Dans ce premier chapitre, nous définissons les grandeurs et notions indispensables à la compréhension des autres chapitres à venir, et surtout de la partie II. Ce chapitre ne se veut pas une référence sur la théorie de l'information, ni un cours ; il a été fortement inspiré par ce qu'on pourrait appeler *une bible* dans le domaine : “*Elements of information theory*” de Thomas et Cover (voir [3]). Pour une compréhension approfondie du codage de sources distribuées, la lecture de la section 4.2 sera des plus utiles.

### 1.1 Entropie et information mutuelle

#### 1.1.1 L'entropie

Soit  $X$  une variable aléatoire (VA) discrète dont les réalisations  $x$  prennent leurs valeurs dans  $\mathbf{X}$ . On définit l'entropie  $H(X)$  comme étant une mesure de l'incertitude de  $X$  : c'est *la quantité moyenne d'information nécessaire à la description* de  $X$  ; pour une source binaire, elle est représentée comme étant le nombre minimal de bits indispensables à la description de  $X$ . Si  $X$  suit une densité  $p(x)$ , alors :

$$H(X) = - \sum_{x \in \mathbf{X}} (p(x) \log(p(x))) = E \left[ \log \left( \frac{1}{p(x)} \right) \right] \quad (1.1)$$

Où  $E$  est l'espérance mathématique. L'entropie possède les propriétés suivantes :

$$\begin{aligned} H(X) &\leq \log(\mathbf{X}) \\ H(X) &\geq 0 \\ H(X, Y) &= E \left[ \log \left( \frac{1}{p(x, y)} \right) \right] = H(X) + H(Y|X) = H(Y) + H(X|Y) \\ H(X, Y, Z) &= H(X) + H(Y|X) + H(Z|Y, X) \end{aligned}$$

Le log est en base 2 et l'entropie s'exprime en bits. Par exemple, l'entropie d'un jet de pièce de monnaie (dont seulement deux valeurs “pile” ou “face” peuvent ressortir, avec une probabilité de  $\frac{1}{2}$  pour chacune) vaut 1 bit.

#### 1.1.2 L'information mutuelle

Considérons deux variables aléatoires  $X$  et  $Y$  ayant une distribution de probabilité conjointe  $p(x, y)$  et des distributions marginales  $p(x)$  et  $p(y)$ . L'*information mutuelle*  $I(X; Y)$  est l'entropie relative entre la distribution conjointe et le produit des distributions marginales.

$$I(X; Y) = \sum_{x \in \mathbf{X}} \sum_{y \in \mathbf{Y}} \left( p(x, y) \log \left( \frac{p(x, y)}{p(x)p(y)} \right) \right) \quad (1.2)$$

Voici quelques relations :  $I(X;Y) = H(X) - H(X|Y) = H(Y) - H(Y|X) = H(X) + H(Y) - H(X,Y)$

L'*information mutuelle* est une mesure de l'information que la variable aléatoire  $X$  contient à propos de l'autre variable aléatoire  $Y$ . C'est une réduction de l'incertitude sur la VA  $Y$ , connaissant  $X$ . C'est l'intersection de l'information portée par  $X$  et celle portée par  $Y$ .

Ces propriétés de  $H$  et  $I$  sont aussi vraies pour les VA continues, en remplaçant les " $\sum$ " par des " $\int$ ".

### 1.1.3 Séquences conjointement typiques :

Soit  $\epsilon > 0$ . L'ensemble  $A_\epsilon^{(n)}$  des séquences  $\{x^n, y^n\}$  conjointement typiques, respectivement à la distribution  $p(x, y)$  est l'ensemble des séquences de longueur  $n$  aux entropies empiriques s'écartant de moins de  $\epsilon$  des vraies entropies. Autrement dit, si  $p(x^n, y^n) = \prod_{i=1}^n (p(x_i, y_i))$  alors :

$$A_\epsilon^{(n)} = \left\{ (x^n, y^n) \in X^n \times Y^n / \begin{aligned} \left| -\frac{1}{n} \log(p(x^n)) - H(X) \right| &< \epsilon, \\ \left| -\frac{1}{n} \log(p(y^n)) - H(Y) \right| &< \epsilon, \\ \left| -\frac{1}{n} \log(p(x^n, y^n)) - H(X, Y) \right| &< \epsilon \end{aligned} \right\}$$

Il y a environ  $2^{nH(X)}$  séquences  $X$  typiques, et environ  $2^{nH(Y)}$  séquences  $Y$  typiques. Cependant, puisqu'il n'y a que  $2^{nH(X,Y)}$  séquences conjointement typiques, toutes les paires de  $X^n$  typiques et  $Y^n$  typiques ne sont pas conjointement typiques. La probabilité qu'une paire choisie aléatoirement soit conjointement typique est d'environ  $2^{-nI(X;Y)}$ . Ainsi, pour un  $Y^n$  donné, on peut analyser plus de  $2^{nI(X;Y)}$  paires avant de tomber sur une conjointement typique. Cela laisse à penser qu'il existe  $2^{nI(X;Y)}$  signaux  $X^n$  distincts.

## 1.2 Codage source (*compression de l'information*)

Un *code source*  $C$  pour une variable aléatoire  $X$  est une fonction de correspondance de  $X$ , l'ensemble des  $X$ , vers  $D$ , l'ensemble des chaînes de symboles d'un alphabet binaire, de longueur finie. Les éléments de  $D$  sont des *mots de code*. Notons  $C(x)$  le mot de code correspondant à  $x \in \mathbf{X}$  et  $l(x)$  la longueur de  $C(x)$ .

Lorsqu'on parle de codage source, il s'agit de supprimer toutes les redondances présentes dans la source afin d'obtenir la version la plus compressée possible de l'information qu'elle contient. La compression de l'information peut s'effectuer en affectant des descriptions courtes aux mots de source  $x$  les plus fréquents et des descriptions plus longues pour les mots de sources les moins probables. Par exemple, dans le code *morse* le symbole le plus fréquent est représenté par un simple point.

Les deux principaux résultats que nous voyons ici est que le taux de compression maximal de toute information est son *entropie*  $H$  (1.2.2), et que le *codage de Huffman* (voir section 1.2.3 ou [11] pour plus de détails) permet de compresser à ce taux.

### 1.2.1 Les différents types de codes

- Un code  $C$  est dit *non-singulier* si chaque élément de  $X$  est associé à une chaîne différente dans  $D$  :  $x_i \neq x_j \leftrightarrow C(x_i) \neq C(x_j)$ . La non singularité d'un code suffit pour une description sans ambiguïté de  $X$  ; mais habituellement ce qui nous intéresse est de coder plusieurs séquences de  $X$ , ce qui se traduirait par le rajout d'un séparateur entre chaque mot de code, ce qui détruit les performances du code.
- Un code est à *décodage unique* si chaque chaîne de  $D$  possède une et une seule source  $x$  de  $X$ .
- Un code est *préfixé*, ou *instantané*, si aucun mot de code n'est le préfixe d'aucun autre. Un code instantané peut se décoder sans aucune référence aux futurs mots de codes puisque la fin d'un mot de code est tout de suite reconnaissable : un code instantané s'*auto-sépare*.

Pour tout code préfixé sur un alphabet binaire, les longueurs des mots de codes  $l_1, l_2, \dots, l_m$  doit satisfaire l'*inégalité de Kraft* :  $\sum_{i \in [1, m]} 2^{-l_i} \leq 1$ .

La réciproque est vraie : étant donné un ensemble de mots de codes vérifiant l'inégalité de Kraft, il existe un code instantané admettant ces longueurs de mots de code.

Les codes préfixés sont *optimaux* (permettent de compresser à hauteur de l'entropie).

### 1.2.2 Théorème du codage source

Les théorèmes 1.2.2 et 1.3.2 sont la base de la théorie de l'information.

**Résultat :** Tous les *taux de compressions*  $R > H$  sont réalisables. L'entropie  $H$  est donc la limite compression de toute information.

Une manière d'atteindre ce taux de compression est le *codage de Huffman* qui utilise des mots de code de longueur minimale, compte tenu de la distribution  $p(x)$ . Les codes de Huffman sont des codes préfixés, donc optimaux.

### 1.2.3 Le codage de Huffman

L'algorithme de Huffman est simple et permet d'associer à chaque symbole de la source un mot de code à la longueur minimale, à la vue de la distribution de la source.

Soit une source  $X = (x_i)_{i=1 \dots m}$ , comportant  $m$  symboles distinctes de longueurs respectives  $l_{i=1 \dots m}$ , ayant la distribution suivante :  $p(x) = (p_1, p_2, \dots, p_m)$ , vérifiant  $p_1 \geq p_2 \geq \dots \geq p_m$ . Alors, il existe un code optimal qui vérifie les trois propriétés suivantes :

- (1) Si  $p_j > p_k$  alors  $l_j \leq l_k$  ;
- (2) Les deux mots de codes les plus longs ont la même longueur ;
- (3) Les deux plus longs mots de code ne diffèrent que du dernier bit et correspondent aux symboles les moins probables.

Pour une description plus détaillée du codage de Huffman, référez-vous à [11] ou à l'annexe 12.

## 1.3 Codage canal (*transmission de l'information*)

Soit un canal de transmission dont l'entrée est la VA  $X$  de réalisations  $x \in \mathbf{X}$ , dont la sortie est la VA  $Y$  dont les réalisations  $y$  prennent leurs valeurs dans  $\mathbf{Y}$ , et la matrice des probabilités de transmissions  $p(x|y)$  exprimant la probabilité qu'on observe  $y$  sachant que le symbole  $x$  a été transmis.

Le *canal* est l'ensemble formé par le triplet  $(\mathbf{X}, \mathbf{Y}, p(x, y))$ .

### 1.3.1 Capacité du canal

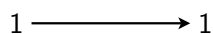
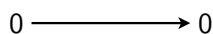
La *capacité informationnelle du canal* est  $C = \max_{p(x)} I(X; Y)$  ; le maximum est pris ici sur l'ensemble des distributions possibles pour  $X$ .

En pratique, on préfère définir la capacité du canal comme le débit maximal (en bits) par utilisation du canal auquel l'information peut être transmis, avec une probabilité d'erreurs arbitrairement faible, c'est la *capacité opérationnelle*. Le second théorème de Shannon, voir [32] ou [29], établit que la *capacité informationnelle* du canal est égale à sa *capacité opérationnelle*.

Dualité avec la compression de l'information : Pendant la compression, on supprime toutes les redondances comprises dans l'information afin d'en créer la version la plus compressée possible, alors que la transmission de l'information consiste à y rajouter des bits de redondance de manière intelligente pour combattre les erreurs introduites par le canal. Dans la dernière partie de ce chapitre, nous verrons que les deux approches sont duales, et un système de communication complet peut se diviser en deux parties *compression* et *transmission* pouvant être optimisées séparément.

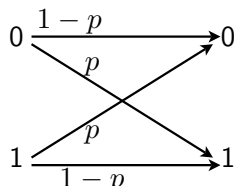
### Exemples de canaux :

*Canal binaire sans bruit* :  $C = 1$  bit. L'entrée est reproduite sans erreurs à la sortie,  $p(x) = (\frac{1}{2}, \frac{1}{2})$ .



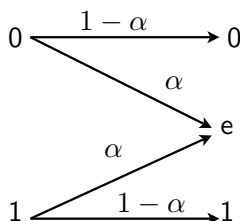
*Canal binaire symétrique (BSC) avec paramètre  $p$*  :  $C = 1 - H(p)$  bit. Les symboles d'entrée sont complémentaires avec une probabilité  $p < \frac{1}{2}$ .

C'est le modèle le plus simple d'un canal induisant des erreurs mais il illustre bien la complexité du problème général ; les bits reçus ne révèlent en rien la position des bits erronés et, en un sens, tous les bits reçus sont incertains.



*Canal binaire à effacement de paramètre  $\alpha$*  :  $C = 1 - \alpha$  bit. Si dans le canal précédent les bits sont corrompus, ici ils sont perdus. Le récepteur sait quels bits ont été perdus.

Il n'est pas évident qu'une transmission sans erreurs peut s'effectuer avec ce débit. C'est pourtant ce qui ressort du second théorème de Shannon ; une manière de procéder est de considérer un canal à contre-réaction sur l'encodeur : si un bit est erroné, on le retransmet jusqu'à avoir un créneau sans effacement.



*Canal discret avec contre-réaction* du décodeur sur le codeur (ou "Feedback channel") :  $C_{FB} = C = \max_{p(x)} I(X; Y)$  bit.

Ce résultat important stipule que la contre-réaction sur le codeur, après observation du canal ne peut augmenter la capacité de ce canal : comme dans le cas d'un canal à effacement, la contre-réaction aide énormément à simplifier le codage et le décodage, mais elle n'augmente en rien la capacité.

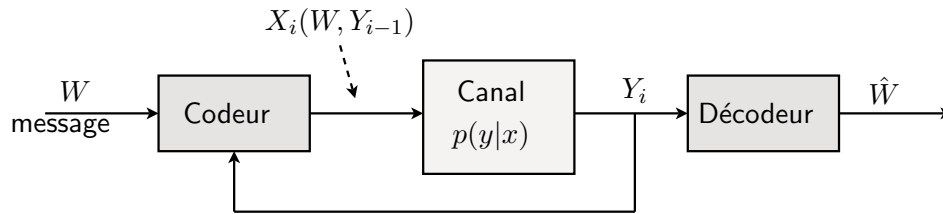
### 1.3.2 Théorème du codage canal :

**Résultat :** Tous les débits de transmissions  $R < C$  sont réalisables.

En d'autres termes, quel que soit le débit  $R < C$ , il existe une séquence de codes  $(2^{nR}, n)$  ayant une probabilité maximale d'erreurs tendant  $\lambda^{(n)}$  vers 0 permettant de transmettre à ce débit. La réciproque est aussi vraie : toute séquence de codes  $(2^{nR}, n)$  ayant  $\lambda^{(n)} \rightarrow 0$  doit avoir un débit  $R \leq C$ .

Ce théorème montre que, pour une longueur de code suffisamment grande, il existe de bons codes<sup>1</sup> à débit approchant la capacité, mais il ne donne pas de méthodes pour construire de tels codes. De fait, la preuve apportée par Shannon repose sur sélection aléatoire des codes.

<sup>1</sup>La théorie veut qu'on puisse transmettre strictement sans aucune erreur ; ici on se limite à l'obtention d'une probabilité d'erreur très faible, de l'ordre de  $10^{-6}$



## 1.4 Codage source-canal conjoint :

**Théorème :** Si  $H$  est l'entropie de la distribution de probabilités et si  $C$  est la capacité du canal (réel ou équivalent, entre deux sources par exemple), alors, si on veut transmettre à un débit  $R$  :

$$\begin{cases} R > H \\ R < C \end{cases}$$

Clairement, cela signifie qu'on a le droit de concevoir des codes sources les plus efficaces possibles afin de représenter l'information, et qu'on a le droit de concevoir séparément les codes canaux appropriés pour le canal de transmission, sans perte d'efficacité dans le traitement de l'information.

Avec ce théorème, on lie ensemble les deux théorèmes fondamentaux précédents (1.2.2 et 1.3.2).

Munis de ces notions fondamentales, nous sommes maintenant prêts à comprendre le *codage de sources distribué*. Mais d'abord, sympathisons avec le *codage LDPC*, qui est le code canal le plus puissant dans le domaine du codage de sources distribué.

# Chapitre 2

## Codage LDPC

Les codes LDPC (*low-density parity-check*) sont une classe de codes en blocs linéaires, découverts en 1963 par Gallager [7]. Ce nom leur vient des caractéristiques de leur matrice de contrôle de parité, qui comporte plus de 0 que de 1. Leur avantage principal est qu'ils fournissent une performance très proche de la capacité pour beaucoup de canaux différents, ainsi qu'une complexité linéaire avec le temps lors du décodage. On peut décrire un code LDPC sous deux manières différentes : *description matricielle* (comme tout code en blocs linéaire) ou sous forme d'un graphe : le *graphe de Tanner* [26].

Construire un code LDPC ayant de bonnes performances n'est pas un problème ; en fait, il y a de fortes chances pour que des codes choisis complètement au hasard soient bons. Le problème provient de la complexité du codeur.

### 2.1 Représentations d'un code LDPC :

#### 2.1.1 Représentation matricielle :

Soit  $H$  la matrice représentant un code LDPC  $(n, k)$  donné.  $H$  est de taille  $(m \times n)$ , si on note  $m = n - k$ .

Si on appelle  $w_r$  le nombre de 1 d'une ligne de  $H$  et  $w_c$  celui d'une colonne, alors, si on veut construire une matrice  $(m \times n)$  à faible densité, on doit vérifier que  $w_r \ll n$  et  $w_c \ll m$ .

Habituellement, les matrices utilisées sont de très grandes tailles. Si  $w_r$  et  $w_c$  sont constants pour les lignes et pour les colonnes, tels que  $w_r = w_c \frac{n}{m}$ , alors le code LDPC est *régulier*.

Une importante taille des blocs donne des matrices de contrôle parité et des matrices génératrices de grandes tailles ; on sait de plus que la complexité d'une multiplication entre un mot de code et une matrice dépend du nombre de 1 qu'elle contient. Si on met la matrice  $H$  sous la forme  $H = [P_T \ I]$ , la matrice génératrice  $G$  correspondante se calcule par  $G = [I \ P]$ . La sous-matrice  $P$  n'est généralement pas creuse, ce qui induit une complexité assez grande lors des calculs.

Voici un exemple de matrice  $H$  pour un code LDPC  $(8, 4)$  :

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

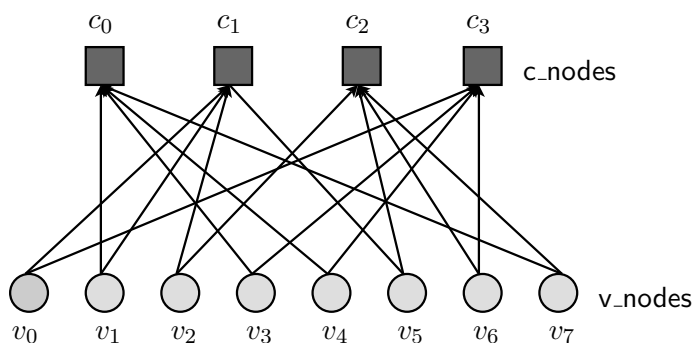
#### 2.1.2 Représentation graphique :

Les *graphes bipartites* représentant les codes LDPC ont été découvertes par *Tanner*. Ces graphes représentent complètement les codes LDPC correspondants, et ils aident dans la description des algorithmes de décodage (que nous allons présenter dans la section 2.2.2).

Le graphe de Tanner comprend deux types de nœuds : les *v-nodes* (nœuds de variable) et les *c-nodes* (nœuds de contrôle, ou *syndrome*). La construction du graphe est assez élémentaire :

- Un code LDPC  $(n, k)$  comportera  $m = n - k$  c\_nodes (le nombre de bits de parité) et  $n$  v\_nodes (le nombre de bits d'un mot à coder).
- Le degré d'un v(c)\_node est le nombre de c(v)\_nodes auxquels il est relié, dans la représentation graphique.
- Le degré d'un c(v)\_node est le nombre de 1 dans la ligne (colonne), dans la représentation matricielle.
- Si l'élément  $h_{ij}$  de la matrice  $H$  vaut 1, alors le c\_node  $c_i$  est connecté au v\_node  $v_j$ .
- Dans la matrice  $H$  correspondant au code LDPC, une ligne (colonne) représente un c\_node (v\_node).

Pour mieux comprendre, voici le graphe de Tanner correspondant à la matrice  $H$  décrite plus haut :



### 2.1.3 Ensemble de codes LDPC :

Pour un code LDPC *irrégulier*, les degrés de chaque type de nœuds peuvent être choisis selon une distribution particulière. Pour une longueur donnée et pour une distribution donnée, on peut définir un *ensemble de codes*, en choisissant aléatoirement les connexions entre les nœuds de variables et les nœuds de contrôles. Tous les éléments de cet ensemble sont équiprobables. Autrement dit, on énumère les connexions émanant des v\_nodes selon un ordre arbitraire, et on procède de même pour les connexions émanant des c\_nodes. En posant que le nombre de connexions =  $E$ ; alors un code (un élément de l'ensemble de codes) peut être identifié avec une permutation sur  $E$  bits.

Soit un polynôme  $\gamma$  de la forme  $\gamma(x) = \sum_{i \geq 2} (\gamma_i x^{i-1})$ .  $\gamma$  est une *distribution de degrés* si  $\gamma(x)$  n'a aucun coefficient négatif et  $\gamma(1) = 1$ . Soient  $n$  la longueur du code (nombre de v\_nodes),  $d_v$  et  $d_c$  les degrés maximaux des v\_nodes et c\_nodes respectivement,  $\lambda(x) = \sum_{i=2}^{d_v} (\lambda_i x^{i-1})$  et  $\rho(x) = \sum_{i=2}^{d_c} (\rho_i x^{i-1})$  les distributions de degrés associées aux v\_nodes et c\_nodes.

On associe à la paire de distributions de degrés  $(\lambda, \rho)$  une séquence d'ensembles de codes  $C^n(\lambda, \rho)$ . Par exemple, pour le code LDPC régulier  $(2, 4)$  précédent,  $\lambda(x) = x$  et  $\rho(x) = x^3$ .

## 2.2 Codage et décodage d'un code LDPC :

### 2.2.1 Codage :

Le codage d'un mot  $x$  (v\_nodes) en un syndrome  $s$  (c\_nodes) s'effectue par la simple opération " $s = Hx$ " (en prenant soin aux dimensions respectives). Coder à partir du graphe bipartite revient à effectuer une addition binaire de toutes les valeurs des v\_nodes connectés au même c\_node.

Le décodage d'un code correcteur d'erreurs revient, très simplement, à comparer les probabilités de différents mots de code.

### 2.2.2 Décodage : canal symétrique binaire

On dispose au décodeur de la matrice de contrôle de parité caractérisant le code LDPC, des mots entachés d'erreurs et des syndromes (éventuellement entachés d'erreurs aussi). La méthode de décodage

des codes LDPC s'appelle "*belief propagation*" (propagation de croyance), qui fait partie de l'ensemble d'algorithmes "*message passing*" (passage de messages); elle est appropriée pour les canaux de type binaires symétriques (canal BSC, *Binary Symmetric Channel*).

La méthode de décodage d'un code LDPC comme codeur canal<sup>1</sup> a été découverte simultanément, et à différentes locations, par plusieurs personnes. En ce qui nous concerne, nous transmettons les syndromes non nuls de "*faux*" mots de codes, l'algorithme de décodage associé a été découvert par Liveris *et al.* dans [15]. C'est ce dernier algorithme que nous nous proposons de présenter ici; la description ici faite est très inspirée de [13].

D'abord quelques notations :

$y_i$  : valeur du v\_node  $v_i$

$P_i = Pr(v_i = 1|y_i)$

$q_{ij} = (q_{ij}(0), q_{ij}(1))$  : message envoyé par  $v_i$  à  $c_j$ ;  $q_{ij}(0)$  est la probabilité que  $y_i$  soit 0.

$r_{ji} = (r_{ji}(0), r_{ji}(1))$  : message envoyé par  $c_j$  à  $v_i$ ;  $r_{ji}(0)$  est la probabilité que  $y_i$  soit 0.

$V_{j \setminus i}$  : l'ensemble des v\_nodes  $v$  sauf l' $i$ -ème.

$C_{i \setminus j}$  : l'ensemble des c\_nodes  $c$  sauf le  $j$ -ème.

$K_{ij}$  : constantes à choisir tels que  $q_{ij}(0) + q_{ij}(1) = 1$ .

En respectant ces notations, voici les étapes principales du décodage d'un mot transmis étant (ou pas) un mot de code du code linéaire en blocs :

1. Tous les v\_nodes envoient leurs messages  $q_{ij}$ . Au départ,  $q_{ij}(1) = P_i$  et  $q_{ij}(0) = 1 - P_i$
2. Les c\_nodes calculent leurs messages de réponse  $r_{ji}$  de la façon suivante :

$$\begin{cases} r_{ji}(0) = \frac{1}{2} + \frac{1}{2} \prod_{i' \in V_{j \setminus i}} (1 - 2q_{i'j}(1)) \\ r_{ji}(1) = 1 - r_{ji}(0) \end{cases}$$

On calcule ici la probabilité qu'il y ait un nombre pair de 1 parmi les v\_nodes privés de  $v_i$ .

3. Les v\_nodes mettent à jour l'estimation actuelle  $v_i^*$  de leur variable  $v_i$ . Pour cela on calcule les probabilités d'avoir un 1 ou un 0 pour chaque v\_node, et on choisit le plus grand des deux. Les équations suivantes sont utilisées à cette fin :

$$\begin{cases} Q_i(0) = K_i(1 - P_i) \prod_{j \in C_i} (r_{ji}(0)) \\ Q_i(1) = K_i P_i \prod_{j \in C_i} (r_{ji}(1)) \end{cases}$$

puis on applique la règle :  $v_i^* = \begin{cases} 1 & \text{si } Q_i(1) \geq Q_i(0) \\ 0 & \text{sinon} \end{cases}$

Si cette estimation du mot de code vérifie les équations de parité, alors l'algorithme se termine; dans le cas contraire, on doit mettre un terme à l'algorithme après un nombre prédéfini d'itérations. Si l'algorithme ne se termine pas, les v\_nodes mettent leurs messages de réponse à jour, en prenant en compte les c\_nodes :

$$\begin{cases} q_i(0) = K_{ij}(1 - P_i) \prod_{j' \in C_{i \setminus j}} (r_{j'i}(0)) \\ q_i(1) = K_{ij} P_i \prod_{j' \in C_{i \setminus j}} (r_{j'i}(1)) \end{cases}$$

4. Aller à l'étape " 2 " ...

---

<sup>1</sup>Un décodeur canal retrouve des mots de code d'un code donné, dont le syndrome est  $s_0 = 000 \dots 0$ . Lors de la transmission sur un canal, un syndrome non nul est synonyme de la découverte d'erreurs dans le mot reçu.

Pour accroître les performances du décodeur, on peut effectuer les calculs de probabilités (qui sont très lourds) dans le domaine logarithmique ; on parle alors de messages sous formes de LLR (*Log-Likelihood Ratio*). Même si les performances des codes LDPC se sont révélées très intéressantes en ce qui concerne la compression de l'information, la spécification du meilleur code LDPC reste toujours un sujet de recherche ouvert. Certains chercheurs proposent des algorithmes performants pour créer des “bonnes” matrices<sup>2</sup>.

### 2.2.3 Décodage : canal à effacement

Le canal à effacement diffère du canal symétrique, comme nous l'avons vu dans le chapitre 1.3.1, en cela qu'une partie  $\alpha$  des bits sont perdus et non pas erronés avec une probabilité  $p$ . Di *et al.* présentent en détails dans [4] les limites du décodage des codes LDPC (de longueurs finies) par des algorithmes de type passage de messages, et proposent un nouvel algorithme, algorithme qui est repris et amélioré par Hossein *et al.* dans [18] puis par Vellambi *et al.* dans [28]. En ce qui nous concerne, nous voulons plus tard poinçonner le code LDPC afin d'approcher la borne de Slepian-Wolf, et le poinçonnage revient à éluder certains bits bien choisis, ce qui peut être représenté par un canal à effacement.

L'algorithme repose sur la découverte, parmi les bits effacés du mot transmis, d'un *ensemble d'arrêt* : c'est un ensemble pour lequel la propagation de croyance ne peut plus proposer de valeurs sans faire d'erreurs, ces bits correspondent à un ensemble de valeurs effacées dont les voisins y sont connectés au moins par deux liens. La définition et la caractérisation exhaustive d'un ensemble d'arrêt sont données dans [4]. Les auteurs montrent que, pour un ensemble d'arrêt de taille  $m$ , il existe un nombre minimal,  $g$ , de bits parmi ces  $m$ -là dont la connaissance permet de retrouver les  $m - g$  autres.

Une fois un tel ensemble trouvé, c'est à dire une fois l'algorithme par propagation de croyance arrêté, l'algorithme propose intelligemment  $g$  valeurs, jusqu'à trouver les  $g$  positions parmi les  $m$  qui permettent de retrouver les  $m$  bits. Leurs améliorations de l'algorithme consiste à mieux choisir, et à diminuer,  $g$  et les  $g$  valeurs parmi les  $m$ .

---

<sup>2</sup>comme MacKay dans [16], dont l'algorithme est proposé en annexe 9

# Chapitre 3

## Codage convolutif

Les *codes convolutifs*, trouvés en 1955 par Elias ([5] et [6]), peuvent être considérés comme un cas particulier de codes en bloc linéaires, mais un point de vue plus large nous fera découvrir que la structure convolutive additionnelle munit le code linéaire de propriétés favorables qui facilitent à la fois son codage et améliorent ses performances.

Les codes convolutifs forment une classe extrêmement souple et efficace de codes correcteurs d'erreurs. Ce sont les codes les plus utilisés dans les systèmes de télécommunications fixes et mobiles. Théoriquement, ils ont les mêmes caractéristiques que les codes en blocs sauf pour la valeur de leur dimension et leur longueur. Les codes convolutifs s'appliquent sur des séquences infinies de symboles d'information et génèrent des séquences infinies de symboles codés.

Dans un premier temps, nous présenterons les codes convolutifs comme un cas particulier des codes en bloc linéaires, avant d'exploiter dans un deuxième temps leur structure spécifique pour aboutir à un procédé de décodage à maximum de vraisemblance de faible complexité. Ce chapitre est très inspiré de [19].

### 3.1 Structure des codes convolutifs

#### 3.1.1 Code convolutif de rendement 1/n

Considérons le code en bloc linéaire spécifié par la matrice génératrice suivante :

$$G = \begin{bmatrix} g_0 & g_1 & g_2 & \cdots & g_{L-1} \\ & g_0 & g_1 & g_2 & \cdots & g_{L-1} \\ & & g_0 & g_1 & g_2 & \cdots & g_{L-1} \\ & & & \ddots & & & \\ & & & & g_0 & g_1 & g_2 & \cdots & g_{L-1} \end{bmatrix} \quad (3.1)$$

Chaque élément  $g_l, 0 \leq l \leq L-1$ , de la matrice est un vecteur ligne binaire à  $n$  composantes de la forme :  $g_l = [g_{l1} \ g_{l2} \ \cdots \ g_{ln}]$ .

Les zones laissées vides en bas à gauche et en haut à droite correspondent à des éléments tous nuls. Chaque ligne s'obtient en décalant la précédente de  $n$  colonnes vers la droite. Cette matrice, de taille  $(B \times (nB + L - 1))$  décrit donc un code linéaire en bloc dont les mots de code pourraient être engendrés par le dispositif de la figure 3.1.

Généralement, les mots de codes sont très longs, c'est à dire  $B \gg L$  et nous pouvons même considérer cette matrice comme *infinie*. Ce code linéaire ne possède donc plus à proprement parler de dimension ni de longueur. Le rendement ne peut alors être défini comme pour les codes en blocs. Cependant en négligeant *la fermeture* du code (le vidage des registres), nous remarquons qu'il existe un rapport  $1/n$  entre le nombre de bits d'information et le nombre de bits du mot de code. Le rendement de ce code est donc :  $r = \frac{1}{n}$

La quantité  $L$  est appelée *longueur de contrainte* du code convolutif.

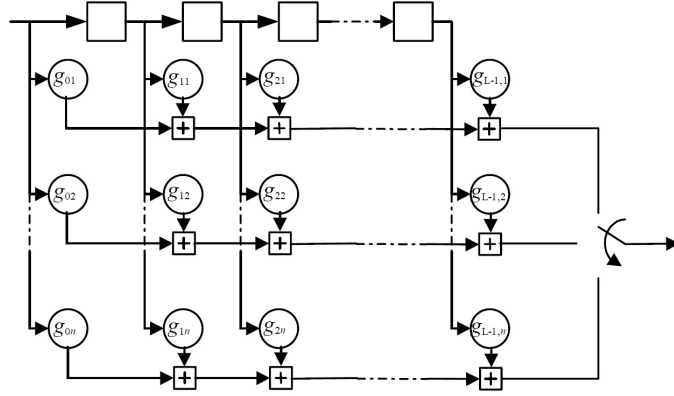


FIG. 3.1 – Code convolutif de rendement  $r = \frac{1}{n}$ .

Le terme "convolutif" s'applique à cette classe de codes parce que la suite de bits codés  $s$  peut s'exprimer comme le résultat de la convolution de la suite de bits d'information  $e$  par les coefficients  $g$ . En effet, puisque le code est linéaire, nous avons :  $s = e.G$ . En observant la forme particulière de  $G$ , les  $n$  bits en sortie du codeur à l'instant  $t$ , correspondant à une entrée, s'écrivent :  $s_t = \sum_{u=\max(1,t-L+1)}^t e_k \cdot g_{u-k}$ .

La complexité du codeur est indépendante de la longueur de la séquence de bits d'information émise et ne dépend que de la longueur de contrainte du code.

### 3.1.2 Code convolutif de rendement $k/n$

Pour obtenir un rendement égal à un nombre rationnel quelconque inférieur à 1, il nous faut généraliser la forme de la matrice génératrice (3.1) et le schéma de la figure 3.1 du codeur. Nous supposons maintenant que  $k$  bits d'information sont introduits en parallèle à un instant donné dans le codeur ou qu'un décalage temporel du vecteur  $e$  des bits d'information se fait par bloc de  $k$  bits. Cela revient à remplacer la ligne  $[g_0 \ g_1 \ \dots \ g_{L-1}]$  de taille  $(1 \times L)$  par une sous-matrice binaire de taille  $(k \times L \cdot n)$  dont les éléments constituant  $g_l$  sont de taille  $(k \times n)$ . La construction générale par décalage de la matrice reste la même.

$$G = \begin{bmatrix} \overset{n}{\leftrightarrow} \\ k \updownarrow g_0 \ g_1 \ g_2 \ \dots \ g_{L-1} \\ \quad \quad \quad g_0 \ g_1 \ g_2 \ \dots \ g_{L-1} \\ \quad \quad \quad \quad \quad g_0 \ g_1 \ g_2 \ \dots \ g_{L-1} \\ \quad \quad \quad \quad \quad \quad \quad \ddots \\ \quad \quad \quad \quad \quad \quad \quad \quad \quad g_0 \ g_1 \ g_2 \ \dots \ g_{L-1} \end{bmatrix} \quad (3.2)$$

**Exemple :** Soit le code convolutif de rendement  $\frac{2}{3}$  et de longueur de contrainte  $L = 2$  défini par les coefficients suivants :

$$g_0 = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \text{ et } g_1 = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$

Son codeur possède la structure suivante :

La sortie d'un codeur convolutif à l'instant  $t$  dépend de ses entrées à l'instant  $t$  et de l'état des registres du codeur à cet instant. En posant  $\nu = L - 1$ , tout code convolutif de rendement  $\frac{k}{n}$  et de longueur de contrainte  $L$  possède  $k \cdot \nu$  registres, qui peuvent donc prendre  $2^{k \cdot \nu}$  états différents. Comme l'entrée est constituée de  $k$  bits différents, il existe  $2^k$  transitions possible à partir d'un état du codeur à un instant donné.

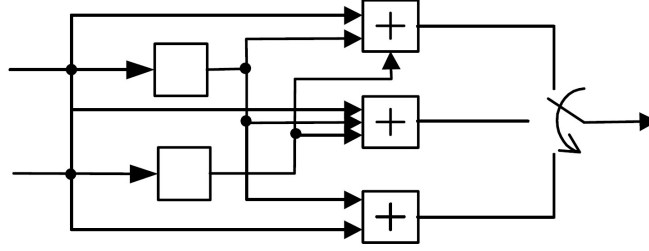


FIG. 3.2 – Code convolutif de rendement  $\frac{2}{3}$  et de longueur de contrainte  $L = 2$ .

## 3.2 Représentations d'un code convolutif

### 3.2.1 Représentation polynômiale

Soit un code convolutif de rendement  $\frac{k}{n}$  et de longueur de contrainte  $L$ . Son entrée peut être vue comme la donnée de  $k$  séquences infinies de bits, que nous représentons par des séries en  $x$  :  $\forall i \in [1, k], e_i(x) = \sum_{l=0}^{+\infty} e_{il} \cdot x^l$ , où l'inconnue  $x$  symbolise un retard d'une période. Le coefficient  $e_{il}$  est le bit présent à la  $i$ -ème entrée du codeur à l'instant  $l$ . De la même manière, les  $n$  sorties du codeur peuvent être indexées par  $j$  et s'écrivent :  $\forall j \in [1, n], s_j(x) = \sum_{l=0}^{+\infty} s_{jl} \cdot x^l$ .

La relation introduite par le code convolutif entre les entrées et les sorties peut s'écrire de façon élégante avec ses notations, en introduisant les *polynômes générateurs*. Le polynôme générateur qui lie la  $i$ -ème entrée à la  $j$ -ième sortie est noté  $g_{ij}(x)$ , et est de degré inférieur ou égal à  $L - 1$ . Le coefficient de degré  $r$  de ce polynôme vaut 1 si une connexion existe entre la sortie  $j$  et le bit présent dans le  $r$ -ième registre correspondant à l'entrée  $i$ .

La notation octale est couramment utilisée pour représenter ces polynômes. Ainsi,  $g(x) = 1 + x + x^3$  sera représenté par le nombre 13.

Puisque la convolution temporelle correspond à la multiplication polynômiale, les sorties se calculent en fonction des entrées à l'aide des polynômes générateurs :

$$s_j(x) = \sum_{i=1}^k g_{ij}(x) e_i(x)$$

Une représentation matricielle compacte de la formule précédente se déduit de l'écriture sous forme vectorielle des entrées et sorties :  $E = [e_1(x) \ e_2(x) \ \dots \ e_k(x)]$ ,  $S = [s_1(x) \ s_2(x) \ \dots \ s_n(x)]$ ,  $S = G \cdot E$ , où :

$$G = \begin{pmatrix} g_{11}(x) & g_{12}(x) & \dots & g_{1n}(x) \\ g_{21}(x) & g_{22}(x) & \dots & g_{2n}(x) \\ \vdots & & \dots & \vdots \\ g_{k1}(x) & g_{k2}(x) & \dots & g_{kn}(x) \end{pmatrix} \quad (3.3)$$

### 3.2.2 Codes convolutifs récurrents systématiques

Un code convolutif est dit *récurrent* lorsque ses polynômes générateurs sont remplacés par les quotients de deux polynômes. Une partie de la sortie est alors réintroduite dans le registre à décalage selon les connexions définies par les polynômes situés aux dénominateurs.

Un code convolutif est dit *systématique* lorsqu'une partie de ses sorties est exactement égale à ses entrées. Cela revient à dire qu'une sous-matrice  $G_{k \times k}$  vaut la matrice identité :

$$G = \begin{bmatrix} g_{11}(x) = 1 & g_{12}(x) = 0 & \dots & g_{1k}(x) = 0 \\ g_{21}(x) = 0 & g_{22}(x) = 1 & \dots & g_{2k}(x) = 0 \\ \vdots & & \dots & \vdots & \dots \\ g_{k1}(x) = 0 & g_{k2}(x) = 0 & \dots & g_{kk}(x) = 1 \end{bmatrix} \quad (3.4)$$

Un cas particulièrement intéressant est celui de la transformation d'un code convolutif non récursif non systématique (NRNSC) de rendement  $\frac{k}{k+1}$  en un code récursif systématique (RSC).

Prenons l'exemple d'un code NRNSC de rendement  $\frac{1}{2}$ . Il est défini par la donnée de ses deux polynômes générateurs  $g_{11}(x) = g_1(x)$  et  $g_{12}(x) = g_2(x)$ , qui forment la matrice  $G = [g_1(x) \ g_2(x)]$ .

La transformation en code RSC correspond à considérer le code défini par la matrice génératrice  $G = [1 \ \frac{g_1(x)}{g_2(x)}]$ .

### 3.2.3 Représentations graphiques des codes convolutifs

L'idée d'une représentation graphique d'un code convolutif provient des caractéristiques markoviennes de la sortie du codeur. En effet, la sortie du codeur dépend de son entrée et de ses états. Les graphes équivalents à la représentation polynômiale sont souvent plus faciles à manipuler et permettent de dériver des résultats plus puissants.

Tout code convolutif est représenté par trois graphes équivalents mais différents : *l'arbre du code*, *le treillis du code* et *le diagramme d'états*.

- *L'arbre* est un graphe de hauteur et de largeur infinies. Un *sommet* dans l'arbre représente un état possible du codeur. Une *arrête* symbolise une transition d'un état à l'autre. Classiquement l'arbre commence à son sommet par l'état 0 (le registre à décalage est initialisé à 0). Tout chemin dans l'arbre du code est une séquence possible (un mot de code) à la sortie du codeur convolutif.
- Le *treillis* est obtenu en repliant l'arbre sur sa largeur, par fusion des sommets représentant le même état au même instant.
- Le *diagramme d'états* est à son tour construit en repliant le treillis sur sa longueur, par fusion des sommets représentant aussi le même état, à des instant différents.

Incontestablement, *le treillis* est l'outil graphique le plus performant pour caractériser un code et concevoir des algorithmes de décodage. Nous illustrons ci-dessous l'arbre, le treillis et le diagramme des deux codes, l'un NRNSC (7, 5) et l'autre RSC (1,  $\frac{5}{7}$ ) à 4 états.

**Représentations graphiques du code NRNSC  $R = \frac{1}{2}$ ,  $g_1 = 7$ ,  $g_2 = 5$  :** Notons  $a = 00$ ,  $b = 01$ ,  $c = 10$  et  $d = 11$ , les 4 états possibles du codeur convolutif. Les deux bits constituant l'indice de l'état ne sont autres que les deux bits contenus dans le registre à décalage de la figure 3.3.

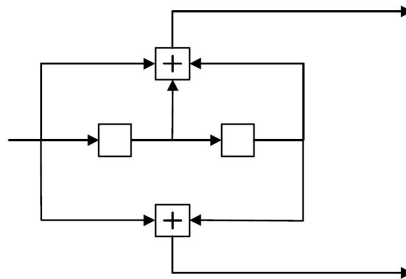


FIG. 3.3 – Code convolutif NRNSC (7, 5).

Voici donc quelques caractéristiques de ce code :

- Le code convolutif possède  $2^{k \cdot (L-1)} = 2^{k \cdot \nu} = 4$  états
- Il y a  $2^k = 2$  transitions par états
- Et donc  $2^{k \cdot L} = 8$  transitions possible entre deux instants consécutifs.

La figure 3.5 représente l'arbre de ce code.

Le treillis est obtenu en repliant l'arbre sur sa hauteur. Il est représenté sur la figure 3.4. Les deux premiers bits sur chaque transition représentent la sortie, et le dernier l'entrée correspondante. Notons que le treillis devient périodique à partir de la 3-ième étape où l'on retrouve les 8 transitions entre les 4 états. En général, le treillis d'un code convolutif devient stationnaire après  $L = \nu + 1$  étapes.

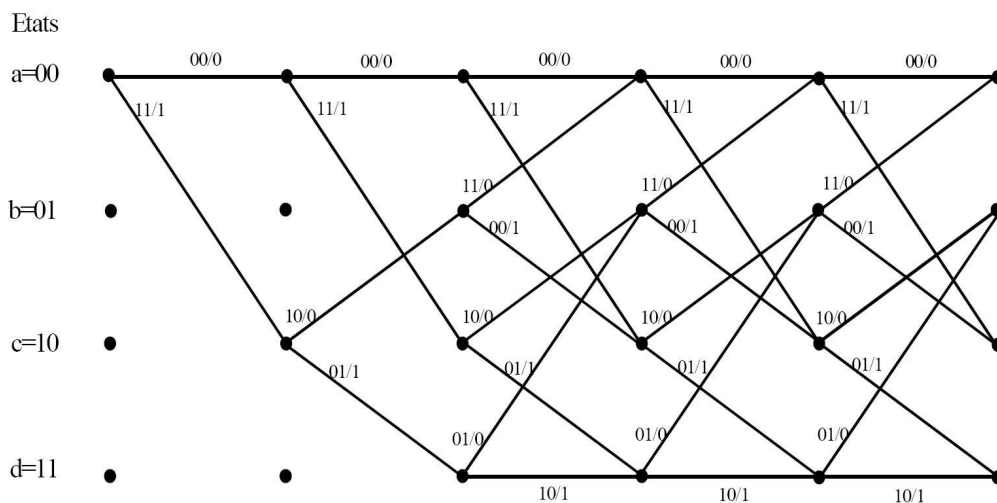


FIG. 3.4 – Treillis du code convolutif NRNSC (7, 5).

Les mots du code (7, 5) sont tous les chemins possibles dans le treillis. Il est facile de vérifier que le code est linéaire, *i.e.* le chemin  $00\dots 0$  appartient au treillis et la somme binaire modulo 2 des bits codés présents sur les branches de deux chemins correspond aux bits présents sur les branches d'un troisième chemin dans le treillis.

D'autre part, le treillis permet de trouver facilement la distance minimale du code. Il s'agit du poids de Hamming minimal entre deux mots de codes, donc deux chemins. En comparant tous les chemins au chemin tout à zéro, dans notre cas, un chemin divergeant à n'importe quel instant du chemin tout à zéro, en suivant  $a \rightarrow c \rightarrow b \rightarrow a$  pour converger à nouveau vers le chemin tout à zéro possède le poids de sortie minimal  $W_{Hmin} = 5$ . Nous en déduisons que la distance minimale  $d_{Hmin}$  (aussi appelée *distance libre dfree*) du code convolutif (7, 5) est égale à 5. Ce chemin correspond à un poids en entrée de 1.

**Représentations graphiques du code RSC  $R = \frac{1}{2}$ ,  $g_1 = 1$ ,  $g_2 = \frac{5}{7}$  :** Ce code est le transformé en code RSC du code NRNSC  $R = \frac{1}{2}$ ,  $g_1 = 7$ ,  $g_2 = 5$ . Il possède de diagramme de la figure 3.6.

La structure des deux treillis est donc parfaitement identique : deux branches arrivent et partent de chaque état. Le treillis devient périodique à la  $L = 3$ -ième étape, lorsque toutes les transitions sont présentes. La différence entre le code récursif et non récursif réside dans l'*étiquetage binaire* des branches. En effet, sachant que le code RSC est systématique, le 1<sup>er</sup> bit de sortie est égal à l'entrée. Ainsi les deux bits associés à une branche du treillis représentent toujours les deux bits de sortie et la valeur de l'entrée est contenue dans le 1<sup>er</sup> bit.

Remarquons que la distance minimale de ce code RSC est elle aussi égale à 5 et se déduit du même chemin  $a \rightarrow c \rightarrow b \rightarrow a$ . Une entrée non nulle permet de quitter l'état 0 dans les deux cas récursifs et non récursifs. En revanche, une entrée nulle n'est pas suffisante pour revenir à l'état 0 dans le cas du code RSC. En effet, le retour  $b \rightarrow a$  est généré par  $e = 0$  pour le code NRNSC, et par  $e = 1$  pour le code RSC. Ainsi, le poids d'entrée du chemin à distance minimale du code RSC est égal à 2.

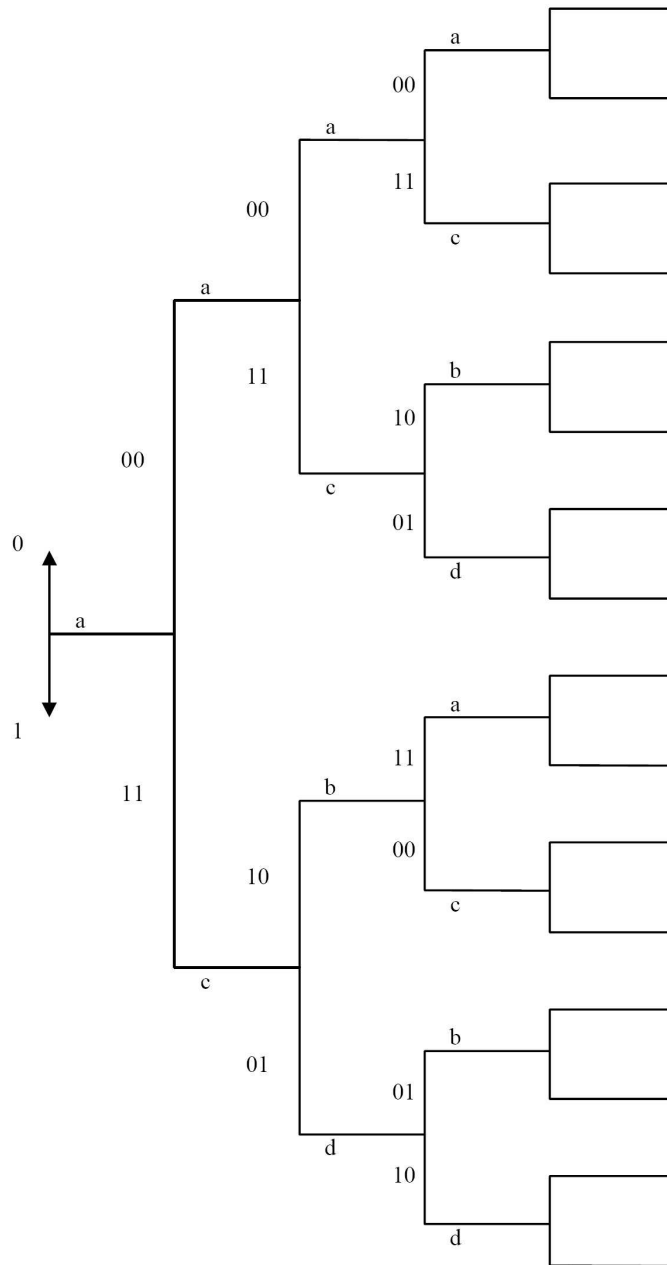


FIG. 3.5 – Arbre du code convolutif NRNSC (7, 5).

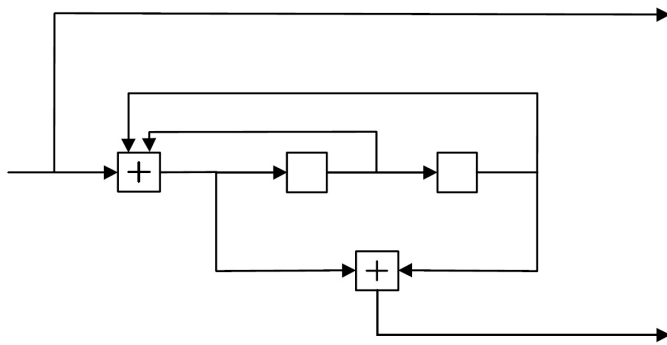


FIG. 3.6 – Diagramme du code convolutif RSC (1,  $\frac{5}{7}$ ).

### 3.2.4 Transformations des codes convolutifs

#### Perforation

Considérons un code convolutif binaire récursif ou non de rendement  $R = \frac{k}{n}$  et de mémoire quelconque. Durant  $u$  étapes dans le treillis, le codeur reçoit  $k.u$  bits d'information et délivre  $n.u$  bits codés. Parmi ces bits codés,  $v$  sont supprimés et  $n.u - v$  sont émis sur le canal. Cette opération, appelée *perforation du code convolutif*, transforme le rendement  $R = \frac{k}{n}$  en un rendement  $\hat{R} = \frac{(k \times u)}{(n.u - v)}$ .

Par exemple, un code convolutif de rendement  $\frac{1}{2}$  est facilement converti par perforation en un code de rendement  $\frac{2}{3}$ , ( $u = 2, v = 1$ ) ou un rendement  $\frac{3}{4}$ , ( $u = 3, v = 2$ ).

Le choix des bits codés non transmis (on dit *perforés*) se fait généralement suivant un motif périodique décrit par une matrice binaire, dite *de perforation*, de taille  $n.u$  qui comporte  $v$  zéro.

#### Fermeture du treillis : transformation en code en bloc

Tout code convolutif est convertible en un code en bloc. Il suffit de stopper l'avance dans le treillis après un nombre de branches fixé par la longueur du code en blocs recherché. Une méthode triviale consiste à couper le treillis sans spécifier l'état final (*troncature*). La bonne méthode transforme le code convolutif en un code en bloc en forçant le retour à l'état 0 par une terminaison du treillis.

Inversement, il est possible de construire un treillis représentant tout code en bloc.

## 3.3 Algorithmes de décodage

Décoder un code convolutif revient à comparer les probabilités des différents chemins possibles dans le treillis, et à choisir le plus probables, étant donné un certain nombre d'informations sur le mot transmis.

### 3.3.1 Les algorithmes traditionnels

Nous ne décrivons ici que succinctement quelques algorithmes de décodage applicables aux codes convolutifs. Ils effectuent leurs opérations sur un treillis, qui représente, dans le cas du décodage des codes convolutifs, la structure du code. Ils sont cependant applicables plus généralement à la recherche du chemin dans un treillis qui minimise un critère de distance additive. Ils sont donc aussi applicables au décodage des codes en bloc (par l'intermédiaire de leur représentation en treillis), à celui des modulations codées en treillis, où à la détection au maximum de vraisemblance sur un canal introduisant de l'interférence entre symboles, problème qui correspond à l'estimation des symboles d'un modèle de chaîne de Markov cachée, représentable lui aussi par un treillis.

Le premier algorithme est celui de *Viterbi* (VA). Il s'agit d'un algorithme de décodage à entrée ferme ou souple mais à sortie ferme (des décisions sont prises sur chaque transition du chemin choisi dans le treillis) qui calcule le chemin dont la vraisemblance est maximale (ML : *Maximum Likelihood*). Il s'agit donc d'un algorithme de décodage ML par séquence (chemin).

Le deuxième algorithme est un *Viterbi modifié* afin de fournir une sortie souple (SOVA : *Soft-Output Viterbi Algorithm*) : à chacun des symboles associés aux transitions du chemin choisi, il fait correspondre une information de fiabilité sous optimale.

Un troisième algorithme est le *Forward-Backward*. Il calcule de manière exacte la probabilité a posteriori des symboles présents sur les branches. Il s'agit donc aussi d'un algorithme à entrée et sortie souples (SISO : *Soft-Input Soft-Output*) à décision MAP (*Maximum A Posteriori*) qui peut tenir compte d'une information probabiliste a priori disponible sur les symboles émis. Contrairement aux algorithmes de Viterbi, il s'agit d'une maximisation symbole par symbole.

Les algorithmes de Viterbi, le SOVA et le Forward-Backward sont couramment utilisés dans les systèmes de transmission mobile et fixe à codage simple ou concaténé.

### **3.3.2 L'algorithme utilisé lors de la présence d'une information adjacente au niveau du décodeur**

Afin de prendre en considération une information adjacente lors du décodage de sources fortement corrélées, il nous sera utile d'utiliser l'algorithme décrit dans l'article préliminaire [21], dont une description détaillée sera prochainement disponible.

Le décodeur est basé sur un super-treillis qui provient du treillis de syndromes proposé dans [31]. Ce qui est remarquable dans cet algorithme est qu'il peut être appliqué à tout code convolutif (et pas seulement les codes systématiques).

Comme nous n'avons pas absolument besoin de connaissances sur les codes turbo pour la compréhension du travail effectué dans la partie II, la description de ces derniers est reportée dans l'annexe 13

Les chapitres précédents ainsi que ce dernier ont présenté les outils pour comprendre ce qui suit : *le codage de sources distribué.*

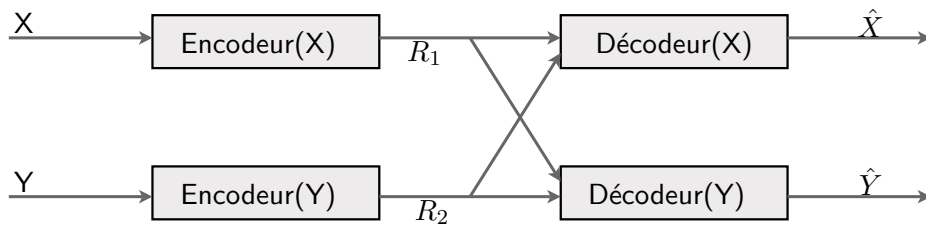
Les trois types de codages que nous avons vus se verront appliquer le codage distribué de sources corrélées, de manière différant d'un codeur à l'autre, comme nous le présenterons dans les chapitres de la partie II.

# Chapitre 4

## Codage de sources distribu 

Il s'agit ici de compresser l'information issue de deux sources corr l es. Ce type d'information est celui qu'on r cup re, par exemple, de r seaux de capteurs sans fil, qui disposent de tr s peu d' nergie utilisable et qui doivent donc transmettre la version la plus compress e possible de leur information   un centre de traitement, voir [23], [33] ou [20]. La meilleure fa on d'y arriver est d'utiliser les redondances existant entre les sources.

Soient deux sources  $X$  et  $Y$ , ind pendantes et identiquement distribu es (*iid*), suivant une distribution de probabilit  conjointe  $p(x, y)$ . Si on peut encoder les deux sources simultan ment, une description<sup>1</sup>  $H(X, Y)$  est suffisante. Dans le cas pr sent, on veut d crire les deux sources s par ment, en les reconstruisant *conjointement* sans erreurs, selon le sch ma suivant :



D'apr s le th or me que *Slepian et Wolf* pr sentent dans [24], pour deux sources corr l es telles que  $p = P_r(X \neq Y) < \frac{1}{2}$ , s par ment encod es et d cod es conjointement, la corr lation  $p$  entre les deux sources  tant connue du codeur et du d codeur, si on veut d crire  $X$  avec  $R_x$  bits en moyenne et  $Y$  avec  $R_y$  bits, alors :

$$\begin{cases} R_x \geq H(X|Y) = H(p) \\ R_y \geq H(Y|X) = H(p) \\ R_x + R_y \geq H(X, Y) = 1 + H(p) \end{cases}$$

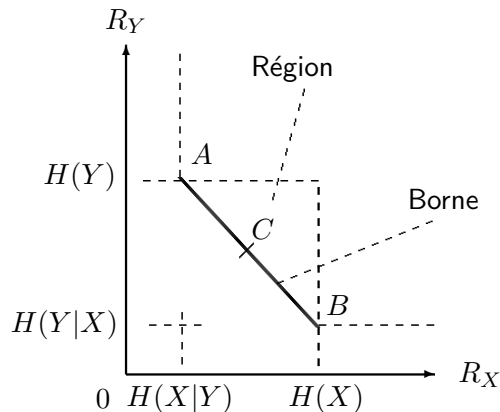


FIG. 4.1 – R gion des d bits autoris s dans le codage distribu  de deux sources

La preuve de cette propri t  a  t  apport e par Slepian et Wolf en 1973 ; elle repose sur la m thode du “*random binning*” ( $\sim$  mise en cases al atoire).

<sup>1</sup>nombre de bits moyen, dans le cas de sources binaires

## 4.1 Les différentes configurations

### 4.1.1 Cas où une source est entièrement connue au décodage :

On se place aux points  $A$  ou  $B$  dans la figure 4.1. Si  $R_y \geq H(Y)$  bits,  $Y$  peut être décrite parfaitement<sup>2</sup> alors  $R_x = H(X|Y)$  bits doivent suffire à décrire  $X$  (point  $A$ ) ; l'autre cas extrême est le point  $B$  avec  $R_y = H(Y|X)$ , on doit alors décrire  $X$  avec  $R_x \geq H(X)$  bits pour pouvoir la retrouver sans erreurs. En général, on utilisera  $R_y = I(Y; \hat{Y})$  bits pour décrire une version appropriée de  $Y$  ; cela permet d'utiliser  $H(X|\hat{Y})$  bits pour la description de  $X$ , en présence de l'information  $\hat{Y}$ .

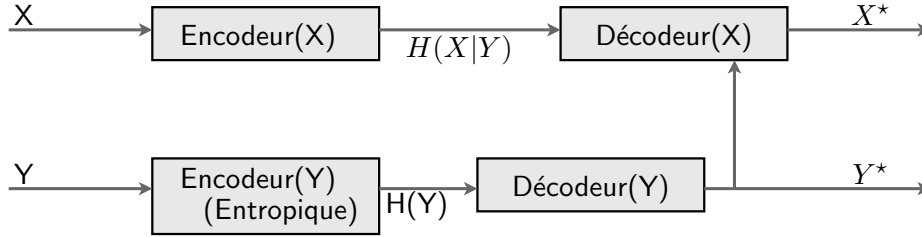
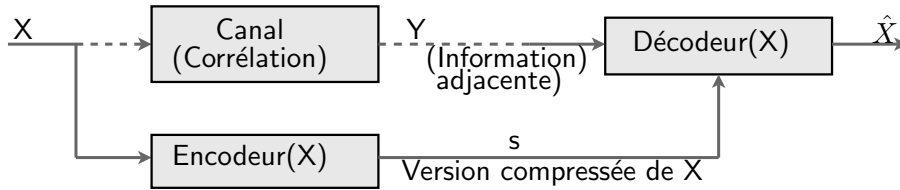


FIG. 4.2 – Codage asymétrique de Slepian-Wolf : une source est connue au décodage

Une manière (intuitive) d'interpréter le système représenté sur la figure ci-dessus est de modéliser la corrélation existant entre les bits  $X_i$  et  $Y_i$  comme étant un canal binaire symétrique (BSC, *binary symmetric channel*), avec une probabilité croisée  $p$  (voir 1.3.1). On fait comme si  $X$  était un mot de code d'un *code canal*. Voici le diagramme correspondant :



### 4.1.2 Cas où les deux sources sont codées à des débits complémentaires

On se place ici sur n'importe quel point du segment  $[AB]$ .

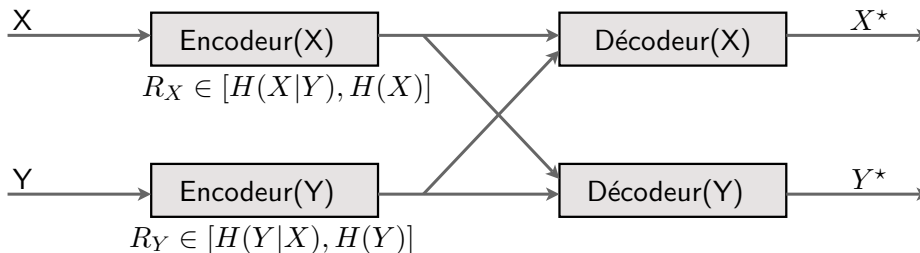


FIG. 4.3 – Codage symétrique de Slepian-Wolf : les deux sources sont connues partiellement

Le cas précédent n'est autre qu'un cas particulier de celui-ci. Deux exemples de méthode permettant d'atteindre tous ces points sont *le time-sharing* et *le random binning*, sur lequel se base la preuve de Slepian et Wolf dans [24] ; *notre travail, présenté dans la partie II consiste à approcher les points de ce segment, en utilisant le codage LDPC (chapitre 2)*.

#### Principe du *time-sharing* :

Pratiquement, tous les points du segment  $[AB]$  peuvent être atteints en appliquant un multiplexage TDM (*time-division multiplexing*) entre les deux modes opératoires extrêmes : c'est ce qu'on appelle "time-sharing". Cependant, cette approche est asymptotique et n'est pas constructive, elle présume en effet de partager les sources en sous-sources à moindres entropies, c'est pourquoi on lui préfère souvent le *binning*.

<sup>2</sup>en utilisant par exemple le codage de Huffman, chapitre 1.2.3 et annexe 12

### Principe du *binning* :

C'est le concept le plus important et le plus utile dans le domaine du codage de sources distribuées. L'idée fondamentale est de grouper des séquences conjointement typiques (voir 1.1.3) dans des *bins* assujettis à certaines contraintes.

Soient deux sources  $X, Y \in \{0, 1\}^n$  indépendantes et identiquement distribuées (iid) et, d'une manière ou d'une autre, corrélées et à corrélation connue.  $Y^n$  est vue comme une version bruitée de  $X^n$ , due à la corrélation entre les deux séquences.

On sait que  $nH(X|Y)$  bits sont nécessaires afin de désigner un bin, et que  $nH(Y)$  bits pour désigner une séquence particulière dans le bin. En se référant de la théorie de l'information, on utilise  $2^{nH(X,Y)}$  séquences conjointement typiques pour la description des sources  $(X^n, Y^n)$  : les séquences sont placées dans  $2^{nH(X|Y)}$  bins disjoints contenant chacun  $2^{nH(Y)}$  séquences (voir figure 4.4). D'un point de vue (algorithmique) plus pratique, le principe du *code binning* revient à diviser l'espace entier des mots de codes d'un code canal en plusieurs sous-espaces disjoints ( $\sim$  bins) de telle sorte que les propriétés de la distance soient conservées dans chaque bin. Cette approche est assez bien expliquée dans [34].

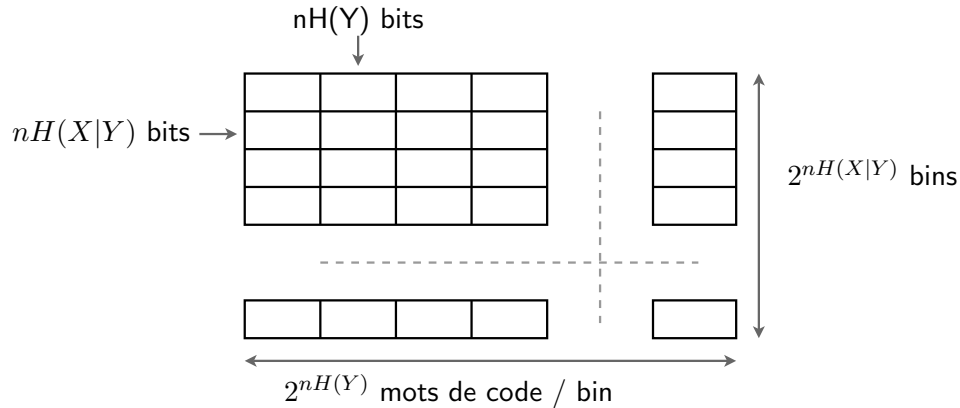


FIG. 4.4 – Le principe du binning : une notion fondamentale

Soit un code canal binaire linéaire dont les séquences sources de longueur  $n$  sont vues comme les mots de code virtuels. L'espace total des mots de code  $X^n = \{0, 1\}^n$  peut être divisé de manière paire en  $2^{n-k}$  bins, pour que chaque bin contienne les mots de code du même syndrome. La contrainte sur la distance est alors remplie grâce à l'uniformité géométrique du codage canal. Naturellement, les  $2^{n-k}$  syndromes peuvent servir à indexer les bins. Dès lors, en transmettant le syndrome  $S^{n-k}$  de longueur  $(n - k)$ , au lieu de la source  $X^n$  de longueur  $n$ , on atteint une compression de rapport  $\frac{n}{n-k}$ . Au décodage, le syndrome  $S^{n-k}$  et l'information adjacente  $Y^n$  aideront à retrouver la séquence  $X^n$  originelle.

Bien sûr, afin d'effectuer un décodage sans perte de  $X^n$ , on peut vérifier que  $\frac{n}{n-k} \geq H(X, Y)$  (comme sur les équations de la figure 4.1).

## 4.2 Exemple de codage de Slepian-Wolf

Soient  $X$  et  $Y$  deux VA binaires discrètes de 7 bits (cad  $X, Y \in \{0, 1\}^7$ ), sans mémoires et iid (*indépendant et identiquement distribuées*), qui sont corrélées de la manière suivante :  $d_H(X, Y) \leq 1$  (les deux VA ne diffèrent que d'(au plus) un bit). Les entropies de  $X$  et  $Y$ , ainsi que leurs entropies conjointes sont les suivantes :  $H(X) = H(Y) = 7$  bits et  $H(X, Y) = H(Y) + H(X|Y) = 10$  bits. D'après le théorème de Slepian-Wolf,  $X$  et  $Y$  peuvent être compressées séparément aux taux respectifs  $R_x$  et  $R_y$ , tels que :  $R_x \geq 3$ ,  $R_y \geq 3$  et  $R_x + R_y \geq 10$ .

### 4.2.1 Codage aux points anguleux A et B :

On peut par exemple compresser  $X$  à  $R_x = 3$  bits, et  $Y$  à  $R_y = 7$  bits (point A sur la figure précédente) en utilisant un code binaire de Hamming  $(7, 4, 3)$ , noté  $\mathbf{C}$ , ayant la matrice de parité  $H$ . Dans ce cas-là, si on note  $x$  la réalisation de  $X$  et  $y$  la réalisation de  $Y$ ,  $Y$  enverrait ses 7 bits

entropiques,  $X$  enverrait le syndrome  $s$  correspondant à  $s = Hx$ , codé sur 3 bits. En clair,  $y$  et  $s$  étant disponibles au décodeur, on peut aisément retrouver la valeur de  $X$ , sachant qu'on doit vérifier que  $d_H(X, Y) \leq 1$ . De la même manière le point où  $R_x = 7$  bits et  $R_y = 3$  bits (point B) peut être atteint en inversant les rôles de  $X$  et  $Y$  dans la description.

#### 4.2.2 Codage symétrique au point C :

Ce point correspond au cas où  $R_x = R_y = 5$  bits. Le principe est de trouver une paire (codeur, décodeur) tel que, si on envoie individuellement  $X$  et  $Y$  au débit de 5 bits chacun, le décodeur puisse les reconstruire sans erreur avec ces seuls 10 bits. Dans ce cas précis, on va utiliser deux codes en blocs linéaires  $C_1$  et  $C_2$ , aux matrices de contrôle de parité respectives  $H_1$  et  $H_2$ , associés respectivement à  $X$  et  $Y$ .

Considérons la matrice génératrice  $G(4 \times 7)$  du code de Hamming binaire. Formons  $G_1$  comme deux lignes de  $G$ , et  $G_2$  comme les deux lignes restantes. Alors  $C_1$  et  $C_2$  sont les sous-codes du code de Hamming ayant une distance minimale de 3. De cette manière, le décodeur pourra retrouver  $X$  et  $Y$  sans erreurs.

Codeur : Notons  $x$  et  $y$  les représentants de  $X$  et  $Y$ . Alors le codeur est donné par  $s_1 = H_1x$  et  $s_2 = H_2y$ ,  $s_1$  et  $s_2$  étant les syndromes des cosets de  $C_1$  et  $C_2$  contenant respectivement  $X$  et  $Y$ .

Décodeur (pourvu d'un algorithme de décodage rapide) : Il faut trouver une paire de mots de codes ayant une distance de Hamming maximale de 1, appartenant chacun à chaque coset de  $C_1$  et  $C_2$ . Soit  $H_1$  et  $H_2$  ( $5 \times 7$ ) sous leurs formes systématiques respectives :  $H_1 = [A_1|I]$  et  $H_2 = [A_2|I]$ .

Alors  $x = u_1G_1 + [0|s_1]$ ,  $y = u_2G_2 + [0|s_2]$  et  $y = x + e$ , tel que  $w(e) \leq 1$  ( $w$  étant le poids de Hamming).  $[0|s_1]$  et  $[0|s_2]$  sont les représentants des cosets de  $C_1$  et  $C_2$  aux syndromes  $s_1$  et  $s_2$  respectifs.

$u_1G_1$  et  $u_2G_2$  représentent des combinaisons linéaires arbitraires des lignes de  $G_1$  et  $G_2$  respectivement. Donc  $e = uG + [0|s]$  où  $u = [u_1|u_2]$  et  $[0|s] = [0|s_1] + [0|s_2]$  est la différence entre  $x$  et  $y$ .

Comme la distance minimale du code vaut 3, la stratégie de décodage est de trouver le mot de code le plus proche de  $[0|s]$  dans  $\mathbf{C}$ .

#### 4.2.3 Généralisation :

En utilisant un code linéaire, on peut atteindre n'importe quelle paire de taux de compression entiers sur le segment  $[AB]$  en se servant de la méthode décrite plus tôt, pourvu que  $R_x \geq 3$ ,  $R_y \geq 3$  et  $R_x + R_y \geq 10$ . Il suffit de prendre  $G_1$  de taille  $[(7 - R_x) \times 7]$  et  $G_2$  de taille  $[(7 - R_y) \times 7]$  où les lignes de  $G_1$  et  $G_2$  sont des partitions disjointes des lignes de  $G$ .

Quelle que soit la paire de sources corrélées uniformément distribuées  $X, Y \in \{0, 1\}^n$  vérifiant  $d_H(X, Y) \leq t$ , celles-ci peuvent être compressées séparément à une paire de taux de compression  $(R_x, R_y)$  tels que :

$$\begin{cases} R_x \geq n - k, R_y \geq n - k, R_x + R_y \geq 2n - k \\ R_x, R_y \in \mathbf{R} \\ k \leq n - \log \left( \sum_{i=0}^t \binom{n}{i} \right) \end{cases}$$

(“Limite de compression sphérique”)

On atteint ces débits à l'aide d'un code en blocs linéaire  $(n, k, 2t + 1)$ , par exemple par un code LDPC (que nous avons présenté dans le chapitre 2).

Pour plus de détails sur le codage de sources distribuées, les différentes configurations permettant de se rapprocher de la borne de Slepian-Wolf et l'avancée générale de l'équipe sur le sujet voyez [2].

Les deux chapitres qui suivent présentent deux méthodes pour approcher la borne de Slepian-Wolf. La première permet d'aller d'un point ( $A$  ou  $B$ ) vers l'autre (voir la figure 4.1), c'est à dire modifier  $R_x$  et  $R_y$  par complémentarité, le canal BSC étant connu ; la deuxième méthode permet de poinçonner les syndromes à transmettre afin de diminuer la somme  $R_x + R_y$  et de s'adapter en débit au gré du canal BSC correspondant à la corrélation entre  $X$  et  $Y$ . Notre travail repose en grande partie sur le rapprochement de ces deux algorithmes, pour pouvoir coder  $X$  et  $Y$  à des *débits complémentaires* (c'est à dire aller librement entre  $A$  et  $B$ ) et *s'adapter en débit*, avec des codes LDPC.

## Chapitre 5

# Approche de la borne de Slepian-Wolf : *Méthode SF-ISF pour un codage symétrique*

Le principe présenté ici a été découvert par Li (Tiffany) *et al.* dans [14] dans le cadre des codes convolutifs, puis appliqué dans [25] pour le codage LDPC.

### 5.1 Principe

Un SF (*Syndrome Former*) et un ISF (*Inverse Syndrome Former*) sont des fonctions qui font correspondre l'espace des mots de codes  $\{X^n\}$  à celui des syndromes  $\{S^{n-k}\}$ , et inversement. Autrement dit, pour un mot donné dans un *bin*<sup>1</sup>, le rôle d'un SF est de trouver le syndrome (index du bin) correspondant ; le rôle de l'ISF est, pour un syndrome donné, de trouver un mot arbitraire de la source qui appartient à un coset particulier. Il est important de noter qu'un couple (SF, ISF) n'est pas unique pour un code canal linéaire<sup>2</sup>  $(n, k)$  donné. Pour obtenir un SF valide, pour autant que le syndrome nul  $(000 \dots 0)$  soit assigné au bin contenant tous les mots de code valides, les assignements peuvent être arbitraires. Donc, il peut y avoir jusqu'à  $(2^{n-k} - 1)!$  SF valides, et pour chacun de ces SF il peut y avoir jusqu'à  $2^k$  ISF correspondants, chacun produisant des ensembles de séquences adjacentes différents. Dès lors, on peut proposer le codeur et le décodeur suivants pour le codage source :

Codeur source : C'est simplement un SF qui fait correspondre une séquence source  $X^n$  (ou  $Y^n$ ), dépendant de quelle source est connue au décodage) à une séquence de syndrome  $S^{n-k}$ . On suppose, dans la suite, que  $Y^n$  est l'information adjacente.

Décodeur source : Il consiste en un ISF correspondant au SF du codeur canal d'origine et à un décodeur canal. Voici comment le décodeur agit pour  $Y^n$  connue au décodeur :

La séquence en sortie de l'ISF est tout d'abord soustraite de l'information adjacente  $Y^n$ , puis le résultat est introduit dans le décodeur canal pour subir le *décodage canal conventionnel*<sup>3</sup>. Si le code canal est assez puissant, alors l'addition de la sortie du décodeur canal et de la séquence adjacente sera, en grande probabilité, la séquence source  $X^n$  de départ.

En ce qui concerne les codes en blocs linéaires, la matrice de contrôle de parité  $H$  est un bon choix de SF ; l'ISF est alors son inverse gauche  $H^{-1}$ , tel que<sup>4</sup>  $HH^{-1} = I$ .

Afin d'atteindre un point arbitraire dans la région de Slepian-Wolf, considérons le plan de la figure 5.1 ci-après et les explication de la section 5.2.

<sup>1</sup>ou *coset* : ensemble des mots de la source donnant le même syndrome

<sup>2</sup>un code canal linéaire  $(n, k)$  a une matrice de contrôle de parité  $H$  de taille  $((n - k) \times n)$

<sup>3</sup>recherche d'un mot de code valide, c'est-à-dire ayant un syndrome nul

<sup>4</sup>En annexe 10, nous détaillons comment obtenir la matrice  $H^{-1}$  correspondant à n'importe quelle matrice  $H$

## 5.2 Mise en oeuvre :

Soit  $H$  la matrice de contrôle de parité du code canal linéaire  $(n, k)$ . SF =  $H$ , ISF =  $H^{-1}$ ;  $x$  et  $y$  sont les entrées corrélées; on pose  $z = x \oplus y$ .

Voici quelles sont les notations que nous utilisons dans le schéma plus bas :

$$x = \begin{bmatrix} x_1 \\ \vdots \\ x_{k_1} \\ \vdots \\ x_k \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} x_1^{k_1} \\ x_{k_1+1}^k \\ x_{k+1}^n \end{bmatrix}, \text{ de même } y = \begin{bmatrix} y_1^{k_1} \\ y_{k_1+1}^k \\ y_{k+1}^n \end{bmatrix}$$

$$H_{(n-k) \times n} = \left[ A_{(n-k) \times k} \mid B_{(n-k) \times (n-k)} \right], \text{ tel que } BB^{-1} = I$$

Le code LDPC utilisé est un code  $(n, k)$ , il encode donc  $n$  bits d'information en  $n - k$  bits de syndrome. En posant  $m = n - k$ , la matrice utilisée est de taille  $(m \times n)$ .

Comme le montre la figure 5.1, les deux sources  $x$  et  $y$  transmettent chacune un syndrome  $s_x = Hx$ ,  $s_y = Hy$ , de longueur  $m$ , ainsi que deux séquences complémentaires sur les  $k$  premiers bits,  $x_1^{k_1}$  (de longueur  $k_1$ ) pour la source  $X$  et  $y_{k_1+1}^k$  (de longueur  $k - k_1$ ) pour la source corrélée  $Y$ , tels que  $0 \leq k_1 \leq k$ . Si le code canal utilisé permet d'atteindre la capacité du canal  $BSC(p)$ , alors  $\frac{k}{n} = 1 - H(p) = 1 - H(X|Y) = 1 - H(Y|X)$ . Puisque les deux sources transmettent  $(n - k + k_1)$  et  $(n - k_1)$  bits respectivement, le total des transmissions s'élève à  $(2n - k) = nH(X, Y)$  bits. Le système a donc un taux de compression de  $(R_X, R_Y) = \left( \frac{n-k+k_1}{n}, \frac{n-k_1}{n} \right)$ , ce qui lui permet d'atteindre n'importe quel point de la borne de Slepian-Wolf.

*Remarque :* pour un code binaire (ce qui est notre cas dans toute cette étude),  $H(X) = H(Y) = 1$  bit, et  $H(Y|X) = H(X|Y) = H(p)$  calculé selon la formule 1.1 et vaut  $H(p) = -p \log(p) - (1-p) \log(1-p)$ .

Le décodeur **(1)** (voir figure 5.1) permet de retrouver la différence  $z = x \oplus y$  à partir des syndromes de  $x$  et de  $y$ .

Le décodeur **(2)** retrouve  $x$  et  $y$  à partir de  $z$ ,  $s_x$ ,  $s_y$ ,  $x_1^{k_1}$  et  $y_{k_1+1}^k$  par la méthode de partitionnement du SF. Les blocs en pointillés ne sont pas obligatoires : une fois l'une ou l'autre des sources décodée, on peut retrouver l'autre par la relation  $z = x \oplus y$ ; plus précisément :

$$\begin{cases} x_1^{k_1} = y_1^{k_1} \oplus z_1^{k_1} \\ y_{k_1+1}^k = x_{k_1+1}^k \oplus z_{k_1+1}^k \\ x_{k+1}^n = B^{-1}(Ax_1^k \oplus s_x), \text{ de même pour } y_{k+1}^n \end{cases} \quad (5.1)$$

Notons que le décodeur ainsi construit limite de façon non négligeable le code LDPC à utiliser pour mettre en oeuvre le principe SF-ISF : la partie carrée  $B$  à droite de la matrice  $H$  doit être inversible. Dans la section 7.4, nous expliquons plus clairement en quoi cette contrainte peut être très gênante pour le cas d'un code LDPC, et surtout pour la mise en oeuvre de la troisième formule des équations (5.1).

Dans le chapitre 8, nous donnons des pistes pour surmonter cette difficulté.

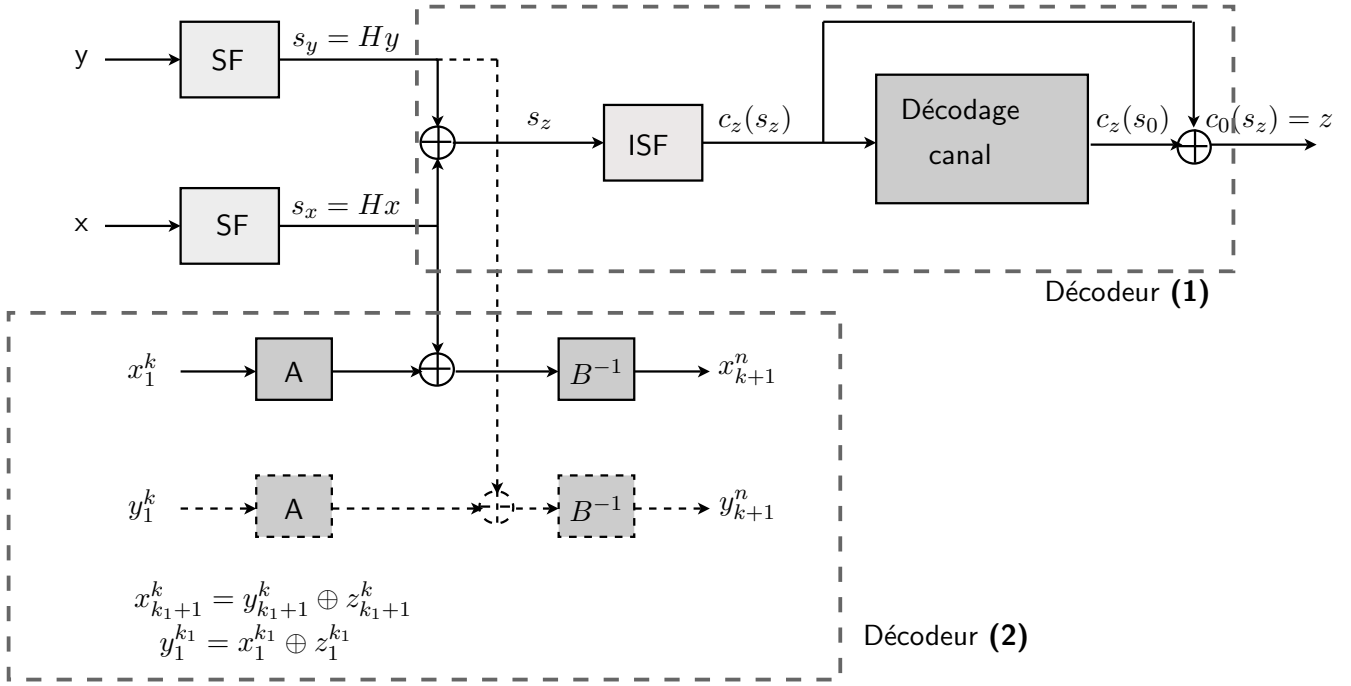


FIG. 5.1 – Approche du parcours de la borne de Slepian-Wolf.

### 5.3 Application et résultats

On veut vérifier que ce modèle est bel et bien valide pour tout code dont la partie  $B$  est inversible.

Tout au long du processus décrit ici, la matrice utilisée pour coder (et décoder) est la même. Comme le taux de compression autorisé par un code LDPC est lié à la taille de la matrice utilisée, dans notre cas  $R_X = \frac{m+k_1}{n}$ ,  $R_Y = \frac{n-k_1}{n}$  ce qui donne un débit total  $R = R_X + R_Y = 1 + \frac{m}{n}$ .

Les courbes obtenues sont représentées sur la figure 5.2.

La courbe en pointillés représente les débits atteints, dans la région de Slepian-Wolf, par une matrice de taille  $(768 \times 426)$  qu'on a construite nous-mêmes afin de tester l'approche. Comme on peut le remarquer, les points (en pointillés) atteints sont alignés, ce qui est normal vu qu'on code à débit constant, mais on code bien au-delà de la borne de Slepian-Wolf (courbe en trait plein), vu que notre code n'est pas optimal.

Chaque source voit son débit réel s'éloigner du débit théorique de 0.1044 bit en moyenne.

Ce test valide le modèle proposé dans [14] pour parcourir la borne de Slepian-Wolf : l'atteinte effective de la borne dépend de l'efficacité attachée au code conçu. Dans le chapitre 6 qui suit, David Varodayan, dans [27], propose une méthode pour améliorer cette proximité.

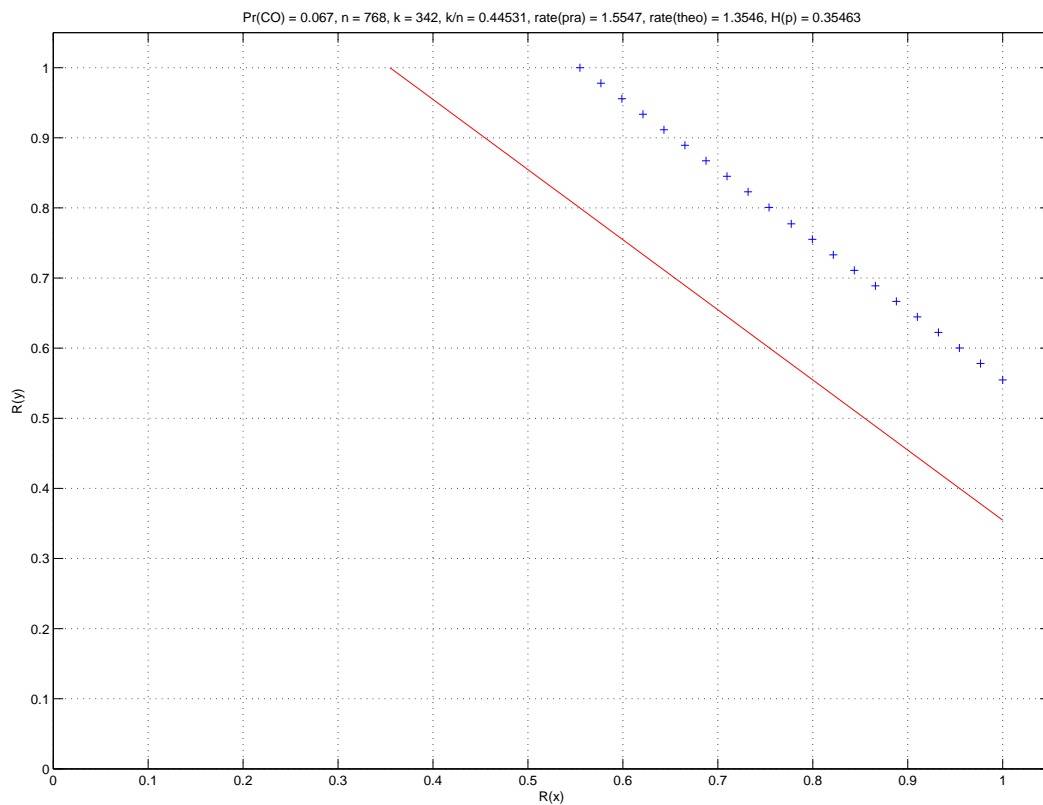


FIG. 5.2 – Approche du parcours de la borne de Slepian-Wolf. La courbe en pointillés représente les débits atteints, dans la région de Slepian-Wolf, par une matrice de taille  $(768 \times 426)$  qu'on a construite nous-mêmes afin de tester l'approche. Comme on peut le remarquer, les points (en pointillés) atteints sont alignés, ce qui est normal vu qu'on code à débit constant, et on code bien au-delà de la borne de Slepian-Wolf (courbe en trait plein).

## Chapitre 6

# Approche de la borne de Slepian-Wolf : *Poinçonnage du syndrome pour une adaptation en débit*

Le poinçonnage d'un code LDPC a été étudié par plusieurs chercheurs, notamment Pishro dans [18], Varodayan [27], Hong [10], McLaughlin[9] ou MacKay [17]. Varodayan, dans [27] se base sur l'accumulation du syndrome pour parer les dégâts que le poinçonnage direct du syndrome implique sur le graphe de Tanner d'origine. C'est précisément cette méthode que nous nous proposons de mettre en œuvre dans notre approche, les résultats qu'il dévoile dans son article étant intéressants (se situant entre 10% et 5% de la borne de Slepian-wolf pour des codes de longueur  $n = 6336$ ) et applicable au principe de l'SF-ISF du chapitre 5, pourvu qu'il y ait un canal à contre-réaction (voir figure 1.3.1), entre le codeur et le décodeur.

*Remarque :* Les opérations se font modulo 2.

### 6.1 Motivations de l'accumulation et principes du poinçonnage

Pour beaucoup d'applications pratiques, la corrélation entre la source et l'information adjacente peut ne pas être connue, ou peut varier, comme dans le codage vidéo, [8]. Comme l'information vidéo est non-ergodique avec une forte probabilité, le débit minimal qu'on peut espérer atteindre varie et ne peut pas être prédit au niveau de l'encodeur. La solution pour s'y adapter est d'utiliser un modèle de canal à contre-réaction.

L'encodeur, basé sur un code agressif<sup>1</sup>, transmet un syndrome de courte taille ( $m$ ), et une information adjacente de longueur  $n$ ; le décodeur essaie d'abord de décoder avec ces informations. Si le décodage réussit, le décodeur le signale au codeur, qui continue avec le prochain syndrome et l'information adjacente correspondante à décoder. Si le décodage échoue, l'encodeur transmet des bits supplémentaires du syndrome, ce qui signifie que le décodeur dispose maintenant d'un syndrome un peu plus long, basé sur un code moins agressif; et le processus continue jusqu'à ce que le décodeur valide qu'il a réussi à décoder le mot avec succès.

Cette approche est optimale, si le canal permet une contre-réaction.

Cependant, pour le codage LDPC en particulier, transmettre le syndrome directement poinçonné serait naïf, et sous-optimal, car le graphe de Tanner obtenu après poinçonnage du syndrome contient des bits d'information non connectés, ou connectés seulement une fois avec un bit de contrôle, comme indiqué figure 6.1(a); le graphe détérioré de cette façon ne permettra pas un décodage itératif. Pour protéger une telle *détérioration du graphe* du code, on se propose d'accumuler le syndrome en sortie du SF :

---

<sup>1</sup>Un code LDPC agressif est un code LDPC pour lequel le niveau de poinçonnage est très élevé. Par exemple, dans les applications de la partie II, au lieu de transmettre les  $m$  bits du syndrome calculé par la matrice  $H$ , on n'en transmet que  $\frac{m}{66}$

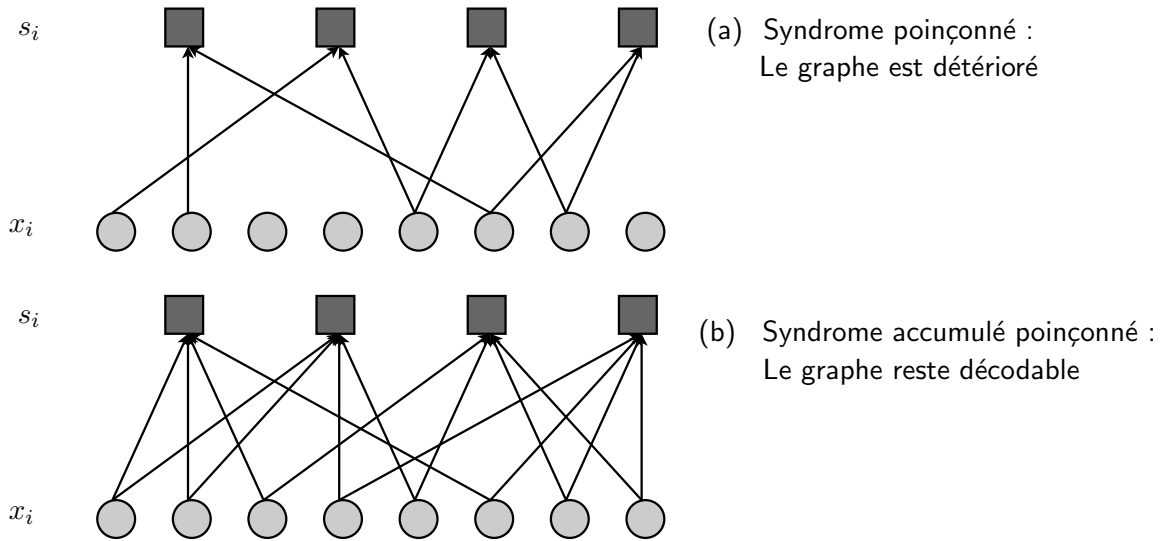


FIG. 6.1 – Le décodeur LDPCA poinçonné est plus performant que le décodeur LDPC poinçonné

**Codeur :** Les bits de la source  $X$  ( $v\_nodes$ ) ( $x_1, x_2, \dots, x_n$ ) sont d'abord sommés au niveau des  $c\_nodes$  conformément au graphe du code de la figure 6.2(c), pour générer les bits du syndrome ( $s_1, s_2, \dots, s_n$ ). Ces bits de syndrome sont ensuite accumulés pour produire les bits du syndrome accumulé ( $a_1, a_2, \dots, a_n$ ). Alors, le codeur met ces syndromes en file (dans un *buffer*) et les transmet petit à petit au décodeur. Le syndrome ainsi accumulé, le graphe de décodage reste viable, figure 6.1(b).

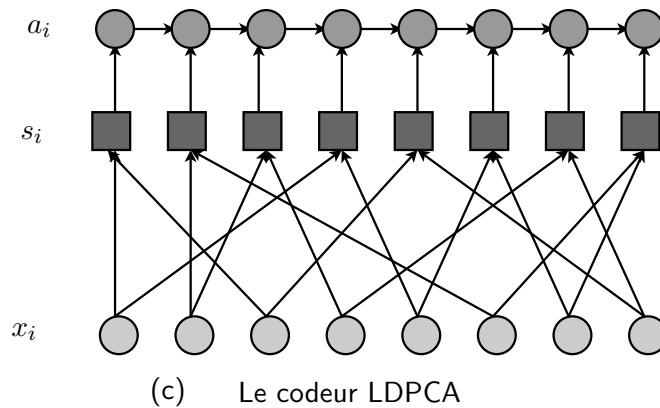


FIG. 6.2 – Le codeur LDPCA est un graphe comportant autant de nœuds de contrôle que de nœuds d'information

**Décodeur :** Le décodeur LDPCA (“A” pour accumulé) gère l’adaptation en débit en modifiant, à chaque fois d’un plus grand nombre de bits du syndrome est reçu, le graphe de Tanner correspondant au décodage. Puis il décode le syndrome (en suivant le principe donné par Liveris dans [15], et présenté au chapitre 2), à l’aide du syndrome dé accumulé.

## 6.2 Construction du code

Pour produire le syndrome non poinçonné de longueur  $n$  à partir du mot de longueur  $n$ , il faut une matrice LDPC ( $n \times n$ ) de rang plein. Une matrice de plus petite taille est obtenue en fusionnant des nœuds de contrôle adjacents : les liens connectés à chaque nœud fusionné sont ceux qui étaient connectés à un de ses nœuds voisins, mais pas aux deux (et ça parce que les liens doubles s’annulent par le modulo 2). Il faut être très vigilant lors de la fusion des nœuds (c’est-à-dire lors de la création des codes de moindres débits) et s’assurer qu’aucun lien n’est perdu.

En fait, pour contourner cette difficulté, on peut commencer la construction des codes par le code correspondant au taux de compression le plus élevé. En effet, en cas d’échec du décodage avec un

tel code, on connaît l'ordre dans lequel les bits de syndromes supplémentaires sont transmis et alors chaque bit de syndrome accumulé reçu résulte en la séparation en deux du nœud correspondant ; l'astuce pour maintenir un nombre constant de liens au travers des séparations successives est de partitionner l'ensemble des liens du nœud précédent en deux ensembles de liens de nœuds adjacents, comme sur la figure 6.3 (les cases vides sont remplies par des 0) :

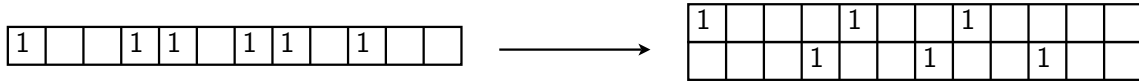
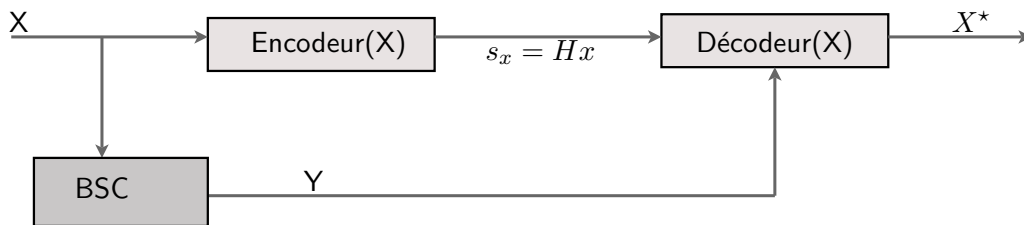


FIG. 6.3 – Les 1 de la ligne sont répartis en deux groupes formant les deux nouvelles lignes du code

Avec cette méthode, on s'assure d'autre part que les nœuds d'information gardent la même distribution de degrés (voir section 2.1.3). Il est montré dans [27] que générer des codes de débit plus élevé par la séparation de nœuds ne diminue pas la capacité de décodage du graphe.

Une méthode efficace pour créer le code de plus fort taux de compression est celle proposée par MacKay et Neal dans leur papier [16] de 1999 <sup>2</sup>.

Le codage et le décodage s'effectuent selon le schéma 4.2, où le décodeur retrouve la source  $X$ , à la connaissance du syndrome  $s_x$  de ce dernier et de l'information adjacente  $Y$  :



### 6.3 Application et résultats

Afin d'appliquer et de tester la faisabilité de cette méthode, il nous faut générer une suite de  $N$  matrices  $(H_i)_{i=1\dots N}$  correspondant à  $N$  débits désirés. Pour cela, on utilise le programme de génération de bons codes de Gallager, proposé par Neal sur son site<sup>3</sup>, pour générer la matrice  $(m_{min} \times n)$  de plus fort taux de compression. Puis, on applique la méthode de séparation des nœuds des syndromes pour obtenir les matrices  $(m_i \times n)$  de plus faibles taux de compression, jusqu'à obtenir la matrice carrée  $(n \times n)$ . Ces matrices sont placées dans un fichier. Varodayan propose sur son site<sup>4</sup> un programme permettant de générer les plus grandes matrices à partir de la plus petite.

L'application de cette méthode est accomplie avec succès, et voici sur la figure 6.4 les points obtenus dans la région de Slepian-Wolf. Lors de cette expérience,  $n = 396$ ,  $p_{moyen} = 0.09$  (fixé).

Cette méthode permet d'améliorer les performances du code, en choisissant, selon le paramètre  $p$  corrélant  $X$  et  $Y$ , le débit minimal acceptable. Si le débit de la source  $Y$  est constante ( $R_Y = 1$ ), le débit de la source  $X$  varie, en recherchant, pour  $p$  donné, le débit minimal. Il est à noter que la borne de Slepian-Wolf n'est pas fixe dans ce cas : il faut comparer chaque point obtenu (*signe "plus" sur le dessin 6.4*) au point théorique correspondant (*signe "carré"*) contrairement aux résultats de la figure 5.2. L'écart moyen entre les valeurs théoriques et les valeurs expérimentales est de 0.1054 bit.

Notre démarche dans la partie qui suit est d'appliquer les deux méthodes de compression vues jusqu'ici en les fusionnant. Cela nous permettra d'atteindre n'importe quel point de la borne de Slepian-Wolf tout en restant adaptatif en débit.

<sup>2</sup>L'algorithme peut être téléchargé à partir de la page web de Neal : c'est celle que nous utiliserons plus loin (pour la partie II)

<sup>3</sup><http://www.cs.toronto.edu/~radford/>

<sup>4</sup><http://www.stanford.edu/~divad/>

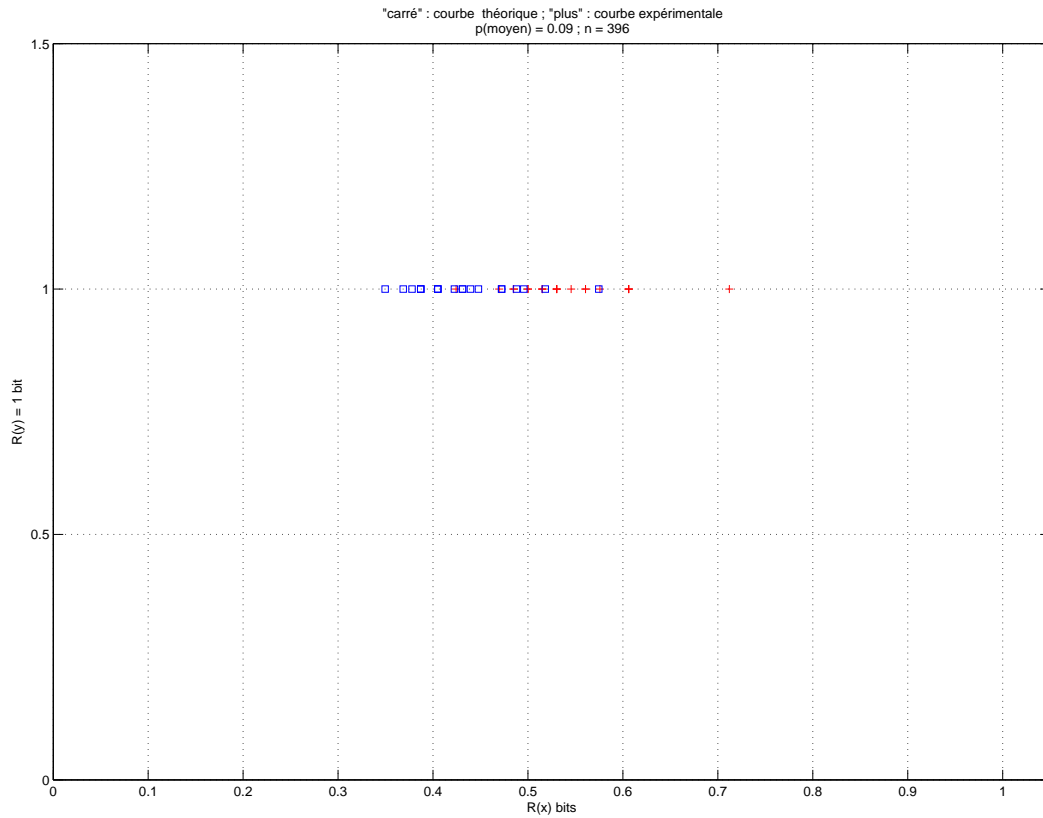


FIG. 6.4 – Approche de la borne de Slepian-Wolf par poinçonnage du syndrome. Si le débit de la source  $Y$  est constante ( $R_Y = 1$ ), le débit de la source  $X$  varie, en recherchant, pour  $p$  donné, le débit minimal. Il est à noter que la borne de Slepian-Wolf n'est pas fixe dans ce cas : il faut comparer chaque point obtenu (*signe plus sur le dessin*) au point théorique (*signe carré*) contrairement aux résultats de la figure 5.2.

Deuxième partie

**Contributions**

## Chapitre 7

# Approche de la borne de Slepian-Wolf : *codage symétrique et adaptatif en débit*

Dans le chapitre 5, les auteurs du modèle SF-ISF ont proposé une méthode qui peut être mise en œuvre avec tout code admettant un ISF (par exemple code convolutif, code turbo ou code LDPC). Dans notre travail, nous nous intéressons au codage LDPC, ce qui simplifie beaucoup le montage de la figure 5.1 car ce codage dispose d’une méthode de décodage particulière<sup>1</sup> pouvant retrouver un mot n’ayant pas forcément le syndrome zéro. La méthode de Varodayan pouvant s’adapter à tout code LDPC<sup>2</sup>, on peut créer une méthode de poinçonnage du syndrome intégrant la méthode SF-ISF.

**Notations :** Dans ce chapitre, on adopte les notations suivantes :

- $X$  et  $Y$  sont deux variables aléatoires, chacun iid, dont les réalisations  $x$  et  $y$ ;  $x$  et  $y$  sont corrélées avec une probabilité  $p = Pr(x \neq y) < \frac{1}{2}$ .  $n$  est la longueur du mot de source  $x$  (ou  $y$ ) à coder en un syndrome  $s_x$  (ou  $s_y$ ) de longueur  $m$ .  $H = [A \ B]$ , de taille  $(m \times n)$ , est la matrice représentant le code LDPC  $(n, k)$  considéré pour effectuer la compression;  $m = n - k$ ;  $A$  et  $B$  sont respectivement de tailles  $(m \times k)$  et  $(m \times m)$ .
- $i \in [1, N]$  est l’itération courante dans la recherche du débit adéquat au décodage de  $z$ . Pour cela, un canal à contre-réaction doit exister entre le codeur et le décodeur.
- $S$  est le syndrome non poinçonné de longueur  $n$ .
- $SA$  est le syndrome accumulé de longueur  $n$ .
- $sa$  est le syndrome accumulé poinçonné de longueur  $m_i$  correspondant à chaque itération.
- $s$  est le syndrome poinçonné de longueur  $m_i$ .  $s_x$  indexe le coset<sup>3</sup> auquel  $x$  appartient.
- On pose  $z = x \oplus y$ .
- $m_0$  est le mot de code tout zéro utilisé pour décoder  $z$ .
- On considère par ailleurs la suite de matrice  $(H_i)_{i=1\dots N} = ([A_i \ B_i])_{i=1\dots N}$ , de tailles  $(m_i \times n)_{i=1\dots N}$ , permettant d’effectuer le décodage adaptatif en débit, comme décrit dans la section 6.2. Ces matrices correspondent à des codes LDPC  $(n, k_i)$  avec la relation évidente :  $\forall i \in [1, N], m_i = n - k_i$ .  $H_j$  est la matrice pour laquelle on estime que  $z$  est convenablement décodé.
- La suite de matrices  $(P_i)_{i=1\dots N}$  représente les matrices des permutations nécessaires à effectuer sur les  $H_i$  pour que les parties  $B_i$  soient inversibles  $\forall i \in [1, N]$ .  $\forall i, P_i$  est de taille  $(n \times n)$ .
- $x_a^b$  représente les bits allant de  $a$  à  $b$  de  $x$ ;  $a, b \in [1, n]$ .
- $P_{i,a\dots b}$  représente les colonnes allant de  $a$  à  $b$  de  $P_i$ ;  $a, b \in [1, k_i]$ .
- “Acc” est l’action d’accumuler, cela peut être représenté sous forme matricielle spécifié dans la section 7.1.
- “P<sub>ç</sub>” est l’action de poinçonner, ce qui peut aussi être mis sous une forme matricielle qu’on spécifie dans la section 7.1.

<sup>1</sup>Cet algorithme est la “propagation de croyance” adapté par Liveris, vu précédemment dans le chapitre 2.

<sup>2</sup>Cela est vrai du moment que ce code peut être poinçonné suivant les directives de la section 6.2.

<sup>3</sup>on rappelle qu’il y a  $2^{m_i}$  index possibles, indexant chacun  $2^{k_i}$  mots, voir 4.1.2

## 7.1 Algorithme développé pour le codage LDPC

### 7.1.1 Principe

Comme on peut décoder directement  $z$  à partir de  $s_z$  par la méthode que Liveris présente dans [15], l'ensemble formé par l'ISF et le décodeur canal, de la figure 5.1, peut être substitué par un seul bloc formé par un décodeur à propagation de croyance; on décode alors  $z = x \oplus y$  à partir du syndrome  $s_z = s_x \oplus s_y$  et du mot de code zéro par la méthode de Liveris.

Ainsi, la première modification que nous proposons pour notre algorithme est une simplification de la méthode "SF-ISF", afin de l'adapter à l'esprit du codage LDPC.

Il faut ensuite adapter le modèle en débit, grâce à la méthode de poinçonnage proposée par Varodayan dans [27]. En fait, une fois le syndrome  $s_z = s_x \oplus s_y$  codé, l'adaptation en débit se déroule au niveau de la recherche du bon mot  $\hat{z}$  (décodeur **(1)** dans le schéma de l'SF-ISF, 5.1) grâce à un canal pourvu d'une contre-réaction entre le codeur et le décodeur. Une fois le bon mot  $z$  retrouvé, on sait à quel débit il a été transmis et on sait donc quelle matrice  $H_j$  utiliser pour retrouver les parties  $x_{k+1}^n$  et  $y_{k+1}^n$  (cette partie correspond au décodeur **(2)** de la démarche SF-ISF).

Le seul problème rencontré est la nécessité de disposer d'une matrice  $H_j = [A_j \ B_j]$  avec une partie  $B_j$  inversible pour pouvoir mettre en œuvre la formule  $x_{k+1}^n = B^{-1}(Ax_1^k \oplus s_x)$ . Ce qu'il faut savoir c'est que pour toutes les matrices  $H_i$  créées par la séparation des nœuds (bits) du syndrome, nous disposons d'un algorithme simple retrouvant les permutations sur les colonnes des  $H_i$  pour isoler dans la partie  $B_i$  des colonnes formant une matrice carrée inversible. Donc, nous disposons d'une matrice de permutation  $P_j$  permettant de passer d'une matrice  $H_j$  décodant  $z$  (et donc décodant  $x_{k+1}^k = y_{k+1}^k \oplus z_{k+1}^k$  et  $y_1^{k_1} = x_1^{k_1} \oplus z_1^{k_1}$ ) à la matrice utilisable pour décoder  $x_{k+1}^n$  et  $y_{k+1}^n$ .

Sur la figure 7.1 est représenté le schéma permettant de tester l'adaptation en débit lors de la compression de deux sources corrélées, conformément à la démarche de Varodayan dans [27], tout en se réservant la possibilité de parcourir la borne de Slepian-Wolf du point A au point B (voir le schéma 4.2), en suivant les principes de l'SF-ISF décrite par Li dans [25].

### 7.1.2 Description de l'algorithme :

Bien que le schéma se comprenne intuitivement, voici les étapes de la compression et de la décompression des mots  $x$  et  $y$  : (*Remarque* : **(1)**, **(2)** et **(3)** se déroulant en parallèle, on ne donne ici d'explications que pour la source  $x$ . Ce sont évidemment les mêmes explications pour la source  $y$ )

- **(0)** : introduction du *canal symétrique binaire* de probabilité<sup>4</sup>  $p < \frac{1}{2}$  entre  $x$  et  $y$ .
- **(1)** calcule le syndrome  $S_x$  non poinçonné du mot  $x$  à compresser.  $\forall h \in [1, n], S_{x(h)} = \sum_{j=1}^n (x_j H_{hj})$ .
- **(2)** accumule le syndrome ainsi formé.  $\forall k \in [2, n], SA_{x(k)} = S_{x_k} \oplus SA_{x(k-1)}$ . Plus simplement, on peut modéliser cette accumulation par la multiplication de  $Sx$  par une *matrice Acc triangulaire inférieure* de taille  $(n \times n)$ .
- **(3)** poinçonne le syndrome accumulé en n'en gardant que  $m_i$  bits parmi  $n$ . La nécessité d'accumuler avant de poinçonner a été soutenue dans la section 6.1. Cette opération peut s'effectuer en multipliant le syndrome précédemment accumulé par une *matrice de taille  $(n \times n)$  comportant des lignes de zéros* là où on veut poinçonner<sup>5</sup>.
- **(4)** est la sommation modulo 2 de  $sa_x^i$  et  $sa_y^i$  pour produire  $sa_z^i$ . À partir de ce nouveau syndrome, notre premier but est de trouver une estimation  $\hat{z}$  du canal équivalent au modèle de la corrélation existante entre  $x$  et  $y$ , introduit en **(0)**.
- **(5)** décode une estimation  $\hat{z}$  du canal équivalent (BSC) entre  $x$  et  $y$ . Ce bloc se subdivise en deux plus petits effectuant une plus petite tâche chacun :
  - Un premier bloc effectue la dé-accumulation du syndrome  $sa_z^i$  reçu en un syndrome  $s_z^i$  exploitable par l'algorithme de propagation de croyance décrit dans la section 2.2.2.

<sup>4</sup>en situation réelle,  $x$  et  $y$  n'ont aucun lien entre eux donc la corrélation  $p$  est a priori peu connue, d'où l'adaptation en débit

<sup>5</sup>intuitivement, on n'en garde que les  $m_i$  premiers bits mais il est possible de supprimer des bits autrement positionnés, pourvu que le décodeur ait connaissance de ces positions

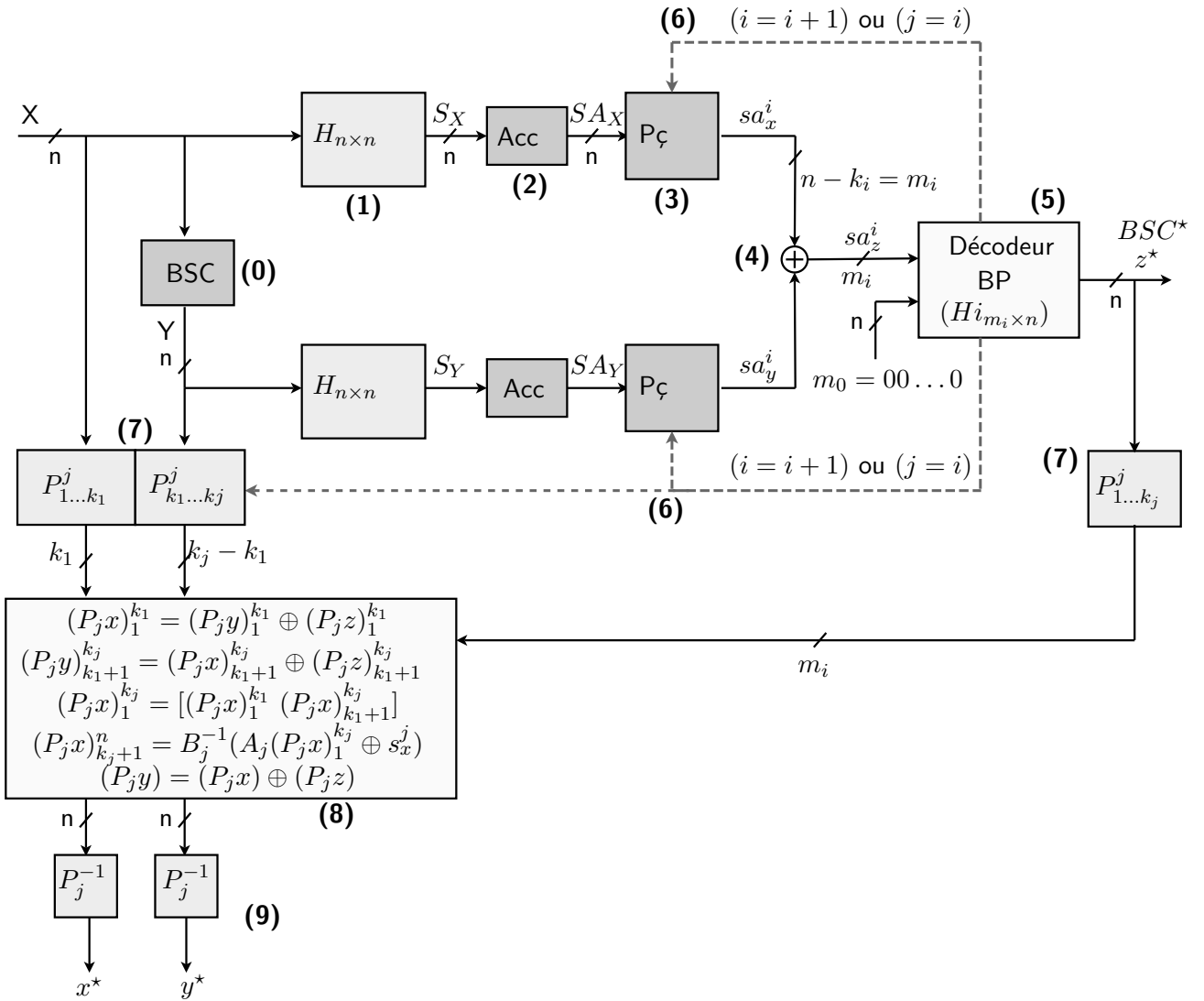


FIG. 7.1 – Déroulement de l’algorithme permettant l’adaptation en débit et le parcours de la borne de Slepian-wolf

- Un deuxième bloc effectue la propagation de croyance, retrouve une estimation  $\hat{z}$  à partir du mot de code  $m_0$  et du syndrome  $s_z^i$ . Si le décodage est effectif (c'est-à-dire si l'algorithme retrouve un mot  $\hat{z}$  vérifiant l'équation  $H_i \hat{z} = s_z^i$ ), alors on va à (7) et on transmet par (6) l'information comme quoi  $j = i$  pour décoder  $z$  sans erreurs, sinon le décodeur demande des bits supplémentaires de la part du codeur, par contre-réaction par la boucle de retour (6).
- (6) est la boucle de retour permettant au décodeur d'informer le codeur sur l'état du décodage. Le décodeur peut informer le codeur que le décodage est effectif avec  $j = i$  courant, et dans ce cas le codeur et le décodeur passent conjointement à l'étape (7), ou alors que le décodage a échoué, et dans ce cas le codeur transmet des bits supplémentaires du syndrome en incrémentant  $i$ .
- (7) utilise la matrice de permutation d'ordre  $j$ ,  $P_j$  pour décider quels bits transmettre au décodeur compte tenu de la permutation à effectuer pour avoir une partie  $B_j$  de  $H_j$  inversible. En fait, on sait que, parmi les  $n$  colonnes de  $H_j$ , il y en a  $m_j$  formant une matrice carrée inversible ; par ailleurs on dispose de la matrice de permutation  $P_j$  pour isoler ces  $m_j$  colonnes dans la partie  $B_j$  qui nous intéresse. Or lors du codage, on ne peut pas utiliser la matrice  $HP$  (matrice permutée) pour encoder car  $P$  dépend de  $j$  et on doit respecter les contraintes de poinçonnage exposées en 6.2 ; notamment on ne doit pas faire de permutations<sup>6</sup> sur les colonnes. Donc on ne transmet pas les bits correspondant aux colonnes formant une matrice carrée inversible, afin de pouvoir les retrouver sans erreurs grâce à la formule  $x_{k_j+1}^n = B_j^{-1}(A_j x_1^{k_j} \oplus s_x^j)$ .  
*Remarque* : il est très important de remarquer qu'on ne doit transmettre les bits systématiques qu'une fois  $z$  décodé, et donc qu'une fois  $j \in [1, N]$  connu parfaitement, pour connaître la position des bits systématiques à transmettre.
- (8) À ce niveau, on connaît tous les "morceaux" nécessaires à la reconstruction de  $(P_j x)$  et  $(P_j y)$ . Les formules sont données sur le schéma, ce sont les mêmes (aux indices et aux matrices de permutations près) que les formules données au 5.1. Afin de retrouver  $x$  et  $y$ , il faut bien veiller à multiplier par  $P_j^{-1}$  pour remettre les bits dans l'ordre où ils ont été introduits dans le codeur (1).
- (9) On obtient les estimations  $\hat{x}$  et  $\hat{y}$ . Le système est maintenant prêt à traiter une autre séquence de bits  $x$  et  $y$  en revenant à (1)...

## 7.2 Résultats

Grâce à la combinaison des codes proposés par Varodayan et MacKay sur leurs sites respectifs, nous sommes capables de générer de bonnes matrices de codes LDPC pour tester notre algorithme et comparer les résultats.

Les tests ont été faits sur une longueur de blocs sources  $n = 2046$  ; pour des raisons de temps nous n'avons pas encore pu les faire sur des longueurs plus importantes vu le temps pris pour retrouver les matrices inverses et les matrices de permutations<sup>7</sup>. Nous disposons de 66 matrices  $H_i$  de tailles allant de  $62 \times 2046$  à  $2046 \times 2046$ , correspondant à des taux de compression conjointe allant de  $\frac{2n}{2n-k} = 1.9412$ ème à 1.

Sur la figure 7.2 est portée la région des points dans la région de Slepian-Wolf que nous avons atteints grâce à notre algorithme.

Le niveau de corrélation  $p$  fluctue autour d'un niveau moyen  $p_{moyen} = 0.1$  fixé. La variation de la corrélation  $p$  entre les deux sources  $X$  et  $Y$  implique le mouvement des points "perpendiculairement" à la borne de Slepian-Wolf, comme sur la figure 6.4 des résultats du modèle de Varodayan *et al.*

La variation de  $k1$ , qui représente la *symétrie* entre les taux de compression de  $X$  et  $Y$ , nous permet de nous déplacer parallèlement à la borne de Slepian-Wolf, comme sur la figure 5.2 des résultats de

<sup>6</sup>Des permutations effectuées sur les lignes ou les colonnes changeraient complètement le code LDPC utilisé. Dans ce cas, on n'utiliserait pas les mêmes matrices pour coder et décoder, ce qui est une aberration.

<sup>7</sup>À titre d'exemple, pour générer les 66 matrices correspondants aux 66 niveaux de compressions du code à  $n = 2046$ , il nous a fallu 4 jours

Li *et al.* Notons que les droites obliques remarquées sur le dessin sont le résultat de cette variation de  $k_1$ , vu que nous menons les tests sur 50 valeurs de  $k_1$  régulièrement espacées.

En moyenne, nos valeurs de  $(R_x + R_y)$  s'écartent de celles théoriques  $H(X, Y)$  de 0.0812 bit et le système a une probabilité d'erreur de l'ordre de  $10^{-4}$ . Ces valeurs sont satisfaisantes, si l'on considère le fait que les deux systèmes précédents (6.4 et 5.2) s'écartaient des valeurs théoriques d'environ 0.1 bit.

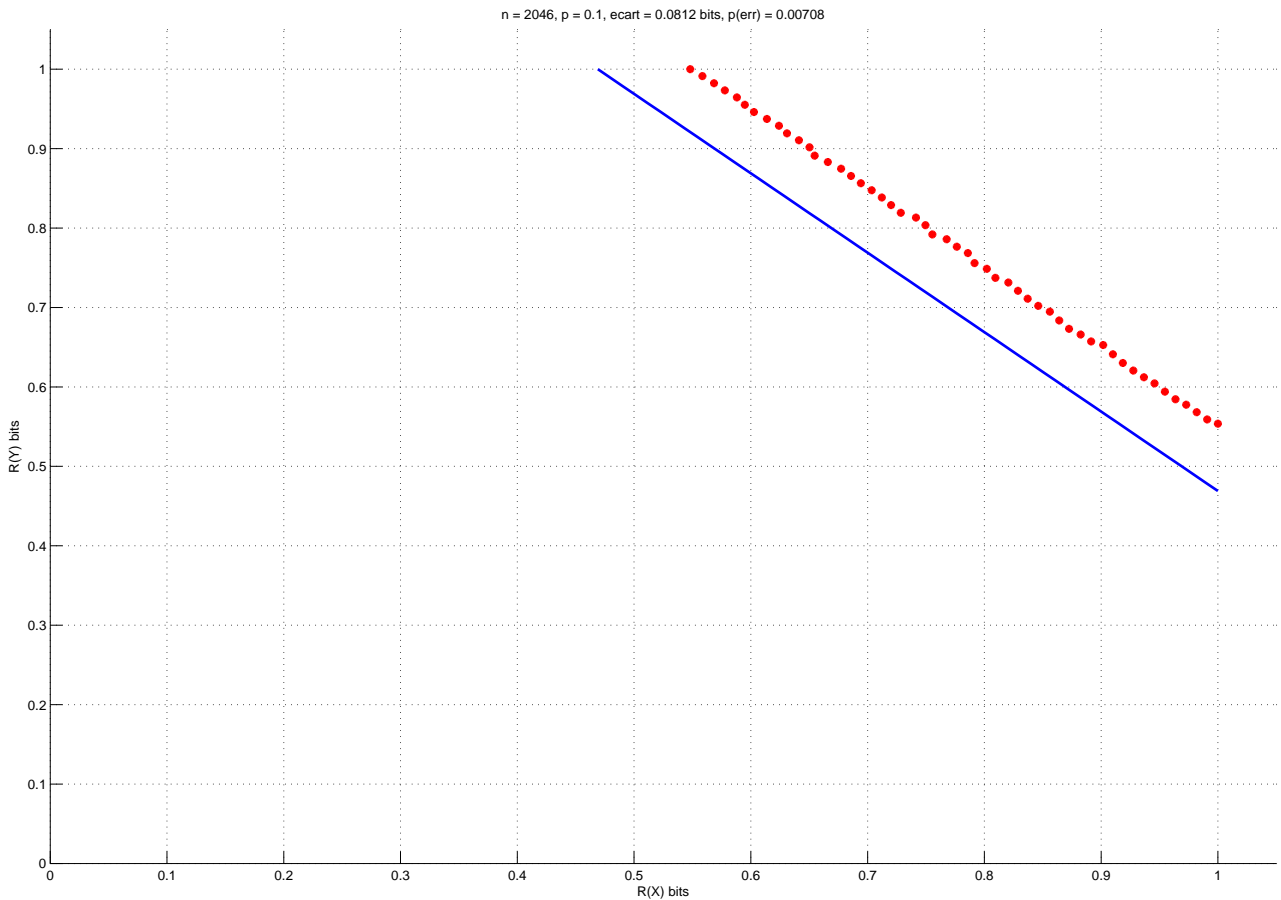


FIG. 7.2 – Avec notre méthode, on assure l'adaptation en débit et le parcours de la borne de Slepian-wolf. Le segment représente les valeurs théoriques et les points "ronds" sont nos valeurs expérimentales ; pour chaque, nous avons tracé la moyenne obtenue.

La figure 7.3 montre quant à elle l'évolution de l'écart par rapport à la borne théorique de Slepian-Wolf pour deux valeurs assez éloignées :  $n = 264$  et  $n = 2046$ . Nous voyons ici que la différence, bien que visible, est minime. En effet, même si  $n = 2046$  semble grand c'est une valeur très faible par rapport aux valeurs de  $n$  habituellement utilisées dans le codage canal : jusqu'à  $n = 10^6$ .

La figure 7.4 présente l'évolution de l'écart par rapport à la borne de Slepian-Wolf pour des valeurs de  $n$  allant de 264 à 2046. Il est bien visible que l'écart diminue avec l'augmentation de  $n$ . Cependant, ce qui est encore plus visible est que la performance des codes dont nous disposons n'a pas la même évolution. En effet, les codes avec  $n = 1452$  forment la meilleure série de codes que nous avons générée.

Enfin, la figure 7.5 analyse l'évolution de la probabilité d'erreurs binaires avec l'évolution de  $n$ . Là encore, la tendance générale vérifie la théorie : plus  $n$  augmente, plus la probabilité d'erreurs binaires diminue. Quoi qu'il en soit, on remarque que le code avec  $n = 1254$  est le meilleur code donnant une probabilité d'erreurs la plus faible.

**Bilan :** Pour les trois comparaisons que nous avons effectuées, la règle générale théorique est observée : on obtiendra de meilleures performances on faisant tendre la longueur de code  $n$  vers l'infini. Cependant, selon la méthode de génération des codes, on observera qu'un code corrigera plus d'erreurs sans permettre un rapport de compression intéressant ou, au contraire, bien compresser mais en laissant passer de nombreuses erreurs. Les codes avec lesquels nous avons effectué nos tests ont été

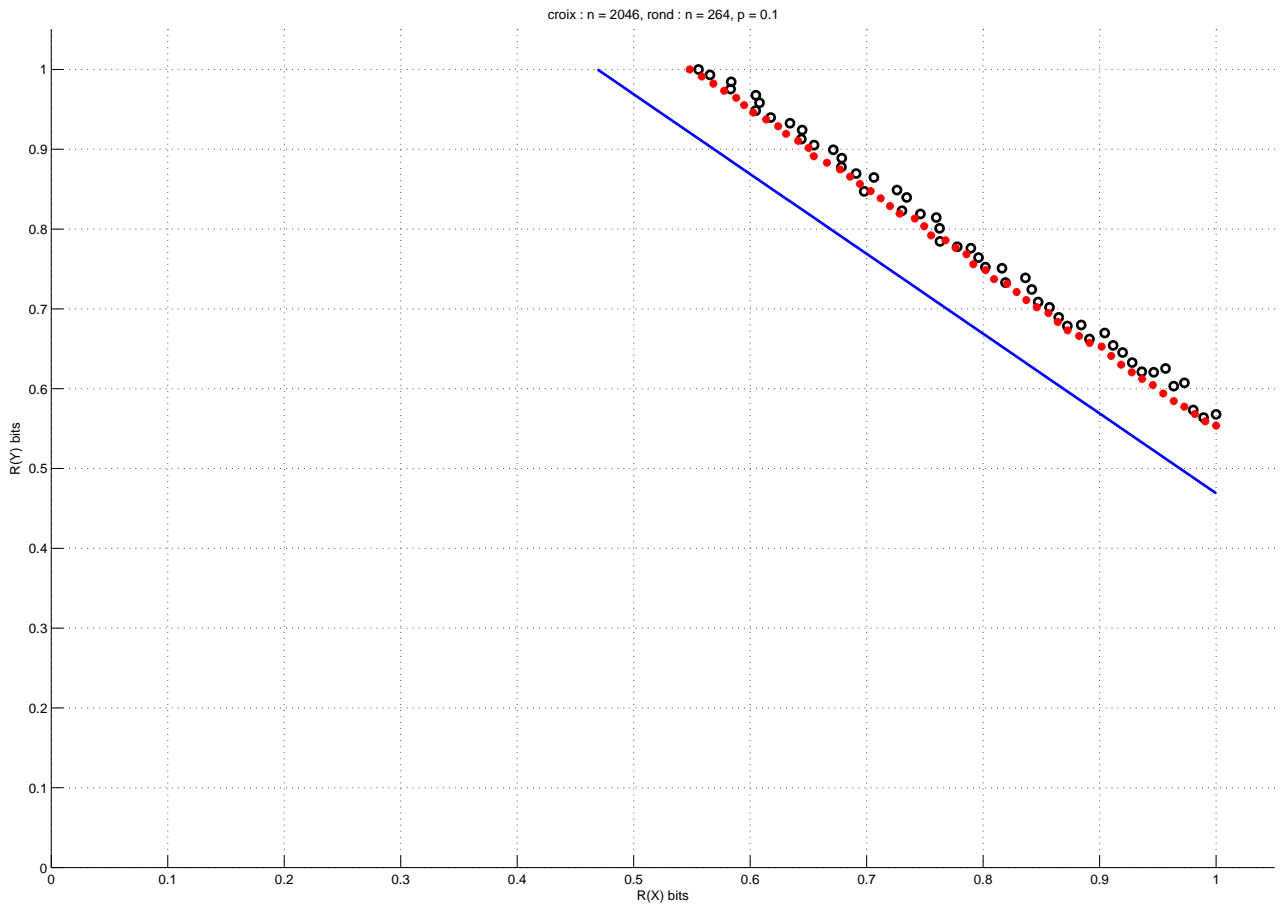


FIG. 7.3 – Comparaison des résultats obtenus avec  $n = 264$  et  $n = 2046$ . Comme prévu, on obtient de meilleurs résultats avec l'augmentation de  $n$ . Cependant, 2046 est une valeur assez faible vu les ordres de grandeurs utilisés dans le codage canal (de l'ordre de  $10^6$ ). Il aurait fallu comparer  $n = 2046$  et  $n = 2.10^5$  pour voir une différence plus significative.

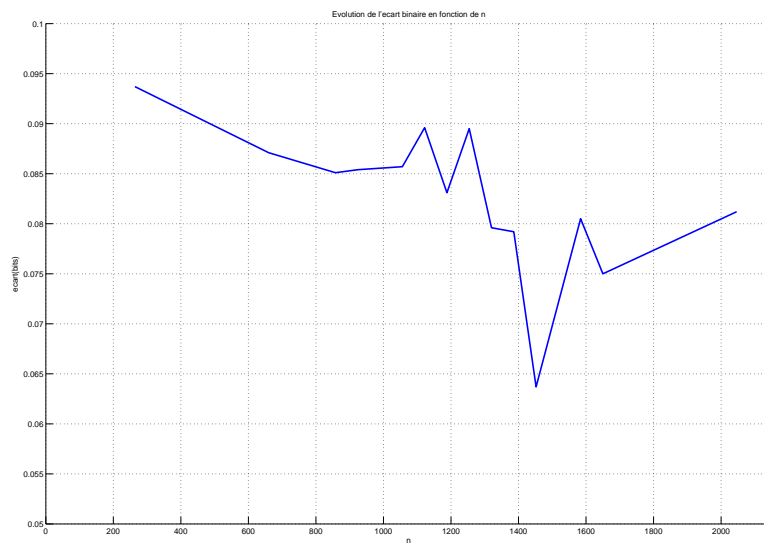


FIG. 7.4 – Comparaison des écarts obtenus pour  $n = 264$  à  $n = 2046$ . Comme prévu, on obtient de meilleurs résultats avec l'augmentation de  $n$ . Cependant, il est très notable que le meilleur code que nous avons est celui avec  $n = 1452$  : ses capacités surpassent celles du code à  $n = 2046$ .

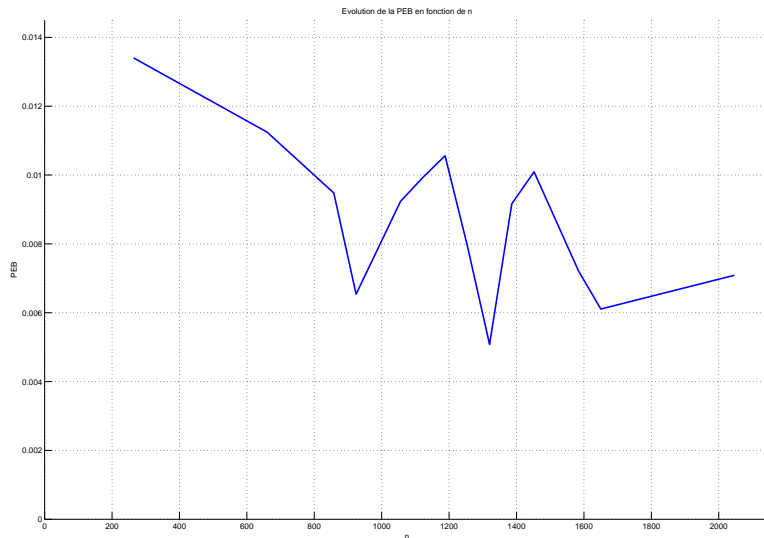


FIG. 7.5 – Evolution de la probabilité d’erreurs obtenue en faisant varier  $n$  entre 264 et 2046. La règle générale se vérifie : plus  $n$  augmente, plus la probabilité d’erreurs binaires diminue; mais il y est visible que le meilleur code est pour  $n = 1254$ .

générés par le même algorithme, et pourtant des incohérences persistent. Le code idéal permettrait de corriger le plus d’erreurs possible, tout en compressant l’information au maximum.

### 7.3 Exemple d’application

Supposons que pour une raison donnée, la source  $X$  ne peut transmettre qu’à  $R_x = r_x$ , quelle que soit le niveau de corrélation  $p$  entre les deux sources. Cela peut être le cas quand  $X$  ne dispose que d’une énergie fixée pour transmettre ses informations. Dans ce cas, la source  $Y$  devra adapter son débit de transmission afin que  $R_y + r_x = H(X, Y)$ .

Soit  $H_{m_j \times n}$  la matrice permettant de décoder  $z = x \oplus y$  sans erreur.  $k_j = n - m_j$  est donc fixé par le débit correspondant. On calcul alors  $k_1$  par la relation :  $r_x = \frac{m_j + k_1^*}{n} \Rightarrow k_1^* = [nr_x] - m$ , où “[ ]” désigne la partie entière. Alors,  $Y$  transmettra au débit  $R_y = \frac{n - k_1^*}{n}$ .

Dans ce cas précis, les points correspondant aux débits admissibles sont alignés sur une droite verticale, décrite sur la figure 7.6, dans la région de Slepian-Wolf.

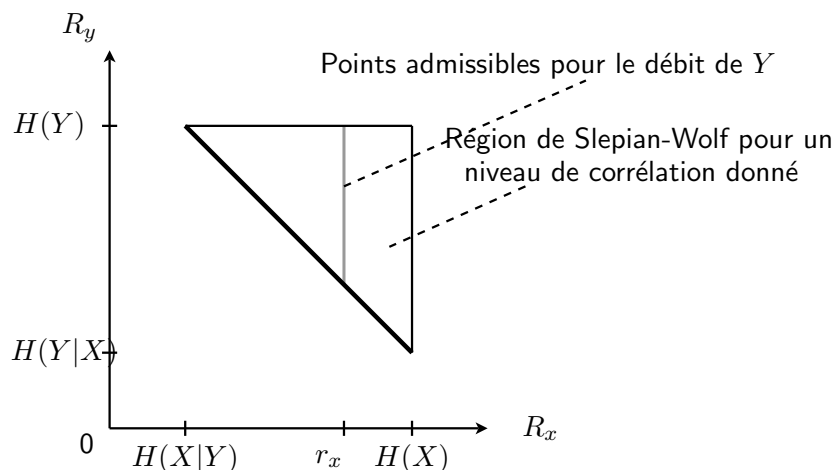


FIG. 7.6 – Quand  $X$  doit envoyer ses informations avec une puissance prédéfinie,  $Y$  doit s’y adapter. Cela forme une droite verticale dans la région de Slepian-Wolf

## 7.4 Problèmes rencontrés

Les deux méthodes utilisées ayant été prouvées optimales par leurs auteurs respectifs, la combinaison des deux pour notre application est optimale, sous réserves que les codes choisis soient aptes à atteindre la borne de Slepian-Wolf. La seule limite que nous connaissons à l'heure actuelle reste la condition imposée sur la partie  $B$  de  $H = [A \ B]$ , difficulté dont nous avons parlé dans la section 5.2.

### 7.4.1 Inversibilité de la partie B

Dans la mise en œuvre de la méthode SF-ISF, il est indispensable de disposer d'une matrice  $B$  inversible en tant que partie à droite de  $H$ . Or, par la méthode de séparation des lignes lors de la construction de la suite de matrices aux taux de compressions décroissants, voir la figure 6.3, si la matrice de taille  $m_i \times n$  présente cette caractéristique, il n'est pas assuré que les  $m_{i+1}$  dernières colonnes de  $H$  soient précisément les  $m_{i+1}$  colonnes libres.

La solution à ce problème a été présentée de façon naturelle dans la section 7.1 : il s'agit d'utiliser la suite de matrices des permutations propres à chaque matrice  $H_i$  de la suite. Lors de la transmission de l'information systématique, on doit être vigilant et transmettre des bits de  $x$  et  $y$  n'étant pas codés par les colonnes de  $H$  formant la matrice carrée inversible.

### 7.4.2 Propagation d'erreurs

Le problème provient de la formule impliquant  $B^{-1}$  dans les équations (5.1). En effet, si  $B$  est, comme  $H$ , une matrice creuse<sup>8</sup>,  $B^{-1}$  est loin d'être creuse. À titre d'exemple, pour une matrice  $H_{228 \times 396}$ , creuse et régulière de distribution de degrés  $\lambda(x) = x^2$  et  $\gamma(x) = \alpha x^5 + \beta x^2$ , comportant 0.0158% de 1 (1429 1 pour être précis), la matrice  $B_{228 \times 228}$  comporte 0.0132% de 1, alors que  $B_{228 \times 228}^{-1}$  comporte 0.4011% de 1. La matrice  $B^{-1}$  n'est donc pas creuse, et pour la formule (5.1), les conséquences sont désastreuses.

Pour s'en convaincre, essayons de voir ce qui arrive quand il y a une erreur de reconstruction dans la partie  $[1 \dots k]$  de  $x$  avant l'utilisation de  $B^{-1}$ . Posons  $X_1 = Ax_1^k$ ,  $X_2 = Ax_1^k + s_x$  et  $X_3 = B^{-1}X_2$ . Comme vu sur la figure 7.7,  $X_1$  et  $X_2$  ne comporteront "que" 3 erreurs alors que  $X_3$  sera en quasi totalité erroné, surtout si les trois bits erronés de  $X_2$  correspondent avec 3 colonnes de  $B^{-1}$  n'ayant pas beaucoup de 1 en commun.

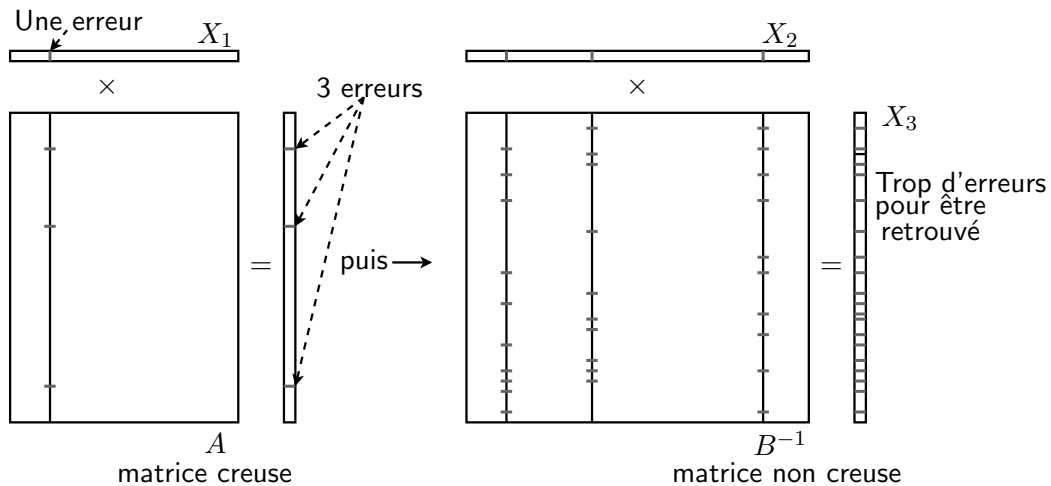


FIG. 7.7 – Répercussion des bits erronés de  $x_1^k = X_1$  sur  $x_{k+1}^n = X_3$

Pour corriger ce problème, on pourrait de créer une matrice  $B$  moins creuse, et ainsi obtenir une inverse  $B^{-1}$  creuse. Malheureusement ce n'est pas si simple que ça car, comme montré par MacKay dans [16], plus la matrice représentant le graphe de Tanner se remplit, moins il y a de chance que le décodage de  $z$  par propagation de croyance converge.

On pourrait se dire aussi que, pour "alléger" au maximum la matrice de codage, il vaudrait mieux prendre la matrice inversible la moins remplie possible, c'est-à-dire la matrice identité ; là encore [16]

<sup>8</sup>Une matrice "creuse" comporte beaucoup plus de 0 que de 1

et [35] stipulent clairement que le décodage serait appauvri et convergerait vers un mot différent de  $z$ , le degré des colonnes de  $B$  étant de 1.

Mise à part la fragilité du décodage, utiliser des matrices inverses de matrices creuses pose un autre problème : c'est celui lié à la mémoire de travail. Non seulement le calcul des inverses est long (même si l'algorithme que nous proposons dans l'annexe 10 est très efficace) vu la taille des blocs mise en jeu, leur stockage et l'accès aux valeurs devient aussi un problème. Pour y remédier, Varodayan dans l'implémentation de [27] sur son site propose une représentation très intéressante en n'enregistrant dans un fichier que les emplacements des 1. Cette solution est intéressante pour l'accès aux matrices  $H$  et  $A$  (creuses), mais le problème reste entier quand il s'agit de stocker les emplacements des  $0.4 \cdot n^2$  quand  $n$  dépasse 3000 (et avoisine 10000 pour certaines applications)...

La conséquence de ces remarques est que la méthode de résolution utilisant  $B^{-1}$  est fragile et lourde, d'où l'utilité pour nous de trouver des méthodes de décodage de la partie  $k + 1 \dots n$  des mots à transmettre plus efficaces.

## 7.5 Solutions envisagées

Une façon alternative de retrouver  $x_{k+1}^n$  est l'élimination de Gauss-Jordan ou pivot de Gauss, présenté dans l'annexe 11. Cette méthode a pour avantage de résoudre le système de  $m$  équations à  $m$  inconnues sans disposer explicitement de la matrice  $B^{-1}$ . C'est cette méthode que nous mettons en œuvre lors de nos tests.

*Remarque* : Si l'on dispose explicitement de  $B^{-1}$  sans avoir à la rechercher dans un fichier à chaque fois que l'on en a besoin, les formules (5.1) sont plus efficaces.

Dans le chapitre 8, nous proposons une méthode de résolution itérative, se rapprochant de la propagation de croyance. Elle a été proposée par Roumy *et al.* dans leur papier [22] de 2007. Cette nouvelle méthode utilise les graphes combinés du code LDPC, et des informations traduites par  $z = x \oplus y$ .

## 7.6 Codage symétrique dans le cadre du codage convolutif

### 7.6.1 Cas particulier du codeur convolutif de rapport de compression 1 :2

#### Modélisation du codeur

Soit  $G(D)$  la matrice polynomiale génératrice du code canal  $n : k$ . Soit  $T(D)$  la matrice vérifiant  $GT^T = 0$ , la matrice qui nous permet de tracer le diagramme du code (et par conséquent le treillis du code).

La description de notre algorithme passe par la représentation matricielle du codeur convolutif, et utilise les descriptions matricielle et en treillis du code pour le décodage. Pour plus de détails, voyons le cas de la compression 1 : 2 (deux entrées pour une sortie) avec le codeur convolutif. Pour une longueur de contrainte  $L = 3$ , nous avons par exemple  $T(D) = [7 \ 5]$  pour la matrice donnant le diagramme<sup>9</sup>. Voici donc la représentation en diagramme de ce codeur, sur la figure 7.8.

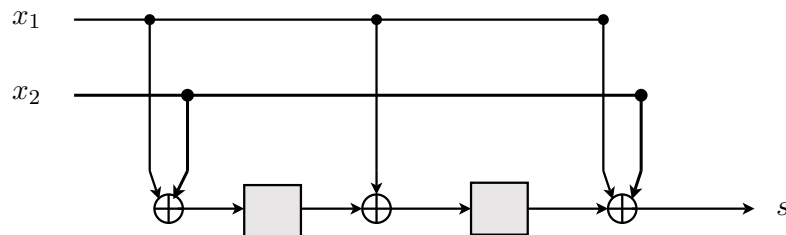


FIG. 7.8 – Diagramme du codeur convolutif 1 : 2.

<sup>9</sup>Par la suite, nous noterons  $T(D)$  la matrice permettant de tracer le diagramme (pas exemple celui de la figure 7.8)



## 7.6.2 Généralisation à tout code convolutif non récursif

### Construction de la matrice $H$

La matrice  $H$  est déduite immédiatement de la matrice  $T$  du code, en répétant le bloc du codeur de façon cyclique. La première chose à faire est de noter les coefficients de  $T$  en binaire, avec un nombre de bits correspondant à la longueur de contrainte du codeur. Par exemple, pour le codeur au rapport de compression 2 : 3 ( $E = 3$  entrées et  $S = 2$  sorties), de longueur de contrainte  $L = 3$

$$T_{10} = \begin{pmatrix} 5 & 3 & 7 \\ 7 & 5 & 3 \end{pmatrix} \Rightarrow T_2 = \begin{pmatrix} 101 & 011 & 111 \\ 111 & 101 & 011 \end{pmatrix}$$

Rappelons que cette matrice correspond à la matrice polynômiale :

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

Le diagramme d'un tel code est représenté sur la figure 7.9.

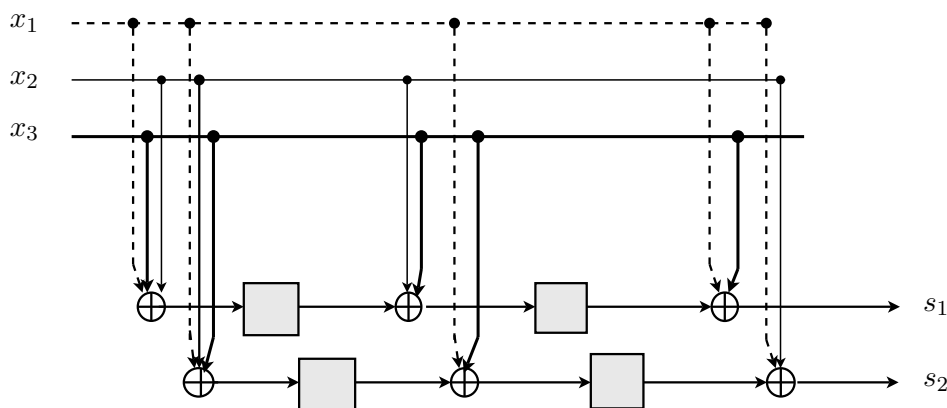


FIG. 7.9 – Diagramme du codeur convolutif 2 : 3.

Pour former un bloc de la matrice  $H$ , il faut mettre les bits dans un ordre particulier, pour prendre en compte les cellules de mémoires du code. La figure 7.10 montre comment s'effectue cette permutation.

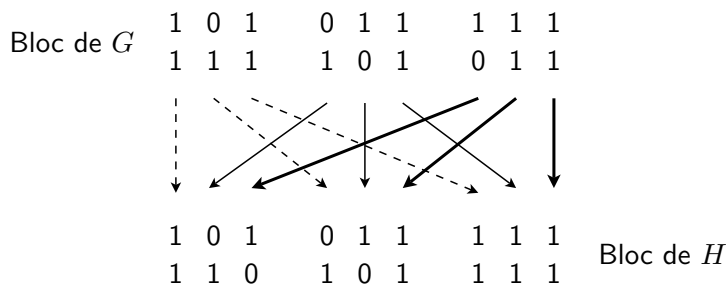


FIG. 7.10 – Construction d'un bloc de la matrice  $H$  du code 2 : 3.

En un mot, le  $j$ -ième bit du  $i$ -ème bloc de  $E$  bits dans  $G$  se retrouve  $i$ -ième bit du  $j$ -ième bloc de  $E$  bits dans  $H$ , pour chaque ligne de  $T$ . Chaque ligne de  $T$  donnera une ligne de  $H$ . Enfin, pour coder le bloc de  $L.E$  bits suivants de  $x$ , on décale le bloc de  $(S \times L.E)$  bits de  $H$  de  $E$  bits vers la droite. Pour coder  $n$  bits en entrée, on répète ce bloc élémentaire de  $H \frac{n}{E}$  fois. Enfin, comme les bits d'états du codeur sont à 0 pour les premiers bits codés, on enlève les  $E.S$  premières colonnes de la matrice obtenue.

**Exemple :** pour  $n = 15$ , la matrice obtenue par concaténations de blocs élémentaires est entre " ( )" et la matrice  $H$  finale est entre "[ ]", dans l'équation (7.2) :

$$H_{10 \times 15} = \left( \begin{array}{c|c} \begin{matrix} 1 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{matrix} & \begin{bmatrix} \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} \end{bmatrix} \end{array} \right) \quad (7.2)$$

### Recherche de la matrice $B$ inversible

La matrice  $H$  obtenue par cette méthode possède  $m$  lignes et  $\frac{E}{5}m$  colonnes. Afin de pouvoir résoudre le système de  $m$  équations à  $m$  inconnues, il nous faut trouver parmi les colonnes de  $H$ ,  $m$  colonnes indépendantes. Il n'y a pas de règle particulière pour trouver ces colonnes, le bon sens suffit. La seule remarque que nous ferons ici est que ces colonnes devraient être extraites de chaque "période" de la matrice : c'est dans ce cas-là qu'elles ont le plus de chance d'être indépendantes.

### Remarque

La matrice (entre crochets) de l'équation (7.2) est intéressante car c'est un *contre-exemple* : les deux premières lignes sont identiques (la matrice est de rang  $m - 1$ ). Ce code ne permettra pas de bonnes performances pour un codage distribué symétrique ; en effet, 1 bit devra être *deviné*, ce qui signifie qu'on devra essayer de décoder les autres bits avec des valeurs différentes de ce bit et voir quel couple de mots  $(\hat{x}, \hat{y})$  résultats possède les syndromes  $s_x$  et  $s_y$  ainsi que la différence  $z$ .

Cette démarche peut doubler le temps de décodage de chaque mot de code, sauf si l'on devine du premier coup la valeur réelle du bit. Une autre solution, plus réaliste, consiste à envoyer un bit supplémentaire pour décoder chaque couple de mots. On n'envoie ce bit que du côté de  $x$  ou  $y$  : la disponibilité de  $z$  permettra de toute façon de retrouver l'autre.

Cette dernière solution dégrade les performances du code comme on pouvait s'y attendre. Cependant, vu la longueur des mots traités, cette dégradation est tout à fait acceptable : le rapport de compression passera de  $\frac{2n-k}{2n}$  à  $\frac{2n-k+1}{2n}$ , soit une perte de moins de  $10^{-5}$  pour une longueur de code  $n$  donnée de  $10^5$ .

En règle générale, si la matrice  $B$  choisie est de rang  $m - g, \forall g$ , il faudra transmettre  $g$  bits supplémentaires. Quand la longueur de contrainte du code augmente, il devient rare que la matrice  $B$  ne soit pas de rang plein.

### 7.6.3 Codes convolutifs récursifs

#### Équations du codeur

Dans le cas du code convolutif 1 : 2 décrit précédemment, l'introduction de la récursivité revient à changer  $T(D) = [7 \ 5]$  en  $T(D) = [1 \ \frac{5}{7}]$ . Dans ce cas, le syndrome dépend aussi du syndrome aux deux états précédents. En effet,

$$\begin{aligned} s &= x_1 \oplus x_2 \frac{1 + D^2}{1 + D + D^2} \\ \Leftrightarrow s(1 + D + D^2) &= x_1(1 + D + D^2) \oplus x_2(1 + D^2) \\ \Leftrightarrow s &= x_1(1 + D + D^2) \oplus x_2(1 + D^2) \oplus s(D + D^2) \end{aligned}$$

Cette équation exprime ni plus ni moins que cette récursivité peut se modéliser comme l'accumulation du syndrome avec ses deux états précédents. D'autre part son diagramme devient celui qui est représenté sur la figure 7.11.

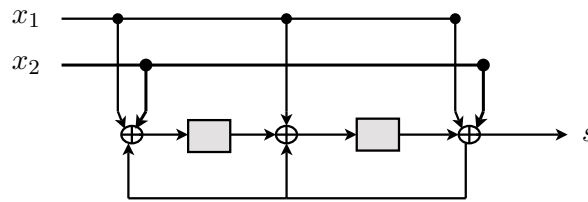


FIG. 7.11 – Diagramme du codeur convolutif 1 : 2 récursif.

## Description de l'algorithme de codage et décodage symétrique

Les équations précédentes nous amènent à décrire l'algorithme de la manière suivante :

1. Calcul et transmission des deux syndromes  $s_x$  et  $s_y$  selon le codeur convolutif de la figure 7.8 ; transmission des parties  $x_1^{k_1}$  et  $y_{k_1+1}^k$  ;
2. Sommation des deux syndromes pour obtenir  $s_z = s_x \oplus s_y$ , le syndrome de  $z = x \oplus y$  ;
3. Accumulation du syndrome  $s_z$  pour modéliser la récursivité du codeur convolutif ;
4. Décodage du syndrome ainsi accumulé selon l'algorithme décrite dans la section 3.3.2, afin de trouver le mot  $z$  le plus proche de  $m_0 = 00 \dots 0$  ayant le syndrome  $s_z$  ;
5. A partir de  $z$ , calcul des parties complémentées de  $x$  et  $y$  :  $x_{k_1+1}^k$  et  $y_1^{k_1}$  ;
6. Mise en oeuvre de la formule  $x_{k_1+1}^n = B^{-1}(A.x_1^k \oplus s_x)$  et  $y_{k_1+1}^n = B^{-1}(A.y_1^k \oplus s_y)$ , où  $A$  et  $B$  sont les matrices décrites dans la section 7.6.1, en prenant bien soin de l'ordre des bits ainsi manipulés ; cette étape peut aussi s'effectuer selon les autres principes de décodage décrits dans la section 7.6.1, le troisième algorithme permettant de décoder au fur et à mesure de la réception des bits des syndromes et des bits systématiques  $z$ ,  $x$  et  $y$ .

Si nous avons pris ici l'exemple assez simple du code convolutif récursif non systématique 1 : 2, cette méthode peut très bien être généralisée à tout code récursif, en trouvant la matrice  $H$  n'incluant pas la récursivité, puis en cherchant les manipulations à effectuer au niveau binaire, afin de modéliser la récursivité, et enfin décoder les bits transmis grâce à la partie  $B$  inversible de la matrice  $H$ . Dans tous les cas,  $z$  est décodé selon l'algorithme de la section 3.3.2.

Afin de pouvoir effectuer un codage distribué symétrique des sources corrélées, notre solution est d'utiliser la représentation matricielle des codes convolutifs. Si le code convolutif choisi est assez puissant pour atteindre la capacité du canal BSC équivalent entre les deux sources, le décodage de  $z$  est assuré.

Néanmoins, les performances atteintes dans le cas des codes convolutifs ne sont pas extraordinaires : par exemple, pour une matrice de taille  $(100000 \times 200000)$ , le niveau maximum de corrélation admettant un taux d'erreur binaire acceptable (de l'ordre de  $10^{-4}$ ) est  $p = 0.005$  ; rappelons que pour un code LDPC dont la taille du bloc est  $n = 2046$ , nous arrivons à coder avec un TEB équivalent des sources corrélées de  $p = 0.1$  (voir 7.2).

Notons enfin que si  $z$  est mal décodé, il y a très peu de chance que  $x$  ou  $y$  soient recouverts sans erreurs.

Pour accroître les performances de notre codeur, il semble donc indispensable de nous pencher sur l'extension de cet algorithme dans le cas des codes turbo (en un mot, c'est la mise en cascade de deux ou plusieurs codes convolutifs intercalés d'un entrelaceur, augmentant ainsi leurs capacités de décodage).

Un autre travail à suivre est l'application de notre algorithme pour le codage distribué convolutif (puis turbo) *adaptatif* des sources corrélées : cela passera par la transmission de syndromes poinçonnés pour le décodage.

# Chapitre 8

## Travaux à venir

### 8.1 Décodage itératif par propagation de croyance conjointe

Comme nous l'avons remarqué dans la section 7.4, la méthode de résolution de  $x$  et  $y$  à partir de  $z$ ,  $s_x$ ,  $s_y$ ,  $x_1^{k_1}$  et  $y_{k_1+1}^k$  par la méthode de partitionnement du SF présente quelques lacunes dont nous aimerions nous passer. Une solution est l'application du décodage itératif proposé par A. Roumy et D. Declercq en mars 2007, dans [22]. Il s'agit d'une méthode permettant de concevoir des codes LDPC pour plusieurs sources avec un décodage conjoint par propagation de croyance sur le graphe conjoint du cas à deux sources. Leur conception des codes LDPC est basée sur le canal gaussien à accès multiple (MAC, *multiple access channel*). L'application de leur méthode sur le cas binaire ne devrait pas poser trop de difficultés.

#### 8.1.1 Présentation de l'algorithme

Dans le cadre du canal MAC gaussien à deux sources, on considère les deux sources indépendantes  $x^{[1]}$  et  $x^{[2]}$  à transmettre à un seul récepteur. Chaque source est codée par un code LDPC irrégulier (pouvant quand même appartenir au même ensemble de codes LDPC), avec une longueur de mot de code  $N$ , leurs puissances respectives reçues sont notées  $\sigma_i^2$ . Les mots de codes sont codées via une modulation BPSK et le modèle discret synchrone de la transmission au moment  $n$  vérifie  $\forall n \in [0, N - 1]$ ,  $y_n = \sigma_1 x_n^{[1]} + \sigma_2 x_n^{[2]} + \omega_n = [\sigma_1 \ \sigma_2] Z_n + \omega_n$ .

On considère le canal MAC à deux sources à *débit égal/puissance égale*, c'est-à-dire  $R_x = R_y = R$  et  $\sigma_1 = \sigma_2 = 1$ . On note  $Z_n = [x_n^{[1]}, x_n^{[2]}]^T$  le *vecteur d'état* du canal à plusieurs sources, et  $\omega_n$  est un bruit blanc gaussien additif de moyenne nulle et de variance  $\sigma^2$  :  $\omega_n \sim N(0, \sigma^2)$ .

Pour conjointement décoder les deux sources, on considère le *graphe de Tanner* [12] du système complet à plusieurs sources et effectuer plusieurs itérations de propagation de croyance. Le graphe de Tanner du LDPC-MAC à deux sources est composé des deux graphes LDPC, connectés entre eux par des nœuds de fonction qui représentent leur lien avec le *nœud de contrôle de l'état*. On a représenté sur la figure 8.1 le voisinage du nœud de contrôle de l'état et les messages sur les liens (mis à jour durant chaque itération du décodage).

Sur la figure 8.1, les nœuds de chaque graphe LDPC sont des *nœuds de variables* (information) et les *nœuds de contrôle* (syndrome), comme dans la section 2.2.2. Appelons  $m_{ab}^{[k]}$  le message que le nœud  $a$  envoie vers le nœud  $b$  pour la source  $k$ ;  $a, b$  peut être soit  $v$  pour un nœud de variable,  $c$  pour un nœud de contrôle ou  $s$  pour un nœud de contrôle de l'état.  $P$  est le message des probabilités provenant de l'observation du canal  $y$ . C'est un vecteur à quatre composantes qui est spécifié par les équations (8.1).

À l'initialisation, les messages log-likelihood sont calculés à partir de l'observation du canal  $y$ . La règle de mise à jour des messages suit celle du décodage LDPC par propagation de croyance usuelle pour tous les messages dans le graphe ( $m_{cv}^{[i]}, m_{vc}^{[i]}, m_{vs}^{[i]}$ ). On doit de plus spécifier la règle de mise à jour pour les messages des nœuds de contrôle de l'état, pour une description complète de l'algorithme :

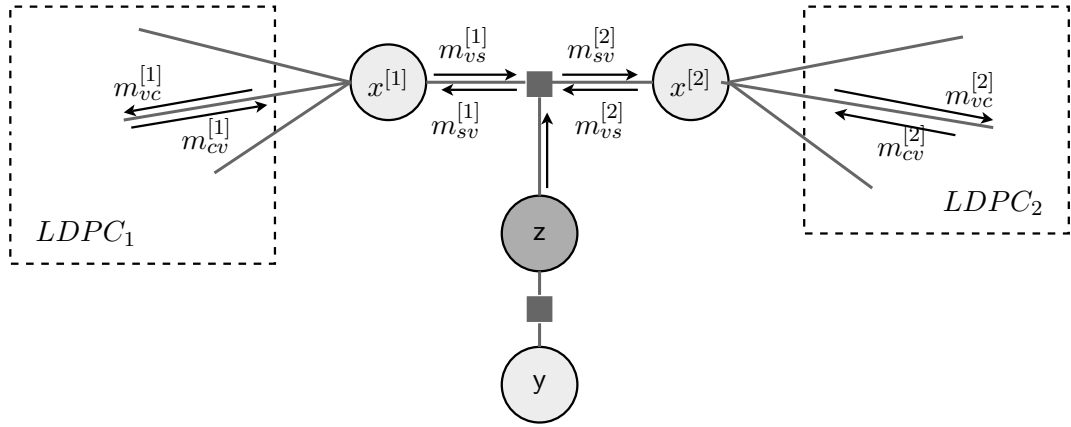


FIG. 8.1 – Graphe de Tanner du canal à accès multiple synchrone à deux sources. Zoom sur le voisinage du nœud de contrôle de l'état

$$P = \begin{bmatrix} P_{00} \\ P_{01} \\ P_{10} \\ P_{11} \end{bmatrix} = \begin{bmatrix} p(y|Z = [+1 \ +1]^T) \\ p(y|Z = [+1 \ -1]^T) \\ p(y|Z = [-1 \ +1]^T) \\ p(y|Z = [-1 \ -1]^T) \end{bmatrix} \quad \begin{cases} m_{sv}^{[1]} = \log \frac{P_{00}e^{m_{vs}^{[2]}} + P_{01}}{P_{10}e^{m_{vs}^{[2]}} + P_{11}} \\ m_{sv}^{[2]} = \log \frac{P_{00}e^{m_{vs}^{[1]}} + P_{10}}{P_{01}e^{m_{vs}^{[1]}} + P_{11}} \end{cases} \quad (8.1)$$

La difficulté de la mise en œuvre de cet algorithme est liée à la limite du décodage des codes LDPC sur un canal à effacement, présenté dans la section 2.2.3. Qui plus est, la difficulté est appuyée par l'absence de toute information adjacente pour le décodage des bits "effacés" lors de la compression des sources corrélées. Cependant, nous sommes persuadés que cet algorithme mérite d'être approfondie.

Notre adaptation de cette méthode au codage de sources distribuées, à deux sources corrélées, avec adaptation en débit et parcours intégral de la borne de Slepian-Wolf, devra prendre en compte le passage du cas gaussien au cas binaire, et le fait que l'information adjacente  $y$  ( $\sim$  observation du canal) n'est pas disponible entièrement au niveau du décodeur.

## 8.2 Codage symétrique utilisant les turbo-codes

Un code turbo repose sur la mise en cascade parallèle de deux codeurs convolutifs séparés par un entrelaceur. Nous savons quels moyens utiliser pour coder symétriquement deux sources corrélées avec les codes convolutifs, et qu'il y a un moyen de représenter le code turbo sous forme matricielle, c'est-à-dire de le considérer comme un code en bloc linéaire (voir annexe 13).

La difficulté ici est liée à la présence de l'entrelaceur, élément clé du codeur turbo, de taille fixe. Rappelons que le codeur convolutif est de taille infinie et que ses mots d'entrées peuvent être décodés au fur et à mesure de la réception de chaque bit du syndrome. Le problème est donc de savoir s'il est possible de faire un tel décodage, tout en restant dans l'esprit du codage turbo (sans passer par un décodage de type propagation de croyance).

L'annexe 13 présente le principe de représentation des codes turbo sous forme matricielle. Une fois la forme matricielle d'un code turbo explicitée, la piste de travail consiste à adapter notre algorithme de la section 7.6 pour le codage turbo.

Enfin, comme dans le cas du codage LDPC, il serait intéressant de développer un algorithme de décodage itératif utilisant le treillis du code pour décoder  $x_{k+1}^n$  et  $y_{k+1}^n$ .

## Conclusion

Au terme de ces six mois de stage à l'Irisa, plusieurs acquis viennent consolider mes compétences professionnelles. Ce sont le codage distribué et la maîtrise des codages LDPC, convolutif et turbo.

La majeure partie de mon temps a été employée à répertorier ce qui a déjà été conçu dans le domaine du codage de sources distribué, sur la manière d'atteindre la limite de compression de Slepian-Wolf et de parcourir la borne d'un point à l'autre, sans restrictions. Ainsi, l'implémentation de ces méthodes et la fusion des deux méthodes nous a permis de concevoir une approche adaptative en débit originale et permettant de parcourir toute la région, dans la partie II. Les deux méthodes ont été modifiées afin que notre contribution puisse être implémentée avec tout code LDPC, ou convolutif, existant, et non pas seulement les codes systématiques. Sous réserves que le code LDPC, ou convolutif, choisi est assez performant, la compression des deux sources atteint la limite de Slepian-Wolf.

Bien que cette approche présente quelques lacunes, notamment au niveau de la durée de décodage d'un mot transmis, elle peut encore être améliorée, comme présenté dans le chapitre 8, en s'inspirant par exemple de [22]. D'autre part, avec l'accumulation de l'expérience dans le domaine du codage de sources distribué, nous devrions être capables d'ici peu de proposer une approche originale de la borne de Slepian-Wolf, et une application sur les sources non discrètes (gaussiennes par exemple) devrait être envisagée.

Pour finir, il est à noter que ce stage se poursuivra avec une thèse qui est sa suite logique : “*Codage de sources distribué dans le contexte multi-caméra*”, à effectuer dans la même équipe TEMICS. Il s'agira là d'exploiter la corrélation existante entre les différents capteurs vidéos, mais aussi entre les différentes trames des images capturées.

# Annexes

**Note :** Ces annexes ne présentent que brièvement les algorithmes qui ont été utiles dans l'accomplissement de la mission. Notamment, les codes c++ conçus lors du stage utilisés ne sont pas donnés ici.

## Chapitre 9

# Conception d'un code LDPC aux bons rendements

Cette méthode proposée par Neal est une implémentation des consignes données par Gallager dans [7] en 1963, puis plus récemment par MacKay et lui-même en 1999 dans [16], en ce qui concerne la création de “bonnes” matrices de codes LDPC.

Ces codes peuvent être construits de nombreuses manières différentes, qui impliquent généralement le choix arbitraire de la position des 1 dans la matrice de contrôle de parité. Ces méthodes ont cela en commun qu'elles donnent des matrices de codes LDPC contenant le même nombre de 1 dans chacune des colonnes, pour toutes les tailles de matrices. En conséquence, lorsque la taille  $n$  des blocs de mots de codes à compresser devient importante, ces matrices sont très creuses.

La création d'une matrice aux performances correctes passe par les étapes suivantes :

1. Créer une matrice de contrôle de parité, par exemple comme [16].
2. Ajouter des 1 dans la matrice pour éviter qu'une ligne ne contienne pas de 1 (lignes qui sont redondantes), ou ne contienne qu'un seul 1 (dans quel cas les bits de mots de code correspondants seront toujours décodés comme étant 0. La place des 1 dans ces lignes sont choisies aléatoirement.
3. Si la matrice de contrôle de parité initialement créée en 1. avait un nombre pair de 1 dans chaque colonne, ajouter davantage de 1 pour éviter que les lignes se s'ajoutent à 0 (alors au moins un nœud de syndrome est redondant). Donc, on doit ajouter au moins deux 1 (parce qu'il est préférable pour la somme des 1 d'une ligne de n'y avoir qu'un 1) à des positions sélectionnés aléatoirement dans la matrice. Cependant, le nombre de 1 ajoutés à cette étape est réduit par celui déjà ajouté à l'étape 2. (*Remarque* : Même si les nœuds de contrôles redondants ne sont pas désastreux, il vaut mieux les éviter).
4. Si nécessaire, il faut essayer de minimiser les situations où une paire de colonnes ont deux 1 dans une même paire de lignes ; cela correspond à un cycle de longueur 4 dans le graphe de Tanner. Au cas où une telle situation se présente, un des 1 en question est déplacé aléatoirement dans sa colonne ; et on recommence jusqu'à ce qu'aucune situation pareille n'est rencontrée, ou après 10 itérations des colonnes ont échoué à les enlever.

## Chapitre 10

# Inversion d'une matrice $H$ de taille quelconque par la *réduction de Gauss*

Soit une matrice binaire  $H$  de taille  $(m \times n)$ . Supposons que  $m \leq n$ . Alors pour que  $H$  soit inversible, il suffit que  $\text{rang}(H) = m$ ; et dans ce cas,  $H_{n \times m}^{-1}$  peut être calculée selon l'algorithme de *réduction de Gauss*.

L'idée est de trouver une matrice  $H^{-1}$  telle que  $HH^{-1} = I$ . Pour cela, comme montré sur la figure 10.1, on part de l'égalité  $HI_{(n \times n)} = H$  puis on réduit  $H$  vers la matrice  $[I_{(m \times m)} \ 0]$  en effectuant des opérations de permutations et de combinaisons linéaires sur les colonnes. En effectuant les mêmes permutations et combinaisons linéaires avec les colonnes de  $I_{(n \times n)}$ , on obtient la matrice  $[H^{-1} \ G]$ , où  $G$  est une matrice  $((n \times n - m))$  vérifiant  $HG = 0$ .

$$\begin{array}{c}
 \boxed{H_{m \times n}} \quad \boxed{I_{n \times n}} = \boxed{H_{m \times n}} \quad \leftrightarrow \quad \boxed{H_{m \times n}} \quad \boxed{\begin{array}{c} H_{n \times m}^{-1} \\ \vdots \\ G_{n \times n-m} \end{array}} = \boxed{\begin{array}{c} I_{m \times m} \\ \vdots \\ 0 \end{array}}
 \end{array}$$

FIG. 10.1 – Inversion d'une matrice de taille quelconque par la réduction de Gauss

## Chapitre 11

# Résolution des systèmes d'équations avec le *pivot de Gauss*

Soit  $B_{m \times m}$  une matrice carrée inversible traduisant un système de  $m$  équations à  $m$  inconnues.

Dans notre cas,  $B$  traduit l'équation  $Bx_{k+1}^n = Ax_1^k$ . Notons  $x = x_{k+1}^n$  et  $y = Ax_1^k$ .

L'élimination de Gauss-Jordan consiste à transformer le système  $[B \ y]$  en un système équivalent dont le bloc gauche est l'identité, c'est-à-dire qu'il faut modifier la matrice  $[B \ y]$  pour qu'elle devienne de la forme  $[I \ x]$  en utilisant les propriétés de l'algorithme.

La complexité algorithmique asymptotique du pivot de Gauss est  $O(n^3)$  (*notations de Landau*), donc le nombre d'instructions nécessaires est proportionnel à  $n^3$  si la matrice est de type  $(n \times n)$ . Cet algorithme peut être utilisé sur un ordinateur pour des systèmes avec des milliers d'inconnues et d'équations. Le pivot de Gauss est une bonne méthode pour les systèmes d'équations sur un champ où les calculs sont par nature exacts comme les *corps finis* (de cardinal fini), or nous travaillons dans le domaine binaire.

*Remarque* : L'algorithme de Strassen, qui est en  $O(n^{2,807})$  a une meilleure complexité algorithmique asymptotique. Sinon, d'autres techniques de résolution existent, notamment la *décomposition LU*, qui est une forme particulière d'élimination de Gauss-Jordan. On transforme la matrice  $B$  en une matrice triangulaire supérieure  $U$  en éliminant les éléments sous la diagonale. Les éliminations se font colonne après colonne, en commençant par la gauche, en multipliant  $B$  par la gauche avec une matrice triangulaire inférieure.

# Chapitre 12

## Le codage de Huffman

Dans cette présentation, on désire compresser un texte composé de caractères Ascii (code compris entre 0 et 255). Un caractère est classiquement représenté par une chaîne de 8 bits. L'idée de Huffman est de coder les caractères par des chaînes de bits de longueurs variables : les caractères fréquents seront représentés par des chaînes courtes, les caractères rares par des chaînes plus longues. Afin que le codage soit correct, la propriété suivante doit être vérifiée : le code d'un caractère ne doit pas être le préfixe d'un autre code (sinon la reconnaissance serait impossible).

Une façon simple de représenter un codage qui vérifie cette propriété est d'utiliser un arbre binaire tel que :

- les feuilles sont étiquetées par les caractères codés ;
- l'arête d'un fils gauche est étiquetée 0, l'arête d'un fils droit, 1 ;
- le code d'un caractère (une feuille) est le chemin depuis la racine de l'arbre.

Par exemple, pour le codage de  $a$  par 000,  $b$  par 11,  $c$  par 01,  $d$  par 001 et  $e$  par 10, on obtient l'arbre de la figure 12.1.

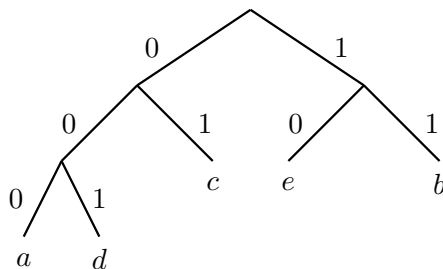


FIG. 12.1 – Arbre de l'algorithme de Huffman

On suppose que pour chaque caractère  $c \in \mathbf{C}$ , on connaît sa probabilité d'apparition  $P(c)$ . Si on note  $l(c)$  la longueur (nombre de bits) du codage du caractère  $c$ , le problème est de trouver un codage tel que  $\sum_{c \in \mathbf{C}} P(c)l(c)$  soit minimal. L'algorithme de Huffman est une technique pour trouver un tel codage optimal.

### 12.1 Algorithme de Huffman : Codage

L'algorithme travaille sur une forêt d'arbres, c'est-à-dire un ensemble d'arbres. Chaque arbre, dont les feuilles sont des caractères, possède un poids. Un arbre réduit à une feuille possède comme poids la probabilité du caractère  $c$  de cette feuille ; un tel arbre est noté  $f[c, P(c)]$ . Un arbre non réduit à une feuille a un poids égal à la somme des poids de ses deux fils ; il est noté  $n[g, d, p]$  où  $g$  et  $d$  sont les fils gauche et droit du nœud et  $p$  son poids.

L'algorithme est le suivant :

- $E := \{f[c, P(c)]/c2C\}$  ;
- Tant que  $E$  n'est pas un singleton
  - $n_1 :=$  nœud de plus faible poids ;
  - $n_2 :=$  nœud de deuxième plus faible poids ;
  - $E := E \setminus \{n_1, n_2\} \{n[n_1, n_2, poids(n_1) + poids(n_2)]\}$  ;



# Chapitre 13

## Le codage turbo

La présentation des codes turbo est ici très succincte, étant donné que qu'ils sont basés sur deux codeurs convolutifs, dont nous avons donné une complète présentation dans le chapitre 3. Pour cet exposé, nous nous basons sur le tutoriel [1] et l'article plus détaillé et plus récent [30] et les références qui y sont citées.

### 13.1 Codage d'un code turbo

L'idée principale du codeur turbo est d'utiliser deux codes convolutifs en parallèle avec un entrelaceur entre les deux. Si les codes convolutifs peuvent encoder un flux continu d'information, le codeur turbo ne peut encoder l'information que par paquets de longueur finie égale à celle de l'entrelaceur. Ainsi, après chaque *block* d'information, le codeur est forcé à un état connu. Alors, les bits de terminaison sont placés à la suite des bits d'information avant d'effectuer le décodage, voir figure 13.1.

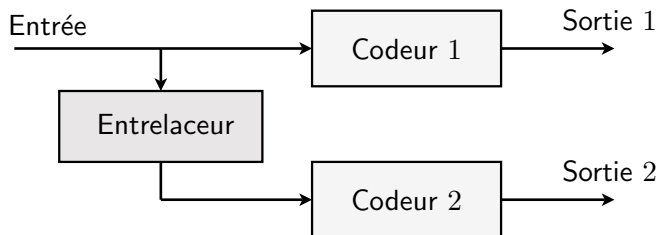


FIG. 13.1 – Codeur turbo.

On peut interpréter le codeur turbo comme un grand code en block : sa performance dépend de la distribution du poids, c'est à dire non pas seulement de la distance minimale, mais aussi du nombre de mots de faible poids. De cette manière, le rôle de l'entrelaceur est de transformer les motifs donnant des mots de poids faibles en sortie du premier codeur en des motifs donnant des mots de forts poids pour le second codeur. Le rôle de l'entrelaceur sera expliqué plus en détails dans la section 13.1.2.

#### 13.1.1 Choix des codeurs convolutifs

Par tradition, les codes convolutifs s'utilisent dans leur forme "alimenté par l'avant" (feed-forward), comme par exemple le codeur convolutif  $(5, 7) : (G_1, G_2) = (1 + D^2, 1 + D + D^2)$ . Cependant, pour ce genre de codes, un seul 1, c'est à dire la séquence  $\dots 0001 \dots$  sera codé en un mot de code correspondant aux vecteurs générateurs, et le poids de ce mot de code sera en général très faible et il est évident qu'un unique 1 se propagera à travers n'importe quel entrelaceur comme un seul 1. Donc si l'on utilise les codes convolutifs dans leur forme feed-forward, le code turbo résultant donnera une majorité de mots de codes de poids très faibles.

Pour éviter cela, les codes convolutifs doivent être utilisés sous leur forme récursive systématique, où l'on divise par l'un des vecteurs générateurs : l'exemple de tout à l'heure donne le code  $(1, G_2/G_1) = (1, \frac{1+D+D^2}{1+D^2})$ , dont le diagramme est donné sur la figure 13.2. Cette opération ne change pas l'ensemble des séquences codées mais la correspondance entre les séquences d'entrée et les séquences en sortie. Le codage est différent mais les propriétés de distances restent les mêmes.

L'effet important de cette modification est qu'une séquence comportant un seul 1 sera codé en un mot de poids semi-infini, donc il y a des possibilités de trouver un entrelaceur avec lequel les motifs d'information donnant des mots de poids faibles à la sortie du premier codeur sont transformés en motifs donnant des mots de poids élevés à la sortie du deuxième codeur.

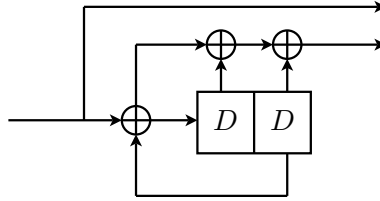


FIG. 13.2 – Codeur turbo récursif systématique.

Il est à remarquer que le fait que le code soit systématique n'est qu'une coïncidence, mais cela se révèle très utile dans plusieurs cas. Cependant, si l'on transmet les parties systématiques des deux codeurs, cela ne serait qu'une répétition, or nous savons qu'il est possible de bien meilleurs codeurs que des codeurs à répétitions. En fait, la partie d'information ne doit être transmise que d'un côté des deux codeurs, ce qui fait que si l'on utilise des codes convolutifs de taux  $\frac{1}{2}$  chacun, le taux final du codeur turbo sera de  $\frac{1}{3}$ . S'il y a besoin de davantage de redondance, on doit choisir des codes convolutifs de plus faible taux de codage.

### 13.1.2 Choix de l'entrelaceur

#### Description mathématique des entrelaceurs

Soit une séquence de longueur  $N$  :  $X = (x_1, \dots, x_N)$ . L'entrelaceur permute  $X$  en une autre séquence de même longueur,  $\tilde{X} = (x_{b(1)}, \dots, x_{b(N)})$ ,  $\tilde{X}$  possède donc les mêmes éléments que  $X$  mais dans un ordre différent.

Soient l'index des éléments de  $X$  et  $\tilde{X}$  respectivement  $I = \{1, \dots, i, \dots, N\}$  et  $I_B = \{b(1), \dots, b(i), \dots, b(N)\}$ . On note ici  $b(i)$  la position entrelacée du  $i$ -ème élément. Alors l'entrelaceur peut être défini par la fonction de correspondance bijective :  $B(I \rightarrow I_B) : i \rightarrow j = b(i); i, j \in I$ .

Inversement, on note  $b^{-1}(i)$  la position désentrelacée du  $i$ -ème élément. Le processus de désentrelacement de l'index peut s'écrire comme :  $B^{-1}(I_B \rightarrow I) : i = b^{-1}(j) \rightarrow i, i, j \in I$ .

#### Catégories de l'entrelaceur

En se basant sur sa structure, on peut classifier l'entrelaceur dans la catégorie "aléatoire" ou "déterministe".

Un entrelaceur possédant une fonction de correspondance générée aléatoirement est appelé "entrelaceur général pseudo aléatoire". Un entrelaceur ayant des propriétés pseudo aléatoires est efficace s'il a une grande taille de bloc. La performance d'un code turbo est dominée par les mots de code de faibles poids, dont la plupart sont générés par des séquences d'information auto-terminés<sup>1</sup> de poids 2. Un entrelaceur général pseudo aléatoire ne peut pas casser ces séquences efficacement, car les poids moyens des mots de code ayant des séquences d'entrée auto-terminés de poids 2 augmente linéairement avec la distance entre leurs deux 1.

Ainsi, le premier critère pour concevoir un entrelaceur est d'augmenter la distance minimale  $S$  entre ces deux 1 dans l'information de poids 2 auto-terminé entrelacé. L'entrelaceur optimisé résultant est appelé "entrelaceur aléatoire d'ordre  $S$ " et peut atteindre de meilleures performances que les entrelaceurs pseudo aléatoires.

La plupart des critères pour concevoir des entrelaceurs sont basés sur l'entrelaceur aléatoire d'ordre  $S$ .

<sup>1</sup>Une séquence d'information d'entrée de taille  $N$  est dite "auto-terminée" si le codeur revient à l'état tout-zéro après le codage de  $N$  symboles d'information sans ajouter aucun bit de terminaison

Pour les entrelaceurs aléatoires, le motif entrelaceur doit être stocké dans des tables du côté du codeur et du décodeur. La complexité exponentielle suivant  $N$  de tels entrelaceurs devient donc très rapidement pénalisant, surtout au niveau de la mémoire de travail. L'implémentation matérielle des codes turbo est souvent mise à mal par cette contrainte, d'où la nécessité de disposer d'*entrelaceurs déterministes*. Ces derniers sont définis complètement par quelques paramètres pouvant être aisément stockés par le codeur et le décodeur pour générer en temps réel les entrelaceurs.

L'entrelaceur déterministe le plus simple à concevoir est un entrelaceur en bloc basé sur une matrice ; la séquence d'entrée est alors écrite dans une matrice suivant les lignes et lue suivant les colonnes. Si un tel entrelaceur est facile à implémenter, son efficacité peut fléchir devant certains motifs d'entrée de faibles poids comme des motifs carrés ou rectangulaires de poids 4, 6 ou 9. Une version améliorée de l'entrelaceur en bloc, appelé "entrelaceur hélicoïdal de comparaison" a été développé : l'information est écrite dans la matrice selon les lignes puis lue suivant les diagonales de la gauche vers la droite et de bas en haut.

Dernièrement, plusieurs entrelaceurs déterministes très structurés ont été trouvés, ils sont basés sur des décalages circulaires et des permutations polynomiales algébriques linéaires.

### 13.1.3 Représentation des codes turbo en tant que codes en blocs

En vertu de la présence d'un entrelaceur de taille finie, il est bien connu que les codes turbo sont une classe de codes en blocs ; cependant il est rare qu'on analyse un code turbo en tant que code en bloc. Pour effectuer explicitement la connexion entre le codage turbo et le codage LDPC, il est nécessaire de trouver les matrices de contrôle de parité des codes turbo.

La structure d'un codeur turbo est montrée sur la figure 13.1, où il est clair qu'un code turbo peut aussi s'appeler "code convolutif parallèle concaténé". Néanmoins, la description de cette concaténation parallèle ne mène pas immédiatement à une méthode pour trouver les matrices génératrice et de contrôle de parité de ce code.

Voici l'algorithme permettant de trouver la matrice génératrice et la matrice de contrôle de parité d'un code turbo arbitraire.

Il est bien connu qu'un code turbo entre dans la catégorie des codes en blocs à cause de la taille fixe de leur entrelaceur. Néanmoins, il est rare qu'on analyse un code turbo à partir de sa représentation en tant que code en bloc, mais pour ce qui nous concerne, c'est-à-dire le codage distribué symétrique de sources corrélées, cela peut s'avérer très utile. Il nous reste alors à expliciter la matrice d'un code turbo.

#### Matrice génératrice d'un code turbo

Les deux représentations (a) et (b) du code turbo de la figure 13.3 sont équivalentes.

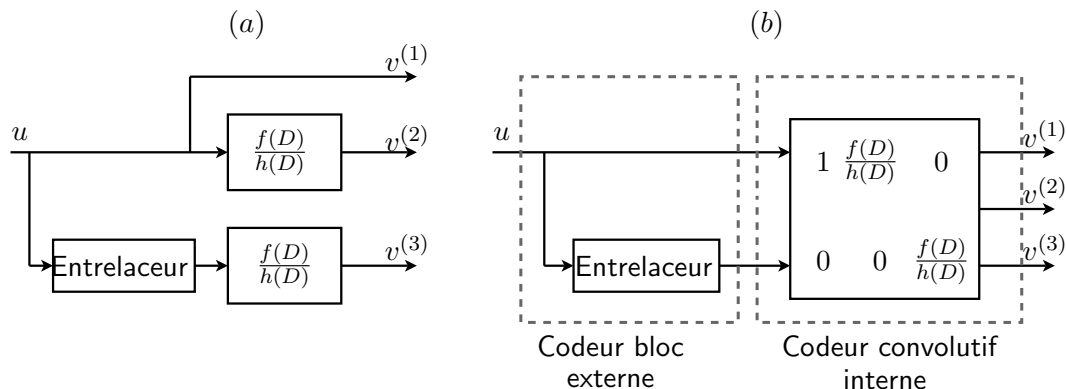


FIG. 13.3 – Représentations du codeur turbo

La (b) est une concaténation en série d'un codeur externe en bloc de rapport  $\frac{1}{2}$  et d'un codeur convolutif interne de rapport  $\frac{2}{3}$ . Le codeur externe encode la séquence d'entrée du codeur turbo en deux séquences : l'une identique et l'autre permutée, respectivement notées  $w^{(1)}$  et  $w^{(2)}$ .

Ces deux séquences sont à nouveau encodées par le codeur convolutif interne de rapport  $\frac{2}{3}$  en trois séquences qui représentent la séquence complète du mot de code du code turbo.

La matrice génératrice polynomiale du codeur convolutif interne est également montrée dans la figure ; remarquons que c'est un codeur à contre-réaction. Pour simplifier l'analyse, le polynôme de contre-réaction peut se factoriser hors du codeur et intégré dans le codeur externe : la structure du codeur turbo résultant est représentée sur la figure 13.4. Si l'on parvient à trouver les matrices génératrices des deux codeurs interne et externe du codeur turbo, la matrice générale du codeur turbo peut s'écrire comme :  $G_{CT} = G_{externe} \times G_{interne}$ .

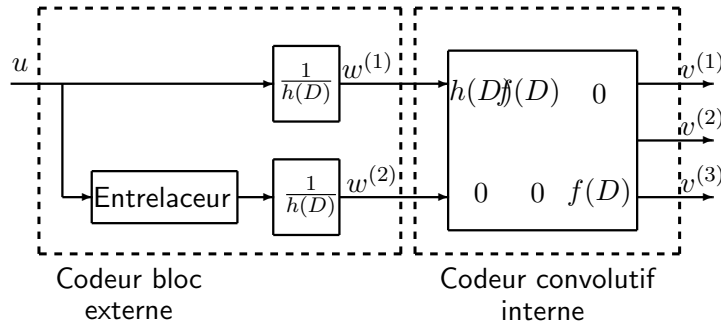


FIG. 13.4 – Structure du codeur turbo résultant

Le codeur interne est juste un codeur convolutif standard, et il est facile d'en obtenir la matrice génératrice.

Pour trouver la matrice génératrice d'un code turbo, prenons un exemple. L'algorithme peut ensuite se généraliser aisément pour tout code turbo.

**Exemple :** Soit le turbo code composé de deux codes convolutifs générés par  $G(D) = \left[1, \frac{1+D^2}{1+D+D^2}\right]$  et un entrelaceur de longueur 9 dont la rangée d'entrelaceur est donnée par la permutation :

$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 6 & 1 & 4 & 7 & 8 & 5 & 3 & 0 & 2 \end{pmatrix} \quad (13.1)$$

Remarquons que, comme chacun des deux codeurs se terminent par 2 bits de fin, 4 bits supplémentaires seront ajoutés à la séquence d'information pour former la séquence d'entrée du codeur turbo. Ainsi, le rapport du code total est  $\frac{9}{3 \cdot (9+2) + 2} = \frac{9}{35}$  et sa matrice génératrice est de dimension  $9 \times 35$  avec des entrées binaires. Puisque  $f(D) = 1 + D^2$  et que  $h(D) = 1 + D + D^2$ , la matrice génératrice du code interne est :

$$G_{interne}(D) = \begin{bmatrix} 1 + D + D^2 & 1 + D^2 & 0 \\ 0 & 0 & 1 + D^2 \end{bmatrix}$$

La recherche de la matrice du codeur externe n'est pas aussi évidente que celle de l'interne : puisque le codeur externe est un code binaire linéaire (18, 9), c'est un sous-espace de dimension 9 de l'espace des vecteurs binaires de dimension 18, donc le codage d'un ensemble de bases générant l'espace des séquences binaires de dimension 9 donnera une matrice génératrice du codeur externe.

On va donc générer cette matrice grâce aux vecteurs de longueur 9 comprenant chacun un unique 1 :  $e_i, i = 0, \dots, 8$ . Cette base nous permettra de manipuler l'entrelaceur du code turbo facilement, vu que la permutation de chaque élément donnera toujours un élément de l'ensemble.

À cause du polynôme de contre-réaction  $h(D) = 1 + D + D^2$ , 2 bits de fin doivent être rajoutés à chaque séquence  $e_i$  pour faire revenir le contenu des registres à décalages à l'état tout-zéro. Donc on a besoin d'une séquence de longueur 11 pour générer une séquence de sortie. Trouver ces deux bits de fin revient à trouver un multiple de  $h(D)$  possédant seulement un bit à 1 dans ses 9 premiers termes.

Soit  $t_i(D)$  la séquence finissant  $e_i$  et  $w_i(D)$  la séquence de sortie correspondant au registre à décalage linéaire de contre-réaction. Comme tous les  $e_i$  ne sont que des permutations de  $e_0$ , on peut ne considérer que  $e_0$ . La séquence de sortie du registre à décalage linéaire de contre-réaction ayant  $e_0$  pour entrée est :  $w_0(D) = \frac{t_0(D)}{h(D)}$ . Le degré de  $t_0(D)$  est au plus 10 et le degré de  $h(D)$  est 2, donc le degré de  $w_0(D)$  est au plus 8. Ainsi les deux derniers bits de  $w_0(D)$  sont toujours à 0 et c'est vrai quel que soit  $w_i(D)$ .

Maintenant que  $w_0(D)$  a été trouvé,  $w_i(D)$  est obtenu en décalant  $w_0(D)$  vers la droite et en tronquant le résultat à la taille 9.

Pour l'entrée  $u = w_0(D)$ , la sortie  $w^{(1)}$  est la suivante :  $w^{(1)} = [1\ 1\ 0\ 1\ 1\ 0\ 1\ 1\ 0]$ , d'où l'expression de la matrice avant et après application de la permutation (13.1) :

$$\text{Avant : } \begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \text{ après : } \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \end{bmatrix} \quad (13.2)$$

La matrice donnant le codeur externe est la concaténation de ces deux matrices (13.2) :

$$G_{\text{externe}} = \begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \end{bmatrix}$$

La matrice génératrice du code turbo considéré est alors  $G_{CT} = G_{\text{externe}} \times G_{\text{interne}}$ , en ayant pris soin de réécrire la matrice  $G_{\text{interne}}(D)$  sous forme binaire. Pour finir, deux colonnes supplémentaires doivent être rajoutées à  $G_{CT}$  pour prendre en compte les deux bits de fin pour le second registre à décalage :  $G_{\text{fin}}^1$  et  $G_{\text{fin}}^2$ .  $G_{\text{fin}}^1$  est déjà contenu dans  $G_{CT}$  et  $G_{\text{fin}}^2$  est une permutation des lignes de  $G_{\text{fin}}^1$  compte tenu de l'entrelaceur.

### Matrice de contrôle de parité d'un code turbo

Si l'on dispose de la matrice génératrice du code turbo,  $H$  peut se trouver très facilement, car comme tout code en bloc linéaire :  $G.H = 0$ . De plus, le caractère systématique de  $G$  nous simplifie la tâche. Pour plus de commodité, il est préférable d'exprimer la matrice de contrôle de parité du code turbo en fonction de ses éléments : l'entrelaceur et les deux codes convolutifs.

Soit un code turbo ayant un entrelaceur de longueur  $N$  et deux codes convolutifs identiques de rapport  $\frac{1}{2}$ , de mémoire<sup>2</sup>  $\nu$  générés par  $G(D) = \left[1, \frac{f(D)}{h(D)}\right]$ , ce qui fait que la longueur de la séquence en entrée du codeur turbo est de longueur  $N + 2\nu$  et la longueur de la séquence d'entrée de chaque code convolutif est  $L = N + \nu$  (voir la figure 13.4).

Soit  $w_0|_i$  la séquence obtenue en décalant  $w_0$  de  $i, i = 0, \dots, N - 1$  fois vers la droite et tronquée à la taille  $N$ . Alors la matrice génératrice du code externe des codes turbo a l'expression suivante :

$$\left[ \begin{array}{c|c} w_0|_{\vec{0}} & w_0|_{b(\vec{0})} \\ \dots & \dots \\ w_0|_{N-\vec{1}} & w_0|_{b(N-\vec{1})} \end{array} \right]$$

<sup>2</sup>Les deux codes convolutifs sont terminés par  $\nu$  bits de fin.

dont l'expression compacte est

$$G_{externe}(D) = \left[ w_0|_{\vec{i}} \mid w_0|_{b(\vec{i})} \right]_{i=0}^{N-1}$$

La matrice génératrice du code interne est :

$$G_{interne}(D) \begin{bmatrix} h(D) & f(D) & 0 \\ 0 & 0 & f(D) \end{bmatrix}$$

Quand le code turbo est systématique, la première sous-séquence  $v^{(1)}$  du mot de code est toujours identique à la séquence d'entrée de l'encodeur, la matrice de contrôle de parité est déterminée essentiellement par la partie non systématique de la séquence du mot de code. Ainsi, la première colonne de  $G_{interne}$  peut être effacée pour former la matrice :

$$G_{interne}^H(D) \begin{bmatrix} f(D) & 0 \\ 0 & f(D) \end{bmatrix}$$

Sa forme binaire  $G_{interne}^H$  est dérivée facilement.

Au final, la matrice de contrôle de parité du code turbo est :

$$H = [I_{2.L}, P], \quad P = \left( \left[ w_0|_{\vec{i}} \mid w_0|_{b(\vec{i})} \right]_{i=0}^{N-1} \times G_{interne}^H, G_{fin}^2 \right)^T$$

Une fois qu'on dispose de ces deux matrices, le codage peut s'effectuer très simplement par la multiplication de la séquence d'information et de la matrice de contrôle de parité, comme dans le cas des codes LDPC

## 13.2 Décodages d'un code turbo

### 13.2.1 Décodage classique : MAP

Dans le cas du décodage d'un code turbo, on dispose de deux séquences codées. L'algorithme commence par décoder l'une d'entre elles pour avoir une première estimation de l'information codée. Cette estimation sera alors utilisée comme information a priori dans le décodage de la deuxième séquence codée. Cela nécessite que le décodeur soit capable d'utiliser une entrée soft et de produire une sortie soft.

Le décodeur standard d'un code turbo est basé sur un décodage à probabilité a posteriori (aussi appelé "*algorithme de décodage à maximum a posteriori (MAP)*"). L'algorithme classique MAP d'un code turbo peut se trouver dans le tutoriel [1].

### 13.2.2 Décodage en tant que code en bloc : Propagation de croyance

Cet algorithme de décodage est pertinent lorsque la matrice de contrôle de parité du code turbo est entièrement trouvée. Dans ce cas, l'algorithme revient à le considérer comme un code LDPC classique et à travailler sur le graphe de Tanner du code.

Comme on peut s'y attendre intuitivement, cet algorithme est sous-optimale : le graphe de Tanner obtenu n'est pas optimisé pour un codage LDPC.

# Bibliographie

- [1] Jakob Dahl Andersen. *A turbo tutorial*. COM Center, Technical university of Denmark.
- [2] F. Bassi, C. Guillemot, M. Kieffer, A. Roumy, and V. Toto Zarasoa. State of art techniques in distributed source coding. Technical Report 0, ANR ESSOR, 2007.
- [3] T. M. Cover and J. M. Thomas. *Elements of Information Theory*. Wiley, New-York, 1991.
- [4] C. Di, D. Proietti, E. Telatar, T. J. Richardson, and R. L. Urbanke. Finite-length analysis of low-density parity-check codes on the binary erasure channel. *IEEE Transactions on information theory*, 48(6), June 2002.
- [5] P. Elias. Predictive coding – part i. *IEEE Transactions on Information Theory*, 1(1) :16–24, March 1955.
- [6] P. Elias. Predictive coding – part ii. *IEEE Transactions on Information Theory*, 1(1) :24–33, March 1955.
- [7] R. Gallager. Low density parity check codes. *MA : MIT Press*, 0(0), 1963. Cambridge.
- [8] Bernd Girod, Anne Aaron, S. Rane, and D. Rebollo-Monedero. Distributed video coding. *Proc. of the IEEE*, 1(71) :93, January 2005.
- [9] J. Ha and S. W. McLaughlin. Optimal puncturing of irregular low-density parity-check codes. *IEEE Transactions on Information Theory*, 0(0), 2003.
- [10] S-N. Hong, H-G. Joo, and D-J. Shin. Optimal puncturing of block-type ldpc codes and their fast convergence decoding. *IEEE Transactions on Information Theory*, 0(0), 2006.
- [11] D.A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the I.R.E.*, 0(0), September 1952.
- [12] F. R. Kschichang, B. J. Frey, and H. A. Loe liger. Factor graphs and the sum-product algorithm. *IEEE Trans. on Information theory*, 47(2) :498–519, February 2001.
- [13] B. M. J. Leiner. *LDPC codes : a brief tutorial*. –, April 2005.
- [14] Jing Li and H. Alqamzi. An optimal distributed and adaptive source coding strategy using rate-compatible punctured convolutional codes. In *Acoustics, Speech, and Signal Processing, 2005. Proceedings. (ICASSP '05). IEEE International Conference on*, volume 3, pages iii/685–iii/688Vol., 18–23 March 2005.
- [15] A.D. Liveris, Zixiang Xiong, and C.N. Georghiades. Compression of binary sources with side information at the decoder using ldpc codes. *Communications Letters, IEEE*, 6(10) :440—442, Oct. 2002.
- [16] D. J. C. MacKay. Good error-correcting codes based on very sparse matrices. *IEEE transactions on information theory*, 45(2), March 1999.
- [17] D. J. C. MacKay. Punctured and irregular high-rate gallger codes. *IEEE Transactions on Information Theory*, 0(0), September 2000. Draft 2.1.
- [18] H. Pishro-Nik and F. Kekri. Results on punctured low-density parity-check codes and improved iterative decoding techniques. *IEEE Transactions on information theory*, 53(2), February 2007.
- [19] O. Pothier. *Codage de canal et turbo-codes*, 2000.
- [20] Sandeep S. Pradhan, J. Kusuma, and Kannan Ramchandran. Distributed compression in a dense micro-sensor network. *IEEE Signal Processing Magazine*, 19(0) :51–60, March 2002.

- [21] A. Roumy, K. Lajnef, and C. Guillemot. Rate-adaptive turbo-syndrome scheme for slepian-wolf coding. *IEEE Transactions on Information Theory*, 0(0), July 2007.
- [22] Aline Roumy and David Declercq. Characterization and optimization of ldpc codes for the 2-user multiple access channel. *EURASIP Journal on Wireless Communications and Networking*, 2007(0) :Article ID 74890, 10, May 2007.
- [23] Mina Sartipi and Faramarz Fekri. Distributed source coding in wireless sensor networks using ldpc coding : The entire slepian-wolf rate region. *IEEE Wireless Communications and Networking Conference*, 4(0) :1939–1944, March 2005.
- [24] David Slepian and Jack Keil Wolf. Noiseless coding of correlated information sources. *IEEE Transactions on Information Theory*, 19(4) :471–480, July 1973.
- [25] Peiyu Tan and Jing Li. A practical and optimal symmetric slepian-wolf compression strategy using syndrome formers and inverse syndrome formers. *Proceeding of 43rd Annual Allerton Conference on Communication, Control and Computing*, 0(0), September 2005.
- [26] R. M. Tanner. A recursive approach to low complexity codes. *IEEE Trans. Inform. Theory*, 27(0), September 1981.
- [27] David Varodayan, Anne Aaron, and Bernd Girod. Rate-adaptive codes for distributed source coding. *EURASIP Signal Processing*, 86(11) :3123–3130, November 2006.
- [28] B. N. Vellambi and F. Fekri. Results on the improved decoding algorithm for low-density parity-check codes over the binary erasure channel. *IEEE Transactions on information theory*, 53(4), April 2007.
- [29] Sergio Verdu. Fifty years of shannon theory. *IEEE Transactions on Information Theory*, 44(6) :2057 – 2078, October 1988.
- [30] B. Vucetic, Y. Li, L. C. Perez, and F. Jiang. Recent advances in turbo code design and theory. *Proceedings of the IEEE*, 95(6) :1323 – 1344, June 2007.
- [31] J. K. Wolf. Efficient maximum likelihood decoding of linear block codes using a trellis. *IEEE Transactions on Information Theory*, 24(0) :76–80, January 1978.
- [32] A. Wyner. Recent results in the shannon theory. *IEEE Trans. Information Theory*, 20(0) :2–10, 1974.
- [33] Zixiang Xiong, Angelos D. Liveris, and S. Cheng. Distributed source coding for sensor networks. *Signal Processing Magazine, IEEE*, 21(5) :80—94, Sept. 2004.
- [34] R. Zamir, S. Shamai, and U. Erez. Nested linear/lattice codes for structured multiterminal binning. *IEEE Trans. Information Theory*, 48 :1250–1276, 2002.
- [35] W. Zhong, H. Chai, and J. Garcia-Frias. Ldgm codes for transmission of correlated senders over mac. *IEEE Trans. Information Theory*, 0(0), 2004. University of Delaware.