QARITH

Décodage souple et/ou itératif de codes quasi-arithmétiques

I. Présentation du logiciel

La technique de décodage séquentiel est sous-optimale pour les codes arithmétiques du fait de l'élagage nécessaire pour limiter la complexité du décodage. Si le modèle de source n'est pas connu, il existe une infinité d'états possibles. S'il est connu, le nombre d'états est fini mais croît exponentiellement avec le nombre de symboles codés.

Les codes arithmétiques à précision réduite, dits *codes quasi-arithmétiques*, permettent de réduire la complexité. Ces codes sont modélisables sous forme d'automates à nombre d'états fini. La réduction de la précision se fait toutefois sans perte excessive en efficacité de compression. Les transitions entre états peuvent être pré calculées. Les opérations arithmétiques sont alors remplacées par des accès dans des tableaux (LUT look-up table).

Grâce à une modélisation appropriée de l'ensemble source et codeur, ce logiciel présente un algorithme de décodage souple optimal, pour une perte minime en compression. Cette modélisation permet l'utilisation de mécanismes de synchronisation souples, ainsi que l'association avec un code convolutif dans un schéma itératif similaire aux turbo codes en série.
Afin d'améliorer l'efficacité en compression, la chaîne de codage incluse une étape de conversion de la source M-aire vers une source binaire

De la redondance est introduite sous la forme de marqueurs de synchronisation afin de favoriser la re-synchronisation de la séquence. Ces marqueurs sont insérés à des positions connues avant le codeur quasi-arithmétique. Ceci permet d'affecter une probabilité nulle aux transitions qui n'émettent pas les marqueurs attendus. Ainsi, certains chemins dans le treillis peuvent être supprimés, ce qui amène une réduction du nombre d'états et une meilleure capacité de re-synchronisation. Ces marqueurs consistent en une séquence fixée de bits connue *a priori*.

Dans cette même optique –amélioration de l'élagage au décodeur–, la transmission d'information adjacente à intervalles réguliers comme la valeur du compteur N_k (nombre de bits à l'instant symbole k) peut être exploitée lors de la reconstruction du treillis, ce qui conduit à une réduction de la taille de ce dernier.

Voici les principales étapes du codeur/décodeur :

1. génération d'une source de Gauss-Markov
2. encodage à l'aide d'un code quasi-arithmétique
3. codage de canal convolutif
4. création du treillis d'estimation
5. passage sur canal gaussien à bruit blanc additif (AWGN)
6. poinçonnage qui permet de contrôler le rendement du code de canal
7. décodage de canal
8. décodage souple (itération sur les deux dernières étapes)

II. Compilation

QARITH se compile à l'aide du workspace Visual Studio 6.0 nommé *qarith.dsw*. La compilation (en mode release) crée un exécutable *qarith.exe* dans le répertoire Release.

III. Utilisation

A. Configuration

Les paramètres de configuration ci-dessous sont à passer en argument de l'exécutable.

T rho nbiter mfreq sfreq k n nbiterCC [permut(8 valeurs)] Seed

Exemple d'utilisation :

```
Release/qarith 4 0.9 500 6000 16   1 1  1  4 1 6 2 3 5 7 0  1
```

B. Description des paramètres

T taille de l'intervalle [0,T] (valeur entière). T permet de contrôler le compromis entre la complexité et l'efficacité de compression (T grand permet une représentation précise de la distribution de la source, T petit permet de remplacer les opérations arithmétiques par des accès à des tableaux LUT pré calculés)

rho facteur de corrélation de la source

nbiter nombre d'itérations

mfreq fréquence des marqueurs de synchronisation

sfreq fréquence d'envoi de l'état du treillis (nombre de bits à l'instant horloge *k*)

k/n rendement du code de canal (entre 1/1 et 1/2)

nbiterCC nombre d'itérations du code turbo

permut index de permutation des 8 symboles au niveau de l'arbre binaire. Diminuer l'incertitude associée à chaque nœud de l'arbre permet d'accroître la robustesse du code.

seed graine du générateur aléatoire

Autres paramètres définis en tête du fichier *qarith.cpp* (entre crochets les valeurs exemples):

NbCtxs nombre de contextes maximal [1024]

Q Quantification sur Q bits [3]

NBSYMBS 2^Q nombre de symboles de l'alphabet [8]

SIZE taille de la source en bits [50*Q]

EbN0min rapport signal à bruit minimal du canal [0]

EbN0max rapport signal à bruit maximal du canal [6]

PEFF probabilité d'effacement [0]

NBITERCCMAX nombre d'itération maximale du code turbo [16]

IV. Résultats

Voici un exemple de résultat obtenu après exécution :

```
N : 4
rho : 0.5
nbctx : 63
75 encoding trellis states.
89 decoding trellis states.
.....................................................................
```

```
mean width of the estimation trellis : 507
H : 111.733 (2.23465 bpss)
lgrtb : 141.063 (2.82125 bpss)
overhead : 26.2502 %
Taux d'erreurs bit a la sortie du canal :
0.0815241 0.0503323 0.0370403 0.0245459 0.0127603 0.00513957 0.00221533
Amplitude moyenne des erreurs :
1.81076 1.65709 1.66333 1.48 1.48352 1.10345 1.58333
SER :
0.27875 0.13925 0.075 0.03125 0.02275 0.00725 0.003
SNR :
12.4988 16.2387 18.9539 23.4983 25.2427 33.6444 33.5316
estimation time : 885.077 symbol bits / s.
```

N	taille de l'intervalle
rho	corrélation
nbctx	nombre de contextes utilisés
H	longueur du train binaire optimale
lgrt	longueur du train binaire atteinte

Ensuite sont donnés pour chaque itération du code turbo (ici une seule) et chaque valeur du rapport signal à bruit du canal :
- le taux d'erreurs bit à la sortie du canal
- l'amplitude moyenne des erreurs
- le taux d'erreurs symbole (SER)
- le rapport signal à bruit (SNR)

CABACSOFTINT

Décodage souple du CABAC avec symbole interdit

V. Présentation du logiciel

Le CABAC (Context Adaptative Binary Arithmetic Coder) est un codeur arithmétique binaire adaptatif utilisé dans le standard de codage vidéo H.264. Ce code permet d'allier une bonne efficacité en compression grâce à l'exploitation de contextes adaptatifs avec une complexité réduite obtenue par des simplifications et l'utilisation de LUTs (Look-Up Tables). En contrepartie, ce code est très sensible au bruit de transmission : une seule erreur peut conduire à une désynchronisation totale du codeur.
Ce programme présente un algorithme de décodage robuste de flux encodés par le CABAC. Cet algorithme est basé sur un estimateur au sens du MAP (Maximum A Posteriori). Des techniques d'élagage permettent de réduire la complexité du décodeur : seuls W chemins les plus probables sont conservés lors du décodage tous les 8 instants de l'horloge bit. Une contrainte de terminaison est ajoutée : le nombre de symboles transmis.

Afin d'améliorer les propriétés de resynchronisation du décodeur, une méthode de détection d'erreurs basées sur un intervalle interdit est introduite. Elle consiste à ajouter un symbole interdit qui n'est jamais transmis dans l'alphabet source. Quand le décodeur tombe dans cet intervalle interdit, cela indique qu'une erreur s'est produite. Cette méthode revient à réserver un intervalle de probabilité amenant de la redondance et nécessite une modification des LUTs (Look-Up Tables) au codeur et au décodeur pour prendre en compte cet intervalle.

Le codeur peut prendre en entrée deux sources théoriques : l'une Gaussienne, l'autre Laplacienne. Toutes deux sont de moyenne nulle, de variance unité et uniformément quantifiées sur 8 valeurs. La première étape de binarisation unaire convertit les symboles de la source d'entrée en éléments binaires appelés bins. Les statistiques des symboles sont passées par un fichier de paramétrage. Les symboles sont supposés statistiquement indépendants. Pour les 8 symboles source, 5 contextes sont définis, un pour chaque bin de la représentation unaire. Ces contextes sont utilisés par le codeur arithmétique pour coder les bins et mettre à jour les probabilités associées à ces contextes.

Voici les principales étapes du codeur/décodeur :

9. génération d'une source aléatoire
10. binarisation unaire
11. encodage à l'aide du CABAC en utilisant l'intervalle interdit
12. passage sur canal binaire symétrique
13. décodage souple

VI. Compilation

cabacSoftInt se compile à l'aide du workspace Visual Studio 6.0 nommé *main.dsw*. La compilation (en mode release) crée un exécutable *main.exe* dans le répertoire Release.

VII. Utilisation

A. Exemple de fichier de configuration

Les paramètres de configuration ci-dessous sont à passer en argument de l'exécutable.
-Input -DecSouple -W -ModeIntInt
Exemple d'utilisation :

Release/main -Input 1 -DecSouple 1 -W 32 -ModeIntInt 1

B. Description des paramètres

- Input choix de la source 1=Gaussienne, 2=Laplacienne
- DecSouple mode décodage souple 1=ON, sinon OFF (décodage hard)
- W nombre de chemins conservés
- ModeIntInt mode intervalle interdit 1=ON sinon OFF

Autres paramètres définis en tête du fichier *main.cpp* (entre crochets les valeurs exemples):

- BER_MIN taux d'erreur bit minimal du canal [0.002]
- BER_MAX taux d'erreur bit maximal du canal [0.005]
- DELTA_BER pas entre les taux d'erreur bit [0.001]
- EPSILON probabilité du symbole interdit [0.2]
- ITERATION_TEST nombre d'itérations [7]

Autres paramètres définis en tête du fichier *sourcenaire.h* (entre crochets les valeurs exemples):

- SIZE_TEST nombre de symboles par paquet [1000]
- N nombre de symboles de la source Naire [8]
- NB_CONTEXT nombre de contextes [5]

VIII. Résultats

Après exécution, les résultats présentés en sortie sont pour chaque itération:

- le débit (en bit/symbole)
- la longueur du train binaire
- le Taux d'Erreur Bit sur le canal
- le temps de décodage
- le résultat du décodage soft : valide ou non valide
- le taux d'erreur symbole

Un fichier *Result1.txt* est également généré. Il récapitule les différents paramètres de configuration et donne pour chaque Taux d'Erreur Bit du canal (BER) le taux de compression, le Taux d'Erreurs Bit et le Taux d'Erreurs Symbole.

```
SIZE_TEST : 1000, ITERATION_TEST : 7
MODE_INTINT : 1, EPSILON : 0.200000
DEC_SOUPLE : 1, SOURCE_MODEL : 1, W : 16
BER_MIN : 0.002000, BER_MAX : 0.005000, DELTA_BER : 5


BER = 0.002000; Compression total = 3.288000, TEB Bit = 0.002259, TES =
0.184857
```

SOFTCCAC

Décodage souple itératif de codes convolutifs – codes arithmétiques

IX. Présentation du logiciel

Le codage arithmétique est très sensible aux erreurs de transmission : un bit altéré par le canal suffit à provoquer une perte de synchronisation du décodeur. Ce logiciel présente une solution de décodage souple sur sources théoriques permettant d'augmenter les performances en présence de bruit.

Ce programme est basé sur un décodage séquentiel avec une sortie *souple*, couplée à un mécanisme d'élagage de l'arbre afin de contrôler la complexité de l'algorithme de décodage. Le codeur arithmétique adaptatif permet de mettre à jour dynamiquement les statistiques de la source au fur et à mesure que les symboles sont lus et codés.

Le critère d'élagage principal est basé sur un seuil appliqué sur la vraisemblance des chemins. A des instant N_k de l'horloge symbole, les chemins du treillis pour lequel la probabilité est inférieure à celle fixée sont supprimés. Egalement, une contrainte est ajoutée sur le nombre de nœuds du treillis. Pour chaque valeur possible de l'horloge symbole N_k, on ne conserve qu'un nombre fixé des nœuds les plus vraisemblables.

De la redondance est introduite sous la forme de marqueurs de synchronisation afin de favoriser la re-synchronisation de la séquence. Ces marqueurs sont insérés à des positions connues avant le codeur arithmétique. Ceci permet d'affecter une probabilité nulle aux transitions qui n'émettent pas les marqueurs attendus. Ainsi, certains chemins dans le treillis peuvent être supprimés, ce qui amène une réduction du nombre d'états et une meilleure capacité de re-synchronisation. Ces marqueurs peuvent être une séquence fixée de bits connue *a priori* ou encore la séquence de bits émise qui aurait été générée à cet instant de l'horloge symbole par la terminaison du codeur arithmétique.

Dans cette même optique –amélioration de l'élagage au décodeur–, la transmission d'information adjacente à intervalles réguliers comme la valeur du compteur N_k (nombre de bits à l'instant symbole k) peut être exploitée lors de la reconstruction du treillis, ce qui conduit à une réduction de la taille de ce dernier.

Enfin, l'algorithme d'estimation peut être associé à un code de canal dans un schéma itératif comparable aux turbo codes en série.

Voici les principales étapes du codeur/décodeur :

14. génération d'une source de Gauss-Markov
15. encodage à l'aide d'un code arithmétique
16. codage de canal convolutif
17. passage sur canal gaussien à bruit blanc additif
18. poinçonnage qui permet de contrôler le rendement du code de canal
19. décodage de canal
20. décodage souple (itération sur les deux dernières étapes)

X. Compilation

SoftCCAC se compile à l'aide du workspace Visual Studio 6.0 nommé *softCCAC.dsw*. La compilation (en mode release) crée un exécutable *softCCAC.exe* dans le répertoire Release.

XI. Utilisation

A. Exemple de fichier de configuration

Le fichier de configuration est à passer en argument de l'exécutable :

Release\softCCAC params

où *params* est le nom du fichier de configuration dont la structure est définie ci-dessous.

```
200   { longueur de la sequence }
3     { nombre de bits pour la quantification de la source }
0.9   { facteur de correlation de la source }
0 6   { plage de Eb/N0 pour la simulation }
10000 { frequence de conservation des Nk }
4     { frequence de terminaison du codeur arithmetique }
10000 { frequence de re-initialisation du codeur arithmetique }
10000 { frequence d'ajout de marqueur de synchro }
2     { longueur des marqueurs de synchro }
1/1   { k/n : rendement du code de canal (entre 1/1 et 1/2) }
80    { nombre de realisations pour chaque Eb/N0 }
1     { nombre d'iterations du principe turbo }
10    { nombre maximum d'etats par Nk et par niveau du treillis (0 pour
decodage hard) }
0     { entrelaceur actif si != 0 }
5     { niveau à partir duquel on commence l'elagage }
5     { seuil sur les probas ( > H) }
0.51 0.51 { seuil sur les mesures ([0;1]), max et min }
10    { seed }
```

B. Description des paramètres

- longueur de la séquence
- nombre de bits pour la quantification de la source : pour *k* bits, l'alphabet comprend 2^k symboles
- facteur de corrélation de la source
- plage de Eb/N_0 pour la simulation : rapports signal à bruit minimal et maximal du canal
- fréquence de conservation des N_k : fréquence de transmission du nombre de bits émis à l'instant symbole *k*
- fréquence de terminaison du codeur arithmétique : ajout de bits générés par une terminaison du codeur arithmétique qui indiquent l'état du codeur
- fréquence de réinitialisation du codeur arithmétique : réinitialisation totale du codeur avec émission de bits de terminaison
- fréquence d'ajout de marqueur de synchro : marqueurs placés à des positions connues *a priori*
- longueur des marqueurs de synchronisation
- k/n : rendement du code de canal (entre 1/1 et 1/2)
- nombre de réalisations pour chaque Eb/N_0
- nombre d'itérations du principe turbo
- nombre maximum d'etats par N_k et par niveau du treillis : critère d'élagage pour l'horloge symbole *k*
- entrelaceur actif si != 0

- niveau à partir duquel on commence l'élagage
- seuil sur les probas (> H) : élagage basé sur la vraisemblance des chemins (probabilité donnée dans le domaine logarithmique)
- seuil sur les mesures ([0;1]), max et min : probabilités calculées sur l'ensemble des observations
- seed : graine pour le générateur aléatoire

XII. Résultats

Après exécution, sont présentés en sortie les résultats pour chaque rapport signal à bruit du canal :

- nombre d'erreur bit moyen en sortie du canal
- nombre d'erreur bit moyen par itération en sortie du canal
- nombre d'erreur bit moyen en sortie du décodage souple
- taux d'erreurs symbole SER
- rapport signal à bruit SNR