UNIVERSITÉ DE SHERBROOKE
Faculté de génie
Département de génie électrique et de génie informatique

ENCODAGE ENTROPIQUE DES INDICES BINAIRES D'UN QUANTIFICATEUR ALGÉBRIQUE ENCASTRÉ

Mémoire de maîtrise
Spécialité : génie électrique

Khaled LAKHDHAR

Jury : Roch LEFEBVRE (directeur)
Frédéric MAILHOT
Philippe GOURNAY

Sherbrooke (Québec) Canada Décembre 2009

Your file Votre référence
ISBN: 978-0-494-64338-9
Our file Notre référence
ISBN: 978-0-494-64338-9

RÉSUMÉ

Ce mémoire propose un algorithme de compression sans perte des indices binaires d'un quantificateur algébrique encastré utilisé par le codec AMR-WB+ pour encoder certaines des trames d'un signal audio. Une étude détaillée des statistiques a été menée dans le but de développer un algorithme efficace de compression et réduire par conséquent la longueur moyenne du code binaire utilisé par le codec AMR-WB+. En se basant sur cette étude des statistiques, deux techniques ont été combinées : l'encodage par plage et l'encodage par contexte qui se sont montrés très efficaces pour estimer les probabilités des différents indices.
En utilisant l'encodage arithmétique en version entière pour générer le code binaire, l'algorithme proposé permet de réduire sans perte jusqu'à 10% de la longueur du code utilisé par le AMR-WB+ tout en respectant la contrainte d'une application temps réel destinée à des terminaux GSM.

Mots-clés : codec AMR-WB+, quantification algébrique encastrée, TCX, compression sans perte, encodage entropique, encodage par plage, encodage arithmétique

REMERCIEMENTS

Je tiens à remercier en premier lieu mon directeur de recherche Roch Lefebvre pour m'avoir accueilli parmi le groupe de recherche GRPA (groupe de recherche sur la parole et l'audio). Je lui suis aussi reconnaissant pour ses encouragements et judicieux conseils. Je remercie aussi tout mes collègues de travail et spécialement Bruno, David et Éric pour toutes les discussions fructueuses et leurs aides. Mes remerciements s'adressent aussi à Danielle Poirier, secrétaire du groupe, pour m'avoir maintes fois aidé avec les procédures administratives, pour sa bonne humeur et sa sympathie.
Je ne peux finir sans remercier aussi la Mission Universitaire de Tunisie en Amérique du Nord pour sa présence tout au long de ma maîtrise et pour son support aussi bien financier que moral.

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

LISTE DES SYMBOLES

Cette liste énumère les symboles et variables utilisés tout au long de ce mémoire.

Symbole	Définition
kb/s	1000 bits par seconde
kHz	1000 Hertz
m_k	Mode de la k^{eme} trame
D_8	Réseau de points dans Z^8 dont la somme des coordonnées est paire
E_8	Réseau de Gosset
RE_8	Réseau de Gosset retourné
Q_n	Etage de quantification n
N_{bit}	Nombre de bits alloués pour chaque trame
n	Indice de l'étage de quantification
r	Ordre d'extension de Voronoï
n'	Indice de l'étage de quantification multiplexé avec l'ordre d'extension de Voronoï
I	Indice binaire d'un vecteur dans RE_8
k	Indice binaire de l'extension de Voronoï
I'	Indice binaire d'un vecteur dans RE_8 avec extension de Voronoï
RSB	Rapport signal à bruit
TCXs	Mode d'encodage où $s = 256, 512$ ou 1024
x	Signal audio à l'entrée du module TCX
x_w	Signal à la sortie du filtre d'accentuation
$\hat{x}_w$	Signal resynthétisé
$RE_8(m)$	Réseau de Gosset délimité par une sphère de rayon égal à m
$P_8(m)$	Nombre de leadeurs d'un réseau de Gosset délimité par une sphère de rayon égal à m
g^*	Gain optimal pour une trame donnée
$\tilde{x}$	Leadeur de x
$\tilde{y}^*$	Voisin le plus proche de y dans R
t	Rang d'une permutation
w^i	Nombre de permutations
Q_x	Quantificateur de numéro d'étage x
$p(x)$	Probabilité d'un événement x
$p(x\|y)$	Probabilité conditionnelle d'un événement x sachant y
$H(x)$	Quantité d'entropie présente dans un événement x
H_S	Entropie d'une source S
$C(S)$	Code binaire utilisé pour encoder une source S
R_c	Redondance d'un code C
Nsv	Nombre de sous-vecteurs

E_1	Redimensionnement de la moitié inférieure de l'intervalle unité
E_2	Redimensionnement de la moitié supérieure de l'intervalle unité
E_3	Redimensionnement du quart de l'intervalle unité
E_i^k	Redimensionnement E_i appliqué k fois
$E_i o E_j$	Application du redimensionnement E_j ensuite le redimensionnement E_i avec $i, j \in \{1, 2, 3\}^2$
N_{occu}	Nombre d'occurrences successives
T_{plage}	Taille de la table utilisée pour l'encodage par plage
m	Nombre de bits utilisés par l'encodage arithmétique.
Somme_Totale	Somme d'une table des probabilités entières
Somme_Cum	Somme cumulative
i	Borne inférieure de l'intervalle utilisé pour générer l'étiquette
s	Borne supérieure de l'intervalle utilisé pour générer l'étiquette
s_3	Nombre de redimensionnements successifs E_1 ou E_2
p	Table des probabilités
τ	Taux de compression

LISTE DES ACRONYMES

Terme technique	Signification
Codec	Codeur Décodeur
TCX	Transform Coded Excitation *Excitation codée par transformée*
ACELP	Algebraic Code Excited Linear Prediction *Prédiction linéaire excitée par codage algébrique*
AGC	Adaptive Gain Control *Contrôle du gain adaptatif*
AMR-WB	Adaptive Multi-Rate Wideband *Codec adaptatif multi-débit large bande*
AMR-WB+	Extended Adaptive Multi-Rate Wideband *Codec adaptatif multi-débit large bande étendu*
CELP	Code Excited Linear Prediction *Prédiction linéaire excitée par codage*
ISF	Immittance Spectral Frequency *Fréquence d'immittance spectrale*
ISP	Immittance Spectral Pair *Paires d'immittance spectrale*
LP	Linear Prediction *Prédiction linéaire*
LPC	Linear Predictive Coding *Codage par prédiction linéaire*
LTP	Long Term Predictor (or Long Term Prediction) *Prédiction à long terme*
S-MSVQ	Split-MultiStage Vector Quantization *Quantification vectorielle multi-étage*
BWE	Band Width Extension *Extension de largeur de bande*
SNR	Signal to Noise Ratio *Rapport signal à bruit*
VQ	Vector Quantization *Quantification Vectorielle*
wMOPS	Weighted Millions of Operations Per Second *Millions d'opérations pondérées par seconde*
RAM	Random Access Memory *Mémoire vive*
ROM	Read Only Memory *Mémoire morte*
GSM	Global System for Mobile Communications *Système Global pour les communications mobiles*

CHAPITRE 1

Introduction

Le souci des applications de télécommunication, et plus particulièrement des systèmes de codage des signaux audios, est d'offrir à l'utilisateur la meilleure satisfaction et le meilleur confort possible, et ce, en restituant un message audio qui soit le plus fidèle au signal naturel.

Particulièrement, un signal de parole est très riche en informations et émotions qu'on cherche à préserver lors du processus de codage. Si on analyse un signal de parole en terme d'informations présentes dans celui-ci, on peut dégager deux aspects, qui réunis permettent de définir correctement le message contenu dans un signal de parole. Le premier aspect est lié aux mots, à savoir leur sens et leur intelligibilité. Le second aspect peut être interprété comme les émotions dégagées par le signal de parole, telles que les intonations et les tonalités.

Un codec (**Co**deur/ **Déc**odeur) efficace est celui qui permet de préserver aussi bien l'intelligibilité du message que les émotions et tonalités, mais comme les ressources des systèmes de traitement sont limitées à savoir la vitesse des processeurs, tailles des mémoires et bandes passantes des systèmes de communications, la priorité est toujours donnée à l'intelligibilité du signal de parole.

Généralement un codec est composé de deux blocs : un codeur dont la tâche se résume à analyser le signal audio, en extraire un nombre d'informations fréquentielles ou temporelles et les représenter sous forme de symboles binaires. À l'autre bout de la chaine de codage se trouve le décodeur, qui à partir des symboles binaires issus du codeur fait pratiquement le processus inverse d'un codeur pour synthétiser le signal de parole en restituant les informations fréquentielles ou temporelles à partir des symboles binaires. Définissant le débit comme étant la quantité de bits utilisée pour représenter ces informations fréquentielles/temporelles par unité de temps, il est évident que plus cette quantité est grande, plus la qualité de synthèse du signal sera meilleure pour un codec efficace. Mais comme on est limité en ressources, généralement on dispose d'un débit limité qu'on ne peut dépasser, et par conséquent l'utilisation du débit disponible doit être optimale pour pouvoir en profiter pleinement.

Sur la plupart des codeurs, le passage d'une information fréquentielle/temporelle à un symbole binaire se fait par recherche dans un dictionnaire. Ce dictionnaire (présentement dans le cadre de ce travail c'est la $RE8$), dont la taille est limitée, est construit de sorte

qu'il contient les éléments qui représentent de la meilleure façon les statistiques d'un signal audio. Il est évident que plus la taille du dictionnaire est grande plus il est fidèle aux statistiques de la source (signal audio) et donc sera plus fidèle aux signaux d'origine, mais en contrepartie, on aura à utiliser plus de débit pour représenter tous les éléments de ce dictionnaire. L'utilisation d'un dictionnaire de taille élevée présente de sérieuses limitations pratiques, vu que le codeur et le décodeur doivent les deux en posséder une copie. Même si le problème du stockage du dictionnaire a été surmonté avec les travaux de J.P.Adoul (Quantification algébrique encastrée), le problème du débit par contre reste non résolu et les mots binaires sont transmis sans compression permettant de réduire le débit. Ce travail est consacré à la réduction de la quantité de bits nécessaire pour représenter les éléments du dictionnaire sans pour autant introduire de dégradation sur la qualité du signal restitué et ce en utilisant un codage entropique sans perte des indices binaires transmis entre le codeur et décodeur. La quantité de bits gagnée par ce codage entropique sera utilisée pour envoyer plus d'informations au décodeur permettant ainsi d'améliorer la qualité.

Le premier chapitre aura pour but de présenter le codec concerné par ce projet à savoir le AMR-WB+ et détailler les différents blocs utilisés pour encoder un signal audio. La quantification algébrique encastrée utilisée par le module TCX sera détaillée dans le but d'expliquer son fonctionnement ainsi que le processus de génération des indices binaires.

Le second chapitre constitue un survol des techniques de compression sans pertes les plus courantes. La première partie de ce chapitre est consacrée aux problèmes d'estimation des probabilités et le lien qui les relient à la compression alors que la deuxième partie a été dédiée à la présentation des algorithmes de construction de code binaire.

Le troisième chapitre présente une étude détaillée des statistiques des indices binaires de la quantification algébrique encastrée. Cette étude à été menée dans le but de vérifier l'optimalité du processus utilisé par le codec AMR-WB+. La deuxième partie, présente l'algorithme proposé dont la tâche consiste à chercher les redondances présentes dans ces indices et de les exploiter pour compresser ces derniers sans perte. Une version adaptative à été développée pour tenir en compte des variabilités des statistiques selon le type de signal audio traité et selon le débit d'encodage.

Le chapitre quatre quant à lui présente les performances de la solution proposée. Trois critères ont été mesurés à savoir l'occupation mémoire, le temps de calcul (mesuré en millions d'opérations par seconde) et les taux de compression réalisée. Les résultats obtenus affirment que les objectifs de ce projet ont été atteints puisque la solution présentée permet de réduire jusqu'à 10% du débit pour la même qualité de codage et est techniquement faisable et implémentable sur un terminal à ressources limitées.

CHAPITRE 2

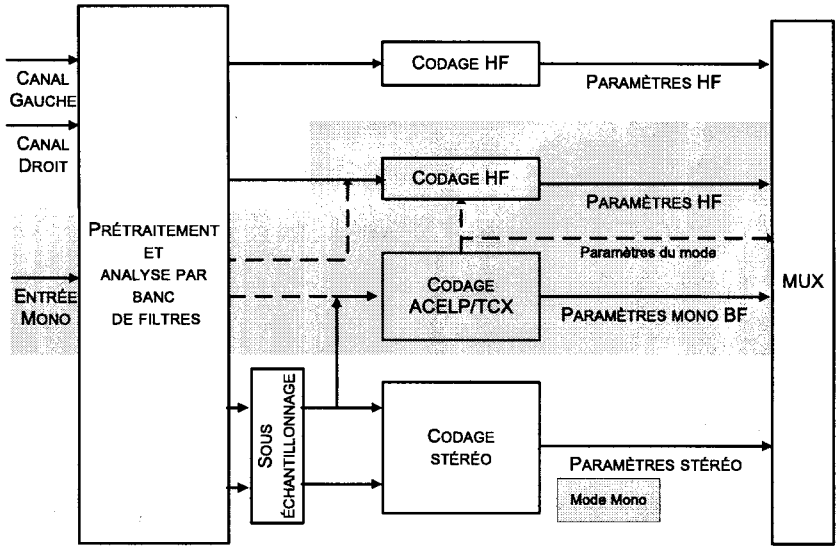
Le Codec AMR-WB+

Dans ce chapitre, on va présenter le codec concerné par ce travail, à savoir le codec AMR-WB+. C'est un codec qui permet de compresser avec perte les signaux audios, et ce, dans le but de les stocker sur un support informatique ou bien pour les envoyer sur une chaine de communication numérique. C'est une extension de son prédécesseur à savoir l'AMR-WB. L'extension ajoutée inclut la possibilité de coder des signaux stéréo, et ce, avec des fréquences d'échantillonnage plus élevées. L'extension inclut aussi le codage par extension de largeur de bande (*Band Width Extention*, BWE).
Un autre ajout majeur consiste dans l'intégration d'un module qui utilise l'excitation codée par transformée (*Transform Coded Excitation*, TCX), additionnellement à la prédiction linéaire excitée par code algébrique (*Algebraic Code Excited Linear Prediction*, ACELP). La commutation automatique entre ces deux modes permet d'exploiter les différentes propriétés des signaux audios, ce qui permet d'avoir de meilleures qualités d'encodage.
Dans le but de compresser le flux binaire généré par ce codec, il est essentiel de comprendre et d'analyser les modules intervenant dans la génération des indices binaires pour le mode TCX. Les sections suivantes présentent ce codec et détaillent en particulier le module TCX ainsi que la procédure de génération des mots binaires utilisée par ce dernier.

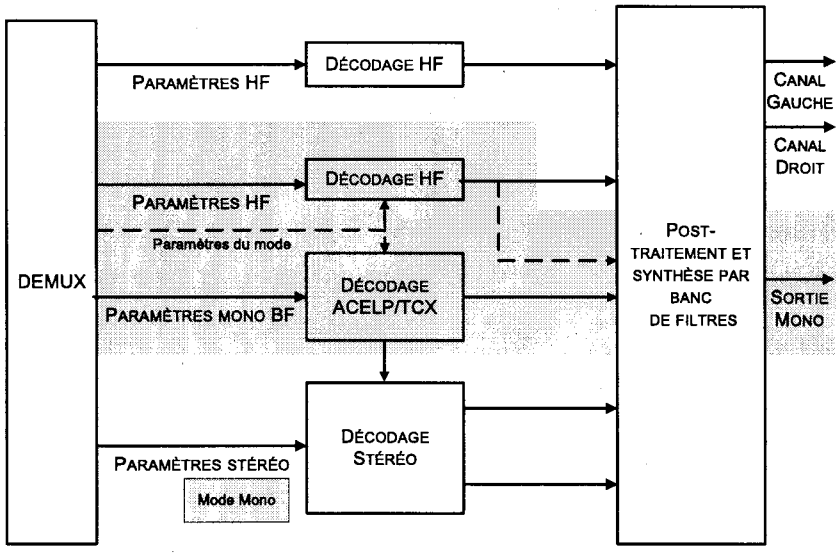
2.1 Fonctionnement du codec AMR-WB+

Le codec AMR-WB+ est un codec à débit variable opérant avec des débits allant de 6kb/s à 36kb/s pour les signaux mono et entre 7kb/s et 48kb/s pour les signaux stéréo. Il traite les signaux d'entrée sous forme de trames de 2048 échantillons échantillonnés avec une fréquence interne allant de 12.8kHz à 38.4 kHz. Ces 2048 échantillons sont subdivisés en deux bandes égales à savoir hautes (**HF**) et basses fréquences (**BF**). Les signaux BF sont décomposés en quatre paquets (sous-trame) de même taille de 256 échantillons chaque , avant d'être analysés par le module ACELP/TCX.
Le signal HF est codé en utilisant la technique d'extension de largeur de bande, alors que le signal BF est codé en se basant sur une commutation entre l'ACELP [Guyader *et al.*, 1995; Salami *et al.*, 1994] et la TCX. Le choix entre ces deux modes est basé sur une analyse en boucle ouverte ou fermée dans le but de déterminer quel module permettant d'avoir une meilleure qualité de codage est à utiliser.

Quand le signal d'entrée est stéréo et quand le mode de codage sélectionné est mono, les deux canaux droit et gauche sont combinés en un seul signal mono pour le mode ACELP/TCX, alors que l'encodage stéréo reçoit les deux canaux séparés. La figure 2.1 représente la structure du codec AMR-WB+ [3GPP, 2007-03]. Le codec ACELP, similaire



(a) Schéma-bloc du codeur AMR-WB+



(b) Schéma-bloc du décodeur AMR-WB+

Figure 2.1 Schéma-bloc du codec AMR-WB+ [3GPP, 2007-03] p.12 (a)Codeur, (b) Décodeur

à celui existant sur l'AMR-WB, utilise une analyse par prédicteur à long terme (*Long Term Predictor*, LTP) et l'excitation du dictionnaire algébrique. Pour le module ACELP seules des trames de 256 échantillons sont permises. Pour le codec TCX, la transformée

de Fourier est calculée pour 256, 512 ou 1024 échantillons passés à travers une fenêtre de pondération. Les coefficients ainsi obtenus sont quantifiés par la suite en utilisant la quantification algébrique encastrée. Pour permettre au décodeur de reconstruire le signal orignal, certains paramètres lui sont transmis de la part du codeur. Indépendamment de la longueur de la fenêtre d'analyse utilisée, ces paramètres sont multiplexés sous forme de trame binaire dont le format, pour le mode TCX, est donné par la figure 2.2 [3GPP, 2005-07]. Les tableaux 2.1, 2.2 et 2.3 donnent la largeur de chaque bloc de la trame en fonction

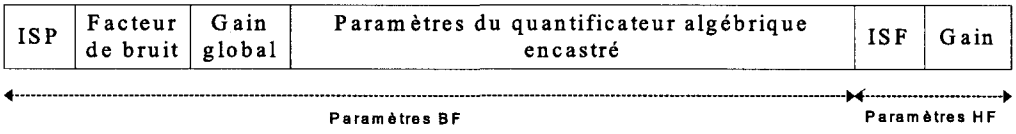


Figure 2.2 Format de la trame binaire pour le module TCX

du débit [3GPP, 2007-03]. Ces tableaux mettent en évidence la grande occupation de

Tableau 2.1 Allocation du débit pour une trame de 256 échantillons

Paramètre	**Nombre de bits**							
Bits de mode	2							
Paramètres ISP	46							
Facteur de bruit	3							
Gain global	7							
Correction du gain	ϕ							
Indices de quantification algébrique	134	166	198	230	262	310	342	408
Paramètres HF ISF	9							
Gain pour les HFs	7							
Correction du gain	ϕ							
Débit (Bit/Trame)	208	240	272	304	336	384	416	480

Tableau 2.2 Allocation du débit pour une trame de 512 échantillons

Paramètre	**Nombre de bits**							
Bits de mode	2+2							
Paramètres ISP	46							
Facteur de bruit	3							
Gain global	7							
Correction du gain	6							
Indices de quantification algébrique	318	382	446	510	574	670	734	862
Paramètres HF ISF	9							
Gain pour les HFs	7							
Correction du gain	8×2							
Débit (Bit/Trame)	416	480	544	608	672	768	832	960

la trame binaire par les indices de la quantification algébrique. En effet, le pourcentage occupé par ces indices est de 64% à bas débits contre 92% à hauts débits.

Tableau 2.3 Allocation du débit pour une trame de 1024 échantillons

Paramètre	**Nombre de bits**							
Bits de mode	2+2+2+2							
Paramètres ISP	46							
Facteur de bruit	3							
Gain global	7							
Correction du gain	3+3+3							
Indices de quantification algébrique	695	823	951	1079	1207	1399	1527	1783
Paramètres HF ISF	9							
Gain pour les HFs	7							
Correction du gain	16 × 3							
Débit (Bit/Trame)	832	960	1088	1216	1344	1536	1664	1920

2.2 Le module hybride TCX/ACELP

L'algorithme d'encodage au cœur du codec AMR-WB+ est basé sur un module hybride TCX/ACELP. Pour chaque bloc du signal en entrée, le codeur décide (soit en boucle ouverte ou fermée) lequel des deux modules (ACELP ou TCX) est plus approprié. Le module ACELP, étant un encodage dans le domaine temporel et utilisant la prédiction linéaire, est plus approprié pour les signaux de parole. Le module TCX, utilisant une transformation dans le domaine des fréquences (DFT), est plus approprié pour les signaux de musique.

Comme vu précédemment, en mode ACELP seules des sous-trames de 256 échantillons sont possibles alors qu'en mode TCX (développé plus loin dans le document), on peut avoir différentes longueurs de sous-trames i.e 256, 512 et 1024. Donc, pour une trame de 1024 échantillons, il est alors possible d'utiliser différents modes d'encodage à savoir : ACELP, TCX256, TCX512 et TCX1024.

Sur le codec AMR-WB+ 26 modes de combinaisons sont possibles. Désignons par (m_0, m_1, m_2, m_3) les combinaisons possibles où m_k désigne le mode que prend la k^{eme} sous-trame avec :

- $m_k = 0$: Le mode sélectionné est ACELP
- $m_k = 1$: Le mode sélectionné est TCX256
- $m_k = 2$: Le mode sélectionné est TCX512
- $m_k = 3$: Le mode sélectionné est TCX1024

Les modes ainsi définis sont représentés sur le tableau 2.4.

Par exemple, le mode $(2, 2, 1, 1)$ indique que les deux premières sous-trames de 256 échantillons chacun, doivent être combinées en une seule sous-trame de 512 échantillons pour la TCX512 alors que les deux dernières sous-trames de 256 sont encodées séparément en

Tableau 2.4 Modes de combinaisons possibles entre ACELP et TCX

(0,0,0,0)	(0,0,0,1)	(2,2,0,0)	
(1,0,0,0)	(1,0,0,1)	(2,2,1,0)	
(1,1,0,0)	(1,1,0,1)	(2,2,1,1)	
(0,0,1,0)	(0,0,1,1)	(0,0,2,2)	
(1,0,1,0)	(1,0,1,1)	(1,0,2,2)	
(0,1,1,0)	(0,1,1,1)	(0,1,2,2)	(2,2,2,2)
(1,1,1,0)	(1,1,1,1)	(1,1,2,2)	(3,3,3,3)

mode TCX256.

La meilleure des 26 combinaisons du tableau 2.4 est déterminée suite à une analyse en boucle ouverte ou fermée. Le critère de décision est le rapport signal à bruit (*Signal-to-Noise Ratio*, SNR) donné par :

$$\overline{SNR} = \frac{1}{N_{ST}} \sum_{0}^{N_{ST}-1} 20 \log_{10} \left(\frac{\sum_{0}^{N-1} x_w^2(n)}{\sum_{0}^{N-1} \left(x_w(n) - \hat{x}_w(n) \right)^2} \right) \tag{2.1}$$

avec x_w le signal audio pondéré en entrée, $\hat{x}_w$ le signal audio resynthétisé soit avec le module TCX ou ACELP, N_{ST} le nombre de sous-trames et N la longueur de la sous-trame analysée. Un SNR plus élevé implique une meilleure qualité de codage.

2.2.1 Sélection en boucle fermée

Le tableau 2.5 représente les essais, effectués en boucle fermée pour choisir entre les différentes possibilités d'encodage.

ST_i correspond à la i^{eme} sous-trame de longueur 256 échantillons. La première sous-trame ST_0 est encodée en mode ACELP ensuite en mode TCX256 au cours des deux premières itérations. Selon le SNR calculé par l'équation (2.1), le meilleur de ces deux modes est choisi. Les itérations 3 et 4 consistent à choisir entre ACELP et TCX256 pour la deuxième sous trame ST_1. Ayant ainsi déterminé la meilleure des 4 combinaisons possibles pour les deux premières sous-trames i.e (ACELP,ACELP), (ACELP,TCX256), (TCX256, ACELP) et (TCX256,TCX25), le mode TCX512 sera testé pour ST_0 et ST_1 considérées comme une seule sous-trame de 512 échantillons. Le choix issu de la 5^{eme} itération est alors soit un des 4 modes précédemment présentés ou bien le mode $TCX512$. Les itérations 6 à 10 refont les mêmes tests que ceux appliqués pour ST_0 et ST_1 mais cette fois-ci pour ST_2 et ST_3. La dernière itération (11) permet de choisir entre les modes choisis au cours des itérations

Tableau 2.5 Sélection en boucle fermée des modes ACELP/TCX

Itération	ST_0	ST_1	ST_2	ST_3
1	ACELP			
2	TCX256			
3		ACELP		
4		TCX256		
5	TCX512	TCX512		
6			ACELP	
7			TCX256	
8				ACELP
9				TCX256
10			TCX512	TCX512
11	TCX1024	TCX1024	TCX1024	TCX1024

5 et 10 et le mode $TCX1024$. A l'issue de ces itérations, le module hybride ACELP/TCX permet de choisir la meilleure configuration permettant d'avoir le meilleur SNR.

2.2.2 Sélection en boucle ouverte

Une méthode alternative pour la sélection des modes ACELP/TCX consiste à utiliser des itérations en boucle ouverte. Sachant qu'un signal de parole est mieux codé en ACELP qu'en TCX et qu'un signal de musique est mieux codé en utilisant le TCX que l'ACELP, la sélection en boucle ouverte exploite les propriétés temporelles et spectrales des deux types de signaux. Cette méthode en boucle ouverte est subdivisée en trois grandes étapes :

- Classification de l'excitation : Au cours de cette étape des paramètres décrivant l'énergie de chaque sous-trame de 256 échantillons sont calculés (essentiellement la déviation standard des énergies des hautes et basses fréquences). Un seuillage est ensuite appliqué en se basant sur ces paramètres pour choisir soit un des modes TCX notés TCXs ou l'ACELP. Il se peut qu'au cours de cette étape, on ne puisse trancher entre les deux modules alors un mode 'incertain' est sélectionné.

- Amélioration de la classification de l'excitation : Quand le mode sélectionné est 'incertain' ou ACELP, de nouveaux paramètres temporels sont calculés. Basée sur ces nouveaux paramètres ainsi que les paramètres fréquentiels issus de l'étape précédente, le mode ACELP est alors revérifié et une décision binaire est alors prise soit TCXs ou ACELP.

- Sélection des modes TCX : Cette étape n'est déclenchée que si le mode choisi est TCXs. Quand c'est le cas, une analyse identique à celle utilisée en boucle fermée est appliquée pour choisir un des trois modes possibles de la TCX.

2.3 Le module TCX et la quantification algébrique

Le signal à l'entrée du module ACELP/TCX est filtré par un filtre passe-haut, ensuite un filtre d'accentuation (*emphasis-filter*) est utilisé avec la fonction de transfert suivante :

$$P(z) = 1 - 0.68z^{-1} \tag{2.2}$$

Ce filtre a l'avantage de réduire l'énergie du signal audio en basses fréquences et de l'augmenter en hautes fréquences. Le résultat est alors un signal ayant une dynamique spectrale moins élevée, ce qui permet d'améliorer les performances du codage indépendamment du mode choisi. La pré-accentuation permet aussi d'effectuer une analyse LPC plus efficace.

2.3.1 Le module TCX

Cette section présente en détail le codeur TCX qui est un mode d'encodage possible du signal mono basses fréquences. La figure 2.3 présente le diagramme fonctionnel du mode TCX. Le principe utilisé par le mode TCX est le même pour les sous-trames de longueur 256, 512 et 1024 échantillons avec une différence mineure concernant le fenêtrage utilisé. Le signal audio x est filtré par un filtre perceptuel donné par (2.3) et dont les coefficients sont variables au cours du temps pour obtenir un signal pondéré x_w. Si la sous-trame passée est codée en utilisant le module ACELP, la réponse impulsionnelle du filtre perceptuel au repos est soustraite au signal x_w.

$$W(z) = \frac{\hat{A}(z/0.92)}{1 - 0.68z^{-1}} \tag{2.3}$$

Le signal ainsi obtenu est ensuite fenêtré en utilisant une fenêtre d'anticipation (lookahead) et transformé dans le domaine des fréquences en utilisant la transformée de Fourrier discrète (*Discrete Fourrier Transform*, DFT) donnée par :

$$X(k) = \frac{1}{L}\sum_{0}^{L-1} x_w(n) e^{-j\frac{2\pi}{L}nk} \tag{2.4}$$

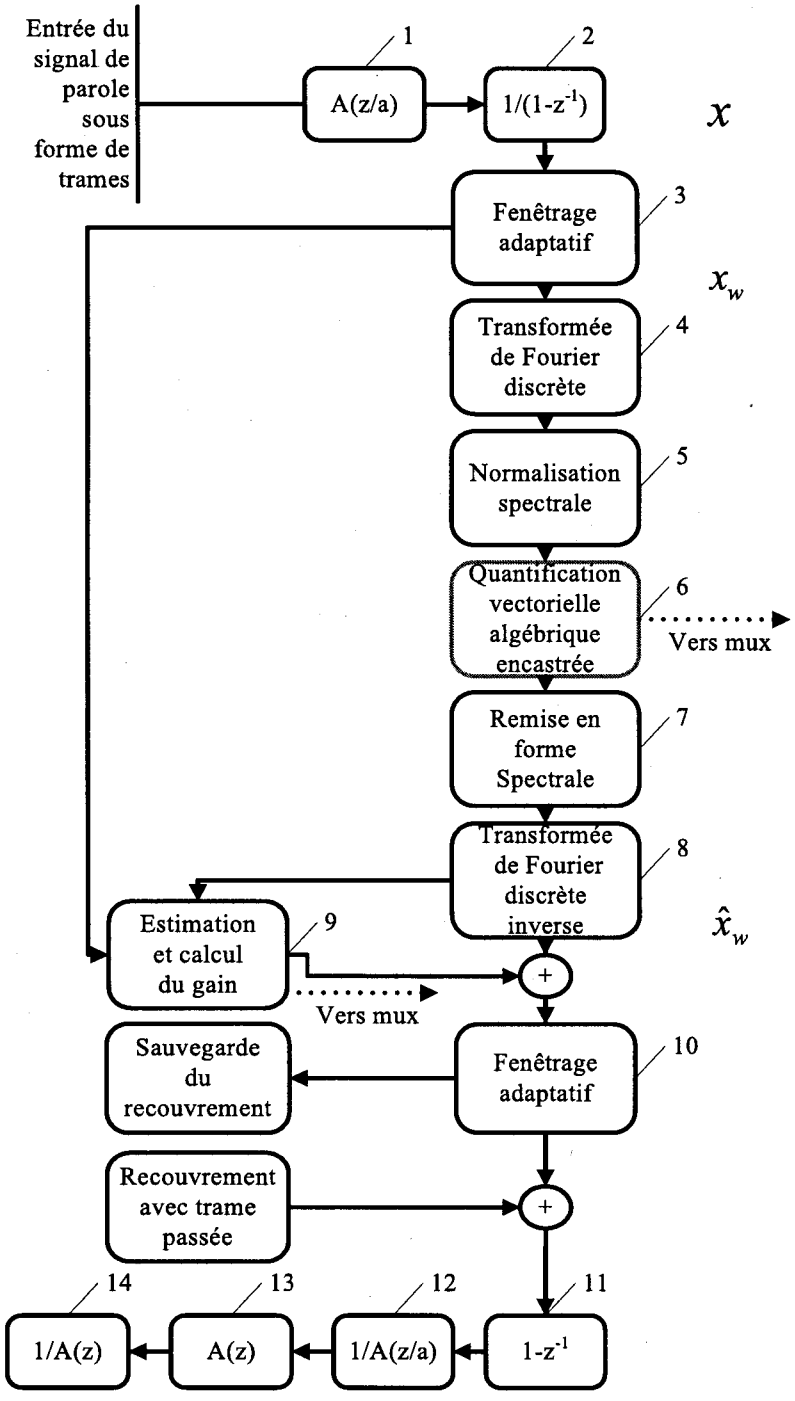


Figure 2.3 Diagramme fonctionnel du module TCX
Source : [3GPP, 2007-03] p.29

Où L représente le nombre de points utilisés selon la longueur de la sous trame c'est à dire $256 + 32$ échantillons d'anticipation pour la TCX256, $512 + 64$ échantillons d'anticipation pour la TCX512 et $1024 + 128$ échantillons d'anticipation dans le cas de la TCX1024.

Dans le domaine de la transformée, le signal $X(k)$ est normalisé pour minimiser les artefacts en basses fréquences. Cette normalisation permet aussi de minimiser les bruits de quantification en basses fréquences. La quantification algébrique encastrée (section 2.3.2) est ensuite appliquée aux coefficients spectraux normalisés. Après quantification, les coefficients spectraux quantifiés $\hat{X}(k)$ sont dénormalisés et la transformée de Fourier discrète inverse (*Inverse Discrete Fourier Transform*, IDFT) est appliquée pour obtenir un signal quantifié dans le domaine temporel $\hat{x}_w$. Le gain pour cette sous-trame est alors optimisé pour avoir une meilleure corrélation entre le signal temporel x_w et le signal quantifié $\hat{x}_w$. La valeur optimale g^* pour ce gain est donnée par (2.5) :

$$g^* = \frac{\sum_{0}^{L-1} x_w(n)\hat{x}_w(n)}{\sum_{0}^{L-1} \hat{x}_w(n)\hat{x}_w(n)} \tag{2.5}$$

Après l'optimisation du gain, le signal quantifié $\hat{x}_w$ est passé à travers une fenêtre dont la forme dépend de la longueur de la sous-trame passée. Un recouvrement est appliqué avec la trame passée, et les coefficients du numérateur $\hat{A}(z)$ du filtre perceptuel $W(z)$ sont mis à jour. Le coefficient de Fourrier à la fréquence de Nyquist est mis à zéro.
Si L représente le nombre de points utilisés pour le calcul de la TFD, les $N = L/2$ valeurs complexes des coefficients sont groupées dans des blocs formés par quatre coefficients complexes chacun, ce qui donne des blocs de dimension huit formés par des coefficients réels. La taille huit de ces blocs a été choisie pour coïncider avec la dimension du quantificateur algébrique en dimension huit utilisé pour quantifier ces blocs. La section suivante présente ce quantificateur particulier et détaille les étapes qui permettent de quantifier n'importe quel vecteur de $\mathbb{R}^8$.

2.3.2 La quantification algébrique

Quand le nombre des valeurs distinctes générées par la source est infini, il est impossible de les représenter sur un système informatique à mémoire limitée. La compression avec perte et plus précisément la quantification permet de résoudre ce problème, et ce, en tolérant des pertes d'information contrôlées. Dans plusieurs applications la perte de l'exactitude de l'information est tolérée. Par exemple sur le codec AMR-WB+, à haut débit, les valeurs des échantillons audios peuvent ne pas être exactes sans pour autant dégrader la qualité du signal. La quantification consiste alors à représenter un ensemble fini ou infini de valeurs de la source par un seul indice dans un dictionnaire représentant un élément qui soit le

plus proche de cet ensemble. La reconstruction par contre consiste à lire à partir de l'indice reçu l'élément du dictionnaire approprié. La quantification utilisée dans le module TCX est la quantification vectorielle qui consiste à coder les symboles successifs de la source sous forme de bloc. La figure 2.4 représente le principe de la quantification vectorielle. A

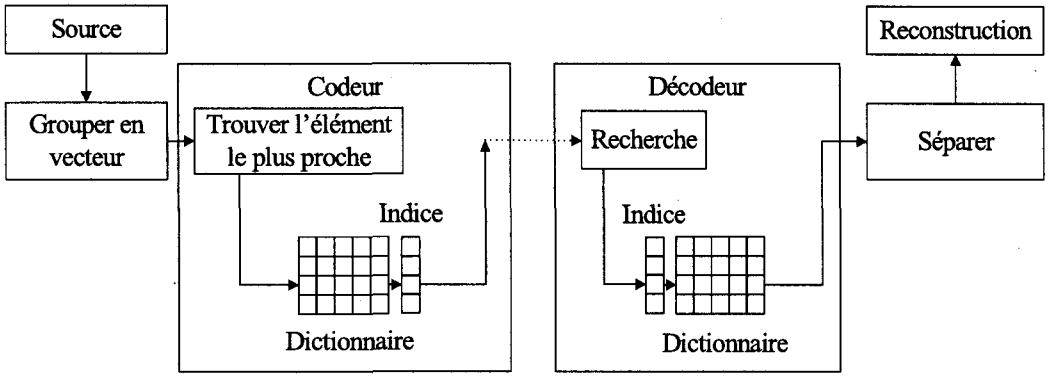


Figure 2.4 Principe de la quantification vectorielle

l'entrée du bloc 6 de la figure 2.3, et dans le but de quantifier les coefficients spectraux de la DFT, une méthode basée sur la quantification vectorielle est utilisée. Cette méthode est appelée « quantification vectorielle sphérique » à cause de la forme géométrique que prennent les vecteurs du dictionnaire utilisé. Le dictionnaire utilisé par cette quantification est le réseau de Gosset RE_8. La section suivante présente ce dictionnaire particulier ainsi que les méthodes de codage/décodage mises en oeuvre.

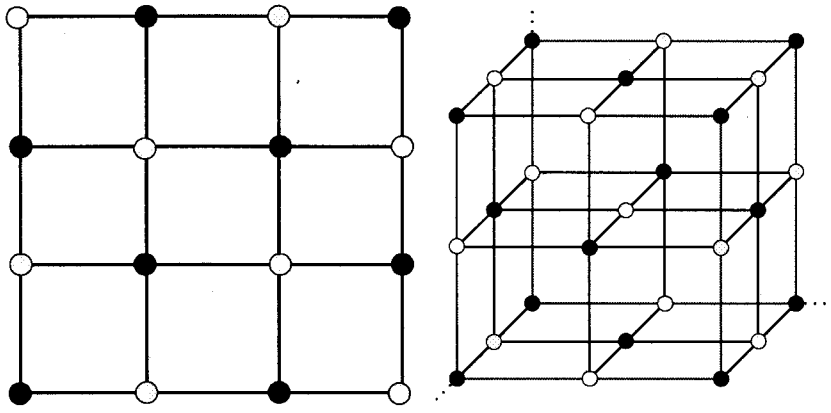
2.3.3 Le réseau de Gosset RE_8

Le réseau D_n contient les points de $\mathbb{Z}^n$ dont la somme des coordonnées est paire. En dimension 2 ce réseau correspond à un damier 2.5(a), alors qu'en dimension 3 les points de D_3 forment un cube à faces centrées 2.5(b). Le réseau de Gosset est défini comme l'union de deux réseaux obtenus par translation du réseau $2D_8$ par les deux mots à code de répétition à savoir $[0,0,0,0,0,0,0,0]$ et $[1,1,1,1,1,1,1,1]$ [Lamblin et Adoul, 1988], soit :

$$RE_8 = (2D_8) \cup (2D_8 + [1,1,1,1,1,1,1,1]) \tag{2.6}$$

D_8 étant le réseau de points à coordonnées entières et dont la somme est paire. Donc tout point x appartenant à RE_8 possède les propriétés suivantes :

1. $x \in \mathbb{Z}^8$ (x à composantes entières)
2. La somme des coordonnées de x est multiple de 4.



(a) Réseau D_2 (b) Réseau D_3

Figure 2.5 Réseau D_n en dimensions 2 et 3

3. Les huit composantes ont la même parité. i.e. toutes paires ou impaires.

Les points de RE_8 sont répartis sur des sphères centrées à l'origine et de rayon $2\sqrt{2m}$ avec m un entier naturel. Donc un point de $RE_8(m)$, à l'addition des trois propriétés précédemment citées, jouit de la propriété suivante :

$$x \in S(0, 2\sqrt{2m}) \Rightarrow \sum_{i=1}^{8} x_i^2 = 8m \tag{2.7}$$

Une propriété très intéressante du réseau de Gosset est que toute permutation des coordonnées d'un point de ce réseau donne un point qui y appartient. Cette propriété permet de définir la relation d'équivalence suivante :

$$x \sim y \Leftrightarrow \text{ S'il } \exists \text{ une permutation des coordonnés de } x \text{ qui donne } y$$

Par exemple $(1, 0, -1, 0, 0, 0, 0, 0) \sim (0, 0, -1, 0, 0, 1, 0, 0)$. Cette relation d'équivalence réalise une partition de $RE_8(m)$ en $P_8(m)$ classes d'équivalence. Dans $RE_8(m)$, on définit une relation d'ordre $\geq$ définie par :

$$x \geq y \begin{cases} \exists j \in [1, \ldots, 8] \ / \forall j,\ x_i \geq y_i \text{ et } x_j \geq y_j \\ x = y \end{cases}$$

où x_i (respectivement y_i) sont les composantes du vecteurs x (respectivement y). "$\geq$" est une relation d'ordre total. En effet chaque classe peut être définie par son élément maximal ou son *leadeur*. Soit $\tilde{x}$ le leadeur de sa classe à q composantes distinctes $(a_0, a_1, \ldots, a_{q-1})$. Si

w^i est le nombre de répétitions de la composante a_i, alors le nombre d'éléments de chaque classe est donné par la formule dénombrant les permutations avec répétition donnée par [Lamblin et Adoul, 1988] :

$$K = \frac{8!}{w^0!w^1!\dots w^{q-1}!} \tag{2.8}$$

Le tableau 2.6 présente la liste des leadeurs des 8 classes d'équivalence de la sphère $m = 1$ dont le nombre d'éléments est $K(j)$, $H(j)$ étant le cumul des classes précédentes. La

Tableau 2.6 Liste des leadeurs des 8 classes d'équivalence de la sphère $m = 1$

j	$K(j)$	$H(j)$	Leadeurs ($m = 1$)							
1	28	0	2	2	0	0	0	0	0	0
2	56	28	2	0	0	0	0	0	0	-2
3	1	84	1	1	1	1	1	1	1	1
4	28	85	1	1	1	1	1	1	-1	-1
5	70	113	1	1	1	1	-1	-1	-1	-1
6	28	183	1	1	-1	-1	-1	-1	-1	-1
7	28	211	0	0	0	0	0	0	-2	-2
8	1	239	-1	-1	-1	-1	-1	-1	-1	-1

formule de *Schalkwijk* [Schalkwijk, 1972], permet de donner le rang t d'un vecteur x parmi les $K(j)$ éléments de sa classe.

- Soit q le nombre de valeurs différentes des composantes de x : a_0, a_1, $\cdots$, a_{q-1} avec $a_0 > a_1 > \cdots > a_{q-1}$
- A $x = (x_1, x_2, \cdots, x_8)$, on fait correspondre l'octuplet $q-$aire $D = (d_1, \cdots, d_8)$ tel que $d_k = d$ si et seulement si $x_k = a_d$, $0 \leq d \leq q-1$
- Soit w_k^d le nombre de répétitions de la valeur a_d dans le vecteur x qui se trouvent dans les positions k à 8.

Le rang t de x est alors donné par :

$$t = \sum_{k=1}^{8} \frac{(8-k)!}{\prod_{i=0}^{q-1} w_k^i!} \Big(\sum_{d=0}^{d(k)-1} w_k^d \Big) \tag{2.9}$$

Algorithme de codage

Soit $x = (x_1, \cdots, x_8)$ un vecteur quelconque de $\mathbb{R}^8$ à quantifier par $RE_8(m)$. L'algorithme de codage se décompose en cinq étapes :

1. *Etape 1 :* Recherche du leader $\tilde{x}$ de x.
   Les relations d'équivalence "$\sim$" et l'ordre total "$\leq$" peuvent être étendues sur $\mathbb{R}^8$. Il suffit alors pour trouver le leader $\tilde{x}$ de x de réordonner les composantes de x par ordre décroissant.

2. *Étape 2 :* Recherche du plus proche voisin $\tilde{y}_j*$ de $\tilde{x}$.
   On démontre [Berger *et al.*, 1972], que le plus proche voisin de $\tilde{x}$, leader dans $\mathbb{R}^8$, est un leader de $RE_8(m)$. Ceci revient à chercher parmi les $P_8(m)$ leaders de $RE_8(m)$ $\tilde{y}_j*$ qui permet de maximiser le produit scalaire :

$$\tilde{x}^T \tilde{y}_j* = \max_{j \in (1,\cdots,P_8(m))} (\tilde{x}^T \tilde{y}_j) \tag{2.10}$$

3. *Etape 3 :* Détermination de y le plus proche voisin de x dans $RE_8(m)$.
   Pour trouver y il suffit d'appliquer à $\tilde{y}_j*$ l'inverse de la transformation qui a permis d'aller de x à $\tilde{x}$ [Berger *et al.*, 1972].

4. *Etape 4 :* Déterminer le rang t de y dans sa classe $j*$ par la formule de Schalkwijk (2.9).

5. *Etape 5 :* Déterminer l'indice I du vecteur y dans $RE_8(m)$ donné par :

$$I(y) = t(y) + \sum_{j=1}^{j*-1} K(j) \tag{2.11}$$

Algorithme de décodage

À partir de l'indice $I(y)$ il est possible de retrouver y en suivant ces trois étapes :

1. *Etape 1 :* Recherche de la classe $j*$ (recherche dans le tableau 2.6).
   $j*$ est tel que $H(j*) \leq I(y) < H(j*+1)$

2. *Étape 2 :* Retrouver le rang t de $j*$
   $t = I(y) - H(j*)$

3. *Etape 3 :* Retrouver y sachant son rang dans la classe $j*$
   Ayant déterminé la classe $j*$ et donc son leadeur $\tilde{y}_{j*}$, on connait q le nombre de composantes distinctes, leurs valeurs $a_0, a_1, \cdots, a_{q-1}$ et le nombre w^i de répétitions de chaque valeur a_i dans $\tilde{y}_{j*}$ et par conséquent dans y. Pour trouver y, il suffit de

déterminer $d_1(x_1)$, $d_2(x_2)$ jusqu'à $d_8(x_8)$ où d_k est tel que :

$$\frac{(8-k)!}{\prod_{i=0}^{q-1} w_k^i!}\left(\sum_{d=0}^{d_k-1} w_k^d\right) \leq t - \sum_{j=1}^{k-1} \frac{(8-j)!}{\prod_{i=0}^{q-1} w_j^i!}\left(\sum_{d=0}^{d_j-1} w_j^d\right) < \frac{(8-k)!}{\prod_{i=0}^{q-1} w_i^k!}\left(\sum_{d=0}^{d_k} w_k^d\right) \tag{2.12}$$

Accélération de la recherche par passage aux valeurs absolues

Les points à composantes paires ont leurs 8 signes libres alors que les points impairs ont sept signes libres, le huitième étant fixé pour assurer $\sum_{1}^{8} x_i \equiv 0[4]$. Cette propriété implique que le changement d'un nombre quelconque des signes des composantes d'un point pair de $RE_8(m)$ donne un point de $RE_8(m)$. Inversement, pour un point impair un changement pair de ses composantes donne un point de $RE_8(m)$. Cette propriété permet de définir une nouvelle relation d'équivalence $\approx$ entre les point de $RE_8(m)$: $x \approx y$ si et seulement si les valeurs absolues des composantes des deux vecteurs, une fois arrangées dans l'ordre décroissant, sont identiques. Les classes d'équivalence de cette relation sont des surclasses de celles définies à la section 2.3.3, au sens qu'elles les englobent. Le tableau 2.7 présente les leadeurs absolus et signés pour $RE_8(m = 1)$. On appelle *leadeur absolu*,

Tableau 2.7 Liste des leadeurs absolus et signés de la sphère $m = 1$

Leadeurs absolus	Leadeurs signés
2 2 0 0 0 0 0 0	2 2 0 0 0 0 0 0 2 0 0 0 0 0 0 -2 0 0 0 0 0 0 -2 -2
1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1 1 1 1 1 1 1 -1 -1 1 1 1 1 -1 -1 -1 -1 1 1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1

le plus grand vecteur, au sens lexicographique, qu'on obtient en ordonnant les valeurs absolues dans l'ordre décroissant. Pour les différencier des *leadeurs* définis sur la section 2.3.3, on appellera ces derniers des *leadeurs signés*. Vu que les leadeurs absolus sont moins nombreux que les leadeurs signés, l'étape 2 de l'algorithme de codage donné à la section 2.3.3 peut être optimisée. En effet, il suffit d'effectuer la recherche sur les $P_8^A(m)$ *leadeurs absolus* et non sur les *leadeurs signés*. Pour coder un point x quelconque de $\mathbb{R}^8$, cinq nouvelles étapes sont à suivre :

1. *Etape 0 :* Passage du vecteur $x = (x_1, \cdots, x_8)$ à $|x| = (|x_1|, \cdots, |x_8|)$. Soit N_{par}, la parité du nombre des composantes négatives.

2. *Etape 1 :* Recherche du leadeur $|\tilde{x}|$ de $|x|$

3. *Étape 2 :* Recherche du plus proche voisin $|\tilde{y}|_{j*}$ de $|\tilde{x}|$
   Recherche parmi les $P_8^A(m)$ leaders absolus de $RE_8(m)$ celui qui maximise le produit scalaire modifié donné par :

$$P_{mod} = |\tilde{x}|^T |\tilde{y}|_{j*} - 2\epsilon \tag{2.13}$$

   avec : $\epsilon = \begin{cases} 0 \text{ Si le leadeur est pair ou si } N_{par}(x) = N_{par}(|\tilde{y}|_{j*}) \\ |\tilde{x_8}|^T |\tilde{y_8}|^T \text{ Le produit des dernières composantes sinon} \end{cases}$

4. *Etape 3 :* Détermination du plus proche voisin $|y|$, de $|x|$ dans $RE_8(m)$ par permutation inverse de celle effectuée à l'étape 1. Dans le cas où $\epsilon \neq 0$, on change le signe de la plus petite composante de $|\tilde{y}|_{j*}$.

5. *Etape 4 :* Passage de $|y|$ à y. Il suffit de changer le signe des composantes de y qui correspondent à des composantes négatives de x.

Sur le module TCX de l'AMR-WB+, la quantification algébrique a été appliquée. La section suivante décrit l'algorithme de quantification utilisé pour coder les différents coefficients spectraux.

2.4 Application de la quantification sphérique

La quantification sphérique ou la quantification par treillis est connue comme étant une technique de quantification efficace. Dans ce type de quantification, le dictionnaire est un réseau régulier de points. Cette structure bien définie des points fait en sorte que la recherche et l'indexation du voisin le plus proche soient très rapides et efficaces. L'aspect le plus intéressant avec ce type de quantification est que les mots du dictionnaire peuvent ne pas être sauvegardés parce qu'ils peuvent être générés par une simple règle algébrique.

2.4.1 Quantification algébrique encastrée

Le codec AMR-WB, étant un codec multi-débit, il fallait aussi que son successeur le soit aussi. Pour cela, la quantification algébrique encastrée est appliquée. Le quantificateur algébrique encastré (QAE) est un quantificateur à débit variable en 8 dimensions qui

contient l'origine Q_0 et 3 sous-quantificateurs sphériques à savoir Q_2, Q_3 et Q_4 [Xie et Adoul, 1996; Ragot *et al.*, 2004] dont les tailles sont respectivement 256, 4096 et 65536 et qui ont été construits par l'inclusion de plusieurs quantificateurs sphériques (Table 2.8). La table 2.8 présente les leadeurs absolus de ces quantificateurs qui ont été sélectionnés

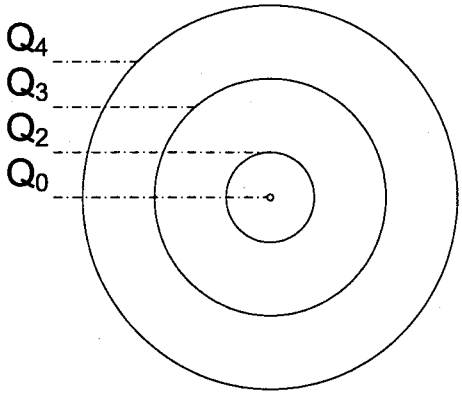


Figure 2.6 Structure du quantificateur algébrique encastrée

selon leurs fréquences d'apparition pour une longue séquence d'entrainement de signaux audios. Chaque vecteur de $\mathbb{R}^8$ est alors indexé par deux indices à savoir :

Tableau 2.8 Liste des leadeurs absolus pour Q_0, Q_2, Q_3 et Q_4 [Ragot *et al.*, 2004]

Taille	Q_0	Q_2	Q_3	Q_4	Leadeurs absolus							
1	✓				0	0	0	0	0	0	0	0
112		✓	✓		2	2	0	0	0	0	0	0
16		✓	✓		4	0	0	0	0	0	0	0
128		✓	✓		1	1	1	1	1	1	1	1
1120			✓		2	2	2	2	0	0	0	0
1344			✓		4	2	2	0	0	0	0	0
112			✓		4	4	0	0	0	0	0	0
224			✓		6	2	0	0	0	0	0	0
16			✓		8	0	0	0	0	0	0	0
1024			✓		3	1	1	1	1	1	1	1
7168				✓	4	2	2	2	2	2	2	0
1792				✓	2	2	2	2	2	2	0	0
3584				✓	3	3	1	1	1	1	1	1
8960				✓	4	2	2	2	2	0	0	0
6720				✓	4	4	2	2	0	0	0	0
448				✓	4	4	4	0	0	0	0	0
1024				✓	5	1	1	1	1	1	1	1
4480				✓	6	2	2	2	0	0	0	0
2688				✓	6	4	2	0	0	0	0	0
112				✓	6	6	0	0	0	0	0	0
1344				✓	8	2	2	0	0	0	0	0
224				✓	8	4	0	0	0	0	0	0
224				✓	10	2	0	0	0	0	0	0
16				✓	12	0	0	0	0	0	0	0
7168				✓	5	3	1	1	1	1	1	1
1024				✓	7	1	1	1	1	1	1	1
256				✓	2	2	2	2	2	2	2	2
8960				✓	3	3	3	3	1	1	1	1
112				✓	8	8	0	0	0	0	0	0
1024				✓	9	1	1	1	1	1	1	1
224				✓	10	6	0	0	0	0	0	0
224				✓	12	4	0	0	0	0	0	0
224				✓	14	2	0	0	0	0	0	0
16				✓	16	0	0	0	0	0	0	0
7168				✓	3	3	3	1	1	1	1	1
112				✓	10	10	0	0	0	0	0	0
224				✓	12	8	0	0	0	0	0	0

- Un indice représentant l'étage de quantification n utilisant exactement n bits.

- Un indice I représentant l'indice du vecteur en question dans cet étage codé sur $4n$ bits.

Quand le mode TCX est sélectionné, la quantification algébrique encastrée est utilisée pour déterminer les différents paramètres à envoyer au décodeur. Pour un débit fixé, et pour un mode fixé (TCX256, TCX512 ou TCX1024), un nombre de bit donné N_{bit} est alloué pour la sous-trame en cours. Le codec doit alors indexer les différents paramètres sans dépasser ce débit donné. Notons par $X(k)$ le spectre à l'entrée de l'étage de quantification, et par B_k les Nsv (36, 72, 144 selon le mode TCX sélectionné) blocs successifs de 8 échantillons chaque. L'algorithme de quantification contient cinq étapes :

1. *Étape 1 :* Trouver E_k l'énergie de chaque bloc B_k

$$E_k = \max(2, \sum_{m=0}^{7} B_k[m]B_k[m]) \tag{2.14}$$

   Ensuite obtenir une première estimation $R_k(1) = 5\log_2(\frac{E_k}{2})$ du budget de bit en supposant que le gain g vaut 1. La formule de $R_k(1)$ est basée sur les propriétés du réseau E^8 [Lamblin et Adoul, 1988]. À moins que l'énergie de la trame soit faible, l'énergie des blocs B_k est très élevée ce qui crée un dépassement du débit alloué pour chaque trame. Pour remédier à ce problème, une estimation d'un gain de normalisation est effectuée dans l'étape 2.

2. *Étape 2 :* Estimation d'un gain global g
   L'estimation du gain g se fait en plusieurs itérations données par l'algorithme 2.1.

Algorithme 2.1 : Algorithme d'estimation du gain g

Entrées : $B_k \in \mathbb{R}^8$ $k = [1 : N]$ et Nsv : Nombre de blocs de longueurs 8 coefficients
Sorties : $g : \in \mathbb{R}$.
Données : fac= 128, offset= 0, $nbit_{max} = 95\% \times (N_{bit} - Nsv)$, $N_{It} = 10$
pour *j=1 :*N_{It} **faire**
  offset=offset+fac;
  $nbits = \sum_1^N \max(0, R_k(1) - \text{offset})$;
  si $nbits \leq nbit_{max}$ **alors**
    offset=offset-fac;
    fac=fac/2;
  fin
  $g = 10^{\text{offset}\times\log_{10}(2)/10}$;
fin

A l'issue de cet algorithme, un gain g a été estimé. Le spectre X alors est normalisé avec ce gain avec l'hypothèse que la quantification utilisera un débit proche du celui qui lui est alloué.

3. *Étape 3 :* Recherche du plus proche voisin dans RE_8
   Un algorithme rapide et efficace de recherche du plus proche voisin a été implémenté sur l'AMR-WB+ [3GPP, 2007-03; Ragot *et al.*, 2004; Conway et Sloane, 1982]. Soit B'_k le bloc à quantifier, le plus proche voisin y_k peut être trouvé suivant l'algorithme 2.2. A l'issue de cette étape, le plus proche voisin de B'_k a été trouvé au cours

Algorithme 2.2 : Algorithme de quantification rapide

Entrées : B'_k : Vecteur de $\mathbb{R}^8$.
Sorties : y_k : Vecteur de $\mathbb{Z}^8$.
Données : $\underline{1}$: Vecteur de 1 dans $\mathbb{Z}^8$
pour *j=0 :1* **faire**
  $z_k = 1/2(B'_k - j \times \underline{1})$;
  $\overline{z_k} = E(z_k)$ Où E est la partie entière;
  $y_{jk} = 2\overline{z_k}$;
  $S = \sum_1^8 y_{jk}$;
  si $S \neq 0[4]$ **alors**
    Trouver i pour le quel $|z_k(i) - y_{jk}(i)|$ est le plus grand ;
    si $z_k(i) > y_{1k}$ **alors**
      $y_{jk}(i) = y_{jk}(i) + 2$
    sinon
      $y_{jk}(i) = y_{jk}(i) - 2$
    fin
  fin
  $y_{jk} = y_{jk} + j \times \underline{1}$;
  $e_{jk} = \|B'_k - y_{jk}\|^2$;
fin
$y_k = y_{j'k}$ tel que $j' = \arg\min_{j=0:1} e_{jk}$

des deux itérations. A noter que dans l'algorithme 2.2 le cas $j = 0$ correspond à la recherche dans $2D_8$ alors que le cas $j = 1$ correspond à $2D_8 + [1, 1, 1, 1, 1, 1, 1, 1]$.

4. *Étape 4 :* Estimation et quantification du bruit de confort g
   Le bruit de confort est un bruit artificiel dont le but est d'améliorer la qualité du son perçu. Le bruit de confort ne sera ajouté qu'aux composantes spectrales non quantifiées.

5. *Étape 5 :* Quantification dans RE_8 et extension de Voronoï
   Cette étape cherche à trouver des indices dans RE_8 pour les y_k trouvés à l'*Étape 3.*

Comme vu précédemment, si $y_k \in Q_n$ alors la formule de *Schalkwijk* [Schalkwijk, 1972] permet de trouver I l'indice de y_k dans Q_n. Quand y_k n'est dans aucun des Q_n (i.e $\sum_{i=1}^{8} y_k(i)^2 > 32$), une méthode basée sur l'extension de Voronoï est utilisée pour ramener y_k à un vecteur qui serait dans un des dictionnaires de base.

2.4.2 L'extension de Voronoï

L'extension de Voronoï est basée sur les similitudes des treillis (dans le sens des fractales). Pour un treillis de rang plein Λ, on a [Ragot *et al.*, 2004]

$$\Lambda = m\Lambda + V_m = \bigcup_{C\in\Lambda, v\in V_m} mc + v \tag{2.15}$$

Où $V_m = \Lambda \cap (mV(\Lambda) + a)$ est un code Voronoï dérivé de Λ avec $\log_2 m$ bits par dimension [Conway et Sloane, 1983]. L'extension de Voronoï d'un dictionnaire de base C à $\mathbb{R}$ bit par dimension est donnée par [Ragot *et al.*, 2004] :

$$C^{(r)} = 2^r C + V_{2^r} \tag{2.16}$$

Il est donc possible d'indexer n'importe quel point dans Λ, il suffit de trouver un $r \in \mathbb{R}$ et un $v \in \mathbb{R}^8$ qui permettent de respecter l'équation 2.16. Sur l'AMR-WB+ l'algorithme permettant d'indexer n'importe quel point de RE_8 est présenté par l'algorithme 2.3.

Ainsi un point de RE_8 qui se trouve à l'extérieur de Q_4 est indexé par un indice composé de trois parties :

- n'[1] indice du dictionnaire de base (Q_3 ou Q_4) et l'ordre d'extension de Voronoï .
- I indice de c_k dans Q_3 ou Q_4.
- Les 8 indices du vecteur de Voronoï k calculé dans l'algorithme 2.3 où chaque composante prend exactement r bits.

L'algorithme utilisé pour décoder l'indice I est simple et présenté par l'algorithme 2.4.

Ayant déterminé les différents indices des sous-vecteurs des coefficients spectraux, la prochaine étape consiste à les empaqueter. La section suivante présente la procédure mise

[1] n' (respectivement I') sera utilisé comme notation dans le cas où n (respectivement I) est multiplexé avec r ordre de l'extension de Voronoï (respectivement avec k indice de l'extension de Voronoï).

Algorithme 2.3 : Algorithme de calcul des indices n' et I' par l'extension de Voronoï

Entrées : y_k : Vecteur de RE_8
Sorties : n' : Etage de quantification, I' : Indice de y_k dans $Q_{n'}$
Données : $r = 0$
si $y_k \in Q_n$ **alors**
  Calculer I l'indice de y_k (équation 2.9)
sinon
  tant que $c \notin Q_n$ *où* $n = 3, 4$ **faire**
    r=r+1;
    Trouver $v \in V_{2^r}$ tel que $y_k \in 2^r RE_8 + v$;
    Calculer $c = \frac{1}{2^r}(y - v)$;
  fin
  $n' = n + 2r$;
  Calculer I indice de c dans $Q_{n'}$;
  Calculer k indice de Voronoï de v dans V_{2^r} [Conway et Sloane, 1983];
  Multiplexer I et k pour avoir l'indice I' de y dans $Q_{n'}$;
fin

Algorithme 2.4 : Algorithme de décodage de l'indice I

Entrées : n : Entier $\in 0, 2, \cdots$, I : Entier dans N
Sorties : y : Vecteur de RE_8
si $n \leq 4$ **alors**
  Décoder I directement dans Q_n
sinon
  $r = (n - 3)/2$;
  $n' = n - 2r$;
  Démultiplexer j et k;
  Décoder j en c dans $Q_{n'}$;
  Décoder k indice de Voronoï en v dans V_{2^r};
  $y = 2^r c + v$;
fin

en oeuvre sur le codec AMR-WB+. On s'intéressera dans la section suivante à la procédure qui permet d'écrire les indices binaires (n' et I') dans une table binaire.

2.5 Empaquetement des indices binaires du quantificateur algébrique

Dans le codec AMR-WB+, les indices n' des étages de quantification sont encodés avec un code unaire alors que les indices I' sont codés suivant un code binaire à longueur fixe

(section 3.2.1). Le code unaire utilisé est le suivant (section 3.3.1 du présent document) :

0	Q_0
10	Q_2
110	Q_3
1110	Q_4
11110	Q_3 Avec l'extension de Voronoï $r = 1$
111110	Q_4 Avec l'extension de Voronoï $r = 1$
...	...

Chaque n' prend exactement n' bits à l'exception de Q_0 qui prend un bit. Les indices I prennent $4n$ bits, en effet si $n' \leq 4$ alors I se décode directement dans Q_n. Q_2 contenant 256 vecteurs, Q_3 4096 vecteur et Q_4 65636 vecteurs, $8 = 2 \times 4$, $12 = 3 \times 4$ et $16 = 4 \times 4$ bits sont respectivement nécessaires pour indexer n'importe quel vecteur de $\mathbb{Z}^8$. Quand $n' > 4$, k (vecteur de Voronoï à 8 composante) utilise r bits par composantes, sachant que $r = (n' - n)/2$, indexer I' utilise $4n + 8(n' - n)/2 = 4n'$ bits. En total $5n'$ bits sont utilisés pour indexer n'importe quel vecteur, à l'exception de Q_0 qui utilise 1 bit.
Cette partie explique comment les paramètres du codeur TCX sont multiplexés dans des paquets en vue de transmission. Pour le mode TCX256 un seul paquet est utilisé, par contre pour le mode $TCX512$ (respectivement $TCX1024$) 2 (respectivement 4) paquets sont utilisés. Pour distribuer les différents paramètres issus de la quantification, le spectre est divisé en plusieurs pistes, chaque piste contenant un ensemble de fragments du spectre quantifié (Tableau 2.9). Les indices binaires issus de la quantification algébrique encastrée

Tableau 2.9 Subdivision du spectre en différents fragments en vue d'empaquetement

	Numéro du fragment						
TCX256	Piste1	0,	1,	2,	3,	4,	...
TCX512	Piste1	0,	2,	4,	6,	8,	...
	Piste2	1,	3,	5,	7,	9,	...
TCX1024	Piste1	0,	4,	8,	12,	16,	...
	Piste2	1,	5,	9,	13,	17,	...
	Piste4	2,	6,	10,	14,	18,	...
	Piste5	3,	7,	11,	15,	19,	...

sont écrits en allant des basses fréquences vers les hautes fréquences en écrivant les n' sous forme de code unaire en commençant de droite à gauche et les I' sont écrits par bloc de 4

bits en allant de gauche à droite sur les différents paquets et ce en suivant ce format :

$$[I_0, I_1, I_2, \cdots, n_2, n_1, n_0] \tag{2.17}$$

Cette forme permet de tenir en compte de la sensibilité des différents bits des indices I' et n'. En effet, n' étant l'indice le plus sensible, il est écrit à la fin du paquet binaire permettant ainsi de mieux le protéger contre les bruits de transmission.

2.6 Conclusion

On a présenté dans la première section de ce chapitre le codec AMR-WB+ ainsi que le module TCX. Ce dernier utilisant la quantification algébrique encastrée, la deuxième section a été consacrée à détailler le fonctionnement de ce module ainsi que la génération des indices binaires. La quantification mise en oeuvre par ce codec permet d'adapter automatiquement le nombre de bits utilisés par chaque coefficient spectral pour un débit donné, et ce, en prédisant le nombre de bits nécessaires à la quantification. Cette méthode d'estimation de débit, basée sur les propriétés des réseaux de treillis dans RE_8, se base sur l'hypothèse que les différents indices sont codés avec des mots binaires de longueur fixe. Cette hypothèse suppose implicitement aussi que les indices binaires sont uniformément distribués pour chaque dictionnaire Q_n.
Ce projet étant consacré à développer un module de compression des indices binaires de la quantification algébrique, le chapitre suivant aura pour but d'énoncer le fondement de la théorie d'information ainsi que les algorithmes de compression les plus présents dans la littérature.

CHAPITRE 3

Compression sans pertes

Compresser revient à représenter les messages de la source avec un nombre de bits plus petit que celui utilisé au départ. La compression sans perte est, comme l'indique son nom, sans perte d'information. Si un message a été compressé sans perte alors il n'y a pas de différence entre le message original et le message reconstruit. La compression de texte est un domaine d'application important pour la compression sans perte. En effet, il est très important que le message de départ et le message reconstruit soient identiques. Une petite différence peut générer une grande différence dans le sens du message (« Viens prendre la marchandise » et « Viens vendre la marchandise »). Suivant le même raisonnement, sur l'AMR-WB+, les indices de la quantification représentant les coefficients spectraux de la TCX doivent être compressés sans perte. Une erreur peut générer un message audio bruité voir incompréhensible.
Un algorithme de compression peut être évalué selon plusieurs critères. On peut mesurer sa complexité, son occupation de la mémoire et son taux de compression. Ce dernier dépend de plusieurs facteurs dont le plus important est les caractéristiques de la source à compresser : « Plus le message est prédictible, plus il est compressible ».
Le développement des algorithmes de compression peut être subdivisé en deux étapes. Une étape de modélisation qui consiste à extraire tout type de redondance présente dans le message de la source. La deuxième étape, appelée encodage, cherche à représenter ces messages par un ou plusieurs mots d'un alphabet binaire. La première partie de ce chapitre sera consacrée aux techniques de modélisation des données alors que la seconde partie aura pour but d'énoncer les algorithmes de compression sans perte les plus répandus dans la littérature.

3.1 Théorie d'information : Entropie de Shannon

Bien que l'idée de mesurer la quantité d'information a été présente depuis longtemps, ce ne fut qu'avec les travaux de Shannon [Shannon, 1948] que cette idée prit une forme mathématique. Soit une source S à n symboles s_i de probabilité p_i alors la quantité d'information

H_{s_i} présente dans l'événement s_i est :

$$H_{s_i} = \log_2 \frac{1}{p_i} \tag{3.1}$$

alors que la quantité d'information H_S (entropie) de la source est donnée par :

$$H_S = -\sum_{i}^{n} p_i \log_2(p_i) \tag{3.2}$$

Shannon dans [Shannon, 1948] a montré que le meilleur encodage sans perte d'une source utilisera un code binaire dont la longueur moyenne est égale à l'entropie de la source. L'équation 3.1 permet alors de calculer une borne inférieure pour la longueur du code binaire. Il faut cependant retenir que cette borne stipule que les symboles de la source sont considérés indépendants. Si ce n'est pas le cas alors on peut exploiter cette dépendance pour réduire l'interprétation de l'entropie et obtenir un meilleur taux de compression. Prenons à titre d'exemple cette séquence $S = [1, 2, 1, 2, 1, 2, 3, 1, 2, 3]$. On a alors $p(1) = p(2) = 4/10$ et $p(3) = 2/10$ donc $H_S = 1.52$ bits/ Symbole et donc encoder S revient à utiliser 15.2 bits. Si par contre on considère la source étant comme $S' = [\{1, 2\} - \{1, 2\} - \{1, 2\} - \{3\} - \{1, 2\} - \{3\}]$ alors $p(\{1, 2\}) = 4/6$ et $p(\{3\}) = 2/6$ et $H_{S'} = H_S = 1.521$ bits alors qu'encoder S' prend $9.131 \leq 15.219$ bits ce qui permet d'achever un taux de compression $\tau_c = 60\%$. Le fait d'avoir un taux de compression de 60% provient de la façon avec laquelle l'entropie de la source est interprétée : est-ce que les symboles qu'elle génère sont indépendants ? Existe-il une structure dans la source qui permet d'améliorer les performances de la compression ? Autrement dit quelle est la meilleure façon d'estimer l'entropie ? Une fois que la structure dans la source a été déterminée comment générer un code binaire efficace pour représenter les symboles de la source ?

3.2 Estimation de l'entropie

Comme vu dans la section 3.1, avoir un bon modèle pour la source peut s'avérer très utile pour estimer son entropie. Comme on verra dans les chapitres suivants, plus le modèle est fidèle à la source, plus les algorithmes de compression sont efficaces. Généralement, pour pouvoir manipuler les symboles de la source, un modèle mathématique est essentiel. La construction de ce modèle peut se faire suivant plusieurs approches.

3.2.1 Modèles probabilistes

Le modèle statistique le plus simple pour une source est de considérer les symboles qu'elle génère comme étant indépendants les uns des autres, et que chaque symbole se produit avec la même probabilité. Ce modèle est appelé *modèle d'ignorance* [Sayood, 2000] qui n'est utile que si on ne connait rien sur la source. C'est ce genre de modèle qui a été utilisé par le AMR-WB+ pour coder les indices I', en utilisant un code binaire de longueur $4n'$ par indice, n' étant l'étage de quantification utilisé. La prochaine étape en complexité consiste à garder l'hypothèse d'indépendance des symboles de la source et d'assigner une probabilité à chaque symbole. Pour une source S on peut avoir le modèle de probabilité suivant $P = \{P(s_1), P(s_2), \cdots, P(s_m)\}$ et utiliser l'équation 3.2 pour calculer l'entropie de la source définissant ainsi une borne inférieure pour la compression sans perte. On verra dans les sections suivantes comment, en utilisant ce modèle de probabilité, il possible de construire des codes binaires efficaces i.e. leur longueur moyenne tend vers l'entropie de la source quand le message à compresser devient plus long. Si l'hypothèse d'indépendance ne correspond pas aux observations des symboles, on peut généralement omettre cette dernière hypothèse et avoir de meilleurs résultats, et ce, en trouvant une façon de décrire l'indépendance entre les symboles.

3.2.2 Chaines de Markov

Une des façons les plus populaires pour décrire la dépendance entre les données est la chaine de Markov discrète [Cormack et Horspool, 1986]. Soit la séquence $\{s_n\}$ de symboles générés par la source. Cette séquence est dite chaine de Markov d'ordre k ssi :

$$P(s_n|s_{n-1}, \cdots, s_{n-k}) = P(s_n|s_{n-1}, \cdots, s_{n-k}, \cdots) \tag{3.3}$$

Autrement dit, la connaissance des derniers k symboles est équivalente à la connaissance du passé entier de la source. Les valeurs $\{s_{n-1}, \cdots, s_{n-k}\}$ sont appelés états de la source et si le nombre de symboles différents générés par la source est n alors on a n^k états. Le modèle le plus couramment utilisé est le modèle premier ordre donné par $P(s_n|s_{n-1}) = P(s_n|s_{n-1}, \cdots)$. Cette écriture affirme la dépendance entre les différents symboles, mais ne décrit pas le type de dépendance. On peut alors développer différents modèles de Markov de premier ordre. Par exemple, on peut supposer que les données sont linéairement dépendantes, $s_n = s_{n-1} + \epsilon_n$ où ϵ_n est un bruit blanc. L'entropie d'un processus à états

finis s_i de nombre n est donné par l'équation 3.4

$$H = \sum_{i=1}^{n} p_{s_i} H(s_i) \tag{3.4}$$

Supposons une source à deux symboles s_0 et s_1 avec les probabilités $p_{s_0} = 9/10$ et $p_{s_1} = 1/10$ alors $H_S = 0.469$ bit. Supposons maintenant que $p_{s_0/s_0} = 99/100$, $p_{s_1/s_0} = 1/100$,

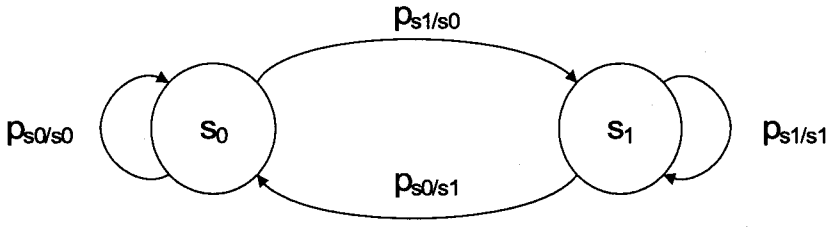


Figure 3.1 Modèle de Markov discret à deux états

$p_{s_1/s_1} = 10/100$ et $p_{s_0/s_1} = 90/100$ alors :

$$\begin{aligned}
H(s_0) &= -1/100 \times \log_2(1/100) - (99/100) \times \log_2(99/100) = 0.0808 \\
H(s_1) &= -10/100 \times \log_2(10/100) - (90/100) \times \log_2(90/100) = 0.4690 \\
H'_S &= +9/10 \times 0.0808 + 1/10 \times 0.4690 = 0.1196
\end{aligned}$$

En utilisant l'équation (3.4), on a trouvé que l'entropie pour le modèle de Markov est de 0.1196 bit, 75% plus petite que l'entropie calculée avec l'hypothèse d'indépendance des symboles.

3.2.3 Transformations réversibles de la source

Les performances d'un algorithme de compression étant très liées à l'entropie de la source, borne inférieure pour la compression sans perte sous l'hypothèse de l'indépendance, l'amélioration de cette borne peut se faire à travers diverses transformations réversibles de la source. On peut soit réarranger les symboles dans le but de créer de la redondance exploitable, soit réduire la longueur du message lui-même.

Codage par plages

Le codage par plage ou le *Run Length Coding*, consiste à coder le nombre d'occurrences successives d'un symbole [Sayood, 2000].
Soit la séquence binaire $S = [1,1,1,1,0,0,0,0,0,0,0,0,1,1,1,1]$ dont l'entropie est 1

bit/Sym. Au lieu de coder S telle que générée par la source, on peut penser à une autre stratégie. On peut coder le nombre d'apparitions de chaque symbole suivant cette stratégie :$[1 \times \mathbf{4}, 0 \times \mathbf{8}, 1 \times \mathbf{4}]$. On peut utiliser alors 1 bit pour encoder les symboles « 1 » et « 0 » et 4 bits pour coder le nombre d'occurrences. Encoder toute la séquence utilisera alors $3 + 12 = 15$ bits ≤ 16 bits. Décoder la séquence reçue $[1 \times \mathbf{4}, 0 \times \mathbf{8}, 1 \times \mathbf{4}]$ est très simple. Le décodage se fait par alternance, un symbole ensuite un nombre d'occurrences. Cette méthode est souvent utilisée pour le codage du texte en noir et blanc si le document est considéré comme un long vecteur, et où il y a souvent de très longues plages de zéro et de un.
Cette méthode est assez facile à mettre en oeuvre et peut être combinée avec d'autres algorithmes de transformation. Cependant, si cette méthode de transformation est mal choisie, elle peut résulter en une augmentation de la taille du message. Il suffit pour comprendre cette idée de considérer le message suivant $[1, 2, 3]$.

La transformation de Burrows-Wheeler

Pour que l'algorithme de codage par plages soit efficace, il est préférable que les symboles identiques soient successifs. La transformation de Burrows-Wheeler [Burrows *et al.*, 1994], est une transformation réversible qui permet d'augmenter les probabilités que des caractères identiques initialement éloignés les uns des autres se retrouvent côte à côte. Considérons l'exemple de la séquence suivante : $S = [1, 2, 1, 3]$. La transformation est faite en triant par ordre lexicographique les rotations de la séquence S ensuite en prenant la dernière colonne. Le résultat de la rotation est une séquence $S' = [3, 2, 1, 1]$ dont le codage

1	2	1	3
3	1	2	1
1	3	1	2
2	1	3	1

			↓	
1	2	1	3	←1
1	3	1	2	
2	1	3	1	
3	2	1	1	

par plages est efficace grâce à la présence de symboles identiques consécutifs. Pour retrouver S, le décodeur à besoin évidement de S' mais aussi de l'indice de S dans la table des rotations tirée. Dans notre cas cet indice correspond à 1.
Reconstruire la séquence initiale S à partir de S' et l'indice 1 est facile. En effet, S' contient tout les caractères contenus dans S. Il suffit de la trier et ensuite de la juxtaposer à S'. Ayant obtenu ainsi un vecteur formé par des pairs, il suffit de le trier, ensuite de juxtaposer S' et ainsi de suite. L'indice 1 permet de trouver S sur la dernière colonne. La transformée de Burrows-Wheeler permet de transformer de façon bijective la source pour créer une redondance exploitable en compression. Cependant, cette technique souffre de deux

3 **31** **312** **3121** 1213 ←
2 **21** **213** **2131** 1312
1 **12** **121** **1213** 2131
1 **13** **131** **1312** 3121

inconvénients majeurs à savoir la nécessité d'avoir toute la séquence à encoder avant de commencer la transformation. Il est aussi difficile de prédire le résultat avant d'effectuer la transformation.

3.3 Construction du code binaire

3.3.1 Codage de Huffman

Une chaine de transmission numérique stipule l'utilisation de symboles binaires pour encoder les différents messages. Dans la suite de ce document, *encoder* fera référence à la représentation des messages de la source par des symboles binaires, la séquence binaire est alors appelé *code* et les membres individuels sont appelés *mots binaires*. La mission du décodeur consiste alors à récupérer (décoder) les messages à partir du code. Dans cette section on va discuter différents algorithmes qui permettent de coder de façon efficace (dans le sens de l'entropie) les différents messages de la source. Par exemple, le code utilisé pour représenter l'alphabet est le code ASCII qui utilise 8 bits par lettre ou ponctuation. Par exemple le code ASCII du "a" est 01100001 alors que celui du "A" est "01000001". Ce code utilise un nombre fixe de bits pour tous les 256 caractères. Ce code est alors appelé code à longueur fixe. Un autre exemple serait le code binaire utilisé par le AMR-WB+ pour représenter les indices I' de la quantification algébrique (section 2.5). Si on veut réduire le nombre de bit utilisé pour représenter différents messages, on a besoin d'utiliser différents nombres de bits pour chaque symbole. Si on utilise moins de bits pour les symboles les plus fréquents, en moyenne on va utiliser moins de bits par symbole. Le nombre de bits/Symbole est appelé débit binaire par symbole qui peut être un nombre non entier. L'idée de représenter les symboles moins fréquents par un nombre de bits plus élevé est la même utilisée par le code *Morse* où le "A" est représenté par : "._" alors que le "Y", moins fréquemment utilisé, est représenté par "_. _ _ _" [Hale, 1858]. La longueur moyenne d'un code n'est pas seulement la plus importante propriété lors de la construction du code binaire. Tous les codes sans pertes se partagent trois propriétés importantes à savoir :

- La non-singularité : C'est à dire que deux mots différents seront codés différemment.
- Chaque symbole est uniquement décodable.

- Chaque symbole n'est le préfixe d'aucun autre symbole.

Ces propriétés peuvent se résumer dans l'inégalité de Kraft-McMillan. Pour un code C à N mots binaires de longueur l_i bits chacun, une condition nécessaire pour que ce code soit uniquement décodable est donnée par l'équation (3.5).

$$K(C) = \sum_{i=1}^{N} 2^{-l_i} \leq 1 \tag{3.5}$$

L'inégalité de McMillan fournit une condition nécessaire pour que le code soit un code à préfixe. Le théorème intéressant qui permet d'exploiter cette condition est celui donné par [McEliece, 1977]. Pour un ensemble d'entier l_i qui vérifient l'inégalité de McMillan, il existe toujours un code à préfixe dont la longueur des mots binaires est l_i. Donc si on a un code uniquement décodable, les longueurs des mots binaires doivent vérifier l'inégalité de McMillan, et si cette inégalité est satisfaite on peut toujours trouver un code à préfixe avec ces longueurs. Un code entropique à mots binaires de longueurs variables cherche à minimiser la longueur de ces mots en se basant sur les statistiques de la source. i.e. plus un symbole est utilisé, plus le mot binaire qui lui est associé est court, permettant ainsi de réduire le débit total moyen. Un code typique inspiré de ce principe est le code de Huffman [Huffman, 1952]. Ce code est basé sur deux observations :

- Dans un code optimal, les symboles qui se produisent plus fréquemment auront des mots binaires de longueurs plus petites que les symboles moins fréquents.

- Dans un code optimal, les deux symboles qui se produisent le moins fréquemment ont la même longueur de mot binaire.

La première remarque est évidente. Pour comprendre la deuxième, un raisonnement par absurde s'impose. Soit un code optimal C pour lequel les deux symboles les moins fréquents ont des mots binaires m_1 et m_2 et dont les longueurs diffèrent de k bits. Comme c'est un code à préfixe, le mot le plus court m_1 ne peut être un préfixe du mot m_2. Ce qui veut dire que même si on enlève les k bits du mot binaire le plus long, les deux mots m_1 et m_2 restent encore distincts ! Comme ces deux mots sont les moins probables, aucun autre mot ne peut être plus long que m_1 et m_2. En plus en enlevant ces k bits, on obtient un code de longueur moyenne plus petite que celle de C, ce qui contredit l'hypothèse de départ (C est un code optimal).
Le code de Huffman est construit en ajoutant une autre contrainte à ces deux observations. Les deux mots binaires m_1 et m_2 ne différent que par leur dernier bit. Cette contrainte ne viole pas les deux observations précédentes et permet de développer une procédure

d'encodage simple qu'on expliquera à travers l'exemple suivant. Soit les symboles a_1, a_2 et a_3 avec les probabilités respectives 0.2, 0.5 et 0.3. Cette source a une entropie de 1.485 bit/symbole. Notons par $C(a_i)$ le code du mot a_i. Les deux mots a_3 et a_1 sont les moins

Symbole	Probabilité	Code
a_2	0.5	$C(a_2)$
a_3	0.3	$C(a_3)$
a_1	0.2	$C(a_1)$

probables donc ils ont la même longueur de mot et ne différent que d'un seul bit. On peut donc choisir $C(a_3) = \alpha * 1$ et $C(a_1) = \alpha * 0$ où α est un mot binaire à déterminer et $*$ représente la concaténation. On se trouve alors avec un nouvel alphabet a_2 et a' tel que $p(a') = 0.5$. Comme il ne reste que a_2 et a' alors $C(a_2) = 0$ et $C(\alpha) = 1$. Donné

Symbole	Probabilité	Code
a_2	0.5	$C(a_2)$
a'	0.5	α

la valeur de α on peut retrouver le reste du code. La longueur moyenne de ce code est

Symbole	Probabilité	Code
a_2	0.5	0
a_3	0.3	10
a_1	0.2	11

$0.5 * 1 + 0.3 * 2 + 0.2 * 2 = 1.5$. La redondance de ce code est donnée par la différence entre la longueur moyenne de ce code et l'entropie et vaut 0.014 bit/symbole. L'inégalité de McMillan est aussi vérifiée puisque $2^{-1} + 2^{-2} + 2^{-2} \leq 1$.

Le code de Huffman, même s'il approche l'optimalité théorique donnée par Shannon c'est à dire qu'il ne nécessite qu'un bit de plus que l'entropie de la source dans le pire des cas, présente deux inconvénients majeurs à savoir la nécessité de connaitre à priori les statistiques de la source et la nécessité de transmettre au décodeur la table de correspondance entre symboles et messages. Pour remédier à cet inconvénient, différentes variantes du code de Huffman ont vu le jour dont la plus intéressante est le code de Huffman adaptatif [Gallager, 1978; Javed et Nadeem, 2000; Desoky et Gregory, 1988]. Le code de Huffman adaptatif se base sur les occurrences d'apparition des symboles de la source pour estimer les probabilités. Donc lors de l'encodage du message les codes changent en fonction du nombre d'apparition de chaque symbole. Au niveau du décodeur, la même tâche est effectuée pour décoder le message et donc plus besoin de transmettre la table de correspondance entre les messages et les mots binaires. Même si le code de Huffman adaptatif

atteint asymptotiquement l'optimalité [Gallager, 1978], il reste encore très sensible aux erreurs au niveau du décodeur, en effet une erreur de décodage ou de transmission peut se propager tout au long du message vu que ce dernier processus se base sur l'occurrence de chaque symbole pour estimer les probabilités.
Il est aussi à noter que le code de Huffman ne permet d'utiliser qu'un nombre entier de bits par symbole. Il n'est donc optimal (i.e. longueur moyenne du code est égale à l'entropie) que si les probabilités des symboles sont des puissances négatives de 2. Quand l'ensemble des symboles de la source est infini, et que les probabilités sont des puissances négatives de 2, le codé de Huffman se résume à encoder les symboles avec $\log_2(p_i)$ de bits. Ce code est appelé aussi code unaire représenté par la table 3.1. Il est facile de vérifier que ce code

Tableau 3.1 Le code unaire

Symbole	Probabilité	Code
a_1	$1/2$	0
a_2	$1/4$	10
a_3	$1/8$	110
$\vdots$	$\vdots$	$\vdots$
a_l	$1/2^l$	$\underbrace{1,\cdots,1}_{l-1}0$

est optimal sous ces dernières hypothèses. En effet

$$\begin{aligned} H_C &= -\sum_{l=1}^{\infty} \frac{1}{2^l} \times \log_2(\frac{1}{2^l}) \\ &= \sum_{1}^{\infty} \frac{1}{2^l} \times l \\ &= \text{Longueur moyenne du code} \end{aligned}$$

Ce code a été utilisé pour coder les indices n' sur le codec AMR-WB+ (Section 2.5).

3.3.2 Codage à base de dictionnaire

Dans la plupart des cas, la sortie d'une source consiste à reproduire des séquences redondantes. Par exemple, dans un document texte le mot 'le' se trouve reproduit plusieurs fois comparé au mot 'subterfuge'. L'idée de base du codage à base de dictionnaire est alors d'utiliser un dictionnaire et d'envoyer un indice à la place de la séquence entière. Pour les séquences qui apparaissent moins fréquemment, ils peuvent être codés différemment. L'intérêt d'utiliser un dictionnaire est directement relié à la redondance des symboles de

la source. Donc pour pouvoir choisir les séquences qui feront partie du dictionnaire, il faut connaitre les statistiques de la source. Si ces statistiques sont connues, alors un dictionnaire statique est suffisant. Dans le cas contraire, on peut former le dictionnaire de façon adaptative.

1. **Codage à base de dictionnaire statique :** Quand les statistiques de la source sont connues, le codage par dictionnaire se présente comme une méthode assez intuitive puisque le dictionnaire va contenir les séquences redondantes. Le codage le plus commun basé sur cette idée est le codage Digram [Sayood, 2000]. Le dictionnaire est construit de façon à contenir tout les caractères de la source et les séquences qui sont les plus probables à rencontrer. Le codeur découpe le message à transmettre en séquences et les encode selon la stratégie suivante :

   - *a*) Si la séquence à encoder est présente dans le dictionnaire alors elle est représentée par un seul symbole binaire.

   - *b*) Si la séquence n'est pas dans le dictionnaire, alors elle découpée en sous forme de séquences de longueurs plus petites et on refait l'étape *a*).

   L'inconvénient de ce type de codage est la difficulté de trouver les bonnes séquences qui sont redondantes ainsi que leurs longueurs.

2. **Codage à base de dictionnaire adaptatif :** Quand les statistiques de la source ne sont pas connues, une technique pour construire le dictionnaire est de le faire de façon adaptative. Les deux premiers algorithmes étaient donnés par Lempel-Ziv [Ziv et Lempel, 1978] et [Ziv et Lempel, 1977]. L'idée de base de cet algorithme est de subdiviser l'entrée en plusieurs séquences qui ne se recouvrent pas et de les utiliser pour construire un dictionnaire. Dans ce type de codage, le dictionnaire est tout simplement des séquences des mots déjà codées. Le codeur examine l'entrée en se basant sur une fenêtre glissante. La fenêtre est composée de deux parties, une partie de *recherche* qui contient la partie codée et une partie *look-ahead* qui contient la partie à encoder (figure 3.2). Le codeur pour chaque symbole de la partie codée cherche à trouver sur la deuxième partie le premier bloc qui commence avec ce même symbole. Donc pour chaque symbole on fait correspondre trois paramètres à savoir la position du symbole qui correspond au symbole sur la partie recherche, la longueur du bloc correspondant et finalement le premier caractère de ce bloc. Si on suppose que la longueur du tampon de recherche vaut S, la longueur de la fenêtre est notée W et que la longueur de l'alphabet de la source est A, alors on aura besoin de $\log_2(W \times A \times S)$ bits [Ziv et Lempel, 1978]. Cet algorithme est très simple, et

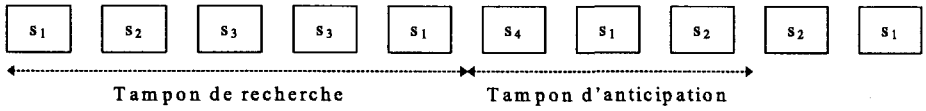


Figure 3.2 Principe du codage Lempel-Ziv

même si les auteurs sont parvenus à montrer son optimalité asymptotique, sa mise en oeuvre peut être améliorée pour donner plusieurs variantes telles que le $LZ77$ ou le $LZ78$.

3.3.3 Codage arithmétique

Dans [Gallager, 1978], la borne supérieure de la longueur moyenne du code de Huffman a été réduite pour devenir égale à $p_{max} + 0.086$ où p_{max} présente la probabilité du symbole le plus souvent produit par la source. Si la source présente plusieurs symboles alors ρ_{max} sera petit ce qui implique l'optimalité du code de Huffman. Dans le cas où la source ne présente pas un grand nombre de symboles (alphabet de petite taille), p_{max} sera grand ce qui va se traduire par une sous-optimalité du code de Huffman. Une solution possible à ce problème est de coder les symboles de la source par bloc, mais cela présente l'inconvénient d'être difficile à mettre en oeuvre pratiquement vu que pour coder une séquence de longueur m, par exemple, il faut disposer de $m!$ probabilités d'où une croissance exponentielle des tailles des dictionnaires.
Le codage arithmétique utilise des *étiquettes* ou symboles particuliers pour séparer les séquences. La virgule correspond à une fraction binaire, qui deviendra plus tard le code binaire de la séquence à encoder. Le processus de codage peut être subdivisé en deux grandes étapes :

1. Pour chaque séquence de la source, une *étiquette* unique est attribuée.
   Dans le but de séparer de façon unique les séquences de la source, on peut utiliser les réels qui sont dans [0, 1). Comme le nombre de réels dans cet intervalle est infini, il est possible de séparer toutes les séquences de la source. Pour parvenir à attribuer pour chaque séquence de la source une *étiquette*, on a besoin de définir une fonction qui définit la correspondance *étiquette* $\rightleftharpoons$ Séquence de la source. Cette fonction est la probabilité cumulative de la source. Les premiers travaux qui ont traité cette méthode de codage se sont toutes arrêtés à la mise en oeuvre pratique vu que les *étiquettes* étaient des valeurs réelles et donc difficiles à encoder efficacement en virgule fixe. Ce ne fut qu'avec les travaux de [Rissanen, 1979] et [Rissanen et Langdon., 1979] que le problème de la précision finie a été surmonté. La procédure de génération des *étiquettes* se base sur la réduction de l'intervalle où résident les *étiquettes* à fur et

à mesure que les symboles de la source sont reçus par le codeur. A l'initialisation l'intervalle [0, 1) est subdivisé en deux sous intervalles dont l'un correspond à la probabilité du premier symbole généré par la source et où sera placé l'*étiquette* pour ce symbole. A la réception du second symbole de la source, le premier intervalle est subdivisé lui même en deux en respectant la probabilité d'apparition du 2^{nd} symbole et ainsi de suite.

Une écriture mathématique de ce code peut être trouvée dans [Sayood, 2000]. Si on suppose qu'une source S délivre des symboles $s_1, \cdots s_m$ alors on peut définir l'étiquette de chaque symbole s_i par :

$$\begin{aligned} T(s_i) &= \sum_{k=1}^{i-1} P(s_k) + \frac{1}{2}P(s_i) \\ &= F_S(i-1) + \frac{1}{2}P(s_i) \end{aligned}$$

Où F_S présente la probabilité cumulative de la source S.

2. L'*étiquette* sera représentée par un mot binaire.
   Les étiquettes $T(S)$ étant des réels dans l'intervalle [0, 1), un code binaire pour les représenter peut être obtenu en prenant la représentation binaire de ces nombres et en les tronquant à la valeur de $\log_2\left(\frac{1}{P(s_i)}\right) + 1$ bits ce qui permet d'avoir un code unique et entièrement décodable. L'étape de décodage consiste à faire le processus exactement inverse tout en garantissant que l'*étiquette* réside toujours dans les bons intervalles. Même si la tâche de codage/décodage est relativement simple, une contrainte limitative de ce type de codage réside dans le fait qu'on doit disposer d'une connaissance exacte des probabilités des symboles de la source. Des versions adaptatives ont été crées pour dépasser cette limitation, mais l'augmentation de la quantité de la mémoire occupée est inévitable en effet les probabilités des symboles sont évaluées à chaque fois ce qui induit implicitement l'utilisation des tables pour compter les occurrences de chaque symbole.

3.3.4 Codage par contexte ou *Context-based coding*

Dans les sections précédentes, on a présenté les techniques de codage dont les performances sont plus évidentes quand les probabilités des symboles sont loin d'être uniformes. i.e Il existe des symboles à forte probabilité d'apparition. Dans le cas où les probabilités d'apparition des symboles sont proches, une méthode intéressante pour les compresser efficacement est de considérer le contexte dans lequel se produisent ces symboles. Shannon

a montré dans [Shannon, 1951], en se basant sur des expériences linguistiques, que le contexte peut être utile pour compresser les messages en biaisant localement les statistiques. Autrement dit, on a l'entropie conditionnelle $H(X/Y)$ est toujours inférieure ou égale à $H(X)$.

$$H(X/Y) \leq H(X) \tag{3.6}$$

En effet l'idée est simple, on demande à une personne (Codeur) de prévoir une lettre à chaque fois. Si la personne trouve la bonne lettre alors elle est informée du fait qu'elle est correcte et elle passe à la lettre suivante, dans le cas où elle se trompe, la lettre lui est donnée et elle passe à la lettre suivante du message. Shannon est arrivé dans [Shannon, 1951] à trouver des bornes limites pour le codage de l'alphabet anglais à savoir 1.3 bits/lettre. La difficulté qui réside dans l'exploitation de ces résultats provient du fait qu'on ne peut développer un modèle mathématique aussi précis et efficace qu'un humain. Si on dispose d'un alphabet de longueur L, alors le contexte d'ordre i sera de taille L^i, si on veut monter en contexte la taille des dictionnaires va augmenter de façon exponentielle se qui limite les performances de ce codage. Plusieurs auteurs ont trouvé différentes façons de contourner ce problème, le plus élégant et le plus simple des algorithmes est le *Prediction with Partial Match* (PPM). Ce code [Cleary et Witten, 1984], au lieu de coder tout les contextes possibles (combinaisons des symboles de la source), se limite à encoder juste ceux qui sont rencontrés au cours du processus de codage. L'algorithme en première approche cherche à utiliser le plus grand contexte possible. Si le symbole à coder n'est pas présent dans le contexte alors la taille de ce dernier est réduite et ainsi de suite jusqu'à ce qu'on arrive à une des deux conclusions suivantes : Soit le symbole est présent dans le contexte, soit non. Dans ce dernier cas, on utilise $\log_2(M)$ bits pour coder ce symbole où M présente la taille de l'alphabet de la source.

3.4 Conclusion

Dans cette section, on a présenté la méthodologie qu'il faut suivre pour compresser un message. Trois techniques de compression ainsi que différentes transformations ont été détaillées dans le but de présenter différentes stratégies possibles. Le choix d'une stratégie, étant dicté essentiellement par la connaissance ou non des statistiques des symboles la source ainsi que l'uniformité de leurs distributions, il est essentiel dans ce projet, de pouvoir étudier avant tout les statistiques des indices binaires présents dans les trames échangés entre le codeur et le décodeur pour le cas du codec TCX. Une fois cette étape accomplie, les résultats obtenus vont permettre de choisir la bonne direction à prendre pour compresser ces indices.

CHAPITRE 4

Algorithme proposé

Le codec AMR-WB+ est un codec multi-débit destiné à encoder aussi bien les signaux de parole que les signaux de musique. L'étude des statistiques des indices binaires de la quantification implique alors une analyse par rapport aux types de signaux ainsi que par rapport aux différents débits. La première partie de ce chapitre sera consacrée à l'étude des indices n' et I, alors que la deuxième partie aura pour rôle de présenter l'algorithme de compression proposé.

4.1 Études des statistiques

Les signaux utilisés dans cette partie sont des signaux de parole et de musique échantillonnés à 48 kHz et codés sur 16 bits par échantillon. On va traiter trois débits, à savoir bas (6 kbps), moyen (21 kbps) et haut (36 kbps). Les signaux sont encodés en mode mono, les résultats obtenus peuvent par la suite être extrapolés pour le mode stéréo.

4.1.1 Étude des statistiques de n'

Sur le codec AMR-WB+, les indices n' représentent les étages utilisés pour quantifier les différents vecteurs en 8 dimensions. La figure 4.1 présente les différentes probabilités pour ces indices en fonction du débit pour les signaux de parole et de musique. La valeur maximale de n' étant de 36 [Ragot *et al.*, 2004], la figure représente les statistiques $n' = \{0, 2, \cdots, \leq 5\}$ pour raison de clarté. A partir de ces probabilités, on peut calculer $C_{n'}$ la longueur moyenne du code utilisé pour représenter les différentes valeurs de n' ainsi que l'entropie $H_{n'}$. Cette longueur moyenne peut être calculée par l'équation (4.1) puisqu'un code unaire est utilisé pour encoder ces indices.

$$C_{n'} = P_{\{n'=0\}} + \sum_{n'=2}^{36} P_{\{n'\}} \times n' \tag{4.1}$$

La redondance du code peut être évaluée par le rapport $R_{n'} = \dfrac{H_{n'}}{C_{n'}}$. Cette quantité décrit de combien un code est loin de l'entropie définie comme borne inférieure pour la compression. Plus ce code est proche de zéro plus ce code est loin de l'entropie et donc non efficace.

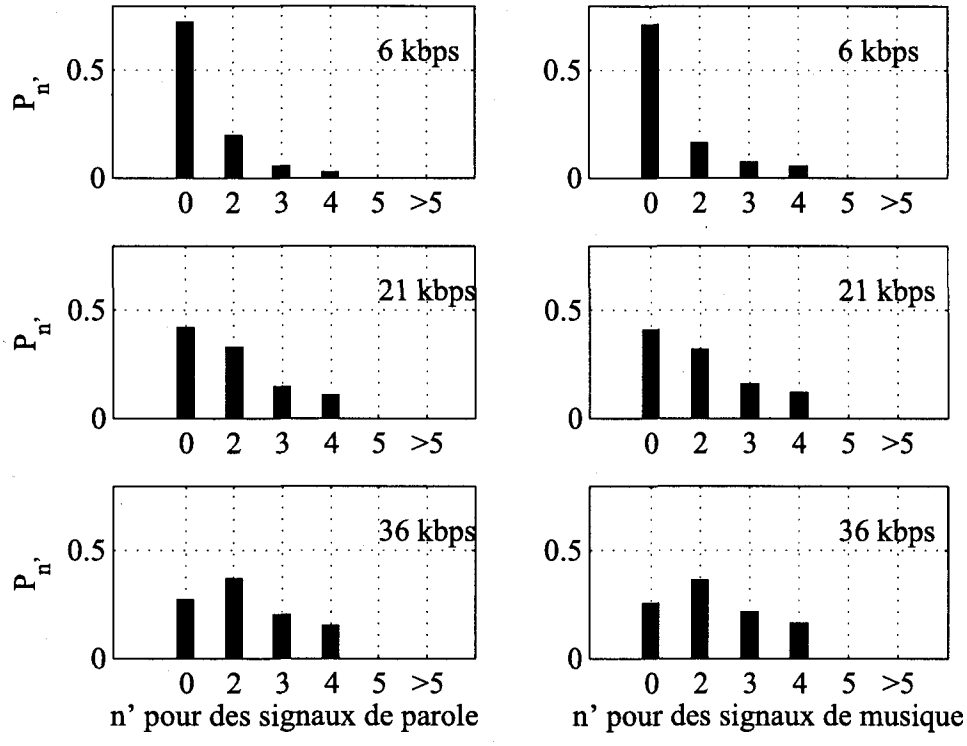


Figure 4.1 Probabilités des indices n' en fonction du débit pour des signaux de parole et musique

Un code optimal a un rapport égal à un. Bien sur, il ne peut exister des codes avec un rapport de redondance supérieur à un sans violer une des hypothèses citées dans la section 3.3.1. Le tableau suivant résume ces différentes quantités en fonction du débit d'encodage et en fonction du type des signaux. En se basant sur le rapport de redondance

Tableau 4.1 Entropie et longueur moyenne du code pour n'

Signal \ Débit		Bas	Moyen	Haut
Parole	$H_{n'}$	1.03	1.58	1.73
	$C_{n'}$	1.32	1.76	2.02
	$R_{n'}$	0.78	0.90	0.85
Musique	$H_{n'}$	1.17	1.71	1.80
	$C_{n'}$	1.38	1.85	2.09
	$R_{n'}$	0.85	0.93	0.86

$R_{n'}$, on peut conclure que le code unaire utilisé pour représenter n' n'est pas optimal. Avant de développer un modèle de compression à partir des statistiques de ces indices, on peut se demander si la disposition de ces indices par rapport aux fréquences peut être

exploitée ? (La i^{eme} position sur la sous-trame représente la i^{eme} sous bande fréquentielle de la TCX). La nature particulière des signaux traités influencerait-elle sur les probabilités des valeurs de n' par rapport à la fréquence ? Les deux graphiques 4.2 et 4.3 représentent la répartition des indices n' en fonction des sous bandes pour les trois débits étudiés.

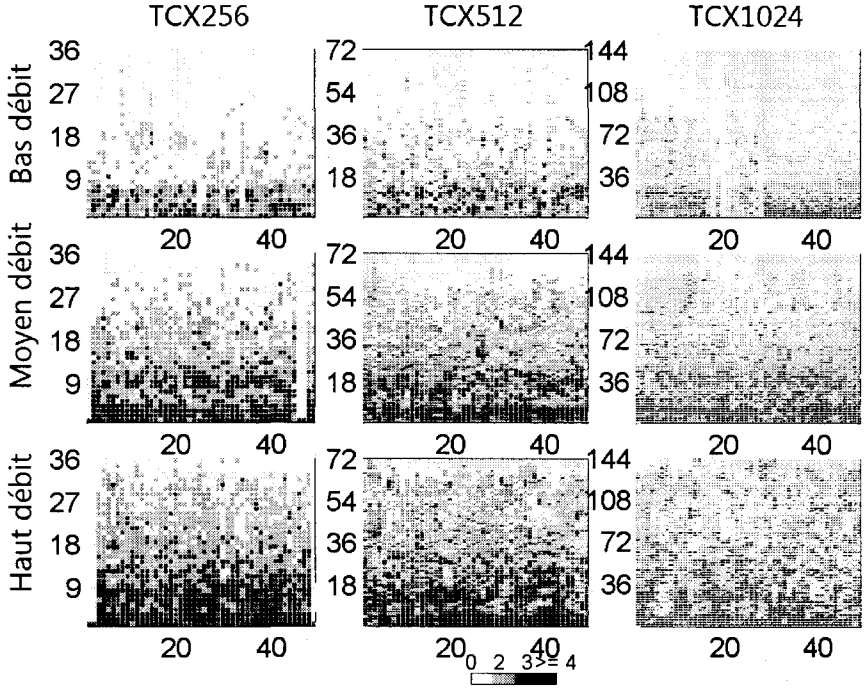


Figure 4.2 Valeur des indices n' en fonction des fréquences pour des signaux de parole

La figure 4.2 montre que pour les signaux de parole, énergétiquement localisés en basses fréquences, la probabilité d'avoir une valeur $n' = 0$ augmente avec la position. Ceci peut être très utile pour la technique basée sur le contexte où l'estimation de la probabilité est faite en fonction des indices passés. Il est aussi à noter qu'il y a des plages où des indices identiques se succèdent. Pour les signaux de parole, c'est plutôt vers les basses fréquences alors que pour les signaux de musique ce sont des zones intermittentes représentant des creux fréquentiels. Cela est surtout visible à bas débit où la priorité d'encodage est donnée aux pics les plus prononcés. Reste à noter aussi qu'une conclusion par rapport aux débits ne peut être faite puisque ce ne sont pas forcément les mêmes trames qui sont traitées par le module TCX vu que le module hybride décide quel module utiliser.

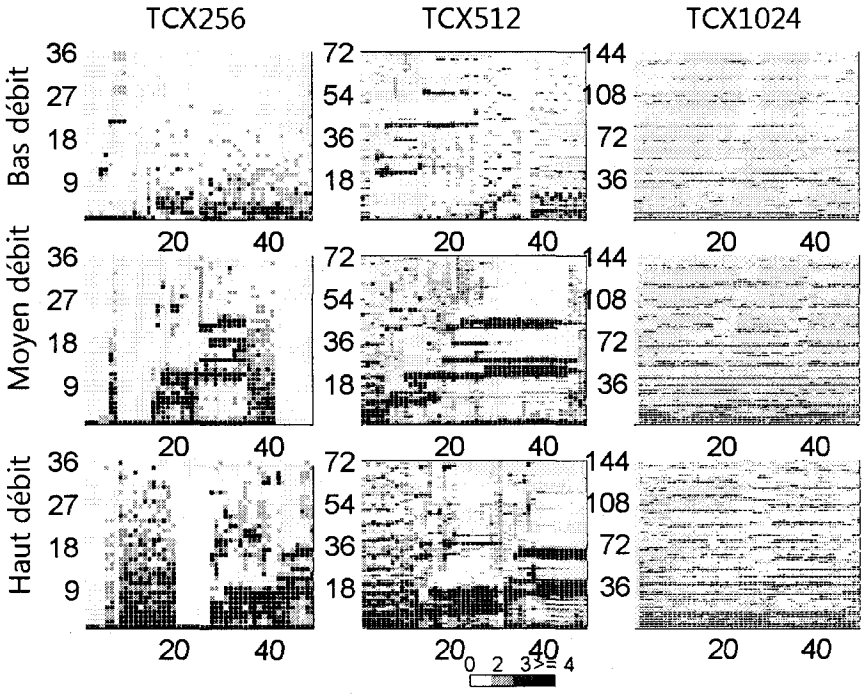


Figure 4.3 Valeur des indices n' en fonction des fréquences pour des signaux de musique

4.1.2 Étude des statistiques de I

Les indices I sont les indices générés à partir de la quantification algébrique encastrée. Chaque indice, représentant un vecteur en dimension 8, est codé selon l'étage n utilisé. Les figures 4.4 et 4.5 représentent les probabilités des indices en fonction du débit pour les signaux de parole et musique. Sachant que les différents QV (Q_2, Q_3, Q_4) sont utilisés

Tableau 4.2 Entropie et longueur moyenne du code pour I pour les différents QV

Signal \ Débit		Bas			Moyen			Haut		
		Q_2	Q_3	Q_4	Q_2	Q_3	Q_4	Q_2	Q_3	Q_4
Parole	C_I	8	12	16	8	12	16	8	12	16
	H_I	7.87	11.31	13.38	7.89	11.48	14.73	7.90	11.53	14.89
	R_I	0.98	0.94	0.84	0.99	0.96	0.92	0.99	0.96	0.93
Musique	H_I	7.74	11.36	14.14	7.88	11.52	14.66	7.91	11.57	14.83
	R_I	0.97	0.95	0.88	0.99	0.96	0.92	0.99	0.96	0.93

avec différentes probabilités, on peut alors calculer aussi bien la longueur moyenne du code

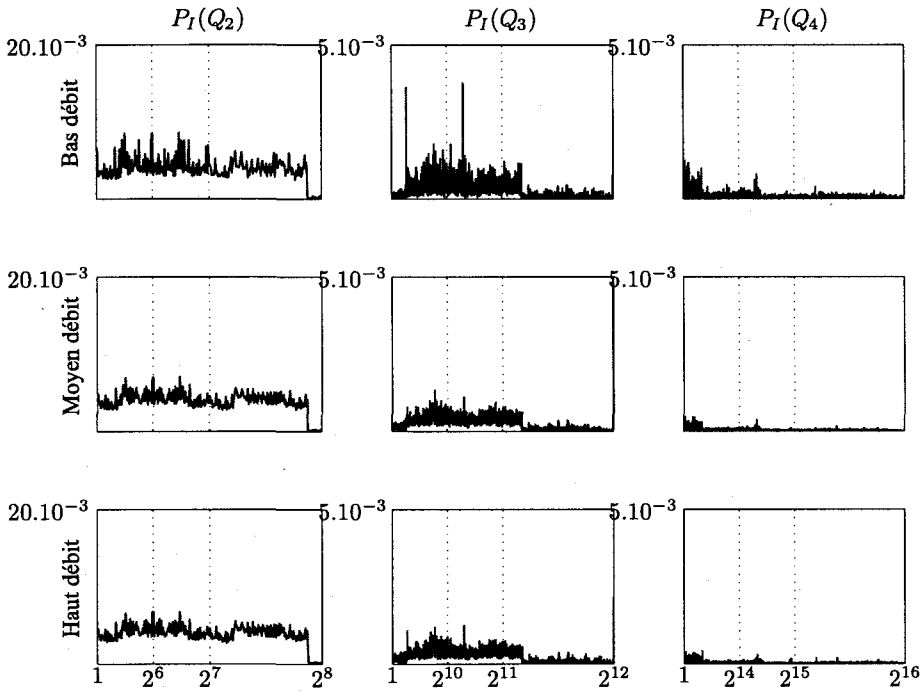


Figure 4.4 Probabilités des indices I pour les signaux de parole

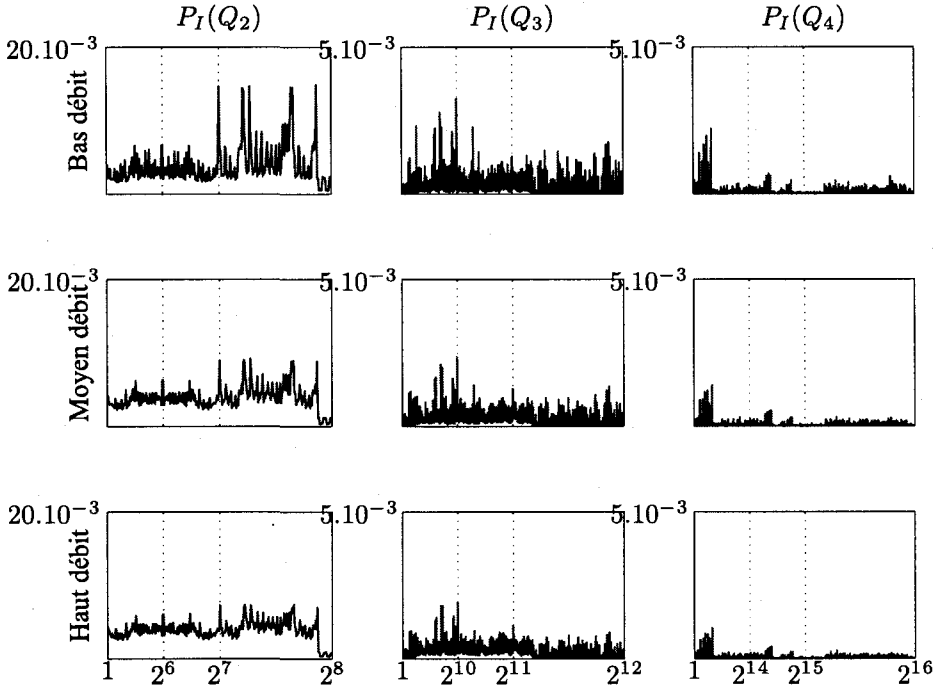


Figure 4.5 Probabilités des indices I pour les signaux de musique

utilisé par le codec AMR-WB+ pour encoder les indices I ainsi que leurs entropies.

$$C_I = \sum_{n=2}^{4} P_n \times 4 \times n \tag{4.2}$$

L'entropie sera donnée par :

$$H_I = \sum_{n=2}^{4} P_n \times H_I \tag{4.3}$$

Le tableau 4.3 résume ces nouvelles quantités pour les trois débits étudiés pour les signaux de parole et musique. A partir des deux tableaux 4.3 et 4.1 on peut calculer la longueur

Tableau 4.3 Entropie et longueur moyenne du code pour I

Signal \ Débit		Bas	Moyen	Haut
Parole	C_I	9.54	10.45	10.80
	H_I	9.07	10.03	10.39
	R_I	0.95	0.96	0.96
Musique	C_I	10.45	10.68	10.92
	H_I	9.80	10.22	10.49
	R_I	0.94	0.96	0.96

moyenne du code utilisé pour encoder les vecteurs de R_8 sans extension de Voronoï. Cette nouvelle quantité est donnée par l'équation (4.4).

$$C_I = 0 \times P_{\{n=0\}} + \sum_{n=2}^{4} P_n \times 5 \times n \tag{4.4}$$

Tableau 4.4 Entropie et longueur moyenne du code pour I et n'

Signal \ Débit		Bas	Moyen	Haut
Parole	C_I	3.98	7.85	9.89
	H_I	3.56	7.42	9.30
	R_I	0.89	0.95	0.94
Musique	C_I	4.32	8.10	10.14
	H_I	3.85	7.64	9.53
	R_I	0.89	0.94	0.94

4.1.3 Étude des statistiques de l'extension de Voronoï

Les indices de Voronoï k sont des vecteurs de $\mathbb{N}^8$ dont chaque composante prend r bits r étant l'ordre de l'extension (Section 2.4.2). Le tableau 4.5 représente l'entropie des indices de Voronoï calculée par l'équation (4.5).

$$H_k = \sum_{1}^{16} p(r) H_{\{k=r\}} \tag{4.5}$$

Le rapport de redondance étant très proche de 1 le code uniforme utilisé pour représenter

Tableau 4.5 Entropie et longueur moyenne du code pour k indices de Voronoï

Signal \ Débit		Bas	Moyen	Haut
Parole	$P(n>4)$	0.01	0.06	0.11
	C_k	1.15	1.31	1.46
	H_k	1.14	1.25	1.36
	R_k	0.99	0.96	0.93
Musique	$P(n>4)$	0.03	0.10	0.13
	C_k	1.16	1.40	1.49
	H_k	1.15	1.34	1.41
	R_k	0.99	0.96	0.94

ces indices est globalement efficace à bas débit. Cependant à haut débit, les probabilités de ces indices deviennent non uniformes ce qui témoigne de la non-optimalité du code uniforme. Même si ces indices ne sont sollicités qu'avec des probabilités proches de zéro, leurs contributions sont assez importantes au débit total. En effet chaque indice prend $8 \times r$ bits où r est l'ordre de l'extension de Voronoï.

4.2 Algorithme proposé

On a choisi pour l'implémentation de l'algorithme de compression la version entière du codage arithmétique [Sayood, 2000]. Les deux avantages de cette version entière sont :

- La possibilité de coder des séquences de longueurs infinies.
- Ce code est incrémental. On n'a pas besoin d'attendre la fin de la séquence pour commencer à décoder.

- Le code est physiquement absent sur le compresseur ainsi que sur le décompresseur. Seules les tables de probabilités sont nécessaires pour encoder les différents symboles de la source.

Le schéma fonctionnel du compresseur proposé est présenté par la figure 4.6 où les variables i, s et s_3 sont des paramètres du codage arithmétique et qui seront développés dans la section suivante. Avant de détailler les différents blocs, on va présenter le fonctionnement du codage arithmétique en version entière.

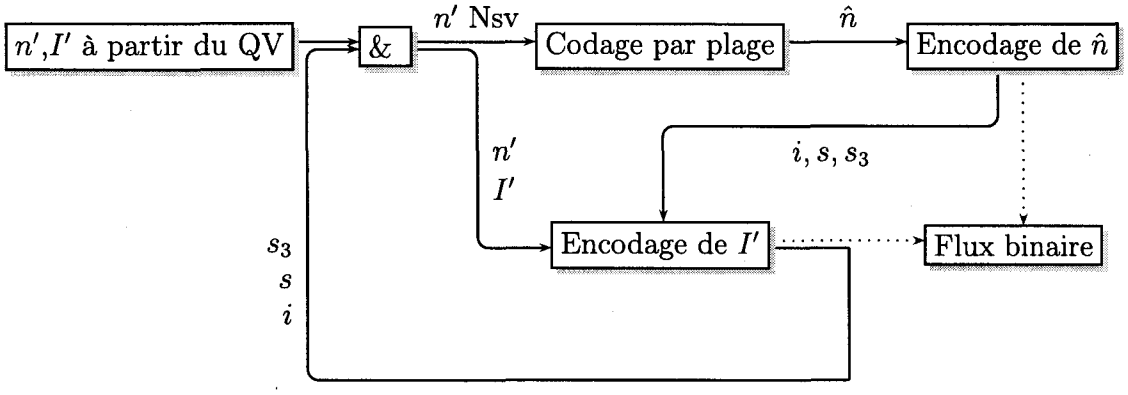


Figure 4.6 Diagramme fonctionnel du compresseur proposé

La figure 4.6 représente le schéma fonctionnel de l'encodeur de la solution proposée. Les indices reçus de l'étage de la quantification vectorielle sont traités par bloc de longueur Nsv (La valeur Nsv est déterminée selon le mode TCX choisi). Les indices n' sont tout d'abord transformés par le codage par plage (section 4.2.3) pour donner une nouvelle séquence $\hat{n}$ dont le dernier élément est toujours un zéro. Cette séquence est encodée en utilisant le codage en version arithmétique incrémentale (section 4.2.2) et le code binaire qui lui est associé est écrit sur le flux binaire. Une fois le dernier élément de $\hat{n}$ encodé (de valeur zéro), ce sont les indices I' qui vont être encodés. Les variables i, s et s_3 issues de l'encodage de $\hat{n}$ sont conservées et utilisées pour encoder I' conjointement avec les indices n' qui jouent cette fois ci le rôle de contexte et permettent de choisir la bonne table de probabilités cumulatives pour encoder I'. Ayant encodé tous les indices de la sous-trame en cours les valeurs des variables i ,s et s_3 sont conservées pour la sous-trame suivante.

La figure 4.7 présente la partie décodage de la solution proposée. La séquence $\hat{n}$ est décodée en premier en utilisant le flux binaire reçu et la valeur de la variable Nsv. Une fois la valeur zéro rencontrée, la séquence $\hat{n}$ est transformée en utilisant l'algorithme donné dans la section 4.2.3 pour donner n'. Les valeurs n' retrouvées sont utilisées conjointement avec

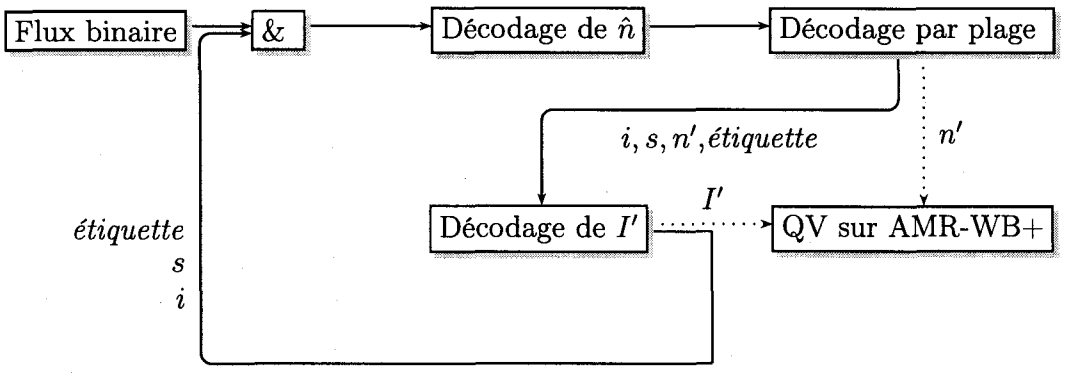


Figure 4.7 Diagramme fonctionnel du décompresseur proposé

le flux binaire et Nsv pour décoder les valeurs de I'. Une fois cette étape accomplie, les valeurs des indices n' et I' sont envoyés à l'étage de quantification vectorielle encastrée pour décoder ces différents indices. Comme sur le compresseur proposé, les valeurs i, s et celle de l'*étiquette* sont conservées pour la sous-trame suivante. Les différents algorithmes et variables sont donnés dans les sections suivantes.

Dans la section 3.3.3 on a présenté le codage arithmétique en version flottante. Cette dernière version souffre d'un inconvénient majeur dû à la nature flottante de l'implémentation. En effet plus la séquence à encoder est longue plus il faut de bits pour la représenter, ce qui implique l'utilisation de réels codés sur un nombre « infini » de bits. Une idée qui permet d'éviter cette saturation consiste à utiliser des « redimensionnements » chaque fois que le code a tendance à saturer. Les sections suivantes ont pour but, de présenter les deux versions du codage arithmétique ainsi que les redimensionnements utilisés pour éviter la saturation, présentant ainsi une justification quant au choix du codage arithmétique en version entière.

4.2.1 Codage arithmétique en version flottante et problème de saturation

L'exemple suivant permet de détailler le fonctionnement du codage arithmétique en version flottante. Soit l'alphabet $\mathbb{X} = \{a_1, a_2, a_3\}$ de taille $N = 3$ avec les probabilités suivantes : $p(a_1) = 0.6$, $p(a_2) = 0.3$ et $p(a_3) = 0.1$. Utilisant les probabilités cumulatives on obtient $F_X(a_1) = 0.6$, $F_X(a_2) = 0.9$ et $F_X(a_3) = 1$. On commence par subdiviser l'intervalle unité

en sous intervalles $\big[F_X(a_{i-1}), F_X(a_i)\big)$[1] pour $i = 1, \cdots, 3$. Pour chaque symbole a_i on associe l'intervalle $\big[F_X(a_{i-1}), F_X(a_i)\big)$. L'apparition du premier symbole dans la séquence à encoder réduit l'intervalle contenant l'étiquette à un de ces sous intervalles. Supposons que le premier symbole est a_k alors l'intervalle contenant l'étiquette est alors $[F_X(a_{k-1}), F_X(a_k))$. Ce dernier intervalle est alors partitionné de la même façon que l'intervalle originel $[0, 1)$ i.e pour le symbole a_j :

$$\Big[F_X(a_{k-1}) + \frac{F_X(a_{j-1})}{F_X(a_k) - F_X(a_{k-1})}, F_X(a_{k-1}) + \frac{F_X(a_j)}{F_X(a_k) - F_X(a_{k-1})}\Big) \tag{4.6}$$

Chaque nouveau symbole de la source réduit l'intervalle contenant l'étiquette à un intervalle qui est lui-même subdivisé suivant l'équation (4.6). Supposons que la séquence à encoder est $\mathbb{S} = a_2, a_2, a_3$. La figure 4.8 présente l'évolution des différents intervalles. Pour

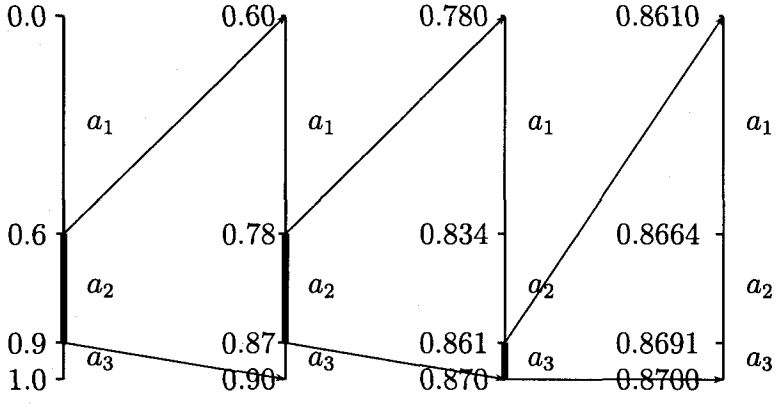


Figure 4.8 Génération de l'étiquette en version flottante

représenter alors la séquence $\mathbb{S}$, il suffit de choisir une étiquette qui réside dans l'intervalle $[0.861, 0.870)$ par exemple la valeur 0.861. À noter qu'avec chaque symbole à encoder l'intervalle où réside l'étiquette se trouve de plus en plus réduit. Avant d'expliquer comment représenter sous forme binaire l'étiquette, on commencera par représenter comment décoder la valeur de l'étiquette.

Pour décoder la valeur de l'étiquette, il suffit de mimer ce que fait le codeur. Le processus de décodage consiste à trouver le sous-intervalle sur $[0, 1)$ noté $[i^0, s^0)$ et qui contient la valeur de l'étiquette. Pour la valeur de l'étiquette valant 0.861, le symbole est a_2 puisque $0.861 \in [0.6, 0.9)$. Après avoir décodé le premier symbole, l'intervalle où réside l'étiquette est mis à jour de la même façon que celle utilisée par le codeur en utilisant l'équation (4.7)

[1] $x \in [a, b) \rightleftharpoons a \leq x < b$

où a_i représente le symbole décodé et k sa position dans toute la séquence.

$$i^k = \frac{F_X(a_{i-1}) - i^k}{s^{k-1} - i^{k-1}} \qquad s^k = \frac{F_X(a_i) - i^k}{s^{k-1} - i^{k-1}} \tag{4.7}$$

L'algorithme de décodage est récursif et peut être représenté par la figure 4.1. On a montré

Algorithme 4.1 : Algorithme de décodage arithmétique

Entrées : étiquette $\in [0, 1[$ et $F_X \in \mathbb{R}^{N+1}$.
Sorties : $\hat{\mathbb{S}}$: Vecteur de $\mathbb{A}$.
Données : $i^0 = 0$; $s^0 = 1$
pour *k=0 :longueur de* $\mathbb{S}$ **faire**
  Calculer $t^* = \frac{\text{étiquette} - i^{k-1}}{s^{k-1} - i^{k-1}}$;
  Trouver la valeur de a_k pour laquelle $F_X(a_{i-1}) \leq t^* < F_X(a_i)$;
  Mise à jour de i^k et s^k suivant l'équation (4.7);
fin

dans cette partie comment le codage arithmétique à partir des tables de probabilités cumulatives permet de représenter de façon unique chaque séquence possible produite par la source. Cependant, deux problèmes restent non résolus. Le premier concerne la façon avec laquelle on peut représenter de façon unique et efficace l'étiquette sous forme binaire. Le second problème concerne la réduction de l'intervalle où réside l'étiquette à chaque apparition d'un nouveau symbole de la source. Si pour l'exemple la séquence à encoder est $\mathbb{S} = \{a_2, a_2, a_3, a_1, a_2, a_3, a_2, a_2, a_3, a_1, a_3, a_2, a_2, a_2, a_3, a_1, a_2, a_2, a_2, a_2, a_3, a_2, a_1, a_3, a_3\}$, on finira avec les intervalles dont les bornes sont données par la figure 4.9. La version entière

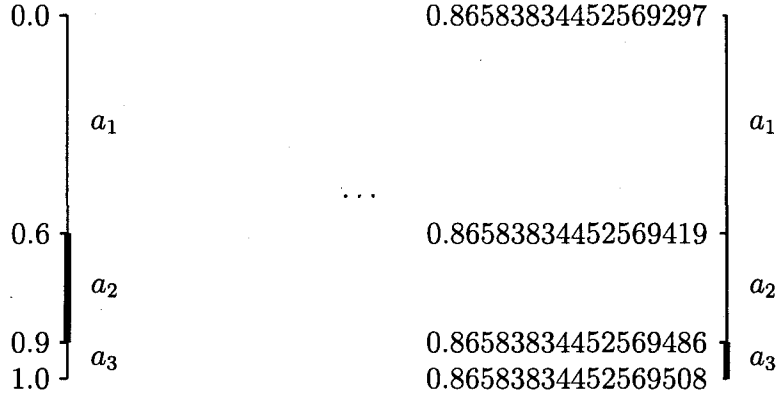


Figure 4.9 Génération de l'étiquette en version flottante et problème de saturation

du codage arithmétique permet de répondre aux deux problèmes cités plus haut.

4.2.2 Codage arithmétique en version entière

L'idée consiste à utiliser un nombre fixe de bits noté m pour représenter l'intervalle $[0, 1)$ avec 2^m mots binaires. Les valeurs importantes de l'intervalle $[0, 1)$ sont représentées par :

$$0 \rightarrow \underbrace{00\cdots0}_{m\ fois} \qquad 1/2 \rightarrow 1\underbrace{00\cdots0}_{m-1\ fois} \qquad 1 \rightarrow \underbrace{11\cdots1}_{m\ fois}$$

L'équation de la mise à jour des intervalles reste la même par contre les tables de probabilités cumulatives sont à changer puisqu'on va utiliser l'arithmétique entière.
Soit n_j le nombre d'occurrence du symbole a_j sur une séquence de longueur Somme_Totale. On peut alors estimer F_X par :

$$F_X(a_j) = \frac{\sum_1^j n_i}{\text{Somme\_Totale}} \tag{4.8}$$

Si on définit

$$\text{Somme\_Cum}(a_j) = \sum_1^j n_i \tag{4.9}$$

on peut changer l'équation (4.7) par les équations (4.10) et (4.11).

$$i^k = i^{k-1} + \left\lfloor \frac{\left(s^{k-1} - i^{k-1} - 1\right) \times \text{Somme\_Cum}(a_k - 1)}{\text{Somme\_Totale}} \right\rfloor \tag{4.10}$$

$$s^k = i^{k-1} + \left\lfloor \frac{\left(s^{k-1} - i^{k-1} - 1\right) \times \text{Somme\_Cum}(a_k)}{\text{Somme\_Totale}} \right\rfloor - 1 \tag{4.11}$$

où a_n est le n^{ieme} symbole à encoder, $\lfloor x \rfloor$ est le plus grand entier inférieur ou égal à x et l'ajout ou la soustraction de 1 permet de prendre en charge l'effet de l'arithmétique entière[2].

[2] i : pour borne inférieure, s : pour borne supérieure

Codage arithmétique en version entière avec redimensionnement [Sayood, 2000]

Dans l'exemple donné par la figure 4.9 on a montré que lorsque la séquence devient de plus en plus longue, les intervalles deviennent de plus en plus étroits et donc nécessitent plus de bits à chaque fois. Par exemple, la séquence donnée dans l'exemple de la figure 4.9 n'est pas pratiquement encodable puisqu'il faut plus que 16 bits pour la représenter. Pour éviter ce problème, on a besoin de redimensionner l'intervalle à chaque fois où il y a un danger de perte de précision.
A chaque fois que l'intervalle devient de plus en plus étroit trois cas de figure sont possibles :

1. L'intervalle est complètement inclus dans $[0, 1/2)$.

2. L'intervalle est complètement inclus dans $[1/2, 1)$.

3. L'intervalle chevauche le point milieu de l'intervalle $[0, 1)$.

Si c'est le premier cas qui se produit, alors le bit le plus signifiant (*Most Significant Bit*, MSB) vaut 0. Si c'est le deuxième cas qui se produit alors le MSB vaut 1. Donc si l'intervalle est inclus dans $[0, 1/2)$ ou $[1/2, 1)$ alors le MSB est complètement déterminé et donc peut être envoyé au décodeur sans attendre la fin de la séquence. Puisque l'étiquette est finalement confinée dans soit la moitié inférieure de l'intervalle $[0, 1)$ soit la moitié supérieure, l'autre moitié peut être alors ignorée. Cela peut se faire en utilisant ces deux redimensionnements :

$$E_1 : [0, 0.5) \rightarrow [0, 1); \qquad E_1(x) = 2x \tag{4.12}$$

$$E_2 : [0.5, 1) \rightarrow [0, 1); \qquad E_2(x) = 2(x - 0.5) \tag{4.13}$$

Si c'est le cas 3 se produit c'est-à-dire que l'intervalle chevauche la moitié de $[0, 1)$ alors si l'intervalle est inclus dans $[0.25, 0.75)$ alors il a tendance à converger vers le point $1/2$. Pour éviter cela on utilise un troisième redimensionnement E_3 :

$$E_3 : [0.25, 0.75) \rightarrow [0, 1); \qquad E_3(x) = 2(x - 0.25) \tag{4.14}$$

Un 1 a été utilisé pour représenter E_1 et un 0 pour représenter E_2 comment représenter E_3 ? Si le 3^{ieme} cas se produit, il suffit de garder cela en mémoire, si un E_1 est à utiliser après alors après avoir envoyé un 1 on transmet un 0 pour indiquer un E_2 au décodeur. Inversement si c'est un E_2 à utiliser après alors un 1 est envoyé. En effet, on peut montrer

[Bodden *et al.*, May 25, 2007] que :

$$E_1 o E_3^k = E_2^k o E_1 \qquad E_2 o E_3^k = E_1^k o E_2 \tag{4.15}$$

En version entière ces différents redimensionnements sont traduits par des opérations élémentaires de décalage binaire. L'algorithme d'encodage est représenté par la figure 4.2.

Algorithme 4.2 : Algorithme du codeur arithmétique en version entière

Entrées : $\mathbb{S}$: Vecteur de $\mathbb{A}$.
Sorties : Code binaire.
Données : Somme_Cum $\in \mathbb{N}^{N+1}$, $m \in \mathbb{N}$
$i = 0$ et $s = 2^m$;
pour *k=0 :longueur de* $\mathbb{S}$ **faire**
  Obtenir le symbole a_k;
  $i = i + \left\lfloor \frac{(s-i-1) \times \text{Somme\_Cum}(a_k - 1)}{\text{Somme\_Totale}} \right\rfloor$;
  $s = i + \left\lfloor \frac{(s-i-1) \times \text{Somme\_Cum}(a_k)}{\text{Somme\_Totale}} \right\rfloor - 1$;
  tant que *Le MSB de i et s sont égaux à b ou si la condition de* E_3 *est vraie* **faire**
    si *Le MSB de i et s sont égaux à b* **alors**
      Envoyer b;
      Décaler i à gauche et insérer un 0 dans le LSB;
      Décaler s à gauche et insérer un 1 dans le LSB;
      tant que $s_3 > 0$ **faire**
        Envoyer le complément de b;
        $s_3 = s_3 - 1$;
      fin
    fin
    si *La condition de* E_3 *est vraie* **alors**
      Décaler i à gauche et insérer un 0 dans le LSB;
      Décaler s à gauche et insérer un 1 dans le LSB;
      Complémenter le nouveau MSB de i et de s;
      $s_3 = s_3 + 1$;
    fin
  fin
fin

L'algorithme de décodage est assez simple et ne fait que mimer ce que fait l'encodeur et est donné par la figure 4.3.

Algorithme 4.3 : Algorithme du décodeur arithmétique en version entière

Entrées : Code bianire
Sorties : $\mathbb{S}$: Vecteur de $\mathbb{A}$.
Données : Somme_Cum $\in \mathbb{N}^{N+1}$, $m \in \mathbb{N}$
$i = 0$ et $s = 2^m$;
Lire les m premiers bits dans l'étiquette t;
pour *k=0 :longueur de* $\mathbb{S}$ **faire**
  tant que $\left(\left\lfloor \frac{(t-i-1) \times Somme\_Totale - 1}{s-i+1} \right\rfloor > Somme\_Cum(k)\right)$ **faire**
    k=k+1;
  fin
  Décoder le symbole a;
  $i = i + \left\lfloor \frac{(s-i-1) \times \text{Somme\_Cum}(a-1)}{\text{Somme\_Totale}} \right\rfloor$;
  $s = i + \left\lfloor \frac{(s-i-1) \times \text{Somme\_Cum}(a)}{\text{Somme\_Totale}} \right\rfloor - 1$;
  tant que *Le MSB de i et s sont égaux à b ou si la condition de* E_3 *est vraie* **faire**
    si *Le MSB de i et s sont égaux à b* **alors**
      Décaler i à gauche et insérer un 0 dans le LSB;
      Décaler s à gauche et insérer un 1 dans le LSB;
      Décaler t à gauche et lire le bit suivant dans le LSB;
    fin
    si *La condition de* E_3 *est vraie* **alors**
      Décaler i à gauche et insérer un 0 dans le LSB;
      Décaler s à gauche et insérer un 1 dans le LSB;
      Décaler t à gauche et lire le bit suivant dans le LSB;
      Complémenter le nouveau MSB de i, de s et de t;
    fin
  fin
fin

4.2.3 Compression de n'

Se référant à la figure 4.3, vu la présence des valeurs successives identiques pour la valeur de n', un encodage par plages est judicieux. Cependant, comme la longueur de la sous-trame est connue et donnée par la variable Nsv, des modifications sur le codage par plage sont apportées. La valeur $n' = 1$ n'est jamais utilisée vu que Q_1 est écarté pour moindre performance. La valeur de $n' = 0$ peut être alors changée par $n' = 1$. Pour tenir compte de la présence des variables identiques et successives qui sont à la fin de la sous-trame pour des signaux de parole et en milieu de la sous-trame pour des signaux de musique, on a choisit de multiplexer la valeur de n' par son nombre d'occurrences N_{occu} successives

suivant l'équation :

$$Sym = n' + (N_{occu} - 1) * \alpha \tag{4.16}$$

où α est un paramètre qui permet de multiplexer la valeur de n' avec son nombre d'occurrence. Puisque $\max(n') = 36$, le paramètre α doit avoir la valeur 36. La valeur de la longueur de la sous-trame étant connue, la dernière valeur d'occurrence n'est pas encodée et la valeur valant 0 est encodée pour indiquer la fin de la sous-trame. L'algorithme utilisé pour transformer n' est donné par la figure 4.4. Cet algorithme proposé prend

Algorithme 4.4 : Algorithme du codage par plage

Entrées : $n' \in \mathbb{N}^{Nsv}$, Nsv : Longueur de n'
Sorties : $\hat{n'}$: Vecteur $\in \mathbb{N}$
Données : α et ,$k = 1$
pour *i=2 :Nsv* **faire**
  si $n'(i-1) = n'(i)$ **alors**
    $N_{occu} = N_{occu} + 1$
  sinon
    si $N_{occu} = 1$ **alors**
      $\hat{n'}(k) = n'(i)$;
      $k = k + 1$;
      $N_{occu} = 1$;
    sinon
      $\hat{n'}(k+1) = n'(i) + \alpha \times (N_{occu} - 1)$;
      $k = k + 1$;
      $N_{occu} = 1$
    fin
  fin
fin
$\hat{n'}(k) = n'(i)$;
si N_{occu}*>1* **alors**
  $\hat{n'}(k+1) = 0$;
fin

en compte la présence des valeurs successives de n' qui se trouveraient soit au milieu de la sous-trame soit à la fin. L'exemple suivant explique le fonctionnement de l'algorithme présenté. Supposons pour des raisons de clarté que Nsv=14 et que :

```
n'=[4 4 0 0 0 0 2 3 2 25 0 0 0 0 ]
```

Après transformation :

```
n'=[4 4 1 1 1 1 2 3 2 25 1 1 1 1 ]
```

En utilisant l'algorithme 4.4, on obtient $\hat{n'} =$

Algorithme 4.5 : Algorithme du décodage par plage

Entrées : $\hat{n'}$, Nsv : Longueur de n'
Sorties : n' : Vecteur $\in \mathbb{N}$
Données : α, Nsv et k
tant que $\hat{n'}(i) \geq 1$ **faire**
  si $\hat{n'}(i) == \hat{n'}(i+1)$ **alors**
    $n'(k) = \hat{n'}(i)$;
    $k = k + 1$;
    $i = i + 2$;
  sinon
    $Q = div(\hat{n'}(i), \alpha)$[1] ;
    $R = res(\hat{n'}(i), \alpha)$;
    pour $s = 1 : (\hat{n'}(i) - R)/\alpha$ **faire**
      $\hat{n'}(i) = R - (R == 1)$;
      $k = k + 1$;
    fin
  fin
  $i = i + 1$;
  pour s=k :Nsv **faire**
    $n'(i) = n'(k)$;
  fin
fin

```
[40 109 2 3 2 25 1 0]
```

Après avoir décodé la séquence avec le décodeur arithmétique et en utilisant l'algorithme 4.5 on obtient, $div(\hat{n'}, P) =$

```
[4 1 2 3 2 25 1]
```

et $\dfrac{\hat{n'} - div(\hat{n'}, 36)}{36} =$

```
[1 3 0 0 0 0 0]
```

Ce qui permet de retrouver le nombre d'occurrence de chaque n'. Vu que la longueur de la sous-trame est connue et vaut 14 l'avant-dernier symbole est répété jusqu'à l'obtention d'une séquence de longueur égale à 14. Dans cet exemple, ce symbole correspond à 1 puisque la valeur zéro est utilisée pour indiquer la fin de la séquence au décodeur arithmétique, le nombre d'occurrences donné par $\dfrac{\hat{n'} - div(\hat{n'}, 36)}{36} + 1$ valant 12 on sait que le dernier zéro est à recopier encore 3 fois. Ce qui donne au final la même séquence de départ

[1] Q : quation de la division euclidienne de a par b. R étant le reste de cette division

```
n'=[4 4 0 0 0 0 2 3 2 25 0 0 0 0 ]
```

4.2.4 Compression de I'

Plusieurs scénarios peuvent se produire lors de la compression de I'. Pour parcourir les

Cas possible		Contexte
Il n'y a pas de I' (Cas de $n' = 0$)	$\Rightarrow$	-1
$I' \in Q_2$	$\Rightarrow$	1
$I' \in Q_3$	$\Rightarrow$	2
$I' \in Q_4$	$\Rightarrow$	3
$I' \in Q_x$ avec $n' \geq 4$	$\Rightarrow$	4

différents scénarios possibles, une variable globale *Contexte* est utilisée. Cette variable calculée à partir de n' est utilisée conjointement avec le codeur arithmétique en version entière présenté dans la section 4.2.2. Cette variable permet aussi de sérialiser le flux binaire. Le problème avec le code arithmétique est que la longueur du code binaire n'est pas prévisible à l'avance. Cela implique que deux codes issus de deux tables de probabilités ne peuvent être concaténés et rester pour autant uniquement décodables. Cela est dû essentiellement au redimensionnement E_3 qui n'est encodé que si un E_1 ou E_2 est rencontré plus tard dans la séquence. Pour contourner ce problème une idée consiste à commuter entre les tables de probabilités et de mettre à jour les bornes i, s et la valeur de l'étiquette. Soit $c(n)$ le contexte à l'instant n et soit a le symbole courant à encoder ou à décoder, les équations de mise à jour des bornes inférieures et supérieures sont modifiées pour tenir compte du changement du contexte :

$$i^n_{c(n)} = i^{n-1}_{c(n-1)} + \left\lfloor \frac{\left(s^{n-1}_{c(n-1)} - i^{n-1}_{c(n-1)} - 1\right) \times \text{Somme\_Cum}_{c(n)}(a-1)}{\text{Somme\_Totale}_{c(n)}} \right\rfloor \tag{4.17}$$

$$s^n_{c(n)} = i^{n-1}_{c(n-1)} + \left\lfloor \frac{\left(s^{n-1}_{c(n-1)} - i^{n-1}_{c(n-1)} - 1\right) \times \text{Somme\_Cum}_{c(n)}(a)}{\text{Somme\_Totale}_{c(n)}} \right\rfloor - 1 \tag{4.18}$$

$$\left\lfloor \frac{\left(t^{n-1}_{c(n-1)} - i^{n-1}_{c(n-1)} - 1\right) \times \text{Somme\_Totale}_{c(n)} - 1}{s^{n-1}_{c(n-1)} - i^{n-1}_{c(n-1)} + 1} \right\rfloor > \text{Somme\_Cum}_{c(n)}(n) \tag{4.19}$$

La valeur de $c(n-1)$ indique la valeur du contexte passé. Par exemple 2 pour un indice encodé dans Q_3, si l'indice suivant à encoder est dans Q_4 alors $c(n)$ prend 3 comme valeur. Dans le cas, où cet indice est dans Q_3, alors $c(n) = c(n-1) = 2$, utilisant ainsi la même table des probabilités pour l'encodage arithmétique. Quand le vecteur est à l'extérieur de

Q_4, un indice de Voronoï k d'ordre d'extension r est utilisé pour ramener ce vecteur soit dans Q_3 ou dans Q_4. L'indice de Voronoï k est un vecteur en dimension 8 dont chaque composante prend r bits. La valeur maximale de n' valant 36, la valeur maximale que peut prendre r est égale à 16. La probabilité $p_{\{n'=36\}} \leq 1\%$, allouer une table de probabilité de taille 2^{16} par exemple pour les indices de Voronoï est injustifié. Pour éviter d'occuper la mémoire pour des performances négligeables, on a choisi d'adopter deux stratégies pour compresser les indices de Voronoï. Basé sur les probabilités d'utilisation de ces indices, on a choisi d'utiliser un modèle de probabilité uniforme quand $r \geq 9$. Pour le cas où $r \leq 8$, deux cas de figure sont envisageables, soit k est calculé pour Q_3 ou Q_4. Donc pour chaque $r \leq 8$, deux tables de tailles 2^r sont utilisées, une pour le cas où Q_3 est utilisé l'autre pour le cas où Q_4 est utilisé. Aucune information supplémentaire n'est à encoder puisque n' déjà encodé contient la valeur de n et celle de r. Utiliser un modèle de probabilité uniforme conjointement à un encodage arithmétique en version entière permet d'éviter d'utiliser des tables de probabilité et par la suite d'optimiser l'utilisation de la mémoire. Soit k_i une des 8 composantes de l'indice de Voronoï k d'ordre r. Les équations (4.17), (4.18) et (4.19) sont alors adaptées pour tenir compte du fait que les probabilités sont uniformes :

$$i^n_{c(n)} = i^{n-1}_{c(n-1)} + \left\lfloor \frac{\left(s^{n-1}_{c(n-1)} - i^{n-1}_{c(n-1)} - 1\right) \times k_i}{2^r} \right\rfloor \tag{4.20}$$

$$s^n_{c(n)} = i^{n-1}_{c(n-1)} + \left\lfloor \frac{\left(s^{n-1}_{c(n-1)} - i^{n-1}_{c(n-1)} - 1\right) \times (k_i + 1)}{2^r} \right\rfloor - 1 \tag{4.21}$$

$$\left\lfloor \frac{\left(t^{n-1}_{c(n-1)} - i^{n-1}_{c(n-1)} - 1\right) \times 2^r - 1}{s^{n-1}_{c(n-1)} - i^{n-1}_{c(n-1)} + 1} \right\rfloor > n \tag{4.22}$$

A noter que les termes intervenant dans les dernières équations sont des scalaires et donc on évite d'allouer des tables de probabilités pour $r \geq 9$. A ce stade on a défini, les tables à utiliser pour estimer les probabilités d'ordre 1 des indices I. On peut se demander alors si on peut améliorer cette estimation par exemple on utilisant un modèle de probabilité d'ordre plus élevé. Intuitivement un signal de parole par exemple voisé est redondant, cette redondance se retrouve aussi spectralement. i.e. Il y a une structure particulière sur le spectre (des harmoniques). On peut qualifier cette structure d'euclidienne, puisqu'elle reflète les distances euclidiennes entre les différents points de la transformée de Fourier. Pourquoi n'est-ce pas le cas pour les indices I, qui n'ont aucune structure particulière ? Deux aspects peuvent être mis en cause : la structure du dictionnaire utilisé et la procédure de quantification. Revenons à la procédure de génération des indices en question. L'indice

I n'est en fait que le rang lexicographique d'un vecteur de RE_8, c'est à dire que les indices successifs décrivent des vecteurs qui différent d'un nombre minimal de composantes. Cette distance dite de Hamming, décrit une norme sur RE_8 qui n'est autre qu'un espace vectoriel en dimension finie. Vu qu'en dimension finie, l'équivalence des normes est prouvée ce qui implique la conservation des topologies, ce n'est pas l'ordre lexicographique qui fait que les indices n'ont pas de structure particulière. La réponse se trouve du dans la façon avec laquelle les vecteurs en R_8 sont construits. On rappelle que les coefficients de la transformée de Fourier sont séparés en partie réelle et imaginaire. Prenons un exemple en dimension deux, où on veut quantifier le vecteur $[e^{i\frac{\pi}{4}}\ e^{-i\frac{\pi}{4}}]$, même si ce vecteur a une norme des composantes valant [1 1], séparer les composantes réelles des imaginaires induit des indices sans corrélations. Le fait de quantifier la norme et la phase ensemble fait que l'aspect aléatoire de la phase l'emporte sur l'aspect corrélé de la norme ce qui génère des indices pratiquement uniforme. En effet, en utilisant la même procédure que celle utilisée par le codec TCX, on se trouve à quantifier deux vecteurs $\frac{\sqrt{2}}{2}[1\ 1]$ et $\frac{\sqrt{2}}{2}[1\ -1]$ qui auront des indices très éloignés, voir décorrelés. Vu la nature différente des signaux traités (Parole et musique) ainsi que la variabilité des statistiques en fonction des débits, une version adaptative s'impose. La section suivante présente les modifications apportées à l'algorithme principal ainsi que les différents paramètres à utiliser pour le fonctionnement de l'algorithme proposé.

4.2.5 Taille de la table des probabilités pour le codage par plage

Dans la section 4.2.3 on a présenté l'algorithme utilisé pour transformer n' avant de l'encoder avec l'encodage arithmétique. Ce dernier a besoin d'une table de probabilité qu'on a choisi de construire de façon adaptative dont il faut définir la taille. On rappelle l'équation utilisée pour multiplexer n', le nombre d'occurrence et α paramètre fixé à 36.

$$Sym = n' + (N_{occu} - 1) \times \alpha$$

La taille T_{plage} de la table de probabilité cherchée est égale au maximum de la valeur que peut prendre la variable Sym et ce par rapport à la longueur de la sous-trame (Nsv).

$$\begin{aligned} T_{plage}(\text{Nsv,D}) &= \max_{N_{occu}}(Sym) \\ &= \max_{N_{occu},\text{D}}(n') + \max_{\text{Nsv,D}}\big((N_{occu} - 1) \times \alpha\big) \\ &\leq 36 + \max_{\text{Nsv}}\big((N_{occu} - 1)\big) \times \alpha \end{aligned}$$

Pour pouvoir majorer $T_{plage}(Nsv)$ il suffit de trouver $\max\big((N_{occu}-1)\big)$. Le max de N_{occu} pour une sous-trame de longueur Nsv est Nsv−1. En effet, il faut au moins que le dernier n' soit différent des autres sinon la sous trame sera transformé en $[n'\ 0)$. En utilisant ces donnés en peut trouver alors la valeur de T_{plage} en fonction de Nsv donné par $T_{plage}(\text{Nsv}) = \alpha \times (\text{Nsv}-1)$. La table 4.6 donne les valeurs de T_{plage} pour les différentes valeurs de Nsv. Sachant que la valeur de Nsv est communiquée entre le codeur et le décodeur AMR-WB+, cette variable permet de réduire la taille de T_{plage} selon le mode TCX utilisé permettant ainsi de gagner quelques bits.

Tableau 4.6 Taille de T_{plage} en fonction de Nsv

Nsv	T_{plage}
36	1260
72	2556
144	5148

4.2.6 m nombre de bits pour l'encodage arithmétique

Un paramètre important à dimensionner est m qui représente le nombre de bits pour différencier les points de chaque table de probabilité. Ce paramètre doit aussi permettre de différencier les symboles de la source. Il faut donc choisir un m qui permet de représenter la plus petite différence entre les extrémités des intervalles $[i^{(k)}, s^{(k)})$. Un redimensionnement est toujours utilisé lorsque les intervalles deviennent plus petits. Afin de s'assurer que les extrémités des intervalles restent distinctes, toutes les valeurs dans l'intervalle $[i^{(k)}, s^{(k)})$ doivent être uniquement représentées sans déclencher un redimensionnement. L'intervalle le plus petit, sans déclencher un redimensionnement, est quand $i^{(k)}$ est juste en dessous du point milieu et $s^{(k)}$ est à trois quarts de l'intervalle ou lorsque $s^{(k)}$ est à droite au milieu de l'intervalle et $i^{(k)}$ est juste en dessous d'un quart de l'intervalle. Autrement dit, le plus petit intervalle sans déclencher un changement d'échelle est le quart de la gamme totale disponible de 2^m valeurs. Cette condition est exprimée par l'équation (4.23).

$$2^m \geq 4 \times \text{Somme\_Totale} \tag{4.23}$$

Cette équation décrit la relation qui existe entre les probabilités cumulatives et le paramètre m. Pour pouvoir choisir une valeur efficace pour ce paramètre représentatif de toutes les tables de probabilités, il faut non seulement respecter l'équation (4.23) mais aussi garder une bonne estimation des probabilités. La version entière utilise des entiers pour représenter les fréquences d'apparition des différents symboles de la source. Pour

pouvoir différencier deux symboles de la source il faut que le minimum de distance entre leurs probabilités cumulatives soit égal à 1.

$$\min\left\{\left|\text{Somme\_Cum}(a_i) - \text{Somme\_Cum}(a_j)\right|\right\} = 1 \ \forall i \neq j \tag{4.24}$$

$\min(\hat{p})$ étant fixé à 1 pour une table de probabilité entière, une mesure de l'efficacité de cette estimation est le rapport $\frac{\max \hat{p}}{\min \hat{p}}$ qui par exemple vaut 1 pour une distribution uniforme. La meilleure estimation $\hat{p}$ des probabilités p par des entiers est celle qui satisfait l'équation (4.25).

$$\frac{\max p}{\min p} = \frac{\max \hat{p}}{\min \hat{p}} = \max \hat{p} \tag{4.25}$$

Le cas extrême se produit lorsque toutes les probabilités sont égales sauf pour un symbole dont la probabilité vaut $\min(p)$. Donc pour un alphabet de taille T, et un rapport $\frac{\max p}{\min p}$, la somme totale Somme_Totale se trouve bornée.

$$T \leq \text{Somme\_Totale} \leq T \times \frac{\max \hat{p}}{\min \hat{p}} \tag{4.26}$$

En combinant l'équation (4.26) et (4.23) on trouve une borne inférieure pour m donnée par l'équation (4.27)

$$m \geq \log_2(T \times \frac{\max p}{\min p}) + 2 \tag{4.27}$$

Le nombre des tables les plus importantes à utiliser est de 4 et sont données par le tableau 4.7. À partir de cette table on peut alors choisir la valeur du paramètre m qui vaut $29 + 2 = 31$. On est sûr avec cette valeur de bien estimer les probabilités des différentes tables.

Tableau 4.7 Taille des différentes tables de probabilités

Table	Taille	Rapport $\frac{\max p}{\min p}$	Taille $\times \frac{\max p}{\min p}$
T_{rle}	5148	325057	2^{30}
T_{Q_2}	256	130	2^{15}
T_{Q_3}	4096	2057	2^{23}
T_{Q_4}	65536	248	2^{24}

4.2.7 Adaptabilité de l'algorithme

L'algorithme présenté dans ce projet est adaptatif. Les probabilités des différents symboles (n', I') de la source sont estimées à fur et à mesure. Dans un algorithme de compression

adaptatif, le codeur et le décodeur commencent avec des modèles de probabilités identiques. Les symboles rencontrés par le codeur sont encodés ensuite les tables sont mises à jour, alors que sur le décodeur le symbole est décodé ensuite les tables de probabilités sont mises à jour. La façon la plus simple de mettre en oeuvre cette idée consiste à utiliser un modèle uniforme de probabilités et de compter le nombre d'apparitions de chaque symbole, ensuite de mettre à jour les tables de probabilités. Le problème de la mise à jour des probabilités de la source repose sur la construction des tables de probabilités cumulatives. Cela restreint par exemple la mise à jour à ne pas à se faire fréquemment et donc détériorer l'estimation des probabilités et par conséquent le rapport de redondance du code. La mise à jour des probabilités doit aussi satisfaire l'équation (4.23).
Soit p la table des probabilités utilisée pour encoder les symboles de la source. Notons par p_{locale} la table des probabilités des k derniers symboles rencontrés construite par un simple comptage du nombre d'apparitions des symboles. Mettre à jour le modèle p peut se faire selon l'équation (4.28)

$$p = \alpha_1 \times p + \alpha_2 \times p_{locale} \tag{4.28}$$

où α_1 décrit l'aspect stationnaire de la source et α_2 décrit l'aspect évolutif de la source. Suite à des expériences, on a choisi de prendre $\alpha_1 = \alpha_2 = 1/2$, ce choix est aussi justifié par la stationnarité de la source. Après la mise à jour, c'est la construction de la table des probabilités cumulatives. Une fois cette étape terminée, et quand Somme_Totale dépasse la valeur de 2^{m-2}, un redimensionnement est utilisé suivant l'équation (4.29)

$$p(a_k) = \frac{p(a_k)}{2} + 1 \ \forall k \tag{4.29}$$

L'ajout de 1 sert à assurer que $min(p) = 1$. On a défini les opérations nécessaires à l'adaptabilité de l'algorithme proposé, cependant on n'a pas défini quand la mise à jour des statistiques doit se faire. Deux façons de le faire sont à envisager, soit de façon statique en faisant la mise à jour de façon cyclique après un nombre fixé de symboles, soit de façon adaptative en surveillant la redondance du code de façon continue : si ce dernier descend en dessous d'un certain seuil la mise à jour se fait de façon automatique. Cette méthode bien qu'elle permet d'adapter la mise à jour des statistiques au taux de redondances locaux, souffre d'un inconvénient majeur : on n'a pas de contrôle sur la complexité que peut engendrer cette façon de mise à jour, vu que la mise à jour implique la construction de plusieurs tables de probabilités à chaque fois. On a donc opté pour la première méthode de mise à jour. L'algorithme de mise à jour des probabilités est donné par la figure 4.6.
La mise à jour étant déclenchée par exemple toutes les 3 secondes du signal audio. Après

Algorithme 4.6 : Mise à jour des tables de probabilités

Sorties : Somme_Cum : Tables de probabilité cumulative
Données : p : probabilités des symboles, p_{locale} : probabilités locales des symboles
tant que *Encore des symboles à encoder/décoder* **faire**
  Encoder/décoder a_k en utilisant Somme_Cum ;
  $p_{locale}(a_k) = p_{locale}(a_k) + 1$;
  si *Mise à jour* **alors**
    $p = \frac{p+p_{locale}}{2} + 1$;
    $p_{locale} = 1$;
    si $sum(p) > 2^{m-2}$ **alors**
      $p = p/2 + 1$;
    fin
    Construire la table des probabilités cumulatives Somme_Cum à partir de p;
  fin
fin

une telle durée, on aura recueilli assez d'indices pour représenter efficacement le contenu de la source.

4.3 Conclusion

On a présenté dans ce chapitre l'algorithme dont le rôle est de réduire la longueur moyenne du code binaire utilisé par le codec AMR-WB+ . En se basant sur les statistiques pour différents types de signaux et différents débits, on a développé un algorithme de compression/décompression sans perte qui prend en compte non seulement les probabilités d'apparition des différents indices, mais aussi leurs dispositions fréquentielles. Les différents blocs ainsi que les opérations nécessaires pour l'encodage/ décodage des différents symboles ont été présentés. Une version adaptative a été aussi développée dans le but de prendre en compte la variabilité des statistiques par rapport aux débits et par rapport au type du signal. Le chapitre suivant a pour but d'évaluer les performances de cet algorithme.

CHAPITRE 5

Évaluation des performances

Dans ce chapitre, on va évaluer les performances de l'algorithme proposé. Un algorithme de compression est généralement évalué par rapport à son taux de compression. Étant dédié à un codec spécifique, l'algorithme proposé doit être aussi évalué par rapport à d'autres critères dont les plus importants sont la vitesse de traitement, et l'occupation de la mémoire. Les sections suivantes permettent d'évaluer ces différents critères. D'abord, l'occupation de la mémoire est évaluée vu que la solution proposée utilise des tables de probabilités. Ensuite une évaluation de la complexité en nombre d'opérations élémentaires est effectuée. Finalement, les performances de la solution proposée seront évaluées pour différents types de signaux à différents débits.

5.1 Évaluation de la complexité et l'occupation de la mémoire

La complexité du codec AMR-WB+ est caractérisée par les trois quantités suivantes [Speech, 1998].

- wMOPS (*weighted Millions of Operations Per Second*)
- Taille occupée dans la RAM (*Random Access Memory*)
- Taille occupée dans la ROM (*Read Only Memory*)

où RAM et ROM sont évalués en koctet (1 koctet= 512 mots de 16 bits). Le calcul de wMOPS est fait en utilisant un ensemble d'opérations élémentaires pondérées. Par exemple, une addition de deux mots sur 16 bits est pondérée par un coefficient valant 1, alors qu'une division est pondérée par un facteur de 18. La copie de donnée d'une variable vers une autre est aussi estimée, ainsi que les tests logiques et les tests arithmétiques. Ces opérations ainsi que leurs pondérations peuvent être trouvées dans [Speech, 2005, 1998]. Dans la plupart des applications, comme seul le décodeur est nécessaire sur le cellulaire et que le codage peut être effectué en temps différé, la complexité du codeur n'est pas vraiment importante. Par exemple dans le cas de la diffusion de la radio, des messages multimédias, seule la présence du décodeur (dans notre cas d'étude) AMR-WB+, sur le

terminal est exigée. La comparaison de l'algorithme proposé se fera alors par rapport au décodeur AMR-WB+. Le tableau 5.1 résume l'occupation de la mémoire utilisée par le décodeur AMR-WB+ [Salami *et al.*, 2006]. Le tableau 5.2 donne une mesure de la

Tableau 5.1 Utilisation de la mémoire par le décodeur AMR-WB+

Condition	RAM(koctet)	ROM(koctet)
Mono	20	13

complexité du décodeur AMR-WB+ en terme de wMOPS [3GPP, 2008-12].

Tableau 5.2 Complexité du décodeur AMR-WB+

Condition	wMOPS moyen	wMOPS pour le cas le plus complexe
14 kbps,	8.415	8.792
24 kbps,	10.382	10.981
36 kbps,	12.463	13.184
18 kbps, stéréo	15.996	16.603
24 kbps, stéréo	17.654	18.303
32 kbps, stéreo	20.288	21.081
48 kbps, stéreo	22.960	23.927

5.1.1 Occupation de la mémoire

L'algorithme proposé utilise deux tables pour chaque contexte : une table de probabilité et une table de probabilité cumulatives. On rappelle les tailles des tables utilisées.

Tableau 5.3 Tailles des tables de probabilités utilisées

Tables	Tailles
Codage par plage	5148
Q_2	256
Q_3	4094
Q_4	65536
Indices de Voronoi $r \leq 8$	2^r
Somme	76056

76056 variables encodées sur 16 bits et 76056 variables encodées sur 32 bits pour les probabilités cumulatives représentent une énorme occupation de la mémoire environ 445 koctet comparée à celle qu'occupe le décodeur AMR-WB+. Devant cette occupation de la mémoire, on a décidé de présenter deux solutions une qui utilise la table de probabilité pour Q_4 (Algorithme A_1), une autre solution où le codage de Q_4 se fait de façon uniforme, de la même façon que celle pour les indices de Voronoi quand l'ordre d'extension est

supérieur à 8 (Algorithme A_2) (Section 4.2.4). Le deuxième algorithme A_2 permet de ramener l'occupation de la mémoire à 62 koctet.

5.1.2 Évaluation de la complexité

Les mêmes règles que celles utilisées pour évaluer la complexité du décodeur AMR-WB+ ont été utilisées pour évaluer la complexité de l'algorithme proposé. Le tableau 5.4 résume

Tableau 5.4 Complexité du décodeur proposé

Condition		wMOPS moyen	wMOPS pour le cas le plus complexe
6 kbps(Parole)	A_1	0.124	2.157
6 kbps(Parole)	A_2	0.103	0.689
6 kbps(Musique)	A_1	0.378	2.217
6 kbps(Musique)	A_2	0.360	0.954
21 kbps(Parole)	A_1	0.511	5.119
21 kbps(Parole)	A_2	0.378	2.217
21 kbps(Musique)	A_1	2.782	11.324
21 kbps(Musique)	A_2	2.534	7.457
36 kbps(Parole)	A_1	0.358	3.144
36 kbps(Parole)	A_2	0.344	2.057
36 kbps(Musique)	A_1	12.463	13.184
36 kbps(Musique)	A_2	3.384	11.908

les complexités des deux solutions proposées pour différents débits et types de signaux. Ces résultats justifient la pertinence de la deuxième solution où les indices pour Q_4 sont compressés en utilisant un modèle uniforme. En effet, le wMOPS pour le cas le plus complexe a été réduit environ d'un pourcentage moyen de 40%. Cette quantité peut encore être réduite, en effet il n'est pas obligatoire de faire la mise à jour de toutes les tables de probabilités à la fois. On peut par exemple faire la mise à jour sur plusieurs sous-trames ce qui va encore réduire le wMOPS pour le cas le plus complexe. En comparison avec la table 5.2, l'intégration de la solution proposée implique une minime augmentation de la complexité du codec AMR-WB+ (1% à bas débit) ce qui prouve la faisabilité technique de la solution proposée.

5.2 Taux de compression

On a présenté dans la section 4.1.1 la notion de redondance d'un code dont rappelle la définition :

$$R = \frac{H}{\bar{C}} \tag{5.1}$$

Où $\bar{C}$ est la longueur moyenne du code C utilisé pour représenter les différents symboles de la source dont l'entropie vaut H. Si on note par $\bar{C}_+$ la longueur moyenne du code utilisé par le AMR-WB+ donnée par le tableau 4.4, et par $\bar{C}_a$ la longueur moyenne du code utilisé par l'algorithme proposé on peut définir le taux de compression par :

$$\tau = \frac{\bar{C}_a}{\bar{C}_+} \tag{5.2}$$

Pour un message M de longueur l codé avec b bits, la longueur moyenne $\bar{C}$ du code C peut être estimée par l'équation (5.3).

$$\bar{C} = \lim_{l \to \infty} \frac{b}{l} \tag{5.3}$$

En utilisant l'équation (5.3), l'équation (5.2) peut être remplacée par l'équation (5.4).

$$\tau = \lim_{l \to \infty} \frac{b_a}{b_+} \tag{5.4}$$

Le tableau suivant résume les valeurs de τ tel que décrites dans l'équation (5.4). Les signaux utilisés sont des signaux *wav* échantillonnés avec une fréquence de 48 kHz, de durée 100 min repartie de manière égale entre les deux types de signaux. Les signaux de parole sont des enregistrements de narration de texte en langue française majoritairement. Les signaux de musique sont des morceaux contenant des percussions, chants et instruments à cordes.

Tableau 5.5 Taux de compression τ des indices binaires pour des signaux de parole et musique

Condition \ Débit		Bas	Moyen	Haut
Parole (Mono)	A_1	90.4%	96.4 %	96.6%
Parole (Mono)	A_2	90.7%	97.3 %	97.3%
Musique (Mono)	A_1	95.5%	98.6%	97.0%
Musique (Mono)	A_2	96.6%	99.8%	98.3%

5.3 Conclusion

On a présenté dans ce chapitre les performances de l'algorithme proposé dont le but est de compresser les indices binaires de la TCX utilisés par le codec AMR-WB+. L'algorithme proposé utilise des tables pour représenter et estimer les probabilités des différents indices ce qui explique son occupation de la mémoire. Une alternative a été proposée pour ramener

l'utilisation de la mémoire au même ordre de grandeur que celle du décodeur AMR-WB+ sans pour autant introduire des dégradations des performances, présentant ainsi une solution techniquement exploitable. L'utilisation des ressources computationelles étant très négligeable de l'ordre de 1% par exemple à bas débits pour des signaux de parole, un gain moyen de 10% sur le débit se révèle très utile par exemple pour rehausser la qualité du signal audio ou par exemple pour renforcer la robustesse du codec AMR-WB+ dans un environnement de transmission bruité.

CHAPITRE 6

Conclusion générale

On a présenté dans ce mémoire un algorithme destiné à compresser les indices binaires du quantificateur algébrique encastré utilisé par le codec AMR-WB+ pour coder certaines trames du signal audio. On a commencé par présenter le codec concerné ainsi que les différents blocs intervenant lors de la génération de ces indices. Une revue de l'état de l'art en compression conjointement à des contraintes de faisabilité technique ont permis d'orienter le choix des différents algorithmes proposés. L'étude des statistiques des différents indices a montré que le modèle uniforme utilisé pour les encoder n'est pas optimal et que certaines redondances peuvent être exploitées dans le but de réduire la longueur moyenne du code binaire utilisé. Les indices n', désignant l'étage de quantification utilisé, ont été encodés en utilisant un algorithme de codage par plage alors que les indices I', représentant les vecteurs en dimension 8, ont été encodés en se basant sur un modèle de probabilité sans mémoire. Le choix d'un codage par plage s'explique par la présence d'indices n' successifs de même valeurs. Le modèle de probabilité d'ordre 1 pour la compression des indices I' se justifie par des contraintes de mémoire puisque par exemple pour Q_2 utiliser un modèle de probabilité conditionnelle d'ordre n prendra 2^{8n} cases mémoire.
Le codage arithmétique en version entière a été choisi pour sa vitesse et son occupation minimale de mémoire comparée à d'autres algorithmes d'encodage puisque que le dictionnaire où résident les mots binaires est absent suivant ainsi le même principe que la quantification algébrique encastrée. Les performances de l'algorithme sont telles qu'on peut économiser entre 10% à 4% sur le débit par exemple sur les signaux de parole. Dans le souci de présenter une solution techniquement exploitable et implémentable sur un terminal GSM, la compression pour les indices de Q_4 peut être exclue ce qui se traduit par une économie sur les ressources mémoire et computationelles sans pour autant induire une dégradation perçue sur les taux de compression. Il est à noter aussi que la solution proposée n'introduit pas de délai additionnel puisque l'encodage arithmétique utilisé est incrémental.
Les perspectives proposées peuvent être orientées dans deux directions différentes. La première va dans le sens de l'amélioration des performances de l'algorithme réalisé où il serait par exemple judicieux d'utiliser le codage arithmétique sans multiplication [Fu et Parhi, 1995]. On peut aussi envisager une solution où la construction des tables de probabilités cumulatives est distribués sur plusieurs sous-trames permettant ainsi de réduire le wMOPS pour le cas le plus complexe. Quant à la deuxième direction, elle consiste à étendre l'utili-

sation de l'algorithme et l'adapter à un milieu bruité par exemple en réinitialisant le flux binaire à intervalles réguliers. Le débit gagné par le module de compression proposé peut être utilisé par exemple pour introduire des codes correcteurs d'erreur ou plus simplement de réécrire dans le flux binaire les informations qu'on juge très pertinentes par exemple les bits du mode TCX utilisé ou bien le gain de la trame en question.

LISTE DES RÉFÉRENCES

3GPP (2005-07) *3rd Generation Partnership Project; Audio codec processing functions; Extended Adaptive Multi-Rate - Wideband (AMR-WB+) codec; Transcoding functions (Technical specification).* Rapport technique, Technical Specification Group Service and System Aspects.

3GPP (2007-03) *3rd Generation Partnership Project; Audio codec processing functions; Extended Adaptive Multi-Rate - Wideband (AMR-WB+) codec; Transcoding functions (Release 7) TS 26.290 V7.0.0 (Technical specification).* Rapport technique, Technical Specification Group Service and System Aspects.

3GPP (2008-12) *3rd Generation Partnership Project; Audio codec processing functions; Extended Adaptive Multi-Rate - Wideband (AMR-WB+) codec; Performance characterization of 3GPP audio codecs.* Rapport technique, Technical Specification Group Service and System Aspects.

Berger, T., Jelinek, F. et Wolf, J. (Jan 1972) Permutation codes for sources. *Information Theory, IEEE Transactions on*, volume 18, n° 1, p. 160–169.

Bodden, E., Clasen, M. et Kneis, J. (May 25, 2007) Arithmetic coding revealed a guided tour from theory to praxis.

Burrows, M., Wheeler, D. J., Burrows, M. et Wheeler, D. J. (1994) *A block-sorting lossless data compression algorithm.* Rapport technique, SRC.

Cleary, J. et Witten, I. (Apr 1984) Data compression using adaptive coding and partial string matching. *Communications, IEEE Transactions on [legacy, pre - 1988]*, volume 32, n° 4, p. 396–402.

Conway, J. et Sloane, N. (Mar 1982) Fast quantizing and decoding and algorithms for lattice quantizers and codes. *Information Theory, IEEE Transactions on*, volume 28, n° 2, p. 227–232.

Conway, J. et Sloane, N. (Nov 1983) A fast encoding method for lattice codes and quantizers. *Information Theory, IEEE Transactions on*, volume 29, n° 6, p. 820–824.

Cormack, G. V. et Horspool, R. N. (1986) Data compression using dynamic markov modelling. *The Computer Journal*, volume 30, p. 541–550.

Desoky, A. et Gregory, M. (Apr 1988) Compression of text and binary files using adaptive huffman coding techniques. *Southeastcon '88., IEEE Conference Proceedings*, volume 1, p. 660–663.

Fu, B. et Parhi, K. (May 1995) Generalized multiplication free arithmetic codes. Dans *Circuits and Systems, 1995. ISCAS '95., 1995 IEEE International Symposium on*, volume 1. p. 437–440.

Gallager, R. (Nov 1978) Variations on a theme by huffman. *Information Theory, IEEE Transactions on*, volume 24, n° 6, p. 668–674.

Guyader, A. L., Lamblin, C. et Boursicaut, E. (1995) Embedded algebraic celp/vselp coders for wideband speech coding. *Speech Communication*, volume 16, n° 4, p. 319–328.

Hale, E. E. (1858) The dot and line alphabet. *Atlantic Monthly.*

Huffman, D. (Sept. 1952) A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, volume 40, n° 9, p. 1098–1101.

Javed, M. et Nadeem, A. (2000) Data compression through adaptive huffman coding schemes. *TENCON 2000. Proceedings*, volume 2, p. 187–190.

Lamblin, C. et Adoul, J.-P. (May 1988) Algorithme de quantification vectorielle shpérique à partir du réseau de gosset d'ordre 8. *Annale de télécommunication*, volume 1, p. 172–186.

McEliece, R. J. (1977) *Theory of Information and Coding.* Addison-Wesley, 390 p.

Ragot, S., Bessette, B. et Lefebvre, R. (May 2004) Low-complexity multi-rate lattice vector quantization with application to wideband tcx speech coding at 32 kbit/s. Dans *Acoustics, Speech, and Signal Processing, 2004. Proceedings. (ICASSP '04). IEEE International Conference on*, volume 1. p. I–501–4.

Rissanen, J. (May 1979) Generalized kraft inequality and arithmetic coding. *IBM Journal of Research and Development*, volume 23, n° 3, p. 337–343.

Rissanen, J. et Langdon., G. (May 1979) Arithmetic coding. *IBM Journal of Research and Development*, volume 23, n° 3, p. 337–343.

Salami, R., Laflamme, C. et Adoul, J.-P. (Apr 1994) 8 kbit/s acelp coding of speech with 10 ms speech-frame : a candidate for ccitt standardization. Dans *Acoustics, Speech, and Signal Processing, 1994. ICASSP-94., 1994 IEEE International Conference on*, volume ii. p. II/97–II100 vol.2.

Salami, R., Lefebvre, R., Lakaniemi, A., Kontola, K., Bruhn, S. et Taleb, A. (May 2006) Extended amr-wb for high-quality audio on mobile devices. *Communications Magazine, IEEE*, volume 44, n° 5, p. 90–97.

Sayood, K. (2000) *Introduction to data compression*, 3e édition. Morgan Kaufmann Publishers Inc., 680 p.

Schalkwijk, J. (May 1972) An algorithm for source coding. *Information Theory, IEEE Transactions on*, volume 18, n° 3, p. 395–399.

Shannon, C. (1948) A mathematical theory of communication. *The Bell System Technical Journal*, volume 27, p. 379–423.

Shannon, C. (Jan 1951) Prediction and entropy of printed english. *Bell system technical journal*, volume 30, n° 4, p. 50–64.

Speech, E. G. T. C. H. R. (1998) *Reference document of GSM full and half-rate complexity evaluation rules for the half-rate selection.*

Speech, E. G. T. C. H. R. (Feb 2005) *Reference docuement of GSM full and half-rate complexity evaluation rules for the half-rate selection.*

Xie, M. et Adoul, J.-P. (1996) Embedded algebraic vector quantizers (eavq) with application to wideband speech coding. *ICASSP '96 : Proceedings of the Acoustics, Speech, and Signal Processing, 1996. on Conference Proceedings., 1996 IEEE International Conference*, volume 1, p. 240–243.

Ziv, J. et Lempel, A. (May 1977) A universal algorithm for sequential data compression. *Information Theory, IEEE Transactions on*, volume 23, n° 3, p. 337–343.

Ziv, J. et Lempel, A. (Sep 1978) Compression of individual sequences via variable-rate coding. *Information Theory, IEEE Transactions on*, volume 24, n° 5, p. 530–536.