Codes joints source-canal pour transmission robuste sur canaux mobiles

Simon Malinowski

▶ **To cite this version:**

Simon Malinowski. Codes joints source-canal pour transmission robuste sur canaux mobiles. Traitement du signal et de l'image [eess.SP]. Université de Rennes 1, 2008. Français. NNT : . tel-01171117

HAL Id: tel-01171117

https://theses.hal.science/tel-01171117v1

Submitted on 2 Jul 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

N° d'ordre: 3833

THÈSE

présentée

devant l'Université de Rennes 1

pour obtenir

le grade de : DOCTEUR DE L'UNIVERSITÉ DE RENNES 1
Mention TRAITEMENT DU SIGNAL

par

Simon MALINOWSKI

Équipe d'accueil : Temics - IRISA
École Doctorale : Matisse
Composante universitaire : UNIVERSITÉ DE RENNES 1

Titre de la thèse :

Codes joints source-canal pour transmission robuste sur canaux mobiles

Soutenue le 02 Décembre 2008 devant la commission d'examen

M. :	Jean Jacques	FUCHS	Président
MM. :	Pierre	DUHAMEL	Rapporteurs
	Luc	VANDENDORPE	
MM. :	Hervé	JÉGOU	Examinateurs
	Ramesh	PYNDIAH	
	Pierre	SIOHAN	
	Christine	GUILLEMOT	Directeur de thèse

à mamie,

Remerciements

Je viens de passer un petit peu plus de trois années à Rennes, en Bretagne à l'élaboration de cette thèse. Cette longue période passée dans cette belle région me donne des visions de grand large. Ô Bretagne, terre de marins, tu m'as converti. J'ai envie de comparer (toutes proportions gardées) l'aventure du doctorat à la traversée de l'Atlantique en solitaire à la voile. Outre l'accueil chaleureux que j'ai reçu de cette Bretagne, il ne faut pas omettre de souligner les moults signes précurseurs n'ayant cessé de lier mon destin à celui de ces marins ayant choisi d'affronter l'univers sans pitié de la grande Bleue (je fais ici référence notamment à mon parcours personnel ainsi qu'à un homonyme célèbre dont beaucoup de bretons m'ont appris l'existence). Peut-être aussi, vois-je dans cet univers trop de choses qui me sont inaccessibles. La cinétose la plus répandue ne m'a pas épargné, l'insatiété caractéristique de l'espèce humaine doit être pour beaucoup dans le parallèle que je suis en train de créer. Pendant ces quelques lignes, je me mets donc dans la peau d'un skipper venant de terminer la Route du Rhum. Des images reviennent, tout s'emballe. Cette traversée en solitaire ne le fût pas, la suite de ce récit va le prouver.

Saint-Malo, Mars 2005

Avant de songer à traverser l'Atlantique en solitaire, il est indispensable de trouver un port d'attache, un point de départ sans lequel il n'y a pas d'arrivée. Là, fut l'importance de Christine Guillemot et Hervé Jégou. Merci infiniment, Christine et Hervé, d'avoir tracé la ligne de départ dans le port de Saint-Malo. Mais votre contribution ne peut se réduire à ce tracé. Comme tout marin respectable, j'ai été victime d'avaries pendant cette traversée. J'ai pu me tourner vers vous, sans avoir eu besoin de refaire le chemin inverse jusqu'au port de Saint-Malo, où tout a commencé. Vous n'étiez jamais très loin. Merci.

Pointe-à-Pitre, Décembre 2008

Pour que la route du Rhum soit ce qu'elle est, il faut (non pas du Rhum, enfin pas que ça en tout cas) avoir la possibilité de franchir une ligne d'arrivée. Messieurs Duhamel, Fuchs, Pyndiah, Siohan et Vandendorpe ont tracé cette ligne en Salle Métivier (Pointe-à-Pitre pour l'occasion) et ont vérifié que mon franchissement fût légal. Je remercie donc Jean-Jacques Fuchs, Professeur à l'INRIA Rennes, de m'avoir fait l'honneur de présider ce jury. Je remercie également Luc Vandendorpe, professeur à l'Université Catholique de Louvain, et Pierre Duhamel, directeur de Recherche CNRS au laboratoire L2S, d'avoir accepté la charge de rapporteur, ainsi que Ramesh Pyndiah, professeur

à Télécom Bretagne à Brest, et Pierre Siohan, Chercheur à France Télécom, d'avoir accepté d'être examinateurs de cette thèse. Un grand merci à vous.

L'océan Atlantique

Si les points de départ et d'arrivée sont des éléments indispensables à la traversée de l'Atlantique à la voile, un tel accomplissement ne saurait être possible sans cette étendue bleutée aux reflets azurs, parfois calme, parfois agitée, qu'est l'océan Atlantique. Ma famille a été mon océan Atlantique pendant ces trois années.
Il m'est impossible de ne pas citer en premier lieu mes parents, Annie et François-Xavier. Paradoxalement, je ne soulignerai pas ici votre aide pendant cette traversée. En effet, elle est infime par rapport à tout l'amour que vous m'avez apporté avant cette épreuve et qui m'a permis - et c'est là le plus important - de m'être présenté au départ de l'épreuve paré de toutes les armes nécessaires à la réalisation de ce grand voyage. Jean-Christophe et Benjamin, mes frères, avez également, est-ce besoin de le préciser, une importance non négligeable dans cet accomplissement. Merci Benjamin d'être venu jusqu'à Pointe-à-Pitre le 2 Décembre.
Mais le cercle familial ne s'arrête pas là. Je pense à vous, grands-parents, oncles, tantes, cousins et cousines, présents autant par le nombre que par les faits (c'est à votre avantage !). J'aime à me souvenir d'instants pendant lesquels vous avez participé, chacun à votre façon, à ce trajet tumultueux. Philippe, tu imprimas le poster qui, quelques jours plus tard sortit du lot à Athènes. Marie-Noelle, tu finis le travail de nettoyage des balles de golf afin de me permettre de me concentrer sur le passage difficile des Açores. Jack, je me souviens de nos quelques parties de golf où tu t'es toujours battu pour que je ne te distance pas trop. Tu as plus que réussi d'ailleurs... ! Tu m'as également accompagné jusqu'à la ligne d'arrivée le 2 Décembre, merci beaucoup.
Clément, nous avons réussi à ne pas nous laisser distancer du monde musical pendant ces dernières années. Je fais ici référence aux nombreux concerts auxquels nous avons assisté (parfois plusieurs fois d'ailleurs.. !), ainsi qu'aux quelques "blind-tests" que nous avons organisés (parfois, ce ne fût pas une bonne idée mais bon !). Je n'oublie pas les moments conviviaux en compagnie de Pierre le tavernier, Roger le c..., Murielle la g... etc ! !
Gauthier, je nous revois remporter haut la main la Coupe du Directeur (merci Guillaume), ainsi qu'arpenter fièrement les fairways (enfin surtout les forêts...) de Chantilly !
Baboucha, je souhaite également te remercier de toujours nous apporter ton sourire, et pour les moments passés à Querrieu entre cousins. Merci Baboucha.
Merci à vous tous, vous m'êtes indispensables (même ceux que je n'ai pas cité, faute de temps, de place...).
Je voudrais dédier cet acomplissement à une personne qui a énormément compté pour moi et qui nous a quitté trop tôt. Mamie, tu m'as donné la force d'aller jusqu'au bout. Tu as été un exemple pour moi. Je ne t'oublierai jamais, tu me manques comme je

respire. Merci pour tout.
Je voudrais remercier également mon grand-père, qui m'a transmis son esprit sportif, de battant qui m'a servi tous les jours à aller de l'avant, pour me rapprocher jour après jour, et à grand pas, des senteurs et rythmes créoles caractéristiques de Pointe-à-Pitre. Mamie,Papi vous serez toujours là.

Kriter VI

En 1978, Michel Malinowski passa à moins de deux minutes de la victoire sur la route du Rhum à bord de Kriter V. Trente ans après, Kriter VI m'emmena jusqu'à Pointe-à-Pitre. Kriter VI, un voilier nouvelle génération construit très minutieusement par une équipe d'armateurs professionnels, issue de la collaboration de plusieurs "équipounettes" localisées en différents endroits.
L'équipounette en chef siège à Amiens, dirigée par la famille Devisme. Les réunions de travail eurent souvent lieu autour d'un barbecue (de janvier à décembre), ou d'un repas composé uniquement de produits naturels ! Merci Pat, Stéph, Clément, Manon ma filleule préférée, et Hugo pour tous ces bons moments.
Ils ont été épaulés par des personnes de métiers issues de la grande famille de l'AAC. Je pense notamment à Stéphane et Laure. Stéphane, membre ô combien important de la grande équipe 4 cuvée 2002, finaliste régionale. Cette équipe avait la chance de pouvoir compter sur un des plus grands joueurs de double de notre génération. Non, pas Stéphane (il "bronze" pendant les doubles...). Olivier j'espère que tu as osé te reconnaître. Merci également à tous les autres membres de cette belle famille, notamment J-Firmin, Patrick Do., Georgette évidemment, et tous ceux que j'oublie ! C'est toujours un plaisir de vous retrouver.
Je ne peux oublier un sous-armateur en chef tel qu'Antoine. Nous avons sillonné la plupart des ports bretons, ainsi que du littoral atlantique afin d'emmagasiner le plus d'expérience possible. Pléneuf et son golf légendaire, Saint-Malo forcément, Dinard et son tennis-hôtel, Argenton et sa famille Jégou !, Lorient et son festival inter-celtique, Guidel et son aimable directrice de camping, La Roche-Bernard et son voilier (préparation idoine), La Baule son 30/1 pas manchot et sa Roucousse, Les Sables d'Olonne (en vue de la suite de ma carrière) et sa Doucousse, puis Royan et sa Jougoune. Nous avons même prospecté dans les Alpes, pour essayer d'importer les techniques de glisses alpines vers celles marines, avec pas mal de succès n'ayons pas peur des mots. Nat et Farid témoigneront. Un tel investissement ne pouvait qu'être profitable. Merci Antoine et BN !
La seconde équipounette se situe à Rennes. Les principaux armateurs ont été d'une efficacité hors norme. Ils ont surtout oeuvré pour améliorer les conditions de vie sur le navire surtout par période de vent calme. Les grands axes de travail ont été

1. le sport. De la nécessité de savoir manier le club de golf de tout temps (vent, neige, pluie, nuit). Oui, sur un bateau, on ne contrôle pas toujours les éléments.

De la persévérance pour gagner un trophée footballistique.

2. la musique. Petit à petit, de la guitare en solitaire au groupe de rock en tournée mondiale. Le trio de choc Alex, Romain et moi a fait beaucoup d'émules. Les fans se sont multipliés (bien que parfois divisés aussi) allant même jusqu'à la voisine du dessous... Bravo à nous.

Merci Alex et Romain T. pour ces précieux moments. Alex aura également contribué à la dépendance à la plus grande série télévisuelle de tous les temps : Plus belle la vie. Lorsque la mer est plate et que le pilote automatique est en route, il n'y a rien de tel qu'un épisode de cette série. pour naviguer vers de nouveaux horizons l'esprit rempli d'images entraînantes. Merci Léo, Roland, Luna et tous les autres. Vous m'avez surpris tous les jours... Je ne peux oublier de remercier et de rendre un hommage sincère à Erik Gerets, ce grand homme. Ne change pas et reste fidèle, Erik !
La troisième équipounette est mobile, pour une meilleure couverture de l'espace vital. Roro : Rennes, Lille, La Flèche et Bachaud, what else ! Tu as couvert tous ces endroits. Il fallait travailler les automatismes du RCF sur tous types de terrain (pelouse avec but en bois, gravier avec buts en église, en pente sans vraiment de buts...). La délégation polonaise de l'équipage au sol a besoin d'entraînement. Bronislaw, Bronislaw et les frères Bronislawouvic ont tout donné, et c'est bien l'essentiel. Je n'oublie pas Baugé et sa chèvre, Lille et ses porte-feuilles, Rennes et ses cochons-tirelire, Bachaud et sa tente intérieure, le Patton et ses matchs le dimanche matin. Tout se mélange, j'ai tellement de souvenirs. Merci Romain, c'était su-per !

Le gréement

Tout cela ne serait rien sans le gréement du navire. J'ai beau disposer d'un port d'attache, une arrivée sous les cocotiers, un océan ravissant et un navire d'une solidité à toute épreuve, je ne pouvais pas être performant sans un bon gréement. J'ai pu l'obtenir en puisant quotidiennement (parfois quasi-quotidiennement tout de même) dans la richesse des échanges que j'ai eu avec mes nombreux collègues de travail. Il y a eu beaucoup de mouvement pendant ces trois années. Je crains donc d'en oublier quelques-uns. Je me lance, en commençant par ceux qui sont encore présents. Denis, fidèle depuis le début mais toujours apprenti jongleur avec bouteilles. Gaël, fidèle aussi, co-présentateur d'awards et grand réalisateur d'affiche de thèse formidable ! Joaquin mon dernier co-bureau, souvent d'une grande aide. Angélique, alias Angelina Jolie, aussi appelée miss Pruneaux d'Agen (sans mauvaises pensées !) et Gertrude, mais surtout Bouclinette ! Le Belge, que dire de plus, tout est dans le nom ! Merci à vous, et à tous les autres : Olivier, Caroline, Thomas, Mathieu, Ana, Mehmet, Simon, Laurent, Aline, Fuxun, Tiaray... J'ai failli oublier des membres d'autres équipes. Christophe, merci d'avoir essayé d'atteindre mon niveau guitaristique ca m'a boosté. Tu es encore loin cependant. Kevin, quel chanteur d'exception...

Un merci très spécial à Huguette, notre maman de l'équipe. Tu as toujours veillé à ce que tout se passe bien sans hésiter à me diriger dans le bon azimut lorsque je ne suivais plus le bon cap (toujours avec le tact et la délicatesse qui te caractérisent ! !). Merci beaucoup Huguette.
Place à ceux qui ne font plus partie de l'équipe mais que je n'ai pas oublié. Christel, bien sur ! On formait un beau trio avec Huguette... ! Radio Vipère sur le balcon ou dans le bureau d'Huguette ! David, annoncé comme mon plus grand challenger golfique. Annoncé seulement,n'a pas tenu longtemps ! Je rigole... Jayant, ancien co-bureau. Très bons souvenirs. Fabien, également ancien co-bureau. Nous avons eu quelques avis en commun. Et tous les autres que j'oublie. Merci à vous.

Et maintenant

Le Vendée-Globe... J'espère que vous me serez tous d'une aide aussi précieuse !

"Nous ne valons que par ce qui nous distingue des autres ; l'idiosyncrasie est notre maladie de valeur"

André Gide

"L'unique, la plus douce protection contre toutes les peurs c'est celle-là - un livre qui commence"

Alessandro Baricco

Table des matières

Introduction

"Jusque là, tout est clair"

Didier Bourdon

Contexte de l'étude

Les communications numériques ont connu un véritable essor ces dernières années avec notamment une demande accrue dans le domaine de la téléphonie mobile. Le but de tout système de communications numériques est de transmettre de l'information d'une source en un point A vers un destinataire en un point B par l'intermédiaire d'un canal de transmission qui peut être à support physique ou sans fil. La transmission est d'autant plus réussie que la distorsion entre le message original et le message reçu est faible.

Dans les schémas de communications classiques, le message à transmettre subit d'abord une opération de compression, afin de lui enlever le maximum de redondance possible. Puis, de la redondance structurée est ajoutée au message, afin de le rendre plus robuste à la transmission. En effet, transmettre un message à travers un canal introduit toujours une détérioration au message original. Le décodeur tente de retrouver le plus fidèlement possible le message original, en s'aidant notamment de la redondance structurée.

En quelque sorte, la redondance présente dans le message est initialement enlevée du message, avant d'être ré-introduite sous une autre forme. Ces opérations ont été motivées par un théorème de Shannon, appelé théorème de séparation. Ce théorème énonce que les opérations de compression et d'ajout de redondance peuvent être réalisées séparément sans perte d'optimalité. Cependant, ce théorème présente certaines limites comme il est montré dans [VVS95] par exemple. Ces limites ont poussé les chercheurs à entreprendre des recherches sur le codage conjoint source-canal, qui consiste à réaliser conjointement les opérations de compression et d'ajout de redondance.

Ainsi, les codes de source, qui réalisent la compression, ont beaucoup été étudiés ces dernières années. De nombreux chercheurs ont notamment travaillé sur des techniques visant à améliorer la robustesse de ces codes, ou des algorithmes d'estimation permettant d'améliorer leurs performances en utilisant des informations supplémentaires.

Nous avons travaillé dans cette thèse sur deux types de codes de sources : les codes à longueur variable (CLV) de type Huffman et les codes quasi-arithmétiques (QA). Ces deux types de codes sont souvent utilisés dans les schémas de compression modernes. Nous avons proposé et étudié un nouveau type de modèle d'état pour le décodage souple de ces codes, ainsi que des résultats complémentaires autour de ce modèle. Puis, dans le dernier chapitre, les codes QA ont été utilisés dans un contexte de codage de source distribué. Les contributions de cette thèse, ainsi que son organisation sont détaillées dans ce qui suit.

Organisation du document et contributions

Le premier chapitre de cette thèse présente le cadre théorique de ce travail. Les notions de théorie de l'information qui seront utiles à la compréhension du document sont présentées. La théorie de l'information a été instaurée par Shannon dans la fin des années 1940 dans le but de donner une formulation mathématique aux systèmes de communication.

Le chapitre 2 introduit le concept de codage de source et présente les deux types de codes utilisés dans ce document : les CLV et les codes arithmétiques et QA. Puis, les principales techniques de la littérature en codage et décodage conjoint source-canal sont passées en revue. Nous insisterons notamment sur les techniques utilisant des CLV et des codes QA. Nous nous focaliserons également sur les modèles d'état sur lesquels le décodage de ces codes est réalisé.

Le modèle d'état pour le décodage souple des CLV proposé dans [JMG05] est rappelé dans le chapitre3. Il est ensuite étendu aux codes QA. Ce modèle d'état permet de réduire de façon significative la complexité du décodage par rapport aux méthodes de la littérature, et pour un niveau de performance équivalent. Dans ce chapitre, nous présenterons également une méthode d'ajout de redondance contrôlé pour les CLV et codes QA. Cette redondance se présente sous la forme de contraintes de longueur partielles référençant différents instants du processus de décodage. La technique proposée est une solution alternative aux solutions proposées dans la littérature. Enfin, nous verrons que les probabilités a posteriori au niveau symbole ne peuvent pas être minimisées directement sur le modèle d'état proposé dans le chapitre 3. Ces probabilités a posteriori sont nécessaires lorsqu'un système itératif est mis en place, ou simplement lorsque l'on souhaite minimiser le taux d'erreur symbole en sortie du décodage.

Dans le chapitre 4, nous utiliserons les propriétés de resynchronisation des codes pour analyser leurs performances sur le modèle d'état agrégé présenté dans le chapitre 3. Pour cela, nous rappellerons d'abord les travaux de Maxted et Robinson, ainsi que ceux de Swaszek et Di Cicco. Ces travaux permettent de calculer la longueur moyenne de propagation d'une erreur bit, ainsi que la loi de probabilité de la différence entre le nombre de symboles encodés et décodés lorsqu'une erreur bit se produit dans un train binaire encodé avec un CLV. Nous verrons que la première quantité reflète les performances des CLV lorsqu'un décodage dur est appliqué. Les travaux de ces auteurs sont basés sur un diagramme, appelé *error state diagram* permettant de calculer les deux quantités décrites ci-dessus. Afin d'étendre ce résultat aux codes QA, nous avons proposé un nouveau type d'*error state diagram* adapté à ces codes. Grâce à ce diagramme, les résultats obtenus pour les CLV peuvent être étendus aux codes QA. Puis, nous montrerons que la densité de probabilité de la différence entre le nombre de symboles encodés et décodés permet d'analyser les performances des codes lorsqu'un décodage souple est appliqué au décodeur, ainsi que les performances sur le modèle agrégé présenté dans le chapitre 3.

Enfin dans le chapitre 5, nous présenterons un schéma de codage de source distribué utilisant des codes QA. Nous verrons que la plupart des schémas de codage de source distribué proposés dans la littérature sont basés sur des codes de canal. Nous proposerons deux approches différentes : l'une basée sur des codes QA poinçonnés et l'autre utilisant un nouveau type de code appelé code QA avec recouvrement d'intervalle. L'étude de ces code a été réalisée en collaboration avec X. Artigas et L. Torres de l'Université Polytechnique de Catalogne. Ces nouveaux codes permettent de réaliser directement la compression d'un message à un taux inférieur à son entropie. Ils sont donc très adaptés au problème de codage de source distribué. Les deux méthodes proposées seront utilisées dans des structures itératives afin d'améliorer leurs performances respectives. Nous verrons que ces solutions basées sur des codes QA sont compétitives par rapport aux solutions basées sur des codes de canal, notamment lorsque l'on utilise des courtes séquences.

En conclusion, nous ferons une synthèse des résultats obtenus et donnerons quelques perspectives sur de futurs axes de travail.

Chapitre 1

Introduction à la théorie de l'information et au codage de source

---

"L'auteur s'écrit à lui-même pour se dire des choses qu'il ne pourrait comprendre autrement"

Carlos Ruiz Zafon

Dans ce chapitre, nous allons présenter des notions qui seront utilisées tout au long de cette thèse. Nous détaillerons tout d'abord quelques principes de la théorie de l'information qui a été fondée par Claude Shannon vers la fin des années 1940. Le fondement de cette théorie est d'analyser le système de la figure 1.1, appelée "système de communication". Le but premier de la théorie de l'information est de donner une formulation mathématique à chacun des blocs de cette figure. La source est une entité qui produit l'information à transmettre, sous forme de messages. L'encodeur associe à chaque message issu de la source un autre message plus adapté à la transmission (message binaire par exemple pour les communications numériques). Le canal est le support sur lequel le message est transmis. Ce canal introduit un bruit dans le message. La sortie du canal est fournie au décodeur, dont la fonction principale est de reconstruire le message original. Nous détaillerons dans ce chapitre les représentations mathématiques de chacun de ces blocs. Nous nous arrêterons dans ce chapitre à définir les notions de théorie de l'information qui nous seront utiles pour la suite, ainsi que quelques propriétés. Il existe de nombreux ouvrages traitant de la théorie de l'information en détail comme [CT91],[Mac03], ou encore [Gra90]. Les démonstrations des propriétés présentées dans ce chapitre peuvent être trouvées dans ces ouvrages.

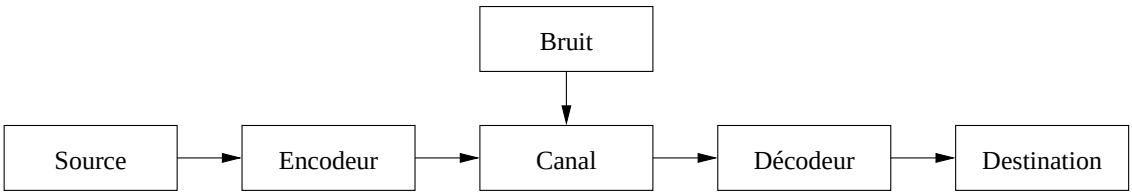

FIG. 1.1 – Système de communication

1.1 Éléments de théorie de l'information

1.1.1 Quantité d'information d'une source

La source représentée sur la figure 1.1 émet un message $\mathbf{S} = S_1, \ldots, S_{L(\mathbf{S})}$ que l'on souhaite transmettre à un utilisateur. On va supposer que ce message est constitué d'une suite de symboles issus d'un alphabet $\mathcal{A} = \{a_1, \ldots, a_n\}$. Ces symboles peuvent représenter par exemple des pixels si l'on transmet une image ou des lettres si l'on transmet du texte. Le message $\mathbf{S}$ est donc constitué de $L(\mathbf{S})$ réalisations d'une variable aléatoire A à valeurs dans l'alphabet $\mathcal{A}$. On note $\mathbb{P}(a_i)$ la probabilité de l'évènement $\mathbb{P}(A = a_i)$. L'entité "source" de la figure 1.1 est donc définie par une variable aléatoire A à valeurs dans un alphabet $\mathcal{A}$ et de loi de probabilité fixée.

Nous allons maintenant définir des notions permettant de mesurer la quantité d'information contenue dans une source (de façon équivalente dans une variable aléatoire A). Cette quantité d'information est appelée entropie d'une source (ou entropie d'une variable aléatoire). Il paraît assez intuitif de penser que plus un évènement est rare, plus la quantité d'information qu'il contient est importante. Inversement, la quantité d'information d'un évènement certain (qui ne comporte aucune incertitude) doit être nulle. La définition de l'entropie d'une variable aléatoire est la suivante

Définition 1.1 *L'entropie $H(A)$ d'une variable aléatoire A est donnée par*

$$H(A) = -C \sum_{1 \leq i \leq n} \mathbb{P}(a_i) \log(\mathbb{P}(a_i)), \tag{1.1}$$

où C est une constante et où la base du logarithme peut être n'importe quel nombre supérieur à 1. En général, le logarithme est pris en base 2 et C est fixé comme étant égal à 1. Ainsi, l'entropie d'une variable aléatoire binaire uniforme est égale à 1. C'est ainsi que nous utiliserons l'entropie dans la suite de cette thèse.
D'après la définition 1.1, l'entropie d'une variable aléatoire est positive ou nulle, l'entropie étant nulle si la variable représente un évènement certain. Cette quantité d'information peut également être vue comme la quantité de hasard associée à une variable aléatoire. L'entropie se mesure en bits. Nous verrons par la suite qu'elle représente également le nombre moyen de bits nécessaires pour représenter les réalisations

de la variable aléatoire associée.

En utilisant la définition 1.1, on peut également définir l'entropie conjointe de deux variables aléatoires A_1 et A_2 en utilisant la densité de probabilité jointe du couple (A_1, A_2). Elle représente la quantité d'information de ce couple.

Théorème 1.1 *L'entropie conjointe de deux variables aléatoires A_1 et A_2 vérifie*

$$H(A_1, A_2) \leq H(A_1) + H(A_2). \tag{1.2}$$

Cas d'égalité lorsque A_1 et A_2 sont indépendantes.

Ce théorème peut s'étendre aux n-uplets de variables aléatoires avec $n \leq 2$.

On peut définir de la même façon l'entropie conditionnelle entre deux variables aléatoires $H(A_1 \,|A_2)$ en utilisant la loi de probabilité conditionnelle $\mathbb{P}(A_1|A_2)$. Supposons que l'on soit en présence de deux variables aléatoires A_1 et A_2, et que l'on ne connaisse que les réalisations de A_1. Intuitivement, on peut penser que la quantité d'information qu'il manque pour connaître celle du couple (A_1, A_2) est $H(A_2 \,|\, A_1)$. Ceci est illustré par le théorème suivant

Théorème 1.2

$$H(A_1, A_2) = H(A_1) + H(A_2 \,|\, A_1) = H(A_2) + H(A_2 \,|\, A_1). \tag{1.3}$$

Ces notions d'entropie et d'entropie conditionnelle permettent également de calculer la quantité moyenne d'information partagée par deux variables aléatoires. Cette quantité est appelée information mutuelle et est définie par

Définition 1.2 *L'information mutuelle $I(A_1, A_2)$ entre deux variables aléatoires A_1 et A_2 est définie par*

$$I(A_1, A_2) = H(A_1) - H(A_1 \,|\, A_2) \tag{1.4}$$

L'information mutuelle I vérifie les propriétés suivantes :

$$\begin{cases} I(A_1, A_2) = 0 & \text{si } A_1 \text{ et } A_2 \text{ sont indépendants.} \\ I(A_1, A_1) = H(A_1) \end{cases} \tag{1.5}$$

Ces propriétés signifient que la quantité d'information partagée entre deux variables aléatoires indépendantes est nulle, et que celle partagée par une variable aléatoire et elle-même est tout simplement l'information de cette variable, soit son entropie.

1.1.2 Codage sans perte de l'information

On va maintenant s'intéresser à la représentation de l'information, appelée codage de source. Cette opération est réalisée par l'encodeur de la figure 1.1. Le rôle de cet encodeur est de décrire la séquence de symboles issue de la source, en utilisant des éléments d'un alphabet $\mathcal{A}'$. Pour des raisons pratiques $\mathcal{A}$ est souvent l'alphabet binaire. L'objectif principal du codage de source est de minimiser la longueur moyenne de représentation du message original. Cette représentation de l'information est faite à l'aide d'un code. Un code de source peut être vu comme une fonction qui à une séquence d'éléments de $\mathcal{A}$ associe une séquence d'éléments de $\mathcal{A}'$. En appelant $\mathcal{A}^*$, l'ensemble des séquences formées à partir d'éléments de $\mathcal{A}$, on peut représenter un code de source $\mathcal{C}$ de la façon suivante :

$$\begin{array}{ccc} \mathcal{A}^* & \rightarrow & \mathcal{A}'^* \\ \mathbf{S} & \mapsto & \mathcal{C}(\mathbf{S}) \end{array} \tag{1.6}$$

Le premier objectif du codage de source est de minimiser la longueur de $\mathcal{C}(\mathbf{S})$. Le deuxième objectif, qui justifie le terme *sans pertes*, est de créer un code qui soit uniquement décodable. En d'autres termes, la fonction qui représente le code $\mathcal{C}$ doit être injective. Dans ce cas, tout élément de $\mathcal{A}'^*$ a au plus un antécédent dans $\mathcal{A}^*$.
Dans la définition d'un code de source (équation 1.6), l'ensemble de départ de la fonction est l'ensemble $\mathcal{A}^*$. Un code de source peut également être défini sur l'ensemble $\mathcal{A}$, et ainsi associer à chaque élément de $\mathcal{A}$ un élément de $\mathcal{A}'^*$. C'est le cas par exemple des codes de Huffman [Huf52]. Nous allons maintenant donner quelques résultats de théorie de l'information sur les codes sources en général.

1.1.2.1 Codes uniquement décodable et codes instantanés

Nous allons considérer, pour présenter les résultats suivants, des codes de source qui associent à chaque symbole de l'alphabet $\mathcal{A}$ une séquence d'éléments de $\mathcal{A}'^*$. Une séquence d'éléments de $\mathcal{A}'^*$ associée à un symbole de $\mathcal{A}$ est appelée un mot de code. Pour assurer le caractère uniquement décodable d'un code de source, il apparaît assez clair que des restrictions doivent être placées sur l'affection des mots de code, comme on peut le constater dans l'exemple suivant.

Exemple 1.1: Prenons le code défini de l'alphabet $\mathcal{A} = \{a_1, a_2, a_3, a_4\}$ vers $\mathcal{A}' = \{0, 1\}$ par :

$$\begin{array}{ccc} a_1 & \mapsto & 0 \\ a_2 & \mapsto & 010 \\ a_3 & \mapsto & 01 \\ a_4 & \mapsto & 10 \end{array} \tag{1.7}$$

La séquence binaire 010 peut être obtenue en encodant un des trois messages a_2, $a_3\, a_1$ ou $a_1\, a_4$. Ainsi cette séquence binaire ne peut pas être décodée correctement. Ce code ne permet pas un codage sans perte.

En théorie de l'information, la définition d'un code sans perte est la suivante :

Définition 1.3 *Un code est dit sans perte si chaque séquence finie de mots de code correspond à au plus une séquence de symboles*

Une façon d'assurer le caractère sans perte d'un code est de former les mots de code tels qu'aucun des mots de code ne soit un préfixe d'un autre mot de code. Un tel code est appelé *code instantané*. Les codes instantanés présentent la propriété suivante :

Propriété 1.1 *Un code instantané est un code uniquement décodable.*

Cependant, la réciproque n'est pas vraie : un code uniquement décodable n'est pas forcément un code instantané.
Les codes instantanés présentent l'avantage d'être plus facilement décodables. De plus, nous verrons dans la suite de cette section que l'on peut, sans perte de généralité, se restreindre à l'étude des codes instantanés pour le problème de codage sans perte.

1.1.2.2 Conditions d'existence des codes

Supposons que l'on souhaite construire un code de l'alphabet $\mathcal{A}$ vers l'alphabet $\mathcal{A}'$. On note $l(a_i)$ longueur du mot de code associé au symbole a_i de l'alphabet $\mathcal{A}$. La communauté de théorie de l'information s'est attaquée dans les années 50 à trouver des conditions sur les longueurs $l(a_i)$ afin d'assurer l'existence de codes instantanés et/ou uniquement décodables entre $\mathcal{A}$ et $\mathcal{A}'$. Un premier résultat a été formulé par Kraft en 1949 par l'intermédiaire du théorème suivant

Théorème 1.3 *Un code instantané dont les mots de code sont de longueurs $l(a_1), \ldots, l(a_n)$ existe si et seulement si*

$$\sum_{i=1}^{n} card(\mathcal{A}')^{-l(a_i)} \leq 1 \tag{1.8}$$

Ce résultat a été étendu par Mc Millan en 1956 aux codes uniquement décodables. Il a montré que les longueurs des mots de code d'un code uniquement décodable vérifient également l'inégalité de Kraft (1.8).

1.1.2.3 Théorème de codage sans perte

Nous venons d'énoncer les conditions d'existence des codes pour des alphabets donnés. Le résultat donné par le théorème 1.3 ne permet néanmoins pas de répondre au problème premier du codage de source qui est de minimiser la longueur moyenne de représentation d'une source. Considérons une variable aléatoire prenant ses valeurs sur un alphabet $\mathcal{A} = \{a_1, \ldots, a_n\}$ de probabilités $\{\mathbb{P}(a_1), \ldots, \mathbb{P}(a_n)\}$ et un code $\mathcal{C}$ associé à cette source. Nous supposerons dans cette section que l'alphabet d'arrivée du

code est l'alphabet binaire $\mathcal{A}' = \{0, 1\}$. Les longueurs des mots de code de $\mathcal{C}$ sont notées $l(a_1), \ldots, l(a_n)$. Le problème du codage sans perte est de créer un code uniquement décodable qui minimise la longueur moyenne $\bar{l} = \sum_{i=1}^{n} \mathbb{P}(a_i) l(a_i)$. Shannon a démontré le théorème suivant :

Théorème 1.4 *Soit A une variable aléatoire prenant ses valeurs dans un alphabet $\mathcal{A} = \{a_1, \ldots a_n\}$ de probabilités $\mathbb{P}(a_1), \ldots \mathbb{P}(a_n)$. Si $\bar{l} = \sum_{i=1}^{n} \mathbb{P}(a_i) l(a_i)$ est la longueur de description moyenne d'un code uniquement décodable associé à la variable aléatoire A, alors $\bar{l} \geq H(A)$. Le cas d'égalité est obtenu si et seulement si $\mathbb{P}(a_i) = 2^{-l(a_i)}$, pour $i = 1, \ldots, n$.*

Ce théorème est appelé *théorème de codage sans perte*. D'après ce résultat, on se rend compte que l'entropie d'une variable aléatoire représente le nombre moyen minimal de bits permettant de la représenter. On remarque également que pour construire un code dont la longueur de description moyenne soit égale à l'entropie de la source, il faut que les probabilités des symboles de l'alphabet soient diadiques, ce qui n'est pas toujours le cas (les probabilités de la source sont fixées). Le théorème suivant a néanmoins été démontré :

Théorème 1.5 *Pour toute variable aléatoire A, il existe un code instantané associé à A et dont la longueur de description moyenne $\bar{l}$ vérifie*

$$H(A) \leq \bar{l} < H(A) + 1. \tag{1.9}$$

Ainsi, il est possible de trouver des codes instantanés dont la longueur de description moyenne est supérieure de moins de 1 bit par symbole à la borne inférieure donnée par l'entropie de la source. Le seul point restant pour résoudre le problème de codage sans perte est la construction de codes *optimaux*, dont l'existence est assurée par le théorème 1.5. On voudrait, pour une source donnée, trouver le meilleur code en terme de longueur de description moyenne, c'est-à-dire le code, parmi tous les codes possibles, dont la longueur de description moyenne est la plus petite. Un premier résultat issu de la théorie de l'information sur le codage sans perte montre que l'on peut se restreindre, sans perte de généralité, à l'ensemble des codes instantanés pour trouver les codes *optimaux*. Un code *optimal* doit forcément vérifier les conditions données par le lemme suivant :

Lemme 1.1 *Soit C un code instantané associé à une variable aléatoire A prenant ses valeurs dans un alphabet $\mathcal{A} = \{a_1, \ldots, a_n\}$ de probabilités $\mathbb{P}(a_1), \ldots \mathbb{P}(a_n)$. On suppose que les symboles de A sont rangés par ordre décroissant de probabilité ($\mathbb{P}(a_1) \geq \mathbb{P}(a_2), \ldots, \geq P(a_n)$) et que les groupes de symboles de mêmes probabilités sont rangés par ordre croissant en fonction de la longueur de leurs mots de codes associés. Alors, si C est optimal, ce code doit satisfaire les conditions suivantes :*

– *les mots de code associés aux symboles les plus probables sont les plus courts*
– *les mots de code associés aux deux symboles les moins probables sont de même longueur*
– *Parmi les mots de code de longueur $l(a_n)$, au moins deux mots de code doivent avoir les mêmes $l(a_n) - 1$ premiers bits*

Huffman a proposé dans [Huf52] une méthode pour construire des codes optimaux. Il existe néanmoins d'autres codes optimaux que les codes de Huffman. Les codes de Shannon généralisés sont également optimaux [Szp00] mais ne sont pas utilisé en pratique car ils sont trop complexes à générer. En revanche, les codes de Huffman sont très utilisés dans les systèmes de compression classiques. Les codes arithmétiques et QA sont également optimaux lorsque la taille de la séquence tend vers l'infini. Ils sont également très utilisés dans les systèmes de communications. Nous reviendrons en détail sur les codes de Huffman et les codes QA dans le chapitre suivant.

1.1.3 Canaux et capacités de canal

Dans un système de communication, le canal est une entité qui représente le bruit subit par le message lors de la transmission. Il peut être représenté par une fonction qui prend un vecteur $\mathbf{X}$ en entrée et lui associe un vecteur $\mathbf{Y}$, version bruitée de $\mathbf{X}$ selon la loi de probabilité du canal $\mathbb{P}(\mathbf{Y}\,|\mathbf{X})$. Les ensembles d'entrée et de sortie du canal peuvent être finis ou continus selon l'application considérée. Dans ce document, deux types de canaux seront principalement utilisés :

– le canal à bruit additif blanc gaussien (BABG)

– le canal binaire symétrique (CBS).

Ces deux types de canaux seront détaillés dans la suite de cette section. Ils sont largement utilisés en théorie car leur simplicité permet de prouver des résultats analytiques. Il existe néanmoins des types de canaux plus complexes modélisant plus précisément les canaux réels.

Les premiers travaux en théorie de l'information sur les canaux ont été initiés par Shannon dans [Sha48]. Il introduit la notion de capacité de canal qui représente la limite supérieure de quantité d'information transmissible sur un canal avec une probabilité d'erreur arbitrairement petite. Dans le cas discret, la capacité du canal est égale au maximum d'information mutuelle entre $\mathbf{X}$ et $\mathbf{Y}$ sur toutes les distributions possibles de X en entrée. En d'autres termes, la capacité d'un canal discret est donnée par

$$C = \max_{p(X)} I(X, Y). \tag{1.10}$$

Nous allons maintenant présenter les deux types de canaux utilisés dans ce document ainsi que leur capacité.

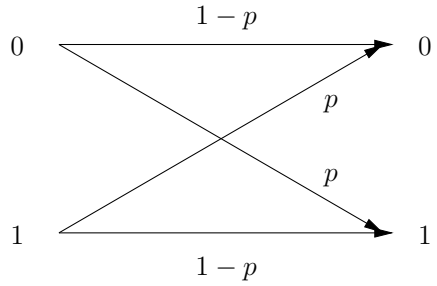

FIG. 1.2 – Canal binaire symétrique

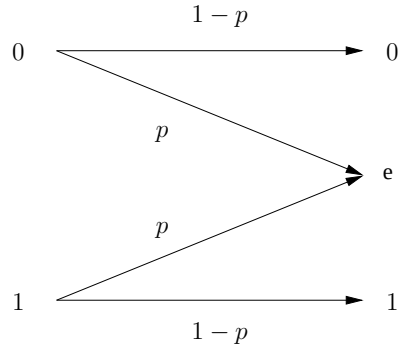

FIG. 1.3 – Canal à effacement

1.1.4 Modèles de canaux usuels et leur capacité

1.1.4.1 Canal binaire symétrique et canal à effacement

Le canal binaire symétrique, comme son nom l'indique, prend des éléments binaires en entrée et renvoie des éléments binaires en sortie. Il est représenté sur la figure 1.2. Il est défini de manière unique par un paramètre p (probabilité de *cross-over*) qui correspond à la probabilité que le symbole reçu Y soit différent du symbole émis X. Ce canal peut ainsi être modélisé par la matrice de transition suivante :

$$\mathbb{P}(Y = y \,|X = x) = \begin{pmatrix} 1-p & p \\ p & 1-p \end{pmatrix}. \tag{1.11}$$

L'information mutuelle entre $\mathbf{X}$ et $\mathbf{Y}$ (équation 1.4) vérifie

$$I(X, Y) \leq 1 - H(p). \tag{1.12}$$

Lorsque la distribution de $\mathbf{X}$ est uniforme, cette borne supérieure est atteinte. Ainsi, la capacité du CBS est donnée par $C = 1 - H(p)$.

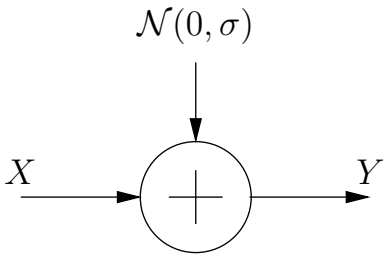

FIG. 1.4 – Canal à bruit additif blanc gaussien

Le canal à effacement est un canal très proche du CBS. Il est représenté sur la figure 1.3. L'état e représente un effacement, c'est à dire que le symbole transmis n'est pas reçu, on le considère alors comme inconnu. La probabilité p représente la probabilité d'effacement. La matrice de transition de ce canal est donnée par

$$\mathbb{P}(Y = y \,|X = x) = \begin{pmatrix} 1-p & 0 \\ 0 & 1-p \\ p & p \end{pmatrix}, \tag{1.13}$$

ce qui donne une capacité de $C = 1 - p$.

1.1.4.2 Canal à bruit additif blanc gaussien

Le canal BABG est représenté sur la figure 1.4. Le vecteur X en entrée peut être à valeurs continues ou à valeurs discrètes. Dans ce document, les données en entrée d'un canal BABG seront toujours discrétisées. L'ensemble des valeurs possibles de X sera toujours l'ensemble $\{-1, 1\}$. Pour obtenir un vecteur X de données à valeurs dans l'ensemble $\{-1, 1\}$, une modulation de type BPSK (*binary phase shift keying*) est appliquée au train binaire représentant le message à transmettre. En sortie du canal BABG, le vecteur $\mathbf{Y}$ est obtenue en additionnant à chaque élément de $\mathbf{X}$, un échantillon de bruit gaussien de moyenne 0 et de variance fixée σ. La capacité du canal BABG est donnée par l'expression suivante

$$C = \frac{1}{2}\log(1 + \frac{1}{\sigma^2}) \tag{1.14}$$

Si une décision dure est prise sur les valeurs en sortie d'un canal BABG, la probabilité d'erreur sur ce canal est donnée par l'expression suivante

$$1 - \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\sqrt{\frac{1}{\sigma^2}}} e^{-\frac{t^2}{2}} dt \tag{1.15}$$

Ainsi, en prenant une décision dure sur les valeurs en sortie d'un canal BABG, ce canal est équivalent à un CBS de probabilité de transition donnée par l'équation 1.15.

Néanmoins, on peut montrer que la capacité de ce canal est inférieure à celle du canal BABG.

1.1.5 Décodeur

La fonction première du décodeur est d'estimer à partir des observations issues du canal la séquence émise par la source et/ou la séquence de bits issues de l'encodage de la source selon l'application considérée. On souhaite évidemment minimiser la distorsion entre le message original et l'estimation faite par le décodeur. Plusieurs mesures peuvent être minimisées :

– le taux d'erreur symbole (TES) : représente le nombre de symboles erronés entre $\mathbf{S}$ et son estimée $\hat{\mathbf{S}}$
– le taux d'erreur bit (TEB) : représente le nombre de bits erronés entre $\mathbf{X}$ et son estimée $\hat{\mathbf{X}}$
– le taux d'erreur séquence (TESQ) : l'estimation $\hat{\mathbf{S}}$ est considérée comme fausse si au moins un de ses symboles est faux.

Afin de minimiser ces trois critères, les deux estimateurs suivants seront considérés dans ce document :

1. *Maximum a posteriori* : cet estimateur sélectionne la séquence $\hat{S}$ dont la probabilité d'avoir été émise sachant les observations Y, c'est à dire

$$\hat{S} = \arg\max_{S} \mathbb{P}(S \mid Y). \tag{1.16}$$

   Cet estimateur minimise le TESQ. En pratique, il est implémenté en utilisant l'algorithme de Viterbi [Vit67]. L'algorithme de Viterbi sera détaillé en annexe.

2. *Maximum of posterior marginals* : cet estimateur sélectionne une séquence $\hat{\mathbf{S}}$ ou $\hat{\mathbf{X}}$ composée à chaque instant k du symbole $\hat{S}_k$ ou du bit $\hat{X}_k$ ayant la plus forte probabilité d'avoir été émis, c'est à dire

$$\begin{aligned} \forall\, k, \quad \hat{S}_k &= \arg\max_{\mathcal{A}} \ \mathbb{P}(S_k \mid Y), \\ \forall\, k, \quad \hat{X}_k &= \arg\max_{\mathcal{A}'} \ \mathbb{P}(X_k \mid Y). \end{aligned} \tag{1.17}$$

Cet estimateur minimise le TES ou le TEB. Il est très souvent implémenté en pratique en utilisant l'algorithme BCJR [BCJR74], également appelé forward/backward ou algorithme somme/produit. Cet algorithme sera détaillé en annexe.

1.1.6 Conclusion

Dans cette première partie, nous avons présenté quelques résultats de théorie de l'information qui seront utilisés dans la suite du document : la mesure de l'information, le codage de source, les canaux et leur capacité ainsi que des éléments d'estimation

bayésienne. Dans le chapitre suivant, les codes de sources usuels utilisés dans ce document seront détaillés, ainsi qu'un état de l'art sur le décodage robuste et le décodage conjoint source/canal de ces codes.

Chapitre 2

État de l'art en décodage robuste des codes à longueur variable et quasi-arithmétiques

---

"Sur mille expériences que nous faisons, nous en exprimons tout au plus une par le langage. Parmi toutes ces expériences muettes sont cachées celles qui donnent secrètement à notre vie sa forme, sa couleur, sa mélodie"

Pascal Mercier

La chaîne de transmission utilisée dans les applications standards est représentée sur la figure 2.1. Un message $\mathbf{S}$ représenté sous la forme d'une suite de $L(\mathbf{S})$ symboles à valeurs dans un alphabet $\mathcal{A}$ est tout d'abord encodé par un codeur de source. Comme nous l'avons vu dans le chapitre 1, le codeur de source a pour but de représenter $\mathbf{S}$ par un message binaire en le compressant au maximum. Cette opération est réalisée en pratique par des codes entropiques, tels les codes à longueur variable ou codes arithmétiques et quasi-arithmétiques. Le message issu de l'encodage de source ne comportant que très peu de redondance, chaque élément binaire de ce message contient une quantité d'information importante. Ce message est donc très sensible au bruit sur le canal. Le rôle du codeur de canal est de rajouter de la redondance à ce message afin de le rendre plus robuste à la transmission. Le codage de canal permet en effet de détecter et corriger les erreurs issues de la transmission, dans la limite de ses capacités de correction. La séquence issue de ces deux opérations d'encodage sera notée $\mathbf{X}$ dans tout ce qui suit. Sa longueur, elle, sera notée $L(\mathbf{X})$. Dans ce document, certaines applications ne considèrent pas d'opération de codage de canal. Dans ce cas, la séquence $\mathbf{X}$ représentera la séquence issue de l'encodage de source de $\mathbf{S}$. L'opération de modulation vise à mettre $\mathbf{X}$ sous une forme plus adaptée à la transmission sur le canal. Au

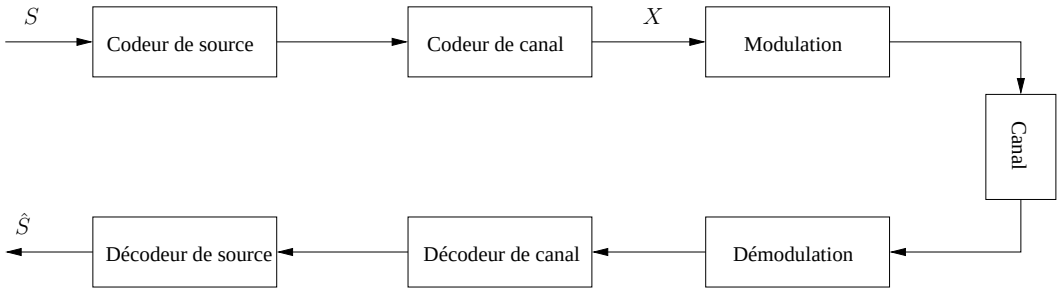

FIG. 2.1 – Chaîne de transmission classique

niveau du décodeur, les opérations inverses sont réalisées : démodulation, décodage de canal et décodage de source.
Ce schéma a été motivé en grande partie par le théorème de séparation, énoncé par Shannon. Cependant, les hypothèses de ce théorème ne sont pas vérifiées en pratique. Ainsi, le codage conjoint source/canal, visant à optimiser conjointement les opérations de codage de source et de codage de canal, s'est progressivement développé depuis quelques années.

Dans ce chapitre, nous présenterons tout d'abord dans la section 2.1 les limites du théorème de séparation de Shannon. Ces limites ont conduit les chercheurs dans le domaine de la compression a étudier les codes de source séparément dans le but de les intégrer dans des schémas de transmission conjoints. Nous présenterons en détail, dans les sections 2.2 et 2.3, les deux types de codes entropiques fréquemment utilisés dans les schémas de transmission : les CLV ainsi que les codes arithmétiques et QA. Enfin, dans la section 2.4, nous ferons un rapide état de l'art des méthodes de codage/décodage conjoint source-canal de la littérature, en se focalisant particulièrement sur les notions en rapport avec les contributions de cette thèse.

2.1 Limites du théorème de séparation

Les travaux en théorie de l'information ont montré qu'il était possible de transmettre des messages sur un canal bruité avec un taux d'erreur arbitrairement faible, à condition que le débit de transmission soit inférieur à la capacité du canal. Shannon a ensuite montré qu'il était possible d'atteindre ce résultat théorique en optimisant de manière séparée le codeur de source et le codeur de canal. Cependant, ce résultat est un résultat asymptotique qui est basé sur plusieurs hypothèses qui ne sont pas toujours vérifiées dans les schémas de communications réels. Le théorème de séparation suppose en effet que les séquences sont de longueur infinie et sont transmises sur un canal ergodique [Sha48][CT91]. Lors des transmissions réelles, les codeurs agissent sur des séquences de longueur finie, et les canaux de transmission ne sont pas forcément

stationnaires. De plus, les modèles de canaux utilisés en théorie de l'information ne reflètent pas exactement les canaux réels. Le théorème de séparation suppose en outre que la capacité de calcul disponible est non bornée. Dans les schémas de communication réels, la complexité au niveau du codeur/décodeur doit souvent être limitée pour assurer des contraintes en terme de délai de transmission et d'utilisation d'énergie notamment.
Enfin, le théorème de séparation propose une solution à l'objectif de transmission avec un taux d'erreur arbitrairement faible, mais ce théorème ne dit pas qu'il est nécessaire de réaliser séparément le codage de source et le codage de canal pour atteindre cet objectif.

Par conséquent, de nombreux travaux ont été menés ces dernières années en codage conjoint source-canal afin de proposer des solutions optimisant conjointement le codage de source et le codage de canal. Cette approche consiste à utiliser ou à ajouter de la redondance résiduelle au niveau du codage de source afin de donner aux codes des propriétés qu'utilise le décodeur pour améliorer les performances des codes de sources.
Les deux types de codes sources le plus souvent utilisés dans les schémas de compression sont les CLV (dans JPEG ou H.263+ par exemple) et les codes arithmétiques ou QA (dans JPEG2000 ou H.264 par exemple). Nous allons présenter ces deux types de code dans le reste de ce chapitre, ainsi que les nombreux travaux qui ont été menés sur ces codes dans le cadre de codage conjoint source-canal.

2.2 Codes à longueur variable

Les codes à longueur variable sont des codes qui à un symbole de l'alphabet de départ $\mathcal{A}$ associent un mot de code constitué d'éléments de l'alphabet d'arrivée $\mathcal{A}'$. Nous allons considérer dans ce qui suit que $\mathcal{A}'$ est l'alphabet binaire. Ainsi, les mots de code formés par un CLV seront binaires. L'association entre les symboles de $\mathcal{A}$ et les mots de code se fait en fonction de la probabilité d'occurrence de chaque symbole. Cela nécessite donc que la loi de probabilité de la source soit connue au moment de l'encodage ainsi qu'au décodage. C'est ce que nous supposerons dans le reste de ce chapitre. On peut néanmoins noter que si les probabilités de la source ne sont pas connues, il existe des méthodes permettant de les estimer, comme dans [MSJ06] par exemple.

2.2.1 Codes de Huffman

Les CLV les plus utilisés dans les systèmes de compression actuels sont les codes de Huffman. Dans [Huf52], Huffman a donné une méthode permettant de construire un code de Huffman associé à une source de loi de probabilité donnée. La construction de ces codes peut se résumer par l'algorithme 1.

Algorithme 1 Algorithme de Huffman

1. Classer les symboles (en ligne) par ordre croissant de probabilité
2. Relier les deux symboles de probabilité les plus faibles à l'aide de deux arrêtes. On obtient alors un noeud appelé "super-symbole" et dont la probabilité est égale à la somme des probabilité des deux symboles qui ont permis de le construire
3. Affecter aux deux symboles les moins probables les bits 0 et 1 respectivement
4. Réitérer l'étape 2 en considérant tous les super-symboles comme des symboles, tant qu'il y a plus d'un symbole restant
5. Associer à chaque symbole initial un mot de code en descendant l'arbre obtenu et en concaténant 0 si on suit le fils gauche et 1 si on suit le fils droit.

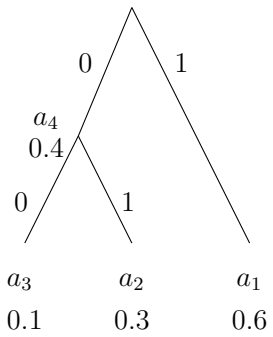

FIG. 2.2 – Construction de l'arbre de Huffman

Pour une source de cardinal n, la complexité de l'algorithme de Huffman est en $\mathcal{O}(n\log(n))$.

Exemple 2.1: Construisons le code de Huffman associé à la source composé de 3 *symboles* a_1, a_2 *et* a_3 *de probabilités respectives* $0.6, 0.3$ *et* 0.1*. L'arbre obtenu par l'algorithme précédant est représenté sur la figure 2.2. Le super-symbole* a_4 *a été construit à partir des symboles* a_2 *et* a_3 *de probabilités les plus faibles. La probabilité de* a_4 *est égale à* $0.1 + 0.3 = 0.4$*. Le code ainsi obtenu associe au symbole* a_1 *le mot de code* 1*, au symbole* a_2 *le mot de code* 01 *et au symbole* a_3 *le mot de code* 00*.*

On remarque que d'après la construction des codes de Huffman, ces codes vérifient bien les 3 conditions du Lemme 1.1, conditions nécessaires pour qu'un code soit optimal. On appellera dans la suite l_m et l_M les longueurs des mots de code respectivement le plus court et le plus long associé à un CLV.

Remarque 2.1 *Nous avons considéré le cas où les symboles de la source était codés un par un. Un code de Huffman peut également être défini pour des groupes de symboles. Il faut pour cela*

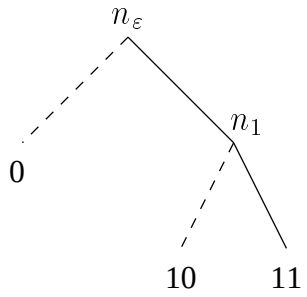

FIG. 2.3 – Arbre binaire associé au code $\mathcal{C}_0$.

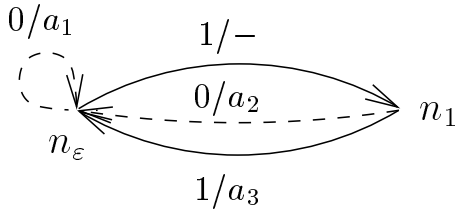

FIG. 2.4 – Automate d'encodage/décodage du code $\mathcal{C}_0$.

considérer toutes les séquences possibles (de la longueur désirée) de symboles de l'alphabet $\mathcal{A}$. Cette version généralisée des codes de Huffman a une complexité importante [Tja00]. Les codes de Huffman sont donc principalement utilisé sur l'alphabet de base de la source. Il est montré dans [BK93] que leur performances en terme de compression sont correctes lorsque l'alphabet $\mathcal{A}$ est assez grand.

2.2.2 Représentation d'un CLV sous forme d'arbre binaire

Un CLV peut se représenter de façon naturelle sous la forme d'un arbre binaire. L'arbre binaire d'un CLV comporte autant de noeuds internes que de préfixes dans les mots de code. Le noeud racine de l'arbre est noté n_ε.

Exemple 2.2: On considère une source d'alphabet $\mathcal{A} = \{a_1, a_2, a_3\}$ et le code associé $\mathcal{C}_0 = \{0, 10, 11\}$. Ce code admet un seul préfixe : 1. L'arbre binaire associé à ce code est représenté sur la figure 2.3.

On peut également associer aux CLV un automate d'encodage/décodage dont les états sont les noeuds de l'arbre binaire. Les probabilités des transitions sur cet automate sont calculées à partir de la loi de probabilité de la source. L'automate associé au code $\mathcal{C}_0$ est représenté sur la figure 2.4.
Un tel automate peut être utilisé pour l'encodage d'une source par un CLV ou pour réaliser le décodage dur d'éléments binaires reçus en sortie du canal. Nous verrons

a_i	a_1	a_2	a_3	a_4	a_5
$\mathbb{P}(a_i) =$	0.4	0.2	0.2	0.1	0.1
$\mathcal{C}_1$	00	01	10	110	111
$\mathcal{C}_2$	00	01	11	100	101
$\mathcal{C}_3$	00	10	11	010	011
$\mathcal{C}_4$	01	00	10	110	111
$\mathcal{C}_5$	01	00	11	100	101
$\mathcal{C}_6$	01	10	11	000	001
$\mathcal{C}_7$	0	10	110	1110	1111
$\mathcal{C}_8$	0	10	111	1100	1101
$\mathcal{C}_9$	0	11	100	1010	1011
$\mathcal{C}_{10}$	0	11	101	1000	1001
$\mathcal{C}_{11}$	0	100	101	110	111
$\mathcal{C}_{12}$	0	100	110	101	111
$\mathcal{C}_{13}$	0	100	111	110	101
$\mathcal{C}_{14}$	0	101	110	100	111
$\mathcal{C}_{15}$	0	101	111	100	110
$\mathcal{C}_{16}$	0	110	111	100	101

TAB. 2.1 – Source et codes de [ZZ02] utilisés dans ce rapport.

également par la suite que cet automate est utilisé pour définir les modèles d'état utilisés pour réaliser le décodage souple de séquences encodées par un CLV.
Les CLV utilisés dans ce document pour présenter des résultats de simulation sont répertoriés dans le tableau 2.1. Ces codes sont définis pour une source discrète sans mémoire de 5 symboles et de loi de probabilité

$$\mathbb{P}(a_1) = 0.4,\ \mathbb{P}(a_2) = \mathbb{P}(a_3) = 0.2,\ \mathbb{P}(a_4) = \mathbb{P}(a_5) = 0.1 \tag{2.1}$$

L'entropie de cette source est égale à 2.12 bits. Les 16 codes du tableau 2.1 sont optimaux. Leur longueur de description moyenne est de 2.2 bits par symbole. Ils ont été introduits dans [ZZ02].

2.3 Codage arithmétique et quasi-arithmétique

Récemment, les codes arithmétiques ont attiré l'attention des chercheurs dans le domaine de la compression par leur utilisation dans des standards de compression tels JPEG-2000, H.264 ou MPEG4. Ces codes permettent d'atteindre des performances en compression arbitrairement proches de l'entropie. Ils sont par contre, au même titre que les codes de Huffman, très sensibles au bruit du canal.
Les codes QA sont une version simplifiée des codes arithmétiques permettant de travailler sur des entiers et non sur des nombres réels afin d'éviter des problèmes de

précision numérique. Nous allons présenter ces deux types de codes dans cette section.

2.3.1 Codes arithmétiques

Le principe de base du codage arithmétique a été évoqué en premier par Shannon dans [Sha48]. Ce principe a ensuite donné lieu à de nombreuses mises en oeuvres pratiques du codage arithmétique comme celles de Rissanen [Ris76, Ris79], Pasco [Pas76], Langdon [Lan84], Witten [WNC87, WNC98] ou encore Howard et Vitter [HV92].

2.3.1.1 Encodeur

Contrairement au codage de type Huffman, le codage arithmétique classique réalise une correspondance entre une séquence complète de symboles et un train binaire. Le processus d'encodage par un code arithmétique est le suivant. Soit une source prenant ses valeurs dans un alphabet $\mathcal{A} = \{a_1, \ldots, a_n\}$, de probabilités $\mathbb{P}(a_1), \ldots, \mathbb{P}(a_n)$. On souhaite encoder une séquence $\mathbf{S} = s_1, \ldots, s_{L(\mathbf{S})}$. Le nombre de symboles $L(\mathbf{S})$ à émettre est supposé connu par l'encodeur. Un intervalle courant I_c est initialisé à $[0.0, 1.0[$. Puis, pour chaque symbole non traité de $\mathbf{S}$, on réalise les opérations suivantes

1. I_c est subdivisé en n sous-intervalles $I_c^1, \ldots, I_c^n$ correspondant à chacun des symboles de $\mathcal{A}$. La taille de chaque sous-intervalle I_c^i est proportionnelle à $\mathbb{P}(a_i)$.

2. On sélectionne le sous-intervalle correspondant au prochain symbole de $\mathbf{S}$ à traiter. Ce sous-intervalle devient l'intervalle courant I_c.

Lorsque tous les symboles de S ont été traités, il faut émettre un train binaire représentant sans ambiguïté le dernier intervalle courant. Ce processus d'encodage est illustré dans l'exemple suivant, ainsi que sur la figure 2.5.

Exemple 2.3: Considérons une source dont l'alphabet est $\mathcal{A} = \{a, b, c\}$ *de probabilité* $\mathbb{P}(a) = 0.6$, $\mathbb{P}(b) = 0.3$, $\mathbb{P}(c) = 0.1$. *On souhaite encoder la séquence* $\mathbf{S} = acb$. *L'intervalle* $I_c = [0.0, 1.0[$ *est d'abord subdivisé en* $I_c^1 = [0.0, 0.6[$, $I_c^2 = [0.6, 0.9[$ *et* $I_c^3 = [0.9, 1.0[$. *On sélectionne l'intervalle* I_c^1 *puisque le premier symbole de* $\mathbf{S}$ *est un* a. *L'intervalle courant devient donc* $I_c = [0.0, 0.6[$. *On répète ces opérations pour le deuxième symbole de* $\mathbf{S}$. *L'intervalle courant devient alors* $I_c = [0.54, 0.6[$. *Enfin, après traitement du dernier symbole de* $\mathbf{S}$, *l'intervalle final est égal à* $I_c = [0.576, 0.594[$. *Notons que la taille de l'intervalle final est égal à la probabilité de la séquence à émettre. L'écriture binaire de l'intervalle final étant* $I_c = [0.1000\ldots, 0.1001\ldots[$, *le train binaire résultant de l'encodage est* 100. *Ces trois bits sont suffisants pour distinguer l'intervalle final de tous les intervalles obtenus en encodant d'autres séquences.*

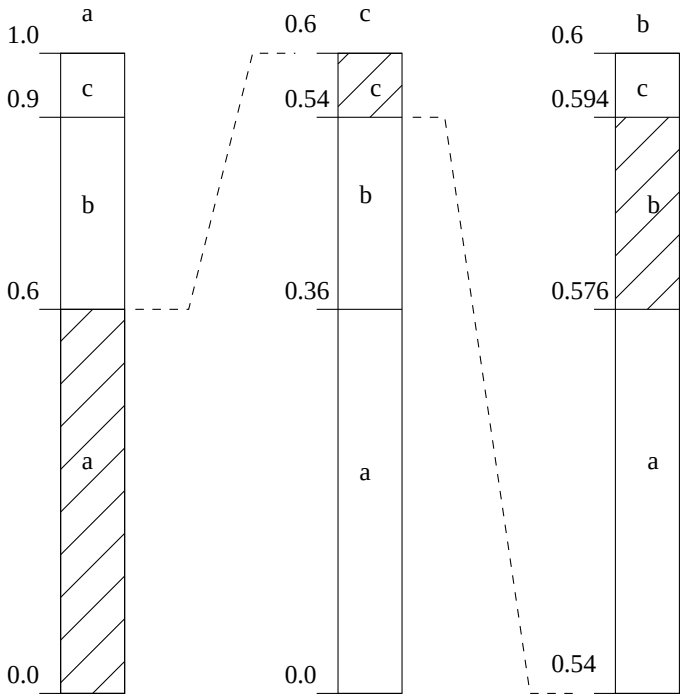

FIG. 2.5 – Exemple d'encodage arithmétique.

2.3.1.2 Décodeur

Le décodeur arithmétique agit de manière similaire au codeur arithmétique pour retrouver la séquence de symboles originale à partir de l'intervalle final, que l'on note ici I_f. On initialise un intervalle courant $I_c = [0.0, 1.0[$ divisé en sous-intervalles selon les probabilités $\mathbb{P}(a_1), \ldots \mathbb{P}(a_n)$. Puis pour chaque symbole qui n'est pas encore décodé (on suppose ici que le nombre total de symboles est connu), le décodeur réalise les opérations suivantes

1. Le sous-intervalle de I_c contenant entièrement I_f est sélectionné, permettant de décoder le symbole correspondant à ce sous-intervalle. Ce sous-intervalle devient l'intervalle courant
2. On subdivise l'intervalle courant I_c selon les probabilités de la source

Ces opérations sont répétées jusqu'à ce que le nombre de symboles décodés soit égal au nombre de symboles transmis. Le décodage du train binaire obtenu dans l'exemple 2.3 est illustré dans l'exemple suivant ainsi que sur la figure 2.6

Exemple 2.4: L'intervalle final I_f est l'intervalle $[0.576, 0.594[$. L'intervalle $I_c = [0.0, 1.0[$ est d'abord subdivisé en $I_c^1 = [0.0, 0.6[$, $I_c^2 = [0.6, 0.9[$ et $I_c^3 = [0.9, 1.0[$. I_f est inclus entièrement dans le sous-intervalle I_c^1. Ainsi, le symbole a est décodé et l'intervalle

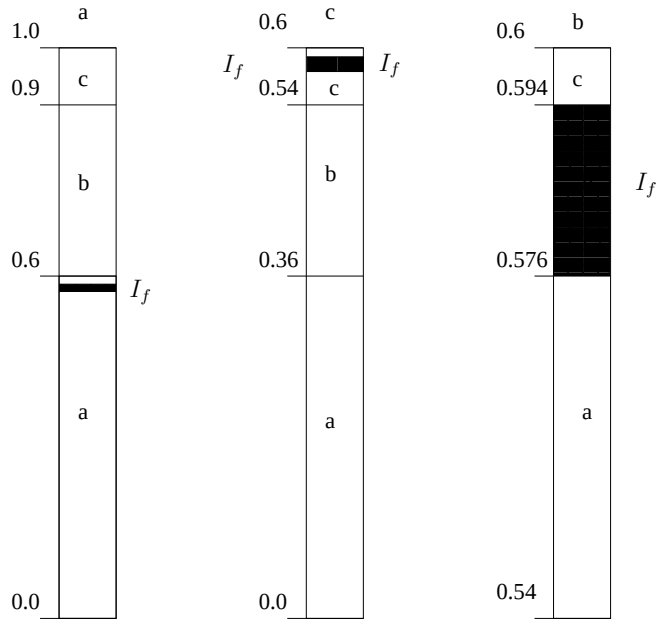

FIG. 2.6 – Exemple de décodage arithmétique.

courant devient $I_c = [0.0, 0.6[$. *La subdivision de* I_c *crée les sous-intervalles* $I_c^1 = [0.0, 0.36[$, $I_c^2 = [0.36, 0.54[$ *et* $I_c^3 = [0.54, 0.6[$. *A présent,* I_f *est inclus entièrement dans* I_c^3. *Le symbole* c *est donc décodé et l'intervalle courant devient* $I_c = [0.54, 0.6[$. *En répétant les mêmes opérations une dernière fois, le symbole* b *est décodé. La séquence originale* acb *a ainsi été retrouvée.*

La description que nous venons de faire d'un code arithmétique est la description théorique, telle que cela a été présenté dans [Sha48] ou dans [Abr63]. Lors de l'implémentation du procédé décrit ci-dessus, plusieurs problèmes peuvent survenir, comme celui de la précision numérique et du délai d'encodage. En effet, lorsque la séquence originale est longue, l'intervalle courant devient vite très petit et nous ne disposons pas forcément de machines permettant de travailler avec une telle précision numérique. De plus, on remarque que le train binaire résultant de l'encodage n'est déterminé qu'une fois que toute la séquence originale a été traitée. Ces problèmes ont été résolus par Witten *et al.* dans [WNC87] en s'appuyant sur la représentation binaire des nombres réels compris entre 0 et 1. En effet, lorsque l'intervalle courant est entièrement inclus dans $[0, 0.5[$, sa représentation binaire commencera forcément par un 0. De même, lorsque l'intervalle courant est contenu entièrement dans $[0.5, 1[$, sa représentation binaire commencera par un 1. Ainsi, lorsque durant le processus d'encodage un de ces deux cas se produit, un bit peut être émis et la taille de l'intervalle courant est doublé. Un traitement particulier doit être mis en oeuvre lorsque l'intervalle cou-

rant chevauche 0.5 et est entièrement contenu dans $[0.25, 0.75[$. Dans ce cas, l'intervalle courant ne peut être identifié par un bit unique. La taille de l'intervalle courant est tout de même doublée, et l'on stocke le nombre de fois où cette opération est effectuée dans un entier que l'on nomme *follow*. Puis, lorsqu'un bit doit être émis (lors du traitement d'un symbole prochain), ce bit sera suivi de *follow* fois son complément. Lorsque ces bits sont émis, *follow* est remis à zéro.
Cette mise en oeuvre pratique du codage arithmétique permet d'assurer que $i < 0.25 < 0.5 \leq s$ ou que $i < 0.5 < 0.75 \leq s$, où i et s représentent respectivement les bornes inférieures et supérieures de l'intervalle courant, permettant ainsi de limiter les problèmes de précision numérique. Cette méthode est décrite de façon plus détaillée dans [HV92, WNC98, Say00] par exemple.

2.3.2 Codes quasi-arithmétiques

Les codes arithmétiques présentent l'inconvénient de ne pas pouvoir être représentés par une machine à états finis ou un automate, contrairement aux codes de Huffman par exemple. De plus, même si la distribution de la source est connue, le nombre de subdivisions possibles de l'intervalle courant croît de manière exponentielle avec le nombre de symboles encodés.
Les auteurs de [HV92] ont montré qu'en réduisant la précision d'un codeur arithmétique, on pouvait diminuer le nombre d'états sans une perte excessive en terme d'efficacité de compression. Ce type de codes est appelé code quasi-arithmétique. Ces codes opèrent sur l'intervalle $[0, N[$, où N est un entier plutôt que sur l'intervalle $[0, 1[$. Ainsi, les intervalles sont toujours des intervalles entiers, ce qui permet de limiter le nombre de subdivisions possibles et donc éviter les problèmes de précision numériques. Suivant la précision adoptée (la valeur de l'entier N), les performances en compression des codes QA varient entre celles des codes de Huffman et celles des codes arithmétiques. L'avantage majeur des codes QA par rapport aux codes arithmétiques est qu'ils peuvent être représentés par une machine à états finis. Le nombre de subdivisions possibles de l'intervalle courant ne varie ainsi pas avec le nombre de symboles à encoder. De nombreux travaux ont été réalisés ces dernières années sur les codes QA [Lan84][Gor94][BJWK06][DHS06], pour assurer notamment la représentation des codes QA sous forme de machine à états finis.

2.3.2.1 Construction d'un code QA

Un code QA est défini par deux paramètres : la probabilité de la source $\mathbb{P}(0) = p$, l'entier N qui représente la précision du code. Pour des raisons de simplicité de notation, on supposera que l'entier N peut s'écrire $N = 2^k$, avec $k > 1$. Il est montré dans [DHS06] et [BJWK06] que l'encodeur d'un code QA pouvait se mettre sous la forme d'une machine à états finis en définissant les états par le triplet $(i, s, follow)$. i et s représentent respectivement les bornes inférieures et supérieures de l'intervalle

courant. On peut tout de même noter que cette représentation de l'encodeur a été utilisée en premier dans [GG04], mais le nombre d'états pouvait ne pas être borné. En effet, une technique a été mise en oeuvre dans [BJWK06] et [DHS06] afin de garder le nombre d'états fini. Nous détaillerons ceci dans cette section. Tout d'abord, nous allons définir des règles de remise à l'échelle d'un intervalle $[i, s[$. Ces règles ont été proposées dans [WNC98] pour le codage arithmétique et QA. Un intervalle $[i, s[$ est remis à l'échelle dans les 3 cas suivants :

1. si $s < N/2$, alors $i \to 2 \times i$ et $s \to 2 \times s$. De plus, un bit 0 est émis, suivi de $follow$ fois 1. Puis $follow$ est remis à 0.
2. si $i > N/2$, alors $i \to 2 \times (i - N/2)$ et $s \to 2 \times (s - N/2)$. De plus, un bit 1 est émis suivi de $follow$ fois 0. Puis $follow$ est remis à 0.
3. si $N/4 \leq i < N/2 \leq s < 3N/4$, alors $i \to 2 \times (i - N/4)$ et $s \to 2 \times (s - N/4)$. De plus, $follow$ est incrémenté de 1.

Les règles de construction d'un code QA binaire sont les suivantes

1. L'état initial de l'automate d'encodage est l'état $(0, N, 0)$. L'intervalle courant initial est $[0, N[$ et $follow$ est initialisé à 0.
2. I_c est partitionné en deux sous-intervalles $I_c^1 = [i^1, s^1[$ et $I_c^2 = [i^2, s^2[$ selon la probabilité p.
3. I_c^1 et I_c^2 sont remis à l'échelle si besoin (cf. règles ci-dessus), et autant de fois que nécessaire.
4. Deux états sont obtenus : $(i^1, s^1, follow^1)$ et $(i^2, s^2, follow^2)$.
5. Les opérations 2 et 3 sont répétés pour chaque nouvel état créé.

En utilisant l'algorithme décrit ci-dessus pour construire l'automate d'encodage d'un code QA, le nombre d'états obtenu n'est pas forcément fini. En effet, il est possible que la valeur de $follow$ augmente infiniment comme on va le voir dans l'exemple suivant.

Exemple 2.5: Supposons que l'on souhaite construire un code QA de précision $N = 8$ pour une source binaire de probabilité $\mathbb{P}(0) = p = 0.6$. L'état initial est l'état $(0, 8, 0)$. Le partitionnement de l'intervalle $[0, 8[$ selon p crée les deux sous-intervalles $[0, 5[$ et $[5, 8[$. Le sous-intervalle $[0, 5[$ ne nécessite pas de remise à l'échelle. Par contre, l'intervalle $[5, 8[$ est remis à l'échelle selon la deuxième règle. On obtient alors l'intervalle $[2, 8[$. Les deux nouveaux états sont alors $(0, 5, 0)$ et $(2, 8, 0)$. On doit maintenant répéter ces opérations pour ces nouveaux états. Le partitionnement de l'intervalle $[2, 8[$ crée les deux sous-intervalles $[2, 6[$ et $[6, 8[$. Le sous-intervalle $[2, 6[$ nécessite d'être redimensionné selon la troisième règle en l'intervalle $[0, 8[$. Comme la troisième règle de remise à l'échelle a été utilisée, $follow$ est incrémenté de 1. Ainsi, on obtient une transition entre l'état $(0, 8, 0)$ et l'état $(0, 8, 1)$ en passant par l'état $(2, 8, 0)$. En répétant

État courant	Symbole source	État suivant	Bits émis
$(0,8,0)$	a	$(0,7,0)$	$-$
$(0,8,0)$	b	$(0,8,0)$	111
$(0,7,0)$	a	$(0,6,0)$	$-$
$(0,7,0)$	b	$(0,8,0)$	110
$(0,6,0)$	a	$(0,5,0)$	$-$
$(0,6,0)$	b	$(0,8,0)$	101
$(0,5,0)$	a	$(0,8,0)$	0
$(0,5,0)$	b	$(0,8,0)$	100

TAB. 2.2 – Construction de l'automate d'encodage d'un code QA pour $N = 8$ et $p = 0.9$

les mêmes opérations, on va créer une transition entre l'état $(0,8,0)$ *et l'état* $(0,8,2)$*, et ainsi de suite. L'automate d'encodage ainsi obtenu n'aura pas un nombre d'états fini.*

Pour s'assurer que le nombre d'états d'un automate d'encodage de code QA soit fini, les auteurs de [BJWK06] et [DHS06] ont introduit une valeur limite à la variable $follow$. Dès que la valeur limite de $follow$ est atteinte, on force l'encodeur à émettre des bits (c.a.d., à suivre les règles 1 ou 2 de remise à l'échelle) et ainsi $follow$ est réinitialisé à 0. Cette technique induit une très légère diminution dans les performances de compression des codes QA, mais assure que le nombre d'états de l'automate d'encodage est fini.

Exemple 2.6: La construction de l'automate d'encodage du code QA défini pour $N = 8$ *et* $p = 0.9$ *est illustrée dans le tableau 2.2. Cet automate comporte 4 états. Il est représenté sur la figure 2.7. Les transitions dont les sorties sont labelisées d'un* $-$ *sont dites muettes, aucun bit n'est émis le long de ces transitions. Ce code sera appelé* $\mathcal{Q}_9$ *dans le reste de ce document.*

L'automate de décodage d'un code QA peut être obtenu à partir de l'automate d'encodage en suivant la méthode présentée dans [BJWK06]. L'automate de décodage pour le code QA de paramètre $N = 8$ et $p = 0.9$ est également représenté sur la figure 2.7.

2.4 Décodage robuste et codage conjoint source-canal

Les CLV et les codes QA sont très performants en termes de compression, mais ils sont comme la plupart des codes entropiques très sensibles au bruit du canal. En effet, les erreurs de transmission sur un train binaire encodé à l'aide d'un CLV ou d'un code QA peuvent provoquer une désynchronisation au niveau du décodeur et des

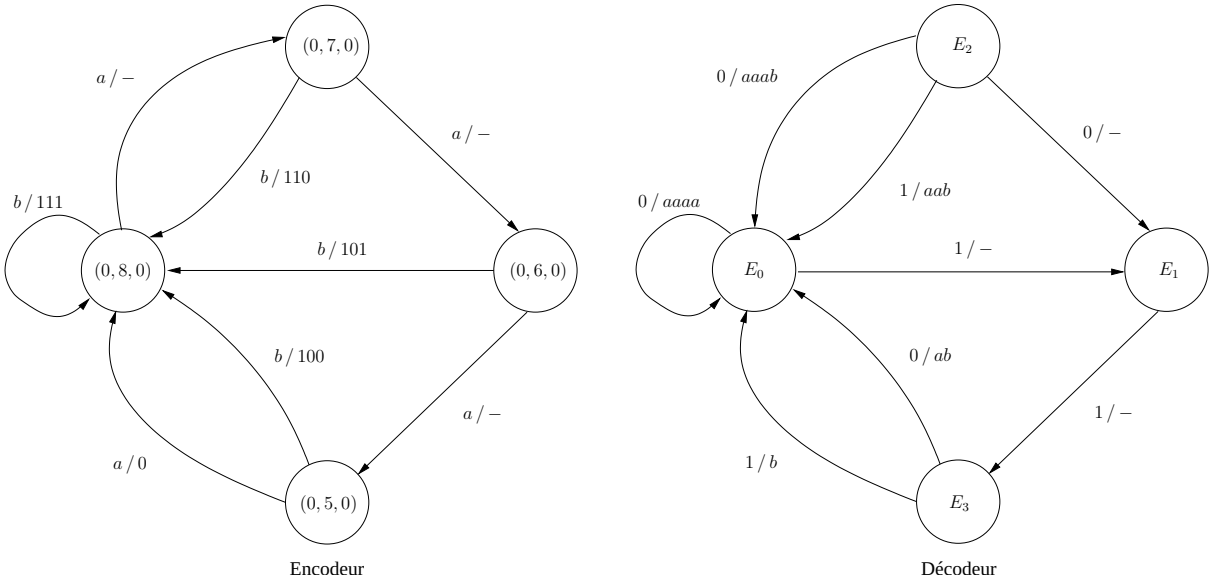

FIG. 2.7 – Automates d'encodage et décodage du code QA $\mathcal{Q}_9$.

taux d'erreur très médiocres. De nombreux auteurs se sont donc penchés sur le problème du décodage robuste et codage/décodage conjoint source-canal de ces deux types de codes. Deux approches différentes ont principalement été traitées dans la littérature. La première approche consiste à exploiter la structure des codes pour développer des algorithmes de décodage souple appliqués sur des treillis. La deuxième approche consiste à insérer volontairement de la redondance dans un train binaire encodé à l'aide d'un VLC ou d'un code QA afin d'améliorer leurs propriétés de resynchronisation et donc leurs performances de décodage. Nous allons passer en revue ces différentes techniques dans cette section.

2.4.1 Décodage souple des CLV et codes QA

De nombreux auteurs ont travaillé sur le décodage souple des CLV et codes QA afin d'exploiter la structure particulière de ces codes et la redondance résiduelle présente dans les trains binaires issus de l'encodage. En 1993, un algorithme a été proposé dans [SB91] pour exploiter le redondance résiduelle pour les CLV. Cet algorithme a été adapté dans [Say99a] et [GG04] aux codes arithmétiques et QA.

Les techniques de décodage souple de CLV ou codes QA sont principalement basées sur des algorithmes d'estimation appliqués sur des treillis. Nous allons maintenant décrire les deux types de treillis principalement utilisés dans la littérature.

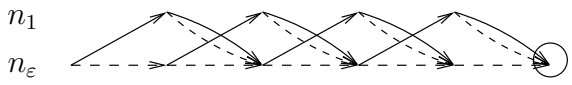

FIG. 2.8 – Treillis bit associé au code $\mathcal{C}_0$

2.4.1.1 Treillis bit

En 1997, Balakirsky a proposé un modèle d'état, appelé treillis bit, pour le décodage souple des CLV. Ce modèle d'état permet de réaliser une estimation bayésienne au niveau bit sur une chaîne de Markov cachée au décodeur, améliorant ainsi les performances de décodage, comparativement au décodage dur. Les états du treillis bit sont les états internes de l'arbre d'encodage d'un CLV. Ce modèle d'état est ainsi défini par la variable aléatoire N_k, représentant l'état interne de l'arbre de codage d'un CLV à chaque instant bit k. Sur ce treillis, les algorithmes d'estimation de type BCJR ou Viterbi peuvent être appliqués afin de réaliser l'estimation de la séquence émise. Lorsque le nombre de bits émis $L(\mathbf{X})$ est connu par le décodeur, ce modèle d'état permet d'intégrer une contrainte de terminaison. En effet, après le décodage du dernier bit, le décodeur doit forcément se trouver dans le noeud racine n_ε de l'arbre du CLV. Lors de l'estimation, tous les chemins dans le treillis qui ne passent pas par l'état n_ε au dernier instant bit ne seront donc pas considérés. Cette contrainte de terminaison est appelée contrainte du noeud racine. Un exemple de treillis bit associé au code $\mathcal{C}_0$ de la figure 2.3 est représenté sur la figure 2.8. Sur cette figure, la contrainte du noeud racine est représentée par un cercle.

En utilisant la représentation sous forme de machines à états finis des codes QA section 2.3.2, le treillis bit peut être adapté aux codes QA. Le modèle d'état pour ce type de codes est alors défini par les états internes de l'automate d'encodage ou de décodage à chaque instant bit k. L'utilisation d'un treillis bit pour les codes QA a été proposée dans [GG04]. Il faut noter que dans le cas des codes QA, la contrainte du noeud racine ne peut pas être appliquée au décodeur. En effet, contrairement au CLV, le décodeur n'est pas forcé de se trouver dans un état particulier après le décodage du dernier bit. L'utilisation d'un treillis pour les codes QA permet donc d'exploiter uniquement la structure du code pour améliorer les performances de décodage.

2.4.1.2 Treillis bit/symbole

Le treillis bit que nous venons de présenter permet d'intégrer une information supplémentaire sur le nombre de bits émis (à condition que celui-ci soit connu). Ce modèle est optimal dans le cas où seule cette information supplémentaire est disponible. Afin de protéger les codes entropiques contre le phénomène de désynchronisation, de nombreux travaux dans la littérature ont traité le cas où le nombre de symboles

émis est également disponible au décodeur [SV98][MF99] [BH00a] [GFG01][PM00]. Si le nombre de symboles est connu, une contrainte de longueur supplémentaire peut être intégrée au décodage. Le treillis bit de Balakirsky ne permet pas d'intégrer une telle contrainte, car le décodeur ne garde pas de traces de l'horloge symbole durant le processus de décodage.

Afin d'exploiter optimalement l'information sur le nombre de symboles émis, un modèle d'état pour les CLV appelé treillis bit/symbole a été proposé dans [BH00a]. Ce modèle intègre les valeurs d'horloge symbole à l'intérieur des états permettant ainsi d'exploiter tout type d'information supplémentaire sur le nombre de symboles. Ce modèle est défini par le couple de variables aléatoires (N_k, T_k), où la variable N_k représente comme pour le treillis bit, le noeud interne de l'arbre du CLV à l'instant bit k, et où T_k représente les valeurs d'horloge symbole possibles à l'instant bit k. Le treillis associé à ce modèle d'état est donc indexé à la fois par l'horloge bit et l'horloge symbole, d'où le nom de treillis bit/symbole. Si l'on suppose que le nombre $L(\mathbf{X})$ de bits transmis ainsi que le nombre $L(\mathbf{S})$ de symboles émis sont connus, l'utilisation du treillis bit/symbole conduit à la contrainte de terminaison suivante :

$$(N_{L(\mathbf{X})}, T_{L(\mathbf{X})}) = (n_\varepsilon, L(\mathbf{S})). \tag{2.2}$$

Cela signifie qu'après le décodage du dernier bit, seules les séquences terminant dans l'état $(n_\varepsilon, L(\mathbf{S}))$ seront considérées, les autres séquences étant écartées par l'algorithme de décodage.

Le treillis bit/symbole associé au code $\mathcal{C}_0$ de la figure 2.3 est représenté sur la figure 2.9. Sur ce treillis, une contrainte de terminaison est représentée par un cercle. Sur cet exemple, toutes les séquences de 5 bits ne correspondant pas à un message de 3 symboles sont écartées par cette contrainte. On remarque sur cette figure que le nombre de valeurs d'horloge symboles possible augmente avec le nombre de bits du message. En fait, à l'instant bit k, l'horloge symbole peut prendre ses valeurs dans l'intervalle entier $[k/l_M, k/l_m]$.

En utilisant la représentation des codes QA sous forme de machines à états finis, un modèle d'état de type bit/symbole peut également être utilisé pour les codes QA. Ce type de modèle a par exemple été utilisé dans [DHS06] et [BJWK06]. Sur ce modèle, la contrainte de longueur est différente de celle disponible sur le modèle des CLV. En effet, après le décodage du dernier bit, l'automate de décodage du code QA peut se trouver dans n'importe quel état. Nous aurons ainsi autant d'états finaux possibles que d'états de l'automate de décodage : ceux dont l'horloge symbole associée est égal au nombre de symboles transmis.

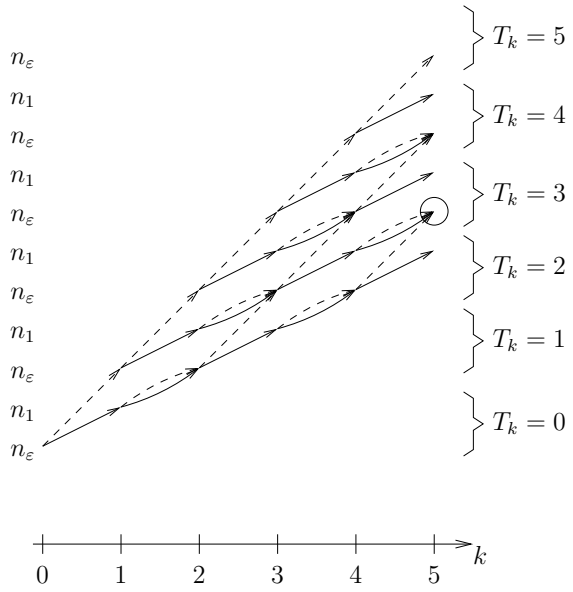

FIG. 2.9 – Treillis bit/symbole associé au code $\mathcal{C}_0$

2.4.1.3 Réduction de complexité

Le treillis bit/symbole est optimal dans le sens où il permet d'exploiter de manière optimale des informations supplémentaires sur l'horloge bit et l'horloge symbole. Cependant, il ne peut être utilisé de manière optimale en pratique à cause de sa complexité prohibitive. En effet, le nombre d'états du treillis bit/symbole est quadratique avec la longueur de la séquence. Ainsi, pour des longueurs de séquence typique, la complexité associée au treillis bit/symbole est trop importante. Pour faire face à ce problème de complexité, de nombreux travaux ont été menés dans la littérature. La plupart des auteurs travaillant sur ce sujet ont développé ou adapté des algorithmes d'estimation sous-optimaux afin de réduire la complexité du décodage. Plusieurs types d'algorithmes sous-optimaux sont utilisés. Dans [MF00] et [LP01], un algorithme de décodage par liste ordonnée de type *Stack Algorithm* [Jel69, Zig66] est considéré pour réduire la complexité du décodage. L'algorithme M [AM84] est également fréquemment utilisé [GMO04, BKK01]. Cet algorithme conserve pour chaque instant bit les M séquences de plus fortes probabilités. Pour calculer les probabilités des séquences, plusieurs métriques différentes peuvent être considérées, la métrique de Fano [Fan63] étant celle le plus souvent utilisée. L'algorithme T [Sim90] est utilisé dans [KT05] associé à un décodage de type *Max-Log-MAP*. L'algorithme T ne garde pour chaque instant bit que les séquences dont la probabilité est supérieure à un seuil T.

Récemment, un algorithme de regroupements de mots de CLV en un nombre minimal de classes a été présenté dans [MKKD05]. Les algorithmes de type BCJR ou Viterbi ont été adaptés pour travailler sur les tables de mots de code réduites. Il est montré que les résultats obtenus en terme de performance de décodage sont équivalents à ceux obtenus sur une table complète.

Dans le chapitre 3 de ce document, nous proposerons un nouveau modèle d'état permettant le décodage souple des CLV et codes QA avec une complexité linéaire avec la longueur de la séquence. Ce modèle d'état est basé sur l'agrégation d'états du treillis bit/symbole. Contrairement aux méthodes de décodage séquentiel, cette solution n'induit pas de sous-optimalité au niveau du décodage et permet d'atteindre les performances optimales du treillis bit/symbole.

2.4.2 Ajout de redondance dans les trains binaires codés entropiquement

De nombreux auteurs se sont penchés sur l'ajout de redondance contrôlé dans les trains binaires issus de CLV ou de codes QA afin de rendre ces codes plus robustes lors de la transmission. On peut différencier deux approches. La première approche consiste à travailler directement sur la structure du code, et la deuxième consiste à travailler sur le train binaire, sans modifier le code en lui-même. Nous allons décrire dans cette section les différentes techniques proposées dans la littérature permettant de rendre plus robustes les CLV et codes QA. Il faut noter que ces techniques, ajoutant de la redondance, ne sont plus optimales en termes de compression, mais elles visent à améliorer les performances en terme de décodage des CLV et codes QA.

2.4.2.1 Codes à longueur variable réversibles

Les CLV classiques (de type Huffman par exemple) sont des codes *prefix-free*, c'est à dire qu'ils sont uniquement décodables lorsqu'on décode le train binaire du début vers la fin. Ces codes ne sont par contre pas *suffix free*. Un code *suffix free* présente la particularité d'être uniquement décodable si l'on décode le train binaire de la fin vers le début.
Les auteurs de [TWM95] ont proposé un algorithme de construction de CLV décodables dans les deux sens. Ces codes admettent deux arbres de codage/décodage : un pour chaque sens de décodage. Ils sont appelés *codes à longueur variable réversible*. La structure particulière de ces codes permet souvent de détecter et de corriger de petites rafales d'erreur. Ceci est dû au fait que ces codes ne sont pas complets, c'est à dire que leur somme de Kraft (équation 1.8) est strictement inférieure à 1. En d'autres termes, il y a des trous dans l'arbre de codage de ces codes (certaines branches de l'arbre ne correspondent pas à une suite de bits valide contrairement à d'autres de la même profondeur à l'intérieur de l'arbre). Lorsqu'une erreur est détectée, le décodage est repris dans le sens inverse, en essayant de corriger le maximum d'erreurs possible.

Les propriétés de correction d'erreurs de ces codes font qu'ils sont largement utilisés dans les systèmes de compression, notamment dans les systèmes itératifs. Ils présentent en effet de bonnes propriétés lorsqu'ils sont mis en série avec un entrelaceur et un code de canal. Des travaux continuent à être menés sur ces types de codes, notamment pour essayer de trouver les meilleurs compromis entre capacité de correction et la quantité de redondance ajoutée [TW01].

2.4.2.2 Codes QA avec symbole interdit

Dans [BCI$^+$97], les auteurs ont proposé de modifier le processus d'encodage d'un code arithmétique en introduisant dans l'intervalle courant un espace réservé à un symbole interdit (SI). Une certaine probabilité ε est associée à ce SI. Ceci revient en fait à ajouter à la source un nouveau symbole de probabilité ε. L'encodage se passe de la même façon que pour un code arithmétique classique, le SI n'est jamais encodé mais une subdivision de l'intervalle courant lui est toujours associée comme pour un symbole normal. Il est montré dans [CR00] que la redondance ajoutée par un SI de probabilité ε est de $-\log(1-\varepsilon)$ bits par symbole. Ainsi, plus l'intervalle réservé au SI est grand, plus la redondance ajoutée est importante.

Pendant le processus de décodage, lorsque l'intervalle sélectionné est à l'intérieur de l'intervalle associé au SI, une erreur est détectée. Il est montré dans [BCI$^+$97] que la probabilité que l'erreur soit détectée n bits au moins après son occurrence est de $(1-\varepsilon)^n$.

Sayir a montré dans [Say99b] qu'en variant l'emplacement de l'intervalle alloué au SI à l'intérieur de l'intervalle courant, les codes arithmétiques avec SI peuvent être vus comme des codes de canal. Un algorithme permettant de calculer les distances libres des codes QA avec SI et les spectres de distance a ensuite été présenté dans [BJWK06]. Dans cet article, les auteurs considèrent également l'éclatement de l'intervalle interdit à l'intérieur de l'intervalle courant et montrent que les propriétés de distance des codes QA avec SI dépendent de la configuration de l'intervalle. Ils proposent également une méthode pour calculer des bornes supérieures sur les taux d'erreur bit et symbole après décodage souples des codes QA avec SI.

De nombreux auteurs se sont également penchés sur le décodage souple de codes arithmétiques ou QA afin de donner à ces codes des capacités de correction d'erreurs. Grangetto *et al.* ont utilisé de méthodes de décodage séquentiel au sens du Maximum a Posteriori comme le Stack Algorithm ou l'algorithme M dans [GMO04, GCO05]. Les auteurs de [DHS06] ont proposé un modèle de treillis en 3 dimensions en s'appuyant sur la représentation des codes QA en machines à états finis pour le décodage souple de ces codes. Ils appliquent sur ce treillis un algorithme de type *List-Viterbi*.

2.4.2.3 Marqueurs de synchronisation

Afin d'améliorer la synchronisation tout au long de la séquence, des marqueurs de synchronisation peuvent être ajoutés dans le train binaire. Ces marqueurs de synchronisation se présentent souvent sous la forme de symboles de l'alphabet et ils sont placés à des positions connues de l'horloge symbole. Différentes techniques ont été proposées dans [BCI+97, DHS01, Elm99, SCW00, GG04] notamment. On peut notamment répéter le dernier symbole encodé ou ajouter un symbole choisi de l'alphabet aux positions choisies initialement. Ces marqueurs favorisent ainsi les séquences synchronisées en plusieurs endroits du processus de décodage, et non uniquement à la fin dans le cas où seule une contrainte de terminaison est disponible.

2.5 Conclusion

Cette partie a présenté tout d'abord les deux types de codes entropiques qui seront utilisés dans ce document : les CLV (en particulier de type Huffman), ainsi que les codes arithmétiques et QA. Puis, un état de l'art en décodage robuste et codage/décodage conjoint source-canal a été présenté, en se focalisant sur les aspects qui nous intéresseront dans ce document : décodage souple, complexité des modèles, et ajout d'information adjacente. Dans le chapitre suivant, un modèle d'état pour le décodage souple des CLV est présenté. Ce modèle d'état, introduit dans [JMG05] pour le cas des CLV, est ensuite étendu au cas des codes QA. Nous verrons que ce modèle permet d'atteindre les performances du treillis bit/symbole pour une complexité linéaire avec la longueur de la séquence. Nous présenterons également un algorithme permettant de calculer les probabilités marginales au niveau symbole sur ce modèle, ainsi qu'une technique d'ajout d'information adjacente basée sur l'introduction de contraintes de terminaison.

Chapitre 3

Modèle d'état à complexité réduite pour le décodage souple des CLV et codes QA

"Mais tout est histoires Charles, absolument tout et pour tout le monde. Le problème, c'est qu'on ne trouve jamais personne pour les raconter"

Anna Gavalda

Dans le chapitre 2, les deux modèles d'état principalement utilisés pour le décodage souple des CLV et codes QA ont été présentés. Le premier modèle est appelé treillis bit. Il a été introduit par Balakirsky dans [Bal97] pour les CLV. Il s'applique également aux codes QA [GG04]. Ce modèle est uniquement composé des états internes du décodeur (noeuds internes de l'arbre pour les CLV et états de l'automate de décodage pour les codes QA). Ainsi, on ne peut pas intégrer, sur ce modèle, d'informations supplémentaires sur l'horloge symbole (comme le nombre total de symboles du message par exemple). Dans le cas où une contrainte de terminaison sur le nombre de symboles est disponible au décodeur, le modèle d'état appelé treillis bit/symbole est plus adapté pour prendre en compte cette information. Ce modèle d'état a été introduit dans [BH00b] pour les CLV et dans [GG04] pour les codes QA. L'horloge symbole est intégrée dans ce modèle d'état afin d'utiliser de façon optimale d'éventuelles informations au niveau symbole. Nous détaillerons ce modèle d'état dans ce chapitre et nous verrons principalement que sa complexité est un obstacle à son utilisation pratique. Des méthodes permettant de réduire la complexité du décodage ont été présentées dans le chapitre 2. La plupart de ces méthodes sont sous-optimales, c.a.d. que les performances de décodage de ces méthodes sont inférieures à celles obtenues sur le treillis

bit/symbole.

Un nouveau modèle d'état pour le décodage souple des CLV et codes QA est présenté dans ce chapitre dans la section 3.1. La complexité associée à ce modèle est linéaire avec la longueur de la séquence. Pour obtenir ce modèle, les états du treillis bit/symbole distants de T valeurs d'horloge symbole sont agrégés. L'entier T est appelé paramètre d'agrégation. Nous verrons que le paramètre d'agrégation permet de doser un compromis entre la complexité du décodage sur le modèle agrégé et les performances associées. Une analyse de performance sur ce modèle, basée sur l'étude de resynchronisation des codes, sera présentée dans le chapitre 4. Un algorithme, appelé algorithme de décodage combiné multi-treillis, permettant une réduction supplémentaire de complexité est présenté dans la section 3.2. Cet algorithme permet d'atteindre les performances d'un treillis de paramètre $T_1 \times T_2$, en utilisant deux treillis de paramètre T_1 et T_2 à condition que T_1 et T_2 soient premiers entre eux.

Les algorithmes de Viterbi [Vit67] et BCJR [BCJR74] peuvent être appliqués sur le modèle agrégé permettant respectivement la minimisation du taux d'erreur séquence et du taux d'erreur bit. Cependant, les probabilités marginales au niveau symbole ne peuvent pas être directement calculée sur le modèle agrégé. Cette probabilité marginale est nécessaire dans le cas où l'on souhaite minimiser le taux d'erreur symbole en sortie du décodeur. Elle est également nécessaire lorsqu'une structure itérative échangeant des probabilités au niveau symbole est appliquée au décodeur, comme dans [KT02] par exemple. Nous présenterons un algorithme dans la section 3.3 qui permet de calculer une marginale symbole sur le modèle agrégé [MJG08a].

Enfin, une solution permettant d'introduire de la redondance contrôlée sera présentée dans la section 3.4. Cette redondance se présente sous la forme de contraintes de longueur. Cette technique est une solution alternative aux solutions classiques d'ajout de redondance qui ont été présentées dans le chapitre 2. Nous verrons que cette solution présente l'avantage d'être très flexible et qu'elle ne modifie pas le train binaire original, contrairement à la solution à base de marqueurs de synchronisation par exemple. La solution proposée sera comparée à des solutions classiques d'ajout de redondance dans des trains binaires à longueur variable.

3.1 Agrégation d'états pour le décodage souple des CLV et codes QA

3.1.1 Préliminaires

Les deux types de modèles d'états principalement utilisés pour le décodage souple des CLV ou codes QA ont été présentés dans le chapitre 2. Sur le premier modèle, appelé treillis bit, l'horloge symbole n'est pas intégrée dans les états. Ce modèle ne peut ainsi

pas intégrer d'information au niveau symbole, comme des contraintes de longueur par exemple. Dans le second modèle, appelé treillis bit/symbole, les valeurs d'horloge symbole possibles à chaque instant bit sont intégrées. Ainsi, en présence d'une contrainte de terminaison sur le nombre de symboles émis, ce modèle est optimal en terme de performance de décodage. Cependant, le nombre d'états de ce modèle est quadratique avec le nombre de symboles émis, limitant son utilisation pratique.

Sur la figure 3.1, les probabilités des états du treillis bit/symbole en sortie de l'algorithme BCJR sont représentées de manière simplifiée. L'horloge bit est représentée en abscisse, et l'horloge symbole en ordonnée. Plus la probabilité d'un état est forte, plus claire est sa représentation sur la figure. Ces probabilités ont été obtenues pour le code $\mathcal{C}_{10}$ et un rapport signal à bruit sur le canal $E_b/N_0 = 0$ dB lors d'une simple transmission d'un message de $L(\mathbf{S}) = 100$ symboles. On peut remarquer que, à chaque instant bit, seul un intervalle réduit de valeurs d'horloge symbole comporte une probabilité non nulle. La taille de cet intervalle dépend de l'instant bit considéré, du code utilisé et du rapport signal à bruit lors de la transmission. La figure 3.2 représente les probabilités de ces mêmes états lorsque le rapport signal à bruit est égal à 2 dB. On voit alors que lorsque le rapport signal à bruit augmente, la taille de l'intervalle de probabilités non nulles diminue.

En regardant ces figures, il paraît envisageable d'effectuer un regroupement d'états à chaque instant bit, sans altérer les performances en sortie de l'algorithme de décodage. Si l'on regroupe en effet chaque état dont la probabilité est non nulle, avec un ou plusieurs autres état(s) dont la probabilité est nulle, l'estimation en sortie de l'algorithme de décodage ne sera pas modifiée.

Sur la figure 3.5, un tel regroupement d'états a été réalisé. Pour obtenir cette figure, les 4 "tranches" de la figure 3.3 ont été superposées. On peut remarquer qu'en faisant cette superposition d'états, aucune ambiguïté n'a été ajoutée. En effet, chaque nouvel état comporte un seul état du treillis original dont la probabilité est strictement supérieure à 0.

En superposant les tranches du treillis de la figure 3.4, on obtient le treillis de la figure 3.6. Cette fois-ci, l'agrégation d'états est trop importante pour ne pas induire d'ambiguïté au niveau du décodeur. On a regroupé des états de probabilités non nulles.

En analysant ces figures, il paraît donc envisageable de construire un modèle agrégé en regroupant "intelligemment" des états du treillis bit/symbole. Le modèle agrégé proposé est décrit de façon mathématique dans la suite de cette section.

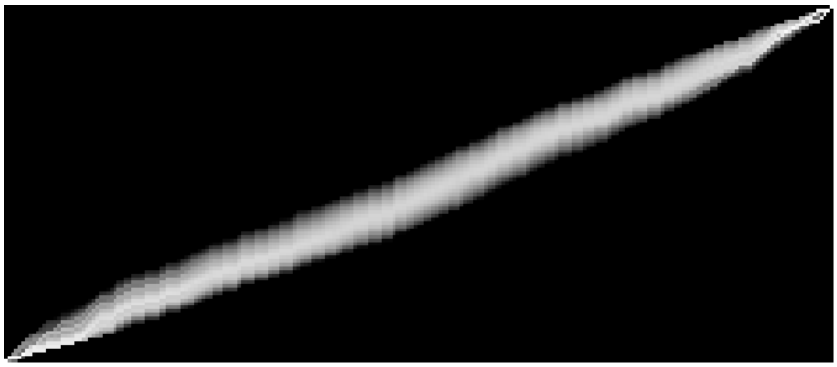

FIG. 3.1 – Probabilités des états sur le treillis bit/symbole pour le code $\mathcal{C}_{10}$ et $E_b/N_0 = 0$ dB.

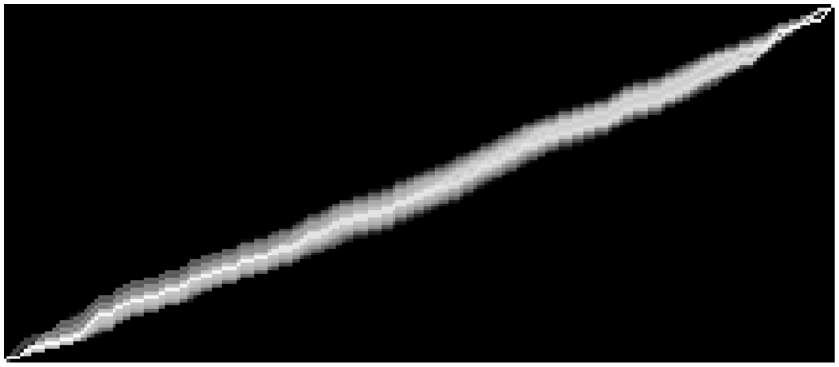

FIG. 3.2 – Probabilités des états sur le treillis bit/symbole pour le code $\mathcal{C}_{10}$ et $E_b/N_0 = 2$ dB.

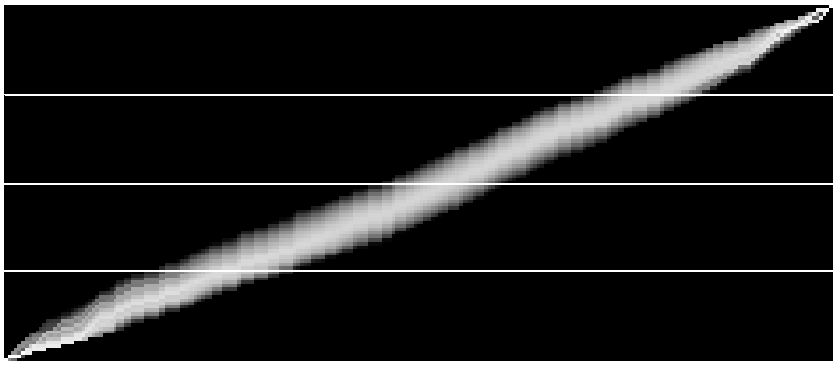

FIG. 3.3 – Probabilités des états sur le treillis bit/symbole pour le code $\mathcal{C}_{10}$ et $E_b/N_0 = 0$ dB.

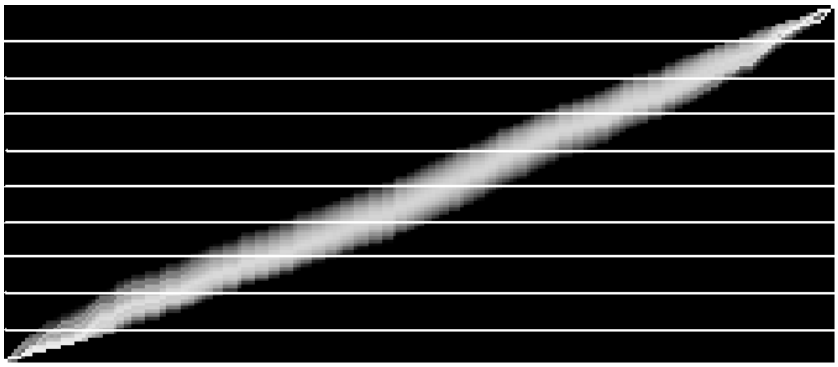

FIG. 3.4 – Probabilités des états sur le treillis bit/symbole pour le code $\mathcal{C}_{10}$ et $E_b/N_0 = 2$ dB.

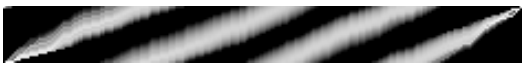

FIG. 3.5 – Probabilités des états sur le treillis agrégé pour le code $\mathcal{C}_{10}$, $E_b/N_0 = 0$ dB et $T = 25$.

FIG. 3.6 – Probabilités des états sur le treillis agrégé pour le code $\mathcal{C}_{10}$, $E_b/N_0 = 0$ dB et $T = 10$.

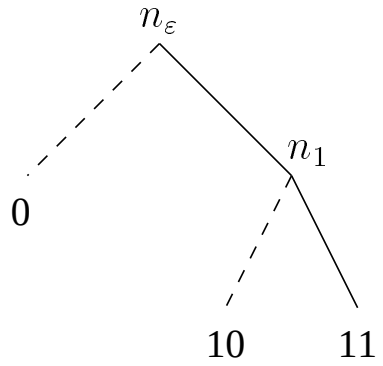

FIG. 3.7 – Code $\mathcal{C}_0$

3.1.2 Présentation du modèle agrégé

Soit le modèle d'état défini par les couples de variables aléatoires (N_k, M_k), où N_k représente l'état du décodeur à l'instant bit k (état interne de l'arbre pour un CLV et état de l'automate de décodage pour un code QA), et où M_k est égal à T_k mod T, soit le reste de la division euclidienne de T_k (variable aléatoire représentant les valeurs d'horloge possibles) par un paramètre entier T ($M_k = T_k$ mod T). L'entier T est appelé paramètre d'agrégation. On peut noter que si $T = 1$, le modèle agrégé résultant est équivalent au treillis bit, alors que si $T > L(\mathbf{S})$, le modèle résultant est équivalent au treillis bit/symbole. Pour les valeurs intermédiaires de T, on peut légitimement s'attendre à un compromis entre complexité et performances de décodage. En effet, le nombre d'états de ce modèle est linéaire avec T et il semble intuitif de penser que les performances de décodage sur ce modèle s'améliorent lorsque T augmente. Nous verrons néanmoins par la suite que ce n'est pas toujours le cas.

Exemple 3.1: Considérons le CLV de la figure 3.7. Ce code est adapté à un alphabet de 3 symboles. L'automate d'encodage/décodage obtenu pour $T = 1$ est représenté en haut à gauche de la figure 3.8. Le treillis ainsi obtenu est représenté en haut à droite de cette même figure. Ce treillis est équivalent au treillis bit de [Bal97]. L'automate d'encodage/décodage et le treillis associé obtenu pour $T = 2$ sont représentés en bas de la figure 3.8.

Les probabilités de transitions sur le modèle agrégé sont calculées de la façon suivante. Soient deux états β_i et β_j de l'automate de décodage du code considéré, et soient m_k et m_{k-1} deux entiers appartenant à l'intervalle $\Gamma_T = \{0, \ldots, T-1\}$. On appelle $\sigma_{i,j}$ le

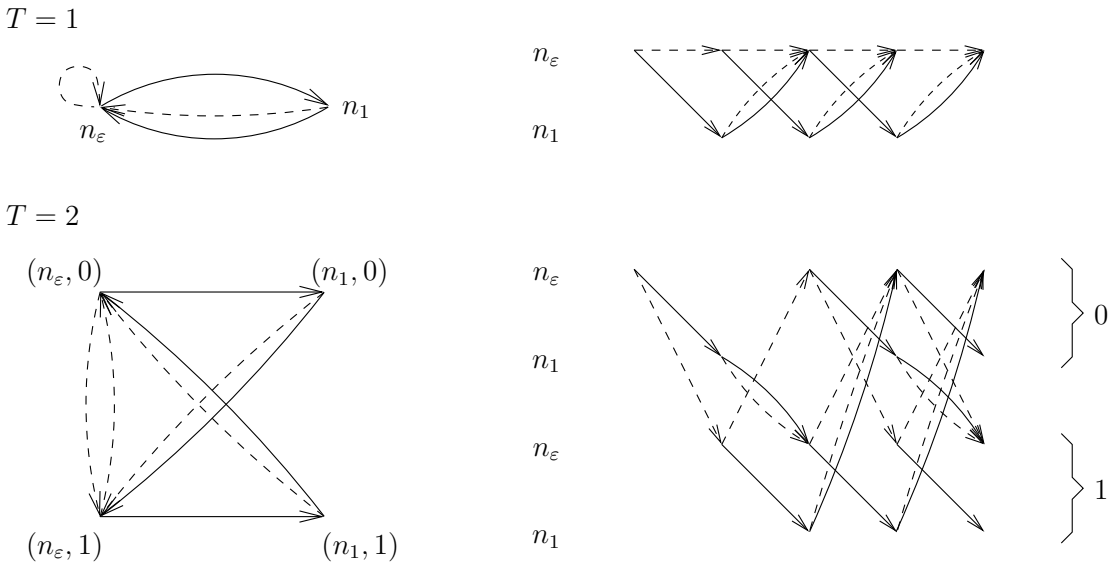

FIG. 3.8 – Treillis agrégé obtenu pour le code $\mathcal{C}_0$ et $T = 1$.

nombre de symboles émis par la transition entre β_i et $beta_j$. On a alors :

$$\begin{aligned} &\mathbb{P}(N_k = \beta_i, M_k = m_k \,|\, N_{k-1} = \beta_j, M_{k-1} = m_{k-1}) = \\ &\begin{cases} \mathbb{P}(N_k = \beta_i \,|\, N_{k-1} = \beta_j) & \text{si } m_k = (m_{k-1} + \sigma_{i,j}) \bmod T \\ 0 & \text{sinon,} \end{cases} \end{aligned} \tag{3.1}$$

où les probabilités $\mathbb{P}(N_k = \beta_i \,|\, N_{k-1} = \beta_j)$ sont déduites de la statistique de la source et de la structure du code considéré.

Sur ce modèle d'état agrégé, les algorithmes classiques d'estimation tels que l'algorithme de Viterbi [Vit67] ou de type BCJR [BCJR74] peuvent être appliqués en utilisant les probabilités de transition de l'équation 3.1. Cependant, il n'est pas possible d'obtenir directement, en utilisant l'algorithme BCJR, la loi de probabilité marginale au niveau symbole en sortie du décodeur. En effet, les valeurs complètes d'horloge symbole ne sont pas inclues dans le modèle agrégé. Nous verrons dans la section 3.3 comment calculer une telle marginale en utilisant le modèle agrégé. Deux mesures peuvent ainsi être minimisées sur le modèle agrégé : le TESQ en utilisant l'algorithme de Viterbi, et le TEB en utilisant l'algorithme BCJR au niveau bit.

Ce modèle d'état agrégé contient les valeurs d'horloge modulo le paramètre T. Cette information peut être exploitée à l'aide d'une contrainte de terminaison. Si l'on suppose que le décodeur connaît la valeur $L(\mathbf{S}) \bmod T$, une contrainte de terminaison peut être définie sur ce modèle. Toutes les séquences qui ne terminent pas dans un état tel que $M_{L(\mathbf{X})} = L(\mathbf{S}) \bmod T$ sont alors écartées par l'algorithme de décodage. Cette contrainte de terminaison est plus faible que celle qui est disponible sur le mo-

dèle optimal de type bit/symbole. Cependant, nous verrons dans le chapitre 4, qu'en choisissant de manière idoine le paramètre T, l'agrégation d'état n'altère pas la quantité d'information apportée par la contrainte de terminaison.

3.1.3 Analyse de complexité du modèle agrégé

Le nombre d'états ν_T du modèle agrégé de paramètre T vérifie

$$\nu_T \leq T \times L(\mathbf{X}) \times \text{card}(\Omega_d), \tag{3.2}$$

où Ω_d est l'ensemble des états de l'automate de décodage du code considéré, et $L(\mathbf{X})$ le nombre de bits transmis. L'inégalité dans l'équation (3.2) vient du fait que, pour certains codes, des paires (n_k, m_k) ne sont pas accessibles d'après la structure du code. De tels états sont situés la plupart du temps aux premiers et derniers instants bits sur le treillis. Pour estimer la complexité du décodage sur le modèle agrégé, on considère le cas le moins favorable, c'est à dire que l'on suppose que le nombre d'états sur le treillis vérifie

$$\nu_T \approx T \times L(\mathbf{X}) \times \text{card}(\Omega_d). \tag{3.3}$$

Ainsi, comme le nombre d'états sur le treillis bit est égal à $L(\mathbf{X}) \times \text{card}(\Omega_d)$, la complexité D_T du décodage sur le modèle agrégé peut s'écrire

$$D_T \approx T \times D_{\text{bal}}, \tag{3.4}$$

où D_{bal} représente la complexité du décodage sur le modèle bit de [Bal97]. La complexité du décodage sur le modèle agrégé est donc linéaire avec la longueur de la séquence et avec le paramètre T. Rappelons que cette complexité est quadratique avec la longueur de la séquence sur le modèle bit/symbole.

3.1.4 Résultats de simulation

Le modèle agrégé présenté dans cette section a été utilisé en simulation dans un schéma classique de transmission. Le train binaire issu de l'encodage du message source est transmis sur un canal gaussien de rapport signal à bruit E_b/N_0 donné. Au décodeur, l'algorithme de Viterbi ou l'algorithme BCJR est appliqué afin d'estimer la séquence originale.

Le TES du code $\mathcal{C}_5$, après le décodage souple à l'aide de l'algorithme de Viterbi, est représenté en fonction de E_b/N_0 sur la figure 3.9 pour différentes valeurs du paramètre d'agrégation. Ces résultats ont été obtenus pour des séquences de $L(\mathbf{S}) = 100$ symboles en moyennant le TES sur 10^5 réalisations conjointes de la source et du canal. On peut remarquer que les performances de ce code s'améliorent lorsque T augmente. De plus, pour $T = 10$ le TES est égal à celui obtenu pour $T = 100$ (treillis bit/symbole) pour $E_b/N_0 \geq 2$ dB. Cela signifie que les performances du treillis optimal ont été obtenues avec une réduction de complexité significative. Pour $E_b/N_0 < 2$, le paramètre

T doit être augmenté pour atteindre les performances optimales.

Le tableau 3.1 contient le TESQ des codes $\mathcal{C}_5$, $\mathcal{C}_7$, $\mathcal{C}_{10}$ et $\mathcal{C}_{13}$ pour les mêmes conditions de simulation que celles de la figure 3.9. On peut également remarquer grâce à ce tableau que les performances du treillis bit/symbole sont obtenues pour des valeurs de T bien inférieures à 100. Cependant, la valeur optimale de T dépend du code utilisé et du rapport signal à bruit du canal. Il est également intéressant de remarquer le comportement des performances du code $\mathcal{C}_{13}$ vis-à-vis du paramètre T. Pour ce code, les performances ne sont pas monotoniquement croissantes avec T. Par exemple, pour $E_b/N_0 = 5$ dB, le TESQ pour $T = 4$ est supérieur à celui obtenu pour $T = 3$, ce qui semble contre-intuitif. De plus, pour ce même code, les performances obtenues pour $T = 2$ sont strictement les mêmes que celles obtenues pour $T = 1$.

Les mêmes simulations ont été réalisées pour les codes QA $\mathcal{Q}_7$ et $\mathcal{Q}_9$. La figure 3.10 représente les performances du code $\mathcal{Q}_7$ en fonction du rapport signal à bruit pour différentes valeurs de T. Ces résultats sont consignés dans le tableau 3.2 avec les performances du code $\mathcal{Q}_9$. Les mêmes observations que pour les CLV peuvent être faites : la valeur de T qui permet d'atteindre les performances du treillis optimal varie en fonction du code et du rapport signal à bruit. On peut également à nouveau remarquer des comportements spéciaux pour certaines valeurs de T : les performances du code $\mathcal{Q}_7$ pour $T = 2q, q \in \mathbb{N}$ sont moins bonnes que celles pour $T = 2q - 1$, et les performances du code $\mathcal{Q}_9$ pour $T = 4$ et $T = 8$ sont moins bonnes respectivement que celles obtenues pour $T = 3$ et $T = 7$.

Un nouveau modèle d'état pouvant s'appliquer aux CLV et codes QA a été présenté dans cette section. Ce modèle d'état présente l'avantage d'avoir une complexité linéaire au décodage en fonction de la longueur de la séquence, contrairement au modèle bit/symbole optimal. Nous avons remarqué sur des résultats de simulation que ce modèle d'état permettait d'atteindre les performances du modèle optimal à partir d'une certaine valeur de T, fonction du code et du rapport signal à bruit. Nous avons également remarqué que les performances de certans codes ne sont pas croissantes avec T, ce qui est assez contre-intuitif.

Dans le chapitre 4, nous utiliserons les propriétés de resynchronisation des CLV et codes QA afin de déterminer des outils permettant d'analyser les performances de ces codes sur le modèle agrégé. Nous verrons particulièrement qu'il est possible d'estimer la valeur de T permettant d'approcher les performances optimales, et qu'il est possible de prévoir les variations des performances de décodage d'un code en fonction du paramètre d'agrégation T. Cette analyse sera détaillée dans le chapitre 4.

Dans la prochaine section, un algorithme permettant de réduire davantage la complexité du décodage sur le modèle agrégé, sans perte de performances, est présenté.

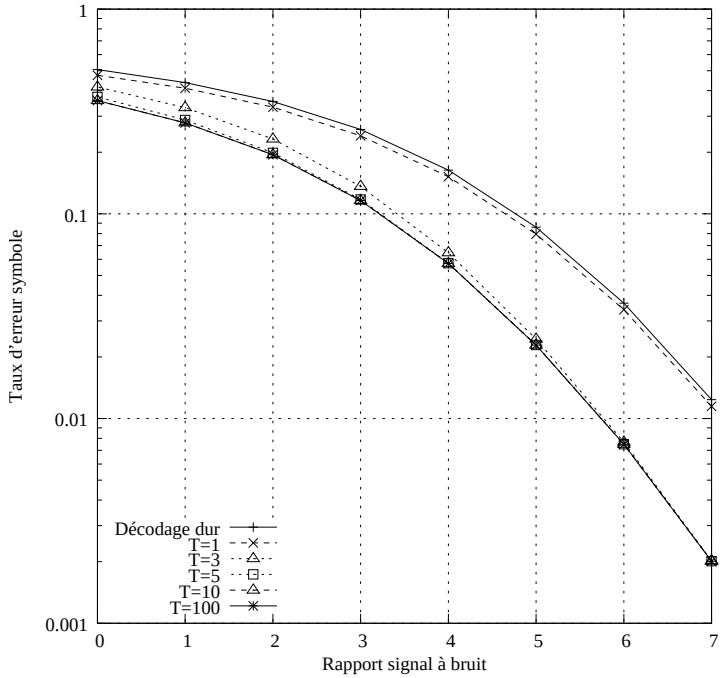

FIG. 3.9 – Taux d'erreur symbole en fonction du rapport signal à bruit pour le code $\mathcal{C}_5$ et différents paramètres d'agrégation.

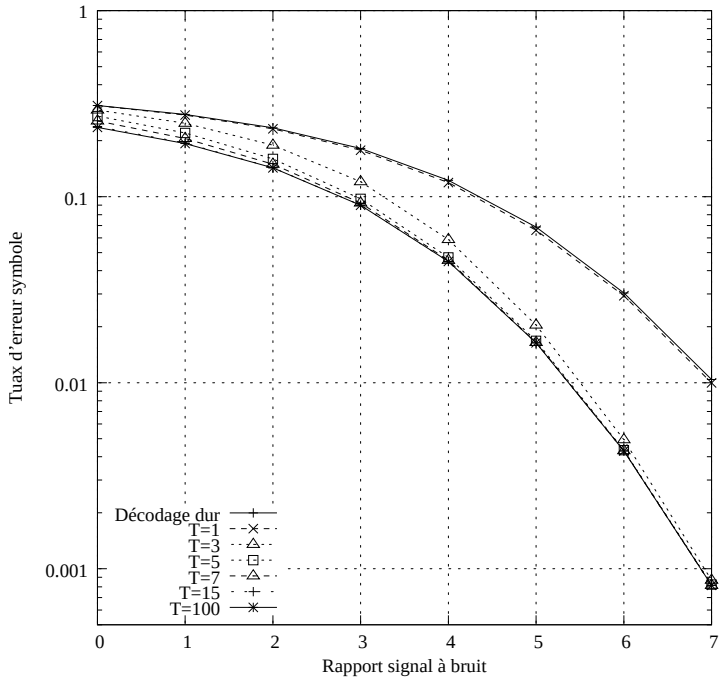

FIG. 3.10 – Taux d'erreur symbole en fonction du rapport signal à bruit pour le code $\mathcal{Q}_7$ et différents paramètres d'agrégation.

E_b/N_0	3	4	5	6	7
			Code $\mathcal{C}_5$		
$T=1$	0.99120	0.92330	0.70464	0.38774	0.14558
$T=2$	0.98805	0.90368	0.66193	0.34633	0.12452
$T=3$	0.98698	0.89901	0.65527	0.34313	0.12388
$T=4$	0.98665	0.89795	0.65457	0.34298	*0.12386*
$T=5$	0.98652	0.89782	0.65449	*0.34296*	
$T=10$	*0.98651*	*0.89780*	*0.65448*		
Bit/symbol($T=100$)	0.98651	0.89780	0.65448	0.34296	0.12386
			Code $\mathcal{C}_7$		
$T=1$	0.99182	0.92604	0.71405	0.39372	0.14885
$T=2$	0.98634	0.88506	0.59864	0.25742	0.06997
$T=3$	0.98247	0.86379	0.55406	0.22571	0.06152
$T=4$	0.98005	0.85387	0.53964	0.21947	0.06059
$T=5$	0.97893	0.84960	0.53581	0.21866	*0.06057*
$T=10$	0.97773	*0.84731*	*0.53468*	*0.21849*	
$T=20$	*0.97772*				
Bit/symbol($T=100$)	0.97772	0.84731	0.53468	0.21849	0.06057
			Code $\mathcal{C}_{10}$		
$T=1$	0.97993	0.87316	0.61783	0.31353	0.11390
$T=2$	0.96917	0.82122	0.51758	0.22232	0.06832
$T=3$	0.96092	0.78516	0.46126	0.18023	0.05207
$T=4$	0.95331	0.75512	0.41127	0.14437	0.03718
$T=5$	0.94755	0.73502	0.38403	0.12851	0.03226
$T=10$	0.93238	0.68744	0.33174	0.10496	0.02631
$T=20$	0.92801	0.67825	0.32560	*0.10354*	*0.02610*
$T=30$	*0.92791*	*0.67811*	*0.32558*		
Bit/symbol($T=100$)	0.92791	0.67811	0.32558	0.10354	0.02610
			Code $\mathcal{C}_{13}$		
$T=1$	0.98973	0.91752	0.69351	0.38031	0.14431
$T=2$	0.98973	0.91752	0.69351	0.38031	0.14431
$T=3$	0.98369	0.88547	0.62816	0.32182	0.11644
$T=4$	0.98552	0.89259	0.63858	0.32711	0.11762
$T=5$	0.98286	0.88356	0.62642	*0.32142*	*0.11638*
$T=10$	0.98286	0.88356	0.62642		
$T=20$	*0.98277*	*0.88348*	*0.62638*		
Bit/symbol($T=100$)	0.98277	0.88348	0.62638	0.32142	0.11638

TAB. 3.1 – Taux d'erreur séquence après décodage souple (Viterbi) sur modèle agrégé avec différents paramètres d'agrégation pour les codes $\mathcal{C}_5$, $\mathcal{C}_7$, $\mathcal{C}_{10}$ et $\mathcal{C}_{13}$.

E_b/N_0	3	4	5	6	7
			Code $\mathcal{Q}_7$		
$T = 1$	0.177725	0.119088	0.066019	0.0293660	0.0099844
$T = 2$	0.177660	0.119029	0.065977	0.0293483	0.0099790
$T = 3$	0.120391	0.058873	0.020400	0.0049425	0.0008687
$T = 4$	0.145964	0.080466	0.031784	0.0085192	0.0014814
$T = 5$	0.097285	0.047133	0.016785	0.0043368	0.0008219
$T = 6$	0.117477	0.057001	0.019686	0.0047921	0.0008442
$T = 7$	0.092465	0.045490	0.016468	0.0043100	*0.0008151*
$T = 8$	0.101290	0.048256	0.016978	0.0043348	*0.0008151*
$T = 9$	0.090876	0.045029	0.016414	*0.0042998*	
$T = 13$	0.090095	0.044894	*0.016394*		
$T = 15$	0.089984	*0.044882*			
$T = 21$	*0.089963*				
$T = 100$	0.089963	0.044882	0.016394	0.0042998	0.0008151
			Code $\mathcal{Q}_9$		
$T = 1$	0.089340	0.056646	0.029854	0.0127630	0.0042457
$T = 2$	0.078041	0.045343	0.021652	0.0083708	0.0026533
$T = 3$	0.067114	0.032231	0.011263	0.0027026	0.0004697
$T = 4$	0.064223	0.032616	0.013335	0.0045597	0.0013501
$T = 5$	0.055957	0.023517	0.006744	0.0013292	0.0001833
$T = 6$	0.050188	0.020412	0.005796	0.0011347	0.0001552
$T = 7$	0.049695	0.020945	0.006313	0.0014643	0.0002736
$T = 8$	0.054915	0.027628	0.011621	0.0041529	0.0012898
$T = 9$	0.049107	0.020842	0.006527	0.0014290	0.0002403
$T = 15$	0.032811	0.011623	0.002845	0.0005035	*0.00005259*
$T = 20$	0.025392	0.009089	0.002322	*0.0004284*	
$T = 35$	0.024457	0.008826	*0.0022843*		
$T = 41$	0.023951	*0.008752*			
$T = 46$	*0.023576*				
$T = 100$	0.023576	0.008752	0.0022843	0.0004284	0.00005259

TAB. 3.2 – Taux d'erreur symbole après décodage souple (Viterbi) sur modèle agrégé avec différents paramètres d'agrégation pour les codes $\mathcal{Q}_7$ et $\mathcal{Q}_9$.

3.2 Décodage combiné multi-treillis

3.2.1 Motivation

Dans cette section, un algorithme permettant une réduction de complexité supplémentaire sur le modèle agrégé sans altérer les performances de décodage est décrit. L'optimalité de cette approche est prouvée pour le taux d'erreur séquence. L'algorithme proposé est basé sur la propriété mathématique suivante :

Propriété 3.1 *Soient T_1 et T_2 deux entiers premiers entre eux. Alors,*

$$\begin{aligned} & L(\mathbf{S}) \mod (T_1 \times T_2) = m \\ \Leftrightarrow & \begin{cases} L(\mathbf{S}) \mod T_1 = m \mod T_1 \\ L(\mathbf{S}) \mod T_2 = m \mod T_2. \end{cases} \end{aligned} \tag{3.5}$$

Cette propriété va servir à démontrer la propriété suivante :

Propriété 3.2 *Soient T_1 et T_2 deux entiers premiers entre eux et $T_3 \triangleq T_1 \times T_2$. On note $\hat{\mathbf{S}}_1$,$\hat{\mathbf{S}}_2$ et $\hat{\mathbf{S}}_3$ les estimations de $\mathbf{S}$ fournies par l'algorithme de Viterbi sur les treillis de paramètres T_1, T_2 et T_3 respectivement. On a alors :*

$$\hat{\mathbf{S}}_1 = \hat{\mathbf{S}}_2 \Rightarrow \hat{\mathbf{S}}_3 = \hat{\mathbf{S}}_1 = \hat{\mathbf{S}}_2. \tag{3.6}$$

Preuve: Il faut souligner, pour commencer, que la probabilité a posteriori d'une séquence calculée par l'algorithme de Viterbi sur un treillis de paramètre T ne dépend pas de T. On supposera que si deux séquences ont la même probabilité a posteriori, une règle subsidiaire est appliquée pour sélectionner une des deux séquences. L'ordre lexicographique peut par exemple être choisi comme règle. Cette hypothèse permet d'assurer que le comportement de l'algorithme de Viterbi est déterministe vis à vis de la séquence estimée. On appelle $\mathcal{S}_T$ l'ensemble :

$$\mathcal{S}_T \triangleq \{s' \mid L(s') \bmod T = L(\mathbf{S}) \bmod T\}. \tag{3.7}$$

$\mathcal{S}_T$ est composé de toutes les séquences qui vérifient la contrainte de terminaison sur le treillis de paramètre T. D'après (3.5), si $T_3 = T_1 \times T_2$ avec T_1 et T_2 deux entiers premiers entre eux, on peut déduire que

$$\mathcal{S}_{T_3} = \mathcal{S}_{T_1} \cap \mathcal{S}_{T_2}, \tag{3.8}$$

ce qui signifie également que

$$\mathcal{S}_{T_3} \subseteq \mathcal{S}_{T_1}. \tag{3.9}$$

De plus, d'après l'hypothèse $\hat{\mathbf{S}}_1 = \hat{\mathbf{S}}_2$, on obtient

$$\hat{\mathbf{S}}_1 \in \mathcal{S}_{T_3}. \tag{3.10}$$

La séquence $\hat{\mathbf{S}}_1$ estimée par l'algorithme de Viterbi sur le treillis de paramètre T_1 vérifie

$$\hat{\mathbf{S}}_1 = \arg \max_{s' \in \mathcal{S}_{T_1}} \mathbb{P}(s' \,|\, \mathbf{X}) \tag{3.11}$$

$$= \arg \max_{s' \in \mathcal{S}_{T_3}} \mathbb{P}(s' \,|\, \mathbf{X}) \tag{3.12}$$

$$= \hat{\mathbf{S}}_3, \tag{3.13}$$

la règle subsidiaire pouvant être appliquée pour la sélection du maximum a posteriori. ■

La propriété 3.2 signifie que si une même séquence est sélectionnée par l'algorithme de Viterbi sur les treillis T_1 et T_2, alors cette séquence sera également celle estimée sur le treillis de paramètre $T_3 = T_1 \times T_2$.

3.2.2 Description de l'algorithme

Nous allons maintenant décrire un algorithme appelé algorithme de décodage combiné multi-treillis. Cet algorithme exploite le résultat de la propriété 3.2 afin de réduire la complexité du décodage pour un niveau de performance égal à celui obtenu sur le treillis de paramètre $T_1 \times T_2$. Le principe de cet algorithme est d'utiliser deux treillis de paramètres T_1 et T_2 au lieu du treillis de paramètre $T_1 \times T_2$. Dans ce qui suit, nous supposerons que les entiers T_1 et T_2 sont premiers entre eux. Le décodage d'une séquence se déroule comme suit :

1. On applique l'algorithme de Viterbi sur les treillis de paramètre T_1 et T_2. Ils fournissent respectivement les estimées $\hat{\mathbf{S}}_1$ et $\hat{\mathbf{S}}_2$.
2. Si $\hat{\mathbf{S}}_1 = \hat{\mathbf{S}}_2$, $\hat{\mathbf{S}}_1$ est utilisée comme estimée de la séquence émise $\mathbf{S}$.
3. Sinon, on applique l'algorithme de Viterbi sur le treillis de paramètre $T_1 \times T_2$.

D'après la propriété 3.2, si une même séquence est sélectionnée sur les treillis T_1 et T_2, cette séquence est aussi sélectionnée sur le treillis $T_1 \times T_2$. Ainsi, les performances en terme de décodage de l'algorithme décrit ci-dessus seront exactement les mêmes que les performances sur un treillis de paramètre $T_1 \times T_2$.

3.2.3 Complexité moyenne de l'algorithme de décodage combiné multi-treillis

Rappelons d'abord que si $T = 1$, le treillis agrégé résultant est équivalent au treillis bit de [Bal97]. Si $T \geq L(\mathbf{S})$, le treillis agrégé résultant est équivalent au treillis bit/symbole. Les valeurs intermédiaires de T permettent de doser un compromis entre complexité et performances de décodage. En utilisant l'approximation de la complexité d'un treillis

de paramètre T (équation (3.4)), la complexité moyenne D_m de l'algorithme de décodage combiné multi-treillis est donnée par

$$D_{\text{m}}(T_1, T_2) = T_1 D_{\text{bal}} + T_2 D_{\text{bal}} + \rho(T_1, T_2)\, T_1\, T_2\, D_{\text{bal}}, \tag{3.14}$$

où $\rho(T_1, T_2) = \mathbb{P}(\hat{\mathbf{S}}_1 \neq \hat{\mathbf{S}}_2)$.

L'algorithme de décodage combiné multi-treillis trouve donc son utilité en terme de complexité si $D_m(T_1, T_2) < T_1\, T_2\, D_{bal}$, c'est à dire si

$$\rho < \rho^* = 1 - \frac{T_1 + T_2}{T_1 \times T_2}. \tag{3.15}$$

La complexité de l'algorithme proposé dépend donc de la probabilité $\rho(T_1, T_2)$ que les deux estimations qu'apporte l'algorithme de Viterbi sur les treillis T_1 et T_2 diffèrent. Cette probabilité décroît lorsque le bruit sur le canal et/ou la longueur de la séquence diminue(nt). Elle est difficilement estimable, mais on peut néanmoins la calculer empiriquement. La figure 3.11 illustre la complexité de l'algorithme de décodage combiné multi-treillis pour différents CLV et pour les paramètres $T_1 = 4$ et $T_2 = 5$. La valeur de $\rho(T_1, T_2)$ utilisée pour calculer cette complexité a été obtenue par simulation. On peut remarquer que l'algorithme de décodage combiné multi-treillis permet de réduire la complexité du décodage sur une plage intéressante de rapports signal à bruit. Cet algorithme est néanmoins moins adapté lorsque le bruit sur le canal est très important puisque les deux treillis vont plus rarement estimer la même séquence.

3.2.4 Optimisation sous contrainte des paramètres T_1 et T_2

On suppose dans cette section que l'on cherche à atteindre un niveau de performance, en terme de décodage, équivalent à celui obtenu sur un treillis de paramètre T_c. D'après l'algorithme de décodage combiné multi-treillis décrit ci-dessus, il est possible d'atteindre ce niveau de performance en utilisant deux treillis de paramètres respectifs T_1 et T_2 premiers entre eux et tels que $T_1 \times T_2 = T_c$. Sans perte de généralité, on peut écrire T_2 sous la forme $T_2 = T_1 + \Delta T$ et donc T_c sous la forme $T_c = T_1 \times (T_1 + \Delta T)$. On peut remarquer que les deux variables T_1 et ΔT suffisent à décrire de manière unique l'ensemble des couples d'entiers (T_1, T_2) premiers entre eux et qui vérifient $T_1 \times T_2 = T_c$. La probabilité ρ étant une fonction de T_1 et T_2, elle est de manière équivalente une fonction de T_1 et ΔT. Sous ces conditions, la complexité moyenne de l'algorithme de décodage combiné multi-treillis de paramètres T_1 et $T_1 + \Delta T$ est donnée par

$$D_{\text{m}}(T_1, \Delta T) = \rho(T_1, \Delta T) D_{T_c} + D_{\text{bal}}(2\, T_1 + \Delta T) \tag{3.16}$$

La quantité $\rho(T_1, \Delta T)$ représente la probabilité que les treillis de paramètre T_1 et $T_1 + \Delta T$ ne fournissent pas la même estimation, et D_{T_c} la complexité du décodage sur un treillis de paramètre T_c. On suppose ici que cette quantité augmente avec ΔT. Cette

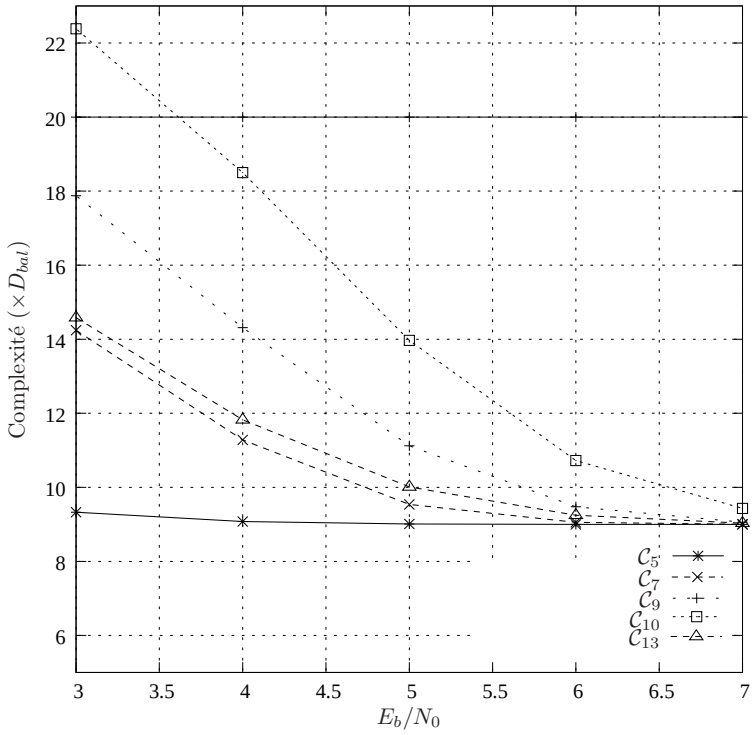

FIG. 3.11 – Complexité de l'algorithme de décodage combiné multi-treillis en fonction de E_b/N_0 pour les codes $\mathcal{C}_5$, $\mathcal{C}_7$, $\mathcal{C}_9$, $\mathcal{C}_{11}$ et $\mathcal{C}_{13}$ et comparaison avec l'approche classique pour $T_1 = 4$ et $T_2 = 5$.

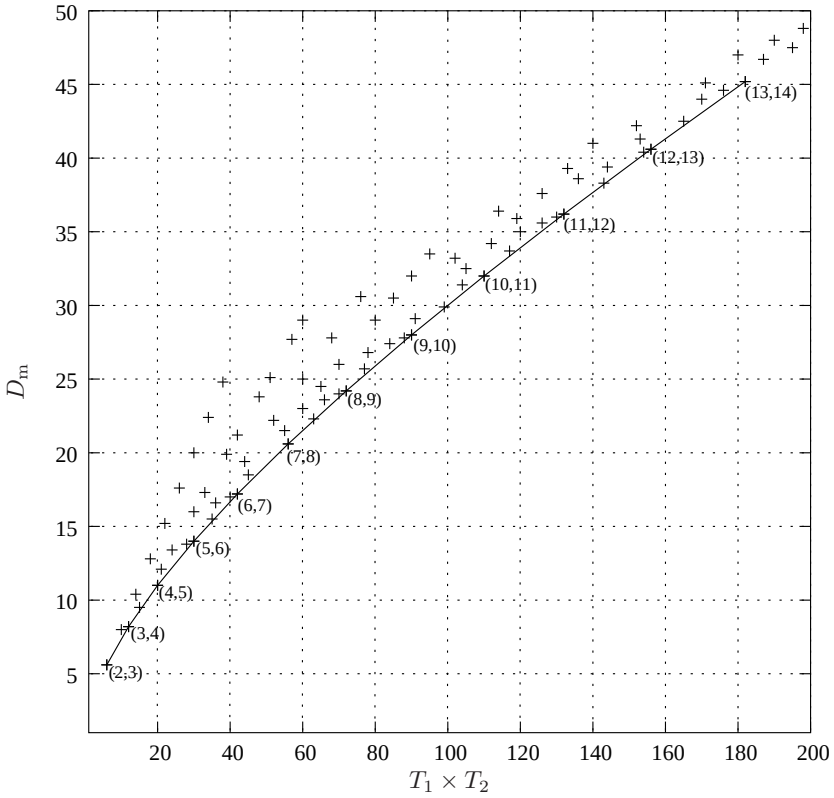

FIG. 3.12 – Complexité de l'algorithme de décodage combiné multi-treillis en fonction du couple (T_1, T_2) pour un ρ fixé.

hypothèse peut parfois ne pas être vérifiée pour certaines valeurs de ΔT lorsque le code considéré présente des propriétés de resynchronisation particulières. Nous verrons par exemple dans le chapitre 4 que cette hypothèse n'est pas toujours vérifiée pour le code $\mathcal{C}_{13}$. Sous cette hypothèse, nous pouvons néanmoins déduire la propriété suivante

Propriété 3.3 *Soit $T_c \in \mathbb{N}^*$ et $\mathcal{R} \subseteq \mathbb{N}^* \times \mathbb{N}^*$ le sous-ensemble des entiers naturels premiers entre eux. Alors,*

$$\arg \min_{T_1,T_2 \in \mathcal{R}_p \,|\, T_1 T_2 = T_c} D_{mtd} = \arg \min_{T_1,T_2 \in \mathcal{R}_p \,|\, T_1 T_2 = T_c} |T_2 - T_1|. \tag{3.17}$$

D'après cette propriété, les couples (T_1, T_2) de la forme $T_2 = T_1 + 1$ sont optimaux en terme de complexité moyenne de décodage pour un niveau de performance fixé. Ce résultat est illustré sur la figure 3.12. On a supposé pour cette figure que ρ était fixé.

3.3 Calcul de la marginale postérieure au niveau symbole sur le modèle agrégé

Le modèle d'état présenté en début de ce chapitre permet d'approcher les performances de décodage obtenues sur un modèle d'état optimal pour une complexité significativement réduite. En utilisant ce modèle d'état agrégé, et en lui appliquant un algorithme de décodage adapté, deux mesures de distance peuvent être minimisées : le TEB (en utilisant un algorithme de type BCJR [BCJR74] et en marginalisant au niveau bit les probabilités a posteriori) et le TESQ (en utilisant l'algorithme de Viterbi). Sur un modèle d'état optimal (treillis bit symbole [BH00b] pour les CLV et [GG04] pour les codes QA), la minimisation du TES peut s'effectuer en appliquant l'algorithme BCJR et en marginalisant les probabilités a posteriori au niveau symbole. Cette marginalisation s'effectue directement sur le treillis bit/symbole puisque l'horloge symbole est entièrement intégrée dans le modèle. Par contre, si le modèle agrégé est utilisé, une telle marginalisation au niveau symbole ne peut être effectuée puisque le modèle ne contient qu'une information sur les valeurs d'horloge symbole modulo le paramètre d'agrégation T. Le calcul de la marginale symbole a posteriori ne peut donc être effectué directement sur le modèle agrégé. Afin de minimiser le TES, ou afin d'utiliser une structure itérative composée de CLV ou de codes QA (comme dans [KT02] par exemple), une marginale symbole est nécessaire.

Dans cette section, un algorithme de décodage sur le modèle agrégé permettant de calculer les probabilités a posteriori au niveau symbole est décrit. Cet algorithme procède en deux étapes. Dans un premier temps, l'algorithme de Viterbi est appliqué sur un treillis agrégé de paramètre T. Cet algorithme sélectionne une séquence de probabilité a posteriori maximale. Cette séquence nous permet d'estimer les valeurs d'horloge symbole t_k associées aux différentes valeurs m_k de l'horloge symbole modulo T, et ce à chaque instant bit k. On définit ainsi un ensemble réduit de valeurs d'horloge symbole possible à chaque instant bit. L'algorithme BCJR est ensuite appliqué sur le modèle agrégé en assignant à chaque état l'estimation de l'horloge symbole associée. Les probabilités marginales au niveau symbole sont directement obtenues par l'algorithme BCJR.

L'algorithme permettant de calculer la probabilité marginale au niveau symbole sur un modèle agrégé est détaillé dans le reste de cette section. Des résultats de simulation sont également proposés en fin de section.

3.3.1 Présentation de l'algorithme

On se place sur un treillis de paramètre T_η^*, où $\eta \in [0, 1]$. Le paramètre T_η^* représente la plus petite valeur du paramètre d'agrégation T telle que la probabilité que l'algorithme de Viterbi appliqué sur le modèle de paramètre T_η^* sélectionne une séquence

de symboles de longueur $L(\mathbf{S})$ (i.e., de longueur correcte) soit supérieure ou égale à $1-\eta$. La façon de calculer la valeur de ce paramètre T_η^* pour un code et un rapport signal à bruit donnés est détaillée dans le chapitre 4.

La probabilité marginale au niveau symbole peut être directement obtenue à partir des probabilités a posteriori du couple de variables aléatoires (N_k, T_k). Rappelons que ces variables aléatoires représentent respectivement l'état du décodeur à l'instant bit k (noeud interne de l'arbre pour les CLV ou état de l'automate de décodage pour les codes QA) et les valeurs possibles d'horloge symbole à l'instant bit k. Ces probabilités a posteriori ne sont pas directement disponibles sur le modèle agrégé. L'algorithme BCJR appliqué au treillis de paramètre T permet "uniquement" de calculer les probabilités a posteriori des états (N_k, M_k), où M_k représente les valeurs possibles de l'horloge symbole modulo le paramètre d'agrégation. Afin de calculer les probabilités a posteriori des couples (N_k, T_k) à partir de celles des couples (N_k, M_k), il est nécessaire d'estimer les valeurs T_k que peut prendre l'horloge symbole pour chaque valeur donnée de M_k. Une fois cette estimation réalisée, l'algorithme BCJR permet de faire une estimation de la probabilité marginale au niveau symbole. Nous verrons dans la section 3.3.3 que cette estimation est fiable. Plus précisément, l'algorithme proposé se déroule de la façon suivante.

3.3.1.1 Première étape : correspondance entre M_k et T_k

L'algorithme de Viterbi appliqué sur le treillis de paramètre T_η^* sélectionne une séquence d'états $\{s_1^* = (n_1^*, m_1^*), \ldots, s_{L(\mathbf{X})}^* = (n_{L(\mathbf{X})}^*, m_{L(\mathbf{X})}^*)\}$ maximisant la probabilité a posteriori $\mathbb{P}(s_1 = s_1^*, \ldots, s_{L(\mathbf{X})} = s_{L(\mathbf{X})}^* \,|\, Y_1^{L(\mathbf{X})})$. La quantité m_k^* représente la valeur prise par la variable aléatoire M_k dans la séquence sélectionnée par l'algorithme de Viterbi (i.e., la séquence la plus probable). La séquence d'états ci-dessus correspond également à une séquence estimée de bits $\hat{X} = \hat{X}_1, \ldots, \hat{X}_{L(\mathbf{X})}$. Une estimation $\hat{S} = \hat{S}_1, \ldots, \hat{S}_{L(\mathbf{S})}$ de la séquence émise est obtenue à partir de $\hat{X}$ et de la structure du code considéré. La séquence $\hat{S}$ nous permet d'obtenir une estimation des valeurs d'horloge symbole $\hat{t}_k$ associées à la valeur modulo m_k^*, et ce pour chaque instant bit k.

Exemple 3.2:
Considérons que l'on souhaite transmettre le message $aabaa$ *en utilisant le code* $\mathcal{Q}_7$. *L'encodage de ce message produit le train binaire* 0100. *On se place sur le modèle agrégé de paramètre* $T = 3$. *Ce treillis est représenté de manière simplifiée sur la figure 3.13. Seules les valeurs d'horloge symbole modulo* T *(soit* 0, 1, *et* 2*) sont représentées sur ce treillis. La suite d'états sélectionnée par l'algorithme de Viterbi est représentée sur ce treillis par des flèches. La première étape de l'algorithme, décrite ci-dessus, consiste à associer à chacun des états sélectionnés une estimation de la valeur complète de l'horloge symbole. Pour cela, on utilise la séquence de bits déterminée par l'algorithme de Viterbi (ici,* 0100*). Les valeurs estimées de l'horloge symbole pour*

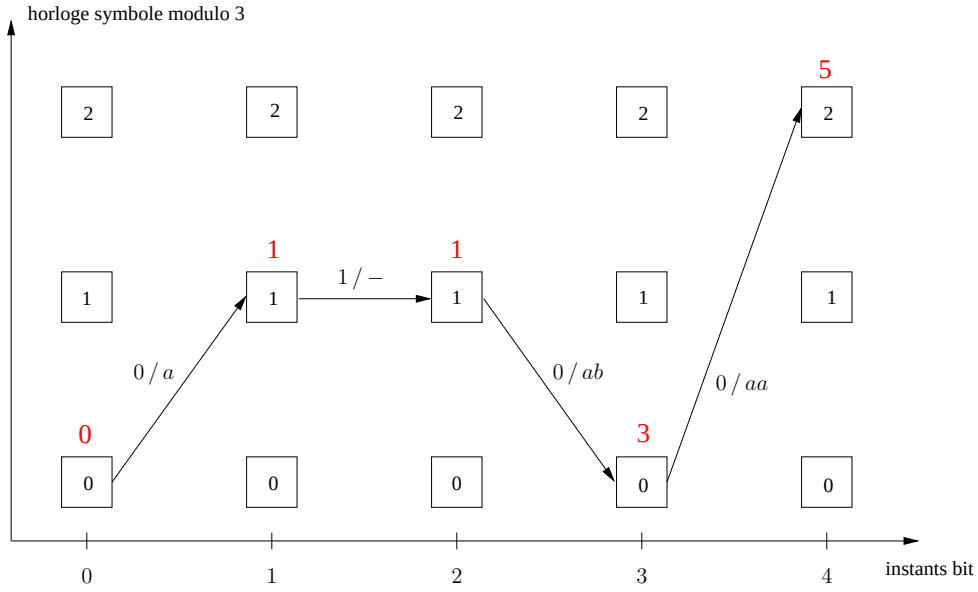

FIG. 3.13 – Estimation de l'horloge symbole pour les états sélectionnés par l'algorithme de Viterbi.

chacun des états sélectionnés sont représentées au dessus des états sur la figure 3.13. Le même cheminement est représenté dans le tableau 3.3.

Nous avons donc obtenu pour l'instant une estimation de l'horloge symbole associée à la valeur modulo m_k^*, et ce pour chaque instant bit k. Ce n'est pas suffisant pour estimer la probabilité marginale au niveau symbole sur le treillis agrégé. Une estimation de l'horloge symbole pour chaque valeur modulo m_k (et non pour m_k^* uniquement) est nécessaire. Pour cela, on va définir pour chaque m_k entre 0 et $T_\eta^* - 1$, la valeur $\varphi(\hat{t}_k, m_k)$ comme la valeur de l'horloge symbole la plus proche de t_k^* et congrue à m_k

Instants bit	État sélectionné	Symboles décodés	Horloge symbole estimée
0	0	–	0
1	1	a	1
2	1	a	1
3	0	aab	3
4	2	$aabaa$	5

TAB. 3.3 – Estimation de l'horloge symbole pour les états sélectionnés par l'algorithme de Viterbi.

Instants bit	État sélectionné	$\hat{t}_k$	État à traiter	Horloge symbole estimée
0	0	0	1	1
0	0	0	2	2
1	1	1	0	0
1	1	1	2	2
2	1	1	0	0
2	1	1	2	2
3	0	3	1	4
3	0	3	2	2
4	2	5	0	6
4	2	5	1	4

TAB. 3.4 – Estimation de l'horloge associée aux états restants

modulo T_η^* :

$$\varphi(\hat{t}_k, m_k) = \hat{t}_k + \left\lfloor \frac{T_\eta^* - 1}{2} \right\rfloor + \left((m_k - m_k^* + \left\lfloor \frac{T_\eta^* - 1}{2} \right\rfloor) \bmod T_\eta^* \right) \tag{3.18}$$

où on force le terme $\left((m_k - m_k^* + \left\lfloor \frac{T_\eta^* - 1}{2} \right\rfloor) \bmod T_\eta^* \right)$ à prendre ses valeurs dans $\{0, \ldots, T_\eta^* - 1\}$. On remarque que lorsque $m_k = m_k^*$, on a bien $\varphi(\hat{t}_k, m_k) = \hat{t}_k$.

En résumé, l'algorithme de Viterbi appliqué sur le modèle agrégé de paramètre T_η^* permet d'estimer la valeur d'horloge symbole $\hat{t}_k$ associée à la valeur symbole modulo T_η^* prise par la séquence la plus probable, à chaque instant bit de cette séquence. Ensuite, l'équation 3.18 permet de sélectionner à chaque instant bit k, un sous-ensemble de T_η^* entiers consécutifs centré sur la valeur $\hat{t}_k$.

Exemple 3.3:
L'estimation de l'horloge symbole associée aux états restants, décrite ci-dessus, est représentée pour les mêmes paramètres que l'exemple 3.2 sur la figure 3.14. Les valeurs estimées de l'horloge symbole pour les états restants sont indiquées au dessus des états. Un récapitulatif est donné dans le tableau 3.4. Ces estimations ont été obtenues avec l'équation (3.18).

3.3.1.2 Deuxième étape : obtention de la probabilité marginale au niveau symbole

Nous allons maintenant appliquer l'algorithme BCJR sur le modèle défini par les couples de variable aléatoire (N_k, M_k). On assigne à chacun de ces couples la valeur d'horloge symbole $\varphi(\hat{t}_k, M_k)$ estimée lors de l'étape 1 décrite ci-dessus. Pour chaque

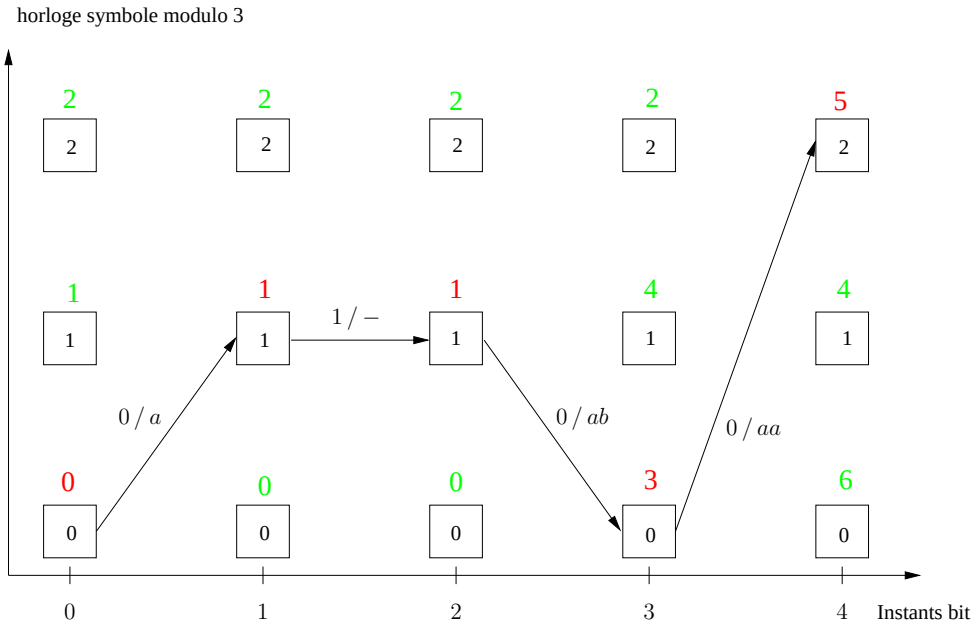

FIG. 3.14 – Estimation de l'horloge symbole associée aux états restants.

n dans l'ensemble des valeurs possibles de N_k (i.e., le nombre d'états du décodeur) et pour chaque $m \in \{0, \ldots, T_\eta^*\}$, les passes avant et arrière de l'algorithme BCJR permettent de calculer les fonctions de probabilité suivantes pour chaque état $v = (n, m)$ du treillis :

$$\alpha_k(v) = \mathbb{P}(\mathcal{V}_k = v; \mathbf{Y}_1^k) \tag{3.19}$$

$$\beta_k(v) = \mathbb{P}(\mathbf{Y}_{k+1}^{L(\mathbf{X})} \mid \mathcal{V}_k = v), \tag{3.20}$$

pour $1 \leq k \leq L(\mathbf{X})$.

Nous allons maintenant définir, pour chaque symbole a_i de l'alphabet du code, l'ensemble $\tau_j(a_i)$ composé des paires d'états $v = (n, m), v' = (n', m')$ du treillis telles que le $j$ème symbole émis par la transition entre v et v' est égal à a_i. Cet ensemble d'états est directement obtenu à partir de l'arbre d'un CLV ou de l'automate de décodage d'un code QA. On peut noter que, dans le cas des CLV, les ensembles τ_j pour $j > 1$ sont vides. Ainsi, on obtient la probabilité marginale au niveau symbole par la relation suivante :

$$\mathbb{P}(U_t = a_i \mid Y_1^{L(\mathbf{X})}) \propto \sum_{1 \leq k \leq L(\mathbf{X})} \sum_{j \geq 1} \sum_{(v,v') \in \mathcal{T}_j^t(a_i)} \alpha_k(v)\, \beta_k(v')\, \gamma_k(v, v'), \tag{3.21}$$

où $\gamma_k(v, v')$ est obtenu à partir des observations issues du canal et où $\mathcal{T}_j^t(a_i)$ est l'ensemble des paires d'états (v, v') dans le treillis telles que $(v, v') \in \tau_j(a_i)$ et $\varphi(\hat{t}_k, m) =$

$t - j$.

Les deux étapes de l'algorithme proposé ont été, pour des raisons de clarté, décrites séparément. Cependant, afin de réduire le temps de calcul associé à cet algorithme, elles peuvent être implémentées en parallèle. En effet, l'algorithme de Viterbi (étape 1) et la passe avant du BCJR (étape 2) peuvent être réalisées simultanément. Il faut, dans ce cas, stocker deux mesures différentes pour chaque état du treillis à un instant bit k donné : $\max(\mathbb{P}(\mathcal{V}_1, \ldots, \mathcal{V}_k; Y_1^k))$ pour l'algorithme de Viterbi et la probabilité de (3.19) pour l'algorithme BCJR. Le calcul de la métrique de branche γ peut ainsi n'être réalisé qu'une fois.

En sélectionnant un intervalle de T_η^* valeurs d'horloge symbole pour chaque instant bit, certaines probabilités sont surestimées. En effet, un état dans le modèle agrégé contient les probabilités, non seulement des séquences de longueur $\varphi(\hat{t}_k, m_k)$, mais également des séquences de longueur $\varphi(\hat{t}_k, m_k) \pm k \times T_\eta^*$, $k \in \mathbb{Z}$. Cependant, d'après la définition de T_η^* (4.30), la probabilité qu'une séquence soit désynchronisée de plus de T_η^* symboles est inférieure ou égale à η. Plus précisément, on a

$$\sum_{i \in \mathbb{Z} - \{-T_\eta^*, \ldots, T_\eta^*\}} \mathbb{P}(\Delta S = i) < \eta, \tag{3.22}$$

où ΔS représente la différence entre le nombre de symboles encodés et décodés lorsque le message original est transmis sur un canal BABG caractérisé par son rapport signal à bruit E_b/N_0. Les probabilités qui seront surestimées en utilisant l'algorithme décrit dans cette section seront donc très faibles (en simulation, la valeur $\eta = 10^{-6}$ est utilisée).

3.3.2 Analyse de complexité

Le nombre d'états d'un treillis agrégé de paramètre T_η^* est linéaire avec T_η^* et avec la longueur $L(\mathbf{X})$ en bits de la séquence et est environ de $T_\eta^* \times L(\mathbf{X}) \times \text{card}(\Omega_d)$, où Ω_d représente l'ensemble des états du décodeur. La complexité $\mathcal{D}(T_\eta^*)$ de l'algorithme BCJR sur le modèle de paramètre T_η^* est proportionnelle au nombre d'états du modèle. Ainsi, $\mathcal{D}(T_\eta^*) = \zeta\, T_\eta^* \times \text{card}(\Omega_d) \times L(\mathbf{X})$, où ζ est une constante. On suppose ici que la complexité de l'algorithme de Viterbi sur le même modèle est inférieure à celle de l'algorithme BCJR, comme seule une passe avant est effectuée dans l'algorithme de Viterbi. On peut donc écrire que la complexité $\mathcal{D}_{pa}$ de l'algorithme proposé dans cette section vérifie

$$\mathcal{D}_{pa} \leq 2\,\zeta\, T_\eta^* \times \text{card}(\mathcal{N}) \times L(\mathbf{X}). \tag{3.23}$$

La complexité de l'algorithme proposé est donc inférieure à celle de l'algorithme BCJR sur le treillis bit/symbole ($T = L(\mathbf{S})$), méthode classiquement utilisée pour le calcul de la marginale symbole) à condition que le paramètre T_η^* vérifie $T_\eta^* \leq L(\mathbf{S})/2$. Nous

verrons dans le chapitre suivant que pour $\eta = 10^{-6}$, le paramètre T_η^* vérifie cette condition pour la plupart des CLV et des codes QA considérés pour $E_b/N_0 > 0$ dB.

3.3.3 Résultats de simulation

L'algorithme permettant de calculer les probabilités marginales au niveau symbole a été testé à l'aide de simulations. Nous l'avons appliqué à différents CLV et codes QA. Les performances de cet algorithme au sens du taux d'erreur symbole ont été comparées à celles de :

- l'algorithme BCJR appliqué sur le modèle agrégé de paramètre T_η^* et calculant les probabilités marginales au niveau bit. La séquence bits transmise est estimée en prenant une décision dure sur les probabilités a posteriori sur les bits calculées par le BCJR. Puis, un décodeur standard CLV/QA est utilisée pour estimer la séquence de symboles
- l'algorithme de Viterbi sur le modèle agrégé de paramètre T_η^*.

Les résultats de cette section sont donnés en terme de TES calculé avec la distance de Hamming normalisée entre les séquences transmises et estimées. La longueur de référence pour le calcul du TES est la longueur de la séquence transmise. Les éventuels symboles manquants ou supplémentaires sont considérés comme erronés.
Les performances en terme de taux d'erreur symboles en fonction du rapport signal à bruit pour le code $\mathcal{Q}_7$ sont représentées sur la figure 3.15. La valeur de η a été choisie ici égale à 10^{-6}, et le paramètre T_η^* a été ajusté en fonction. Les valeurs de T_η^* appartiennent à l'intervalle $[25, 141]$. Les séquences utilisées pour ces simulations sont de longueur $L(\mathbf{S}) = 300$ symboles. Les performances de l'algorithme BCJR sur le modèle complet ($T = L(\mathbf{S})$) ne sont pas représentées sur cette figure car ce sont exactement les mêmes que celles de l'algorithme proposé. On peut également remarquer que les performances de l'algorithme de Viterbi correspondent à celles de la première étape de l'algorithme proposé.

Les mêmes résultats pour les codes $\mathcal{C}_5$, $\mathcal{C}_{10}$ et $\mathcal{Q}_9$ sont représentés dans le tableau 3.5. Ces résultats ont été obtenus pour des séquences de $L(\mathbf{S}) = 100$ symboles. Pour $\eta = 10^{-6}$, le paramètre T_η^* est *optimal*, c'est à dire que les performances de ces codes sur le modèle agrégé de paramètre T_η^* sont les mêmes que celles sur le modèle optimal. Les valeurs de ce paramètre optimal sont également reportées dans le tableau.
L'impact du paramètre d'agrégation sur les performances de l'algorithme proposé a été également étudié. La figure 3.16 représente le TES en fonction de T pour le code $\mathcal{C}_{10}$ et un rapport signal à bruit de 5 dB. Les valeurs de T sur cette figure correspondent à $\eta \in [10^{-6}, 10^{-1}]$. On peut donc remarquer l'impact de η sur les performances de l'algorithme. Plus η est faible, plus précise est l'estimation de l'horloge symbole (correspondant à la première étape de l'algorithme), et meilleurs seront les résultats à la

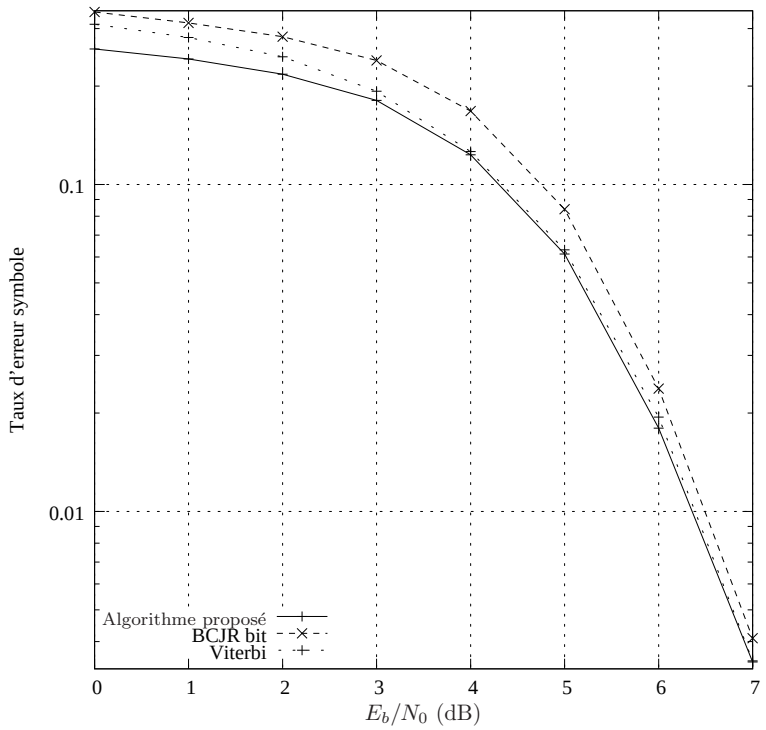

FIG. 3.15 – Taux d'erreur symbole pour le code $\mathcal{Q}_7$ en fonction du rapport signal à bruit pour différentes techniques de décodage

$E_b/N_0(dB)$	0	1	2	3	4	5	6	7
	Code $\mathcal{C}_{10}$							
Viterbi	0.5896460	0.5394457	0.4613242	0.3393017	0.1829289	0.0615358	0.0124061	0.0016876
BCJR bit	0.6521376	0.6221372	0.5681771	0.4567565	0.2593654	0.0828331	0.0148831	0.0017665
Algorithme proposé	0.5096099	0.4722298	0.4129241	0.3130139	0.1741670	0.0597935	0.0123041	0.0016345
T^*_η	43	39	34	30	25	21	16	12
	Code $\mathcal{C}_5$							
Viterbi	0.3572242	0.2781811	0.1944515	0.1161866	0.057206	0.0228583	0.0074904	0.0020121
BCJR bit	0.4198277	0.336879	0.2382734	0.1409759	0.0666855	0.0250575	0.0076668	0.0020160
Algorithme proposé	0.3324591	0.2620679	0.1851243	0.1125048	0.0563842	0.0227737	0.0074707	0.0020085
T^*_η	15	12	10	9	8	6	5	4
	Code $\mathcal{Q}_9$							
Viterbi	0.1466013	0.1326017	0.1103956	0.0773081	0.0397315	0.0140524	0.0030577	0.0004332
BCJR bit	0.1628432	0.1531363	0.1353369	0.1016360	0.0545963	0.0175466	0.0034213	0.0004613
Algorithme proposé	0.1035528	0.0931538	0.0848605	0.0674245	0.0380343	0.0133155	0.0029066	0.0004297
T^*_η	49	44	39	33	28	22	17	11

TAB. 3.5 – Taux d'erreur symbole en fonction du rapport signal à bruit pour les codes $\mathcal{C}_5$, $\mathcal{C}_{10}$ et $\mathcal{Q}_9$ pour trois techniques de décodage différentes.

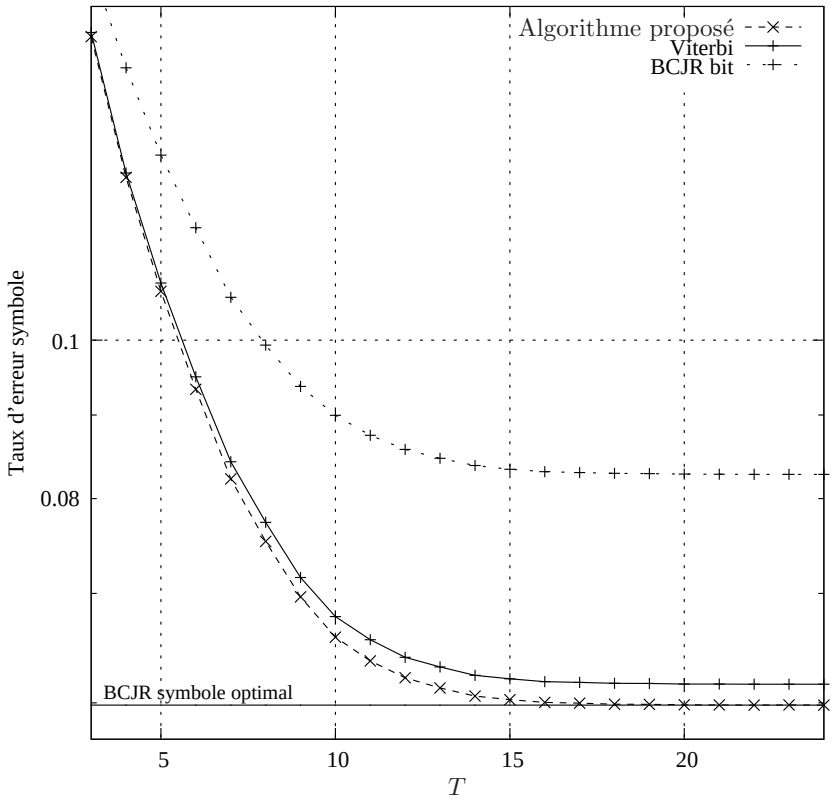

FIG. 3.16 – Taux d'erreur symbole pour le code $\mathcal{C}_{10}$ en fonction de T pour différentes techniques de décodage

fin de la deuxième étape. La ligne BCJR symbole optimal correspond au TES obtenu sur le treillis complet bit/symbole.

Dans les schémas de décodage itératif, une information extrinsèque sur les probabilités des symboles est échangée. Les probabilités extrinsèques sont calculées à partir des probabilités a posteriori issues de l'algorithme de décodage. Nous avons donc évalué la qualité des probabilités a posteriori issues de l'algorithme proposé et l'avons comparé à celles issues de l'algorithme BCJR sur treillis bit/symbole, ainsi qu'à celles issues de l'algorithme de Viterbi à sorties souples (SOVA) [Bat87][LC97]. L'algorithme SOVA est une extension de l'algorithme de Viterbi afin qu'il puisse calculer les probabilités a posteriori sur chacun des symboles de la séquence décodée. Pour comparer ces trois algorithmes de décodage, nous avons calculé la probabilité a posteriori moyenne des symboles de la séquence originale. On appellera cette mesure indice de confiance moyen sur les symboles. En d'autres termes, nous avons calculé l'indice de

$E_b/N_0(dB)$	0	1	2	3	4	5	6	7
	Code $\mathcal{Q}_7$							
SOVA	0.7006647	0.7372412	0.7814617	0.8303492	0.8756119	0.9088100	0.9291074	0.9415482
Algorithme proposé	0.7204538	0.7615880	0.8142597	0.8761704	0.9352687	0.9755301	0.9934656	0.9987168
BCJR symbole	0.7204547	0.7616008	0.8142599	0.8762152	0.9353449	0.9755583	0.9935396	0.9987495

TAB. 3.6 – Indices de confiance moyens au niveau symbole en fonction de E_b/N_0 issus de différents algorithmes de décodage.

confiance i_c suivant :

$$i_c = \frac{1}{L(\mathbf{S})} \sum_{1 \leq k \leq L(\mathbf{S})} \mathbb{P}(S_k | Y_1^{L(\mathbf{X})}), \tag{3.24}$$

où les termes $\mathbb{P}(S_k|Y_1^{L(\mathbf{X})})$ correspondent à la probabilité a posteriori issue de l'algorithme de décodage considéré. La mesure de cet indice de confiance est indiquée dans le tableau 3.6 pour le code $\mathcal{Q}_7$ et pour différentes valeurs du rapport signal à bruit. On peut remarquer dans ce tableau que l'utilisation de l'algorithme proposé pour calculer les probabilités marginales au niveau symbole n'affecte que très peu la qualité de la probabilité a posteriori par rapport au décodage optimal sur le treillis bit/symbole. On remarque également que l'indice de confiance moyen issu de l'algorithme SOVA est bien inférieur à celui issu de l'algorithme proposé dans cette section.

3.4 Décodage souple avec information adjacente

Les codes entropiques, comme les CLV ou codes QA sont très sensibles au bruit du canal à cause de la désynchronisation que peuvent engendrer les erreurs bits. Une quantité très importante de symboles peuvent être décodés en erreur à cause d'une simple erreur bit. C'est pourquoi, comme nous l'avons vu dans le chapitre 2, de nombreux auteurs se sont penchés sur le problème de décodage souple et d'ajout d'information adjacente afin de rendre les codes entropiques plus robustes à la transmission. Nous proposons dans cette section une technique d'ajout d'information adjacente pour le décodage souple des CLV ou des codes QA. Cette information adjacente se trouve sous la forme de contraintes de longueur, référençant de multiples instants du processus de décodage, afin de favoriser la resynchronisation du décodeur à ces instants.

Nous avons vu dans les sections précédentes que la contrainte de terminaison sur un treillis de type bit/symbole ou agrégé permet d'améliorer les performances de décodage. Cette contrainte de terminaison représente un coût en terme de débit. Sur le treillis bit/symbole, $\lceil \log_2(L(\mathbf{S})) \rceil$ bits sont nécessaires pour représenter le nombre de symboles émis. Sur le treillis agrégé, la contrainte de terminaison représente $\lceil log_2(T) \rceil$ bits. Nous proposons dans cette section une méthode permettant d'améliorer les per-

formances des codes en introduisant des contraintes de longueur à différents instants du décodage. Principalement, deux approches sont utilisées : une première approche consiste à utiliser différemment les bits de la contrainte de terminaison et la deuxième consiste à introduire un taux fixé d'information adjacente au décodeur. La première approche sera comparée en termes de résultats à l'approche classique utilisant une contrainte de terminaison à la fin du processus de décodage, alors que la deuxième approche sera comparée à des techniques usuelles d'amélioration de la robustesse des codes.

Dans la suite, nous appellerons contrainte de longueur une contrainte sur le nombre de symboles décodés à un instant bit donné et nous appellerons contrainte de terminaison la contrainte de longueur au dernier instant bit. La contrainte de terminaison assure la resynchronisation du décodeur aux deux extrémités du message. Par contre, comme on peut le voir dans [KT05], celle-ci a très peu d'impact en terme de synchronisation au milieu du message. Les erreurs symboles se produisent beaucoup plus souvent au milieu qu'en ses deux extrémités.

Dans cette section, nous allons d'abord étudier l'impact de la position d'une contrainte de longueur sur les performances en terme de décodage. Puis, nous proposerons une stratégie pour remplacer une unique contrainte de longueur par plusieurs contraintes plus faibles (en terme de quantité d'information), placées à différents instants bits du processus de décodage. Ces contraintes augmentent les probabilités des séquences synchronisées, améliorant ainsi les performances de décodage. Ensuite, nous utiliserons cette méthode afin d'introduire de la redondance, sous la forme d'information adjacente, au décodeur. Cette approche est une alternative aux CLV réversibles ou au code QA avec symbole interdit. Cependant, la redondance introduite selon l'approche proposée n'est pas insérée dans le train binaire. La redondance peut donc être ajoutée de façon beaucoup plus flexible.

3.4.1 Impact de la position d'une contrainte de longueur

Une contrainte de longueur sur le treillis agrégé invalide les séquences pour lesquelles le nombre de symboles décodés modulo T (M_k) à un instant bit k donné diffère d'une certaine valeur m_k. Toutes les séquences qui ne passent pas par les états du treillis tels que $M_k = m_k$ ne sont pas considérées. Une telle contrainte peut être exploitée à la fin du décodage, en tant que contrainte de terminaison. Dans ce cas, la contrainte devient $M_{L(\mathbf{X})} = L(\mathbf{S}) \bmod T$. La quantité d'information nécessaire pour la signaler au décodeur est de l'ordre de $\nu = \lceil log_2(T) \rceil$ bits d'information. Dans le reste de cette section, nous allons optimiser l'usage de ces ν bits d'information.
Nous allons considérer le fait qu'une contrainte de longueur peut être exploitée à des instants bits intermédiaires $k = 1, \ldots, L(\mathbf{X})$, aidant ainsi le décodeur à resynchroniser à ces instants. Le TES dépend en fait de la position relative de la contrainte de

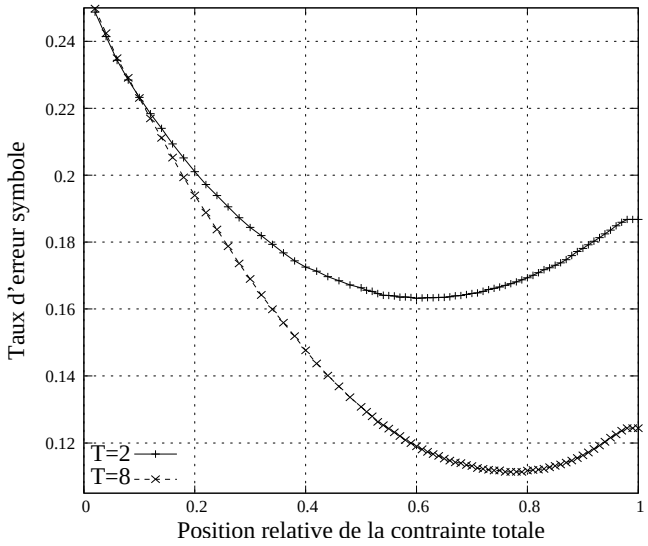

FIG. 3.17 – Taux d'erreur symbole pour le code $\mathcal{C}_5$ en fonction de la position relative d'une contrainte totale.

longueur sur le modèle agrégé. Le TES en fonction de la position relative d'une contrainte de longueur pour le code C_5 est représenté sur la figure 3.17. Il a été obtenu par l'algorithme de Viterbi appliqué sur les modèles de paramètre $T = 2$ et $T = 8$ et pour un rapport signal à bruit de 3 dB. La position relative optimale de la contrainte de longueur se situe à 0.6 pour $T = 2$ et 0.79 pour $T = 8$. Pour ces positions de contrainte, le TES obtenu est plus faible que lorsque la contrainte est placée à la fin. Il faut noter que pour les CLV, la contrainte du noeud racine est utilisée à la fin. En effet, on sait qu'après le dernier bit, le décodeur doit forcément se trouver dans le noeud racine de l'arbre du CLV. Étant donné que la contrainte de longueur est plus forte (en terme de quantité d'information) que la contrainte du noeud racine, la position optimale de la contrainte de longueur est supérieure à 0.5 et augmente avec la valeur de T, comme on peut le remarquer sur la figure 3.17. Les mêmes simulations ont été réalisées pour le code QA $\mathcal{Q}_7$ pour $T = 3$ et $T = 7$ lorsque le rapport signal à bruit est égal à 5 dB. Les résultats sont présentés sur la figure 3.18. Les positions relatives optimales sont 0.80 pour $T = 3$ et 0.82 pour $T = 7$. Globalement, les positions optimales pour les codes QA sont toujours plus proches de 1 que pour les CLV. Ceci s'explique principalement par le fait que, contrairement au CLV, les codes QA ne peuvent pas utiliser la contrainte du noeud racine. Ainsi, seule la contrainte de longueur est disponible pour ces codes.

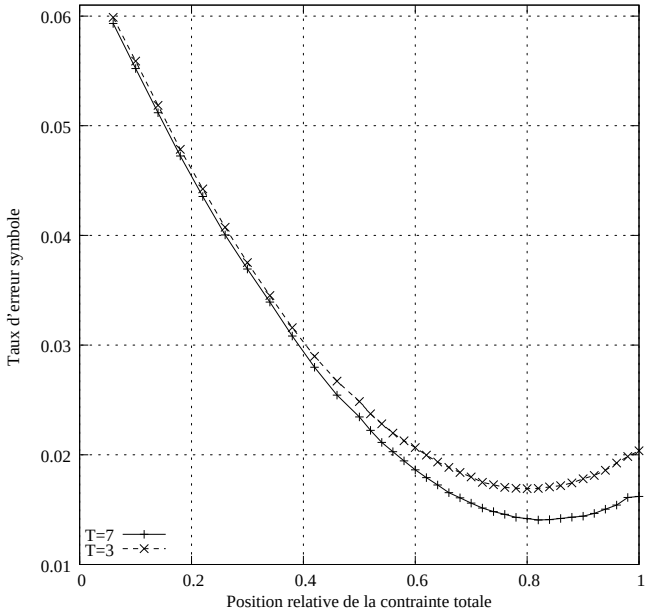

FIG. 3.18 – Taux d'erreur symbole pour le code $\mathcal{Q}_7$ en fonction de la position relative d'une contrainte totale.

3.4.2 Utilisation de contraintes de longueur partielles

Une contrainte de longueur sur le modèle agrégé peut être séparée et répartie tout au long du processus de décodage. Une contrainte est dite *partielle* si on ne donne au décodeur qu'une partie de la représentation binaire de m_k. Plus généralement, si on ne donne que ω bits de la représentation de m_k (ν bits), la contrainte partielle partitionne l'ensemble $\Gamma_T = \{0, \ldots, T-1\}$ des valeurs possibles de l'horloge symbole modulo T en 2^ω sous-ensembles de cardinal $2^{\nu-\omega}$. Par exemple, les partitions de $\Gamma_{T=8}$ correspondant respectivement au premier, second et dernier bit de m_k sont données par

$$\mathcal{P}_1 = \{\{0,1,2,3\},\{4,5,6,7\}\},$$
$$\mathcal{P}_2 = \{\{0,1,4,5\},\{2,3,6,7\}\},$$
$$\mathcal{P}_3 = \{\{0,2,4,6\},\{1,3,5,7\}\}.$$

Pour un taux fixé d'information adjacente, l'utilisation de contraintes partielles permet d'augmenter la fréquence de celles-ci, afin de lutter contre la désynchronisation en plusieurs endroits pendant le décodage.
Il faut maintenant déterminer le type de contraintes à utiliser, à quelle fréquence, et à quels instants, afin d'obtenir un TES le plus faible possible. Une telle configuration optimale dépend du rapport signal à bruit, du paramètre T du modèle agrégé et du code

considéré. Pour trouver une telle configuration optimale un algorithme de type recuit simulé permet d'obtenir la configuration optimale pour un ensemble de paramètres fixés.

Exemple 3.4: On suppose dans cet exemple que $T = 8$*. On dispose de* 3 *bits d'information adjacente. Nous allons utiliser le code* $\mathcal{C}_5$ *avec* $E_b/N_0 = 3$ *dB et le code* $\mathcal{Q}_7$ *avec* $E_b/N_0 = 6$ *dB. Supposons que l'on place 3 contraintes partielles de 1 bit plutôt que la contrainte totale de 3 bits. Ces trois contraintes peuvent être positionnées à différents endroits et trois configurations sont possibles pour chacune de ces contraintes (*$\mathcal{P}_1, \mathcal{P}_2$ *ou* $\mathcal{P}_3$*). Dans le tableau 3.7 sont reportés les TES associés à différents positionnements et configurations de ces contraintes partielles pour le code* $\mathcal{C}_5$*. La dernière ligne de ce tableau correspond à la solution optimale obtenue avec l'algorithme de recuit simulé. Le TES associé à cette configuration optimale est de* 0.092272*. Le TES associé au placement optimal d'une contrainte totale est de* 0.111325*. On obtient donc de meilleurs performances en utilisant 3 contraintes partielles de 1 bit plutôt qu'une contrainte totale de 3 bits à condition d'utiliser la solution optimale donnée par l'algorithme de recuit simulé. Les mêmes simulations ont été réalisées pour le code QA* $\mathcal{Q}_7$*. Les résultats sont présentés dans le tableau 3.8. L'algorithme de recuit simulé permet cette fois encore de trouver la configuration optimale, et ainsi d'obtenir un meilleur TES que la solution classique qui consiste à utiliser une contrainte de terminaison.*

L'occurrence moyenne des erreurs symboles en fonction de leur position à l'intérieur du message est représentée sur la figure 3.19 pour le code $\mathcal{Q}_7$ et pour 3 techniques différentes d'utilisation des bits référençant des contraintes de longueur : la contrainte de terminaison classique, une contrainte totale placée en position optimale (d'après les résultats de 3.4.1), et la configuration optimale de 3 contraintes partielles. Ces résultats ont été obtenus sur le modèle agrégé de paramètre $T = 8$ et pour un rapport signal à bruit de 6 dB. On peut remarquer que dans le cas où une contrainte de terminaison est utilisée, les premiers et derniers symboles du message sont rarement erronés en comparaison avec le reste des symboles. L'utilisation de contraintes partielles en plusieurs endroits permet de protéger le message en ces endroits et ainsi de limiter la propagation des erreurs dues à la désynchronisation du décodeur. En utilisant des contraintes partielles, les erreurs sont réparties de manière quasi-uniforme à l'intérieur du message, et sont moins nombreuses qu'en utilisant les autres méthodes présentées.

3.4.3 Décodage robuste avec information adjacente

Afin d'améliorer la robustesse des CLV ou des codes QA, plusieurs techniques ont été proposées dans la littérature, notamment l'utilisation de CLV réversibles [TWM95], l'introduction d'un SI à l'encodage des codes QA [BCI+97] ou encore l'utilisation de marqueurs de synchronisation à l'intérieur du train binaire [GG04]. Dans cette section, une méthode alternative est proposée afin d'introduire de la redondance, qui peut être vue comme de l'information adjacente sur le message à transmettre. Cette information

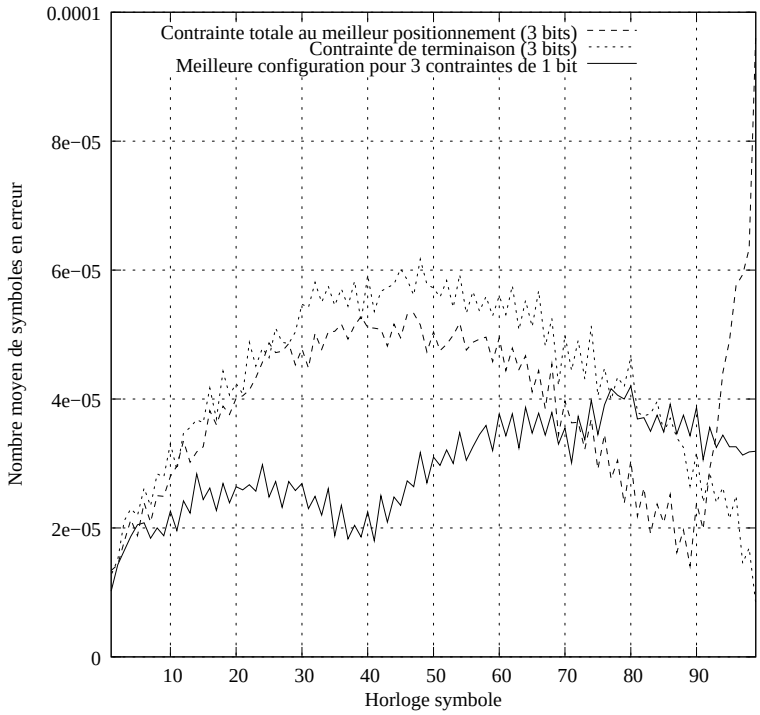

FIG. 3.19 – Occurrence des erreurs symboles en fonction de leurs positions pour le code Q_7, $T = 8$ et $E_b/N_0 = 6$ dB.

Positions relatives des contraintes partielles	Partitions	TESQ
0.45 0.70 0.90	$\mathcal{P}_3\ \mathcal{P}_3\ \mathcal{P}_3$	0.119660
0.45 0.70 0.90	$\mathcal{P}_3\ \mathcal{P}_3\ \mathcal{P}_2$	0.099410
0.45 0.70 0.90	$\mathcal{P}_3\ \mathcal{P}_2\ \mathcal{P}_3$	0.093786
0.45 0.70 0.90	$\mathcal{P}_2\ \mathcal{P}_2\ \mathcal{P}_2$	0.123442
0.45 0.70 0.90	$\mathcal{P}_3\ \mathcal{P}_2\ \mathcal{P}_1$	0.120153
0.45 0.70 1.00	$\mathcal{P}_3\ \mathcal{P}_3\ \mathcal{P}_3$	0.120066
0.45 0.70 1.00	$\mathcal{P}_3\ \mathcal{P}_3\ \mathcal{P}_2$	0.101525
0.45 0.70 1.00	$\mathcal{P}_3\ \mathcal{P}_2\ \mathcal{P}_3$	0.097617
0.45 0.70 1.00	$\mathcal{P}_2\ \mathcal{P}_2\ \mathcal{P}_2$	0.127222
0.45 0.70 1.00	$\mathcal{P}_3\ \mathcal{P}_2\ \mathcal{P}_1$	0.109124
0.40 0.65 0.95	$\mathcal{P}_3\ \mathcal{P}_3\ \mathcal{P}_3$	0.115764
0.40 0.65 0.95	$\mathcal{P}_3\ \mathcal{P}_3\ \mathcal{P}_2$	0.102030
0.40 0.65 0.95	$\mathcal{P}_3\ \mathcal{P}_2\ \mathcal{P}_3$	0.098405
0.40 0.65 0.95	$\mathcal{P}_2\ \mathcal{P}_2\ \mathcal{P}_2$	0.125231
0.40 0.65 0.95	$\mathcal{P}_3\ \mathcal{P}_2\ \mathcal{P}_1$	0.109208
0.42 0.70 0.82	$\mathcal{P}_3\ \mathcal{P}_2\ \mathcal{P}_3$	0.092272

TAB. 3.7 – Positionnements et configurations de 3 contraintes partielles de 1 bit et les taux d'erreur séqueces associés pour le code $\mathcal{C}_5$, $E_b/N_0 = 3$ dB et $T = 8$.

Positions relatives des contraintes partielles	Partitions	TES
0.40 0.64 0.88	$\mathcal{P}_3\ \mathcal{P}_3\ \mathcal{P}_3$	0.0138598
0.40 0.64 0.88	$\mathcal{P}_3\ \mathcal{P}_3\ \mathcal{P}_2$	0.0085099
0.40 0.64 0.88	$\mathcal{P}_2\ \mathcal{P}_3\ \mathcal{P}_2$	0.0057394
0.40 0.64 0.88	$\mathcal{P}_3\ \mathcal{P}_2\ \mathcal{P}_1$	0.0054139
0.40 0.64 0.88	$\mathcal{P}_3\ \mathcal{P}_2\ \mathcal{P}_2$	0.0053022
0.42 0.66 0.84	$\mathcal{P}_3\ \mathcal{P}_3\ \mathcal{P}_3$	0.0139033
0.42 0.66 0.84	$\mathcal{P}_3\ \mathcal{P}_3\ \mathcal{P}_2$	0.0057883
0.42 0.66 0.84	$\mathcal{P}_2\ \mathcal{P}_3\ \mathcal{P}_2$	0.0043527
0.42 0.66 0.84	$\mathcal{P}_3\ \mathcal{P}_2\ \mathcal{P}_1$	0.0055276
0.42 0.66 0.84	$\mathcal{P}_3\ \mathcal{P}_2\ \mathcal{P}_2$	0.0045336
0.44 0.66 0.82	$\mathcal{P}_3\ \mathcal{P}_3\ \mathcal{P}_3$	0.0144611
0.44 0.66 0.82	$\mathcal{P}_3\ \mathcal{P}_3\ \mathcal{P}_2$	0.0056628
0.44 0.66 0.82	$\mathcal{P}_2\ \mathcal{P}_3\ \mathcal{P}_2$	0.0045738
0.44 0.66 0.82	$\mathcal{P}_3\ \mathcal{P}_2\ \mathcal{P}_1$	0.0054556
0.44 0.66 0.82	$\mathcal{P}_3\ \mathcal{P}_2\ \mathcal{P}_2$	0.0047592
0.40 0.71 0.99	$\mathcal{P}_2\ \mathcal{P}_3\ \mathcal{P}_2$	0.0029031

TAB. 3.8 – Positionnements et configurations de 3 contraintes partielles de 1 bit et les taux d'erreur symboles associés pour le code $\mathcal{Q}_7$, $E_b/N_0 = 5$ dB et $T = 8$.

adjacente se trouve sous la forme de contraintes partielles associées au message original. Ainsi que nous l'avons vu dans cette section, il est préférable d'utiliser plusieurs contraintes partielles plutôt qu'une contrainte totale, à condition que la position et la configuration des contraintes soient choisies de façon idoine.

Considérons une séquence de $L(\mathbf{S})$ symboles à transmettre. Cette séquence est encodée à l'aide d'un CLV ou d'un code QA de taux de compression R_c. La longueur en bit du train binaire obtenu est notée $L(\mathbf{X})$ comme précédemment. Afin d'atteindre un taux de transmission désiré R_t ($R_t > R_q$), $\nu_b = \lfloor (R_t - R_q) \times L(\mathbf{S}) \rfloor$ bits d'information adjacente sont utilisés pour générer des contraintes partielles utilisables par le décodeur. En se basant sur les résultats de cette section, ν_b contraintes partielles de 1 bit sont utilisées. Ces ν_b contraintes sont données aux positions bits uniformément réparties $\{\lfloor k \times L(\mathbf{X})/\nu_b \rfloor, 1 \leq k \leq \nu_b\}$. Les bits d'information adjacente référençant les contraintes partielles sont transmis indépendamment du message binaire, et peuvent ainsi être protégés ou non contre le bruit issu du canal.

3.4.3.1 Comparaison en termes de performances avec des approches classiques

La méthode présentée ci-dessus permettant de fournir de l'information adjacente sous forme de contraintes de longueur au décodeur de CLV ou de codes QA a été comparée en termes de performances de décodage à des approches plus classiques introduisant de la redondance dans les trains binaires à longueurs variables. Tout d'abord, deux codes QA avec SI ont été considérés, nommés respectivement $\mathcal{Q}_7^{FS}$ et Q_8^{FS}. Ces codes sont adaptés des codes Q_7 et Q_8 définis respectivement pour les probabilités de source $P(a) = 0.7$ et $P(a) = 0.8$. La probabilité de l'intervalle correspondant au SI associée à

ces codes est égale à 0.1. Les automates de ces codes ont été construits selon la méthode présentée dans [BJWK06]. Les longueurs de description moyenne des codes $\mathcal{Q}_7$ et $\mathcal{Q}_8$ sont respectivement égales à $\text{edl}_7 = 0.890$ et $\text{edl}_8 = 0.733$. Celles des codes $\mathcal{Q}_7^{FS}$ et $\mathcal{Q}_8^{FS}$ sont égales à $\text{edl}_7^{FS} = 1.074$ et $\text{edl}_8^{FS} = 1.045$ respectivement. Ainsi, pour comparer impartialement les performances obtenues avec $\mathcal{Q}_7$ et $\mathcal{Q}_7^{FS}$, $\lfloor(\text{edl}_7^{FS} - \text{edl}_7) \times L(\mathbf{S})\rfloor$ bits d'information adjacente ont été introduits sous la forme de contraintes partielles de longueur . De façon similaire, $\lfloor(\text{edl}_8^{FS} - \text{edl}_8) \times L(\mathbf{S})\rfloor$ bits supplémentaires ont été introduits pour comparer $\mathcal{Q}_8$ et $\mathcal{Q}_8^{FS}$. Deux situations ont été considérées pour la transmission de ces bits d'information adjacentes : ils sont soit protégés par un code correcteur (moyennant une hausse du débit), soit ils subissent le bruit issu du canal au même titre que le train binaire original.

Les performances de ces deux stratégies de décodage, pour les codes $\mathcal{Q}_7$ et Q_7^{FS} sont représentées sur la figure 3.20, et sur la figure 3.21 pour les codes $\mathcal{Q}_8$ et $\mathcal{Q}_8^{FS}$. Ces résultats ont été obtenus pour 10^5 réalisations conjointes de la source et du canal pour des séquences de $L(\mathbf{S}) = 100$ symboles. L'algorithme de Viterbi a été appliqué sur le treillis agrégé de paramètre $T = 8$ pour calculer le taux d'erreur symbole moyen en sortie du décodeur. On peut remarquer que pour ces deux séries de résultats, la technique proposée atteint de meilleurs performances que celle basée sur un SI, et ce même dans le cas où les bits d'information adjacente ne sont pas protégés contre le bruit du canal. Il est intéressant de noter également que cette possibilité de protéger les bits d'information adjacente représente une flexibilité supplémentaire qui ne peut pas être appliquée avec un code QA utilisant un SI.

De la même façon, nous avons comparé les performances du CLV $\mathcal{C}_{10}$ avec son CLV réversible $\mathcal{C}_{10}^R$ associé. Le code $\mathcal{C}_{10}^R$ a été crée selon la méthode présentée dans [TWM95]. La redondance apportée par le caractère réversible du code $\mathcal{C}_{10}^R$ est de 0.1 bits par symboles. Ainsi, pour des séquences de 100 symboles, 10 bits d'information adjacente ont été introduits pour le décodage de $\mathcal{C}_{10}$. Les résultats sont présentés sur la figure 3.22. Cette fois encore, la méthode proposée est plus efficace en termes de performances de décodage.

3.5 Conclusion

Un nouveau modèle d'état pour le décodage souple des CLV et des codes QA a été présenté dans ce chapitre. Ce modèle d'état est défini par un paramètre T permettant de doser un compromis entre la complexité et les performances de décodage. Ce modèle permet d'atteindre les performances obtenues sur le treillis bit/symbole à condition de choisir le paramètre d'agrégation de façon appropriée.

Un algorithme de décodage combiné multi-treillis a également été présenté. Cet algorithme permet de réduire davantage la complexité du décodage pour un niveau

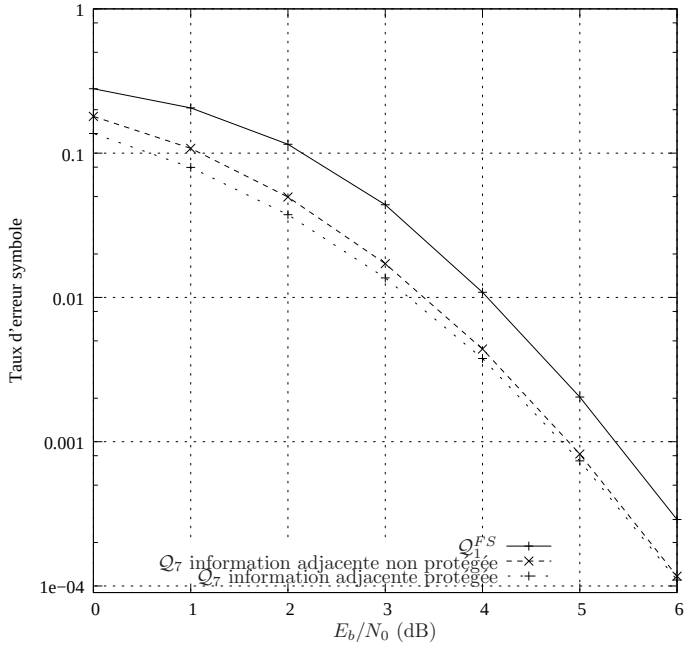

FIG. 3.20 – Taux d'erreur symbole en fonction de E_b/N_0 pour les codes $\mathcal{Q}_7$ et $\mathcal{Q}_7^{FS}$. Le taux de transmission est de 1.074 bits par symbole.

de performance équivalent, en utilisant deux treillis de plus petite taille, au lieu d'un treillis de plus grande taille.

Puis, nous avons proposé un algorithme d'estimation des probabilités marginales au niveau symbole sur le modèle agrégé. Ces probabilités ne peuvent en effet être obtenues directement sur ce modèle. L'algorithme proposé se déroule en deux étapes. Une étude de complexité a été présentée, montrant que cet algorithme est avantageux comparativement à un algorithme classique de type BCJR s'appliquant sur modèle d'état complet.

Enfin, un schéma de décodage souple avec information adjacente a été présenté dans ce chapitre. L'information adjacente se présente sous la forme de contraintes de longueur partielles référençant plusieurs instants bit dans le processus de décodage. Ces contraintes permettent de protéger la synchronisation tout au long du décodage. Des résultats de simulation ont montré que cette approche permettait d'atteindre de meilleures performances que les approches classiques d'ajout de redondance.

Dans le chapitre suivant, nous allons analyser de manière théorique les performances des codes sur le modèle agrégé. Pour cela, nous allons utiliser les propriétés de resynchronisation des codes pour quantifier la quantité d'information disponible au déco-

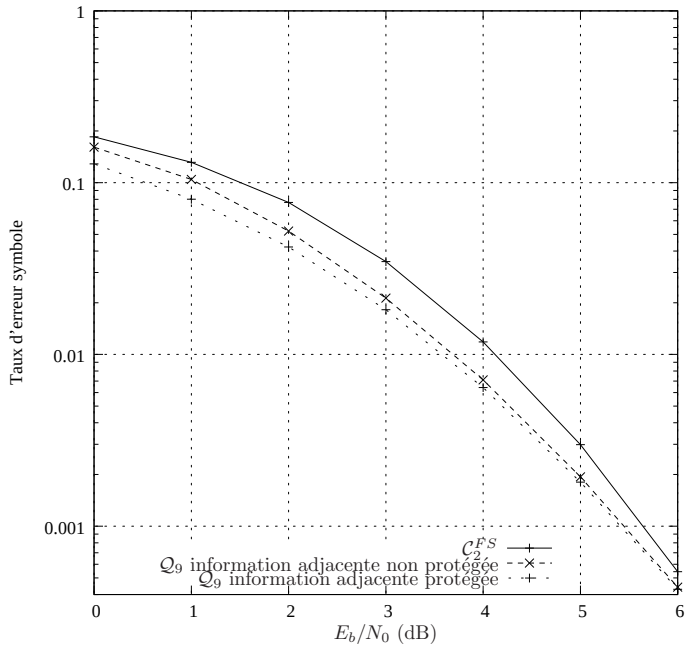

FIG. 3.21 – Taux d'erreur symbole en fonction de E_b/N_0 pour les codes $\mathcal{Q}_8$ et $\mathcal{Q}_8^{FS}$. Le taux de transmission est de 1.045 bits par symbole.

deur sur un modèle de paramètre T. Nous verrons alors qu'il est possible de prévoir les variations de performances des codes en fonction de T, ainsi que d'estimer la valeur de T, pour un code et un rapport signal à bruit donné, qui permet d'atteindre les performances optimales.

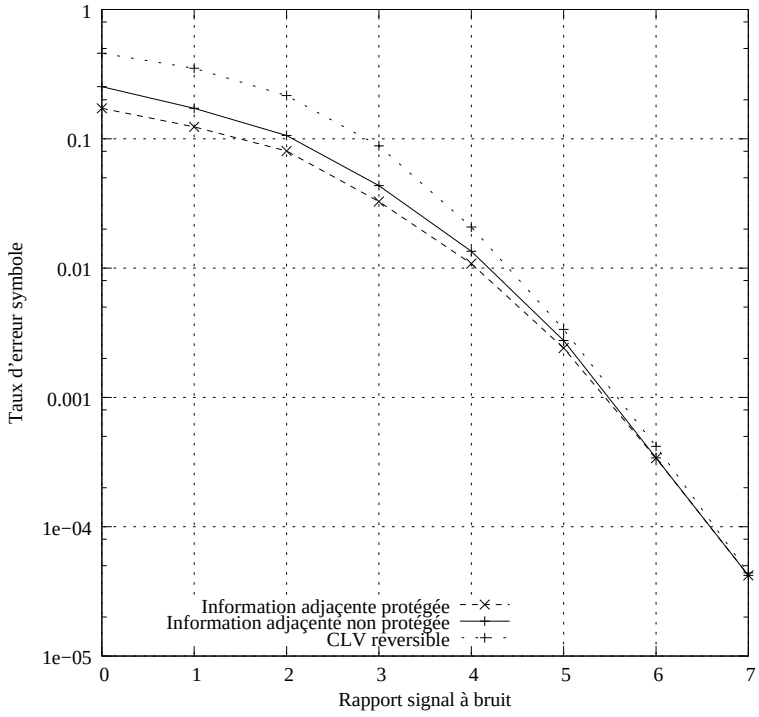

FIG. 3.22 – Taux d'erreur symbole en fonction de E_b/N_0 pour les codes $\mathcal{C}_{10}$ et $\mathcal{C}_{10}^R$. Le taux de transmission est de 2.3 bits par symbole.

Chapitre 4

Lien entre les propriétés de resynchronisation des codes et les performances sur modèle agrégé

"Celui-là est très-très-très bon, Valérie"

Didier Bourdon

À l'intérieur d'un train binaire issu de l'encodage d'une séquence de symboles par un CLV ou un code QA, les séparations entre les mots de code dépendent de la séquence source originale. Ces séparations ne sont pas connues par le décodeur si aucune information supplémentaire (autre que le train binaire en lui-même) ne lui est fourni. Cette caractéristique des codes à longueur variable les rend très sensibles aux erreurs de transmission. En effet, même une simple erreur bit peut provoquer une désynchronisation au niveau du décodeur, et de forts taux d'erreur symboles en sortie.
Prenons par exemple un code à longueur variable utilisé dans [MR85]. Ce code est illustré sur la figure 4.1. On envoie le message constitué des symboles $a_1\ a_4\ a_3\ a_2\ a_3$. Le train binaire associé à ce message est $00\,110\,10\,01\,10$. Si le troisième bit de ce train binaire est inversé par le canal, la séquence de symboles décodés sera alors $a_1\ a_2\ a_2\ a_1\ a_4$. Les états de l'encodeur et du décodeur associés à cet exemple sont représentés dans le tableau 4.1. L'encodeur et le décodeur se désynchronisent sous l'effet de l'erreur bit, et ne se resynchronisent que 8 bits après cette erreur. L'erreur bit s'est donc propagée et a engendré le remplacement de la séquence $a_1\ a_4\ a_3\ a_2\ a_3$ par $a_1\ a_2\ a_2\ a_1\ a_4$, entraînant donc 4 erreurs symboles.
Supposons maintenant qu'une erreur bit se produise en septième position dans le train binaire de ce même message. Dans le tableau 4.2, sont représentés les états de l'encodeur et du décodeur associés à cette transmission. Dans ce cas, la séquence $a_1\ a_4\ a_4\ a_4$

sera décodée. Ainsi, la séquence décodée comporte moins de symboles que la séquence originale. La resynchronisation entre l'encodeur et le décodeur n'est, dans ce cas, pas du même type que lors de l'exemple ci-dessus puisqu'a également eu lieu un décalage de l'horloge symbole (on a ici "perdu" un symbole).
Ces deux exemples illustrent les deux types de resynchronisation qu'il convient de distinguer :

– Resynchronisation **forte** [KN00] : l'encodeur et le décodeur sont à nouveau dans le même état (pas de décalage de l'horloge symbole)
– Resynchronisation **faible** [KN00] : l'encodeur et le décodeur sont dans le même état interne de l'arbre, mais le nombre de symboles décodés est différent du nombre de symbole encodés.

Deux quantités apparaissent donc pour l'analyse des propriétés de resynchronisation des CLV :

1. la longueur de propagation d'une erreur bit
2. la différence entre le nombre de symboles encodés et décodés.

Dans [MR85], les auteurs ont proposé une méthode pour calculer l'espérance de la longueur de propagation d'une erreur bit (appelée *expected error span* par les auteurs). Dans ce qui suit, nous appellerons E_s la variable aléatoire représentant la longueur de propagation d'une erreur bit. Sa moyenne sera appelée MEPL (*Mean Error Propagation Length*) comme dans [ZZ02] et sa variance VEPL (*Variance of Error Propagation Length*). Dans [ML87], les auteurs ont étendu la méthode de Maxted et Robinson afin de calculer le VEPL d'un code. Les auteurs de [SD95] ont ensuite adapté la méthode proposée dans [MR85] afin de calculer la densité de probabilité (ddp) de la variable aléatoire représentant la différence entre le nombre de symboles encodés et décodés suite à une erreur bit. Nous appellerons cette quantité *gain/loss* et la représenterons par la variable ΔS dans tout ce qui suit.

Dans ce chapitre, les méthodes permettant de calculer le MEPL et la loi de probabilité du *gain/loss* pour un code donné seront présentées. Ces méthodes ont été établies pour des CLV, et dans le cas où une seule erreur bit se produit lors de la transmission. Elles sont basées sur le calcul d'une fonction de transfert sur un diagramme d'état appelé *error state diagram* (ESD). Une approche alternative a été proposée dans [ZZ02], utilisant un calcul matriciel. Ces différentes méthodes seront détaillées dans ce chapitre. Les auteurs de [ZZ02] ont montré que le MEPL permettait de refléter les performances des CLV lorsqu'un décodage dur est appliqué au décodeur. Plus le MEPL est élevé, moins un CLV est robuste en termes de performances de décodage. Cependant, cette variable n'est pas adaptée si un décodage souple avec contrainte de longueur est appliqué au décodeur.

Les méthodes de calcul du MEPL et de la loi de probabilité de ΔS ont été créées pour

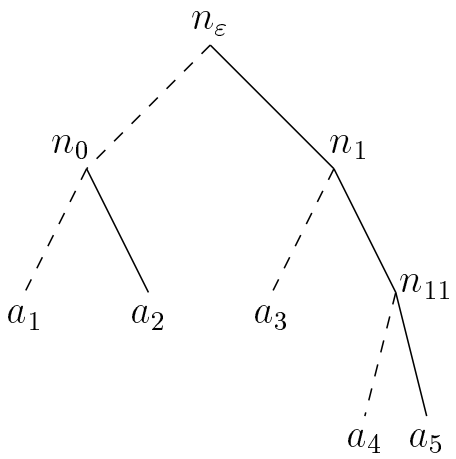

FIG. 4.1 – Code à longueur variable proposé dans [MR85]

les CLV. Afin d'analyser les performances des codes QA après décodage souple sur le modèle agrégé, nous allons d'abord proposer une extension des méthodes de [MR85] et [SD95] afin de calculer le MEPL et la loi de probabilité de ΔS pour les codes QA. Cette extension a été presentée dans [MJG08b]. Pour ce faire, un nouveau type d'ESD a été créé.

Le calcul de la ddp de ΔS selon la méthode de [SD95] n'est valable que lorsqu'une seule erreur bit se produit. Pour évaluer les propriétés de resynchronisation des codes sur un canal de type BABG, il est nécessaire de calculer la ddp de ΔS lorsque de multiples erreurs se produisent. Nous proposons dans ce chapitre une méthode d'estimation de la ddp de ΔS lorsque le train binaire est émis sur un canal binaire symétrique (CBS).

Dans la dernière partie de ce chapitre, nous montrerons que la ddp de ΔS permet d'analyser les performances des codes sur le modèle agrégé présenté dans le chapitre 3. Cette analyse a été proposée dans [MJG07]. Nous verrons notamment que l'on peut prévoir quelles valeurs du paramètre T d'agrégation sont à éviter pour un code donné. Nous dériverons également une borne supérieure sur la valeur de T permettant d'approcher les performances du modèle optimal.

4.1 Calcul de l'espérance de E_s pour les CLV

Pour calculer l'espérance de la longueur de propagation d'une erreur bit, les auteurs de [MR85] ont modélisé à l'aide d'un diagramme le comportement du décodeur suivant une erreur bit. Ce diagramme est appelé *error state diagram* (ESD). Il est com-

Symbole émis	État interne de l'encodeur	Bit émis	Bit reçu	État interne du décodeur	Symbole décodé
	n_0	0	0	n_0	
a_1	n_ε	0	0	n_ε	a_1
	n_1	1	0	n_0	
	n_{11}	1	1	n_ε	a_2
a_4	n_ε	0	0	n_0	
	n_1	1	1	n_ε	a_2
a_3	n_ε	0	0	n_0	
	n_0	0	0	n_ε	a_1
a_2	n_ε	1	1	n_1	
	n_1	1	1	n_{11}	
a_3	n_ε	0	0	n_ε	a_4

TAB. 4.1 – Propagation d'une erreur bit avec resynchronisation forte

posé des état internes de l'arbre du CLV, ainsi que de deux états supplémentaires n_l et n_s représentant respectivement la perte de synchronisation et la resynchronisation. A l'aide de ce diagramme, sont représentées toutes les réalisations possibles de l'état du décodeur lorsque l'encodeur est dans le noeud racine de l'arbre. L'état n_s correspond à un retour de l'automate d'encodage et de celui de décodage au noeud racine n_ε. Lorsque cet état est atteint, le décodeur est donc resynchronisé avec l'encodeur. Cependant, cet état ne correspond pas forcément à une resynchronisation forte. En effet, le nombre de symboles décodés pendant la désynchronisation peut être différent du nombre de symboles encodés. Les transitions entre les états de ce diagramme correspondent à l'émission d'un symbole. Les probabilités de ces transitions sont donc calculées à l'aide de la loi de probabilité de la source.

4.1.1 Élaboration de l'error state diagram

L'état initial de l' ESD est l'état n_l. Cet état est en fait le noeud racine de l'arbre du CLV juste avant que l'erreur bit n'intervienne dans le train binaire. Ainsi, on considère que le mot de code associé au prochain symbole reçu comporte une erreur bit. Cette erreur bit peut avoir deux répercussions différentes dans l'ESD :

1. l'erreur dans le mot de code du symbole courant l'a transformé en un autre (ou d'autres) mot(s) de code valide(s)
2. l'erreur dans le mot de code courant l'a transformé en un mot de code invalide (préfixe ou mot(s) de code suivi(s) d'un préfixe).

Le premier cas décrit ci-dessus interviendra par exemple si le deuxième bit du symbole a_1 du code $\mathcal{C}_1$ (Fig. 4.1) est erroné. En effet, dans ce cas le symbole a_2 sera décodé à

Symbole émis	État interne de l'encodeur	Bit émis	Bit reçu	État interne du décodeur	Symbole décodé
	n_0	0	0	n_0	
a_1	n_ε	0	0	n_ε	a_1
	n_1	1	1	n_1	
	n_{11}	1	1	n_{11}	
a_4	n_ε	0	0	n_ε	a_4
	n_1	1	1	n_1	
a_3	n_ε	0	1	n_{11}	
	n_0	0	0	n_ε	a_4
a_2	n_ε	1	1	n_1	
	n_1	1	1	n_{11}	
a_3	n_ε	0	0	n_ε	a_4

TAB. 4.2 – Propagation d'une erreur bit avec resynchronisation faible

la place d'a_1 et le décodeur sera dans le noeud n_ε (il y a eu resynchronisation). Si, par contre, le deuxième bit du symbole a_4 est erroné (les bits $1\,0\,1$ sont transmis au décodeur), le symbole a_3 est décodé (bits $1\,0$) mais le décodeur sera dans l'état n_1. En effet, le bit 1 restant ne correspond pas à un mot de code valide.
Ainsi, à la suite du décodage du mot de code erroné, le décodeur sera soit dans un des états interne de l'arbre du CLV $n_{\alpha_1}, n_{\alpha_2}, \ldots$ (cas 2 ci dessus), soit dans l'état de resynchronisation n_s (cas 1 ci dessus).

Exemple 4.1: Les transitions partant de l'état initial n_l pour le code $\mathcal{C}_1$ sont indiquées dans le tableau 4.3. Tous les motifs d'erreur possibles dans un mot de code ont été considérés, ainsi que la probabilité de la transition associée. Ces probabilités sont égales à la probabilité du mot de code sur lequel l'erreur se produit divisée par la longueur de description moyenne du code (ici 2.2). En regroupant les transitions arrivant dans le même état, on obtient en partant de n_l quatre transitions possibles décrites dans le tableau 4.4.

Il est ensuite nécessaire de calculer les transitions partant de tous les états de l'ESD atteints à partit de n_l, hormis l'état n_s qui correspond à la resynchronisation du décodeur. Dans un état n_{α_i}, le préfixe α_i n'a pas encore été décodé. Ce préfixe précédera donc les bits du prochain mot de code. Il y a autant de transitions possibles partant d'un état n_{α_i} dans l'ESD que de symboles dans l'alphabet du code. Les transitions arrivant dans un même état du diagramme sont toutefois regroupées.

Exemple 4.2: Le tableau 4.5 détaille le calcul des probabilités des transitions partant de l'état n_{11}. Être dans l'état n_{11} signifie que les bits 11 n'ont pas encore été décodés et

Mot de code	Bit erroné	État atteint	Probabilité de transition
00	1	n_s	4/22
00	2	n_s	4/22
01	1	n_{11}	2/22
01	2	n_s	2/22
10	1	n_s	2/22
10	2	n_{11}	2/22
110	1	n_0	1/22
110	2	n_0	1/22
110	3	n_s	1/22
111	1	n_1	1/22
111	2	n_1	1/22
111	3	n_s	1/22

TAB. 4.3 – Élaboration des transitions initiales du diagramme pour le code $\mathcal{C}_1$

État de départ	État d'arrivée	Probabilité de transition
n_l	n_s	7/11
n_l	n_0	1/11
n_l	n_1	1/11
n_l	n_{11}	2/11

TAB. 4.4 – Probabilités des transitions partant de n_l

donc qu'ils précéderont les bits du mot de code suivant. On obtient par exemple

$$\mathbb{P}(n_{11} \rightarrow n_0) = \mathbb{P}(a_1) + \mathbb{P}(a_3) = 0.6 \tag{4.1}$$

Ce processus est répété pour tous les états d'erreur du diagramme. L'ESD associé au code $\mathcal{C}_1$ est représenté sur la figure 4.2. Nous allons maintenant voir comment calculer l'*error span* à partir d'un tel diagramme.

4.1.2 Calcul de l'espérance de E_s

Chaque transition dans l'ESD correspond à l'encodage d'un symbole du message original. Ainsi, le nombre de transitions effectuées entre l'état n_l et l'état n_s est égal au nombre de symboles source sur lesquels l'erreur s'est propagée. Pour calculer l'espérance de E_s, Maxted et Robinson ont multiplié chaque probabilité de transition par une variable indéterminée z, qui correspond à l'envoi d'un symbole de source sur

État d'erreur	Mot de code suivant	Train binaire résultant	Symboles décodés	État suivant
11	00	1100	a_4	n_0
11	01	1101	a_4	n_1
11	10	1110	a_5	n_0
11	110	11110	a_5a_3	n_s
11	111	11111	a_5	n_{11}

TAB. 4.5 – Probabilité des transitions partant de n_{11}.

chaque transition. Ainsi la fonction de transfert du diagramme entre n_l et n_s est un polynôme G de la variable z dont les coefficients de z^k sont égaux aux probabilités que le CLV considéré resynchronise exactement i symboles de source après l'erreur de transmission. On obtient donc

$$G(z) = \sum_{k\in\mathbb{N}} g_k\, z^k, \tag{4.2}$$

où $g_k = \mathbb{P}(E_s = k)$.
Ainsi, le polynôme dérivé de G pris en $z = 1$ est égal à

$$G'(z)|_{z=1} = \sum_{k\in\mathbb{N}} k\,\mathbb{P}(E_s = k), \tag{4.3}$$

ce qui correspond à la longueur moyenne de propagation d'une erreur bit. Par conséquent, la dérivée en 1 de ce polynôme est égale à l'*expected error span*. Le calcul de cette quantité nécessite donc celui de la fonction de transfert du diagramme. Cette fonction de transfert peut se calculer en utilisant la formule de Mason [Mas56], ou par réduction de graphes. Ce calcul n'étant pas systématique, la méthode matricielle suivante présente l'avantage d'être plus rapide et directe.

4.1.3 Méthode matricielle

La méthode matricielle de calcul de l'espérance de E_s est basée sur les résultats présentés dans [ZZ02]. Elle se différencie de la méthode de [MR85] uniquement pour le calcul de l'espérance de E_s. L'ESD nécessite d'être construit de la même manière que précédemment. Ce diagramme peut être mis sous la forme de la matrice de transition suivante :

$$\overline{H} = \begin{pmatrix} \mathbb{P}(n_l \to n_l) & \mathbb{P}(n_{\alpha_1} \to n_l) & \cdots & \mathbb{P}(n_s \to n_l) \\ \mathbb{P}(n_l \to n_{\alpha_1}) & \mathbb{P}(n_{\alpha_1} \to n_{\alpha_1}) & \cdots & \mathbb{P}(n_s \to n_{\alpha_1}) \\ \vdots & \vdots & \ddots & \vdots \\ \mathbb{P}(n_l \to n_s) & \mathbb{P}(n_{\alpha_1} \to n_s) & \cdots & \mathbb{P}(n_s \to n_s) \end{pmatrix} \tag{4.4}$$

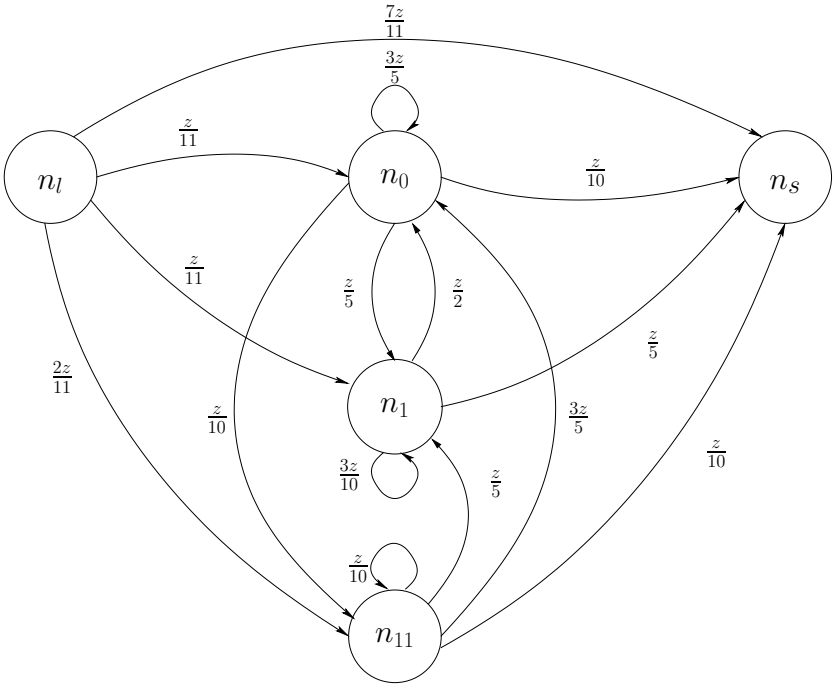

FIG. 4.2 – ESD associé au code $\mathcal{C}_1$

Notez que les probabilités de transition dans cette matrice comportent leur étiquetage (ici z) afin de calculer l'espérance de E_s. Appelons $h^k_{i,j}(z)$ l'élément placé à la ligne i et à la colonne j de la matrice $\overline{H}^k$. $h^k_{i,j}(z)$ représente la probabilité d'aller de l'état n_{α_i} à l'état n_{α_j} en k étapes, i.e. après décodage de k symboles source dans le cas présent. L'élément en haut et à droite de $\overline{H}^k$ est noté $h^k(z)$. Il représente la probabilité d'aller de n_l à n_s après décodage de k symboles sources. Ainsi, le polynôme $G(z)$ de l'équation 4.2 peut également être obtenu comme suit :

$$G(z) = \sum_{k \in \mathbb{N}^*} h^k(z). \tag{4.5}$$

Si la matrice $(\overline{I} - \overline{H})$, où $\overline{I}$ est la matrice identité de mêmes dimensions que $\overline{H}$, est inversible, alors $G(z)$ est également l'élément situé en haut à droite de la matrice $(\overline{I} - \overline{H})^{-1}$.

Exemple 4.3: La matrice de transition du code $\mathcal{C}_1$ s'écrit :

$$\overline{H}_{\mathcal{C}_1} = \begin{pmatrix} 0 & \frac{z}{11} & \frac{z}{11} & \frac{2z}{11} & \frac{7z}{11} \\ 0 & \frac{3z}{5} & \frac{z}{5} & \frac{z}{10} & \frac{z}{10} \\ 0 & \frac{z}{2} & \frac{3z}{10} & 0 & \frac{z}{5} \\ 0 & \frac{3z}{5} & \frac{z}{5} & \frac{z}{10} & \frac{z}{10} \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

Le polynôme $G(z)$ est alors :

$$G(z) = \frac{z(175 - 35z + 36z^2)}{11(z-5)^2} \tag{4.6}$$

Remarquez que $G(1) = 1$, ce qui signifie que la probabilité que le code $\mathcal{C}_1$ se resynchronise après une erreur bit est égale à 1. L'espérance de E_s est obtenue en dérivant $G(z)$ et en évaluant sa dérivée pour $z = 1$.

$$E_s = 301/176 = 1.7102 \tag{4.7}$$

L'espérance de la longueur de propagation d'une erreur simple est un critère de performance des CLV lorsqu'un décodage **dur** est appliqué au décodeur[ZZ02]. Plus E_s est faible, meilleures sont les performances en terme de résistance aux erreurs (pour le TES) du CLV considéré.

Remarque 4.1 *La longueur moyenne de propagation d'une erreur bit telle que nous venons de la calculer représente le nombre moyen de symboles* **source** *sur lesquels l'erreur se propage. La méthode présentée ci-dessus permet également de calculer le nombre moyen de symboles* **décodés** *sur lesquels l'erreur se propage. Pour cela, la variable z sur chacune des transitions de l'ESD doit être modifiée en z^i ou i est égal au nombre de symboles décodés par chacune des transitions.*

4.2 Calcul de la ddp de ΔS pour les CLV

Les travaux présentés dans la section précédente, initiés en grande partie dans [MR85], ne permettent pas de calculer la ddp de la variable aléatoire ΔS, représentant la différence entre le nombre de symboles encodés et décodés. Swaszek et Di Cicco dans [SD95] ont étendu la méthode présentée ci-dessus afin de calculer la ddp de ΔS pour un CLV donné, et dans le cas où une seule erreur bit interviendrait dans le train binaire reçu par le décodeur. La méthode proposée dans [SD95] est détaillée dans cette section. Elle est également basée sur la construction d'un *error state diagram* (ou de la représentation matricielle de celui-ci). Cependant, afin de garder une trace d'un possible décalage entre le nombre de symbole encodés et décodés le long du diagramme, l'étiquetage des transitions a été modifié. En effet, ces transitions sont maintenant étiquetées par une variable indéterminée y^j, où l'exposant est égal au nombre de symboles décodés supplémentaires (positif ou négatif) sur la transition correspondante.
Dans le tableau 4.6, l'exposant j associé aux transitions partant de n_{11} est indiqué. Pour chaque transition, il est égal à 1 moins le nombre de symboles décodés sur cette transition (le nombre de symbole encodés sur une transition est toujours égal à 1 pour l'ESD d'un CLV).

État d'erreur	Mot de code suivant	Train binaire résultant	Symboles décodés	État suivant	Exposant j
11	00	1100	a_4	n_0	$j = 1 - 1 = 0$
11	01	1101	a_4	n_1	$j = 1 - 1 = 0$
11	10	1110	a_5	n_0	$j = 1 - 1 = 0$
11	110	11110	a_5a_3	n_s	$j = 1 - 2 = -1$
11	111	11111	a_5	n_{11}	$j = 1 - 1 = 0$

TAB. 4.6 – Calcul de l'exposant j associé aux transitions partants de n_{11}.

En utilisant cet étiquetage des transitions, la fonction de transfert du diagramme entre les états n_l et n_s est maintenant un polynôme en y :

$$H(y) = \sum_{i \in \mathbb{Z}} h_i y^i, \tag{4.8}$$

où

$$h_i = \mathbb{P}(\Delta S = i). \tag{4.9}$$

Rappelons que ces coefficients h_i représentent la ddp de ΔS dans le cas où **une** seule erreur bit intervient dans le message.

Exemple 4.4: L'ESD associé au code $\mathcal{C}_1$ pour le calcul de la ddp de ΔS est indiqué sur la figure 4.3. Le calcul de la fonction de transfert de ce diagramme permet d'obtenir

$$H(y) = 0.0992y^{-1} + 0.8350 + 0.0658y, \tag{4.10}$$

ce qui signifie également :

$$\begin{cases} \mathbb{P}(\Delta S = -1) & = 0.1023 \\ \mathbb{P}(\Delta S = 0) & = 0.8352 \\ \mathbb{P}(\Delta S = 1) & = 0.0625. \end{cases} \tag{4.11}$$

Par conséquent, le code $\mathcal{C}_1$ présente une forte probabilité de resynchronisation forte suite à une erreur bit. Il est à noter également qu'une erreur ne peut provoquer un décalage de l'horloge que d'au plus 1 symbole en valeur absolue.

Nous venons donc de voir les méthodes permettant de calculer la MEPL et la ddp de la variable ΔS pour les CLV et dans le cas où une seule erreur bit intervient lors de la transmission. Dans la section 4.3, nous allons étendre ces résultats afin de pouvoir calculer ces deux quantités pour des codes QA.

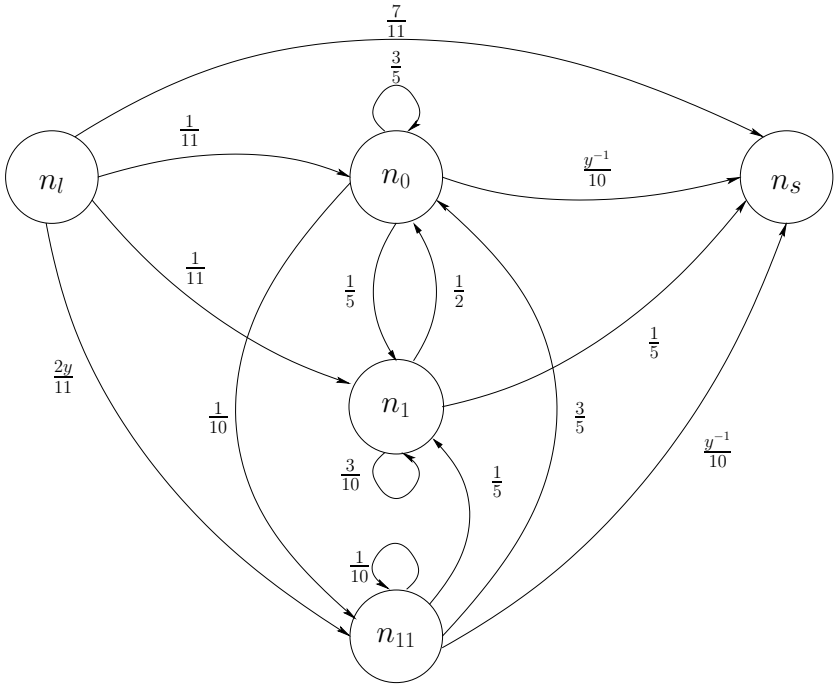

FIG. 4.3 – ESD associé au code $\mathcal{C}_1$ pour le calcul de la d.d.p de ΔS

4.3 Propriétés de synchronisation des codes quasi-arithmétique

Il a été expliqué dans le chapitre 2 que les codes QA pouvaient se mettre sous la forme d'automates (ou de machines à états finis), au même titre que les CLV. Cependant, les méthodes proposées dans [MR85] et [SD95] pour calculer respectivement l'espérance de la longueur de propagation d'une erreur bit et la ddp de la variable ΔS ne peuvent s'appliquer directement pour les codes QA. En effet, les arbres de CLV (de la même façon les automates des CLV) présentent la particularité d'avoir une transition vers le noeud racine n_ϵ à chaque symbole encodé (ou décodé). Cette particularité des CLV est exploitée pour construire l'ESD qui a été présenté dans les sections 4.1 et 4.2. En effet, chaque transition dans l'ESD correspond à une émission de symboles (et donc à un retour de l'automate d'encodage au noeud racine de l'arbre). Il suffit donc que le décodeur se trouve dans le noeud n_ϵ après décodage d'un symbole pour que l'encodeur et le décodeur soient resynchronisés. Les automates de codes QA ne présentent pas cette particularité, l'automate ne revenant pas forcément dans le même état après chaque symbole émis. Dans cette section, nous présenterons un nouveau type d'ESD adapté pour le calcul des propriétés de resynchronisation des codes QA, ainsi que la méthode pour calculer l'espérance de la longueur de propagation d'une erreur bit et la ddp de ΔS pour un code QA donné.

4.3.1 Error state diagram adapté aux codes QA

Dans le chapitre 2, nous avons montré comment un code QA pouvait se mettre sous la forme d'une machine à états finis (ou d'un automate). Plus précisément, un code QA peut être défini par deux automates : un pour l'encodage et un pour le décodage. Dans cette section, l'automate de décodage des codes QA sera utilisé afin de dériver un ESD adapté à ces codes. Rappelons que l'automate de décodage d'un code QA est un ensemble d'états $\Omega_d = \{\beta_0, \ldots, \beta_{N_d-1}\}$. Les transitions entre ces états comportent un bit en entrée et un ou plusieurs symboles en sortie. Une transition peut également ne pas comporter de symboles en sortie, une telle transition est appelée transition muette. Des exemples d'automate de décodage de codes QA ont été donnés dans le chapitre 2.

Nous nous plaçons dans le cas où une seule erreur bit intervient dans le train binaire. Nous allons considérer que les messages $\mathbf{X}$ et $\mathbf{Y}$, correspondant respectivement au message transmis et au message reçu, diffèrent à la position k_p. Cela signifie que le $k_p^{\text{ème}}$ bit de $\mathbf{X}$ a été inversé par le canal. On note $(N_k^{\mathbf{X}}, N_k^{\mathbf{Y}})$ le couple de variables aléatoires représentant l'état de l'automate de décodage après décodage des k premiers bits de $\mathbf{X}$ et $\mathbf{Y}$ respectivement. Les réalisations de ce couple de variables aléatoires sont notées $(n_k^{\mathbf{X}}, n_k^{\mathbf{Y}})$. L'erreur bit dans le train binaire $\mathbf{Y}$ intervenant à la position k_p, nous avons :

$$\forall k < k_p,\ n_k^{\mathbf{X}} = n_k^{\mathbf{Y}}. \tag{4.12}$$

L'erreur bit en position k_p dans $\mathbf{Y}$ peut éventuellement provoquer $n_{k_p}^{\mathbf{X}} \neq n_{k_p}^{\mathbf{Y}}$, c'est à dire la désynchronisation du décodeur. La resynchronisation aura lieu à la position bit k_r, k_r étant défini comme

$$k_r = \min_{k_p \leq k \leq L(\mathbf{X})} k \,|\, n_k^{\mathbf{X}} = n_k^{\mathbf{Y}} \tag{4.13}$$

L'ESD adapté aux codes QA représente toutes les réalisations possibles du couple $(n_k^{\mathbf{X}}, n_k^{\mathbf{Y}})$ à partir de l'erreur bit (instant bit $k_p - 1$) jusqu'à la resynchronisation. La resynchronisation du décodeur sera différente selon l'état dans lequel est l'automate de décodage au moment où l'erreur bit intervient. Il est ainsi nécessaire de construire N_d diagrammes, soit un pour chaque élément de l'ensemble Ω_d des états de l'automate de décodage. L'état n_l (état représentant la perte de synchronisation) qui apparaît dans les ESD adaptés aux CLV est représenté pour les codes QA par l'état β_i, $\beta_i \in \Omega_d$. Les états finaux du diagramme adapté aux codes QA (représentants la resynchronisation) sont les états du type $(\beta_i, \beta_i), \beta_i \in \Omega_d$. Il y a ainsi autant d'états finaux possibles dans le diagramme que d'états dans l'automate de décodage du code QA. Quand un de ces états finaux est atteint, le décodeur est resynchronisé.

Exemple 4.5: Les transitions de l'ESD d'état initial β_0 associé au code $\mathcal{Q}_7$ ainsi que leurs probabilités sont représentées dans le tableau 4.7. L'ESD correspondant est représenté sur la figure 4.4. On peut remarquer que si une erreur bit intervient dans l'état β_0 du

État du diagramme	Bit envoyé	Bit reçu	État suivant	Probabilité
β_0	0	1	(β_1, β_2)	p
β_0	1	0	(β_2, β_1)	$1-p$
(β_1, β_2)	0	0	(β_1, β_1)	p^2
(β_1, β_2)	1	1	(β_3, β_4)	$1-p^2$
(β_2, β_1)	0	0	(β_1, β_1)	p^2
(β_2, β_1)	1	1	(β_4, β_3)	$1-p^2$
(β_3, β_4)	0	0	(β_2, β_2)	$p/(1+p)$
(β_3, β_4)	1	1	(β_0, β_0)	$1/(1+p)$
(β_4, β_3)	0	0	(β_2, β_2)	$p/(1+p)$
(β_4, β_3)	1	1	(β_0, β_0)	$1/(1+p)$

TAB. 4.7 – Transitions et probabilités de transitions pour l'ESD d'état initial β_0

décodeur du code $\mathcal{Q}_7$, la resynchronisation aura lieu au maximum 3 bits après l'erreur.

4.3.2 Calcul de la d.d.p de ΔS pour les codes QA

En utilisant un ESD adapté aux codes QA, le calcul de la ddp de ΔS est inspiré de la méthode proposée pour les CLV par les auteurs de [SD95]. Les probabilités de transitions sur l'ESD sont étiquetées avec la variable indéterminée y^j, où l'exposant j représente la différence entre le nombre de symboles encodés et décodés le long de cette transition. Cet exposant peut être, comme pour les CLV, positif ou négatif. Sur l'ESD de la figure 4.4, les exposants sur chaque transition sont égaux à 0. Cela signifie que, pour le code $\mathcal{Q}_7$, si une erreur bit intervient alors que le décodeur se trouve dans l'état β_0, la resynchronisation sera forte (ΔS sera toujours égal à 0). Cette particularité n'est pas forcément vraie si l'erreur intervient dans un autre état du décodeur.

En suivant la méthode expliquée ci-dessus, N_d diagrammes sont construits (un pour chaque état du décodeur). Pour $\beta_j \in \Omega_d$, la fonction de transfert entre β_j et S de l'ESD d'état initial β_j est un polynôme, noté G_{β_j}, de la variable z :

$$G_{\beta_j}(z) \triangleq \sum_{k \in \mathbb{Z}} g_{\beta_j,k} z^k. \tag{4.14}$$

Cette fonction de transfert est calculée de la même façon que pour les CLV. La méthode matricielle présentée dans la section 4.1 peut également être utilisée. D'après la définition de l'ESD, les coefficients $g_{\beta_j,k}$ de G_{β_j} sont égaux à $\mathbb{P}(\Delta S = k | N^{\mathbf{X}}_{k_p-1} = \beta_j)$. On définit maintenant le polynôme G comme

$$G(z) \triangleq \sum_{k \in \mathbb{Z}} g_k z^k = \sum_{\beta_j \in \Omega_d} G_{\beta_j}(z) \mathbb{P}(N^{\mathbf{X}}_{k_p-1} = \beta_j). \tag{4.15}$$

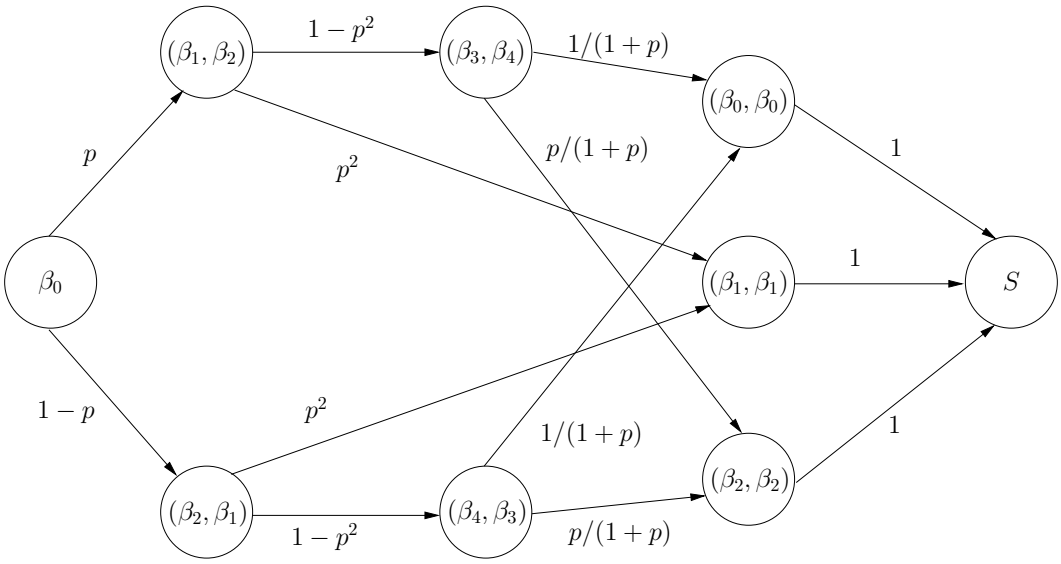

FIG. 4.4 – ESD d'état initial β_0 associé au code $\mathcal{Q}_7$

Les probabilités $\mathbb{P}(N^{\mathbf{X}}_{k_p-1} = \beta_j)$ sont calculées à partir du vecteur propre associé à la valeur propre 1 de la matrice de transition de l'automate de décodage, comme expliqué dans le chapitre 2. D'après la définition du polynôme G, les coefficients g_k sont égaux à $\mathbb{P}(\Delta S = k)$.

Remarque 4.2 *La longueur moyenne de propagation d'une erreur bit peut également être calculée pour les codes QA en utilisant l'ESD présenté dans cette section. Pour cela, l'étiquetage des transitions doit être remplacé par $z^{j'}$, où l'exposant j' est égal au nombre de symboles décodés sur la transition correspondante. La méthode présentée dans le section 4.1 s'applique ensuite afin de calculer la valeur moyenne de E_s.*

Nous venons donc de présenter un nouveau type d'ESD adapté aux codes QA et qui permet de calculer leurs propriétés de resynchronisation. Nous pouvons donc maintenant calculer pour un code QA donné, l'espérance de la longueur de propagation d'une erreur bit, ainsi que la densité de probabilité de ΔS suite à une erreur bit.

Afin d'analyser les performances de décodage des CLV et des codes QA sur le modèle d'état agrégé présenté dans le chapitre 3, il nous est nécessaire d'avoir une estimation de la ddp de ΔS dans le cas où le train binaire original est transmis sur un CBS, caractérisé par sa probabilité de transition π.

4.4 Estimation de la ddp de ΔS sur un CBS

Dans cette section, nous proposons une méthode d'estimation de la ddp de ΔS dans le cas où le train binaire issu de l'encodage d'un message de $L(\mathbf{S})$ symboles par un CLV ou un code QA est transmis sur un CBS de probabilité de transition π. Pour faire ce calcul, nous supposerons que le décodeur utilise une information dure sur les bits reçus (de façon identique au calcul pour une erreur bit présenté ci-dessus). Ainsi, cette estimation est également valide pour un canal BABG caractérisé par son rapport signal à bruit E_b/N_0 en prenant $\pi = \frac{1}{2}\,\mathrm{erfc}\left(\sqrt{\frac{E_b}{N_0}}\right)$.

Le message original, composé de $L(\mathbf{S})$ symboles, est encodé par un CLV ou un code QA en un message binaire $\mathbf{X}$ de $L(\mathbf{X})$ bits. $\mathbf{X}$ est transmis sur un CBS de probabilité de transition π. Le décodeur dispose du message $\mathbf{Y}$ en sortie du CBS. On appelle E la variable aléatoire correspondant au nombre d'erreurs bit dans le train binaire Y. $\forall e \in \mathbb{N}$, on a :

$$\mathbb{P}(E = e) = \sum_{i \in \mathbb{N}} \mathbb{P}(E = e \,|\, L(\mathbf{X}) = i)\mathbb{P}(L(\mathbf{X}) = i). \tag{4.16}$$

La probabilité $\mathbb{P}(E = e \,|\, L(\mathbf{X}) = i)$ est fonction uniquement de la probabilité π de transition du CBS et peut s'écrire :

$$\mathbb{P}(E = e \,|\, L(\mathbf{X}) = i) = \begin{cases} \binom{i}{e} \pi^e (1-\pi)^{i-e} & \text{si } e \leq i \\ 0 & \text{si} e > i. \end{cases} \tag{4.17}$$

Pour calculer la loi de probabilité de la variable aléatoire E, il faut également calculer les probabilités $\mathbb{P}(L(X) = i)$. Si un code à longueur variable est utilisé à l'encodage, la probabilité $\mathbb{P}(L(\mathbf{X}) = i)$ s'obtient directement à partir des probabilités de la source et de la structure du code. En revanche, cette probabilité ne peut pas se calculer directement pour un code QA. Une estimation de cette probabilité est proposée ci-dessous.

4.4.1 Estimation de la ddp de $L(\mathbf{X})$ pour un code QA

Le calcul de la longueur de description moyenne d'un code QA est proposé dans [Gor94]. Nous allons l'utiliser pour estimer la ddp de $L(\mathbf{X})$. On appelle, pour tout état α_i dans l'ensemble Ω_e des états de l'automate d'encodage du code QA, $\bar{l}_{\alpha_i}$ le nombre moyen de bits émis par les transitions issues de l'état α_i. $\bar{l}_{\alpha_i}$ est égal à :

$$\bar{l}_{\alpha_i} = \mathbb{P}(a) \times o_{\alpha_i,a} + (1 - \mathbb{P}(a)) \times o_{\alpha_i,b}, \tag{4.18}$$

où $o_{\alpha_i,a}$ et $o_{\alpha_i,b}$ représentent le nombre de bits émis par les transitions issues de α_i générées par les symboles a et b respectivement. Ainsi, la longueur de description moyenne $\bar{l}$ d'un code QA est donnée par

$$\bar{l} = \sum_{\alpha_i \in \Omega_e} \bar{l}_{\alpha_i} \mathbb{P}(\alpha_i). \tag{4.19}$$

Pour l'estimation de la ddp de $L(\mathbf{X})$, on va supposer que cette densité de probabilité est une densité de type gaussienne de moyenne $\bar{l} \times L(\mathbf{S})$ et dont la variance est estimée ci-après. En notant

$$v_{\alpha_i}^2 \triangleq p \times (o_{\alpha_i,a} - \bar{l})^2 + (1-p) \times (o_{\alpha_i,b} - \bar{l})^2, \tag{4.20}$$

la variance du nombre de bits émis par symbole source du code QA est donnée par

$$v^2 = \sum_{\alpha_i \in \Omega_e} v_{\alpha_i}^2 \mathbb{P}(\alpha_i). \tag{4.21}$$

La d.d.p de $L(\mathbf{X})$ est ainsi approchée par une densité gaussienne de moyenne $\bar{l} \times L(\mathbf{S})$ et de variance $v \times L(\mathbf{S})$.

4.4.2 Estimation de la ddp de ΔS sur un CBS

Nous avons à présent tous les termes nécessaires pour calculer la densité de probabilité de la variable aléatoire E (4.16). Le terme $\mathbb{P}(E = e \,|\, L(\mathbf{X}) = i)$ se calcule en utilisant (4.17) pour les CLV ainsi que pour les codes QA. Le terme $\mathbb{P}(L(\mathbf{X}) = i)$ se calcule pour les CLV en utilisant la structure du code et les probabilités de la source, et s'estime comme expliqué précédemment pour les codes QA.
Pour estimer la ddp de ΔS sur un CBS, nous allons faire l'hypothèse que le décodeur se resynchronise toujours entre une erreur bit et la suivante. Cette hypothèse est d'autant moins restrictive que l'espérance de E_s du code considéré est faible et que π est faible. Sous cette hypothèse, la variable aléatoire ΔS est indépendemment affectée par les possibles erreurs multiples dans le train binaire. En définissant le polynôme G_e par :

$$G_e(z) = \sum_{i \in \mathbb{Z}} g_{i,e} z^i \triangleq (G(z))^e, \tag{4.22}$$

les coefficients $g_{i,e}$ de G_e vérifient :

$$g_{i,e} = \mathbb{P}(\Delta S = i \,|\, E = e). \tag{4.23}$$

Remarque 4.3 *Le polynôme $G_{e \,|\, e=1}$ correspond aux polynômes de (4.8) et (4.15)*

On définit maintenant le polynôme $\tilde{G}$ par :

$$\tilde{G}(z) \triangleq \sum_{e \in \mathbb{N}} G_e(z)\, \mathbb{P}(E = e) = \sum_{i \in \mathbb{Z}} \tilde{g}_i\, z^i, \tag{4.24}$$

où le terme $\mathbb{P}(E = e)$ est calculé à partir de (4.16), et dépend uniquement de la probabilité π de transition du CBS. D'après cette définition, les coefficients de $\tilde{G}$ vérifient :

$$\begin{aligned} \tilde{g}_i &= \sum_{e \in \mathbb{N}} g_{i,e}\, \mathbb{P}(E = e) && (4.25)\\ &= \sum_{e \in \mathbb{N}} \mathbb{P}(\Delta S = i \,|\, E = e) \mathbb{P}(E = e) && (4.26)\\ &= \mathbb{P}(\Delta S = i). && (4.27) \end{aligned}$$

Une estimation de la ddp de ΔS sur un CBS peut donc être réalisée en utilisant la méthode décrite ci-dessus. Cette méthode a été appliquée aux différents CLV et codes QA présentés dans le chapitre 2.

Exemple 4.6: L'estimation de la d.d.p de ΔS pour le code $\mathcal{C}_5$ et pour un rapport signal à bruit $E_b/N_0 = 5$ dB est la suivante :

$$\begin{cases} \mathbb{P}(\Delta S \leq -4) & = 0.0000002 \\ \mathbb{P}(\Delta S = -3) & = 0.0000235 \\ \mathbb{P}(\Delta S = -2) & = 0.0013201 \\ \mathbb{P}(\Delta S = -1) & = 0.0493389 \\ \mathbb{P}(\Delta S = 0) & = 0.9186664 \\ \mathbb{P}(\Delta S = 1) & = 0.0301524 \\ \mathbb{P}(\Delta S = 2) & = 0.0004930 \\ \mathbb{P}(\Delta S = 3) & = 0.0000053 \\ \mathbb{P}(\Delta S \geq 4) & = 0.0000001. \end{cases} \tag{4.28}$$

Ces estimations ont été obtenues pour $L(\mathbf{S}) = 100$ symboles. Les valeurs obtenues par simulation sont très proches des valeurs estimées :

$$\begin{cases} \mathbb{P}(\Delta S \leq -4) & = 0.0000002 \\ \mathbb{P}(\Delta S = -3) & = 0.0000207 \\ \mathbb{P}(\Delta S = -2) & = 0.0012587 \\ \mathbb{P}(\Delta S = -1) & = 0.0500770 \\ \mathbb{P}(\Delta S = 0) & = 0.9185508 \\ \mathbb{P}(\Delta S = 1) & = 0.0296306 \\ \mathbb{P}(\Delta S = 2) & = 0.0004578 \\ \mathbb{P}(\Delta S = 3) & = 0.0000041 \\ \mathbb{P}(\Delta S \geq 4) & = 0.0000001 \end{cases} \tag{4.29}$$

Ces valeurs ont été obtenues en moyennant 10^7 réalisations conjointes de la source et du canal. Les valeurs estimées et simulées de la ddp de ΔS pour le code $\mathcal{Q}_7$ et un rapport signal à bruit $E_b/N_0 = 4$ dB sont illustrées sur la figure 4.5. L'estimation de la ddp de ΔS est également très proche des valeurs simulées. On peut tout de même remarquer que pour le code $\mathcal{Q}_7$, les probabilités que ΔS soit impaire sont nulles en théorie (une erreur bit ne peut produire qu'un décalage paire de symboles pour ce code). Les valeurs obtenues par simulation ne sont pas exactement nulles. Il peut se produire un décalage impair de symboles lorsqu'une erreur bit arrive vers la fin du train binaire. Dans ce cas là, il se peut que la resynchronisation n'ait pas lieu avant la fin de la transmission, produisant des décalages impairs de symbole.

Nous venons donc de voir comment estimer la ddp de ΔS sur un CBS. Cette estimation est valable pour les CLV (en utilisant l'ESD classique [MR85]) et pour les codes QA (en utilisant l'ESD étendu présenté dans ce chapitre [MJG08b]).
Dans la suite de ce chapitre, nous allons montrer que la ddp de ΔS est un outil qui permet de comparer les performances des CLV dans le cas où un décodage souple avec contrainte de terminaison est utilisé au décodeur. Dans [ZZ02], il est montré que si un

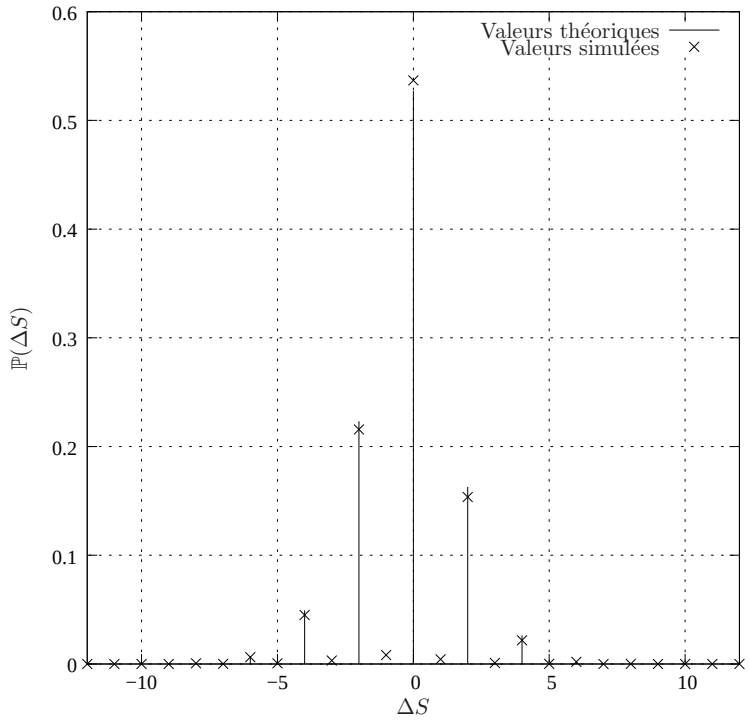

FIG. 4.5 – Valeurs estimées et simulées de la ddp de ΔS : Code $\mathcal{Q}_7$, $E_b/N_0 = 4\,\text{dB}$

décodage dur est utilisé au décodeur, la valeur moyenne de E_s permet de comparer les performances des CLV. Par contre, ces critères ne sont plus adaptés lorsqu'un décodage souple avec contrainte de terminaison est appliqué au décodeur. On montrera également que la ddp de ΔS permet d'analyser les performances des CLV et codes QA sur le modèle d'état agrégé présenté dans le chapitre 3. On verra en particulier qu'il est possible de calculer une borne supérieure sur la valeur optimale du paramètre d'agrégation T, et de prévoir la pertinence du choix de T pour un code donné et un rapport signal à bruit donné.

4.5 Pseudo-degré et critères de sélection des CLV

4.5.1 Pseudo-degré d'un code

Les polynômes issus du calcul de la ddp de ΔS peuvent être de degré infini si l'ESD comporte des boucles modifiant le décalage symbole. Néanmoins, pour de tels polynômes, les coefficients de z^k et de z^{-k} tendent vers 0 lorsque k tend vers l'infini. Nous allons définir le pseudo-degré d'un polynôme représentant la ddp de ΔS de la façon suivante :

Code	d_η	$\mathbb{P}(\Delta S = 0)$	$H(\Delta S)$	MEPL[ZZ02]	VEPL[ZZ02]	DLN	TEB	TESQ
$\mathcal{C}_1$	3	0.9185	0.499	3.89256	34.721	0.00877	0.00193	0.34053
$\mathcal{C}_2$	4	0.9005	0.578	2.02273	2.003	0.00632	0.00191	0.33641
$\mathcal{C}_3$	4	0.8971	0.595	2.06061	2.107	0.00626	0.00192	0.33636
$\mathcal{C}_4$	4	0.8913	0.608	4.07692	27.800	0.00759	0.00177	0.31548
$\mathcal{C}_5$	3	0.9187	0.497	1.71023	1.200	0.00586	0.00194	0.34296
$\mathcal{C}_6$	4	0.8996	0.578	3.54546	18.854	0.00758	0.00182	0.32368
$\mathcal{C}_7$	5	0.7088	1.287	1.55556	0.370	0.00619	0.00154	0.21849
$\mathcal{C}_8$	10	0.7006	1.553	2.34861	2.045	0.00646	0.00134	0.19543
$\mathcal{C}_9$	9	0.6703	1.632	1.95707	1.025	0.00571	0.00123	0.16739
$\mathcal{C}_{10}$	36	0.6401	2.267	6.18182	36.231	0.00483	0.00074	0.10354
$\mathcal{C}_{11}$	8	0.8797	0.655	1.85227	2.233	0.00614	0.00183	0.32219
$\mathcal{C}_{12}$	8	0.8882	0.620	1.71678	1.506	0.00617	0.00187	0.32951
$\mathcal{C}_{13}$	8	0.8860	0.634	1.79798	1.914	0.00615	0.00182	0.32142
$\mathcal{C}_{14}$	8	0.8957	0.599	2.03104	2.952	0.00666	0.00186	0.32698
$\mathcal{C}_{15}$	8	0.8941	0.610	2.20321	4.144	0.00685	0.00189	0.33244
$\mathcal{C}_{16}$	6	0.9044	0.564	1.98086	2.615	0.00672	0.00193	0.33829

TAB. 4.8 – Pseudo-degrés d_η pour $\eta = 10^{-6}$, critères proposés, critères de [ZZ02], et performances de décodage sur le treillis bit-symbole pour $E_b/N_0 = 6\,\text{db}$, et $L(\mathbf{S}) = 100$.

Définition 4.1 *Soit η une probabilité. Le pseudo-degré de précision η du polynôme $\tilde{G}$ de coefficients $\tilde{g}_i$ est l'entier d_η qui vérifie*

$$d_\eta \triangleq \min_{\mathbb{N}^*} d \;\Big|\; \sum_{i \in \mathbb{Z} - \{-d,\ldots,d\}} \tilde{g}_i < \eta, \tag{4.30}$$

où les coefficients $\tilde{g}_i$ sont issus de (4.27).

D'après cette définition, on a :

$$\mathbb{P}(\Delta S \in \{-d_\eta, ..., d_\eta\}) > 1 - \eta \tag{4.31}$$

L'intervalle $\mathcal{D}_\eta = \{-d_\eta, ..., d_\eta\}$ représente l'intervalle typique de désynchronisation d'un code (avec une précision η). Cet intervalle dépend du code (CLV ou QA) et du rapport signal à bruit du canal.

Exemple 4.7: En utilisant la ddp du code $\mathcal{C}_5$ pour $E_b/N_0 = 6$ dB (cf exemple 4.6), le pseudo-degré de précision $\eta = 10^{-6}$ du code $\mathcal{C}_5$ est égal à $d_\eta = 3$ puisque $\mathbb{P}(|\Delta S| \geq 3) > 10^{-6}$ et $\mathbb{P}(|\Delta S| \geq 4) < 10^{-6}$. Les pseudo-degrés des CLV $\mathcal{C}_1$ à $\mathcal{C}_{16}$ pour un rapport signal à bruit de 6 dB sont donnés dans la deuxième colonne du tableau 4.8.

4.5.2 Critères de sélection des CLV en décodage souple

L'espérance et la variance de la longueur moyenne de propagation d'une erreur bit sont de bons critères pour comparer les performances des CLV dans le cas où un décodage dur est appliqué au décodeur [ZZ02]. Un CLV est d'autant plus robuste aux erreurs issues du canal que les deux quantités MEPL et VEPL sont faibles. Nous allons montrer dans cette section que ces deux quantités ne sont plus adaptées pour refléter les performances des CLV lorsqu'un décodage souple avec contrainte de terminaison est appliqué au décodeur. Par contre, nous verrons que la ddp de ΔS permet de fournir deux critères pour comparer les performances des CLV pour ce type de décodage. Nous allons pour cela considérer le schéma de transmission suivant. Des séquences de $L(\mathbf{S})$ symboles issues de la source, sont encodées par un CLV. Le train binaire résultant, de longueur $L(\mathbf{X})$, est modulé par une modulation BPSK. Les bits sont transmis sur canal BABG de rapport signal à bruit E_b/N_0. Le décodeur réalise une estimation du type Maximum a Posteriori sur le treillis bit/symbole présenté dans le chapitre 2. Nous supposons de plus que le décodeur utilise une contrainte de terminaison sur la longueur de la séquence émise. En d'autres termes, $L(\mathbf{S})$ est connu par le décodeur. Cette contrainte de terminaison élimine pendant le décodage toutes les séquences qui ne contiennent pas $L(\mathbf{S})$ symboles, c'est à dire toutes les séquences de $L(\mathbf{X})$ bits qui ne vérifient pas la contrainte $\Delta S = 0$. En adoptant ce schéma de transmission, le décodeur dispose de deux types d'information outre les observations issues du canal : la redondance résiduelle du CLV et l'information apportée par la contrainte de terminaison. La redondance résiduelle du CLV est donnée par la différence entre la longueur de description moyenne du CLV et l'entropie de la source. L'information apportée par la contrainte de terminaison est elle donnée par l'entropie de la ddp de la variable aléatoire ΔS. On va considérer des CLV de même longueur de description moyenne, c'est à dire dont la redondance résiduelle n'impacte pas les différences de performance des codes entre eux. Pour les codes $\mathcal{C}_1$ à $\mathcal{C}_{16}$, la redondance résiduelle est 0.0781 bits d'information.

La ddp de ΔS nous permet de calculer les deux quantités suivantes :
– la probabilité $\mathbb{P}(\Delta S = 0)$ d'avoir une resynchronisation forte
– l'entropie $H(\Delta S)$.

Si la probabilité $\mathbb{P}(\Delta S) = 0$ est faible, le nombre de séquences désynchronisées (au sens du nombre de symboles) qui vont être éliminées par la contrainte de terminaison sera élevé, augmentant ainsi la probabilité de corriger les erreurs issues du canal. L'entropie $H(\Delta S)$ représente la quantité d'information apportée par la contrainte de terminaison sur le treillis bit/symbole. Ces deux quantités, issues du calcul de la ddp de ΔS, représentent donc des critères pour comparer les performances des CLV lorsqu'un décodage souple avec contrainte de terminaison est appliqué au décodeur. Les valeurs de ces deux critères pour les codes $\mathcal{C}_1$ à $\mathcal{C}_{16}$ sont indiquées dans le tableau 4.8,

$Code$	d_η	$\mathbb{P}(\Delta S = 0)$	$H(\Delta S)$	TEB	TESQ
		$L(\mathbf{S}) = 500$			
$\mathcal{C}_5$	5	0.67565	1.39229	0.002210	0.90012
$\mathcal{C}_7$	9	0.30597	2.49437	0.002200	0.84090
$\mathcal{C}_{10}$	40	0.13111	4.38846	0.001687	0.64636
		$L(\mathbf{S}) = 1000$			
$\mathcal{C}_5$	7	0.49590	1.91479	0.002290	0.99062
$\mathcal{C}_7$	14	0.19019	3.00963	0.002275	0.98422
$\mathcal{C}_{10}$	40	0.03215	4.97712	0.001899	0.92838

TAB. 4.9 – Pseudo-degrés d_η pour $\eta = 10^{-6}$, critères proposés, critères de [ZZ02], et performance de décodage pour $E_b/N_0 = 6$ db, et $L(\mathbf{S}) = 500$ et $L(\mathbf{S}) = 1000$.

ainsi que les valeurs de MEPL et VEPL pour ces mêmes codes. Les performances de décodage de ces codes sont également indiquées dans le tableau. Trois différentes mesures de performance ont été considérées : la distance de Levenshtein normalisée (DLN) [Lev66], le TEB au sens de la distance de Hamming et le TESQ. Parmi tous ces codes, le code $\mathcal{C}_{10}$ est celui dont la MEPL et la VEPL sont les plus élevées. On peut donc s'attendre à ce que ce code soit le moins performant. Cependant, cette conclusion ne peut être tirée que lorsqu'un décodage dur est appliqué au décodeur. En effet, les critères proposés ($\mathbb{P}(\Delta S = 0)$ et $H(\Delta S)$) sont ici mieux appropriés pour analyser les performance des codes. Le code $\mathcal{C}_{10}$ est celui dont $\mathbb{P}(\Delta S = 0)$ est la plus faible et dont l'entropie $H(\Delta S)$ est la plus forte. C'est aussi le code qui présente les meilleurs performances pour les trois mesures considérées. De façon similaire, le moins bon code en terme de performance est le code $\mathcal{C}_5$. Ce code a la probabilité $\mathbb{P}(\Delta S = 0)$ la plus élevée et l'entropie $H(\Delta S)$ la plus faible. Les mêmes observations peuvent être faites pour des séquences plus longues, comme indiqué dans le tableau 4.5.2.

La même analyse a été réalisée pour une source avec un alphabet plus grand. Nous avons utilisé l'alphabet anglais, ainsi que trois codes qui lui sont associé dans [MR85] et [SD95]. La source et les codes considérés sont représentés dans le tableau 4.5.2. Ces codes ont une longueur de description moyenne de 4.1557 bits par symboles source. Dans le tableau 4.5.2, sont indiquées les valeurs de $\mathbb{P}(\Delta S = 0)$ et de $H(\Delta S)$ pour ces trois codes, ainsi que les valeurs de MEPL et de VEPL et les performances de décodage de ces codes. Le code $\mathcal{C}_{17}$ est le moins bon code selon les critères MEPL et VEPL, mais le meilleur selon les critères $\mathbb{P}(\Delta S = 0)$ et $H(\Delta S)$ ainsi qu'en terme de performance de décodage.

Code ASCII	Probabilité	$\mathcal{C}_{17}$[MR85]	$\mathcal{C}_{18}$[MR85]	$\mathcal{C}_{19}$[SD95]
A	0.08833733	0000	0100	0100
B	0.01267680	011111	111101	000101
C	0.02081665	11111	11100	01100
D	0.04376834	00010	10110	01101
E	0.14878569	001	000	100
F	0.02455297	11100	11010	00011
G	0.01521216	011101	111011	001100
H	0.05831331	1000	1000	1100
I	0.05644515	1001	1001	1111
J	0.00080064	111010101	111111110	001110100
K	0.00867360	1110100	1111110	0011100
L	0.04123298	00011	10111	00100
M	0.02361889	11110	11011	01110
N	0.06498532	0110	0111	0101
O	0.07245796	0100	0101	1101
P	0.02575393	10111	11001	01111
Q	0.00080064	1110101000	1111111110	0011101010
R	0.06872164	0101	0110	0000
S	0.05537763	1010	1010	1110
T	0.09354149	110	001	101
U	0.02762209	10110	11000	00101
V	0.01160928	111011	111110	001111
W	0.01868161	011100	111010	001101
X	0.00146784	11101011	11111110	00111011
Y	0.01521216	011110	111100	000100
Z	0.00053376	1110101001	1111111111	0011101011

TAB. 4.10 – Source et codes pour l'alphabet anglais

Code	$\mathbb{P}(\Delta S = 0)$	$H(\Delta S)$	MEPL[ZZ02]	VEPL[ZZ02]	TEB	TESQ
$\mathcal{C}_{17}$	0.7312	1.376	5.456	5.868	0.002082	0.53768
$\mathcal{C}_{18}$	0.8338	0.861	3.863	3.906	0.002094	0.56607
$\mathcal{C}_{19}$	0.8433	0.844	1.915	1.192	0.002105	0.56900

TAB. 4.11 – Pseudo-degrés d_η pour $\eta = 10^{-6}$, critères proposés, critères de [ZZ02], et performance de décodage pour $E_b/N_0 = 6\,\text{db}$, et $L(\mathbf{S}) = 100$ pour les codes de l'alphabet anglais

4.6 Analyse de performances du modèle agrégé

Nous allons maintenant utiliser les propriétés de resynchronisation des CLV et QA pour analyser leurs performances sur le modèle agrégé, et les comparer avec leurs performances sur le modèle optimal. Les propriétés de resynchronisation des codes vont notamment nous permettre d'estimer la valeur minimale du paramètre T d'agrégation pour laquelle les performances sur le modèle agrégé tendent vers les performances optimales. Nous montrerons également comment il est possible de déterminer les valeurs de T les plus pertinentes pour un code et un rapport signal à bruit donné.
D'après la définition (4.30) du pseudo-degré d_η d'un code, la probabilité que ΔS appartienne à l'intervalle $\mathcal{D}_\eta = \{-d_\eta, ..., d_\eta\}$ est supérieure ou égale à $1-\eta$. On en déduit la propriété suivante :

Propriété 4.1 *Soit le paramètre d'agrégation T tel que $T \geq d_\eta$. L'algorithme de Viterbi appliqué sur le treillis de paramètre T sélectionne, avec une probabilité supérieure ou égale à $1-\eta$, une séquence comportant le nombre correct de symboles.*

Cependant, cette propriété ne signifie pas que les performances en terme de taux d'erreur sur le treillis agrégé de paramètre T seront les mêmes que sur le treillis optimal. Afin d'analyser les performances respectives des codes sur les deux modèles, il nous faut quantifier l'information apportée par la contrainte de terminaison sur chacun de ces modèles. Cette quantité d'information est égale à l'entropie de la variable ΔS sur le modèle optimal et à l'entropie de la variable $(\Delta S \bmod T)$ sur le modèle agrégé. Ces deux quantités dépendent de la longueur $L(\mathbf{S})$ de la séquence et du rapport signal à bruit E_b/N_0. Nous supposerons ici qu'ils sont fixés. Nous allons montrer qu'en choisissant $T = 2d_\eta + 1$, la quantité d'information sur le modèle agrégé tend vers la quantité d'information disponible sur le modèle optimal, et un encadrement de $H(\Delta S \bmod T)$ sera dérivé.

Rappelons d'abord que l'on peut représenter la ddp de la variable aléatoire ΔS pour un code et un rapport signal à bruit donné par un polynôme

$$\tilde{G}(z) = \sum_{i \in \mathbb{Z}} \tilde{g}_i \, z^i, \tag{4.32}$$

avec $\forall i \in \mathbb{Z}$, $\tilde{g}_i = \mathbb{P}(\Delta S = i)$. Il a été montré dans la section 4.4.2 comment calculer les coefficients $\tilde{g}_i$.
On définit maintenant, pour un entier T, la quantité $\tilde{g}_i^T$ par

$$\tilde{g}_i^T \triangleq \mathbb{P}(\Delta S \bmod T = i). \tag{4.33}$$

Cette quantité peut être calculée à partir des coefficients $\tilde{g}_i$:

$$\tilde{g}_i^T = \sum_{j \in \mathbb{Z}} \tilde{g}_{jT+i}. \tag{4.34}$$

Ainsi, la quantité d'information apportée par la contrainte de terminaison sur un treillis agrégé de paramètre T est donnée par

$$H(\Delta S \bmod T) = -\sum_{i\in\{0,\dots,T-1\}} \tilde{g}_i^T \log_2 \tilde{g}_i^T \tag{4.35}$$

$$\geq H(\Delta S) + \sum_{i\notin\{0,\dots,T-1\}} \tilde{g}_i \log_2 \tilde{g}_i. \tag{4.36}$$

En prenant $T = 2d_\eta + 1$, (4.36) s'écrit aussi

$$H(\,\Delta S \bmod (2d_\eta+1)\,) \geq H(\Delta S) + \sum_{i\notin\mathcal{D}_\eta} \tilde{g}_i \log_2 \tilde{g}_i. \tag{4.37}$$

On va maintenant supposer que $\eta < \frac{1}{e}$. Sous cette hypothèse, la fonction $x \mapsto x \log_2(x)$ est décroissante sur l'intervalle $[0, \eta]$. D'après la définition (4.30) du pseudo-degré, on a $\forall i \notin \mathcal{D}_\eta,\ \tilde{g}_i \leq \eta$. Ainsi,

$$\sum_{i\notin\mathcal{D}_\eta} \tilde{g}_i \log_2 \tilde{g}_i \geq |\{i \notin \mathcal{D}_\eta\,,\ \tilde{g}_i > 0\}|\,\eta \log_2 \eta, \tag{4.38}$$

où le cardinal $|\{i\,,\ \tilde{g}_i > 0\}|$ de l'ensemble des valeurs non nulles de $\tilde{g}_i$ est majoré par la longueur en bit $L(\mathbf{X})$ du message. On peut donc écrire

$$|\{i \notin \mathcal{D}_\eta, \tilde{g}_i > 0\}| \leq L(\mathbf{X}) - 2\,d_\eta - 1. \tag{4.39}$$

Ainsi, pour un η donné, on a l'encadrement suivant de $H(\,\Delta S \bmod (2d_\eta+1)\,)$:

$$H(\Delta S) + (L(\mathbf{X}) - 2\,d_\eta - 1)\,\eta \log_2 \eta \leq H(\Delta S \bmod 2\,d_\eta + 1) \leq H(\Delta S). \tag{4.40}$$

Cet encadrement signifie également que pour η suffisamment petit (de façon analogue pour $2d_\eta+1$ suffisamment grand), la quantité d'information apportée par la contrainte de terminaison sur le treillis agrégé tend vers celle du modèle optimal. La vitesse de convergence de $H(\Delta S \bmod T)$ vers $H(\Delta S)$ est la même que la vitesse de convergence des performances d'un code sur le modèle agrégé vers les performances sur le modèle optimal. Le comportement de $H(\Delta S \bmod T)$ est représenté pour quelques CLV sur la figure 4.6. L'entropie $H(\Delta S)$ qui correspond à l'information disponible sur le modèle optimal est également représentée pour chacun des codes. Les valeurs de la figure 4.6 ont été obtenues pour $L(\mathbf{S}) = 100$ et $E_b/N_0 = 7$ dB. On peut remarquer que pour le code $\mathcal{C}_{10}$, $H(\Delta S \bmod T)$ n'a pas encore convergé pour $T = 10$, contrairement aux autres codes de cette figure. Les TESQ des codes de la figure 4.6 dans les mêmes conditions de simulations sont représentés sur la figure 4.7. On voit ici clairement apparaître le lien entre le comportement de $H(\Delta S \bmod T)$ et les performances des codes sur le modèle agrégé. Comme il a été expliqué dans la section 4.5, les codes les plus performants sont ceux dont l'entropie $H(\Delta S)$ est la plus élevée. Par contre, la

valeur optimale de T pour ces codes est plus grande, puisque leur pseudo-degré est plus élevé. On peut également remarquer que pour le code $\mathcal{C}_{13}$, on a $H(\Delta S \bmod 1) = H(\Delta S \bmod 2) = 0$. Cette particularité de ce code explique pourquoi ses performances ne sont pas améliorées en passant de $T = 1$ à $T = 2$. Pour ce même code, on a également $H(\Delta S \bmod 4) < H(\Delta S \bmod 3)$ et $H(\Delta S \bmod 6) < H(\Delta S \bmod 5)$. Parallèlement, les FER pour $T = 4$ et $T = 6$ sont respectivement supérieurs aux FER pour $T = 3$ et $T = 5$. Les performances des codes ont été obtenues en appliquant l'algorithme de Viterbi sur le modèle agrégé. Cet algorithme minimise le taux d'erreur séquence. Le parallèle entre la convergence de $H(\Delta S \bmod T)$ et les performances sur le modèle agrégé est aussi observé pour les codes QA, comme indiqué sur les figures 4.8 et 4.9.

4.7 Conclusion

Dans ce chapitre, nous avons d'abord présenté les méthodes de calcul des propriétés de resynchronisation des CLV. Ces méthodes ont été introduites par Maxted *et al.* dans [MR85] et Swaszek *et al.* dans [SD95]. Elles permettent notamment de calculer la valeur moyenne du nombre de bits sur lesquels une erreur se propage, ainsi que la loi de probabilité de la différence entre le nombre de symboles encodés et décodés (ΔS). Le calcul de ces deux quantités est valable pour les CLV et lorsqu'une seule erreur bit se produit dans le message original. Il a été montré dans [ZZ02] que la première de ces deux quantités permettait de comparer les performances des CLV entre eux lorsque l'on applique un décodage dur.

Afin de réaliser la même analyse lorsque l'on se place sur un canal BABG et lorsque l'on réalise un décodage souple sur le modèle agrégé, nous avons proposé une méthode permettant d'estimer la loi de probabilité de ΔS lorsque le message binaire est transmis sur un CBS. En effet, cette loi de probabilité permet de calculer la quantité d'information apportée par la contrainte de terminaison sur le modèle agrégé, et donc d'analyser les performances des CLV sur ce même modèle (en fonction du paramètre d'agrégation) et et de les comparer aux performances sur le modèle optimal.

Enfin, nous avons également adapté les méthodes de [MR85] et [SD95] afin d'étendre ces résultats aux codes QA.

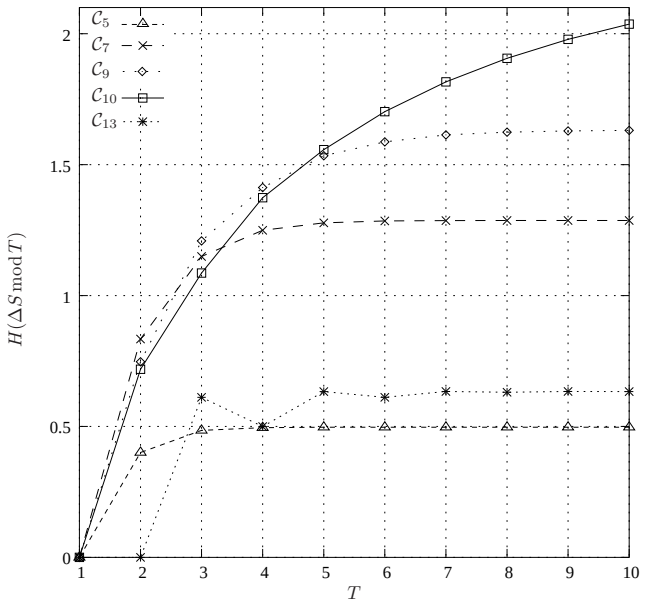

FIG. 4.6 – Entropie de $\Delta S \bmod T$ en fonction de T pour les codes $\mathcal{C}_5, \mathcal{C}_7, \mathcal{C}_9, \mathcal{C}_{10}$ et $\mathcal{C}_{13}$.

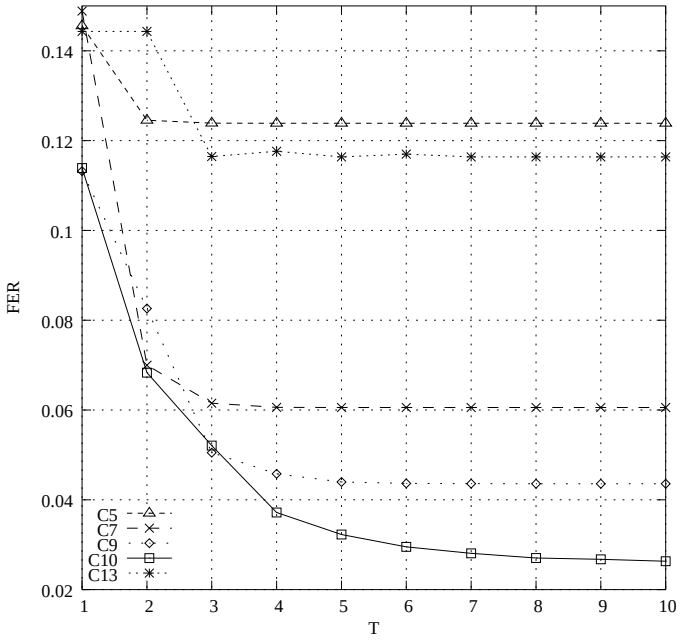

FIG. 4.7 – Taux d'erreur séquence en fonction de T pour les codes $\mathcal{C}_5, \mathcal{C}_7, \mathcal{C}_9, \mathcal{C}_{10}$ et $\mathcal{C}_{13}$.

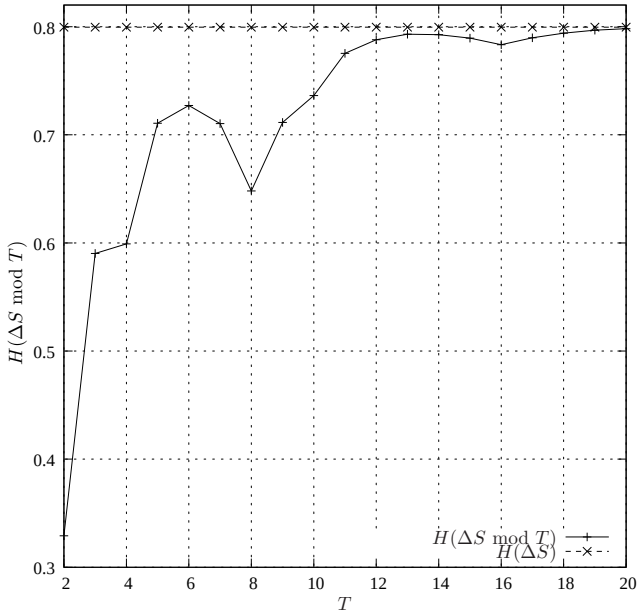

FIG. 4.8 – Entropie de $\Delta S \bmod T$ en fonction de T pour le code $\mathcal{Q}_9$ et $E_b/N_0 = 6$ dB.

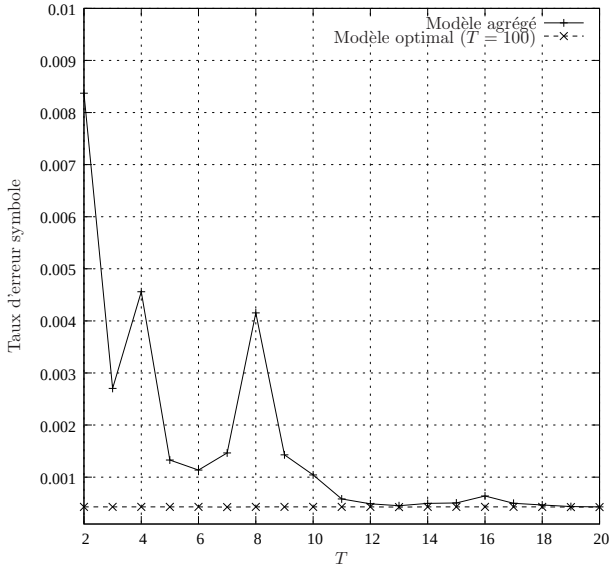

FIG. 4.9 – Taux d'erreur symbole en fonction de T pour le code $\mathcal{Q}_9$ et $E_b/N_0 = 6$ dB.

Chapitre 5

Codage de source distribué à l'aide de codes quasi-arithmétique

---

"Le pire n'est jamais décevant"

Claude Lelouch

Dans ce chapitre, nous nous intéresserons au codage de source distribué. Le codage de source distribué considère une situation où deux (ou plusieurs) sources de données statistiquement corrélées sont encodées séparément par deux encodeurs qui ne communiquent pas entre eux. Quelles performances en terme de compression peut-on obtenir dans ce cas ? Slepian et Wolf se sont penchés sur ce problème en 1973 et ont montré que, de façon assez surprenante, les deux sources peuvent être encodées au même taux minimal que lorsque les deux encodeurs communiquent entre eux.

Nous présenterons d'abord le cadre théorique basée sur les travaux de Slepian et Wolf. Puis, nous détaillerons quelques solutions pratiques visant à atteindre les limites théoriques. Nous verrons que ces solutions sont très souvent basées sur des codes de canal, qui sont performants dans le cas où les sources en entrée sont uniformes et lorsqu'ils agissent sur des séquences de très grande taille.

L'utilisation des codes de source pour le problème du codage de source distribué est motivé par le fait que ces codes présentent de très bonnes performances en terme de compression, et sont très aptes à exploiter les statistiques de la source.

Nous présenterons dans ce chapitre deux solutions au problème du codage de source distribué utilisant des codes QA. Nous considérerons premièrement des codes QA dont la sortie est poinçonnée, afin d'atteindre des taux de compression inférieurs à

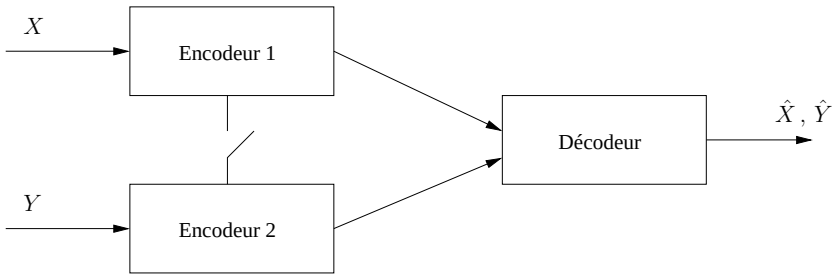

FIG. 5.1 – Codage de source distribué de deux sources corrélées

l'entropie de la source. Puis, nous présenterons les codes QA avec recouvrement d'intervalle (QARI), qui permettent d'atteindre sans poinçonnage des taux de compression très faibles. Ce type de codes a été mis en oeuvre en collaboration avec Xavi Artigas, doctorant à l'Université Polytechnique de Catalogne à Barcelone. Ces codes ne sont évidemment pas uniquement décodables. Cependant, dans le cadre du problème du codage de Slepian-Wolf asymétrique, nous verrons que l'utilisation de l'information adjacente permet d'atteindre des résultats intéressants comparativement aux solutions basées sur des codes de canal. Le schéma de codage de source distribué proposé sera décrit dans ce chapitre pour les deux types de code ci-dessus. Ces schémas seront appliqués à des sources théoriques avec et sans mémoires.

5.1 Cadre théorique

Dans [Sha48], Shannon a montré que l'on pouvait compresser sans perte deux sources corrélées et discrètes X et Y à un taux minimal donné par l'entropie conjointe $H(X,Y)$ de ces deux sources. Pour cela, les deux sources doivent théoriquement être encodées et décodées conjointement. Le théorème de Slepian-Wolf [SW73] énonce que ce taux minimal $H(X,Y)$ peut également être atteint théoriquement si X et Y sont encodées *séparément* et décodées conjointement.
Ce résultat est illustré sur la figure 5.1. Même si la communication entre l'encodeur 1 et l'encodeur 2 est coupée, un taux $H(X,Y)$ est suffisant théoriquement pour compresser sans perte X et Y, dès lors que ces deux sources sont décodées conjointement.
Pour reconstruire X et Y avec un taux d'erreur arbitrairement proche de zéro, le théorème de Slepian-Wolf donne la région admissible des taux R_X et R_Y de compression de X et Y. Cette région est délimitée par les relations suivantes :

$$\begin{aligned} R_X &\leq H(X|Y) \\ R_Y &\leq H(Y|X) \\ R_X + R_Y &\leq H(X,Y) \end{aligned} \tag{5.1}$$

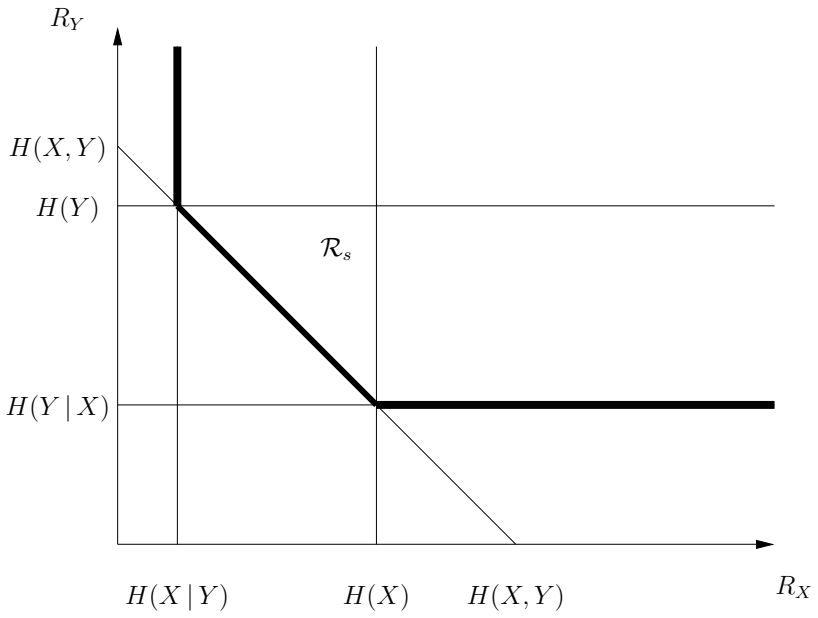

FIG. 5.2 – Région des taux admissibles dans le cas symétrique du théorème de Slepian-Wolf

Cette région des taux admissibles $\mathcal{R}_s$ est représentée sur la figure 5.2. Les traits en gras représentent les bornes inférieures données dans l'équation 5.1.

Le cas représenté sur la figure 5.1 est appelé communément problème de Slepian-Wolf symétrique. Dans ce chapitre, nous nous intéresserons plus particulièrement au problème de Slepian-Wolf asymétrique. Dans le cas asymétrique, la source Y (information adjacente) est simplement encodée à son entropie $H(Y)$. Elle est disponible au décodeur de X. Dans ce cas, l'équation 5.1 devient :

$$\begin{array}{rcl} R_X & \leq & H(X|Y) \\ R_Y & \leq & H(Y) \\ R_X + R_Y & \leq & H(X,Y) \end{array} \tag{5.2}$$

La région des taux admissibles $\mathcal{R}_a$ dans le cas asymétrique est représentée sur la figure 5.3. Les limites de cette région (en gras) correspondent aux bornes inférieures de l'équation 5.2.
Le théorème de Slepian-Wolf dans le cas asymétrique a été étendu au cas des sources à valeurs continues Gaussiennes par Wyner et Ziv dans [WZ76]. Dans cet article, les auteurs ont dérivé le débit minimal nécessaire pour coder la source X sous contrainte que la distorsion moyenne entre X et son estimée soit égale à une certaine valeur.
Les résultats proposés par Slepian et Wolf (et par Winer et Ziv) sont des résultats théoriques. De nombreuses propositions de mise en oeuvre concrètes afin d'approcher

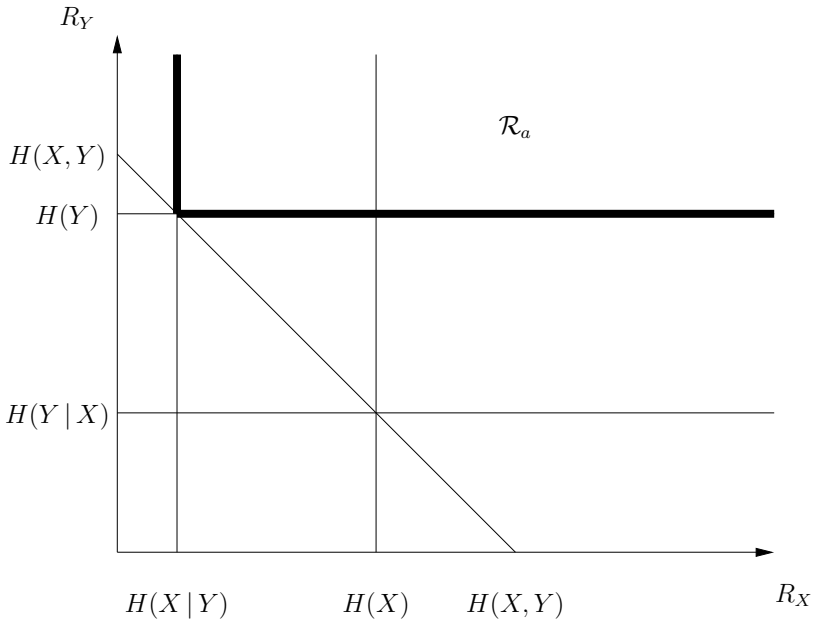

FIG. 5.3 – Région des taux admissibles dans le cas asymétrique du théorème de Slepian-Wolf

ces bornes théoriques ont été développées ces dernières années. Nous allons, dans la section suivante, présenter les différentes mises en oeuvre pratiques existantes à ce jour.

5.2 État de l'art en codage de Slepian-Wolf

Les solutions pratiques en codage de source distribué sont basées soit sur des codes de source [JAI97][ZE01], soit sur des codes de canal comme les codes en bloc [PR99, PR00], les turbo-codes [GFZ01b, GFZ01a, AG02, BM01] ou les codes LDPC [SRP04].

5.2.1 Techniques basées sur des CLV

Dans [JAI97], les auteurs ont utilisés des codes de Huffman pour réaliser la compression des sources $\mathbf{X}$ et $\mathbf{Y}$. Ils se sont placés dans le cadre du problème de Slepian-Wolf asymétrique. La source $\mathbf{Y}$ (information adjacente) définie sur un alphabet $\mathcal{A}_Y$ est encodée par un code de Huffman classique à un taux très proche de son entropie. Puis, $\mathbf{X}$ est encodée à un taux très proche de l'entropie conditionnelle $H(X \mid Y)$ en utilisant un code de Huffman sur un alphabet modifié. En effet, le cardinal de l'alphabet $\mathcal{A}_X$ de $\mathbf{X}$ est réduit afin d'approcher l'entropie conditionnelle. La méthode de construction de l'alphabet modifié de $\mathbf{X}$ est détaillée dans [JAI97]. Nous allons ici l'expliquer

succinctement. L'idée principale est de créer virtuellement une nouvelle source $\mathbf{X}'$ définie sur un alphabet $\mathcal{A}_{X'}$ de cardinal inférieur à celui de $\mathbf{X}$. Pour cela, les symboles x'_i de $\mathcal{A}_{X'}$ représentent en fait des regroupements de symboles x_i de $\mathcal{A}_X$. Ces regroupements sont réalisés en tenant compte de la corrélation entre $\mathbf{X}$ et $\mathbf{Y}$, représentée par la loi de probabilité conjointe $\mathbb{P}(x, y)$. Les symboles x'_i peuvent être représentés par des ensembles :

$$\forall\, 1 \leq i \leq \text{card}(\mathcal{A}_{X'}), x'_i = \{x_{k_1}, \ldots, x_{k_m} \,/\, k_1, \ldots, k_m \in [1, \text{card}(\mathcal{A}_X)]\}, \tag{5.3}$$

et tels que pour tout élément y_j de $\mathcal{A}_Y$, la probabilité $\mathbb{P}(Y = y_j \,|X' = x'_i)$ ne soit strictement positive que pour un seul élément de x'_i. Sous cette condition, en utilisant l'information adjacente Y, l'ambiguïté induite par le regroupement de symboles de $\mathbf{X}$ peut être levée.
Cette technique de regroupements de mots de code ne peut être mise en place que si la matrice représentant la loi de probabilité $\mathbb{P}(x, y)$ comporte des zéros.
Les auteurs de [YOB00] ont montré que la technique de Al Jabri et Al-Issa présentée ci-dessus était sous-optimale. Les auteurs de [ZE01] ont proposé une technique alternative menant à la construction de codes optimaux. La méthode de partitionnement de [ZE01] est plus compliquée que celle de [ZE01] et est basée sur un arbre de partitionnement. L'idée de base est cependant la même que pour [ZE01] : si la densité de probabilité $\mathbb{P}(x, y)$ de X et de l'information adjacente vérifie certaines conditions, des symboles de l'alphabet de X peuvent être regroupés formant ainsi une nouvelle source virtuelle X' dont le cardinal est plus petit et l'entropie plus faible. L'information adjacente est également utilisée pour lever l'ambiguïté issue du regroupement de symboles de $\mathbf{X}$.

5.2.2 Techniques basées sur des codes de canal

Les premières mises en oeuvre du codage de source distribué ont été le plus souvent réalisées à l'aide de codes de sources, comme on vient de le voir par exemple avec [JAI97]. Puis, les codes de canal ont été utilisés car les chercheurs ont établi un lien entre la corrélation entre X et Y et les observations issues d'un canal de transmission classique. Y est alors vue comme une version bruitée de X, et les codes de canal sont très adaptés à ce genre de situation.

5.2.2.1 Utilisation de codes en bloc

La première solution pratique au problème de Slepian-Wolf utilisant des codes de canal a été proposée dans [PR99], où les auteurs utilisent des codes en bloc. Cette technique est nommée DISCUS (en anglais "DIstributed Source Coding Using Syndromes"). Considérons que les sources X et Y sont telles que la distance de Hamming entre X et l'information adjacente Y est toujours inférieure ou égale à un entier t. Dans ce cas, un code en bloc de type $(n, k, 2t + 1)$ est utilisé. Les mots de code de X, de taille

n, vont être codés par leur syndrome. Ce syndrome est calculé à partir de la matrice de parité H du code en bloc, en multipliant le mot de code par cette matrice. La taille de ce syndrome est égale $n-k$. Ainsi, chaque mot de code de X peut être codé avec $n-k$ bits. La connaissance de l'information adjacente permet de retrouver X à partir de son syndrome.

Exemple 5.1: Nous allons d'abord énoncer l'exemple qui a servi de support aux travaux de Pradhan et Ramchandran, puis l'illustrerons avec un code en bloc. Considérons que la distance de Hamming entre X et Y est d'au plus 1, et que les mots de code de X et Y sont composés de 3 bits. La connaissance de Y permet de ne coder X avec uniquement deux bits, puisqu'il n'y a ambiguïté que sur deux bits entre X et Y. Les 8 mots de code de X sont répartis en 4 ensembles de deux mots de code, tels que la distance entre les deux mots de code de chaque ensemble soit maximale. On obtient donc les quatre ensembles

$$\delta_0 = \{000, 111\},\ \delta_1 = \{001, 110\},\ \delta_2 = \{010, 101\},\ \delta_3 = \{011, 100\}. \tag{5.4}$$

Chacun de ces ensembles peut être codé sur deux bits. Le décodeur reçoit ces deux bits, et a donc un choix à faire entre les deux mots de code possibles de l'ensemble. Il va choisir le mot de code dont la distance de Hamming à Y est la plus faible. La distance de Hamming de 1 entre X et Y assure un décodage sans erreur, à condition que Y soit connu par le décodeur. Supposons par exemple, que $X = 100$ et que $Y = 110$. Le décodeur reçoit les bits 11 correspondant à δ_3. La connaissance de Y lui permet de choisir le mot de code de δ_3 le plus proche de Y : 100. Cet exemple correspond au code en bloc $(3, 1, 3)$ de matrice de parité

$$H = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} \tag{5.5}$$

Les syndromes des mots de code d'un ensemble δ_i sont bien égaux à la représentation binaire de i.

5.2.2.2 Utilisation de turbo-codes

Les turbo-codes [BGP93, BG96] ont été pour la première fois utilisés dans le cadre du problème de Slepian-Wolf symétrique par Garcia-Frias et Zhao dans [GFZ01b, GFZ01a]. Les sources X et Y sont encodées par un turbo-code. Seuls la moitié des bits systématiques issus de X et de Y sont transmis. Des bits de parité de chacun des turbo-codes sont poinçonnés pour atteindre le taux de compression désiré. Chacun des décodeurs turbo utilise la corrélation entre X et Y pour faire leur estimation.
Bajcsy et Mitran ont considéré dans [BM01] un turbo-code constitué de deux codes convolutifs (CC) réalisant directement la compression. Ces codes convolutifs prennent plus de bits en entrée qu'en sortie. Ainsi, les trains binaires issus de ces CC sont déjà

compressés. Le décodage se fait de façon itérative en utilisant la structure en treillis des codes convolutifs. Un algorithme BCJR est utilisé pour chacun des décodeur convolutifs, prenant en compte le train binaire issu de l'encodage, l'information adjacente ainsi qu'une information extrinsèque venant de l'autre décodeur.
Aaron et Girod ont eux proposé dans [AG02] d'utiliser un turbo-code poinçonné pour le problème de Slepian-Wolf asymétrique. La corrélation entre X et Y est modélisée par un canal binaire symétrique (CBS) de probabilité de corrélation π. Y est encodé à un taux très proche de son entropie par un code source classique alors que X est encodé par un turbo-code. Les bits systématiques de ce turbo-code sont poinçonnés puisqu'une version bruitée de ces bits est disponible à travers l'information adjacente Y. Seuls des bits systématiques sont ainsi transmis, après en avoir poinçonné un nombre suffisant pour atteindre le taux de compression désiré. Un décodeur de type turbo est utilisé pour estimer X à l'aide de Y et des bits de parité reçus.

5.2.2.3 Utilisation de codes LDPC

Plus récemment, les codes LDPC [Gal62, Gal63] ont été utilisés pour le codage de source distribué, notamment dans [LXG02, TGFZ03, VAG05]. Ces codes permettent d'obtenir de meilleurs résultats que les turbo-codes pour le codage de source distribué. De façon similaire à la méthode de [JAI97] pour des codes en bloc, X est encodé par son syndrome calculé à l'aide de la matrice de parité du code LDPC. Le décodage est un décodage itératif basé sur un algorithme de propagation de confiance, très souvent utilisé pour les codes LDPC dans le cadre de transmissions classiques. Il s'agit d'un décodage itératif. Les croyances sur les bits d'information et les bits de parité sont échangées au niveau du décodeur. Toutefois, pour le problème de Slepian-Wolf asymétrique, l'information adjacente Y est utilisée par le décodeur pour réaliser l'estimation de X.

Dans la suite de ce chapitre, nous allons décrire une solution au problème de Slepian-Wolf asymétrique utilisant des codes QA. La plupart des solutions que nous venons de présenter sont basées sur des codes de canal, qui sont optimaux lorsque la source en entrée est uniforme, et moins performant lorsque la source X présente une forte asymétrie. Les codes QA sont eux à même d'exploiter une asymétrie de la source. Nous proposerons deux solutions utilisant des codes QA : l'une se basant sur des codes QA poinçonnés et l'autre en utilisant un nouveau type de code, les codes QA avec recouvrement d'intervalle.

5.3 Schéma de codage distribué à l'aide de codes QA

Le schéma de codage de source distribué que nous proposons est illustré sur la figure 5.4. Nous considérerons que le message $\mathbf{S} = S_1, \ldots, S_{L(\mathbf{S})}$ issu de la source est binaire. Les symboles possibles seront notés a et b, pour éviter la confusion avec les

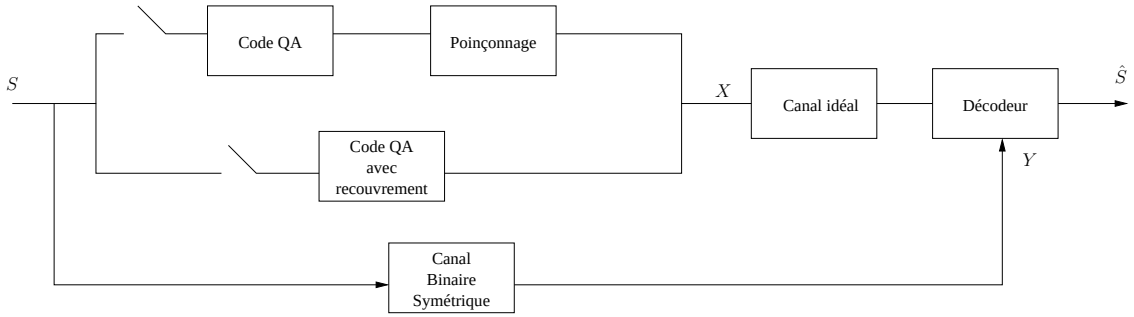

FIG. 5.4 – Schéma de codage de source distribué à l'aide de codes QA

trains binaires issus de l'encodage. La probabilité de a, $\mathbb{P}(a)$ est notée p. Pour des raisons de simplicité, nous ne considérerons que des sources binaires, mais le schéma de codage de source distribué proposé peut s'appliquer à des sources avec un alphabet plus grand. Nous proposons deux méthodes différentes pour encoder la source à un taux inférieur à son entropie. La première méthode utilise des codes QA poinçonnés et la deuxième utilise un nouveau type de code : les codes QA avec recouvrement d'intervalle, présentes dans [AMGT07].
L'encodage de $\mathbf{S}$ produit un message binaire $\mathbf{X} = X_1, \ldots, X_{L(\mathbf{X})}$ qui est transmis sur un canal idéal. Le décodeur applique un algorithme de type BCJR afin de minimiser le TES en sortie. Il utilise le message binaire reçu $\mathbf{X}$, ainsi qu'une information adjacente $\mathbf{Y}$. Cette information adjacente est corrélée à $\mathbf{S}$. Pour l'obtenir, $\mathbf{S}$ est transmis sur un CBS de probabilité de transition π. En d'autres termes, la corrélation entre $\mathbf{S}$ et $\mathbf{Y}$ s'exprime par la relation suivante

$$\forall i \in \{1, \ldots, L(\mathbf{S})\}, \mathbb{P}(S_i = Y_i) = 1 - \pi. \tag{5.6}$$

5.3.1 Codes quasi-arithmétique poinçonnés

Nous allons utiliser la représentation des codes QA en machines à états finis, comme expliqué dans le chapitre 2. Cependant, nous allons considérer, en plus des sources sans mémoire, des sources avec une mémoire d'ordre 1. Une source binaire avec une mémoire d'ordre 1 peut être définie de manière unique par la probabilité $\mathbb{P}(a) = p$ ainsi qu'un coefficient de corrélation ρ_m ($\rho_m \in [-1, 1]$). Les probabilités contextuelles se déduisent à partir de ces deux paramètres par les relations suivantes :

$$\begin{aligned} \mathbb{P}(S_k = a \,|\, S_{k-1} = a) &= 1 + (\rho_m - 1)(1 - p) \\ \mathbb{P}(S_k = b \,|\, S_{k-1} = b) &= 1 + (\rho_m - 1)p. \end{aligned} \tag{5.7}$$

La construction de l'automate d'encodage d'un code QA, telle que cela a été décrit dans la section 2.3.2, n'est valable que pour les sources sans mémoire. Pour tenir

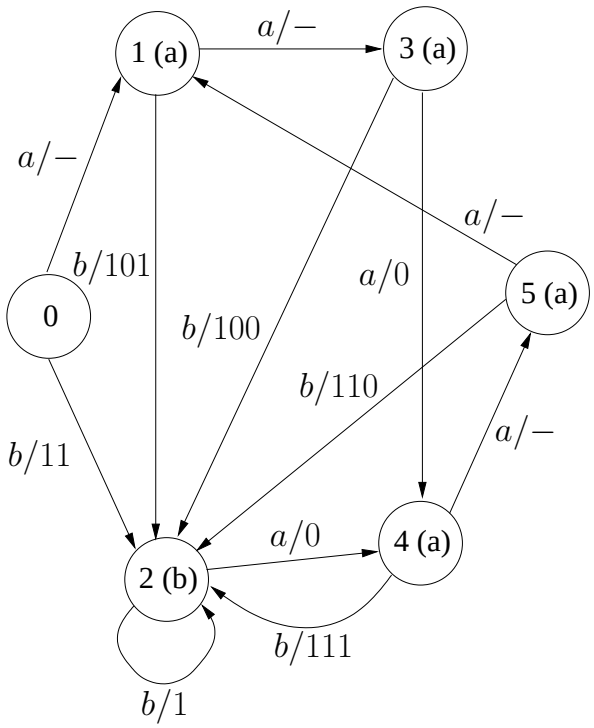

FIG. 5.5 – Automate d'encodage d'un code QA adapté à une source avec une mémoire d'ordre 1 ($N = 8, p = 0.8$, et $\rho_m = 0.3$).

compte des statistiques d'ordre 1 de la source, la procédure doit être modifiée. L'état initial est, comme pour les codes sans mémoire, composé de l'intervalle initial $[0, N[$ et de l'entier $follow = 0$. Cet état initial est sans contexte, il ne contient pas la mémoire du symbole précédent. Deux nouveaux états sont créés en subdivisant l'intervalle initial suivant p et $1 - p$. Ces nouveaux états sont atteints respectivement si un a ou un b est émis. Le symbole précédent est ainsi gardé en mémoire dans chaque nouvel état créé. Puis, pour chaque état non traité, deux nouvelles transitions sont calculées à partir des probabilités $\mathbb{P}(a|a)$ et $\mathbb{P}(b|a)$ si la mémoire de l'état courant contient un a, ou a partir des probabilités $\mathbb{P}(a|b)$ et $\mathbb{P}(b|b)$ si la mémoire de l'état courant contient un b. L'automate d'encodage est ainsi créé lorsque tous les états ont été visités.

Exemple 5.2: L'automate d'encodage du code QA associé à une source avec mémoire de paramètres $N = 8, p = 0.8$ et $\rho_m = 0.3$ est illustré sur la figure 5.5. Cet automate a été construit selon la méthode décrite ci-dessus. L'état initial de cet automate est l'état 0, correspondant à l'intervalle initial $[0, 8[$ (cet état ne contient pas la mémoire du symbole précédent). Les états $1\,(a)$ et $2\,(b)$ ont été obtenus en partitionnant l'intervalle initial selon les probabilités p et $1 - p$ respectivement. Le symbole entre parenthèse dans l'étiquetage d'un état correspond au symbole précédent. Ainsi, un état étiqueté

avec un (a) ne peut être atteint que si la réalisation suivante de la source est un a. Ensuite, les probabilités $\mathbb{P}(a|a)$ et $\mathbb{P}(b|a)$ ont été utilisées pour partitionner les intervalles des états (a) et $\mathbb{P}(a|b)$ et $\mathbb{P}(b|b)$ pour partitionner ceux des états (b).

L'encodage de $\mathbf{S}$ par un code QA classique produit un message binaire $\mathbf{X}$ de longueur $L(\mathbf{X})$ très proche de l'entropie de la source. Le taux de compression par symbole de source d'un code QA est noté dans la suite R_q. Ainsi, la longueur moyenne de $\mathbf{X}$ est égale à $R_q \times L(\mathbf{S})$. Afin d'obtenir un taux de compression R_t (inférieur à l'entropie de la source), des bits de $\mathbf{X}$ sont poinçonnés. Plus précisément, $\lceil (R_t - R_q) \times L(\mathbf{S}) \rceil$ bits de $\mathbf{X}$ sont poinçonnés pour atteindre le taux cible R_t. Il existe différentes techniques pour trouver les meilleures positions de poinçonnage. Par exemple, dans [HBS04], les bits de $\mathbf{X}$ sont insérés ligne par ligne dans une matrice carrée et sont poinçonnés colonne par colonne. Dans notre cas, la technique de poinçonnage donnant les meilleurs résultats consiste à espacer régulièrement les positions des bits poinçonnés. Les bits poinçonnés sont séparés de $\lfloor (R_t - R_q) \times L(\mathbf{S})/L(\mathbf{X}) \rfloor - \delta$ positions bits, où $\delta = 1$ si $(R_t - R_q) \times L(\mathbf{S})/L(\mathbf{X}) \in \mathbb{N}$ et $\delta = 0$ sinon. Dans le cas, où le nombre de bits à poinçonner est supérieur à la moitié de la longueur de $\mathbf{X}$, cette méthode de positionnement des bits poinçonnés est utilisée pour trouver les positions des bits non poinçonnés. En utilisant cette méthode, il suffit de donner au décodeur uniquement l'intervalle entre deux bits poinçonnés (on suppose en effet que le premier bit est poinçonné) pour retrouver l'intégralité des positions de poinçonnage.

5.3.2 Codes quasi-arithmétique avec recouvrement d'intervalle

Une solution au problème du codage de source distribué à l'aide de codes QA poinçonnés a été présentée dans la sous-section précédente. Nous proposons maintenant une solution alternative, basée sur des codes QARI. Ces codes sont tout à fait similaires aux codes QA classiques, mais ils permettent d'atteindre des taux de compression inférieurs à l'entropie de la source. Ce sont des codes avec perte : ils ne sont pas uniquement décodables. Cependant, nous verrons qu'ils permettent d'atteindre des performances intéressantes lorsqu'une information adjacente est disponible au décodeur.

Le principe de recouvrement d'intervalle est le suivant : lorsque l'intervalle courant est subdivisé selon les probabilités de la source, les sous-intervalles obtenus sont redimensionnés afin qu'ils se recouvrent partiellement. Cette procédure conduit à agrandir la taille de l'intervalle courant après chaque encodage de symbole et donc ainsi à diminuer le taux de compression.

La technique de construction d'un code QARI est très semblable à celle d'un code QA classique détaillée dans le chapitre 2. Seul le partitionnement de l'intervalle courant est modifié à l'encodage de chaque symbole de source. L'intervalle courant est noté $I_c = [i, s[$. Pour un code QA classique, le partitionnement de I_c se fait selon la probabilité

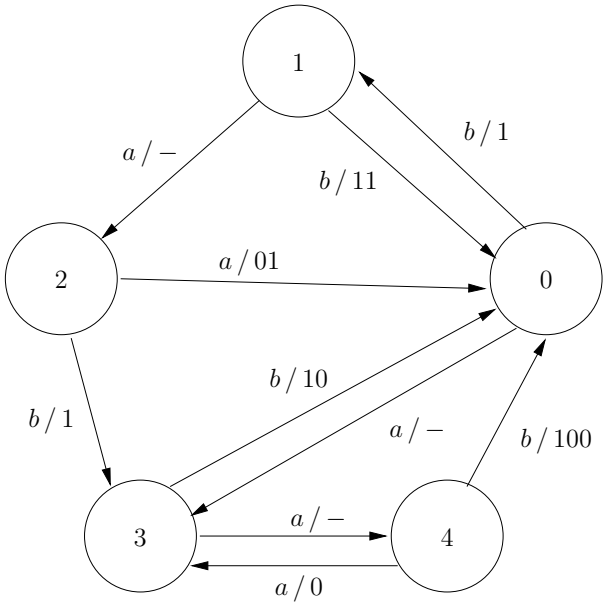

FIG. 5.6 – Code QA avec recouvrement d'intervalle de paramètres $p = 0.7$, $N = 8$ et $\tau = 0.1$

$p = \mathbb{P}(a)$ et conduit aux deux sous-intervalles suivants

$$\begin{cases} I_c^a = & [i\,,\, p \times (s-i)[\\ I_c^b = & [p \times (s-i)\,,\, s[\end{cases} \tag{5.8}$$

Pour un code QARI, le partitionnement de l'intervalle courant est défini par

$$\begin{cases} I_c^a = & [i\,,\, p \times (s-i) + \tau N/2[\\ I_c^b = & [p \times (s-i) - \tau N/2\,,\, s[\end{cases} \tag{5.9}$$

Ainsi, les deux sous-intervalles obtenus se recouvrent sur une largeur de τN. Il faut noter que cette technique de recouvrement d'intervalle peut s'appliquer aussi bien aux codes arithmétiques classiques ($N = 1$) qu'aux codes QA ($N > 1$ et avec des intervalles entiers).
En utilisant cette technique de partitionnement d'intervalle, on peut obtenir une machine à états finis représentant un code QARI.

Exemple 5.3: Sur la figure 5.6 est représenté un code QARI obtenu pour $N = 8$, $p = 0.7$ et $\tau = 0.1$. L'automate d'encodage est tout à fait similaire à celui d'un code QA classique, mais il n'est pas uniquement décodable. On voit par exemple, que l'encodage des séquences baa et abb mène à la même séquence de bits 101.

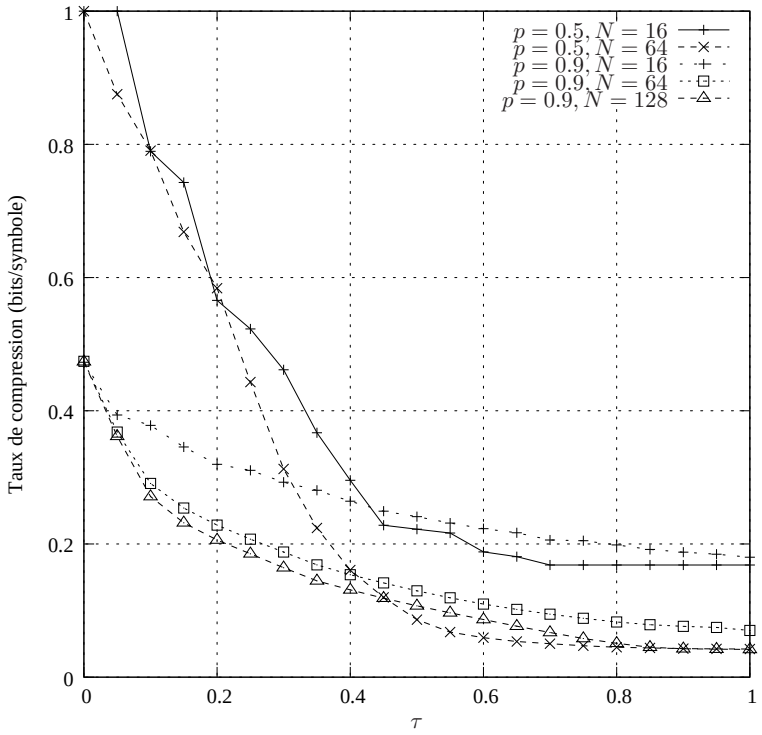

FIG. 5.7 – Taux de compression en fonction de τ de codes QARI pour $p = 0.5$ et $p = 0.9$.

Les performances en compression d'un code QARI peuvent être calculées de la même façon que pour un code QA classique, comme expliqué dans la section 4.4.1. Nous avons représenté le taux de compression des codes QARI en fonction de τ pour des sources de probabilité $p = 0.5$ et $p = 0.9$ sur la figure 5.7. On peut remarquer sur cette figure l'effet du paramètre τ de recouvrement ainsi que du paramètre N du code QA. Les codes QARI permettent, comme nous venons de le constater, d'atteindre des taux de compression inférieurs à l'entropie de la source. Cependant, ces codes ne sont pas uniquement décodables. Nous les utiliserons dans le contexte de codage de source distribué, lorsqu'une information adjacente est disponible au décodeur. Cette information est utilisée pour lever l'ambiguïté induite par le recouvrement d'intervalle. Dans la section suivante, nous montrerons comment cette information est utilisée au niveau du décodeur.

5.3.3 Décodage souple avec information adjacente

Nous allons détailler dans cette sous-section l'algorithme de décodage souple avec information adjacente utilisé dans le schéma de codage de source distribué proposé dans ce chapitre.

Pour chaque transition $t(i,j)$ entre les états α_i et α_j de l'automate d'encodage d'un code QA ou d'un code QARI, nous allons appeler $b_t(i,j)$ la séquence de bits émise par cette transition et $\overline{b_t}(i,j)$ la longueur de cette séquence. Dans le schéma de codage de source distribué présenté sur la figure 5.4, le décodeur dispose du message binaire $\mathbf{X}$, ainsi que de l'information adjacente $\mathbf{Y}$ pour réaliser l'estimation de $\mathbf{S}$. Afin d'exploiter de manière optimale l'information adjacente (sur les symboles de $\mathbf{S}$), nous allons utiliser un modèle d'état de type bit/symbole pour réaliser un décodage sur treillis. Les états de ce modèle sont définis par les paires de variables aléatoires $\mathcal{V}_k = (N_k, M_k)$, où N_k représente l'état de l'automate d'encodage du code QA et M_k les valeurs possibles d'horloge bit à l'instant symbole k. Les probabilités de transition sur ce modèle sont données par

$$\begin{array}{l} \forall(\alpha_i,\alpha_j) \in \Omega_e \times \Omega_e, \forall(m,m') \in \mathbb{N} \times \mathbb{N} \\ \mathbb{P}(N_k = \alpha_i, M_k = m' \,|\, N_{k-1} = \alpha_j, M_{k-1} = m) = \\ \left\{ \begin{array}{ll} \mathbb{P}(N_k = \alpha_i \,|\, N_{k-1} = \alpha_j) & \text{si } m' - m = \overline{b_t}(i,j) \\ 0 & \text{sinon,} \end{array} \right. \end{array} \tag{5.10}$$

où les probabilités $\mathbb{P}(N_k = \alpha_i \,|N_{k-1} = \alpha_j)$ se déduisent à partir de l'automate du code QA et des statistiques de la source. L'information adjacente peut aisément être intégrée au niveau du décodage grâce à ce modèle. Pour tenir compte de cette information supplémentaire, la métrique de branche $\gamma_k(N_k, M_k \,|N_{k-1}, M_{k-1})$ d'une transition déclenchée par un symbole s de l'alphabet est modifiée de la façon suivante

$$\begin{array}{l} \gamma_k(N_k = \alpha_i, M_k = m' \,|\, N_{k-1} = \alpha_j, M_{k-1} = m) = \\ \quad \mathbb{P}(N_k = \alpha_i, M_k = m' | N_{k-1} = \alpha_j, M_{k-1} = m) \times \mathbb{P}(X_m^{m'} = b_t) \times \mathbb{P}(S_k = s \,|\, Y_k), \end{array} \tag{5.11}$$

où la probabilité $\mathbb{P}(X_m^{m'} = b_t)$ est déduite de $\mathbf{X}$ et $\mathbb{P}(S_k = s \,|\, Y_k)$ est déduite de la probabilité π de corrélation entre $\mathbf{S}$ et $\mathbf{Y}$.

L'algorithme BCJR est appliqué sur le modèle (N_k, M_k) défini ci-dessus. Pour chaque état $v = (n,m), n \in \Omega_e, 1 \leq m \leq L(\mathbf{X})$ du treillis, les fonctions de probabilités suivantes sont calculées

$$\alpha_k(v) = \mathbb{P}(\mathcal{V}_k = v; \mathbf{Y}_1^k) \tag{5.12}$$

$$\beta_k(v) = \mathbb{P}(\mathbf{Y}_{k+1}^{L(\mathbf{S})} \,|\, \mathcal{V}_k = v), \tag{5.13}$$

pour $1 \leq k \leq L(\mathbf{S})$. On définit maintenant l'ensemble $\Gamma(s)$ des paires d'états $(v = (n,m), v' = (n',m'))$ dans le treillis telles que la transition entre v et v' soit déclenchée par le symbole $s \in \mathcal{A}$. Ainsi, les probabilités a posteriori au niveau symbole sont

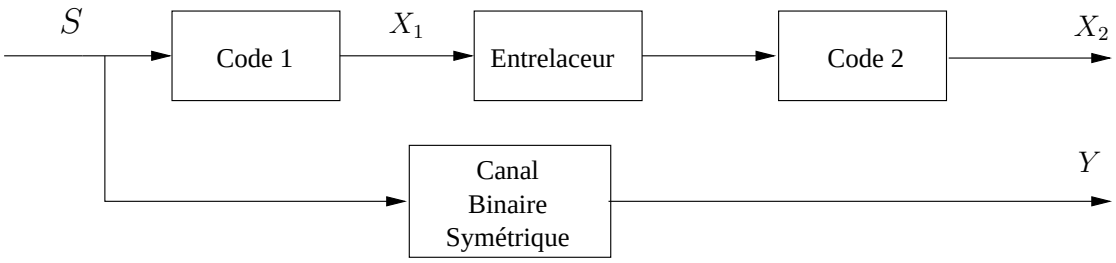

FIG. 5.8 – Encodeur d'une structure itérative en série

obtenues sur le treillis par

$$\begin{aligned} &\forall k \in [1, L(\mathbf{S})],\ \forall s \in \mathcal{A}, \\ &\mathbb{P}(S_k = s \,|\, X_1^{L(\mathbf{X})}; Y_1^{L(\mathbf{S})}) \propto \sum_{(v,v') \in \Gamma(s)} \alpha_k(v)\, \beta_{k+1}(v')\, \gamma_{k+1}(v' \,|\, v), \end{aligned} \tag{5.14}$$

où $\gamma_{k+1}(v' \,|\, v)$ est calculé en utilisant l'équation 5.11. L'algorithme BCJR appliqué sur ce modèle d'état permet ainsi de calculer les probabilités marginale postérieures au niveau symbole, et donc de minimiser le TES en sortie du décodeur.
Nous allons maintentant présenter des structures itératives comprenant des codes QA poinçonnés et codes QARI dans le cadre du schéma de codage de source distribué proposé.

5.3.4 Structures itératives

Les Turbo-codes [BGP93] ont montré que des gains significatifs en terme de performance de décodage peuvent être atteints en utilisant des structures itératives. Deux types de structures itératives sont principalement utilisés dans la littérature : les structures en série et les structures parallèle. Nous allons détailler dans cette section les types de structures que nous avons utilisés en codage de source distribué.

5.3.4.1 Structure en série

Le schéma d'encodage d'une structure en série classique est donné sur la figure 5.8. Ce schéma sera utilisé avec un code QA poinçonné comme Code 1 et un CC comme Code 2. L'utilisation de codes QARI dans une structure en série ne peut pas apporter d'amélioration en termes de performances. En effet, si l'on utilise un code QARI comme Code 1 de la figure 5.8, le train binaire $\mathbf{X}_1$ est connu parfaitement. Le code 2 ne pourra donc apporter aucune information supplémentaire sur X_1. En revanche, en utilisant un code QA poinçonné, le code 2 aura pour rôle, en collaboration avec le Code 1, de retrouver au mieux les bits poinçonnés. Les codes QARI ne seront donc pas intégrés dans des structures en série.

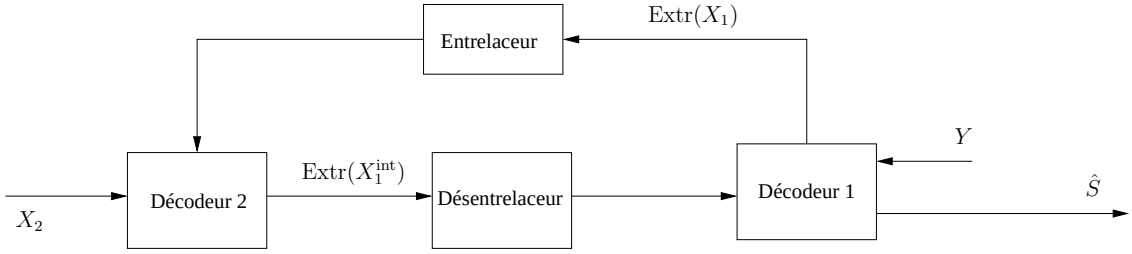

FIG. 5.9 – Décodeur d'une structure itérative en série

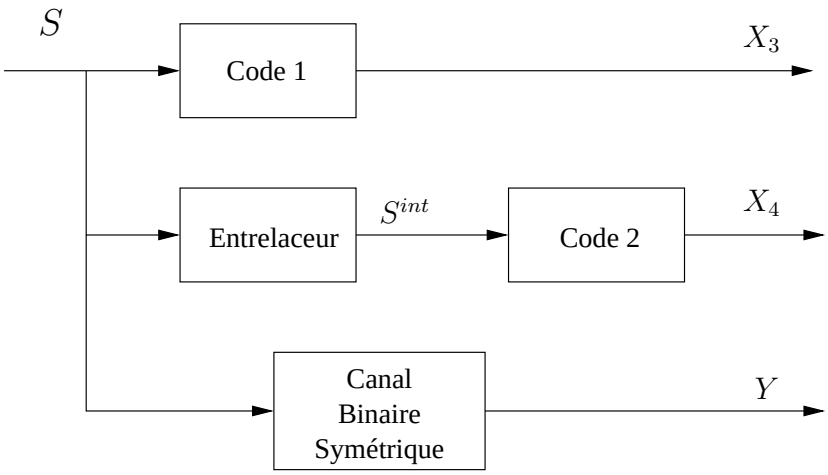

FIG. 5.10 – Encodeur d'une structure itérative en parallèle

Nous utiliserons des CC récursifs systématiques dont tous les bits systématiques seront poinçonnés, ainsi que le nombre de bits de parité permettant d'atteindre le taux de compression désiré.

Le décodeur itératif associé à l'encodeur de la figure 5.8 est représenté sur la figure 5.9. Les décodeurs 1 et 2 s'échangent leurs informations extrinsèques respectives. L'estimation au niveau de chaque décodeur est réalisée au moyen d'un algorithme BCJR appliqué sur un modèle associé au code considéré (modèle bit/symbole pour les codes QA et modèle classique pour les CC). Il faut noter que l'information adjacente Y n'est disponible pour cette structure qu'au décodeur 1 (décodeur QA ici). Cette information sera utilisée de la même façon que pour un code QA poinçonné seul, comme expliqué dans la section 5.3.3. Le terme $\mathbb{P}(X_m^{m'} = b_t)$ de l'équation 5.11 est simplement remplacé par $\mathbb{P}(Extr(X_m^{m'}) = b_t)$.

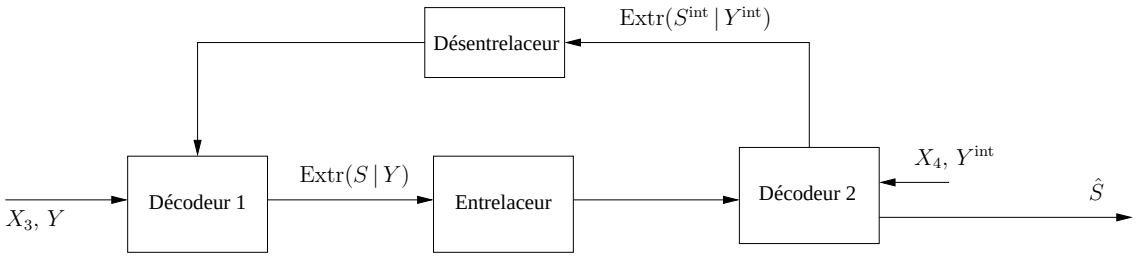

FIG. 5.11 – Décodeur d'une structure itérative en parallèle

Code 1	Code 2
QA poinçonné	QA poinçonné
QA avec recouvrement	QA avec recouvrement
QA avec recouvrement	CC

TAB. 5.1 – Structures parallèles testées

5.3.4.2 Structure en parallèle

Le schéma d'encodage d'une structure en parallèle classique est donné sur la figure 5.10. Le message de symboles $\mathbf{S}$ et sa version entrelacée sont encodés par les codes 1 et 2 respectivement, donnant les trains binaires $\mathbf{X}_3$ et $\mathbf{X}_4$. Les différentes combinaisons (Code 1,Code 2) qui ont été testées sont répertoriées dans le tableau 5.1.

Lorsque les codes QARI sont utilisés, le coefficient τ est choisi de façon à atteindre le taux de compression désiré. Dans le cas des codes QA classiques, on poinçonne de façon régulière des bits de $\mathbf{X}_3$ et de $\mathbf{X}_4$.

Il faut noter que lorsque la source est une source avec une mémoire d'ordre 1, seul le code 1 peut tenir compte des statistiques d'ordre 1 de la source. En effet, la mémoire est cassée dans la seconde branche de l'encodeur par l'entrelaceur. Ainsi, le code 2 ne peut pas tenir compte des statistiques d'ordre 1 éventuelles de la source.

Le décodeur itératif associé à l'encodeur parallèle est représenté sur la figure 5.11. Les décodeurs 1 et 2 disposent du train binaire (X_3 ou X_4), de l'information adjacente $\mathbf{Y}$ (entrelacée ou non) ainsi qu'une information extrinsèque pour réaliser leur estimation. L'information extrinsèque à la sortie d'un décodeur est calculée en retirant à la probabilité a posteriori l'information a priori, soit la probabilité p de la source, l'information adjacente et l'extrinsèque venant de l'autre décodeur. Une fois calculée, l'information extrinsèque est fournie à l'autre décodeur, et utilisée comme l'information adjacente : elle est intégrée dans la métrique de branche des transitions (équation 5.11).

5.3.5 Résultats de simulation

Le schéma de codage de source distribué proposé sur la figure 5.4 ainsi que les structures itératives de la section précédente ont été testés en simulation sur des sources sans mémoire et des sources avec une mémoire d'ordre 1, en utilisant l'information adjacente $\mathbf{Y}$ au décodeur.

5.3.5.1 Sources binaire sans mémoire

Dans un premier temps, nous avons considéré des sources binaires sans mémoire de probabilité $\mathbb{P}(a) = 0.9$ et $\mathbb{P}(a) = 0.8$. Les entropies de ces deux sources sont respectivement égales à 0.4690 bits et 0.7219 bits. Nous avons appliqué les schémas proposés pour des séquences de $L(\mathbf{S}) = 100$ symboles, et moyenné le TES en sortie du décodeur sur 10^5 réalisations de la source pour un taux de compression global de 0.4 bits par symbole.

Les résultats sont donnés sur les figures 5.12 ($\mathbb{P}(a) = 0.9$) et 5.14 ($\mathbb{P}(a) = 0.8$) pour la solution utilisant des codes QA poinçonnés et sur la figure 5.13 ($\mathbb{P}(a) = 0.9$) pour celle utilisant des codes QARI. Sur ces courbes, le taux d'erreur en sortie du décodeur est représenté en fonction de la corrélation $H(\mathbf{Y} \mid \mathbf{X})$ entre $\mathbf{S}$ et $\mathbf{Y}$.

Dans le cas des codes QA classiques (figures 5.12 et 5.14), nous avons considéré des codes QA adaptés à la probabilité de la source et construits selon la méthode décrite dans le chapitre 2 pour $N = 8$. Pour la source telle que $\mathbb{P}(a) = 0.9$, le code QA adapté à cette source a un taux de compression moyen de 0.48 bits par symbole source. Ainsi, pour atteindre un taux de 0.4 bits par symbole, 8 bits ont été poinçonnés pour chaque réalisation de la source. Le code QA associé à la source de probabilité $\mathbb{P}(a) = 0.8$ a un taux de compression de 0.72 bits par symbole. Nous avons donc poinçonné 32 bits par message binaire issu de l'encodage de la source. Pour la structure QA-QA en parallèle, le taux de compression issu de chaque branche a été fixé à 0.2 bits par symboles afin d'atteindre un taux global de 0.4 bits par symboles. Enfin, le CC de la structure QA-CC en série est un code (21,37) octal. En sortie de ce CC, les bits systématiques ont tous été poinçonnés ainsi qu' un nombre de bits de parité suffisant pour atteindre le taux global désiré. Sur ces figures, nous avons également ajouté les performances obtenues par des turbo codes parallèles classiques. Ces turbo codes sont composés de deux CC (21,37) octal en parallèle. Nous avons appliqué ces turbo codes à des séquences courtes (100 symboles) et à des séquences longues (5000 symboles). La probabilité stationnaire de la source $\mathbb{P}(a)$ est utilisée par le turbo code, puisqu'elle l'est également par les codes QA.

Pour les codes QARI, le coefficient de recouvrement a été adapté afin d'obtenir un taux global très proche de 0.4 bits par symbole. Pour les structures parallèles, le taux a été réparti sur les deux branches.

On peut remarquer que les codes QA poinçonnés et les codes QARI sont très compétitifs par rapport aux turbo-codes pour des séquences courtes. Ceci peut s'expliquer par le fait que les turbo codes sont moins performants pour des séquences courtes. Les structures itératives proposées permettent d'améliorer sensiblement les performances de décodage, notamment quand la corrélation entre $\mathbf{S}$ et $\mathbf{Y}$ est importante. En revanche, pour des séquences longues, les performances des turbo sont sensiblement améliorées.

5.3.5.2 Sources binaires avec mémoire

Enfin, nous avons appliqué les schémas proposés à une source avec une mémoire d'ordre 1, définie par $\mathbb{P}(a) = 0.9$ et $\rho_m = 0.9$. L'entropie de cette source est de 0.1164 bits. Le code QA adapté à cette source a un taux de compression de 0.125 bits par symbole. Les résultats associés à cette source et à différents schémas de décodage sont proposés sur la figure 5.15 pour un taux de compression de 0.09 bits par symbole. Ces performances sont comparées à celles obtenues avec un turbo code parallèle. Il faut noter que les statistiques d'ordre 1 de la source ne peuvent être utilisées que par le premier constituant du turbo code (la mémoire est cassée dans la seconde branche par l'entrelaceur).

Les mêmes conclusions peuvent être tirées dans le cas des sources avec mémoire. Les performances des codes QA ainsi que celles de la structure QA-CC en série sont compétitives par rapport aux turbo codes pour des courtes séquences. La structure QA-QA en parallèle n'a pas été représentée ici car elle n'est pas très adaptée aux sources avec mémoire. En effet, comme la version entrelacée de la source ne comporte plus de mémoire, un code QA sans mémoire doit être utilisé dans la seconde branche, nécessitant un poinçonnage très important.

5.4 Conclusion

Nous avons présenté dans ce chapitre deux schémas de codage de source distribué utilisant des codes QA, l'un réalisant la compression par poinçonnage et l'autre se basant sur des codes QA avec recouvrement d'intervalles, réalisant directement la compression. Ces deux schémas ont été intégrés dans des structures itératives afin d'améliorer leurs performances respectives. Les résultats de simulation ont permis de montrer que ces schémas de codage de source distribué utilisant des codes QA présentent des performances intéressantes comparativement aux schémas basés sur des codes de canal pour des séquences courtes notamment. Nous verrons en conclusion générale quelques perspectives qui pourraient permettre d'améliorer davantage les performances de ces schémas de codage de source distribué utilisant des codes QA.

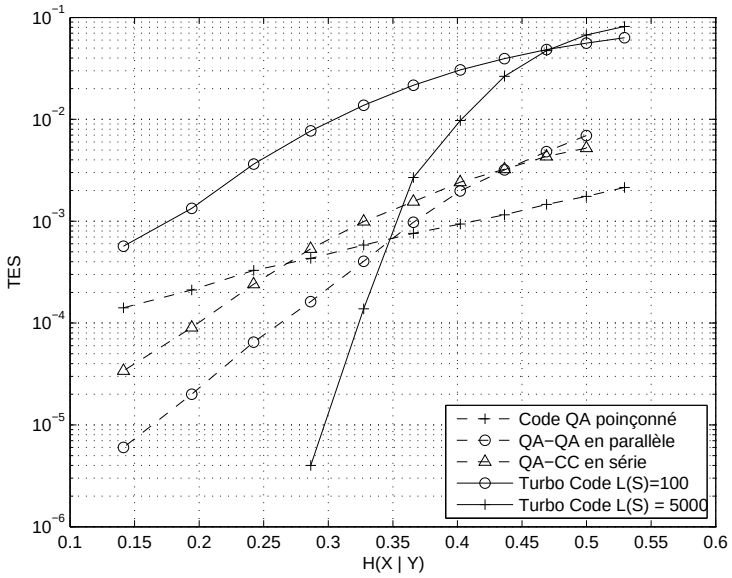

FIG. 5.12 – Performances de différents schémas de codage de source distribué à un taux de 0.4 bps pour une source sans mémoire avec $p = 0.9$.

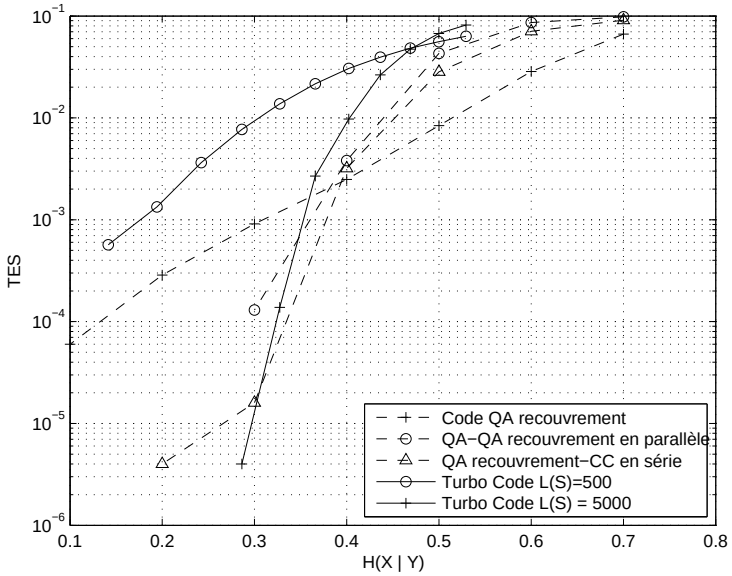

FIG. 5.13 – Performances de différents schémas de codage de source distribué à un taux de 0.4 bps pour une source sans mémoire avec $p = 0.9$

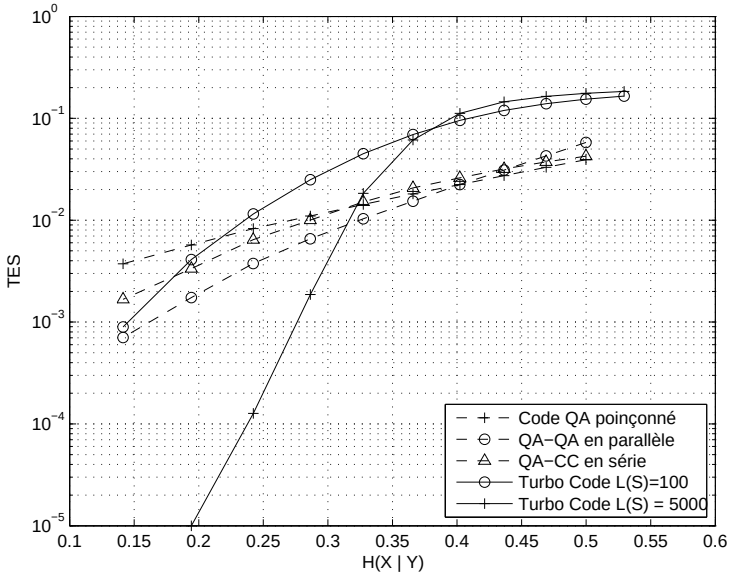

FIG. 5.14 – Performances de différents schémas de codage de source distribué à un taux de 0.4 bps pour une source sans mémoire avec $p = 0.8$

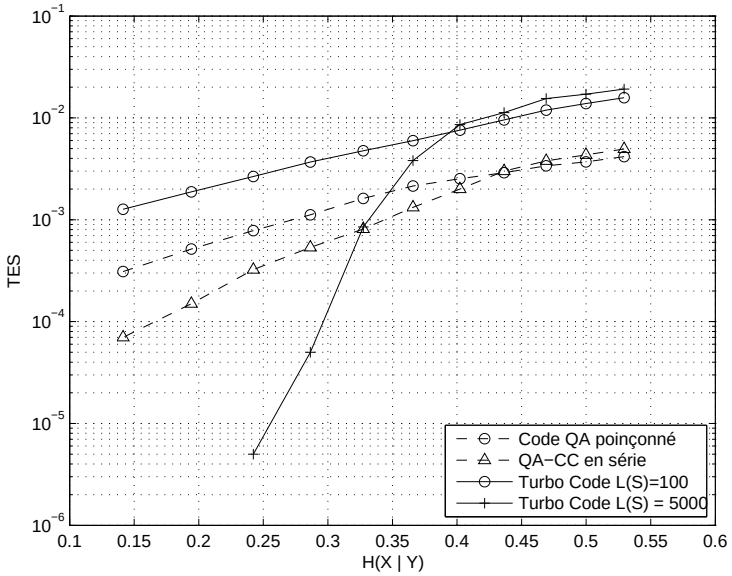

FIG. 5.15 – Performances de différents schémas de codage de source distribué à un taux de 0.09 bps pour une source avec mémoire d'ordre 1 avec $p = 0.9$ et $\rho_m = 0.9$.

Conclusion

"Pour connaître l'origine et la qualité d'un vin, il n'est pas nécessaire de boire le tonneau entier"

Oscar Wilde

L'étude menée dans cette thèse s'inscrit dans le contexte du codage conjoint source-canal. Depuis quelques années, les chercheurs dans le domaine se sont de plus en plus penchés sur ce problème, notamment sur la construction de codes sources robustes ou de schémas permettant de rendre plus robustes les codes sources classiques. Dans ce chapitre, nous synthétisons l'ensemble des contributions apportées dans ce domaine, puis nous donnerons quelques perspectives.

Synthèse

Décodage souple des CLV et codes QA

Les codes de source, de type CLV ou codes QA présentent de très bonnes performances en terme de compression. Ils sont de ce fait souvent utilisés dans les schémas de compression moderne. Cependant, ils sont très sensibles au bruit de transmission, notamment à cause du phénomène de désynchronisation. Afin d'améliorer leurs performances de décodage, un modèle d'état optimal permettant le décodage souple de ces codes a été introduit. Ce modèle prend en compte une information sur le nombre de symboles émis. Cependant, la complexité (quadratique avec la longueur de la séquence) sur ce modèle est trop importante pour qu'il soit utilisé tel quel en pratique. Le modèle d'état agrégé pour le décodage souple des CLV proposé dans [JMG05] a été rappelé puis nous l'avons étendu aux codes QA. Ce modèle d'état permet de rendre la complexité du décodage linéaire avec la longueur de la séquence. Il est paramétré par un entier T qui dose un compromis entre performances et complexité du décodage.
Le calcul des probabilités marginales au niveau symbole n'est pas directement possible sur le modèle agrégé car l'horloge symbole n'est pas gardée en mémoire tout

au long du décodage. Cette marginale est nécessaire lorsque l'on souhaite minimiser le taux d'erreur symbole ou réaliser un décodage itératif. Nous avons donc proposé un algorithme permettant d'estimer ces probabilités marginales en utilisant le modèle agrégé. Les simulations ont montré que l'estimation de ces marginales selon l'algorithme proposé est très fiable.
De nombreuses techniques visant à ajouter de la redondance structurée aux CLV ou codes QA ont été développées ces dernières années, afin d'améliorer les performances de ces codes dans un contexte de décodage conjoint source/canal. Nous avons proposé une solution alternative basée sur l'introduction de contraintes de longueur partielles référençant plusieurs moments du processus de décodage. Ces contraintes permettent d'aider le décodeur à se resynchroniser en tous ces moments. Cette approche permet d'obtenir de très bons résultats en terme de performance, comparativement aux méthodes classiques de la littérature. De plus, cette approche est très flexible dans le sens où la redondance n'est pas ajoutée directement dans le train binaire original et peut donc ainsi être envoyée séparément.

Analyse des propriétés de resynchronisation des codes

Les CLV (et les codes QA) sont très sensibles au bruit de transmission, notamment à cause du phénomène de désynchronisation. De nombreux chercheurs se sont donc penchés sur les propriétés de resynchronisation des CLV de type Huffman. Maxted et Robinson ont notamment proposé une méthode pour calculer l'espérance de la longueur de propagation d'une erreur bit (MEPL). Cette méthode a ensuite été étendue pour calculer la variance de cette longueur de propagation (VEPL). Des résultats ont ensuite montré que ces deux quantités reflétaient les performances des CLV pour un décodage dur. Plus elles sont élevées, moins le code est performant en terme de taux d'erreur en sortie du décodeur. Swaszek et Di Cicco ont ensuite adapté la méthode de Maxted et Robinson pour calculer la loi de probabilité du nombre de symboles supplémentaires décodés (positif ou négatif) (variable ΔS).
Nous avons dans un premier temps étendu ces méthodes afin de pouvoir calculer ces quantités pour les codes QA. Il nous est donc possible de calculer les quantités MEPL,VEPL et la densité de probabilité de ΔS pour les CLV et codes QA. Le calcul de ces quantités est néanmoins valable que lorsqu'une seule erreur bit se produit dans le train binaire. Nous avons ensuite proposé une méthode permettant d'estimer ces quantités lorsque un message est transmis sur un canal de type BABG, caractérisé par son rapport signal à bruit.
Nous avons montré que la loi de probabilité de ΔS permettait de quantifier la quantité d'information apportée par une contrainte de terminaison sur un treillis de type bit/symbole, et donc de refléter les performances des CLV lorsqu'un décodage souple est appliqué sur ce modèle. Nous avons également montré que $\Delta S \bmod T$ pouvait être utilisée pour quantifier l'information apportée par la contrainte de terminaison sur un modèle agrégé de paramètre T. Nous avons donc pu analyser les performances

des CLV et codes QA sur le modèle agrégé. On peut notamment prévoir quelles valeurs de T sont les plus adaptées à chaque code en fonction du rapport signal à bruit, ainsi qu'estimer le T permettant d'atteindre les performances obtenues sur le treillis bit/symbole.

Codage de source distribué à l'aide de codes QA

Le codage de sources distribué concerne les signaux corrélés qui sont encodés séparément et décodés conjointement. La plupart des solutions pratiques au problème du codage de source distribué sont mises en oeuvre avec des codes de canal, notamment de type turbo-code. Ces codes sont très efficaces lorsque les sources en entrée sont uniformes et lorsqu'ils travaillent sur des longues séquences, et ne sont pas bien adaptés à travailler avec des sources très asymétriques, contrairement aux codes QA.
Nous avons proposé deux solutions pratiques au problème du codage de source distribué utilisant des codes QA. La première solution utilise des codes QA classiques dont la sortie est poinçonnée afin d'atteindre des taux de compression très faibles. La solution a été mise en oeuvre conjointement avec Xavi Artigas, doctorant à l'université Polytechnique de Catalogne, dans le cadre du projet européen IST-DISCOVER. Elle est basée sur un nouveau type de codes : les codes QA avec recouvrement d'intervalle. Ces codes réalisent directement la compression. Ils ne sont pas uniquement décodables, mais l'information adjacente présente dans les schémas de codage de source distribué aide à lever l'ambiguïté induite par le recouvrement d'intervalle. Ces deux solutions ont été intégrées dans des structures itératives afin d'améliorer les performances. Les résultats obtenus par ces deux schémas de codage de source distribué sont très compétitifs par rapport aux solutions basées sur des codes de canal, notamment lorsque la source en entrée présente une forte asymétrie.

Perspectives

Nous proposons quelques développements possibles pour un futur travail de recherche.

Étude des propriétés de resynchronisation sur canal à effacement

L'*error state diagram* utilisé pour le calcul des propriétés de resynchronisation des codes lorsqu'une erreur bit se produit peut être adapté au cas où un bit serait effacé. Le calcul des MEPL, VEPL, et de la densité de probabilité de ΔS pourrait ainsi être effectué dans ce cas. Une estimation de ces quantités sur un canal à effacement (caractérisé par une probabilité d'effacement) pourrait ensuite être réalisée.
La connaissance de ces propriétés permettrait d'adapter le poinçonnage en fonction du code QA dans le schéma de codage de source distribué que l'on a proposé dans le chapitre 5. Certains états de l'automate de décodage des codes QA doivent être plus

sensibles au poinçonnage que d'autres. Les propriétés de resynchronisation des codes QA sur un canal à effacement permettraient de trouver les états les moins sensibles au poinçonnage. De plus, la connaissance de l'information adjacente peut être intégrée dans l'*error state diagram* afin de se placer dans la situation décrite dans le chapitre 5.

Utilisation des EXIT charts pour les structures itératives

Les performances des structures itératives présentées dans le chapitre 5 peuvent certainement être améliorées en se basant sur l'étude des EXIT charts [Bri01]. La manière de calculer ces courbes a été modifiée dans [Jas08] afin de considérer des sources non-uniformes en entrée. Cette méthode pourrait être utilisée dans notre cas. L'étude des EXIT charts permettrait de prévoir la convergence des structures itératives utilisées dans le chapitre 5. La convergence d'une structure itérative peut également être modifiée par l'amoindrissement des probabilités extrinsèques en utilisant un certain coefficient. Cette idée a été proposée à l'origine par Claude Berrou et Alain Glavieux. Les EXIT charts peuvent permettre notamment de prévoir si un tel coefficient peut permettre d'améliorer la convergence des structures itératives du chapitre 5.

Annexe A

Algorithmes de Viterbi et BCJR

"Il parle tout le temps de toi. Tu l'as couché tous les soirs pendant six ans. C'est avec toi qu'il a appris à faire des blagues, c'est avec toi qu'il a appris à parler, et à écrire. C'est idiot mais tu connais beaucoup d'enfants qui disent : « Je vais préparer mes impedimenta » ?"

Arnaud Desplechin

Les algorithmes de Viterbi [Vit67] et BCJR [BCJR74] ont été utilisés dans cette thèse pour réaliser les estimations bayésiennes au décodeur des différentes structures utilisées. Nous allons ici détailler ces deux algorithmes tels qu'ils ont été utilisés dans ce document.
Dans cette annexe, nous supposerons qu'une séquence de symboles $\mathbf{S} = S_1, \ldots, S_{L(\mathbf{S})}$ à valeurs dans un alphabet $\mathcal{A} = \{a_1, \ldots, a_n\}$ est encodé par un CLV ou un code QA. L'encodage de $\mathbf{S}$ produit le train binaire $\mathbf{X} = X_1, \ldots X_{L(\mathbf{X})}$. Nous supposerons que le décodeur reçoit une suite d'observations $\mathbf{Y}$ sur la séquence $\mathbf{X}$ qui permettent de calculer les probabilités $\mathbb{P}(X_k|Y_k), 1 \leq k \leq L(\mathbf{X})$.
Nous allons détailler les algorithmes de Viterbi et BCJR sur un treillis de type bit/symbole. Ainsi, les états du treillis sont notés $\nu_k = (N_k, T_k)$, où N_k représente les états de l'automate représentant le code et M_k les valeurs d'horloge symbole possibles à l'instant bit k. Les transitions sur ce modèle sont étiquetées par un bit en entrée et un ou plusieurs symboles en sortie. Les probabilités de transition sur ce modèle se calculent comme expliqué dans le document en utilisant les statistiques de la source.

A.1 Algorithme de Viterbi

L'algorithme de Viterbi va permettre de minimiser le taux d'erreur séquence dans le sens où cet algorithme sélectionne la séquence de bits (ou de symboles) de plus forte

probabilité a posteriori. Ainsi, la séquence $\hat{X}$ sélectionnée par l'algorithme de Viterbi est telle que la quantité $\mathbb{P}(\hat{X}|\mathbf{Y}$ soit maximisée sur l'ensemble de séquences binaires possibles de longueur $L(\mathbf{X})$.
Pour sélectionner cette séquence de probabilité a posteriori maximale, l'algorithme procède comme suit. Une métrique $\mu_k(m)$ va être associé à chaque état m du treillis de l'instant bit k. Cette métrique est initialisée pour l'instant bit 0. Dans notre cas, la métrique de l'état initial de l'automate du code associée à l'horloge symbole 0 est initialisée à 1, alors que les métriques des autres états du treillis sont mises à 0. Pour chaque état m du treillis à l'instant bit k, on va également associer un état "survivant" que l'on note $p_k(m)$. Les "survivants" sont initialisés à 0. Ensuite, à chaque instant bit k, les opérations suivantes sont réalisées :

1. Pour chaque état m du treillis à l'instant bit k, et pour chaque état m' du treillis à l'instant bit $k+1$, on calcule la métrique temporaire
$$\mu^t_{k+1}(m') = \mu_k(m) \times \mathbb{P}(X_{k+1} = i(m, m')|Y_{k+1}), \tag{A.1}$$
où $i(m, m')$ est égal au bit associé à la transition entre m et m'.
2. Si $\mu^t_{k+1}(m') > \mu_{k+1}(m')$, alors on met à jour la métrique $\mu_{k+1}(m') = \mu^t_{k+1}(m')$ et on met à jour le "survivant" $p_{k+1}(m') = m$.

Ces opérations sont répétées pour chaque instant bit. Puis, on sélectionne l'état du dernier instant bit $L(\mathbf{X})$ qui a la plus forte métrique et dont l'horloge symbole associée est égale à $L(\mathbf{S})$. On appelle cet état $e_{L(\mathbf{X})}$. On sélectionne ensuite le survivant de $e_{L(\mathbf{X})}$: $e_{L(\mathbf{X})-1} = p_{L(\mathbf{X})}(e_{L(\mathbf{X})})$. On remonte ainsi jusqu'à e_1.
Finalement, on obtient une séquence d'états $e_1, \ldots, e_{L(\mathbf{X})}$ du treillis. Cette séquence peut être convertie en une séquence binaire $\hat{\mathbf{X}}$, ou en une séquence de symboles $\hat{\mathbf{S}}$ à partir de l'automate du code. Ces séquences sont de probabilité a posteriori maximale.

A.2 Algorithme BCJR

L'algorithme BCJR permet de sélectionner une séquence de bits ou une séquence de symboles constituées des éléments (bits ou symboles) les plus probables à chaque instant (bit ou symbole). Il repose sur deux passes : une passe avant et une passe arrière. La passe avant permet de calculer pour chaque état m du treillis à l'instant bit k la probabilité $\alpha_k(m) = \mathbb{P}(\nu_k = m; \mathbf{Y}_1^k)$. La passe arrière permet de calculer pour chaque état m du treillis à l'instant bit k la probabilité $\beta_k(m) = \mathbb{P}(Y_{k+1}^{L(\mathbf{X})}|\nu_k = m)$.
Plus précisément, les probabilités α_k sont obtenues récursivement par la relation suivante :
$$\alpha_k(m) = \sum_{m'} \alpha_{k-1}(m') \times \mathbb{P}(X_k = i(m', m)|Y_k). \tag{A.2}$$
Les probabilités β_k sont obtenues par la relation suivante :
$$\beta_k(m) = \sum_{m'} \beta_{k+1}(m') \times \mathbb{P}(X_{k+1} = i(m, m')|Y_{k+1}). \tag{A.3}$$

Les probabilités α_0 doivent être initialisées selon la situation. Dans notre cas, la probabilité α_0 associée à l'état initial de l'automate et dont l' horloge symbole est 0 est initialisée à 1, et 0 pour les autres états. Les probabilités $\beta_{L(\mathbf{X})}$ doivent de même être initialisées. Nous fixons un $\beta_{L(\mathbf{X})}$ non nul à tous les états de l'instant bit $L(\mathbf{X})$ dont l'horloge symbole associée est égale à $L(\mathbf{S})$.
Il faut noter que pour éviter que les probabilités α_k et β_k diminuent rapidement, elles sont normalisées à chaque instant bit k.
Pour obtenir une séquence de bits composés des bits les plus probables à chaque instant, il faut calculer les probabilités a posteriori $\mathbb{P}(X_k = 0; \mathbf{Y})$ pour chaque instant bit. Ces probabilités sont données par la relation suivante

$$\mathbb{P}(X_k = 0; \mathbf{Y}) \propto \sum_{m,m'/i(m,m')=0} \alpha_k(m)\, \beta_{k+1}(m')\, \mathbb{P}(X_k = i(m,m')|Y_k). \tag{A.4}$$

La séquence $\hat{\mathbf{X}}$ est donc constituée des éléments $\hat{x}_k$ définis par

$$\hat{x}_k = \begin{cases} 0 & \text{si } \mathbb{P}(X_k = 0; \mathbf{Y}) > \mathbb{P}(X_k = 1; \mathbf{Y}) \\ 1 & \text{sinon .} \end{cases} \tag{A.5}$$

Cette séquence minimise bien le taux d'erreur bit.
Pour obtenir une séquence de symboles en utilisant le modèle d'état à horloge bit, il faut utiliser la méthode décrite dans la section 3.3.1.2 de ce document. Autrement, il faut procéder tel que nous venons de l'expliquer, mais sur un modèle à horloge symbole.

Glossaire

Acronyme	*Signification*
BABG	Brut additif blanc gaussien
BPSK	Binary phase shift keying
CBS	Canal binaire symétrique
CC	Code convolutif
CLV	Code à longueur variable
ddp	densité de probabilité
DLN	Distance de Levenshtein normalisée
ESD	Error state diagram
LDPC	Low density parity check codes
MEPL	Mean error propagation length
QA	Quasi-arithmétique
QARI	Quasi-arithmétique avec recouvrement d'intervalle
SI	Symbole interdit
SOVA	Soft-output Viterbi algorithm
TEB	Taux d'erreur bit
TES	Taux d'erreur symbole
TESQ	Taux d'erreur séquence
VEPL	Variance of the error propagation length

"Et quand tu m'auras lu, jette ce livre - et sors. Je voudrais qu'il t'eût donné le désir de sortir - sortir de n'importe où, de ta ville, de ta famille, de ta chambre, de ta pensée. N'emporte pas mon livre avec toi. [...] Que mon livre t'enseigne à t'intéresser à, toi plus qu'à lui-même, - puis à tout le reste plus qu'à toi.

Si je cherchais tes aliments, tu n'aurais pas de faim pour les manger, si je te préparais ton lit, tu n'aurais pas sommeil pour y dormir. Jette mon livre ; dis-toi bien que ce n'est là qu'une des milles postures possibles en face de la vie. Cherche la tienne. Ce qu'un autre aurait aussi bien fait que toi, ne le fais pas. Ce qu'un autre aurait aussi bien dit que toi, ne le dis pas, - aussi bien écrit que toi, ne l'écris pas. Ne t'attache en toi qu'à ce que tu sens qui n'est nulle part ailleurs qu'en toi-même, et crée de toi, impatiemment ou patiemment, ah ! le plus irremplaçable des êtres"

André Gide

Publications

Journaux :

– S. Malinowski, H. Jégou et C. Guillemot - Synchronization recovery and state model reduction for soft decoding of Variable Length Codes - *IEEE Trans. on Information Theory* - Janv. 2007.

– S. Malinowski, H. Jégou et C. Guillemot - Error recovery properties and soft decoding of quasi-arithmetic code - *EURASIP Journal on Applied Signal Processing* - Janv. 2008.

– S. Malinowski, H. Jégou et C. Guillemot, Computation of posterior marginals on aggregated state models for soft source decoding - à paraître dans *IEEE Trans. on Communications*.

Conférences :

– H. Jégou, S. Malinowski et C. Guillemot - Trellis state aggregation for soft decoding of Variable Length Code - *IEEE Workshop on Signal Processing Systems* - Athènes, Nov. 2005.

– S. Malinowski, H. Jégou et C. Guillemot - On the link between the synchronization recovery and soft decoding of Variable Length Codes - *IEEE Conference on Source and Channel Coding* - Munich, Avr. 2006.

– S. Malinowski, H. Jégou et C. Guillemot - Error recovery properties of quasi-arithmetic codes and soft decoding with length constraint - *IEEE International Symposium on Information Theory* - Seattle, Juil. 2006

– X. Artigas, S. Malinowski, C. Guillemot et L. Torres - Overlapped quasi-arithmetic codes for distributed video coding - *IEEE International Conference on Image Processing* - San Antonio, Sept. 2007

– X. Artigas, S. Malinowski, C. Guillemot et L. Torres - Overlapped arithmetic codes with memory - *Proc. of EUSIPCO 2008* - Lausanne, Aout 2008

Bibliographie

[Abr63] N. Abramson. *Information theory and coding*. McGraw-Hill, New York, 1963.

[AG02] A. Aaron and B. Girod. Compression with side information using turbo-codes. In *Proc. Data Compression Conf.*, pages 252–261, Jan. 2002.

[AM84] J. Anderson and S. Mohan. Sequential coding algorithms : a survey and cost analysis. *IEEE Trans. Commun.*, 32 :169–176, 1984.

[AMGT07] X. Artigas, S. Malinowski, C. Guillemot, and L. Torres. Overlapped quasi-arithmetic codes for distributed video coding. In *Proc. Intl. Conf. Image Processing*, 2007.

[Bal97] V. B. Balakirsky. Joint source-channel coding with variable length codes. In *Proc. Intl. Symp.. Inform. Theory*, 1997. p.419.

[Bat87] G. Battail. Pondération des symboles décodés par l'algorithme de Viterbi. *Ann. Telecommmun.*, 42 :31–38, Jan. 1987.

[BCI+97] C. Boyd, J. G. Cleary, S. A. Irvine, I. Rinsma-Melchert, and Ian H. Witten. Integrating error detection into arithmetic coding. *IEEE Trans. Inform. Theory*, 45(1), Jan. 1997.

[BCJR74] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv. Optimal decoding of linear codes for minimizing symbol error rate. *IEEE Trans. Inform. Theory*, 20 :284–287, Mar. 1974.

[BG96] C. Berrou and A. Glavieux. Near optimum error correcting coding and decoding : turbo codes. *IEEE Trans. Commun.*, 44 :1261–1271, Oct. 1996.

[BGP93] C. Berrou, A. Glavieux, and Thitimajshima P. Near Shannon limit error-correcting coding and decoding : Turbo codes. In *Proc. Intl. Conf. Commun.*, pages 1064–1070, 1993.

[BH00a] R. Bauer and J. Hagenauer. Iterative source-channel decoding using reversible variable length codes. In *Proc. Data Compression Conf.*, pages 93–102, Mar. 2000.

[BH00b] R. Bauer and J. Hagenauer. Symbol by symbol MAP decoding of variable length codes. In *Proc. 3rd ITG Conf. Source and Channel Coding, Munich, Germany, Jan. 17.-19. 2000*, 2000.

[BJWK06] S. Ben-Jamaa, C. Weidmann, and M. Kieffer. Asymptotic error-correcting performance of joint source-channel schemes based on arithmetic coding. In *Proc. MMSP*, pages 262–266, October 2006. Victoria, Canada.

[BK93] A. Bookstein and S.T. Klein. Is Huffman coding dead ? *Computing*, 50 :279–296, 1993.

[BKK01] M. Bystrom, S. Kaiser, and A. Kopansky. Soft source decoding with applications. *IEEE Trans. Circuits Syst. Video Technol.*, 11(10) :1108–1120, Oct. 2001.

[BM01] J. Bacsy and P. Mitran. Coding for the Slepian-Wolf problem with turbo-codes. In *Proc. of Global Communication Symposium*, pages 1400–1404, Nov 2001.

[Bri01] S. Ten Brink. Converegence behavior of iteratively decoded parallel concatenated codes. *IEEE Trans. Commun.*, 49(10) :1727–1737, Oct. 2001.

[CR00] J. Chou and K. Ramchandran. Arithmetic coding-based continuous error detection for efficient ARQ-based image transmission. *IEEE J. Select. Areas Commun.*, 18, June 2000.

[CT91] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. John Wiley, 1991.

[DHS01] C. Demiroglu, M.W. Hoffman, and K. Sayood. Joint source channel coding using arithmetic codes and trellis coded modulation. In *Proc. Data Compression Conf.*, pages 302–311, Mar. 2001.

[DHS06] B. Dongsheng, W. Hoffman, and K. Sayood. State machine interpretation of arithmetic codes for joint source and channel coding. In *Proc. Data Compression Conf.*, pages 143–152, 2006.

[Elm99] G.F Elmasry. Embedding channel coding in arithmetic coding. *IEE Proc.-Communications*, 146(2) :73–78, Apr. 1999.

[Fan63] R.M. Fano. A heuristic discussion of probabilistic decoding. *IEEE Trans. Inform. Theory*, pages 64–73, 1963.

[Gal62] R. G. Gallager. Low density parity check codes. *Trans. IRE Prof. Group on Inform. Theory*, 8 :2l–28, Jan. 1962. *http ://web.mit.edu/gallager/www/notes/pubs.pdf*.

[Gal63] R. G. Gallager. Low density parity check codes. *Research Monograph Series, MIT Press*, 1963.

[GCO05] M. Grangetto, P. Cosman, and G. Olmo. Joint source/channel coding and MAP decoding of arithmetic codes. *IEEE Trans. Commun.*, 53 :1007–1016, 2005.

[GFG01] A. Guyader, E. Fabre, and C. Guillemot. Joint source-channel turbo decoding of VLC encoded markov sources. In *Proceedings of GRETSI'01*, Sept. 2001.

[GFZ01a] J. Garcia-Frias and Y. Zhao. Compression of correlated binary sources using turbo-codes. *IEEE Commun. Lett.*, 5 :417–419, Oct. 2001.

[GFZ01b] J. Garcia-Frias and Y. Zhao. Data compression of unknown single and correlated binary sources using punctured turbo-codes. Oct. 2001.

[GG04] T. Guionnet and C. Guillemot. Soft and joint source-channel decoding of quasi-arithmetic codes. *EURASIP Journal on Applied Signal Processing*, 2004(3) :393–411, Mar. 2004.

[GMO04] M. Grangetto, E. Magli, and G. Olmo. Joint source-channel iterative decoding of arithmetic codes. In *Proc. Intl. Conf. Commun.*, volume 2, pages 886–890, June 2004.

[Gor94] M. J. Gormish. *Source Coding with Channel, Distortion and Complexity Constraints*. PhD thesis, Stanford, Mar. 1994.

[Gra90] R. M. Gray. *Entropy and Information Theory*. Springer-Verlag, 1990.

[HBS04] J. Hagenauer, J. Barros, and A. Schaefer. Lossless turbo source coding with decremental redundancy. In *Proc. Intl. Conf. Source and Channel Coding*, Erlangen, Germany, January 2004.

[Huf52] D. Huffman. A method for the construction of minimum redundancy codes. In *Proc. of the IRE*, volume 40, pages 1098–1101, 1952.

[HV92] P. G. Howard and J. S. Vitter. Practical implementations of arithmetic coding. *Image and Text compression*, pages 85–112, 1992.

[JAI97] A.K. Al Jabri and S. Al-Issa. Zero-error codes for correlated information sources. *Proc. of Cryptography*, pages 17–22, Dec. 1997.

[Jas08] X. Jaspar. *Joint source-channel turbo techniques and variable length codes*. PhD thesis, Ecole Polytechnique de Louvain, Apr.. 2008.

[Jel69] F. Jelinek. Fast sequential decoding algorithm using a stack. *IBM J. Res. Develop.*, 13 :675–685, 1969.

[JMG05] H. Jégou, S. Malinowski, and C. Guillemot. Trellis state aggregation for soft decoding of variable length codes. In *IEEE Workshop on Signal Processing Systems*, Athens, Greece, Nov. 2005.

[KN00] N. Kashyap and D. L. Neuhoff. Data synchronisation with timing. In *Proc. Intl. Symp.. Inform. Theory*, page 427, June 2000.

[KT02] J. Kliewer and R. Thobaben. Combining FEC and optimal soft-input source decoding for the reliable transmission of correlated variable-length encoded signals. In *Proc. Data Compression Conf.*, pages 83–89, April 2002.

[KT05] J. Kliewer and R. Thobaben. Iterative joint source-channel decoding of variable-length codes using residual source redundancy. *IEEE Trans. Wireless Commun.*, 4(3) :919–929, May 2005.

[Lan84] G.G Langdon. An introduction to arithmetic coding. *J. Research and Development*, 28(2) :135–149, 1984.

[LC97] L. Lin and R.S. Cheng. Improvements in SOVA-based decoding for turbo codes. In *Proc. ICC'97*, Montreal, Canada, June 1997.

[Lev66] V. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10 :707–710, 1966.

[LP01] C. Lamy and O. Pothier. Reduced complexity maximum a posteriori decoding of variable length codes. In *Globecom*, pages 1410–1413, Nov. 2001.

[LXG02] A.D. Liveris, Z. Xiong, and C.N. Georghiades. Compression of binary sources with side information at the decoder using LDPC codes. *IEEE Commun. Lett.*, 6 :440–442, Oct. 2002.

[Mac03] D. J.C. MacKay. *Information Theory, Inference and Learning algorithms*. Cambridge University Press, 2003.

[Mas56] S.J. Mason. Feedback theory : Further properties of signal flow graphs. *Proc. Inst. Radio Eng.*, 44 :920–926, July 1956.

[MF99] A.H. Murad and T.E. Fuja. Robust transmission of variable-length encoded sources. In *Proc. IEEE Wireless Communications and Networking Conf.*, pages 964–968, Sept. 1999.

[MF00] A. H. Murad and T. E. Fuja. Improving the performance of variable-length encoded systems through cooperation between source and channel decoders. In *Proc. Intl. Symp.. Inform. Theory*, page 264, June 2000.

[MJG07] S. Malinowski, H. Jégou, and C. Guillemot. Synchronization recovery and state model reduction for soft decoding of variable length codes. *IEEE Trans. Inform. Theory*, 53(1) :368–377, Jan. 2007.

[MJG08a] S. Malinowski, H. Jégou, and C. Guillemot. Computation of posterior marginals on aggregated state models for soft source decoding. *IEEE Trans. Commun.*, 2008. *accepted for publication*.

[MJG08b] S. Malinowski, H. Jégou, and C. Guillemot. Error recovery properties and soft decoding of quasi-arithmetic codes. *Eurasip journal on advanced signal processing*, 2008, Jan. 2008.

[MKKD05] G.R. Mohammad-Khani, M. Kieffer, and P. Duhamel. Simplification of VLC tables with application to ML and MAP decoding algorithms. *IEEE Trans. Commun.*, 2005. à paraître.

[ML87] M. E. Monaca and J. M. Lawler. Corrections and additions to "error recovery for variable length codes". *IEEE Trans. Inform. Theory*, 33(3) :454–456, May 1987.

[MR85] J.C. Maxted and J.P. Robinson. Error recovery for variables length codes. *IEEE Trans. Inform. Theory*, 31(6) :794–801, Nov. 1985.

[MSJ06] V. Mannoni, P. Siohan, and M. Jeanne. A simple on-line turbo estimation of source statistics for joint source channel VLC decoders : application to MPEG-4 video. In *Proc. International Symposium on Turbo-Codes and related topics*, April 2006.

[Pas76] R. C. Pasco. *Source Coding Algorithms for Fast Data Compression*. PhD thesis, Stanford University, CA, may 1976.

[PM00] M. Park and D. Miller. Joint source-channel decoding for variable-length encoded data by exact and approximate MAP sequence detection. *IEEE Trans. Commun.*, 48 :1–6, Jan. 2000.

[PR99] S.S. Pradhan and K. Ramchandran. Distributed source coding using syndromes(discus) : Design and construction. In *Proc. Data Compression Conf.*, pages 158–167, Mar. 1999.

[PR00] S.S. Pradhan and K. Ramchandran. Distributed source coding :symmetric rates and applications to sensor networks. In *Proc. Data Compression Conf.*, pages 363–372, Mar. 2000.

[Ris76] J.J. Rissanen. Generalized Kraft inequality and arithmetic coding. *J. Research and Development*, 20(3) :198–203, 1976.

[Ris79] J.J. Rissanen. Arithmetic coding as number representations. *Acta Polytech. Scand. Math.*, 31 :44–51, 1979.

[Say99a] J. Sayir. Arithmetic coding for noisy channels,. In *IEEE Information Theory Workshop*, June 1999. (Kruger Park, South Africa.

[Say99b] J. Sayir. *On coding by probability transformation*. PhD thesis, ETH Zurich, 1999.

[Say00] J. Sayir. Iterating the Arimoto-Blahut algorithm for faster convergence. In *Proc. Intl. Symp.. Inform. Theory*, page 235, June 2000.

[SB91] K. Sayood and J.C. Borkenhagen. Use of residual redundancy in the design of joint source/channel coders. *IEEE Trans. Commun.*, 39(6) :838–846, June 1991.

[SCW00] I. Sodagar, B.B. Chai, and J. Wus. A new error resilience technique for image compression using arithmetic codes. In *Proc. Intl. Conf. Acc. Speech Signal Processing*, June 2000.

[SD95] P. F. Swaszek and P. DiCicco. More on the error recovery for variable length codes. *IEEE Trans. Inform. Theory*, 41(6) :2064–2071, Nov. 1995.

[Sha48] C. E. Shannon. A mathematical theory of communication. *Bell System Tech. J*, 27 :379–423 623–656, 1948.

[Sim90] S.J. Simmons. Breadth-first treillis decoding with adaptative effort. *IEEE Trans. Commun.*, 38 :3–12, Jan. 1990.

[SRP04] D. Schonberg, K. Ramchandran, and S.S. Pradhan. Distributed code constructions for the entire Slepian-Wolf rate region for arbitrarily correlated sources. In *Proc. Data Compression Conf.*, pages 835–839, Mar. 2004.

[SV98] K.P. Subbalakshmi and J. Vaisey. Optimal decoding of entropy coded memoryless sources over noisy channels. In *Proc. Data Compression Conf.*, page 573, Mar. 1998.

[SW73] D. Slepian and J.K. Wolf. Noiseless coding of correlated information sources. *IEEE Trans. Inform. Theory*, 19 :471–480, July 1973.

[Szp00] W. Szpankowski. Asymptotic average redundancy of Huffman (and Shannon-Fano) block codes. In *Proc. Intl. Symp.. Inform. Theory*, page 370, June 2000.

[TGFZ03] T. Tian, J. Garcia-Frias, and W. Zhong. Compression of correlated sources using LDPC codes. In *Proc. Data Compression Conf.*, Mar. 2003.

[Tja00] T. Tjalkens. The complexity of minimum redundancy coding. In *Proc. Intl. Symp.. Inform. Theory*, page 373, June 2000.

[TW01] C.-W. Tsai and J.-L. Wu. On constructing the Huffman-code-based reversible variable-length codes. *IEEE Trans. Commun.*, 49(9) :1506–1509, Sept. 2001.

[TWM95] Y. Takishima, M. Wada, and H. Murakami. Reversible variable length codes. *IEEE Trans. Commun.*, 43(2/3/4) :158–162, Feb. 1995.

[VAG05] D. Varodayan, A. Aaron, and B. Girod. Rate-adaptative distributed source coding using LDPC codes. In *Proc. of Signals, Systems and Computers Conf.*, pages 1203–1207, Oct. 2005.

[Vit67] A. Viterbi. Error bounds for convolution codes and an asymptotically optimum decoding algorithm. *IEEE Trans. Inform. Theory*, 13(2) :260–269, Apr. 1967.

[VVS95] S. Vembu, S. Verdu, and Y. Steinberg. The source-channel separation theorem revisited. *IEEE Trans. Inform. Theory*, 41(1) :44–54, Jan. 1995.

[WNC87] I.H. Witten, R.M. Neal, and J.G. Cleary. Arithmetic coding for data compression. *Communications of the ACM*, 30(6) :520–540, July 1987.

[WNC98] I.H. Witten, R.M. Neal, and J.G. Cleary. Arithmetic coding revisited. *ACM Trans. on Information systems*, 16(3) :256–294, July 1998.

[WZ76] A.D. Winer and J. Ziv. The rate-distorsion function for source coding with side information at the decoder. *IEEE Trans. Inform. Theory*, 22 :1–10, Jan. 1976.

[YOB00] Y. Ying-On and T. Berger. On instantaneous codes for zero-error coding of two correlated sources. In *Proc. Intl. Symp.. Inform. Theory*, pages 344–, 2000.

[ZE01] Q. Zhao and M. Effros. Optimal code design for lossless and near lossless source coding in multiple access networks. In *Proc. Data Compression Conf.*, pages 263–272, Mar. 2001.

[Zig66] K. Zigangirov. Some sequential decoding procedures. *Probl. Predeach. Inform.*, 2 :13–15, 1966.

[ZZ02] G. Zhou and Z. Zhang. Synchronization recovery of variable length codes. *IEEE Trans. Inform. Theory*, 48(1) :219–227, Jan. 2002.

Table des figures

Résumé

L'étude menée dans cette thèse s'inscrit dans le contexte du codage conjoint source-canal. L'émergence de ce domaine depuis quelques années est dû aux limites, notamment au niveau applicatif, du théorème de séparation de Shannon. Ce théorème stipule que les opérations de codage de source et de codage de canal peuvent être réalisées séparément sans perte d'optimalité. Depuis quelques années, de nombreuses études visant à réaliser conjointement ces deux opérations ont vu le jour. Les codes de source, comme les codes à longueur variables ou les codes quasi-arithmétique ont beaucoup été étudié. Nous avons travaillé sur ces deux types de codes dans le contexte de codage conjoint source-canal dans ce document.

Un modèle d'état pour le décodage souple des codes à longueur variable et des codes quasi-arithmétique est proposé. Ce modèle est paramétré par un entier T qui permet de doser un compromis entre performance et complexité du décodage.

Les performances des codes sur ce modèle sont ensuite analysées en étudiant leurs propriétés de resynchronisation. Les méthodes d'étude de ces propriétés ont dû être adaptées aux codes QA et au canal à bruit additif blanc gaussien pour les besoins de cette analyse. Les performances des codes sur le modèle agrégé peuvent ainsi être prévues en fonction de la valeur du paramètre T et du code considéré.

Un schéma de décodage robuste avec information adjacente est ensuite proposé. La redondance ajoutée se présente sous la forme de contraintes partielles appliquées pendant le décodage. Cette redondance peut être ajoutée de manière très flexible et ne nécessite pas d'être introduite à l'intérieur du train binaire.

Enfin, deux schémas de codage de source distribué utilisant des codes quasi arithmétiques sont proposés. Le premier schéma réalise la compression en poinçonnant la sortie du code, tandis que le deuxième utilise un nouveau type de code : les codes quasi-arithmétiques avec recouvrement d'intervalle. Les résultats présentés par ces deux schémas sont compétitifs comparativement aux schémas classiques utilisant des codes de canal.