Etude et mise au point d'algorithmes rapides de quantification vectorielle : application au codage d'images

Alain Nyeck

Université de Metz — UFR Mathématiques, Informatique, Mécanique

Laboratoire de Mécatronique Industrielle - I.S.G.M.P.

THESE

Présentée à l'Université de Metz
en vue d'obtenir le titre de

DOCTEUR DE L'UNIVERSITE DE METZ EN ELECTRONIQUE

par

Alain NYECK

ETUDE ET MISE AU POINT D'ALGORITHMES RAPIDES DE QUANTIFICATION VECTORIELLE. APPLICATION AU CODAGE D'IMAGES

Soutenue publiquement le 30 janvier 1992
devant la Commission d'Examen

Président	M. A. TOSSER-ROUSSEY	Professeur à l'ENI de Metz **(Directeur de Thèse)**
Rapporteurs	M. D. FLAENDER	Directeur de TDF/CERLOR - Metz
	M. L. LEGRAND	Maître de Conférences - Praticien Hospitalier Université de Dijon
	Mme M. LUMBRERAS	Professeur à l'Université de Metz
Examinateurs	M. ACKERMANN	Professeur à l'Université de Clermont - Ferrand II
	M. LHERMITTE	Professeur à SUPELEC - Metz
	M. VIGOUROUX	Professeur à l'IUT de Troyes
	M. YVROUD	Directeur de Recherche au CNRS - Nancy

Université de Metz — UFR Mathématiques, Informatique, Mécanique

Laboratoire de Mécatronique Industrielle - I.S.G.M.P.

THESE

Présentée à l'Université de Metz
en vue d'obtenir le titre de

DOCTEUR DE L'UNIVERSITE DE METZ EN ELECTRONIQUE

par

Alain NYECK

ETUDE ET MISE AU POINT D'ALGORITHMES RAPIDES DE QUANTIFICATION VECTORIELLE . APPLICATION AU CODAGE D'IMAGES

Soutenue publiquement le 30 janvier 1992
devant la Commission d'Examen

Président	: M. A. TOSSER-ROUSSEY	Professeur à l'ENI de Metz **(Directeur de Thèse)**
Rapporteurs	: M. D. FLAENDER	Directeur de TDF/CERLOR - Metz
	M. L. LEGRAND	Maître de Conférences - Praticien Hospitalier Université de Dijon
	Mme M. LUMBRERAS	Professeur à l'Université de Metz
Examinateurs :	M. ACKERMANN	Professeur à l'Université de Clermont - Ferrand II
	M. LHERMITTE	Professeur à SUPELEC - Metz
	M. VIGOUROUX	Professeur à l'IUT de Troyes
	M. YVROUD	Directeur de Recherche au CNRS - Nancy

AVANT - PROPOS

Le travail présenté a été réalisé dans le cadre de la coopération entre le Laboratoire de Mécatronique Industrielle de l'Université de Metz et le Centre de Recherche CERLOR de TéléDiffusion de France.

Je tiens par conséquent à exprimer toute ma gratitude à Monsieur le Professeur André TOSSER-ROUSSEY, Directeur du Laboratoire de Mécatronique Industrielle, pour toutes les années passées dans son laboratoire. Je lui suis également reconnaissant pour les encouragements et les conseils qu'il m'a prodigués pour mener à bien ce travail et pour l'honneur qu'il me fait de présider mon jury de thèse.

Je suis très reconnaissant à Monsieur Didier FLAENDER, Directeur du CERLOR, pour m'avoir accueilli comme chercheur dans son centre de recherche, je le remercie en outre d'avoir accepté d'examiner ce travail et d'en être rapporteur.

Sincères remerciements à Monsieur Louis LEGRAND, Maître de Conférence - Praticien des Hôpitaux à l'Université de Dijon, et à Madame Martine LUMBRERAS, Professeur à l'Université de Metz, pour avoir accepté d'être rapporteurs de ce travail et de manifester leur intérêt pour cette contribution au domaine du codage d'images.

Je remercie vivement Messieurs ACKERMANN, Professeur à l'Université de Clermont-Ferrand II, LHERMITTE, Professeur à l'ESE de Metz, VIGOUROUX, Professeur à l'IUT de Troyes, YVROUD, Directeur de Recherche au CNRS, pour l'attention qu'ils ont bien voulu porter à ce travail et pour l'honneur qu'ils me font de participer à ce jury.

Mes remerciements s'adressent également à Monsieur Hervé CAUDRON, Directeur du laboratoire TMI du CERLOR, pour l'environnement matériel et scientifique de qualité mis à ma disposition pour réaliser ce travail.

Que l'ensemble de mes collègues du laboratoire de mécatronique industrielle et tout le personnel du CERLOR trouvent ici l'expression de mes remerciements chaleureux pour la collaboration efficace et pour l'ambiance amicale et sympathique qui a toujours régné au sein des différentes équipes.

A ma famille, à tous les anonymes qui par leurs conseils, leurs critiques pertinentes, leur soutien perpétuel ont contribué directement ou indirectement à la réalisation de ce travail, qu'ils trouvent ici le témoignage de ma profonde considération.

Table des matières

Introduction I

Chapitre 1 - La Quantification Vectorielle

1 - Principes de base 1

1.1 Généralités 1
1.2 La Quantification Vectorielle 1

2 - Application de la QV au codage d'image 2

3 - Construction du dictionnaire 6

3.1 But de la classification 6
3.2 Définition d'une partition 6
3.3 Problèmes de la classification automatique 6
3.4 Algorithmes de génération du dictionnaire 7

3.4.1 Quantificateur optimal 7
3.4.2 Algorithme LBG 8

3.4.2.1 Justification de l'algorithme 9

3.4.2.1.1 Moment d'ordre deux d'une partition 9
3.4.2.1.2 Théorème de Huyghens 9
3.4.2.1.3 Application à une partition 10
3.4.2.1.4 Application à l'algorithme LBG 11

3.4.2.2 Limitations de l'algorithme LBG 12

3.4.3 Algorithme d'apprentissage à seuil 13

3.4.3.1 Ce qu'il faut retenir sur l'algorithme 13

3.4.4 Autres méthodes de construction du dictionnaire 14
3.4.5 Utilisation des réseaux de neurones en QV 14

3.4.5.1 Construction du dictionnaire à partir du modèle auto-adaptatif de Kohonen 15

4 - Choix du dictionnaire initial 19

4.1 Méthode de séparation maximale 19
4.2 Méthode des histogrammes multidimensionnels 20
4.3 Initialisation paramétrique 20
4.4 Méthode du découpage binaire 21
4.5 Initialisation par réduction de l'erreur de quantification 21
4.6 Méthode du maximum d'entropie 23

4.6.1 Algorithme 23
4.6.2 Résultats expérimentaux 25

5 - Conclusion du chapitre 26

6 - Références bibliographiques 34

Chapitre 2 - Algorithmes rapides de quantification vectorielle

1 - Introduction 36

2 - Le problème 36

3 - La recherche séquentielle 37

3.1 Algorithme 37
3.2 Complexité 38

4 - Recherche par arbre binaire 39

4.1 Objectif de la méthode 39
4.2 Complexité 39

5 - Dictionnaire ordonné par moyennes croissantes 41

6 - Elimination par inégalité triangulaire 45

6.1 Principe 45
6.2 Quelques notions sur les espaces métriques 45

6.2.1 Normes 45
6.2.2 Exemples de normes 46
6.2.3 Distances 47

6.3 Utilisation de l'inégalité triangulaire en quantification vectorielle 49
6.4 Analyse du nombre moyen d'opérations 53

7 - Recherche par calcul partiel de la distance 55

7.1 Principe de la méthode 55
7.2 Algorithme 55
7.3 Résultats des simulations et discussion 57
7.4 Recherche autoadaptative 59
7.5 Recherche autoadaptative exploitant la redondance spatiale de l'image 60

7.5.1 Discussion et perspectives 65

8 - Algorithmes rapides de QV exploitant les caractéristiques statistiques de l'image 65

8.1 Introduction 65
8.2 Caractérisation de la redondance spatiale de l'image 66
8.3 Algorithme rapide de recherche autoadaptative par arrangement progressif du dictionnaire 66

8.3.1 Position du problème 66
8.3.2 Algorithme 66
8.3.3 Résultats des simulations 71

8.4 Algorithme rapide de recherche sur un dictionnaire dont les vecteurs sont ordonnés par moyenne croissante 79

8.4.1 Position du problème 79
8.4.2 Algorithme 80
8.4.3 Résultats des simulations 82

9 - Conclusion du chapitre 83

10 - Références bibliographiques 84

Chapitre 3 - Utilisation des transputers en quantification vectorielle des images

1 - Introduction 85

2 - Aspect matériel du transputer 86

2.1 Famille T8 86

3 - Programmation des systèmes à base de transputers 88

3.1 Utilisation du langage C 88

3.1.1 Modèle de programmation 89
3.1.2 Programmation multiprocesseurs 89
3.1.3 Programmation temps réel 90

3.2 Développement de programmes 90

4 - Implémentation d'algorithmes rapides de QV sur des systèmes multi-transputers 92

4.1 Méthode de parallélisation 92
4.2 Configuration du réseau de transputers 93
4.3 Résultats expérimentaux 96

5 - Références bibliographiques 98

Conclusion générale 99

Annexes 100

Introduction

L'image et en particulier l'image animée est un fluide délicat à transmettre. L'énorme quantité d'informations qui la constituent requiert non seulement des technologies de haut débit mais aussi des techniques efficaces de compression du volume de données, afin de gagner en vitesse et en capacité de stockage, l'objectif étant principalement d'utiliser une artère de communication à faible débit pour transmettre une image de bonne qualité.

De plus, comme les applications du codage d'images sont nombreuses et en rapide développement : images satellite, vidéo-conférence, transmission de fac-similés de documents imprimés, images de télévision, stockage d'images, le but second d'un système de compression de données est d'obtenir la meilleure fidélité de restitution pour un débit donné, en d'autres termes, il s'agit d'accroître le taux de compression pour une même qualité de restitution.

A cet effet, plusieurs techniques de codage d'images ont été développées ces dernières années. Parmi les plus connues, on distingue le codage MICD (Modulation par Impulsions Codées Différentielles) , le codage utilisant une transformation orthogonale, le codage hybride, ou des versions adaptatives de ces techniques. Ces travaux ont abouti à la définition de standards de compression-décompression de données comme JPEG (Joint Photographic Experts Group) pour la compression-décompression d'images fixes couleur, ou MPEG (Motion Picture Experts Group) pour la vidéo intégrale et la transmission à grande vitesse, plutôt dédié aux systèmes multimédia et à la télévision haute définition. D'une façon générale, les méthodes présentées précédemment exploitent la redondance spatiale de l'image et adoptent des critères psychovisuels pour diminuer le débit de transmission. Toutefois, un des défauts typiques de ces techniques et qui réduit considérablement leur optimalité est que la quantification est effectuée sur des scalaires, alors que les données traitées sont fortement corrélées. En effet, un résultat fondamental de la théorie de l'information de Shannon concernant le débit et la distorsion est qu'il est plus économique à distorsion égale, de coder un ensemble de n points considéré comme un vecteur de R^n plutôt que n points indépendants.

En Quantification Vectorielle, la procédure usuelle est la suivante : l'image à coder est d'abord traitée de façon à générer un ensemble de vecteurs d'apprentissage appelé ensemble d'entraînement ou training set. Ensuite, à l'aide d'un algorithme de classification de données, un dictionnaire de vecteurs représentatifs de l'image est constitué à partir de ce training set. Le codage d'un vecteur consiste alors à chercher son représentant dans le dictionnaire selon un critère de distorsion minimale et transmettre le code du vecteur sélectionné. La reconstruction de l'image est effectuée à l'aide d'une simple table de transcodage (lookup table), et il suffit d'adresser la table par l'indice du représentant choisi pour obtenir le vecteur de reconstruction.

Le problème majeur de la quantification vectorielle en termes de vitesse de traitement et de dimension de mémoire occupée est l'énorme quantité de calculs nécessaires pour générer le dictionnaire d'une part, et coder un vecteur d'autre part. Ceci rend particulièrement délicat l'utilisation de cette technique pour des applications de codage d'image en temps réel.

Nous nous sommes donc intéressés dans ce travail à la mise au point de techniques rapides de quantification vectorielle pour le codage d'images.

Le travail présenté se divise en trois parties comme suit :

Le premier chapitre est un exposé introductif à la quantification vectorielle, l'accent étant mis sur les techniques de construction du dictionnaire; la méthode LBG et le modèle neuronal auto-adaptatif de Kohonen sont analysés. La qualité du dictionnaire généré par l'algorithme LBG étant fortement dépendante de la technique d'initialisation utilisée, une nouvelle méthode d'initialisation efficace qualifiée de méthode du maximum d'entropie est introduite.

Afin de diminuer les temps de calculs prohibitifs inhérents à la génération du dictionnaire et au codage d'une image, de nouveaux algorithmes rapides de quantification vectorielle sont proposés au chapitre 2. Ces algorithmes exploitent les propriétés métriques de la distance aussi bien que l'intercorrélation entre les blocs image adjacents pour accélérer le codage de chaque vecteur, et augmenter la vitesse de convergence de l'algorithme de classification utilisé pour la construction du dictionnaire.

Enfin, le dernier chapitre décrit l'implémentation des algorithmes rapides de quantification vectorielle mis au point sur des systèmes multi-processeurs à base de transputers.

Chapitre 1

LA QUANTIFICATION VECTORIELLE

1. Principes de base

1.1. Généralités

Au cours des dernières années, les travaux de recherche en codage d'image se sont intensément focalisés sur la quantification vectorielle (en abrégé QV) [GRA-84] [BUH-86] [NAS-88b] . En effet, un résultat fondamental de la branche de la théorie de l'information de Shannon concernant le débit et la distorsion est qu'une meilleure compression du débit peut toujours être obtenue en codant des vecteurs plutôt que des scalaires. L'optimalité de la compression est d'autant meilleure que la dimension du vecteur est élevée. Ainsi, on peut déjà constater la supériorité du codage par transformée (TCD , Walsh-Hadamard, ...) qui agit sur des vecteurs, sur tout codeur scalaire du type MICD (Modulation par Impulsions Codées Différentielles). Cet avantage est encore plus significatif si les coefficients transformés sont en plus codés par quantification vectorielle. La Quantification Vectorielle apparaît ainsi d'emblée comme une méthode de codage extrêmement puissante et ouvrant un champ d'évolution important, du moins en théorie.

1.2. La Quantification Vectorielle

La Quantification Vectorielle consiste à discrétiser un vecteur $x = (x_i ; i = 1,...,k)$ de dimension k dont les composantes sont des variables aléatoires réelles et continues, en un vecteur $w = (w_i ; i = 1,...,k)$ de même dimension, dont les composantes sont des variables réelles et discrètes. Le vecteur de reconstruction w (codeword) appartient à un ensemble fini et ordonné de **représentants** W appelé **dictionnaire** (codebook).

Ainsi

$$Q : \quad R^k \text{ --------> } W$$
$$x \text{ --------> } w = Q(x)$$

Pour tout vecteur x à coder, un représentant $w = Q(x)$ est sélectionné dans le dictionnaire selon la règle du plus proche voisin, et l'indice du vecteur sélectionné est transmis : c'est la phase codage de la quantification vectorielle.

Les éléments du vecteur à coder peuvent être des pixels, des coefficients de transformée, des valeurs d'erreur de prédiction ou des écarts avec le vecteur représentatif choisi lors d'une première quantification vectorielle.

De l'autre côté du canal de transmission, le mot de code correspondant au représentant sélectionné est utilisé par le décodeur pour déterminer le vecteur de reproduction dans le dictionnaire, ainsi que le montre la figure 1.1.

Ce schéma permet de conserver quelle que soit la complexité du codeur, un décodeur très simple : seule la connaissance du dictionnaire lui est nécessaire. Il lui suffit d'adresser le représentant désigné par l'indice reçu pour obtenir le vecteur de reconstruction, c'est donc en sorte, une Look Up Table.

Le nombre de bits nécessaires pour transmettre le code du vecteur sélectionné représente le débit du quantificateur. Ainsi, si le dictionnaire contient N représentants c'est-à-dire $W = \{w_j ; j = 1,...,N\}$ où $w_j = (w_{ji} ; i = 1,...,k)$ est un vecteur de dimension k, et que les indices sont transmis sans aucun codage statistique, le débit du codeur est $\log_2 N$ bits par vecteur, soit $(\log_2 N)/k$ bits par dimension.

2. Application de la QV au codage d'image

De nombreuses méthodes de codage d'image utilisant la quantification vectorielle ont été proposées ces dernières années, un excellent article de Nasrabadi passe en revue toutes ces techniques dans [NAS-88b].

D'une façon générale, la QV appliquée au codage d'image peut être considérée comme un processus d'extraction de redondances dans l'image, processus qui se décompose en quatre phases comme suit :

a - Vectorisation de l'image

Il s'agit de découper l'image en blocs rectangulaires de taille nxm, chaque bloc sera ensuite identifié à un vecteur de même dimension. Les composantes du vecteur seront les valeurs de luminance des pixels adjacents, ou alors ces valeurs préalablement traitées. A cet effet, les deux opérations les plus indiquées pour le prétraitement sont la normalisation et la transformation orthogonale.

- Normalisation

Elle consiste à opérer une transformation bilinéaire sur le vecteur en fonction de sa moyenne μ et de son écart-type σ [MUR-82].

Soit $s = (s_i ; i = 1,...,k)$ un vecteur de dimension k.

Le vecteur normalisé correspondant x est de la forme :

$x = (s - \mu) / \sigma$

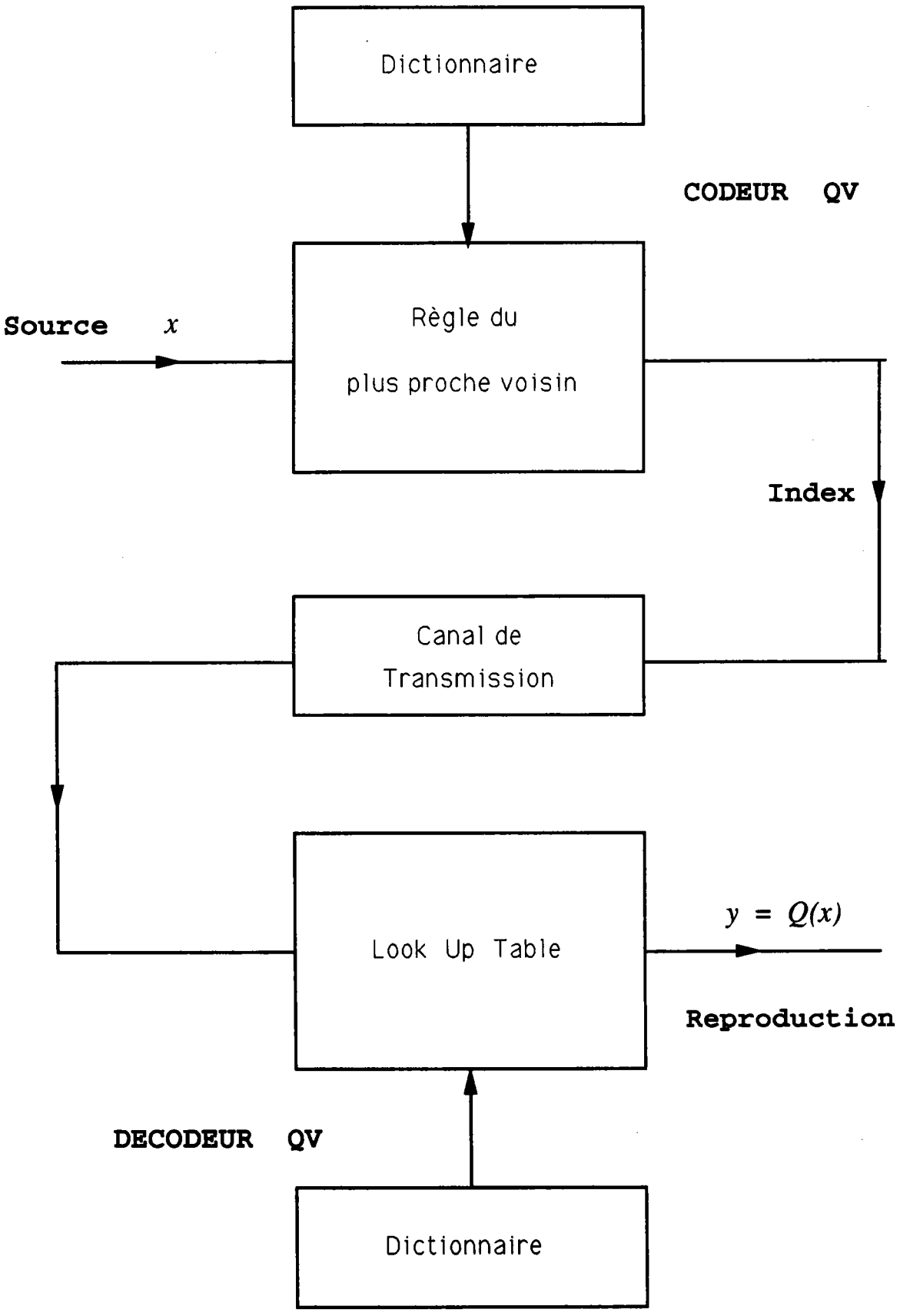


Fig 1.1. Schéma de compression de données utilisant la QV.
Seul le mot de code correspondant au vecteur représentatif choisi est transmis.
Le décodeur utilise le code reçu pour sélectionner le vecteur de reproduction.

avec

$$\sigma = (1/k \sum_{i=1}^{k} (s_i - \mu)^2)^{1/2}$$

$$\mu = 1/k \sum_{i=1}^{k} s_i$$

Ainsi, la densité de probabilité des vecteurs normalisés peut être considérée comme fixe pour tous les vecteurs.

- Transformation orthogonale

Un autre prétraitement possible consiste à appliquer sur chaque bloc de pixels une transformation orthogonale bidimensionnelle [MAR-86]. La Transformée en Cosinus Discrète (TCD) présente des propriétés intéressantes adaptées au codage du signal image : toute l'énergie du bloc image transformé est répartie sur un ensemble de coefficients dont le premier est proportionnel à la moyenne des intensités sur le bloc original. Le reste des coefficients fournit une information de type fréquentielle. Une des propriétés principales de la TCD est de décorréler les composantes des vecteurs. Ainsi, on peut éliminer la redondance intravecteur et utiliser la QV pour éliminer la redondance intervecteur. La décorrélation permet aussi de diminuer l'effet de moyennage dû à la QV, particulièrement au niveau des contours. Une autre propriété est la concentration possible de l'énergie sur un nombre restreint de coefficients.

b - Génération de l'ensemble d'apprentissage (training set)

C'est la constitution de l'ensemble d'apprentissage à partir duquel un dictionnaire de vecteurs représentatifs de l'espace à coder sera généré. Si le signal considéré est stationnaire et ergodique, ce dictionnaire est utilisable pour coder des vecteurs issus de la même source, mais non contenus dans la séquence initiale. Dans le cas contraire et typique du signal image, il faudrait choisir une séquence d'entraînement très large et construire un dictionnaire de grande taille.

Une autre solution consiste à établir une préclassification des vecteurs de la source. A cet effet, Gersho et Ramamurthi préconisent de séparer les blocs homogènes et les blocs contenant des contours en deux training set différents [RAM-86]. Ensuite, un dictionnaire relatif à chaque type de blocs est généré. Avec cette approche, on peut appréhender plus facilement les caractéristiques locales de l'image, et éliminer l'effet de bloc perceptible sur les images codées à faible débit, effet de bloc dû notamment au mauvais raccordement des bords de blocs image adjacents.

Ainsi, en construisant des dictionnaires émanant de training set différents et liés à la perception visuelle, on peut améliorer la qualité de l'image codée tout en gardant un débit total acceptable.

c - Construction du dictionnaire

Le dictionnaire est constitué des vecteurs les plus représentatifs de l'espace à coder. Il est généré à partir du training set par l'utilisation d'un algorithme de classification du type *k-means* [MAC-67].

Pour un critère de distorsion donné, cet algorithme vise itérativement à optimiser la partition de l'espace des réalisations en fonction du dictionnaire courant, puis à optimiser le dictionnaire à partir de la partition courante. Le dictionnaire est considéré comme optimal lorsqu'une itération supplémentaire n'entraîne plus de diminution significative de la distorsion moyenne finale.

Nous présenterons plus en détails les techniques de génération du dictionnaire au paragraphe 3 de ce chapitre.

d - La quantification

Elle consiste pour un vecteur source donné, à chercher le vecteur de reproduction correspondant dans le dictionnaire. Ce processus nécessite une énorme quantité de calculs si la recherche du plus proche voisin s'effectue par un parcours exhaustif de tout le dictionnaire. Il est donc nécessaire de mettre au point des algorithmes de recherche rapide : ce sera l'objet du prochain chapitre.

3. Construction du dictionnaire

3.1. But de la classification

La classification fait partie de la grande famille des techniques d'analyse de données, son but comme le précise Roux dans [ROU-85] est d'obtenir une représentation schématique simple d'un tableau rectangulaire de données dont les colonnes sont des descripteurs de l'ensemble des observations placées en lignes.

L'objectif de la classification est donc de répartir l'échantillon de données en groupes d'observations homogènes ou classes, chaque classe étant bien différentiée des autres.

En ce qui nous concerne, l'échantillon de données ou ensemble d'entraînement (Training Set) sera constitué d'une série d'images de référence (Fig. 1.6) , chaque image sera divisée en petits blocs ou vecteurs de dimension nxm. Après un éventuel prétraitement de chaque bloc, une transformation par cosinus discrète par exemple, les blocs ainsi transformés serviront de base de données pour la construction d'un dictionnaire (Codebook) de N vecteurs.

3.2. Définition d'une partition

Soit un ensemble S de M éléments.
On dit que Q(S) est une partition de S s'il existe un ensemble de sous-ensembles de $S = \{S_i ; i = 1,...,N\}$ non vides, tels que:

$$\bigcup_{i=1,...,N} S_i = S \quad \text{avec} \quad S_i \neq \varnothing$$

$$S_i \cap S_j = \varnothing \quad \forall \; i \neq j$$

3.3. Problèmes de la classification automatique

Le problème majeur de la classification automatique est lié au volume de données à analyser. Chercher la meilleure partition d'un ensemble S au sens d'un critère donné consiste à examiner toutes les partitions possibles de l'ensemble S. C'est une solution difficile à envisager pour des ensembles de grande dimension.

A titre d'exemple, considérons un ensemble S de dimension M et notons P(M , N) le nombre de partitions de S en N classes.

On a $P(M , N) = 0$ pour $N > M$
$P(M , M) = 1$
$P(M , 1) = 1$

Pour $N = 2$, on a $P(M , 2) = 2^{M-1} - 1$.

Pour N > 2, le nombre de partitions de S en N classes prend vite des proportions peu réalistes pour un traitement informatique. Une expression sous la forme d'une récurrence de ce nombre est donnée dans [JAM-89] par :

$$P(M , N) = N \cdot P(M-1 , N) + P(M-1 , N-1)$$

Ceci étant, il est difficile d'imaginer un algorithme fournissant une partition optimale au sens d'un critère fixé. C'est pourquoi l'on a recours à des heuristiques, c'est-à-dire des algorithmes dont on considère qu'ils sont suffisamment raisonnables pour donner des résultats satisfaisants. Ces algorithmes consistent à déterminer des partitions en s'imposant un certain nombre de contraintes pour rendre le calcul possible.

Nous présentons dans la suite de ce chapitre, les techniques de construction du dictionnaire généralement utilisées en codage d'image, les avantages et les inconvénients des unes et des autres seront exposées, laissant ainsi la perspective à l'utilisateur de choisir la méthode qui convient le mieux à son application.

3.4. Algorithmes de génération du dictionnaire

3.4.1. Quantificateur optimal

Soit $S = \{x_j ; j = 1,...,M\}$ une séquence d'entraînement de M vecteurs.

Pour apprécier la performance d'un quantificateur vectoriel par rapport à un dictionnaire donné C, on définit une mesure de distorsion totale comme suit:

$$D = 1/M \sum_{j=1}^{M} d(x_j , Q(x_j))$$

où $d(x_j , Q(x_j))$

représente la **distance** entre le *jième* vecteur x_j de la source et son représentant $Q(x_j)$ dans le dictionnaire.
Nous n'explicitons pas plus la notion de **distance** dans ce chapitre, nous y reviendrons en détails dans le prochain chapitre en justifiant notamment le choix porté sur la mesure de distorsion adoptée.

Un quantificateur vectoriel est optimal s'il minimise la distorsion moyenne D. Pour y parvenir, deux conditions sont nécessaires:

1 - Pour tout vecteur x , le quantificateur doit choisir le représentant $Q(x)$ fournissant la distorsion la plus faible:

$$d(x , Q(x)) < d(x , y) \qquad \forall\ y \in C , y \neq Q(x)$$

2 - Le choix de chaque représentant doit être tel qu'il minimise la distorsion moyenne de la cellule correspondante. Ceci revient à représenter chaque cellule par son barycentre [LIN-80].

3.4.2. Algorithme LBG [LIN-80]

L'algorithme LBG (Linde Buzo Gray) est une généralisation de l'algorithme de Lloyd [LLO-82] pour des variables multidimensionnelles. Partant d'un dictionnaire initial de N vecteurs et d'une séquence d'entraînement, on construit de manière itérative un dictionnaire optimal en respectant les critères d'optimalité définis précédemment.

Cet algorithme est du reste, décrit dans la procédure suivante :

Initialisations

Fixer un nombre N de représentants du dictionnaire
un seuil $\varepsilon > 0$
un dictionnaire initial $\hat{A}_0$
une séquence d'entraînement $S = \{x_j ; j = 1,...,M\}$
une distorsion initiale $D_{-1} = \infty$
une itération initiale $m = 0$

Phase 1

Pour le dictionnaire courant $\hat{A}_m = \{y_i ; i = 1,...,N\}$,

trouver la partition optimale $P(\hat{A}_m) = \{S_i ; i = 1,...,N\}$ minimisant la distorsion moyenne sur la séquence d'entraînement S,

avec $x_j \in S_i$ si $d(x_j , y_i) < d(x_j , y_l) \quad \forall\ l \in [1 , N] , l \neq i$.

Calculer la distorsion moyenne pour la partition obtenue :

$$D_m = D(\hat{A}_m , P(\hat{A}_m)) = 1/M \sum_{j=1}^{M} \min(d(x_j , y_i)) \quad y_i \in \hat{A}_m$$

Phase 2

Si $(D_{m-1} - D_m) / D_m < \varepsilon$ arrêter l'algorithme, le dictionnaire optimal est $\hat{A}_m$.

Sinon continuer.

Phase 3

Trouver le dictionnaire optimal $\hat{A}_{m+1}$ pour la partition $P(\hat{A}_m)$.

$\hat{A}_{m+1} = \{y_i ; i = 1,...,N\}$ avec y_i barycentre de la cellule S_i .

Incrémenter m à m+1 et aller à la ***phase 1*** .

Arrêt de la procédure

Le dictionnaire final est obtenu si l'une des deux conditions suivantes est vérifiée :

- Le nombre maximal d'itérations est atteint.
- Deux étapes consécutives ne modifient pas le contenu des classes.

3.4.2.1. Justification de l'algorithme

L'algorithme LBG repose sur d'intéressantes propriétés mathématiques qu'on va examiner ci-dessous.

3.4.2.1.1. Moment d'ordre deux d'une partition

Considérons un nuage matériel $I = \{x_j ; j = 1,...,n\}$ représentant n observations (vecteurs) de l'espace des données, et pondérons chaque observation $x_j = (x_{ji} ; i = 1,...,k)$ par une masse ponctuelle m_j.

Le centre de gravité g de ce nuage est défini comme suit :

$$g = (x_1 + x_2 + ... + x_n) / m$$

où $m = m_1 + m_2 + ... + m_n$ est la masse totale du nuage.

Le moment centré d'ordre deux ou moment par rapport au centre de gravité est la quantité :

$$M^2(I/g) = \sum_{j=1}^{n} m_j \, d^2(x_j , g)$$

où $d^2(x_j , g)$ est le carré de la distance entre x_j et g.

Dans le cas de la distance euclidienne,

$$d^2(x_j , g) = \sum_{i=1}^{k} (x_{ji} - g_i)^2$$

Le moment centré d'ordre 2 est minimum lorsque toutes les observations x_j (j = 1,...,n) sont regroupées autour du centre de gravité g. Le moment centré d'ordre 2 est donc une mesure de la dispersion du nuage.

Par ailleurs, la variance d'une variable qui est la mesure usuelle de la dispersion en statistique s'écrit :

$$var(x_i) = 1/m \sum_{j=1}^{n} [m_j (x_{ji} - g_i)^2]$$

Le moment centré d'ordre 2 est donc une variance généralisée au cas de k dimensions.

3.4.2.1.2. Théorème de Huyghens

Examinons le cas du moment d'ordre 2 par rapport à un point x différent du centre de gravité.

D'après le théorème de Huyghens, le moment d'un solide par rapport à un point quelconque x est égal au moment du solide par rapport à son centre de gravité, augmenté du moment du point g, affecté de la masse totale m du solide par rapport au point x.

Ce qui se traduit par :

$M^2(I/x) = M^2(I/g) + m \cdot d^2(g, x)$

3.4.2.1.3. Application à une partition

Soit Q une partition de l'ensemble des observations $I = \{x_j ; j = 1,...,n\}$.

Soit q un élément de Q, on appellera m_q la masse du sous-ensemble des points de q, et g_q le centre de gravité de la classe q.

Le moment centré d'ordre 2 de la partition Q est donné par :

$$M^2(I/g) = \sum_{j=1}^{n} m_j \, d^2(x_j, g)$$

Cette expression peut se décomposer de la façon suivante:

$M^2(I/g) = \sum_{q \in Q} [\sum_{x \in q} m_x \, d^2(x, g)]$.

La somme entre crochets représente le moment de la classe q par rapport au point g différent du centre de gravité g_q.

En appliquant le théorème de Huyghens à la cellule q, on obtient :

$M^2(I/g) = \sum_{q \in Q} [M^2(q/g_q) + m_q \, d^2(g_q, g)]$

ou encore

$M^2(I/g) = \sum_{q \in Q} [M^2(q/g_q)] + \sum_{q \in Q} [m_q \, d^2(g_q, g)]$.

L'expression précédente représente la décomposition de la dispersion totale en dispersion à l'intérieur des classes (intra-classe), et en dispersion entre les classes (inter-classe).

Une bonne classification devra rendre la dispersion intra-classe aussi petite que possible de façon à fournir des classes homogènes, et donc une dispersion inter-classe maximale pour avoir des classes représentatives de tout l'espace des données [ROU-85].

3.4.2.1.4. Application à l'algorithme LBG

La justification de l'algorithme est donnée par sa convergence, c'est-à-dire qu'à partir d'un certain rang, le moment centré d'ordre 2 de la partition ne décroît plus, donc, que la partition et les centres sont stables.

Pour démontrer cette convergence, il faut montrer que le passage du dictionnaire $\hat{A}_{m-1}$ généré à l'étape de rang (m-1) au dictionnaire $\hat{A}_m$ construit à l'étape suivante diminue la dispersion intra-classe.

Ceci revient à démontrer que

$$\sum_{q\in Q} M^2(q^{(m-1)} / g_{q^{(m-1)}}) \geq \sum_{q\in Q} M^2(q^{(m)} / g_{q^{(m)}})$$

où $M^2(q^{(m)} / g_{q^{(m)}})$

est le moment centré d'ordre 2 de la classe q par rapport à son centre de gravité g_q à l'étape de rang m.

Considérons $q^{(m)}$ comme la classe constituée autour du centre de gravité $g_{q^{(m-1)}}$ de l'ancienne classe $q^{(m-1)}$.

Appelons $M^{(m-1)}$ le moment intra-classe de la partition à l'étape de rang (m-1),

$$M^{(m-1)} = \sum_{q\in Q} [\sum_{x\in q^{(m-1)}} m_x\, d^2(x, g_{q^{(m-1)}})]$$

Appelons M^* la valeur de $M^{(m-1)}$ après réaffectation des points au centre de gravité le plus proche,

$$M^* = \sum_{q\in Q} [\sum_{x\in q^{(m)}} m_x\, d^2(x, g_{q^{(m-1)}})]$$

M^* n'est plus la somme des moments centrés puisque le passage de l'algorithme L.B.G de l'itération (m-1) à l'itération m, a changé les barycentres des classes.

Soit alors x un élément de la classe $q^{(m-1)}$. Si x n'a pas changé de classe, sa contribution au moment intra-classe reste la même. Par contre si x provient d'une autre classe, c'est qu'il est plus proche du centre de gravité de cette classe que de $g_{q^{(m-1)}}$, et sa contribution à M^* est inférieure à celle qu'il avait dans $M^{(m-1)}$.

On a donc $M^* \leq M^{(m-1)}$.

Par ailleurs, appelons $M^{(m)}$ le moment intra-classe de la nouvelle partition,

$$M^{(m)} = \sum_{q\in Q} [\sum_{x\in q^{(m)}} m_x\, d^2(x, g_{q^{(m)}})].$$

D'après le théorème de Huyghens, $M^* = M^{(m)} + \sum_{q\in Q} [m \cdot d^2(g_{q^{(m)}}, g_{q^{(m-1)}})]$.

On a donc $M^{(m)} \leq M^*$

et par conséquent $M^{(m)} \leq M^{(m-1)}$.

3.4.2.2. Limitations de l'algorithme LBG

- L'algorithme fournit une solution-partition dépendante du nombre de classes N fixé a priori. Ce qui a pour conséquence que les partitions en (N+1) classes ou en (N-1) classes peuvent être très différentes de la partition en N classes.

- L'algorithme suppose que le signal est stationnaire et ergodique. Si c'est le cas, le dictionnaire construit à partir d'une séquence d'entraînement suffisamment longue reste optimal pour coder des vecteurs hors de cette séquence.

- L'algorithme fournit une solution-partition dépendante du choix des centres initiaux. Une autre partition initiale peut donc donner une partition finale pour laquelle le critère du moment d'ordre 2 soit encore meilleur. Un tel algorithme itératif ne converge pas en général vers la distorsion moyenne minimale, mais vers un optimum local dépendant des classes initiales et de la nature des données traitées.

Une stratégie visant à pallier cet inconvénient est l'examen des groupements stables ou formes fortes obtenus en opérant plusieurs passages de l'algorithme, effectuant à chaque passage un choix différent des classes initiales. Ces formes fortes représentent des sous-ensembles d'objets qui ont toujours été réunis dans la même classe finale au cours des différents essais des partitions initiales. Mais, il n'y a aucune garantie mathématique à une telle pratique [JAM-89] : la partition obtenue qui peut avoir plus de classes que les N classes fixées au départ, ne respecte donc pas le critère d'optimalité de l'algorithme de base car l'optimum obtenu par l'algorithme pour une partition en N classes n'est pas forcément un optimum pour une partition en $N' \neq N$ classes.

- Le domaine d'application de l'algorithme est celui où le nombre N de classes fixées correspond à un nombre N réel de classes à reconnaître. Ce qui suppose une analyse préalable des données pour estimer le nombre de classes optimal. Il en est de même du choix des classes initiales, élément capital pour la convergence de l'algorithme. A cet effet, nous présenterons dans le paragraphe **4** diverses techniques d'initialisation des classes, adaptées à l'algorithme LBG.

3.4.3. Algorithme d'apprentissage à seuil

L'algorithme d'apprentissage à seuil qu'on peut aussi appeler algorithme d'agrégation autour de centres variables en classes de diamètre fixé a priori, est une approche simple et rapide de construction du dictionnaire [MIC-85] , [FLA-79].

Le principe de base de l'algorithme est de déterminer une partition de l'ensemble des données en autant de classes qu'il est nécessaire pourvu qu'elles soient de diamètre inférieur à un seuil fixé a priori par l'utilisateur de la méthode, ainsi que le montre la procédure suivante :

soit $S = \{x_j ; j = 1,...,M\}$ une séquence d'entraînement de M vecteurs.

Le premier vecteur x_1 formera à lui tout seul une classe, et il en sera le représentant.

Au cours du fonctionnement, l'arrivée du vecteur x_j $(j > 1)$ aura l'effet suivant :

si les distances $d(x_j , c_i)$ entre le vecteur x_j et le représentant c_i de la *iième* classe, sont toutes supérieures à un seuil donné s, on crée une nouvelle classe dont x_j est le seul élément et le représentant à la fois.

Sinon, on affecte x_j à la classe la plus proche (règle du plus proche voisin), et on réactualise éventuellement son représentant, en prenant par exemple comme nouveau centre de la classe son centre de gravité.

Arrêt de la procédure

L'algorithme s'arrête si l'une des deux conditions est vérifiée :

- Tous les vecteurs de la séquence d'entraînement sont traités $(i = M)$.

- On fait le choix d'un nombre maximum de classes souhaitées; dans ce cas, si le nombre maximum de classes est dépassé, il faut réitérer la procédure entièrement en augmentant la valeur du seuil.

3.4.3.1. Ce qu'il faut retenir sur l'algorithme

- L'algorithme fournit une solution dépendante du diamètre fixé a priori. Ce diamètre peut être estimé par essais successifs ou par analyse préalable du nuage associé aux données.

- Quoique très sensible à la distribution locale des éléments et à l'ordre dans lequel les données arrivent, l'algorithme d'apprentissage à seuil reste assez simple en comparaison à l'algorithme L.B.G, et son peu de complexité algorithmique fait que l'on peut l'utiliser sur des masses de données importantes. Par ailleurs, cet algorithme présente une bonne part de flexibilité dans la mesure où on peut interrompre l'apprentissage, obtenir un dictionnaire partiel, l'évaluer et l'agrandir au besoin, en poursuivant l'apprentissage avec de nouvelles données.

3.4.4. Autres méthodes de construction du dictionnaire

Nous avons présenté deux techniques efficaces de construction du dictionnaire pour la quantification vectorielle des images. Loin d'être une illustration exhaustive des méthodes de classification de données, ces deux algorithmes ont des propriétés intéressantes et à ce titre, ils sont bien adaptés au codage d'image. Notons toutefois qu'il existe de nombreuses versions antérieures de ces algorithmes : *k-means* [MAC-67] , *isodata* [BAL-65] , *nuées dynamiques* [DID-71] , *partitions en groupements stables* [LEB-77] , pour ne citer que celles-là.

Dans la suite de ce chapitre, nous étudierons les possibilités offertes par l'utilisation des réseaux neuronaux et notamment, le réseau neuronal de Kohonen, comme méthode d'apprentissage pour la génération du dictionnaire en quantification vectorielle d'images.

3.4.5. Utilisation des réseaux de neurones en QV

Les réseaux de neurones sont des structures de traitement parrallèle composées d'unités élémentaires appelées neurones, et un ensemble d'interconnexions entre ces unités et les entrées du réseau. L'objectif de ces structures est la modélisation du cerveau humain à partir de son élément le plus simple, le neurone.

Un réseau neuronal est caractérisé par les entités suivantes [DAV-90] [LEM-91] :

- **Le neurone, noeud du réseau:** c'est un automate relié à des automates voisins; il en reçoit des signaux, les intègre, et les transmet à son tour aux autres automates.

- **La topologie du réseau:** chaque élément du réseau est relié à d'autres éléments par des connexions. Reste à déterminer la structure de l'ensemble des liaisons et la nature de chaque connexion.
D'un point de vue physiologie, ces connexions représentent des synapses: elles peuvent donc être inhibitrices ou excitatrices. Pour modéliser ce rôle, on affecte à chaque connexion entre deux neurones formels un poids qui pondère le signal transmis.

- **L'évolution du réseau:** c'est le comportement du réseau pendant la phase d'apprentissage. Les liaisons inter-neuronales sont progressivement modifiées selon les variations de la fonction d'activation du neurone. L'état stable est atteint lorsque les connexions ne se modifient plus, c'est-à-dire lorsque l'apprentissage est terminé.

Un des avantages majeurs de l'utilisation des réseaux neuronaux en QV est que la plupart des techniques d'apprentissage sont adaptatives et de surcroît, elles s'adaptent bien aux tâches de quantification vectorielle.

Toutefois parmi les nombreuses méthodes existantes, le modèle de Kohonen [KOH-84] [NAS-88a] semble être le plus apte à être utilisé comme quantificateur vectoriel. En effet, il réalise une projection d'un espace d'entrées dans un ensemble fini de vecteurs de sortie, suivant une procédure proche de celle mise en oeuvre dans l'algorithme L.B.G, une méthode qui conduit à un quantificateur optimal localement.

Par ailleurs, dans l'algorithme LBG, la constitution du dictionnaire initial conditionne fortement l'optimalité du dictionnaire final, alors que dans le modèle de Kohonen, les valeurs de convergence des vecteurs du réseau ne dépendent pas des valeurs initiales.

3.4.5.1. Construction du dictionnaire à partir du modèle auto-adaptatif de Kohonen

La topologie du réseau de Kohonen [KOH-84] est formée de deux couches: une d'entrée classique et une de sortie où les neurones sont entièrement connectés. Chaque entrée est toujours reliée à tous les neurones du réseau par l'intermédiaire de KxN coefficients ou poids.

Pour chaque neurone **j** , on appelle w_j (j = 1,...,N) le vecteur de poids des connexions qui lui proviennent du signal d'entrée :

$w_j = (w_{ji} ; i = 1,...,K)$ le signal d'entrée étant de dimension K.

Une propriété importante du réseau est que les poids évoluent de façon à conserver entre les vecteurs de sortie des relations topologiques présentes entre les vecteurs d'entrée. On dit que le réseau s'auto-organise (self-organizing).

L'entraînement du réseau consiste d'abord à définir autour de chaque neurone, un voisinage qui évoluera dans le temps en se rétrécissant. Ensuite, après avoir initialisé les connexions aléatoirement, on calcule la distance qui s'étend entre l'entrée $x = (x_i ; i = 1,...,K)$ et chaque neurone de sortie, puis on choisit le neurone de distance minimale. Les poids des neurones appartenant au voisinage du neurone gagnant sont alors mis à jour selon la procédure :

$$w(t+1) = w(t) + \alpha(t)(x(t) - w(t))$$

$\alpha(t)$ est un coefficient d'apprentissage compris entre 0 et 1, et qui décroit progressivement vers zéro au fur et à mesure de l'apprentissage.

L'algorithme de Kohonen est du reste résumé dans la procédure suivante :

(1) Initialisations

Soit un réseau neuronal de taille (N , K) où N est le nombre de vecteurs (neurones) et K la taille de chaque vecteur.

Pour $i = 1,...,N$ et $j = 1,...,K$ initialiser les connexions w_{ij} aléatoirement.

Appelons $S = \{x(t) ; t = 1,...,M\}$ une séquence d'entraînement de M vecteurs.

Soit R un réel positif et D(**j** , R) le disque ayant pour centre le neurone **j** et de rayon R.

(2) Présenter un nouveau vecteur $x(t) = (x_i(t) ; i = 1,...,K)$ à l'entrée du réseau.

(3) Calculer la distance d_j entre ce vecteur et tous les neurones **j** $(j = 1,...,N)$

$$d_j^2 = \sum_{i=1}^{K} (x_i(t) - w_{ji}(t))^2$$

(4) Sélectionner le neurone gagnant **g** tel que $d_g = \min d_j \quad (j = 1,...,N)$

(5) Pour $i = 1,...,N$ et pour les neurones appartenant à D(**g** , R) ajuster les poids par :

$$w_{ij}(t+1) = w_{ij}(t) + \alpha(t)(x_i(t) - w_{ij}(t))$$

où $0 \leq \alpha(t) \leq 1$ est un coefficient d'apprentissage qui décroit dans le temps; il en est de même du rayon R qui décroit vers zéro au fur et à mesure de l'apprentissage.

(6) Aller à l'étape *(2)*

Contrairement à l'algorithme L.B.G, l'algorithme d'apprentissage de Kohonen n'est pas itératif, et la séquence d'entrainement n'est pas partitionnée en cellules dont le représentant est le centre de gravité. Le réseau neuronal de Kohonen change dynamiquement les vecteurs du dictionnaire en utilisant l'expression adaptative de l'étape *(5)* :

$$w_{ij}(t+1) = w_{ij}(t) + \alpha(t)(x_i(t) - w_{ij}(t))$$

Reste à déterminer le coefficient d'adaptation $\alpha(t)$ et le rayon R(t) correspondant au voisinage du neurone gagnant, de façon à exploiter les caractéristiques particulières des images à coder.

Nasrabadi et Feng proposent dans [NAS-88a], des expressions différentes selon les types de vecteurs traités.

Pour des vecteurs à forte variance, représentant les blocs image de contours, Nasrabadi et Feng utilisent les fonctions suivantes :

$$\alpha(t) = 0.1 \times e^{(-t / 20000)}$$

$$R(t) = 5 + 495 \times e^{(-t / 100)}$$

Pour des vecteurs à faible variance, les expressions suivantes sont proposées:

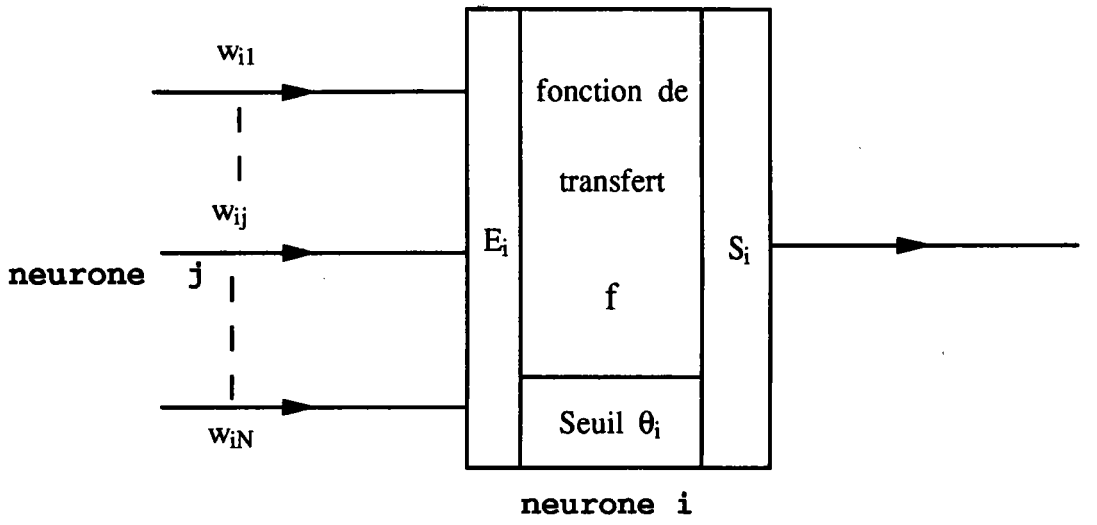
$$\alpha(t) = 0.01 \times e^{(-t / 10000)}$$

$$R(t) = 3 + 5 \times e^{(-t / 100)}$$

L'approche neuronale de Kohonen et l'algorithme LBG sont par leur efficacité, les deux techniques généralement utilisées pour la construction du dictionnaire en quantification vectorielle. Les résultats obtenus par l'une ou l'autre méthode sont comparables en termes de qualité du dictionnaire [NAS-88a] [LEB-89]; ceci se justifie par le fait que ces deux algorithmes convergent vers un dictionnaire réputé localement optimal.

Le modèle neuronal de Kohonen a comme avantage la parallélisation des traitements et surtout l'auto-adaptation.
L'algorithme LBG s'impose comme méthode de référence pour la génération du dictionnaire, son inconvénient majeur est que le dictionnaire final obtenu dépend étroitement du dictionnaire d'initialisation utilisé. A cet effet, nous présentons dans le prochain paragraphe, des techniques d'initialisation visant à améliorer les performances de l'algorithme LBG.

Fig 1.2. Le neurone formel



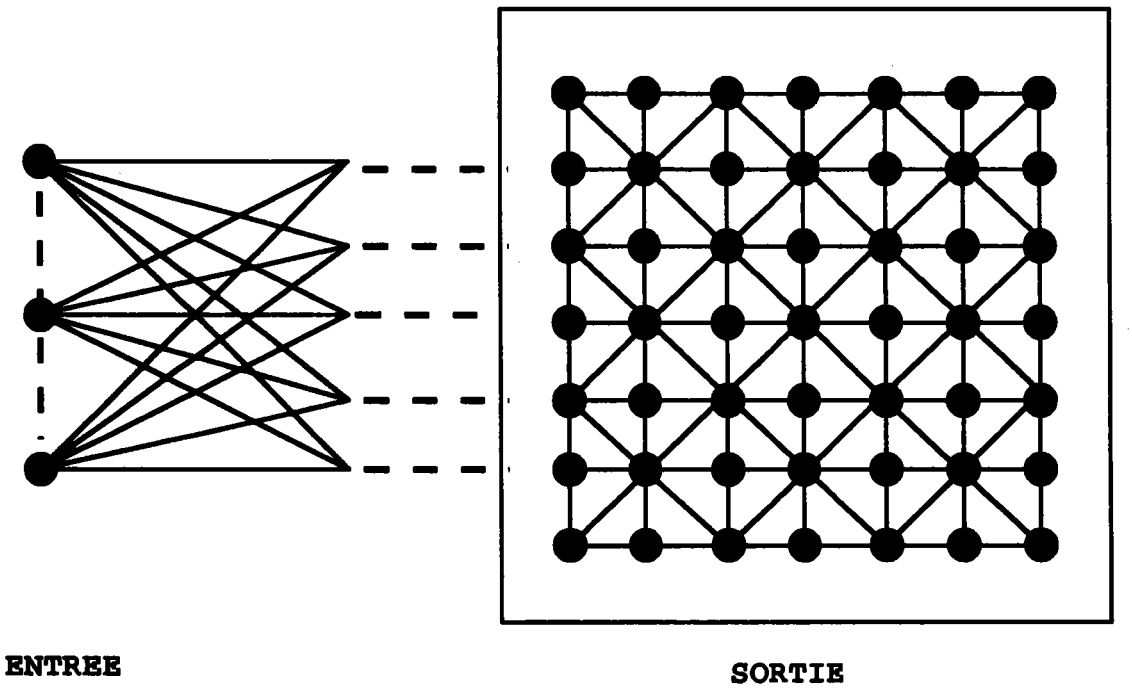
E_i est la somme des sorties S_j des neurones connectés au neurone i , pondérée par les poids w_{ij}

$$E_i = \sum_{j=1}^{N} w_{ij}\, \theta_i$$

La sortie S_i du neurone est fonction de E_i et du seuil θ_i

$$S_i = f(E_i - \theta_i)$$

Fig 1.3. Le modèle auto-adaptatif de Kohonen



Chaque neurone d'entrée est connecté à chaque neurone de sortie

4. Choix du dictionnaire initial

L'algorithme L.B.G. fournit une solution partition dépendante du choix des centres initiaux. La vitesse de convergence de l'algorithme et la validité du dictionnaire final dépendront étroitement du choix des classes initiales.

Il n'existe pas de méthode de choix universelle qui assure l'optimalité certaine du quantificateur, toutefois, une bonne technique d'initialisation doit remplir les objectifs suivants:
garantir une convergence rapide de l'algorithme de classification et une erreur de quantification minimale. D'autre part, il va de soi que le processus d'initialisation lui-même doit être bref donc aussi peu complexe que possible.

Après avoir présenté un certain nombre de techniques d'initialisation couramment utilisées, nous proposerons une nouvelle méthode d'initialisation efficace, optimisant l'entropie du dictionnaire, et qualifiée de méthode du maximum d'entropie.

4.1. Méthode de la séparation maximale [SUN-84]

La technique d'initialisation la plus simple consiste à utiliser les N premiers vecteurs de la séquence d'entraînement, mais intuitivement, on est tenté de choisir N vecteurs convenablement espacés, et les N premiers vecteurs ne peuvent l'être si les variables sont corrélées. La méthode de séparation maximale consiste à choisir les N premiers vecteurs les plus espacés les uns par rapport aux autres, comme l'indique la procédure suivante :

(1) Fixer un seuil de distorsion initial, $D = \infty$.

(2) Trouver N vecteurs dans la séquence d'entraînement tels que la distance entre chacun des vecteurs et tous les autres soit supérieure à D.
Ces N vecteurs constituent le dictionnaire d'initialisation.

(3) Si la valeur de D est trop grande pour rassembler N vecteurs, alors diviser D par 2

(D <------ D/2) et aller à l'étape *(2)*.

Cette méthode est applicable sur des données corrélées ou non corrélées. Toutefois, cette technique n'exploite pas directement les caractéristiques statistiques de l'ensemble des données, en particulier, pour le codage du signal image, la distribution des vecteurs de la séquence d'entraînement est a priori inconnue, cette approche peut alors fournir un dictionnaire fortement sous-optimal.

4.2. Méthode des histogrammes multidimensionnels

Cette méthode consiste à générer l'histogramme multidimensionnel de l'ensemble des données et d'utiliser les pics de l'histogramme comme vecteurs d'initialisation [WAR-83].

Cette technique, facilement utilisable dans le cas scalaire, est peu pratique pour des vecteurs de grande dimension du fait de la capacité mémoire importante qu'elle nécessite et surtout de la difficulté à calculer l'histogramme. Un autre problème est que la méthode ne garantit pas une maximisation de la dispersion interclasse dans la mesure où deux vecteurs correspondant à deux pics de l'histogramme peuvent être fortement corrélés.

4.3. Initialisation paramétrique [HIL-77]

Soit $S = \{x_l ; l = 1,....,M\}$ une séquence d'entraînement de M vecteurs,

où $x_l = (x_{li} ; i = 1,....,k)$ est un vecteur de dimension k.

Il s'agit de former le dictionnaire d'initialisation $C_0 = \{c_j ; j = 1,....,N\}$ constitué de N vecteurs de dimension k, $c_j = (c_{ji} ; i = 1,....,k)$.

On commence par calculer la valeur moyenne et l'écart-type des composantes des vecteurs x_j sur chaque dimension de l'espace.

On appellera

$$m(i) = 1/M \sum_{l=1}^{M} x_{li}$$ la valeur moyenne de la *iième* composante des vecteurs x_l

et $$\sigma(i) = (1/M \sum_{l=1}^{M} (m(i) - x_{li})^2)^{1/2}$$ l'écart-type des *iièmes* composantes des vecteurs x_l

La valeur de la *iième* composante du *jième* vecteur du dictionnaire d'initialisation C_0 est donnée par :

$$c_{ji} = (m(i) - c \cdot \sigma(i)) + (j - 0.5) \cdot 2 \cdot c \cdot s(i) / k ,$$

où c est un paramètre qui dépend de la distribution des vecteurs de la séquence d'entraînement.

Chaque dimension de l'espace est donc partionnée en fonction de la variance. Cette technique d'initialisation conduit généralement à des résultats acceptables, à condition que les données soient corrélées dans toutes les dimensions de l'espace.

4.4. Méthode du découpage binaire [LIN-80]

Cette méthode consiste à construire itérativement des dictionnaires de taille croissante jusqu'à l'obtention du nombre de représentants voulus.

A partir du barycentre c de la séquence d'entraînement, on forme deux nouveaux vecteurs comme suit :

$c(1) = c - \varepsilon$ et $c(2) = c + \varepsilon$

ε étant un vecteur perturbation.

L'espace est ainsi découpé en deux partitions ayant respectivement comme représentant $c(1)$ et $c(2)$.

Ce nouveau dictionnaire est optimisé par l'algorithme de classification. Chaque vecteur résultat est découpé en deux vecteurs en ajoutant la perturbation ε , et on réitère le processus optimisation-découpage jusqu'à l'obtention d'un dictionnaire dont la taille correspond à nos exigences.

Cette technique produit généralement des résultats satisfaisants. Toutefois, la perturbation apportée doit être faible pour que la distorsion moyenne n'augmente pas. Faute de connaître la distribution des vecteurs dans l'espace, la perturbation est souvent choisie de manière aléatoire. Il faut cependant être prudent lors du découpage des cellules afin de ne pas affecter un trop grand nombre de représentants dans une zone de l'espace peu peuplée, ce qui risquerait d'entraîner une sous-optimalité du quantificateur due à l'apparition de cellules vides.

4.5. Initialisation par réduction de l'erreur de quantification [YUA-88]

Dans cette méthode, les vecteurs initiaux sont choisis séquentiellement de façon à minimiser l'erreur de quantification due à la génération d'une nouvelle classe.

Supposons que pour une séquence d'entraînement, i vecteurs aient déja été sélectionnés. Nous pouvons considérer qu'ils constituent un quantificateur vectoriel de i niveaux QV(i), les vecteurs choisis en étant les représentants.

Appliquer QV(i) à la séquence d'entraînement conduit à une erreur de quantification ou distorsion E_i. Si un nouveau quantificateur vectoriel de i+1 niveaux, QV(i+1), est construit en ajoutant un nouveau représentant à QV(i), alors la distorsion résultante est donnée par $E_{i+1}(p)$, où p est la position du nouveau représentant dans l'espace.

Par ailleurs, $E_{i+1}(p) \leq E_i$

et la différence $R_i(p) = E_i - E_{i+1}(p)$ représente la réduction de l'erreur de quantification due à l'ajout d'un niveau supplémentaire au quantificateur.

Sous ces conditions, la fonction de réduction de l'erreur de quantification $R_i(p)$ varie avec la position de la (i+1)*ième* cellule.

Le problème est comment déterminer p, position de la (i+1)*ième* cellule, de façon à maximiser la fonction $R_i(p)$.

Yuan et Goldberg estiment dans [YUA-88] qu'une bonne approximation de $R_i(p)$ est la fonction suivante :

$F_i(p) = f(p) \cdot (D_i(p))^2$

où f(p) est la fréquence d'occurrence des cellules au voisinage de la position p, et $D_i(p)$ la distance à la cellule la plus proche.

Cette technique est du reste exposée dans la procédure ci-dessous :

(1) Diviser l'espace vectoriel en N cellules.
Compter la fréquence d'occurrence f(p), dans la séquence d'entraînement, de chaque cellule; en d'autres termes, générer un histogramme.

(2) Eliminer toutes les cellules ayant une fréquence d'occurrence inférieure à un certain seuil.

(3) Choisir la cellule la plus fréquente comme premier représentant.

(4) Sélectionner comme (i+1)*ième* représentant, celui se trouvant à la position p, de façon à maximiser $F_i(p)$.

(5) Repéter l'étape *(4)* jusqu'à l'obtention du nombre désiré de représentants.

La technique précédente donne de bons résultats dans la mesure où elle tient compte des caractéristiques statistiques de la base de données. Les résultats obtenus en simulation montrent que l'algorithme de classification converge rapidement avec une erreur de quantification minimale.

4.6. Méthode du maximum d'entropie [NYE-92]

Le problème majeur des techniques d'initialisation précédentes est que l'algorithme d'apprentissage génère généralement un dictionnaire dans lequel certains représentants sont très peu utilisés. Dans le cas extrême, plusieurs cellules sont vides. Afin de contourner cet obstacle, nous introduisons dans ce paragraphe une technique d'initialisation optimisant l'entropie du dictionnaire. Les vecteurs initiaux sont choisis de telle sorte qu'ils aient des fréquences d'occurrence quasi identiques (entropie maximum). Les résultats expérimentaux illustrent l'application de la technique au codage d'image: l'algorithme de classification converge rapidement avec une erreur de quantification minimale, vers un dictionnaire d'entropie maximale.

4.6.1. Algorithme

Un quantificateur vectoriel optimal consiste à désigner N vecteurs de reproduction de façon à minimiser la distorsion moyenne. D'après le théorème de Huyghens, ceci revient à rendre la dispersion à l'intérieur d'une classe aussi petite que possible, et par la même occasion, rendre maximale la dispersion entre les classes: c'est le critère du moment d'ordre deux d'une partition.

Un autre critère efficace pour avoir une bonne classification des données est de maximiser l'entropie du dictionnaire, en d'autres termes, s'assurer que chaque représentant est utilisé avec la même fréquence pendant le codage de la base de données [AHA-90] , [KRI-90] . C'est l'approche que nous adopterons dans ce paragraphe.

D'un point de vue mathématique, l'entropie du dictionnaire est définie par:

$$E = - \sum_i p_i \log_2(p_i) \qquad i = 1,...,N$$

où N est la taille du dictionnaire et p_i est la fréquence relative à laquelle le *iième* vecteur de reproduction est utilisé pour coder la base de données.

Appelons σ_p l'écart-type moyen entre les diverses probabilités d'occurrence p_i ; σ_p est défini par :

$$\sigma_p = \left[1/N \sum_{i=1}^{N} (p_i - m_p)^2 \right]^{1/2}$$

où $m_p = 1/N \sum_i p_i$ est la fréquence d'occurrence moyenne.

L'entropie E de la source est maximum si l'écart-type σ_p est minimum, c'est-à-dire lorsque tous les vecteurs du dictionnaire sont équiprobables ($p_i = m_p \quad \forall \; i = 1,...,N$).

La distribution des vecteurs de la séquence d'entraînement étant inconnue a priori, le problème est comment trouver un ensemble de N vecteurs ayant approximativement des fréquences d'occurrence identiques.

Il serait extrêmement coûteux en temps de calcul d'examiner toutes les combinaisons de N vecteurs de la séquence d'apprentissage afin de déterminer laquelle conduit à la meilleure entropie. Pour une séquence de M vecteurs, ceci reviendrait à tester

$$\frac{M!}{N!\,(M-N)!}$$ combinaisons.

Cette approche est loin d'être réaliste, et nous proposons ci-après une solution beaucoup plus simple et efficace d'un point de vue statistique.

Soient N vecteurs $\{y_i\ ;\ i = 1,...,N\}$ de dimension k, correspondant à N cellules S_i $(i = 1,...,N)$ de l'espace des données.

Pour tout vecteur x de la séquence d'entraînement,

$x \in S_i$ si $N_i \cdot d(x, y_i) \leq N_j \cdot d(x, y_j)$ pour tout $i \neq j$,

où N_i représente la taille de la cellule S_i et $d(x, y_i)$ la distance entre le vecteur x et le vecteur y_i représentant la cellule S_i.

Par ailleurs, il est démontré dans [LIN-80] qu'une condition nécessaire pour obtenir un dictionnaire optimal est que chaque cellule de l'espace des données soit représentée par son barycentre.
Soit c_i le barycentre de la cellule S_i, nous choisirons comme vecteur d'initialisation, le vecteur $x \in S_i$ qui se rapproche le plus de c_i.

Se basant sur les idées émises précédemment, nous proposons ci-dessous un algorithme d'initialisation efficace pour la désignation d'un quantificateur vectoriel.

*(1) **Initialisations***

Soit un ensemble initial de N vecteurs de l'espace $Y_0 = \{y_i\ ;\ i = 1,...,N\}$
Nous supposerons qu'aucune structure particulière n'est introduite dans Y_0 ; le problème est de déterminer la partition optimale $P(Y_0) = \{S_i\ ;\ i = 1,...,N\}$ qui maximise l'entropie E.
Soit N_i la taille de la cellule S_i, initialisons N_i à 1 pour tout i.

(2) Entrer un vecteur $x = (x_i\ ;\ i = 1,...,k)$ de la séquence d'entraînement.

(3) $x \in S_i$ si $N_i.d(x, y_i) \leq N_j.d(x, y_j)$

$i = 1,....,N$ $\quad$ $j = 1,....,N$ $\quad$ $i \neq j$

(4) Tester si c'est le dernier vecteur entré ou non.
Si oui, aller à l'étape *(5)*, sinon incrémenter N_i à N_i+1 et aller à l'étape *(2)*.

(5) Soit C l'ensemble des barycentres des différentes cellules.

$$C = \{c_i\ ;\ i = 1,....,N\}$$

avec $$c_i = 1/N_i \sum_{i=1}^{N_i} (y_i \in S_i)$$

(6) Le dictionnaire d'initialisation Y nécessaire à l'algorithme de classification est obtenu par:

$Y = \{y_i\ ;\ i = 1,....,N\}$ où $y_i \in S_i$ et y_i est le vecteur qui se rapproche le plus de c_i.

4.6.2. Résultats expérimentaux

La technique d'initialisation proposée pour la désignation d'un quantificateur vectoriel a été simulée sur ordinateur et ses performances ont été évaluées sur des dictionnaires de différentes tailles. La séquence d'entraînement était composée de quatre images 256x256 pixels sur 8 bits (Fig 1.6) , partitionnées en blocs de dimension k = 2x2 pixels. Nous avons adopté l'algorithme d'apprentissage LBG [LIN-80] pour évaluer les performances de la méthode en comparaison avec la technique d'initialisation par séparation maximum et la technique d'initialisation par découpage binaire. Les résultats obtenus sont présentés dans le tableau ci-dessous, et les images codées à partir de dictionnaires obtenus en initialisant l'algorithme LBG par chacune des méthodes sont montrées à la fin de ce chapitre.

N	Méthode	Iter.	EQM1	EQM2	E1	E2
	ISM	21	1152.6	217.2	1.99	3.40
16	IDB	12	1092.4	214.3		3.28
	IME	9	259.6	213.4	3.74	3.86
	ISM	19	510.4	157.5	2.66	3.97
32	IDB	18	1092.4	155.1		4.11
	IME	13	200.4	156.2	4.81	4.72
	ISM	15	414.4	113.3	3.39	4.78
64	IDB	16	1092.4	112.6		4.93
	IME	16	157.2	111.2	5.76	5.62
	ISM	22	265.2	77.3	4.08	5.58
128	IDB	19	1092.4	78.1		5.76
	IME	24	131.8	76.9	6.78	6.56
	ISM	20	162.1	53.1	5.07	6.47
256	IDB	21	1092.4	52.4		6.22
	IME	19	121.9	53.3	7.68	7.52

Table 1.1
Comparaison des performances des trois techniques d'initialisation.

N : taille du dictionnaire.
Iter. : nombre d'itérations pour la convergence de l'algorithme de classification.
EQM1 : distorsion initiale (Erreur Quadratique Moyenne).
EQM2 : distorsion finale.
E1 : entropie initiale du dictionnaire.
E2 : entropie finale du dictionnaire.
ISM : Initialisation par Séparation Maximale.
IDB : Initialisation par Découpage Binaire.
IME : Initialisation par Maximum d'Entropie.

Le tableau précédent montre que la technique proposée procure des résultats satisfaisants, notamment par rapport à la méthode de séparation maximum et la méthode de découpage binaire. Le nombre d'itérations nécessaires à la convergence de l'algorithme de classification est minimal et l'entropie du dictionnaire est maximale, ce qui laisse penser que tous les vecteurs de reproduction contribuent équitablement à la distorsion globale du système.

L'erreur quadratique moyenne ou distorsion est évaluée dans nos calculs comme suit:

$$EQM = 1/M \sum_{i=1}^{M} [d(x_i , Q(x_i))]^2$$

où M est la taille de la séquence d'apprentissage,
x_i est le *iième* vecteur de la source et $Q(x_i)$ le représentant de x_i dans le dictionnaire.

Par ailleurs, on peut observer sur la figure 1.5 que la distorsion induite par la technique d'initialisation proposée est faible dès la première itération de l'algorithme LBG. Une application de cette technique peut donc être la génération rapide de classes sur un ensemble de données avec contrainte de temps réel. On pourrait alors dans ce cas, arrêter l'algorithme LBG dès la première itération en ayant un dictionnaire d'une qualité acceptable.

5. Conclusion du chapitre

Nous avons présenté dans ce chapitre, les différentes techniques de construction du dictionnaire pour la quantification vectorielle. Deux de ces méthodes sont particulièrement adaptées au codage d'image: l'algorithme de partitionnement LBG et le modèle neuronal de Kohonen, qui convergent tous les deux vers un dictionnaire localement optimal.

Il reste néanmoins qu'un apprentissage utilisant ces algorithmes est très consommateur en temps de calculs.
A titre d'exemple, prenons un training set constitué de trois images de 512x512 pixels sur 8 bits. La génération d'un dictionnaire de 256 vecteurs de taille $k = 4$ nécessite plus de 6 heures de calculs sur une station *SPARC1*.

Cette quantité considérable de calculs est essentiellement due à l'effort nécessaire à tout vecteur de la source pour déterminer son plus proche voisin dans le dictionnaire à chaque étape de la classification.

Afin d'avoir des temps de calculs raisonnables pour la construction du dictionnaire et la quantification des vecteurs de la source, il est donc nécessaire de mettre au point des algorithmes rapides de recherche : c'est ce que nous verrons dans le prochain chapitre.

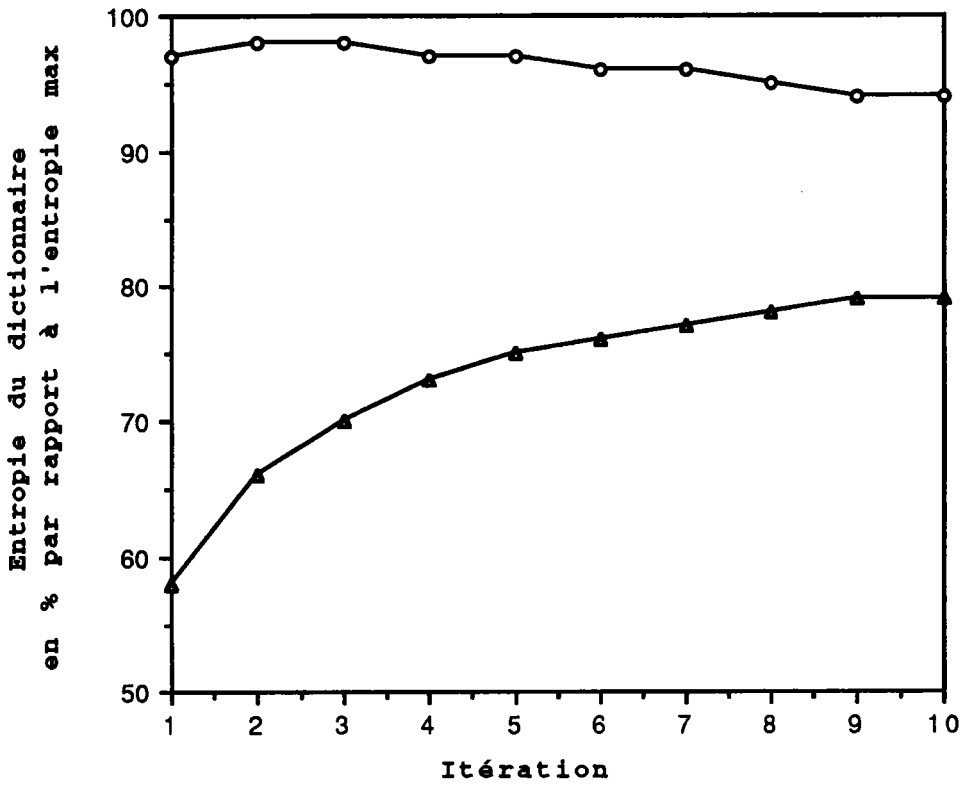


Fig 1.4. Evolution de l'entropie du dictionnaire après chaque itération de l'algorithme de classification

Initialisation par la méthode de séparation maximum (128 vecteurs de dimension k = 4)
Initialisation par la méthode du maximum d'entropie (128 vecteurs de dimension k = 4)

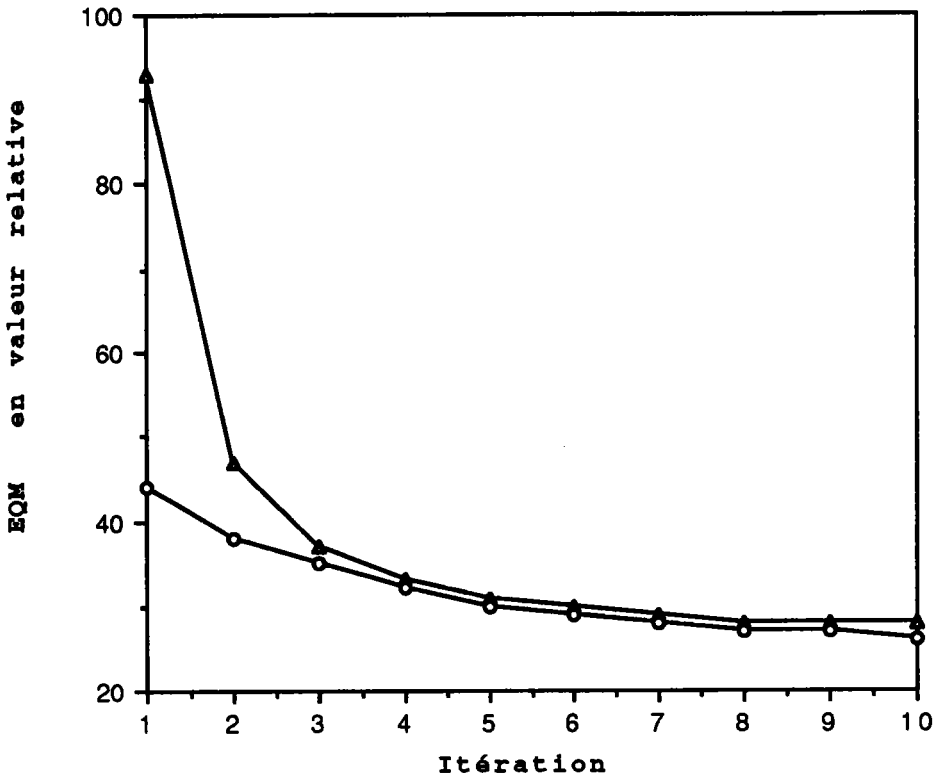
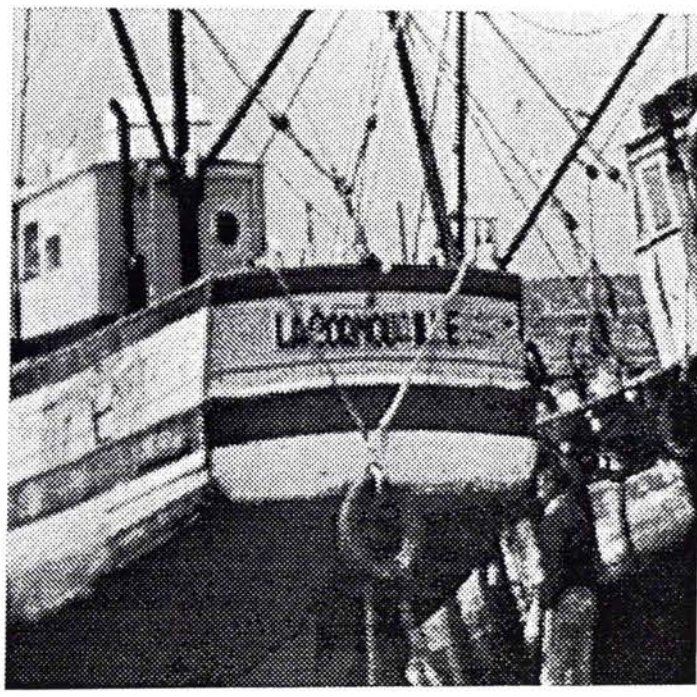
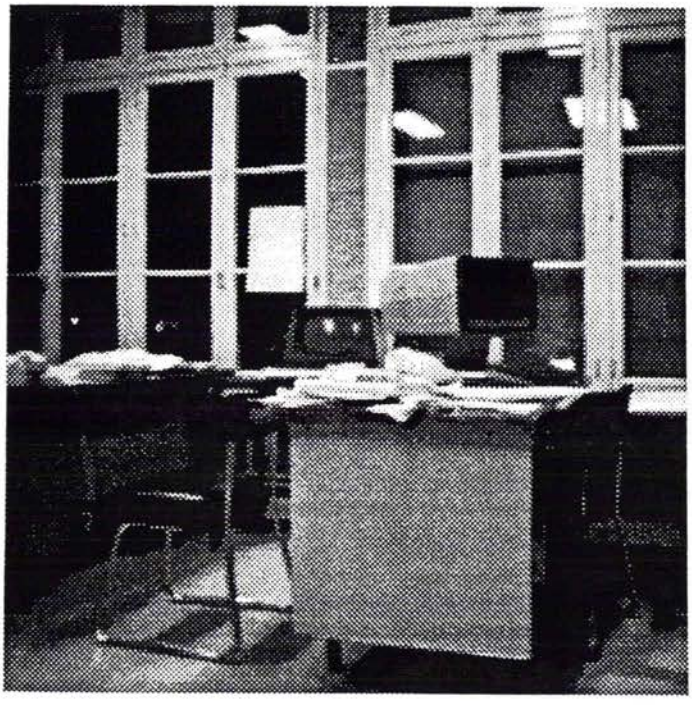


Fig 1.5. Evolution de la distorsion après chaque itération de l'algorithme de classification

—▲— Initialisation par la méthode de séparation maximum (128 vecteurs de dimension k = 4)

—o— Initialisation par la méthode du maximum d'entropie (128 vecteurs de dimension k = 4)





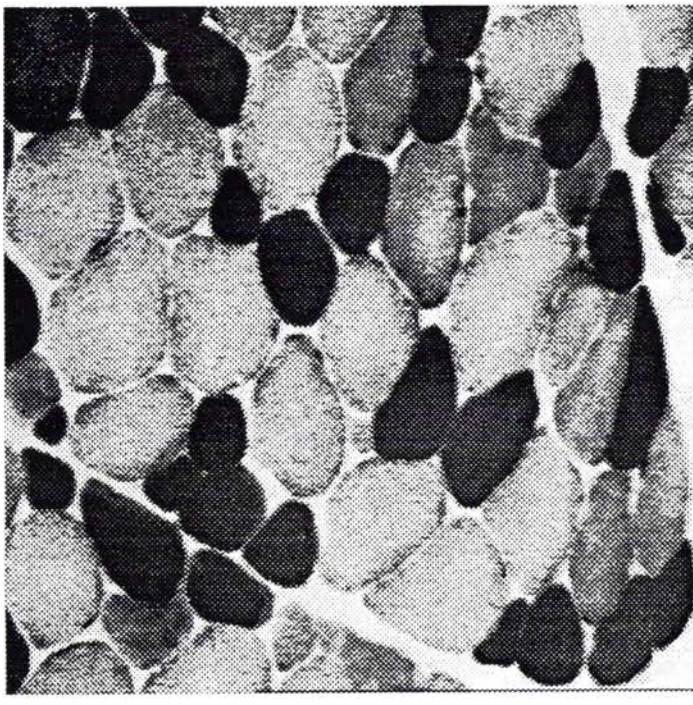


Fig 1.6. Images de référence constituant le training set - Taille 256x256 pixels sur 8 bits.
En haut à gauche : femme.ima.
En haut à droite : lacornou.ima.
En bas à gauche : bureau.ima.
En bas à droite : muscle.ima.

Fig 1.7. Image femme.ima codée à partir d'un dictionnaire de 16 vecteurs de taille k=2x2
En haut : dictionnaire initial obtenu par la méthode de séparation maximale.
En bas : dictionnaire initial obtenu par la méthode du maximum d'entropie.

Fig 1.8. Image femme.ima codée à partir d'un dictionnaire de 32 vecteurs de taille k=2x2
En haut : dictionnaire initial obtenu par la méthode de séparation maximale.
En bas : dictionnaire initial obtenu par la méthode du maximum d'entropie.

Fig 1.9. Image femme.ima codée à partir d'un dictionnaire de 64 vecteurs de taille k=2x2
En haut : dictionnaire initial obtenu par la méthode de séparation maximale.
En bas : dictionnaire initial obtenu par la méthode du maximum d'entropie.

Fig 1.10. Image femme.ima codée à partir d'un dictionnaire de 128 vecteurs de taille k=2x2
En haut : dictionnaire initial obtenu par la méthode de séparation maximale.
En bas : dictionnaire initial obtenu par la méthode du maximum d'entropie.

6. Références bibliographiques

[AHA-90] **S. C. Ahalt, A. K. Krishnamurthy, P. Chen and D. E. Melton**, "Competitive learning algorithms for vector quantization", Neural Networks, vol. 3, 1990, pp. 277-290.

[BAL-65] **G. H. Ball and D. J. Hall**, "Isodata - An iterative method of multivariate analysis and pattern classification", in Proc. IFIPS Congr., 1965.

[BEN-73] **J. P. Benzécri**, " L'analyse des données - Tome 1 La taxinomie - Tome 2 L'analyse des correspondances", Dunod, Paris, 1973.

[BUH-86] **Y. Buhler**, "Etude d'une méthode hiérarchique pour le codage à faible débit des images numériques", Mémoire de Maîtrise ès Sciences présenté à l'INRS, Mai 1986.

[DAV-90] **E. Davalo et P. Naïm**, "Des réseaux de neurones", Eyrolles, Paris, 1990.

[DID-71] **E. Diday**, "La méthode des nuées dynamiques", Rev. Stat. appliquée, vol. XIX, N° 2, 1971, pp. 19-34.

[FLA-79] **G. Flamembaum, J. Thiery et J. P. Benzécri**, "Agrégation en boules de rayon fixé et centres optimisés", Cahiers de l'analyse des données, vol. 4, N° 3, pp. 137-143.

[GER-82] **A. Gersho**, "On the structure of vector quantizers", IEEE Trans. Inform. Theory, vol. IT-28, Mar. 1982, pp. 157-166.

[GOL-86] **M. Goldberg and H. Sun**, "Image sequence coding using vector quantization", IEEE Trans. Commun., vol. COM-34, N° 7, July 1986, pp. 703-710.

[GRA-84] **R. M. Gray**, "Vector quantization", IEEE ASSP Magazine, vol. 1, N° 2, April 1984, pp. 4-29.

[HIL-77] **E. E. Hilbert**, "Cluster compression algorithm, a joint clustering data compression concept", Jet Propulsion Laboratory Publication, 1977, pp. 77-83.

[JAM-89] **M. Jambu**, "Exploration informatique et statistique des données", Dunod, Paris, 1989.

[KOH-84] **T. Kohonen**, "Self-organization and associative memory", Springer-Verlag, Berlin, 1984.

[KRI-90] **A. K. Krishnamurthy, S. C. Ahalt, D. E. Melton and P. Chen**, "Neural networks for vector quantization of speech and images", IEEE journal on selected areas in communications, vol. 8, N° 8, Oct. 1990, pp. 1449-1457.

[LEB-77] **L. Lebart, A. Morineau et N. Tabard**, "Techniques de la description statistique - Méthodes et logiciels pour l'analyse de grands tableaux", Dunod, Paris, 1977.

[LEB-89] **E. Le Bail et A. Mitiche**, "Quantification vectorielle d'images par le réseau neuronal de Kohonen", Traitement du signal, vol. 6, N° 6, 1989, pp. 529-539.

[LEE-90] **Tsu-Chang Lee and Allen M. Peterson**, "Adaptive vector quantization using a self-development neural network", IEEE journal on selected areas in communications, vol. 8, N° 8, Oct. 1990, pp. 1458-1471.

[LEM-91] **H. Lemberg**, "Les réseaux neuronaux artificiels", Langages & Systèmes - INFOPC, N° 71, Mai 1991, pp. 20-28.

[LER-81] **I. C. Lerman**, "Classification et analyse ordinale des données", Dunod, Paris, 1981.

[LIN-80] **Y. Linde, A. Buzo and R. M. Gray**, "An algorithm for vector quantizer design", IEEE Trans. Commun., vol. COM-28, Jan. 1980, pp. 84-95.

[LLO-82] **S. P. Lloyd**, "Least squares quantization in PCM", IEEE Trans. Inform. Theory, vol. IT-28, N° 2, Mar. 1982, pp. 129-137.

[MAC-67] **J. MacQueen**, "Some methods for classification and analysis of multivariate observations", Proc. of the fifth Berkeley Symposium on Math., Stat. and Prob., vol. 1, 1967, pp. 281-296.

[MAR-86] **J. P. Marescq**, "Etude de schémas de quantification vectorielle adaptative multiclasses. Application au codage de séquences d'images télévisuelles", Thèse de Docteur-Ingénieur, Université de Rennes I, Dec. 1986.

[MIC-85] **L. Miclet et M. Dabouz**, "Un vocodeur à classification: transmission de parole à très faible débit par quantification vectorielle du spectre", Actes du séminaire sur la quantification vectorielle pour le traitement de la parole, ENST, Paris, 14 Fev. 1985, pp. 53-90.

[MUR-82] **T. Murakami, K. Asai and E. Yamasaki**, "Vector quantizer of video signals", Electronics Letters, vol. 18, N° 23, Nov. 1982, pp. 1005-1006.

[NAS-88a] **N. M. Nasrabadi and Y. Feng**, "Vector quantization of images based upon the Kohonen self-organizing feature maps", in Proc. IEEE Int. Conf. on Neural Networks, San Diego, California, July 24-27, 1988, pp. 1101-1108.

[NAS-88b] **N. M. Nasrabadi and R. A. King**, "Image coding using vector quantization - A review", IEEE Trans. Commun., vol. 36, N° 8, August 1988, pp. 957-971.

[NYE-92] **A. Nyeck and A. Tosser-Roussey**, "A maximum entropy initialization technique for image coding vector quantizer design", IEE Electronics Letters, 30th Jan. 1992, vol. 28, N° 3, pp. 273-274.

[RAM-86] **B. Ramamurthi and A. Gersho**, "Classified vector quantization of images", IEEE Trans. Commun., vol. 34, N° 11, Nov. 1986, pp. 1105-1115.

[ROU-85] **M. Roux**, "Algorithmes de classification", Masson, Paris, 1985.

[SUN-84] **H. F. Sun and M. Goldberg**, "Image sequence coding using vector quantization", IEEE Proceedings of Intl. Communic. and Energy Conf., Montreal (Canada), 1984, pp. 266-269.

[WAR-83] **W. Warton Stephen**, "A generalized histogram clustering scheme for multidimensional image data", Pattern Recognition 16, N° 2, 1983, pp. 193-199.

[YUA-88] **G. Yuan and M. Goldberg**, "A sequential initialization technique for vector quantizer design", Pattern Recognition Letters 7, March 1988, pp. 157-161.

[ZHA-89] **Y. Zhao**, "Etude et réalisation d'un système de transmission d'images numériques à bas débit", Thèse d'Université, Nancy I, Oct. 1989.

Chapitre 2

ALGORITHMES RAPIDES DE QUANTIFICATION VECTORIELLE

1. Introduction

Après avoir mis en évidence dans le chapitre précédent les techniques de génération du dictionnaire, nous abordons dans ce chapitre la quantification vectorielle proprement dite, à savoir pour un vecteur source x, la recherche du représentant $Q(x)$ de x dans le dictionnaire selon le critère de la moindre distorsion.

Les algorithmes proposés sont analysés du point de vue de leur complexité en place mémoire ainsi qu'en nombre d'opérations arithmétiques. Quelques exemples de temps d'exécution sur station *SPARC 1* sont présentés, ce qui permet une étude comparative des différentes méthodes de recherche.

2. Le problème

La Quantification Vectorielle est une technique efficace de compression de données que ce soit pour des besoins de stockage ou de transmission. Toutefois, l'application de la quantification vectorielle aux problèmes de codage d'image en temps réel reste un exercice délicat compte tenu du volume considérable de données à traiter.

Rappelons que la quantification vectorielle consiste à représenter un vecteur x de dimension k dont les composantes sont des variables réelles, en un vecteur y de dimension k appartenant à un ensemble fini C de N vecteurs appelé **dictionnaire**.

$$R^k \text{ --------> } C$$
$$Q : x \text{ --------> } y = Q(x)$$

Le vecteur y de C est déterminé par la règle du plus proche voisin. Typiquement, pour tout vecteur x à coder, le vecteur de reconstruction $Q(x)$ est sélectionné dans le dictionnaire de façon à satisfaire la condition suivante :

$d(x , Q(x)) < d(x , y_j)$

où y_j est le *jième* vecteur du dictionnaire C

$j = 1,...,N \qquad y_j \neq Q(x)$.

L'expression $d(x , y)$ représente la distance entre le vecteur x et le vecteur y.

Pour des raisons de simplicité et de vitesse de traitement, nous utiliserons dans ce chapitre la distance de Minkowski comme mesure de base de la distorsion.

La distance de Minkowski utilisée est définie par :

$d(x , y) = \max_i |x_i - y_i| \qquad i = 1,...,k$

Le dictionnaire C est généré à partir d'un algorithme d'apprentissage. Nous avons adopté à cet effet, le conventionnel algorithme L.B.G qui converge comme on l'a vu au chapitre précédent vers un dictionnaire localement optimal.

Pour un vecteur x donné, le vecteur de reconstruction $Q(x)$ dans le dictionnaire est déterminé selon la règle de la distorsion minimum. Généralement, un algorithme de recherche séquentielle est appliqué pour sélectionner le meilleur représentant de x dans le dictionnaire, mais cette méthode bien que très simple à mettre en oeuvre est très consommatrice en temps CPU. Par conséquent, après une brève analyse de l'algorithme de recherche séquentielle, nous proposerons des méthodes de recherche plus rapides et nous discuterons de leur complexité dans le cadre de la quantification vectorielle appliquée au codage d'images.

3. La recherche séquentielle

C'est le plus simple des algorithmes de quantification vectorielle par son principe. Il s'agit de calculer la distance entre le vecteur à coder x et successivement tous les vecteurs c_i ($i = 1,...,N$) du dictionnaire. Le vecteur de reconstruction sera le vecteur dont la distance par rapport à x est minimum.

3.1. Algorithme

Soit $C = \{c_j ; j = 1,...,N\}$ un dictionnaire de N vecteurs où $c_j = (c_{ij} ; i = 1,...,k)$ est un vecteur de dimension k.

c_j est le *jième* vecteur du dictionnaire et c_{ij} est la *iième* composante de c_j.

Soit $x = (x_i ; i = 1,...,k)$ un vecteur de l'espace.

La recherche séquentielle du plus proche voisin $Q(x)$ de x dans le dictionnaire est effectuée par la procédure suivante :

(1) Calculer la distance $d_j = d(x , c_j)$ entre le vecteur x et le *jième* vecteur c_j (j = 1,...,N) du dictionnaire.

(2) Déterminer l'indice i tel que $d_i < d_j$ pour tout $j = 1,...,N \quad j \neq i$.

(3) c_i est le vecteur de reproduction de x dans le dictionnaire.
$Q(x) = c_i$.

3.2. Complexité

Le temps d'exécution d'un algorithme dépend directement du processeur utilisé. Ce temps va donc varier sensiblement selon que l'algorithme est exécuté sur une machine très puissante ou sur une autre aux performances moins élevées. Nous considérerons pour notre part comme indicateur de la complexité d'un algorithme de quantification vectorielle, le nombre d'opérations arithmétiques élémentaires nécessaires à l'exécution de l'algorithme, à savoir, les multiplications, additions, valeurs absolues et comparaisons, auxquelles il faudra ajouter les transferts mémoire et la taille mémoire occupée par les variables. Nous admettrons en outre qu'une soustraction est équivalente à une addition, et qu'une opération logique simple (ET logique, OU logique) est comptée comme équivalente à une comparaison.

La distance utilisée dans nos simulations est la distance de Minkowski définie plus haut.

Compte tenu de la mesure de distorsion choisie, la complexité de l'algorithme de recherche séquentielle peut donc être évaluée à :

k·N additions
k·N valeurs absolues
k·N-1 comparaisons.

En effet, le calcul d'une distance de Minkowski entre vecteurs de dimension k nécessite k additions, k valeurs absolues et k-1 comparaisons.

Sachant qu'une recherche exhaustive sur un dictionnaire de N vecteurs implique N calculs de distance et N-1 comparaisons par vecteur, on obtient bien une complexité de N·k additions, N.k valeurs absolues et N·(k-1) + (N-1) = N·k - 1 comparaisons.

4. Recherche par arbre binaire

4.1. Objectif de la méthode

La recherche séquentielle présentée précédemment est très simple à mettre en oeuvre, toutefois, elle est très lente car elle nécessite un parcours exhaustif de tout le dictionnaire pour le codage d'un vecteur de l'espace. Afin de diminuer le temps de recherche, on peut utiliser une procédure directement inspirée de la méthode de construction du dictionnaire par découpage [LIN-80].

A chaque étape du découpage de l'espace, les représentants intermédiaires de chaque partition sont mémorisés suivant un arbre binaire. Le codage d'un vecteur source s'effectue en parcourant cet arbre selon le principe des méthodes de recherche arborescentes : la comparaison avec les valeurs d'un noeud de l'arbre permet d'orienter la suite de la recherche dans l'arbre.

Concrètement, à chaque étage, le représentant fils de celui précédemment choisi fournissant la distorsion de reconstruction la plus faible est sélectionné. En suivant l'arbre ainsi constitué lors de la génération du dictionnaire, on obtient rapidement un représentant du vecteur à coder, mais ce n'est pas toujours le meilleur au sens de la moindre distorsion. En effet, l'optimisation du dictionnaire et plus précisément la représentation de toute cellule de l'espace par son barycentre à chaque étape de l'algorithme d'apprentissage LBG peut faire passer la frontière à un vecteur entre deux cellules voisines dont les barycentres n'ont pas le même ascendant dans l'arbre.

4.2. Complexité

L'algorithme de recherche dichotomique est très rapide: le codage d'un vecteur sur un dictionnaire de N prototypes ne nécessite que $2 \cdot \log_2 N$ calculs de distance et $\log_2 N$ comparaisons [GRA-82] .

Par contre la recherche est effectuée sur $2 \cdot (N-1)$ vecteurs, ce qui entraîne un accroissement de la taille mémoire nécessaire pour stocker le dictionnaire. D'autre part, le parcours du dictionnaire selon une structure arborescente engendre une erreur systématique au codage.

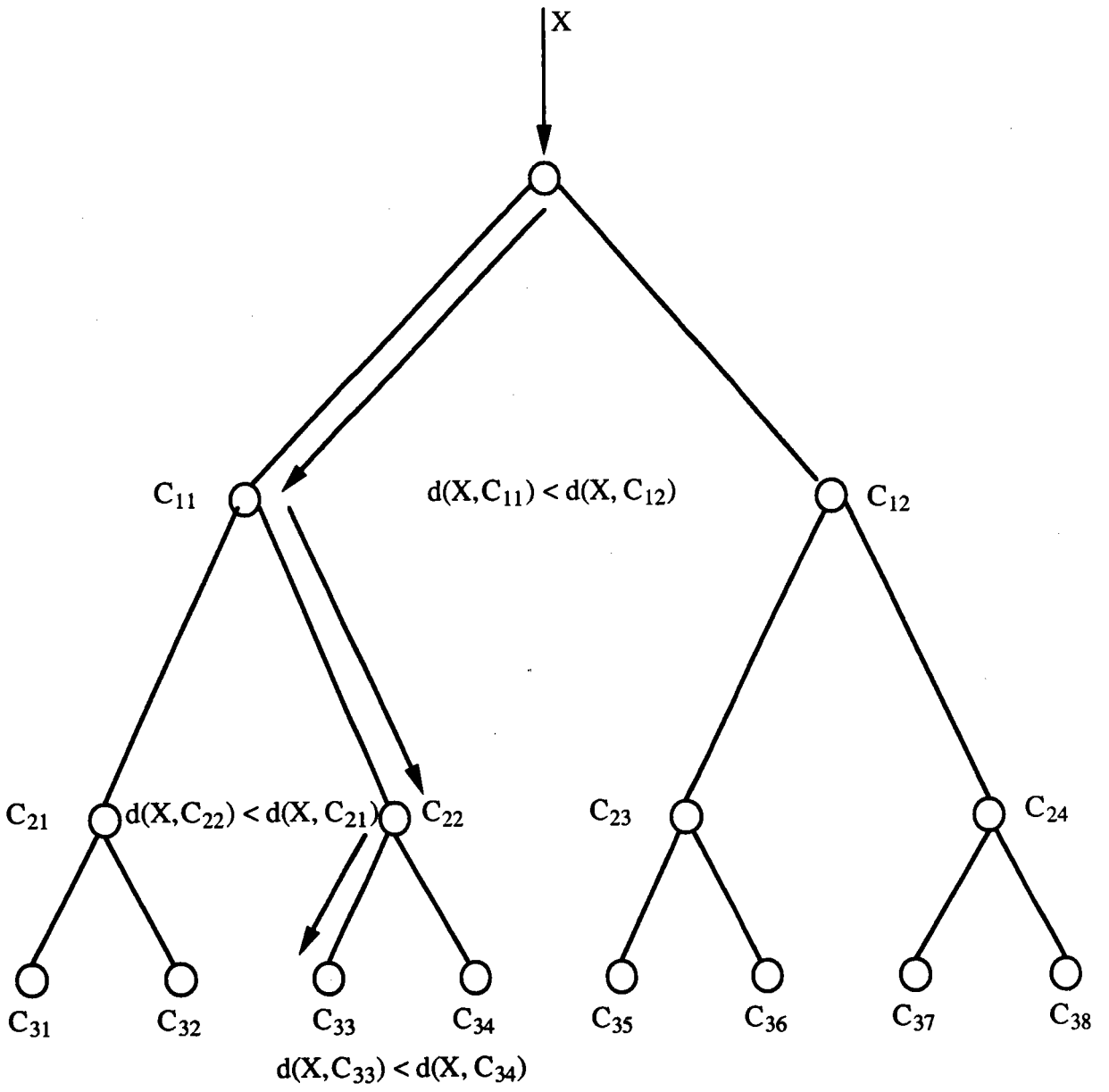


Fig 2.1 Recherche des Représentants par arbre binaire
C_{33} est le vecteur de reproduction de X

5. Dictionnaire ordonné par moyennes croissantes

Les performances des algorithmes rapides de quantification vectorielle sont liées au dictionnaire utilisé. D'une façon générale, une structure est introduite dans le dictionnaire par un calcul préliminaire; ainsi, la recherche du plus proche voisin se fait en utilisant cette structure de manière rapide. La rapidité de la décision qui dépend de la structure du dictionnaire peut se faire au détriment de la précision: autrement dit, on s'impose de chercher dans le dictionnaire un élément qui, s'il n'est pas toujours le plus proche voisin, en sera en moyenne proche, et ce dans un temps aussi bref que possible. La solution la plus retenue est celle présentée dans le paragraphe précédent, elle consiste à construire une structure d'arbre sur les vecteurs du dictionnaire, et par une suite descendante de comparaisons de la racine à une feuille, à extraire le vecteur de reproduction en un temps d'ordre de grandeur logarithmique par rapport à la taille du dictionnaire.

Une autre solution intuitive découle directement des caractéristiques statistiques de l'image: prenons l'exemple d'une image de 512x512 pixels partitionnée en blocs de taille 4x4. Calculons la moyenne et l'écart-type de chaque bloc et traçons l'histogramme des moyennes (Fig 2.2) et des écarts-type ainsi calculés (Fig 2.3).

On peut remarquer que l'histogramme des moyennes a une distribution plus régulière sur l'ensemble des points. La moyenne est donc une caractéristique essentielle d'un bloc image [KOH-88], cette caractéristique est exploitée en quantification vectorielle par l'algorithme suivant :

(1) Soit $C = \{c_j ; j = 1,...,N\}$ un dictionnaire de N vecteurs de dimension k.

Arrangeons C de telle sorte que les moyennes des vecteurs soient en ordre croissant.

(2) Soit $x = (x_i ; i = 1,...,k)$ un vecteur à coder.

Calculer la moyenne de x

$$m(x) = 1/k \sum_{j=1}^{k} x_j$$

(3) Déterminer le vecteur c_j (j = 1,...,N) du dictionnaire dont la moyenne se rapproche le plus de m(x)

(4) Faire une recherche séquentielle au voisinage de c_j.

La complexité de l'algorithme précédent est proportionnelle à la taille de l'intervalle de recherche. Pour un intervalle de taille $L < N$, la quantification vectorielle d'un élément ne nécessite que L distances à calculer. Par contre la taille de la mémoire requise augmente légèrement car il faut mémoriser le dictionnaire et les moyennes des vecteurs du dictionnaire.

Par ailleurs, la restriction de la recherche sur une partie du dictionnaire introduit une légère dégradation sur l'image codée, cette distorsion est d'autant plus importante que l'intervalle de recherche est réduit. Les resultats expérimentaux montrent que pour un dictionnaire de 256 vecteurs de taille 16, un intervalle de recherche égal au quart de la taille du dictionnaire était suffisant pour restituer une image de bonne qualité visuelle.

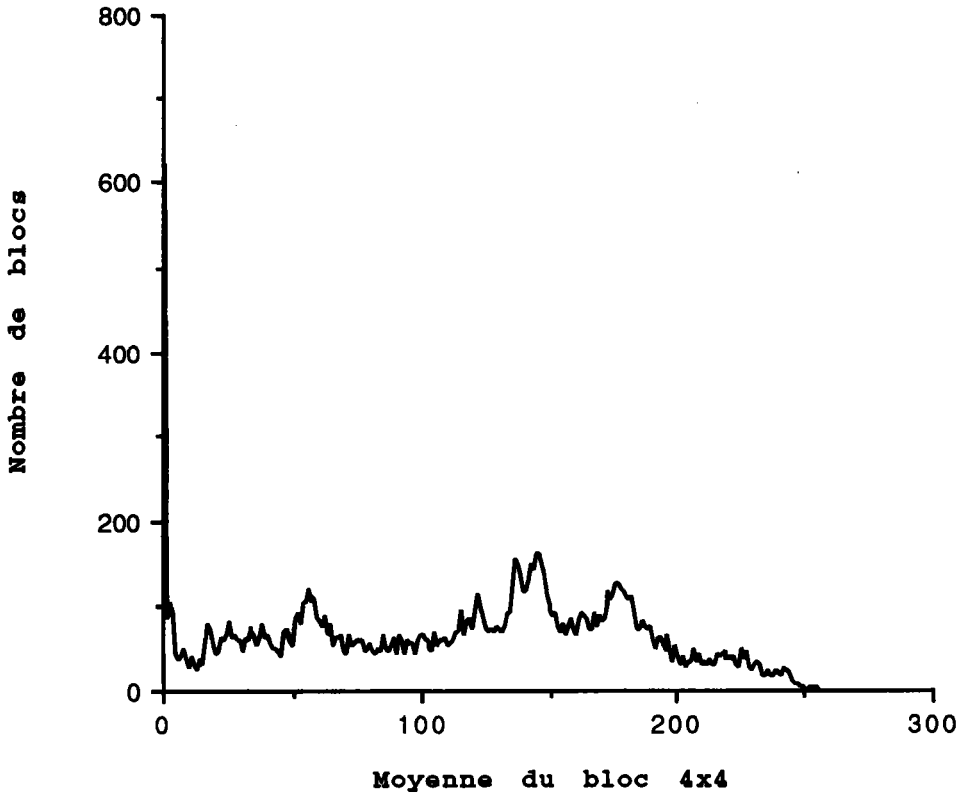


Fig 2.2 Histogramme des moyennes de blocs 4x4
Image 512x512 pixels

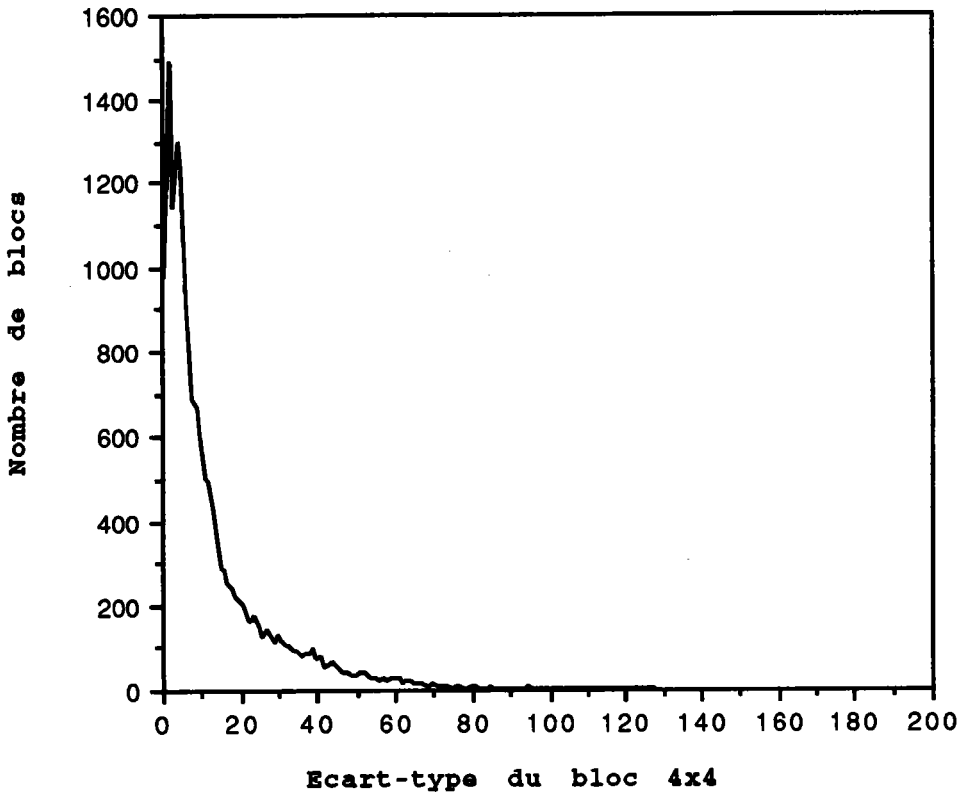


Fig 2.3 Histogramme des écarts-type de blocs 4x4
Image 512x512 pixels

6. Elimination par inégalité triangulaire

6.1. Principe

Les algorithmes de recherche présentés dans les paragraphes précédents permettent certes, de gagner du temps par rapport à la recherche séquentielle sur tout le dictionnaire mais ils présentent l'inconvénient majeur d'introduire une erreur systématique au codage. Afin de contourner cet handicap, il était utile d'introduire des techniques d'élimination mathématiques basées sur les propriétés métriques de la distance adoptée, de façon à réduire considérablement le nombre d'opérations nécessaires pour déterminer le plus proche voisin d'un vecteur x de l'espace.

Ces dernières années, plusieurs chercheurs se sont penchés sur le problème [FUN-75] , [KAM-85] , [VID-86] , [CHE-89] , et ont tous proposé des méthodes efficaces basées sur les propriétés de l'inégalité triangulaire, c'est-à-dire l'élimination du calcul de la distance entre le vecteur x à coder et les vecteurs du dictionnaire ne pouvant être plus proches de x que le plus proche voisin actuellement sélectionné par la procédure de recherche. L'objectif de ces travaux était d'introduire des règles d'élimination plus efficaces d'une part, et d'autre part d'organiser la recherche du plus proche voisin de façon à optimiser l'efficacité des règles existantes.

Avant d'aborder l'application des règles d'élimination géométrique en quantification vectorielle, un bref rappel sur les espaces métriques.

6.2. Quelques notions sur les espaces métriques

6.2.1. Normes

Soit E un espace vectoriel réel.

On appelle ***norme*** sur E toute fonction x ---> $\|x\|$, définie sur E, à valeurs réelles positives ou nulles, vérifiant les conditions suivantes :

(1) $\|x\| = 0$ si et seulement si $x = 0$

(2) $\|\lambda \cdot x\| = |\lambda| \cdot \|x\|$ quels que soient λ dans **R** et x dans E

(3) $\|x + y\| \leq \|x\| + \|y\|$ quels que soient x et y dans E.

Un espace vectoriel réel muni d'une norme s'appelle un *espace vectoriel normé.*

6.2.2. Exemples de normes

Exemple 1

Prenons $E = R^k$ un espace vectoriel réel de dimension k.

Soit $x = (x_i ; i = 1,...,k)$ un élément de E.

La fonction x ------> $(\sum_i (x_i)^2)^{1/2}$ $(i = 1,...,k)$ est une norme sur E.

Le seul point non évident est que

$(\sum_i (x_i + y_i)^2)^{1/2} \leq (\sum_i (x_i)^2)^{1/2} + (\sum_i (y_i)^2)^{1/2}$.

En élevant au carré l'inégalité précédente, ceci revient à vérifier que

$\sum_i (2.x_i.y_i) \leq 2.(\sum_i (x_i)^2)^{1/2} \cdot (\sum_i (y_i)^2)^{1/2}$

c'est-à-dire, en élevant de nouveau au carré, que

$(\sum_i (2.x_i.y_i))^2 \leq (\sum_i (x_i)^2) \cdot (\sum_i (y_i)^2)$.

Or cette inégalité résulte de l'égalité

$(\sum_i (x_i)^2) (\sum_i (y_i)^2) = (\sum_i (x_i.y_i))^2 + \sum_{ij} (x_i.y_j - x_j.y_i)^2$ $(i < j)$

Exemple 2

Prenons $E = R^k$ et posons

$\|x\| = \sum_i |x_i|$ $i = 1,...,k$

On peut facilement vérifier qu'on a ainsi défini une norme sur R^k, cette norme est différente de la norme de l'exemple précédent.

Exemple 3

Prenons toujours $E = R^k$ et posons

$\|x\| = \max_i(|x_i|)$ $i = 1,...,k$

Il est évident que $\|x\| = 0$ si et seulement si $x = 0$.

Par ailleurs, $\|\lambda.x\| = |\lambda| \cdot \|x\|$ quels que soient λ dans **R** et x dans E;

Enfin , soient $x = (x_i ; i = 1,...,k)$ et $y = (y_i ; i = 1,...,k)$ deux vecteurs de R^k;

on a $|x_i| \leq \|x\|$ et $|y_i| \leq \|y\|$ pour tout i

donc $|x_i + y_i| \leq |x_i| + |y_i| \leq \|x\| + \|y\|$ pour tout i

donc $\|x + y\| = \max_i(|x_i + y_i|) \leq \|x\| + \|y\|$ (i = 1,...,k)

Nous avons ainsi défini une autre norme sur R^k.

Relations entre les trois normes précédemment définies

Pour tout x dans R^k, appelons $N_1(x)$, $N_2(x)$ et $N_3(x)$, les normes définies respectivement dans les exemples 1, 2 et 3.

On a $N_3(x) \leq N_1(x) \leq N_2(x)$.

En effet,

$N_3(x)^2 = (x_i)^2$ pour un certain i = 1,...,k

donc $N_3(x)^2 \leq N_1(x)^2$

et par conséquent $N_3(x) \leq N_1(x)$.

D'autre part,

$N_1(x)^2 = \sum_i (x_i)^2 \leq (\sum_i |x_i|)^2 = N_2(x)^2$.

On a donc bien $N_1(x) \leq N_2(x)$.

6.2.3. Distances

Soit E un espace vectoriel réel de dimension k.

On appelle ***distance sur*** **E** toute fonction d, définie sur ExE, à valeurs réelles positives ou nulles, vérifiant les conditions suivantes :

(1) $d(x, y) = 0$ si et seulement si $x = y$.

(2) $d(x, y) = d(y, x)$ quels que soient x et y dans E.

(3) $d(x, z) \leq d(x, y) + d(y, z)$ quels que soient x, y et z dans E.

L'inégalité (3) s'appelle **l'inégalité triangulaire.**

Un ensemble muni d'une distance s'appelle un *espace métrique.*

Relation entre espace normé et espace métrique

Soit E un espace vectoriel normé.
Quels que soient x et y dans E, posons $d(x, y) = \|x - y\|$.

On peut montrer facilement que d est une distance sur E.
Ainsi, un espace vectoriel normé est automatiquement un espace métrique.

Tous les espaces vectoriels normés vus en **6.2.2** sont donc des espaces métriques.

On peut donc définir sur R^k les distances suivantes :

$d_1(x, y) = (\sum_i (x_i - y_i)^2)^{1/2}$ $\quad i = 1,...,k.$

Cette distance est communément appelée *distance euclidienne.*

De la même façon, on peut définir les *distances de Minkowski* suivantes :

$d_2(x, y) = \sum_i |x_i - y_i| \quad i = 1,...,k.$

$d_3(x, y) = \max_i (|x_i - y_i|) \quad i = 1,...,k.$

Les distances d_1, d_2 et d_3 sont les distances les plus couramment utilisées en quantification vectorielle pour le codage d'image. Nous avons en ce qui nous concerne, adopté la distance de Minkowski d_3 car elle est plus simple à calculer que la distance euclidienne par exemple et puis, compte tenu des propriétés des espaces normés et de l'inégalité entre les normes N_1, N_2, N_3 du paragraphe **6.2.2**, on a :

$$\mathbf{d_3(x, y) \leq d_1(x, y) \leq d_2(x, y)}$$

6.3. Utilisation de l'inégalité triangulaire en Quantification Vectorielle

Le but de ce paragraphe est de montrer comment les propriétés métriques de la distance peuvent être exploitées comme règles d'élimination pour les algorithmes de recherche du plus proche voisin.

Il est supposé que les vecteurs sont arrangés dans le dictionnaire dans un ordre quelconque, seules les relations géométriques existant entre ces vecteurs seront mises à contribution de façon à réaliser une recherche rapide et dynamique par approximations successives vers le vecteur de reproduction.

Ainsi pour un dictionnaire de N vecteurs, un calcul préliminaire des $N \cdot (N - 1)/2$ distances entre tous les vecteurs sera nécessaire. La connaissance de ces distances est l'information clé dans l'utilisation des règles d'élimination par inégalité triangulaire dont quelques formulations sont exposées ci-après.

Une forme classique de l'élimination par inégalité triangulaire est celle présentée par Vidal dans [VID-86] :

Soit $C = \{c_i ; i = 1,...,N\}$ un dictionnaire de taille N.

Soit x un vecteur de l'espace.

Supposons que l'actuel plus proche voisin de x dans le dictionnaire soit c_n,

on posera $d_n = d(x , c_n)$.

Supposons en outre qu'un vecteur c_s du dictionnaire ait été examiné et dont la distance à x est d_s.

Alors, éliminer le calcul de la distance entre x et tous les vecteurs c_p du dictionnaire tels que :

$$d(c_p , c_s) \geq d(x , c_s) + d(x , c_n) \quad (1)$$

$$d(c_p , c_s) \leq d(x , c_s) - d(x , c_n) \quad (2)$$

démonstration

d'après l'inégalité (1)

$$d(c_p , c_s) \geq d(x , c_s) + d(x , c_n)$$

donc

$d(x , c_n) \leq d(c_p , c_s) - d(x , c_s)$ (3).

Or

$d(c_p , c_s) \leq d(x , c_s) + d(x , c_p)$ (inégalité triangulaire).

L'inégalité (3) devient donc

$d(x , c_n) \leq d(x , c_s) + d(x , c_p) - d(x , c_s)$.

On a donc bien

$d(x , c_n) \leq d(x , c_p)$.

Par ailleurs, d'après l'inégalité (2),

$d(c_p , c_s) \leq d(x , c_s) - d(x , c_n)$.

Or

$d(x , c_s) \leq d(x , c_p) + d(c_p , c_s)$ (inégalité triangulaire).

L'inégalité (2) devient alors,

$d(c_p , c_s) \leq d(x , c_p) + d(c_p , c_s) - d(x , c_n)$

Et par conséquent,

$d(x , c_n) \leq d(x , c_p)$ cqfd.

Une autre forme simple et efficace de l'élimination par inégalité triangulaire est proposée dans [CHE-89] et se traduit par :

Soit $d(x , c_i)$ la distance entre le vecteur test x et le *iième* vecteur du dictionnaire.

S'il existe dans le dictionnaire un vecteur c_j tel que $d(c_i , c_j) > 2 \cdot d(x , c_i)$, alors

éliminer le calcul de la distance $d(x , c_j)$ car cette distance est toujours plus grande que $d(x , c_i)$.

démonstration

D'après l'inégalité triangulaire,

$d(c_i , c_j) \leq d(x , c_i) + d(x , c_j)$

D'où

$d(x, c_j) \geq d(c_i, c_j) - d(x, c_i)$ (1)

Supposons que $d(c_i, c_j) > 2 \cdot d(x, c_i)$

L'inégalité (1) devient alors,

$d(x, c_j) > 2 \cdot d(x, c_i) - d(x, c_i) = d(x, c_i)$.

On a donc bien

$d(x, c_j) > d(x, c_i)$ cqfd.

Il est clair que les règles d'élimination par inégalité triangulaire sont d'autant plus efficaces que le vecteur de reproduction se trouve parmi les premiers vecteurs du dictionnaire, le gain en nombre d'opérations est du reste évalué dans le prochain paragraphe.

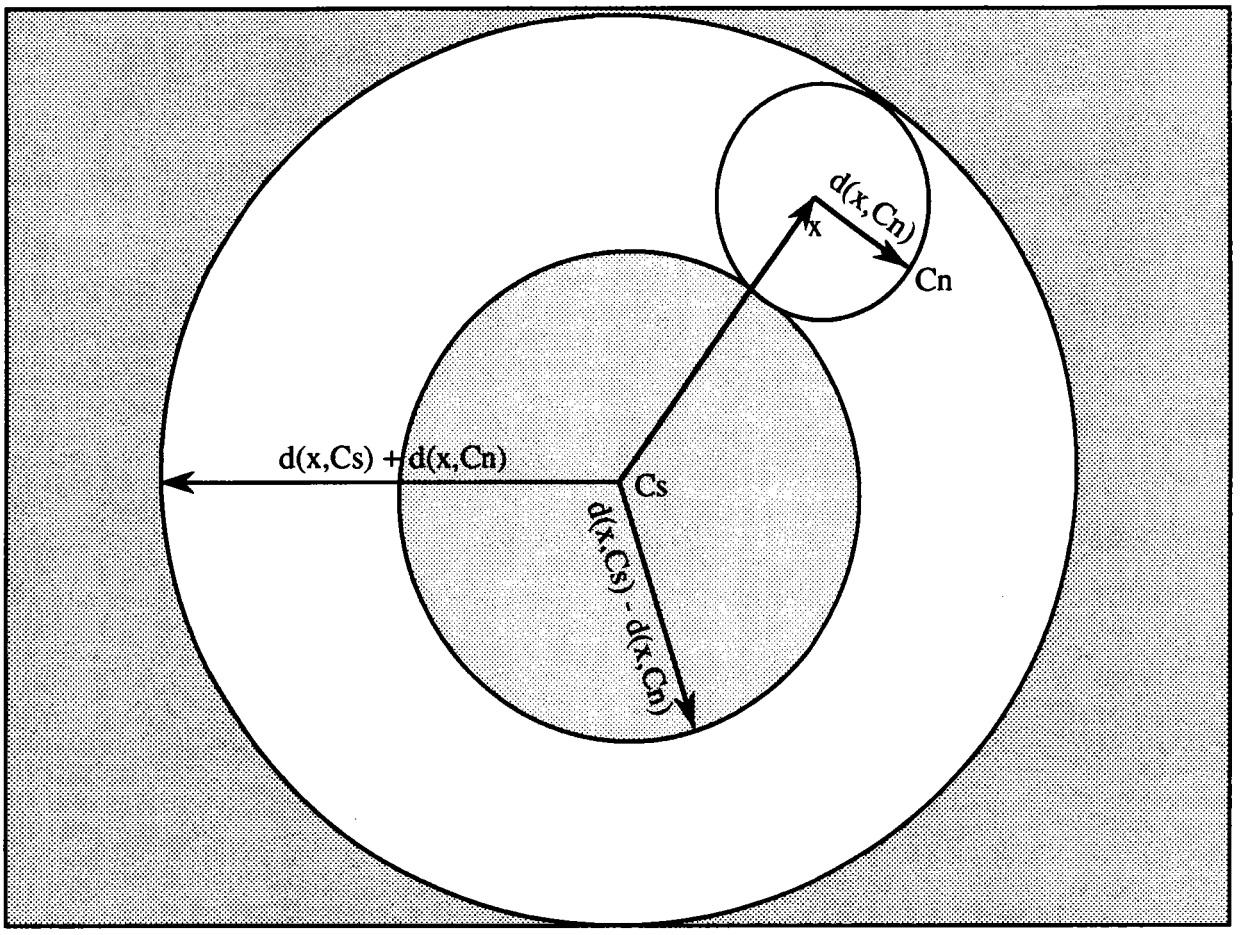


Fig 2.4 Elimination par Inégalité Triangulaire (E.I.T) dans le plan euclidien. Tous les vecteurs des zones sombres sont éliminés

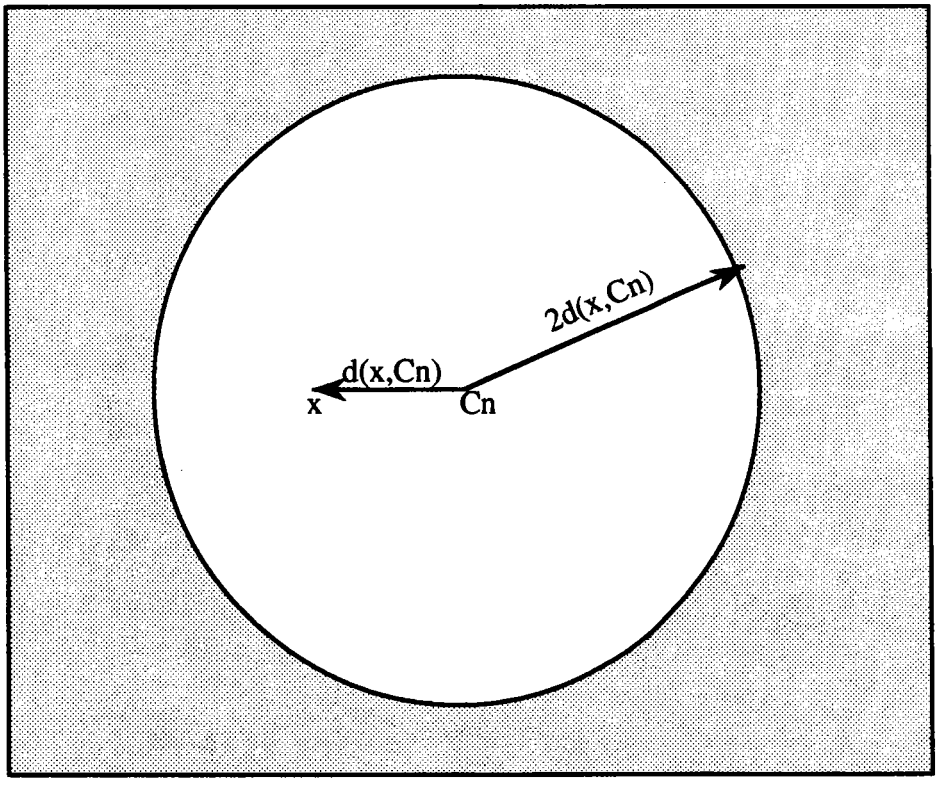


Fig 2.5 (E.I.T) : tous les vecteurs se trouvant dans les zones sombres ne peuvent être plus proches de x que l'actuel plus proche voisin Cn, ils peuvent donc être éliminés.

6.4. Analyse du nombre moyen d'opérations

Soit Op le nombre moyen d'opérations à effectuer pour le calcul d'une distance.
Pour un dictionnaire de taille N, l'algorithme de recherche séquentielle nécessite N·Op opérations pour la détermination du vecteur de reproduction.

D'une manière générale, soit C un dictionnaire de taille N et x un vecteur de l'espace.

Appelons p_k la probabilité que le vecteur de reproduction $Q(x)$ apparaisse à la place k ($k = 1,...,N$) dans le dictionnaire.

Le nombre moyen d'opérations nécessaires à la détermination de $Q(x)$ est donc évalué à :

$$M = Op\cdot(1.p_1 + 2.p_2 + ... + N.p_N)$$

avec $p_1 + p_2 + ... + p_N = 1.$

L'ensemble des probabilités p_i ($i = 1,...,N$) étant fixé, le nombre moyen d'opérations M est minimal lorsque

$$p_1 \geq p_2 \geq ..... \geq p_N \quad (1)$$

c'est-à-dire lorsque les éléments les plus recherchés sont en début de liste. Ainsi, quand on connaît les probabilités de recherche des éléments d'un dictionnaire, on sait dans quel ordre il faut les ranger pour optimiser la recherche.

Analysons quelques cas simples de distributions de probabilité afin de mettre en évidence le gain en nombre d'opérations lorsque la procédure de recherche exploite l'élimination par inégalité triangulaire. Nous supposerons en outre que les éléments du dictionnaire sont arrangés comme l'indique l'inégalité (1).

Si $p_1 = p_2 = ... = p_N = 1/N$, hypothèse où tous les éléments sont équiprobables, le nombre moyen d'opérations est alors :

$$M = Op \cdot \sum_i (i / N) = Op\cdot(N + 1)/2$$

Supposons maintenant qu'on ait la distribution de probabilité suivante :

$$p_1 = 1/2 \quad , \quad p_2 = (1/2)^2 \quad , \quad ..... \quad , \quad p_N = (1/2)^N$$

On peut montrer [KNU-73] que le nombre moyen d'opérations vérifie l'inégalité suivante :

$$M \leq Op\cdot(2 - 2^{1-N})$$

Le nombre moyen d'opérations est donc inférieur à 2·Op, ce qui est une nette amélioration par rapport à une distribution équiprobable, surtout lorsque le dictionnaire se compose d'un grand nombre de vecteurs.

Ainsi, plus les premiers vecteurs du dictionnaire ont une grande probabilité d'apparition, moins on a d'opérations à effectuer.

Le cas le plus favorable apparaît lorsque $\mathbf{p_1 = 1}$, ce qui est équivalent à $p_i = 0$ pour $i > 1$.

Le nombre moyen d'opérations est alors réduit à :

$\mathbf{M = Op \times p_1 = Op.}$

Evaluons Op dans le cas de la distance euclidienne. Nous prendrons x comme vecteur test et y un vecteur du dictionnaire.

$d^2(x, y) = \sum_i (x_i - y_i)^2 \qquad i = 1,...,k.$

En comptant une soustraction comme équivalente à une addition, le calcul de cette distance nécessite $2 \cdot k - 1$ additions et k multiplications. Nous montrons dans le paragraphe suivant qu'un calcul restreint sur une partie des composantes de x et y peut être suffisant pour éliminer le vecteur y.

7. Recherche par calcul partiel de la distance

7.1. Principe de la méthode

La notion de calcul partiel de la distance a été introduite par Cheng dans [CHE-85]. Elle a été reprise par Bei et Gray dans [BEI-85] et plus tard, par beaucoup d'autres chercheurs [PAL-89] , [CHE-89].
Le but de cette technique est d'éviter les calculs sur les autres composantes du vecteur si la valeur actuelle de la distance dépasse un certain seuil. Explicitons ceci dans le cas de la distance euclidienne :

Soit $C = \{c_j ; j = 1,...,N\}$ un dictionnaire de N vecteurs de dimension k.

Soit $x = (x_i ; i = 1,...,k)$ un vecteur à coder.

Il s'agit de trouver le vecteur de reproduction $Q(x)$ de x dans le dictionnaire.

Supposons que c_{min} (min < N) soit l'actuel plus proche voisin de x dans le dictionnaire.

Posons $d_{min} = d(x , c_{min})$.

Soit $c_l = (c_{li} ; i = 1,...,k)$ (l > min) le prochain vecteur à explorer dans le dictionnaire.

Si la somme partielle $\sum_i (x_i - c_{li})^2$ (i < k) est supérieure à d_{min}

alors on peut éliminer le vecteur c_l sans terminer le calcul de $d(x , c_l)$ car c_l n'est certainement pas le plus proche voisin de x dans le dictionnaire.

Cette simple technique appliquée à un large dictionnaire peut faire gagner jusqu'à 70 % des multiplications par rapport à la recherche séquentielle. L'algorithme de recherche par calcul partiel de la distance est du reste présenté ci-dessous dans le cas de la distance de Minkowski.

7.2. Algorithme

Soit $C = \{c_j ; j = 1,...,N\}$ un dictionnaire de N vecteurs de dimension k.

où $c_j = (c_{ji} ; i = 1,...,k)$ est le *jième* vecteur du dictionnaire et c_{ji} la *iième* composante de c_j.

Soit $x = (x_i ; i = 1,...,k)$ un vecteur à coder.

Il s'agit de trouver le vecteur de reproduction $Q(x)$ de x dans le dictionnaire selon le critère de la moindre distorsion.

La mesure de distorsion adoptée est la distance de Minkowski définie par:

$d(x, y) = \max_i |x_i - y_i| \qquad i = 1,...,k$

La structure de base de la recherche par calcul partiel de la distance est la suivante :

(1) ***Initialisations***

$d_{min} = d(x, c_1)$ — *d_{min} est la distance entre x et le 1er vecteur du dictionnaire*
$Q(x) = c_1$ — *Le vecteur de reproduction est le 1er vecteur du dictionnaire*
$i = 2$ — *La prochaine distance calculée sera* $d(x, c_2)$

(2) tant que $i \leq N$ faire

(3) $d = 0$ — *Initialisation de la distance* $d(x, c_i)$

(4) pour $j = 1$ à k faire

(5) $d = \max_j(d, |x_j - c_{ij}|)$ *Distance partielle sur les j premiers coefficients*

(6) si $d \geq d_{min}$ alors
sortie anticipée de la boucle *(4)*
aller à l'étape *(10)*

(7) fin si

(8) fin pour

(9) $d_{min} = d$ — *La distance minimale est* d_{min}
$Q(x) = c_i$ — *Le vecteur de reproduction est* c_{min}

(10) incrémenter i — *On explore le vecteur suivant dans le dictionnaire*

(11) fin tant que

Le gain en nombre d'opérations par rapport à la recherche séquentielle est quantifié par la sortie anticipée de la boucle à l'étape *(6)* de l'algorithme précédent, quand la condition $d > d_{min}$ est vérifiée. L'arrêt prématuré du calcul de la distance permet ainsi une économie substantielle du nombre d'opérations car les candidats peu probables à la sélection du plus proche voisin sont rejetés le plus tôt possible, juste après un calcul de distance sur les premières composantes des vecteurs.

Par ailleurs, à l'instar des techniques d'élimination par inégalité triangulaire, la recherche par calcul partiel de la distance est d'autant plus efficace que la condition $d > d_{min}$ est satisfaite le plus souvent possible. Ceci ne peut être réalisé que si d_{min} est minimal dès les premières phases de la recherche, autrement dit, lorsque les candidats les plus probables à la représentation du vecteur à coder x se trouvent aux premières places dans le dictionnaire.

Nous avons vu au paragraphe **6.4** que le nombre d'opérations était minimal lorsque les vecteurs les plus recherchés du dictionnaire se trouvaient en début de liste, arrangés dans l'ordre suivant :

$p_1 \geq p_2 \geq ..... \geq p_N$

où p_j est la probabilité pour que le vecteur de reproduction apparaisse à la place j dans le dictionnaire.

En pratique, on ne connaît pas les probabilités de recherche des différents éléments du dictionnaire, il est toutefois possible après apprentissage sur un grand nombre de vecteurs, de déterminer plus ou moins précisément ces probabilités: c'est la démarche proposée par Paliwal et Ramasubramanian dans [PAL-89].

Soit $S = \{x_j ; j = 1,...,T\}$ une séquence d'entraînement constituée de T vecteurs représentatifs de l'espace. On prendra T suffisamment grand, T -----> ∞.

Après apprentissage avec un algorithme approprié, un dictionnaire $C = \{c_j ; j = 1,...,N\}$ de N vecteurs est généré à partir de S. Les vecteurs c_j sont respectivement les barycentres de N cellules S_j de l'espace.

Chaque groupe de vecteurs S_j est défini par:

$S_j = \{x \in S ; c_j = Q(x)\}$,

autrement dit, S_j est l'ensemble des vecteurs de S dont le représentant dans le dictionnaire est c_j. Soit T_j le cardinal de S_j.

La probabilité p_j qu'un vecteur x de l'espace appartienne à la partition S_j est donc T_j / T.

La séquence d'entraînement étant suffisamment large, les probabilités p_i peuvent être considérées comme représentatives de l'espace à coder. Aussi, pouvons nous arranger les vecteurs du dictionnaire dans l'ordre des probabilités p_i décroissantes avec l'espoir légitime d'accroître l'efficacité de la recherche par calcul partiel de la distance.

7.3. Résultats des simulations et discussion

L'algorithme précédent a été testé sur des dictionnaires de différentes tailles obtenus à partir de l'algorithme d'apprentissage LBG.

Les simulations ont été faites pour des dictionnaires bien ordonnés ($p_1 \geq p_2 \geq ..... \geq p_N$) d'une part, et des dictionnaires arrangés dans un ordre défavorable ($p_1 \leq p_2 \leq ..... \leq p_N$) d'autre part.

Une image de 512x512 pixels sur 8 bits, divisée en blocs de 2x2 points est codée. Les résultats obtenus sont comparés avec ceux obtenus par l'algorithme de recherche séquentielle, la distance de Minkowski étant utilisée comme mesure de distorsion. La complexité des deux algorithmes est mesurée par le nombre moyen d'additions, valeurs absolues et comparaisons requises pour le codage d'un bloc image.

L'entropie d'un dictionnaire est désignée par :

$E = - \sum_i p_i \log_2(p_i) \qquad i = 1,...,N$

où N représente la taille du dictionnaire et p_i la probabilité qu'un vecteur de l'espace soit codé par le *iième* vecteur du dictionnaire. L'entropie du dictionnaire dépend donc de la distribution de probabilité de la source, nous verrons dans l'analyse des résultats que la valeur de l'entropie influence considérablement la réduction du nombre de calculs.

Les résultats des simulations sont du reste présentés dans les tableaux ci-dessous :

		Recherche séquentielle			Recherche par distance partielle			
N	E	add.	abs.	comp.	add.	abs.	comp.	%G
16	2.3	64	64	63	40.9	40.9	81.8	20.2
32	2.9	128	128	127	74.0	74.0	148.0	27.7
64	3.7	256	256	255	131.7	131.7	263.4	35.7
128	4.4	512	512	511	232.2	232.2	464.4	43.3
256	5.1	1024	1024	1023	396.5	396.5	793.0	51.6

Table 2.1
Comparaison des performances de l'algorithme de recherche séquentielle et de l'algorithme de recherche par calcul partiel de la distance lorsque les dictionnaires sont arrangés dans un ordre défavorable ($p_1 \leq p_2 \leq ..... \leq p_N$).

N : Taille du dictionnaire.
E : Entropie du dictionnaire.
add. : Nombre moyen d'additions requises pour coder un vecteur.
abs. : Nombre moyen de valeurs absolues requises pour coder un vecteur.
comp. : Nombre moyen de comparaisons requises pour coder un vecteur.
%G : Pourcentage moyen du nombre d'opérations arithmétiques économisées par rapport à l'algorithme de recherche séquentielle.

		Recherche séquentielle			Recherche par distance partielle			
N	E	add.	abs.	comp.	add.	abs.	comp.	%G
16	2.3	64	64	63	24.8	24.8	49.6	51.5
32	2.9	128	128	127	44.1	44.1	88.2	56.9
64	3.7	256	256	255	81.5	81.5	163.0	60.2
128	4.4	512	512	511	129.8	129.8	259.6	68.3
256	5.1	1024	1024	1023	222.0	222.0	444.0	72.9

Table 2.2
Comparaison des performances de l'algorithme de recherche séquentielle et de l'algorithme de recherche par calcul partiel de la distance lorsque les dictionnaires sont arrangés dans un ordre favorable ($p_1 \geq p_2 \geq ..... \geq p_N$).

Les résultats des tableaux mettent en évidence la vitesse de l'algorithme de recherche par calcul partiel de la distance ainsi que l'effet produit sur les performances de l'algorithme par un arrangement optimal ($p_1 \geq p_2 \geq ..... \geq p_N$) ou un arrangement critique ($p_1 \leq p_2 \leq ..... \leq p_N$) des vecteurs du dictionnaire.

Dans le cas d'un arrangement favorable du dictionnaire, les performances de l'algorithme pourront augmenter en fonction de l'entropie de la source. Une entropie faible signifie que les probabilités de recherche des vecteurs en début de liste sont très élevées par rapport à celles des vecteurs en fin de liste. Par conséquent, la probabilité de trouver le vecteur de reproduction dès les premières étapes de la recherche va s'accroître.

Par contre, lorsque les vecteurs du dictionnaire sont équiprobables, l'entropie de la source est maximale et vaut $\log_2(N)$. Un arrangement des vecteurs du dictionnaire dans le sens probabilités de recherche décroissantes n'entraîne donc aucune réduction significative en nombre de calculs.

En résumé, si on connaît les probabilités de recherche des vecteurs du dictionnaire, on sait dans quel ordre il faut ranger ces vecteurs pour optimiser la recherche. Mais en pratique, on ne connaît pas les probabilités de recherche des différents éléments d'une part, et d'autre part, supposons que le dictionnaire ait été effectivement organisé dans l'ordre $p_1 \geq p_2 \geq ..... \geq p_N$, et qu'il faille coder une image composée essentiellement de vecteurs ayant une probabilité de recherche p_N , l'algorithme de recherche par calcul partiel de la distance n'offre dans ce cas qu'un gain minimum en nombre d'opérations (cf. table 2.1). Par conséquent, si l'on doit effectuer un grand nombre de recherches, il est plus intéressant de faire évoluer progressivement le dictionnaire de sorte que les vecteurs les plus recherchés se retrouvent en tête; on emploiera alors le terme de recherche autoadaptative.

7.4. Recherche autoadaptative

Les méthodes de recherche autoadaptative consistent à réorganiser les vecteurs du dictionnaire au fur et à mesure des recherches, afin de minimiser le nombre d'opérations lors des recherches suivantes.

Une première possibilité est de comptabiliser en mémoire le nombre de fois que chaque élément est sélectioné et réarranger constamment les vecteurs du dictionnaire par ordre de fréquence d'occurrence décroissante. Les éléments tendent alors à se stabiliser en ordre décroissant de leurs probabilités de recherche ($p_1 \geq p_2 \geq ..... \geq p_N$) . Le dictionnaire est donc organisé de façon optimale pour toute recherche du type calcul partiel de la distance ou élimination par inégalité triangulaire.
Cette méthode a toutefois l'inconvénient d'être coûteuse en place mémoire à cause des compteurs.

On peut alors envisager d'autres algorithmes de recherche dynamique qui ne gardent aucune information sur la fréquence de recherche des éléments, et donc n'utilisent aucune mémoire supplémentaire. On pourrait par exemple après chaque recherche, faire progresser le vecteur sélectionné d'un rang vers la tête du dictionnaire autrement dit, chaque vecteur retenu est échangé avec le vecteur qui le précède dans la liste.

Appliquées au codage d'image, les deux méthodes de recherche autoadaptative précédentes exploitent la distribution statistique des blocs image, et les vecteurs du dictionnaire sont dynamiquement ordonnés en fonction de cette distribution, c'est-à-dire en ordre décroissant de leurs probabilités de recherche. Par contre, les méthodes citées ci-dessus n'exploitent pas directement les caractéristiques spatiales de l'image et notamment l'intercorrélation existant entre des blocs image contigus.

Forts de ce constat, nous proposons dans le prochain paragraphe un algorithme simple et rapide de quantification vectorielle exploitant la redondance spatiale de l'image.

7.5. Recherche autoadaptative exploitant la redondance spatiale de l'image

La méthode proposée est une technique simple et efficace de quantification vectorielle pour le codage d'image. L'algorithme exploite d'une part la distribution statistique des blocs de l'image, et d'autre part l'intercorrélation existant entre des blocs consécutifs.
La mise en oeuvre de la technique est tout aussi simple: il s'agit après chaque recherche, de placer le vecteur sélectionné en tête du dictionnaire [NYE-92].

Soit $C = \{c_j ; j = 1,...,N\}$ un dictionnaire de N vecteurs de dimension k.

Soit $x = (x_i ; i = 1,...,k)$ un vecteur de l'espace.

Pendant le codage du vecteur x, le dictionnaire est progressivement arrangé comme suit :

étape 1 $\quad c_1 \quad c_2 \quad c_3 \quad \ldots\ldots \quad c_{j-1} \quad c_j \quad c_{j+1} \quad \ldots\ldots \quad c_{N-1} \quad c_N$

Si le plus proche voisin $Q(x)$ de x est le vecteur c_j, alors les vecteurs du dictionnaire sont organisés comme l'indique l'étape 2 :

étape 2 $\quad c_j \quad c_1 \quad c_2 \quad c_3 \quad \ldots\ldots \quad c_{j-1} \quad c_{j+1} \quad \ldots\ldots \quad c_{N-1} \quad c_N$

Ainsi, le vecteur c_j est placé en tête de liste et tous les vecteurs précédant c_j sont décalés d'un rang vers la fin de la liste. Cette disposition des vecteurs du dictionnaire après chaque étape de la quantification vectorielle offre les avantages suivants :

- Les vecteurs les plus recherchés du dictionnaire se retrouvent progressivement en début de liste. Ceci peut être observé sur les figures 2.6. et 2.7 de ce chapitre.

Fig 2.6 représente un dictionnaire où les vecteurs sont ordonnés par moyennes croissantes. Les vecteurs les plus recherchés sont régulièrement répartis dans tout le dictionnaire.

Fig. 2.7 représente le même dictionnaire organisé progressivement par l'algorithme proposé: les vecteurs les plus recherchés se retrouvent en début de liste à la fin de la procédure de recherche.

- Par ailleurs, soit x_{j-1} le bloc image précédemment codé et $Q(x_{j-1})$ son représentant dans le dictionnaire.

Pour coder le bloc courant x_j, on commence par tester $Q(x_{j-1})$.
En effet, la redondance spatiale étant une caractéristique importante de l'image, des blocs image consécutifs ont de grandes chances d'être représentés par le même vecteur du dictionnaire.

Le dictionnaire étant dynamiquement ordonné comme le préconise la méthode proposée, la recherche par calcul partiel de la distance gagne en rapidité. Ceci est illustré par les tableaux 2.3 et 2.4.

Afin de mettre en évidence l'efficacité de la méthode, nous avons repris les mêmes images et les mêmes dictionnaires que ceux du paragraphe 7.3.
Les vecteurs des dictionnaires ont été ordonnés par probabilités de recherche croissantes ($p_1 \leq p_2 \leq \ldots\ldots\ldots \leq p_N$) d'une part, et décroissantes ($p_1 \geq p_2 \geq \ldots\ldots\ldots \geq p_N$) d'autre part. L'algorithme de recherche par calcul partiel de la distance a été utilisé, la distance de Minkowski étant la mesure de base de la dissimilarité. Les résultats obtenus ont alors été confrontés à ceux obtenus par arrangement progressif des dictionnaires.

		Algorithme 1				Algorithme proposé			
N	E	add.	abs.	comp.	%G	add.	abs.	comp.	%G
16	2.3	40.9	40.9	81.8	20.2	23.9	23.9	47.8	53.3
32	2.9	74.0	74.0	148.0	27.7	41.4	41.4	82.8	59.6
64	3.7	131.7	131.7	263.4	35.7	73.5	73.5	147.0	64.1
128	4.4	232.2	232.2	464.4	43.3	127.0	127.0	254.0	69.0
256	5.1	396.5	396.5	793.0	51.6	211.4	211.4	422.8	74.2

Table 2.3
Comparaison des performances de l'algorithme de recherche par calcul partiel de la distance sur un dictionnaire dont les vecteurs sont ordonnés par probabilités de recherche croissantes ($p_1 \leq p_2 \leq \ldots\ldots\ldots \leq p_N$) et sur le même dictionnaire arrangé dynamiquement par la méthode proposée.

N : Taille du dictionnaire.
E : Entropie du dictionnaire.
add. : Nombre moyen d'additions requises pour coder un vecteur.
abs. : Nombre moyen de valeurs absolues requises pour coder un vecteur.
comp. : Nombre moyen de comparaisons requises pour coder un vecteur.
%G : Pourcentage moyen du nombre d'opérations arithmétiques économisées par rapport à l'algorithme de recherche séquentielle.
Algorithme 1 : Algorithme de recherche par calcul partiel de la distance.

		Algorithme 1				Algorithme proposé			
N	E	add.	abs.	comp.	%G	add.	abs.	comp.	%G
16	2.3	24.8	24.8	49.6	51.5	23.8	23.8	47.6	53.5
32	2.9	44.1	44.1	88.2	56.9	41.2	41.2	82.4	59.8
64	3.7	81.5	81.5	163.0	60.2	72.9	72.9	145.8	64.4
128	4.4	129.8	129.8	259.6	68.3	126.2	126.2	252.4	69.2
256	5.1	222.0	222.0	444.0	72.9	209.7	209.7	419.4	74.4

Table 2.4
Comparaison des performances de l'algorithme de recherche par calcul partiel de la distance sur un dictionnaire dont les vecteurs sont ordonnés par probabilités de recherche décroissantes ($p_1 \geq p_2 \geq \ldots\ldots\ldots \geq p_N$) et sur le même dictionnaire arrangé dynamiquement par la méthode proposée.

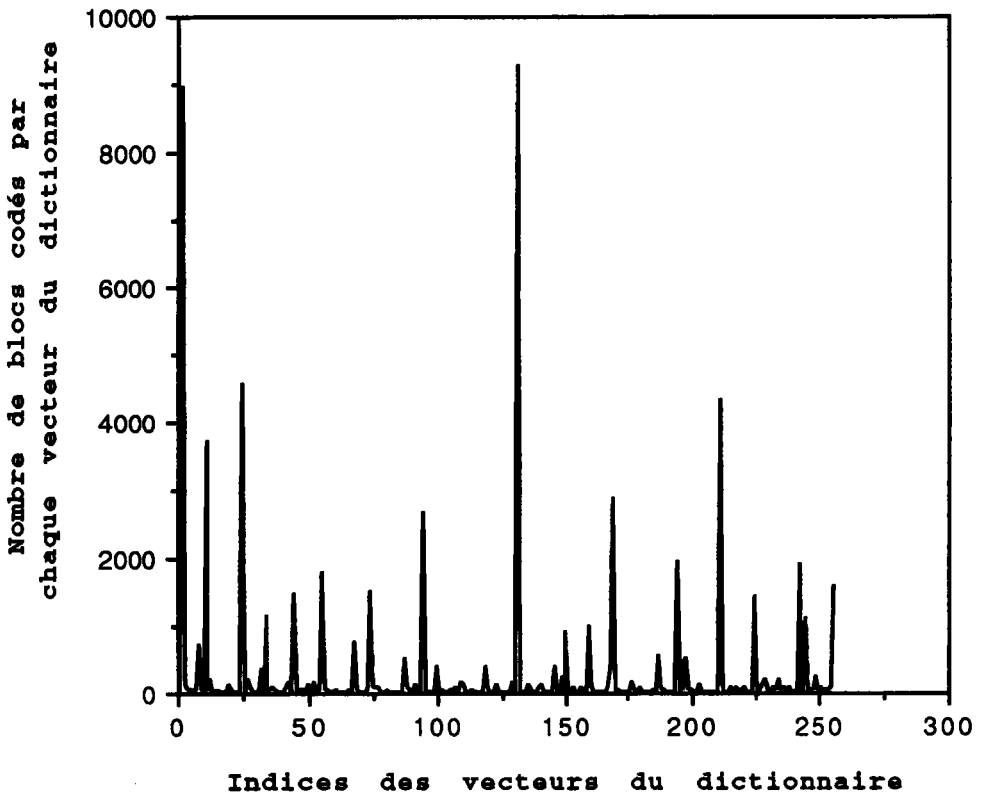


Fig 2.6 Fréquence d'occurrence des éléments dans un dictionnaire de 256 vecteurs de taille k = 4 , classés par moyennes croissantes

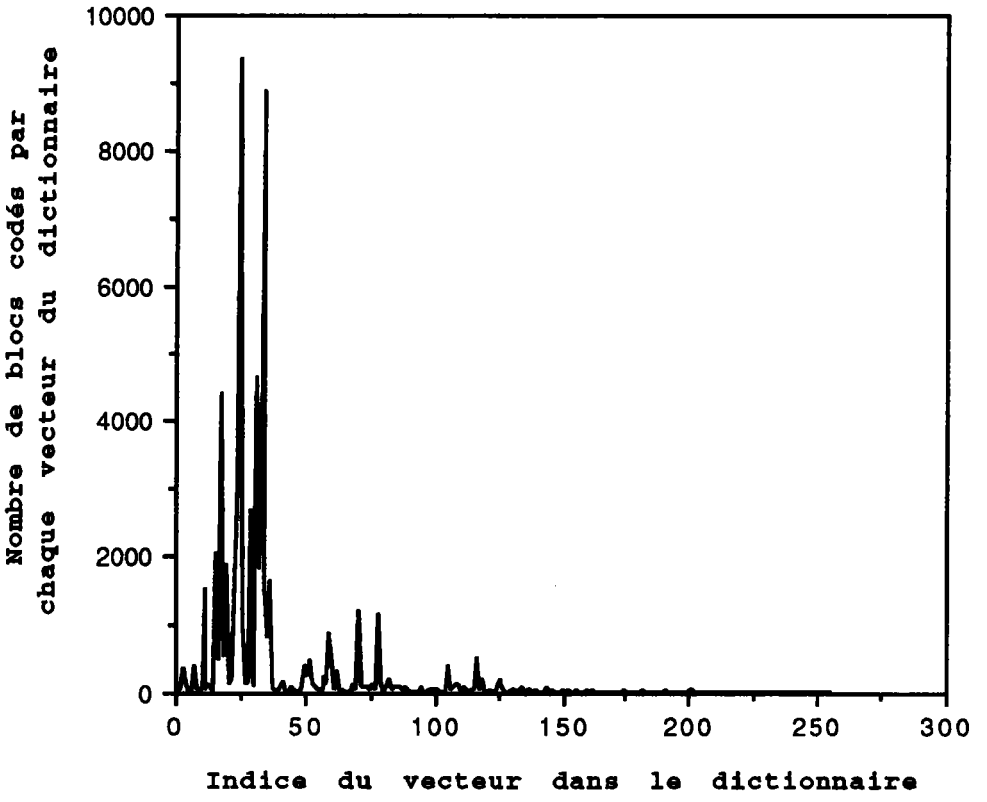


Fig 2.7 Fréquence d'occurrence des éléments dans un dictionnaire de 256 vecteurs de taille k = 4, obtenu après auto-organisation dynamique.
Les vecteurs les plus recherchés se retrouvent progressivement en tête de liste.

7.5.1. Discussion et perspectives

Les résultats présentés dans les tableaux précédents montrent bien que l'arrangement adaptatif proposé procure de meilleures performances que tout autre arrangement du dictionnaire dans le cadre d'une recherche par calcul partiel de la distance. Ces performances doivent toutefois être pondérées par deux facteurs: l'entropie du dictionnaire et la taille des blocs image utilisés.

Mettons nous dans l'hypothèse où les vecteurs du dictionnaire sont organisés de telle sorte que $(p_1 \geq p_2 \geq \ldots\ldots\ldots \geq p_N)$, où p_j est la probabilité qu'un bloc donné de l'image soit codé par le *jième* vecteur du dictionnaire. Il a été vu au paragraphe **7.3** que les performances de l'algorithme de recherche par calcul partiel de la distance étaient d'autant plus élevées que l'entropie de la source était faible, et justement, l'arrangement dynamique proposé tend à organiser progressivement les vecteurs du dictionnaire dans l'ordre $(p_1 \geq p_2 \geq \ldots\ldots\ldots \geq p_N)$.

La redondance spatiale de l'image est mise en évidence par la taille des blocs: plus les blocs adjacents de l'image sont de petite dimension, plus ils sont corrélés. Dans le cas extrême c'est-à-dire si le bloc est réduit au pixel, cette corrélation est encore plus forte.
Les résultats des tableaux précédents ont été obtenus pour des images partitionnées en blocs de dimension 2x2, des blocs de taille plus élevée auraient conduit à d'autres résultats, nous le verrons du reste dans le prochain paragraphe.

Enfin, on peut observer sur les tableaux 2.3 et 2.4 que pour un dictionnaire donné, le nombre d'opérations nécessaires à la quantification vectorielle est pratiquement constant et donc indépendant de l'organisation initiale des vecteurs dudit dictionnaire. L'arrangement dynamique proposé permet ainsi en adoptant la recherche par calcul partiel de la distance, de trouver rapidement le meilleur représentant $Q(x)$ d'un vecteur x de l'espace en un temps quasiment constant. Une version optimisée de l'algorithme, prenant en compte l'élimination par inégalité triangulaire est présentée dans la suite de ce chapitre.

8. Algorithmes rapides de QV exploitant les caractéristiques statistiques de l'image

8.1. Introduction

Nous présentons dans cette partie, deux algorithmes rapides de quantification vectorielle pour le codage d'image. Ces algorithmes exploitent la forte corrélation existant entre blocs image contigus d'une part, et d'autre part tirent profit des règles d'élimination par inégalité triangulaire aussi bien que de la recherche par calcul partiel de la distance pour minimiser les temps de recherche. Pour un dictionnaire de 256 vecteurs de dimension k = 4, les résultats des simulations montrent une réduction de plus de 90% du nombre d'opérations CPU par rapport au classique algorithme de recherche séquentielle.

8.2. Caractérisation de la redondance spatiale de l'image

Partons d'une image de 512x512 pixels divisée en blocs de taille 2x2, puis 4x4, puis 8x8. Calculons ensuite les distances entre les blocs consécutifs de l'image et comptons le nombre de blocs relatifs à chaque distance calculée. Les figures 2.8, 2.9 et 2.10 représentent les histogrammes obtenus. Ces histogrammes caractérisent l'intercorrélation existant entre blocs consécutifs. On observe clairement que les pics de ces histogrammes sont concentrés à l'origine, ce qui signifie qu'une grande partie de blocs contigus sont fortement corrélés. Ce phénomène est d'autant plus sensible que les blocs sont de petite dimension.

8.3. Algorithme rapide de recherche autoadaptative par arrangement progressif du dictionnaire

8.3.1. Position du problème

L'algorithme proposé est une extension de la technique exposée au paragraphe **7.5.** Il s'agit après chaque recherche, d'organiser dynamiquement le dictionnaire de façon à placer le vecteur sélectionné en tête de liste [NYE-92]. L'algorithme incorpore en outre les règles d'élimination par inégalité triangulaire pour rejeter les vecteurs improbables à la sélection du plus proche voisin. Une analyse détaillée des résultats obtenus pour différents cas de figure sera ensuite présentée.

8.3.2. Algorithme

Soit $C = \{c_j ; j = 1,...,N\}$ un dictionnaire de N vecteurs de dimension k.

où $c_j = (c_{ji} ; i = 1,...,k)$ est le *jième* vecteur du dictionnaire et c_{ji} la *iième* composante de c_j.

Soit $x = (x_i ; i = 1,...,k)$ un vecteur à coder.

Il s'agit de trouver le vecteur de reproduction $Q(x)$ de x dans le dictionnaire suivant un critère de distorsion minimale.

La mesure de distorsion adoptée est la distance de Minkowski définie par:

$$d(x, y) = \max_i |x_i - y_i| \qquad i = 1,...,k$$

La structure de base de l'algorithme rapide de recherche proposé est décrite par la procédure suivante :

Phase 1

(1) ***Initialisations***

$d_{min} = d(x , c_1)$ — *d_{min} est la distance entre x et le 1er vecteur du dictionnaire*
$c_{min} = c_1$ — *Le vecteur de reproduction est le 1er vecteur du dictionnaire*
$i = 2$ — *La prochaine distance calculée sera* $d(x , c_2)$

(2) tant que $i \leq N$ faire

(3) *Elimination des vecteurs peu probables par inégalité triangulaire*

si $d(c_{min} , c_i) < 2.d_{min}$ alors

(4) *Calcul partiel de la distance entre le vecteur source x et le vecteur c_i du dictionnaire non éliminé par inégalité triangulaire*

$d = 0$ — *Initialisation de la distance* $d(x , c_i)$

(5) pour $j = 1$ à k faire

$d = \max_j (d , |x_j - c_{ij}|)$ — *Distance partielle sur les j premiers coefficients*

(6) si $d \geq d_{min}$ alors
sortie anticipée de la boucle ***(5)***
aller à l'étape ***(8)***.
fin si

fin pour

(7) $d_{min} = d$ — *L'actuelle distance minimale est* d_{min}
$min = i$ — *L'actuel vecteur de reproduction est à la position* i

fin si

(8) incrémenter i

fin tant que

(9) $Q(x) = c_{min}$ — *Le vecteur de reproduction est* c_{min}

Phase 2

Organiser le dictionnaire C de telle sorte que le vecteur de reproduction c_{min} soit placé en tête de liste alors que tous les vecteurs précédant c_{min} sont décalés d'un rang vers la fin de la liste.

(1) pour $i = 2$ à min faire
c_i <------ c_{i-1} — *Tous les vecteurs précédant c_{min} sont décalés d'un rang vers la fin*
fin pour

(2) $c_1 = Q(x)$ — *Le vecteur de reproduction $Q(x)$ est placé en tête de liste*

Phase 3

Tester si tous les vecteurs source sont codés.
Si oui, terminer l'algorithme, sinon entrer le prochain vecteur x à coder et aller à la ***phase 1***

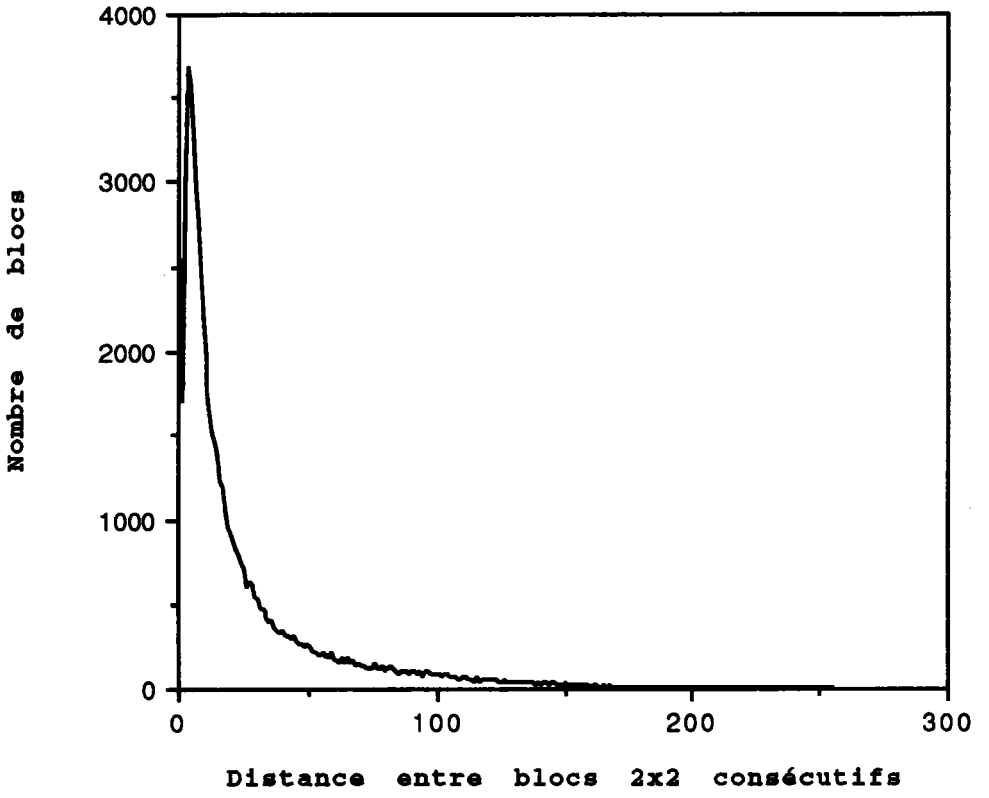


Fig 2.8 Histogramme de distances entre blocs 2x2 consécutifs
Image 512x512 pixels

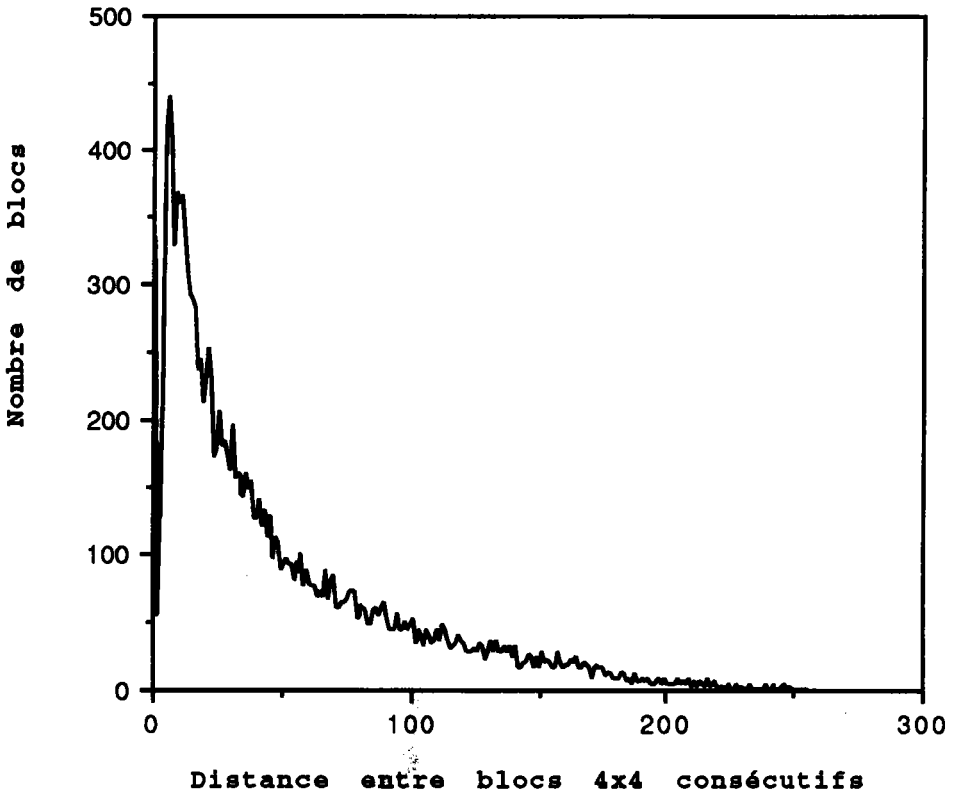


Fig 2.9 Histogramme de distances entre blocs 4x4 consécutifs
Image 512x512 pixels

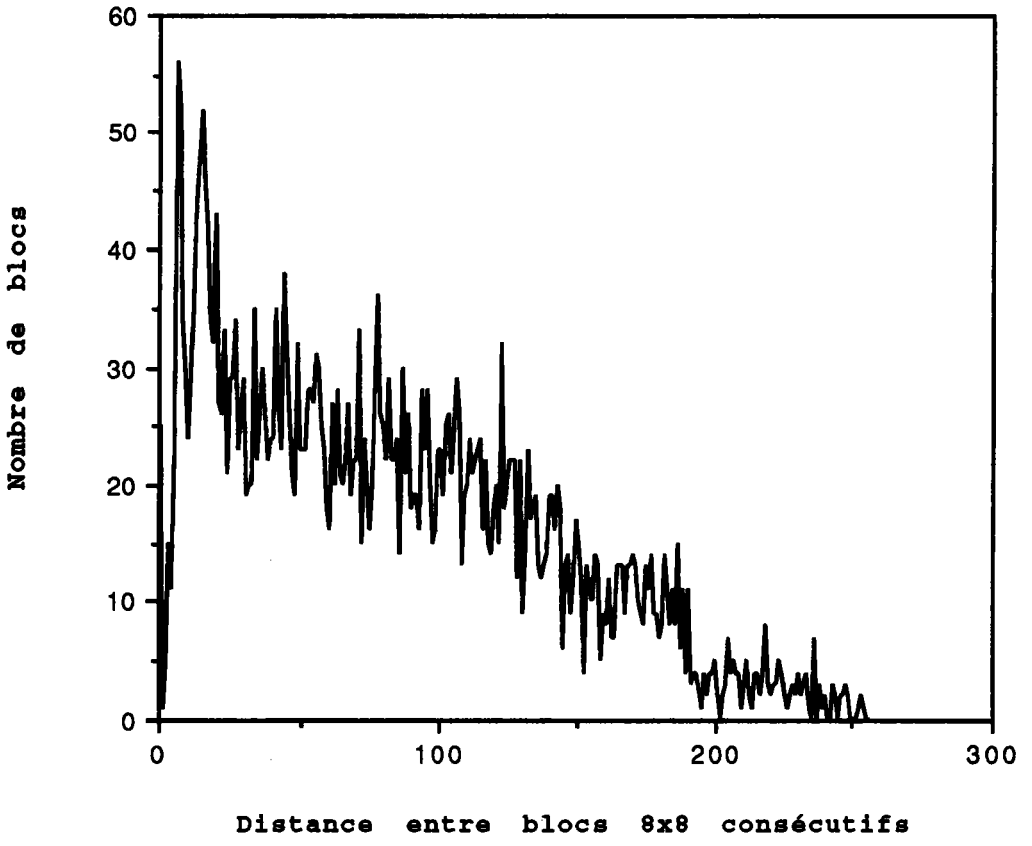


Fig 2.10 Histogramme de distances entre blocs 8x8 consécutifs
Image 512x512 pixels

8.3.3 Résultats des simulations et discussion

Afin d'évaluer les performances de l'algorithme précédent, nous avons généré des dictionnaires de différentes tailles: 16, 32, 64, 128 et 256. Nous avons en outre choisi une image test de taille 512x512 pixels sur 8 bits, partitionnée successivement en blocs de 2x2, puis 4x4, pour le codage. Les performances de l'algorithme sont comparées à celles de l'algorithme de recherche séquentielle d'une part, et d'autre part, afin de ressortir le gain de performances dû à l'arrangement adaptatif du dictionnaire, les résultats sont comparés à ceux obtenus avec l'algorithme optimisé de recherche par calcul partiel de la distance, les vecteurs du dictionnaire étant respectivement ordonnés par fréquence d'occurrence croissante ($p_1 \leq p_2 \leq \ldots\ldots\ldots \leq p_N$), et décroissante ($p_1 \geq p_2 \geq \ldots\ldots\ldots \geq p_N$). Nous entendons par optimisation de la méthode de recherche par calcul partiel de distance, l'incorporation dans l'algorithme des règles d'élimination par inégalité triangulaire. Tous ces résultats sont résumés dans les tableaux ci-dessous, la distance de Minkowski étant la mesure de base de la distorsion.

La légende suivante sera adoptée pour une compréhension plus facile des tableaux:

N : Taille du dictionnaire.
E : Entropie du dictionnaire.
add. : Nombre moyen d'additions requises pour coder un vecteur.
abs. : Nombre moyen de valeurs absolues requises pour coder un vecteur.
comp. : Nombre moyen de comparaisons requises pour coder un vecteur.
%G : Pourcentage moyen du nombre d'opérations arithmétiques économisées par rapport à l'algorithme de recherche séquentielle.
Algorithme 1 : Algorithme optimisé de recherche par calcul partiel de la distance.

QV pour des blocs image de taille k = 2x2 pixels.

		Recherche séquentielle			Algorithme proposé			
N	E	add.	abs.	comp.	add.	abs.	comp.	%G
16	2.3	64	64	63	6.5	6.5	22.4	83.6
32	2.9	128	128	127	8.1	8.1	40.8	87.3
64	3.7	256	256	255	10.9	10.9	63.6	90.6
128	4.4	512	512	511	14.4	14.4	116.8	92.2
256	5.1	1024	1024	1023	20.6	20.6	214.3	93.3

Table 2.5
Comparaison des performances de l'algorithme de recherche séquentielle et de l'algorithme proposé. Le dictionnaire est celui obtenu directement après la séquence d'apprentissage.

N	E	Algorithme 1 add.	abs.	comp.	%G	Algorithme proposé add.	abs.	comp.	%G
16	2.3	29.4	29.4	62.5	41.1	6.6	6.6	22.8	83.4
32	2.9	45.6	45.6	102.3	53.3	8.4	8.4	42.1	86.9
64	3.7	72.3	72.3	161.7	63.0	11.4	11.4	65.2	90.3
128	4.4	103.8	103.8	251.0	72.5	15.6	15.6	118.7	91.9
256	5.1	149.8	149.8	415.9	78.9	22.3	22.3	218.6	93.0

Table 2.6
Comparaison des performances de l'algorithme proposé et de l'algorithme optimisé de recherche par calcul partiel de distance, les vecteurs du dictionnaire étant ordonnés par fréquence d'occurrence croissante ($p_1 \leq p_2 \leq \ldots\ldots\ldots \leq p_N$).

N	E	Algorithme 1 add.	abs.	comp.	%G	Algorithme proposé add.	abs.	comp.	%G
16	2.3	7.2	7.2	23.5	82.4	6.4	6.4	22.2	83.8
32	2.9	10.1	10.1	39.7	86.3	7.8	7.8	39.7	87.7
64	3.7	13.4	13.4	67.4	89.5	10.5	10.5	62.4	90.8
128	4.4	17.1	17.1	121.0	91.6	14.1	14.1	114.6	92.3
256	5.1	23.6	23.6	226.7	92.7	20.2	20.2	211.3	93.4

Table 2.7
Comparaison des performances de l'algorithme proposé et de l'algorithme optimisé de recherche par calcul partiel de distance, les vecteurs du dictionnaire étant ordonnés par fréquence d'occurrence décroissante ($p_1 \geq p_2 \geq \ldots\ldots\ldots \geq p_N$).

Comparaison des temps d'exécution sur station *SPARC1* lors du codage d'une image 512x512 pixels sur 8 bits (65536 vecteurs), avec un dictionnaire de 256 vecteurs.

- Recherche séquentielle avec distance euclidienne : **24 min 25 sec.**
- Recherche séquentielle avec distance de Minkowski : **7 min 40 sec.**
- Recherche optimisée par calcul partiel de distance, les vecteurs du dictionnaires étant arrangés par probabilité de recherche croissante ($p_1 \leq p_2 \leq \ldots\ldots\ldots \leq p_N$) : **2 min 28 sec.**
- Recherche optimisée par calcul partiel de distance, les vecteurs du dictionnaire étant arrangés par probabilité de recherche décroissante ($p_1 \geq p_2 \geq \ldots\ldots\ldots \geq p_N$) : **1 min 4 sec.**
- Recherche rapide avec organisation adaptative du dictionnaire par la méthode proposée, la distance euclidienne étant la mesure de distorsion : **1 min 35 sec.**
- Recherche rapide avec organisation adaptative du dictionnaire par la méthode proposée, la distance de Minkowski étant la mesure de distorsion : **1 min 2 sec.**

QV pour des blocs image de taille k = 4x4 pixels.

		Recherche séquentielle			Algorithme proposé			
N	E	add.	abs.	comp.	add.	abs.	comp.	%G
16	2.8	64	64	63	9.4	9.4	16.6	82.5
32	3.1	128	128	127	12.6	12.6	24.3	87.9
64	3.6	256	256	255	15.9	15.9	36.6	91.8
128	3.9	512	512	511	21.2	21.2	54.7	94.2
256	4.5	1024	1024	1023	28.4	28.4	90.5	95.7

Table 2.8
Comparaison des performances de l'algorithme de recherche séquentielle et de l'algorithme proposé. Le dictionnaire est celui obtenu directement après la séquence d'apprentissage.

		Algorithme 1				Algorithme proposé			
N	E	add.	abs.	comp.	%G	add.	abs.	comp.	%G
16	2.8	18.2	18.2	30.2	66.8	10.1	10.1	17.2	81.4
32	3.1	30.2	30.2	51.1	72.3	13.5	13.5	26.2	87.0
64	3.6	54.2	54.2	93.1	75.0	17.2	17.2	38.8	91.2
128	3.9	87.8	87.8	155.5	79.5	23.6	23.6	57.1	93.7
256	4.5	139.5	139.5	257.1	83.5	29.3	29.3	96.7	95.5

Table 2.9
Comparaison des performances de l'algorithme proposé et de l'algorithme optimisé de recherche par calcul partiel de distance, les vecteurs du dictionnaire étant ordonnés par fréquence d'occurrence croissante ($p_1 \leq p_2 \leq \ldots\ldots\ldots \leq p_N$).

		Algorithme 1				Algorithme proposé			
N	E	add.	abs.	comp.	%G	add.	abs.	comp.	%G
16	2.8	10.6	10.6	18.7	80.3	9.2	9.2	16.5	82.8
32	3.1	13.3	13.3	25.8	87.2	12.2	12.2	24.1	88.2
64	3.6	16.5	16.5	36.6	91.6	15.4	15.4	34.9	92.1
128	3.9	21.0	21.0	55.4	94.2	20.1	20.1	53.9	94.4
256	4.5	29.1	29.1	91.5	95.6	28.2	28.2	90.1	95.7

Table 2.10
Comparaison des performances de l'algorithme proposé et de l'algorithme optimisé de recherche par calcul partiel de distance, les vecteurs du dictionnaire étant ordonnés par fréquence d'occurrence décroissante ($p_1 \geq p_2 \geq \ldots\ldots\ldots \geq p_N$).

Temps d'exécution sur station *SPARC1* pour une image de 512x512 pixels sur 8 bits (16384 vecteurs), avec un dictionnaire de 256 vecteurs.

- Recherche séquentielle avec distance euclidienne : **22 min 50 sec.**
- Recherche séquentielle avec distance de Minkowski : **6 min 30 sec.**
- Recherche optimisée par calcul partiel de distance, les vecteurs du dictionnaire étant arrangés par probabilité de recherche croissante ($p_1 \leq p_2 \leq \ldots\ldots\ldots \leq p_N$) : **1 min 44 sec.**
- Recherche optimisée par calcul partiel de distance, les vecteurs du dictionnaire étant arrangés par probabilité de recherche décroissante ($p_1 \geq p_2 \geq \ldots\ldots\ldots \geq p_N$) : **29 sec.**
- Recherche rapide avec organisation adaptative du dictionnaire par la méthode proposée, la distance euclidienne étant la mesure de distorsion : **42 sec.**
- Recherche rapide avec organisation adaptative du dictionnaire par la méthode proposée, la distance de Minkowski étant la mesure de distorsion : **30 sec.**

Les résultats présentés dans les tableaux précédents montrent que l'algorithme rapide de recherche proposé offre des performances intéressantes comparativement aux méthodes conventionnelles. Ces performances sont dues essentiellement à l'utilisation des propriétés métriques de la distance pour la réduction du nombre de calculs, et d'une grande part à l'organisation adaptative du dictionnaire pendant la recherche de façon à exploiter au mieux la distribution statistique des blocs de l'image. La rapidité de l'algorithme est donc à juste titre fonction de la taille des blocs utilisés , de l'entropie du dictionnaire et aussi de la mesure de distorsion adoptée.

Les blocs images contigus de petite dimension (k = 2x2 pixels) sont très corrélés. Ceci est illustré par les figures 2.11, 2.12 et 2.13 où, pour des dictionnaires de 256 vecteurs, on peut constater que plus de 80% des vecteurs de dimension 4 sont éliminés dès la première itération, alors que moins de 50% sont éliminés quand la dimension des vecteurs est 64.
Cette élimination permet de réduire le nombre de calculs de distance à effectuer, et par conséquent, l'algorithme proposé converge rapidement en moins de dix itérations.

Le gain par rapport à la recherche séquentielle est particulièrement mis en évidence lorsque le calcul de la distance est complexe. La complexité de la distance ici est fonction d'une part de la taille des vecteurs et d'autre part de la mesure de distorsion utilisée. Ainsi, l'élimination par inégalité triangulaire sera plus efficace sur des vecteurs de dimension 4x4 que sur des vecteurs de taille 2x2, il en sera de même si on choisit comme mesure de distorsion la distance euclidienne au lieu de la distance de Minkowski.
Dans le cas de la distance euclidienne par exemple, la réduction du temps de recherche du vecteur de reproduction dans le dictionnaire est essentiellement liée à la réduction du nombre de multiplications. Les multiplications sont à la distance euclidienne ce que sont les valeurs absolues à la distance de Minkowski, or sur les tableaux précédents, on peut observer que pour un dictionnaire de 256 vecteurs, on a une réduction de plus de 95% du nombre de valeurs absolues. Une multiplication nécessitant plus de cycles machine qu'une valeur absolue, on comprend que le gain en temps CPU par rapport à la recherche séquentielle soit plus important pour la distance euclidienne.

En effet, pour des blocs de 2x2 pixels, l'algorithme proposé est 7 fois plus rapide que la recherche exhaustive si nous adoptons la distance de Minkowski, et 15 fois plus rapide si nous adoptons la distance euclidienne.

Pour des blocs de 4x4 pixels, le rapport de rapidité est de 13 avec la distance de Minkowski et plus de 30 avec la distance euclidienne.

Enfin, nous avons vu au paragraphe **7.5.1** que la rapidité de la recherche par arrangement progressif du dictionnaire était aussi fonction de l'entropie de la source. Pour une entropie maximale (vecteurs équiprobables), on a une légère baisse des performances de l'algorithme.

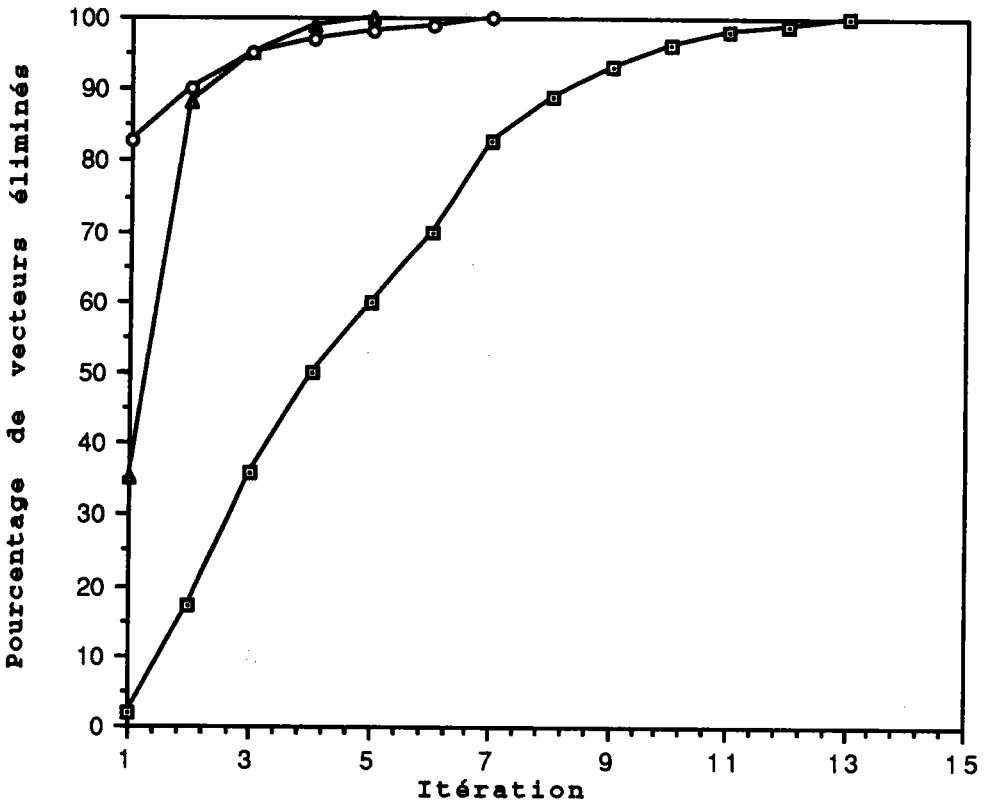


Fig 2.11 Efficacité des règles d'élimination pour les trois méthodes.
Dictionnaire: 256 vecteurs de dimension 4 (blocs 2x2 pixels)

Recherche optimisée par calcul partiel de distance, vecteurs du dictionnaire ordonnés par fréquence d'occurence croissante

Recherche optimisée par calcul partiel de distance, vecteurs du dictionnaire ordonnés par fréquence d'occurrence décroissante

Recherche rapide avec arrangement adaptatif des vecteurs du dictionnaire par la méthode proposée

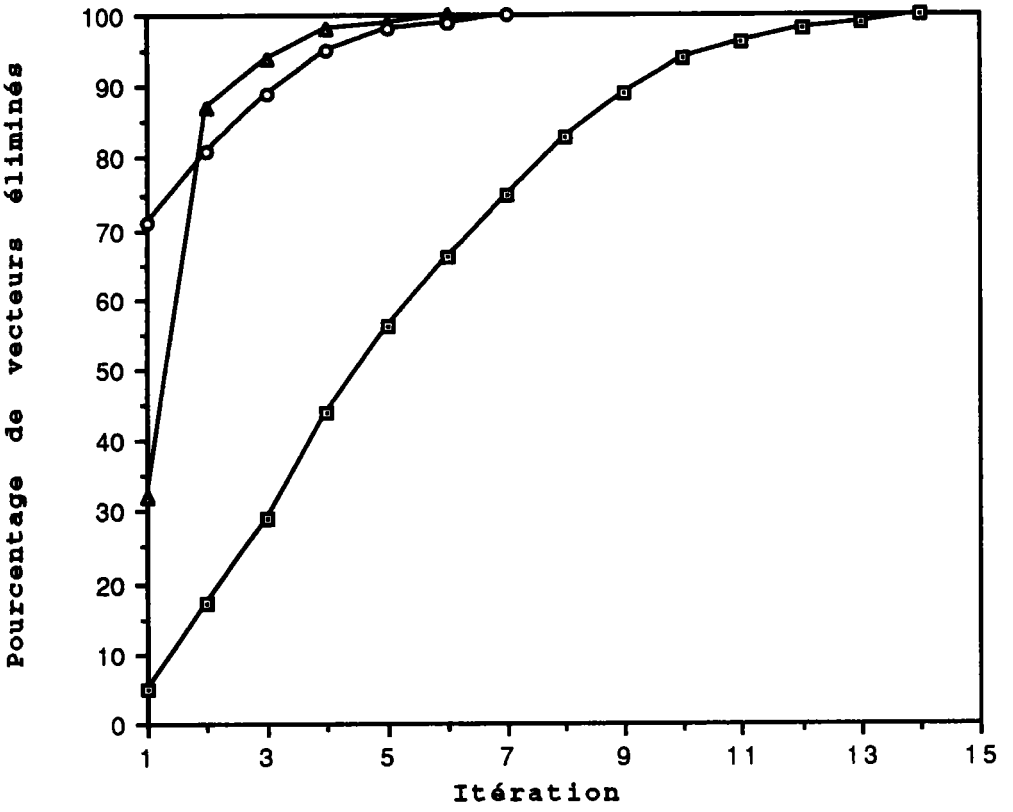


Fig 2.12 Efficacité des règles d'élimination pour les trois méthodes
Dictionnaire: 256 vecteurs de dimension 16 (blocs 4x4 pixels)

Recherche optimisée par calcul partiel de distance, vecteurs du dictionnaire ordonnés par fréquence d'occurrence croissante

Recherche optimisée par calcul partiel de distance, vecteurs du dictionnaire ordonnés par fréquence d'occurrence décroissante

Recherche rapide avec arrangement adaptatif des vecteurs du dictionnaire par la méthode proposée

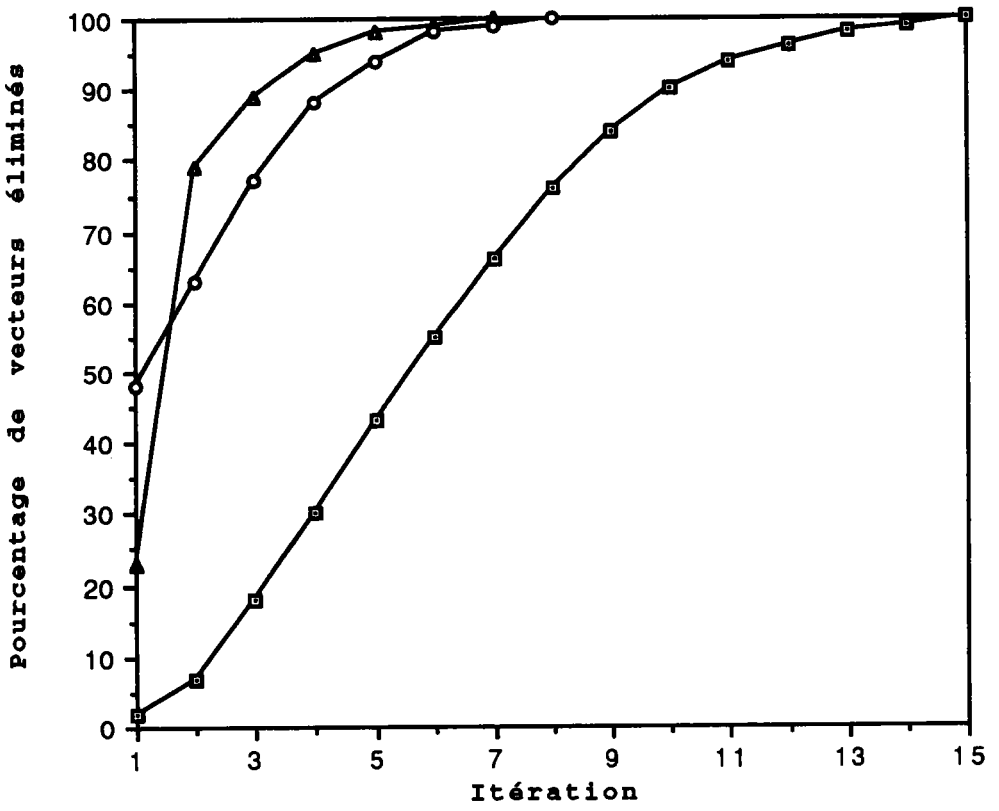


Fig 2.13 Efficacité des règles d'élimination pour les trois méthodes
Dictionnaire: 256 vecteurs de dimension 64 (blocs 8x8 pixels)

Recherche optimisée par calcul partiel de distance, vecteurs du dictionnaire ordonnés par fréquence d'occurrence croissante
Recherche optimisée par calcul partiel de distance, vecteurs du dictionnaire ordonnés par fréquence d'occurrence décroissante
Recherche rapide avec arrangement adaptatif des vecteurs du dictionnaire par la méthode proposée

8.4. Algorithme rapide de recherche sur un dictionnaire dont les vecteurs sont ordonnés par moyenne croissante

8.4.1. Position du problème

L'algorithme de Q.V. proposé est une optimisation de la technique exposée au paragraphe 5. Les vecteurs du dictionnaire sont ordonnés par moyenne croissante et la recherche du plus proche voisin d'un vecteur test x s'effectue autour des éléments dont la moyenne se rapproche le plus de celle de x. Dans cette méthode, on choisit délibérément de perdre un peu de précision au bénéfice de la vitesse de traitement. L'algorithme incorpore en outre les techniques d'optimisation citées dans les paragraphes précédents à savoir, exploitation de la redondance spatiale entre blocs adjacents de l'image (cf. fig 2.14) en utilisant les propriétés de l'élimination par inégalité triangulaire et de la recherche par calcul partiel de la distance. Pour un dictionnaire de 256 vecteurs de dimension k = 4, les résultats des simulations montrent une réduction de plus de 95% du nombre d'opérations par rapport à la classique recherche séquentielle. Le coût de ces performances est une légère dégradation de la qualité de l'image (0.1 à 0.5 dB) et une petite augmentation de la taille mémoire [NYE-91].

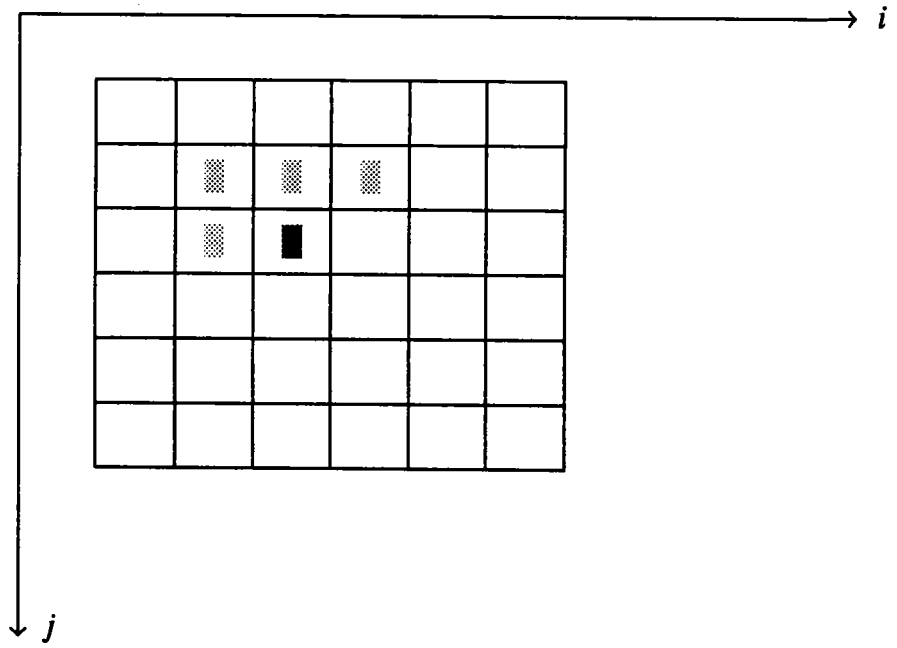


Fig 2.14
Détermination du voisinage d'un bloc image à coder

Bloc *B(i,j)* actuellement codé.
Blocs *B(i-1, j), B(i-1, j-1), B(i, j-1), B(i+1, j-1)*
sélectionnés dans le but d'exploiter la redondance spatiale de l'image.

8.4.2. Algorithme

Soit $C = \{c_j ; j = 1,...,N\}$ un dictionnaire de N vecteurs de dimension k.

où $c_j = (c_{ji} ; i = 1,...,k)$ est le *jième* vecteur du dictionnaire et c_{ji} la *iième* composante de c_j.

Soit $x = (x_i ; i = 1,...,k)$ un vecteur à coder.

Il s'agit de trouver le vecteur de reproduction $Q(x)$ de x dans le dictionnaire suivant un critère de distorsion minimale.

La mesure de distorsion adoptée est la distance de Minkowski définie par:

$d(x , y) = \max_i |x_i - y_i|$ $\quad i = 1,...,k$

La structure de l'algorithme rapide de QV proposé peut se diviser en deux parties comme suit :

Arrangement du dictionnaire et calculs préliminaires

Appelons C_m le dictionnaire des moyennes

$C_m = \{m(c_i) ; i = 1,...,N\}$

où $\quad m(c_i) = 1/k \sum_{j=1}^{k} c_{ij}$

est la moyenne des composantes du *ième* vecteur c_i du dictionnaire C.

- Arrangeons C_m de telle sorte que ses éléments soient dans l'ordre croissant, et organisons les vecteurs du dictionnaire C dans l'ordre correspondant à celui de leurs moyennes respectives.

- Calculer les $N \cdot (N - 1)/2$ distances entre tous les vecteurs du dictionnaire et stocker en mémoire les distances ainsi calculées.

Codage des vecteurs

(1) Entrer un vecteur $x = (x_i ; i = 1,...,k)$ et calculer la moyenne de ses composantes $m(x)$

avec $m(x) = 1/k \cdot \sum_{j=1}^{k} x_j$

(2) Déterminer la moyenne la plus proche de $m(x)$ dans le dictionnaire des moyennes C_m et sélectionner le vecteur correspondant dans le dictionnaire C.

(3) Soit i l'index du vecteur sélectionné,
déterminer l'intervalle de recherche [u , v] tel que:

$u = -L/2 + i$ $\qquad$ $v = L/2 + i$

vérifiant les conditions suivantes:

$u = 1$ si $u < 1$ $\qquad$ et $\qquad$ $v = N$ si $v > N$

Soit C_r le dictionnaire réduit dans lequel on restreint la recherche, L représente la taille de C_r

(4) Test des blocs adjacents au bloc en cours de codage (cf. fig 2.14).
Utiliser la recherche par calcul partiel de la distance et l'élimination par inégalité triangulaire pour déterminer le plus proche voisin de x parmi les vecteurs du dictionnaire retenus comme représentants des blocs contigus au bloc en cours de codage.

(5) Soient c_{min} l'actuel vecteur de reproduction et d_{min} l'actuelle distorsion minimale.
Eliminer par inégalité triangulaire tous les vecteurs c_j $(u \leq j \leq v)$ du dictionnaire réduit C_r tels que:
$d(c_{min} , c_j) > 2 \cdot d_{min}$.

(6) Si tous les vecteurs de C_r sont éliminés, alors le représentant $Q(x)$ de x dans le dictionnaire est c_{min} . Aller à l'étape *(7)* .
Sinon, utiliser la recherche par calcul partiel de distance pour sélectionner le prochain plus proche voisin parmi les candidats restant dans le dictionnaire réduit C_r et aller à l'étape *(5)* .

(7) Vérifier si le dernier vecteur de la source a été codé. Si oui, terminer le processus de quantification vectorielle, sinon aller à l'étape *(1)* .

8.4.3. Résultats des simulations

L'algorithme précédent a été simulé sur calculateur et ses performances ont été évaluées sur des dictionnaires de diverses tailles générés à partir de l'algorithme d'apprentissage LBG. Une image test de 256x256 pixels sur 8 bits, partitionnée en blocs de 2x2 pixels a été codée. La table 2.11 présente le nombre d'opérations arithmétiques obtenues par vecteur en comparaison avec l'algorithme de recherche séquentielle, la mesure de distorsion utilisée étant la distance de Minkowski.

Le rapport signal sur bruit calculé dans nos simulations est défini par :

$SNR = 10 \log_{10}(255^2 / EQM) dB$

où EQM est l'Erreur Quadratique Moyenne définie ci-dessous:

$$EQM = \sum_{i=1}^{256x256} (x_i - y_i)^2 / (256x256)$$

x_i et y_i sont les *iièmes* pixels des images originale et codée respectivement.

Nous adopterons les notations suivantes pour une compréhension aisée du tableau 2.11.

N : Taille du dictionnaire.
L : Intervalle de recherche.
add. : Nombre moyen d'additions requises pour coder un vecteur.
abs. : Nombre moyen de valeurs absolues requises pour coder un vecteur.
comp. : Nombre moyen de comparaisons requises pour coder un vecteur.
SNR : Rapport signal sur bruit (Signal to Noise Ratio).
%G : Pourcentage moyen du nombre d'opérations arithmétiques économisées par rapport à l'algorithme de recherche séquentielle.

Recherche séquentielle					Algorithme proposé					
N	add.	abs.	comp.	SNR	L	add.	abs.	comp.	SNR	%G
16	64	64	63	24.84	2	11.9	8.8	22.9	24.78	79.7
32	128	128	127	26.05	4	16.2	12.7	34.0	25.92	85.2
64	256	256	255	27.73	8	21.2	17.6	48.4	27.67	89.8
128	512	512	511	29.28	16	28.8	24.0	65.3	29.03	93.0
256	1024	1024	1023	30.88	24	22.9	20.1	62.8	30.48	96.9

Table 2.11
Comparaison des performances de l'algorithme de recherche séquentielle et de l'algorithme rapide de recherche sur dictionnaire ordonné par moyennes croissantes.

Les résultats du tableau 2.11 montrent que l'algorithme proposé est très rapide. Cette rapidité est étroitement liée à la taille de l'intervalle de recherche adopté. Plus cet intervalle est petit, plus on gagne en vitesse et plus l'image codée est dégradée. Pour une distorsion supplémentaire de 0.5 dB, le tableau 2.11 montre qu'on peut avoir plus de 95% de réduction en opérations arithmétiques. Bref, l'algorithme proposé offre un excellent compromis entre rapidité et précision, avec la possibilité pour l'utilisateur de fixer les paramètres relatifs au critère qu'il souhaite privilégier.

Le cas le plus défavorable apparaît lorsque tous les vecteurs du dictionnaire ont des moyennes identiques, c'est-à-dire lorsque le dictionnaire des moyennes C_m a une entropie maximale (éléments équiprobables). Dans ce cas, la sélection d'un intervalle de recherche réduit introduit incontestablement une dégradation importante sur l'image quantifiée bien que la rapidité de l'algorithme ne soit pas altérée.

Un exemple typique pouvant illustrer cette situation critique est le suivant:

Soit $x = (x_j ; j = 1,...,k)$ un vecteur de l'espace.

Appliquons à x la transformation: $x \text{ -----> } y(x) = x - m(x)$

avec $m(x) = 1/k \cdot \sum_{j=1}^{k} x_j$ moyenne des composantes du vecteur x.

On peut observer que toutes les moyennes $m(y(x))$ sont identiques et nulles.

9. Conclusion du chapitre

Nous avons proposé dans ce chapitre de nouveaux algorithmes rapides de recherche pour le codage d'image par quantification vectorielle. La vitesse de ces algorithmes est due essentiellement à l'utilisation des propriétés métriques de la distance d'une part, et d'autre part à l'exploitation des caractéristiques spatiales du signal image, à savoir la redondance. Les résultats des simulations ont montré qu'on pouvait atteindre des performances de l'ordre de 95% de réduction du nombre d'opérations arithmétiques, moyennant une légère augmentation de la capacité mémoire.
Outre la quantification vectorielle, les algorithmes proposés peuvent être facilement étendus à d'autres domaines comme le codage de la parole, la reconnaissance de formes, la classification des données et notamment l'accélération de l'algorithme d'apprentissage en vue de la génération du dictionnaire.

10. Références bibliographiques

[BEI-85] **C. D. Bei and R.M. Gray,** "An improvement of the minimum distortion encoding algorithm for vector quantization", IEEE Trans. Comm., Vol. COM-33, No. 10, Oct. 1985, pp. 1132-1133.

[CHE-85] **D. Y. Cheng, A. Gersho, B. Ramamurthi and Y. Shoham,** "Fast search algorithms for vector quantization and pattern matching", Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing, Vol. 1, Mar. 1985, pp. 9.11.1-4.

[CHE-86] **D. Y. Cheng and A. Gersho,** "A fast codebook search algorithm for neirest-neighbour pattern matching", IEEE ICASSP, Tokyo, 1986, pp. 6.14.1-4.

[CHE-89] **S. H. Chen and J. S. Pan,** "Fast search algorithm for VQ-based recognition of isolated word", IEE Proceedings, vol. 136, Pt. I, No. 6, Dec. 1989, pp. 391-396.

[DIX-82] **J. Dixmier**, "Cours de mathématiques du premier cycle, première année", 1982, Gauthier-Villars.

[FUN-75] **K. Funkunaga and P. M. Narendra**, "A branch and bound algorithm for computing k-nearest neighbors", IEEE Trans. Comput. 24, 1975, pp. 750-753.

[GRA-82] **R. M. Gray and H. Abut**, "Full search and tree searched vector quantization of speech waveforms", Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing, May 1982, pp. 593-596.

[KAM-85] **B. Kamgar-Parsi and L. N. Kanal**, "An improved branch and bound algorithm for computing k-nearest neighbors", Pattern Recognition Letters 3, 1985, pp. 7-12.

[KNU-73] **D. E. Knuth**, "The art of computer programming - Vol. 3 - Sorting and Searching", Addison-Wesley, 1973.

[KOH-88] **J. S. Koh and J. K. Kim,** "Fast sliding search algorithm for vector quantization in image coding", Electronics Letters, 18th August 1988, Vol. 24, No. 17.

[LIN-80] **Y. Linde, A. Buzo and R. M. Gray,** "An algorithm for vector quantizer design", IEEE Trans. Comm., Vol. COM-28, No. 1, Jan. 1980, pp. 84-95.

[NYE-91] **A. Nyeck, M. Charfi, A. Laprévote and A. Tosser-Roussey**, "An improved vector quantization fast search algorithm for image coding", Al-Azhar Engineering Second International Conference, Dec. 1991, Cairo.

[NYE-92] **A. Nyeck, H. Mokhtari and A. Tosser-Roussey**, "Fast adaptive search algorithm for vector quantization by progressive codebook arrangement", to be published in Pattern Recognition, Pergamon Press, Oxford.

[PAL-89] **K. K. Paliwal and V. Ramasubramanian,** "Effect of ordering the codebook on the efficiency of the partial distance search algorithm for vector quantization", IEEE Trans. Comm., Vol. 37, No. 5, May 1989, pp. 538-540.

[SOL-87] **M. R. Soleymani and S. D. Morgera,** "A high-speed search algorithm for vector quantization", IEEE ICASSP, 1987, pp. 45.6.1-3.

[VID-86] **E. Vidal Ruiz,** "An algorithm for finding nearest neighbour in (approximately) constant average time", Pattern Recog. Lett. 4, 1986, pp. 145-157.

Chapitre 3

UTILISATION DES TRANSPUTERS EN QUANTIFICATION VECTORIELLE DES IMAGES

1. Introduction

La quantification vectorielle d'images nécessite une énorme quantité de calculs, compte tenu du volume des données à traiter et de la nature de ces données. Cette application nécessite donc une architecture matérielle conjuguant à la fois la puissance de traitement et la performance en échange avec l'environnement par les entrées/sorties.

Des systèmes matériels justifiant de telles performances ne peuvent vraiment être réalisés qu'en faisant appel à la parallélisation, c'est-à-dire l'utilisation de plusieurs processeurs interconnectés d'une part, et d'autre part, la mise au point d'outils logiciels prenant en compte le parallélisme.

Le transputer, processeur spécialement conçu pour être interconnecté en réseau, est particulièrement adapté pour des applications du type codage d'images. A la différence des processeurs conventionnels actuels (680XX , 80X86 , SPARC , ...) , l'architecture du transputer a été optimisée de manière à fournir un maximum de fonctionnalités. En effet, le transputer intègre sur sa puce un ensemble d'éléments pour les calculs, les communications entre processeurs, la gestion des tâches, la gestion du temps, faisant de lui un monochip très performant. Ainsi , le transputer peut être considéré comme un comPUTER monoboîtier et un composant silicium comme un TRANSistor, conçu de manière à être l'élément de base de systèmes complexes et hautement concurrents.

Par ailleurs, l'existence de langages de programmation structurés comme OCCAM et plus récemment ANSI-C de INMOS [INM-90a] [INM-90b], mettant l'accent sur l'interdépendance entre processus concurrents et intégrant les notions de priorité entre processus, d'interruption et de temps réel, rend aisé le développement d'applications sophistiquées sur transputers. Nous nous intéresserons dans ce chapitre au cas de la QV des images et notamment, au gain de temps obtenu en utilisant un réseau de transputers.

2. Aspect matériel du transputer

En dehors des transputers spécialisés, on peut distinguer actuellement quatre versions de transputers caractérisés par leurs performances et la largeur des données manipulées.
Chacun de ces transputers intègre sur sa puce un microprocesseur (16 ou 32 bits), de la mémoire locale, une interface mémoire, et éventuellement une unité de calcul en virgule flottante sur 64 bits, autorisant les opérations en simple ou double précision selon la norme ANSI-IEEE. L'exécution de ces opérations se fait concurremment avec les activités du processeur et peut atteindre une puissance de 200 MIPS et 25 MFLOPS pour la famille T9.

Le transputer est aussi caractérisé par un mécanisme d'échanges par des liens séries qui permet la communication avec un environnement compatible, et l'interconnexion entre transputers suivant des topologies variées (noeud, arbre, pipeline, ...). Ainsi quatre liens bidirectionnels sont disponibles, autorisant chacun des transferts de données avec un débit pouvant aller jusqu'à 100 Mbits/sec pour la famille T9.
Les transferts sont assurés sans intervention du processeur central, ce qui permet l'obtention de performances importantes en communication, sans dégradation notable de la puissance de calcul pour les traitements qui se déroulent en parallèle. Ainsi pour des algorithmes faciles à distribuer sur un réseau et dont l'exécution implique que les échanges entre processeurs soient beaucoup plus courts en temps que le traitement à l'intérieur du transputer, les performances évoluent linéairement avec le nombre de transputers du réseau.
La réalisation d'architectures multiprocesseurs complexes du type MIMD (Multiple Instructions Multiple Data streams) habituellement réservées aux gros ordinateurs est naturelle sur transputers grâce à ce mécanisme d'échange par liens.

Une autre particularité du transputer est son aptitude à gérer en parallèle un ensemble de processus. L'ordonnancement qui doit être réalisé par logiciel pour les autres composants, est câblé sur transputer, ce qui conduit à des commutations très rapides.

Nous présentons brièvement dans les paragraphes suivants les caractéristiques générales de la famille de transputers T8 qui nous a servi de support de développement.

2.1. Famille T8 (Transputer IMS T800)

Architecture CPU : 32 bits , 17.5 - 30 MHz
Temps de cycle interne : 33 ns
Unité de calcul en virgule flottante sur 64 bits suivant la norme ANSI-IEEE
Vitesse maximale d'exécution des instructions : 30 MIPS , 4.3 MFLOPS
Mémoire locale : 4 Kbytes de RAM statique
Vitesse de transfert de la mémoire interne : 120 Mbytes/sec
Espace d'adressage de la mémoire externe : 4 Gbytes
Vitesse de transfert de la mémoire externe : 40 Mbytes/sec
Temps de réponse aux interruptions : 630 ns
Quatre liens séries bidirectionnels 5 / 10 / 20 Mbits/sec

Vitesse maximale de transfert au niveau du lien : 2.35 Mbytes/sec
Timers internes de 1 µs et 64 µs
Instructions de transfert de bloc à 2 dimensions pour les applications graphiques
Lancement à partir de la ROM ou des liens de communication.
Fréquence horloge externe : 5 MHz
Alimentation : +5V ± 5%

Le transputer T800 est un composant VLSI qui intègre sur sa puce un microprocesseur 32 bits, une unité de calculs en virgule flottante sur 64 bits, quatre liens de communication bidirectionnels, 4 Koctets de mémoire interne, une interface mémoire configurable, et des modules de traitement pour les applications graphiques.

L'architecture du processeur permet une implémentation directe du langage OCCAM, en particulier, des modèles de la concurrence et de la communication qu'il définit. Ce qui conduit à des programmes compilés particulièrement compacts et efficaces. Les appels de procédures, le changement de processus et la reconnaissance d'interruption se font en moins d'une microseconde. De plus, le code a été particulièrement optimisé pour le traitement des opérations en virgule flottante. Ainsi, le langage de programmation naturel des transputers est OCCAM, mais d'autres outils de développement efficaces sont disponibles. Nous avons pour notre part utilisé le package ANSI-C pour implémenter nos algorithmes de quantification vectorielle.

La vitesse de l'horloge interne peut être sélectionnée, sans dépasser une vitesse maximale autorisée, entre 20 et 30 MHz. Typiquement, un T800 fonctionnant à 30 MHz offre en standard une puissance de 15 MIPS, puissance qui peut atteindre une valeur maximale de 30 MIPS.

L'unité de calcul en virgule flottante sur 64 bits traite des opérations en simple ou double précision selon la norme ANSI-IEEE 754 pour l'arithmétique en virgule flottante. L'exécution de ces opérations se fait concurremment avec les activités du processeur et peut atteindre une vitesse de 1.5 MFLOPS pour une horloge interne à 20 MHz , et 2.25 MFLOPS pour une fréquence d'horloge de 30 MHz.

A l'aide d'un matériel spécifique et intégré, l'exécution de programmes écrits en langages de haut niveau est rendue plus sûre par la détection d'erreurs. Une autre particularité du T800 est l'existence d'une instruction de comptage des bits à un dans un mot, s'avérant très utile pour les applications de reconnaissance de formes.

Par l'intermédiaire de son interface mémoire, le T800 peut adresser un espace de 4 Goctets. Le bus de communication de 32 bits multiplexe adresses et données et sa vitesse de transfert peut aller jusqu'à 40 Moctets/sec pour une unité fonctionnant à 30 MHz. Un contrôleur de mémoire configurable fournit tous les diagrammes de temps, les signaux de contrôle gérant l'accès direct à la mémoire, les signaux de contrôle du rafraîchissement de mémoires dynamiques, pour une grande variété de systèmes de mémoire distincts.

3. Programmation des systèmes à base de transputers

3.1. Utilisation du langage C

OCCAM est le premier outil de développement sur transputer. Il a été conçu de façon à tirer une efficacité maximale des possibilités de ce composant, ceci est dû au fait que le transputer intègre matériellement des caractéristiques importantes du langage OCCAM, en particulier, la notion de processus. Un processus est une tâche élémentaire et autonome, avec ses propres données et son propre code, qui peut communiquer avec d'autres processus s'exécutant en même temps. Toutefois, des langages évolués peuvent être utilisés dans la mesure où ils intègrent les constructions particulières d'OCCAM : c'est le cas du langage ANSI-C de INMOS.

Le langage ANSI-C de INMOS [INM-90a] [INM-90b] représente le modèle traitement parallèle de processus séquentiels communiquant à travers un ou plusieurs canaux dans le formalisme CSP (Communicating Sequential Processes).
Dans le formalisme CSP, une communication a lieu quand un processus émetteur désigne un autre processus comme destinataire d'une sortie et que ce destinataire désigne le premier comme source d'information. De plus, il n'y a aucune possibilité pour l'émetteur de continuer son calcul tant que la valeur émise n'a pas été transmise à son destinataire. Sur le plan général, l'exécution d'une commande d'entrée ou de sortie est retardée tant que le processus correspondant n'est pas prêt à exécuter la sortie ou l'entrée correspondante [BAN-91].

Par ailleurs, la flexibilité inhérente du langage C, la capacité de mixer du code provenant de langages différents, et la capacité à exploiter la topologie concurrente du transputer font du INMOS ANSI-C un outil puissant pour la programmation des systèmes concurrents.

D'autre part, le package INMOS ANSI-C représente une amélioration considérable par rapport aux versions antérieures de compilateurs parallèles C proposés pour transputers, on peut citer :

- Conformité à la norme ANSI pour le C, au lieu de la définition originale du C par Kernighan et Ritchie

- Meilleure définition du préprocesseur

- Introduction d'énumérations

- Utilisation des structures comme paramètres d'entrée et de sortie de fonctions

- Génération du code optimisée

- Exécution du compilateur plus rapide

- Nouveaux outils de débogage incluant un débogueur symbolique interactif avec des points d'arrêt, un débogage post mortem amélioré et un simulateur de transputer sur la machine hôte.

- Langage de configuration mis à jour pour supporter les futures générations de transputers.
- Librairie améliorée de fonctions parallèles
- Mixage amélioré du C et du langage OCCAM
- Support pour l'assembleur
- Outils de programmation d'EPROM
- Générateur de fichier Makefile

3.1.1. Modèle de programmation

Le modèle de programmation parallèle consiste en un nombre de processus indépendants s'exécutant simultanément et communiquant à travers des canaux. Ces canaux sont les chemins de communication par lesquels les processus s'échangent des données.

Un processus peut être généré à partir d'un nombre quelconque d'autres processus parallèles, de telle sorte qu'un système logiciel entier peut être décrit comme une hiérarchie de processus parallèles communiquant entre eux. Ce modèle est conforme à plusieurs approches modernes de programmation parallèle.

La communication entre processus est synchronisée. Lorsqu'une information est transmise d'un processus à un autre, le processus émetteur ne peut continuer tant que le processus récepteur n'est pas prêt.

3.1.2. Programmation multiprocesseurs

Des processus parallèles peuvent être exécutés indépendamment sur un ou plusieurs processeurs connectés en réseau. Un langage spécial de configuration est utilisé pour distribuer les processus à travers le réseau de transputers et peut ainsi être utilisé pour programmer des systèmes multiprocesseurs complexes.

3.1.3. Programmation temps réel

La topologie concurrente du transputer est particulièrement adaptée à la programmation temps réel. Ses caractéristiques sont décrites ci-dessous :

- Implémentation directe et efficace de processus parallèles dans l'architecture du transputer.

- Mise en oeuvre de priorités entre plusieurs processus concurrents de manière à optimiser le temps d'exécution de séquences critiques.

- Possibilité d'implémenter des interruptions software comme processus de haute priorité.

- Programmation facile de l'horloge interne du transputer (timer) de façon à pouvoir contrôler le temps alloué aux différents processus, constituants du programme.

3.2. Développement de programmes

Le développement d'un programme en C sur un système multitransputers est conforme au schéma suivant :

Edition des fichiers source à l'aide d'un éditeur de textes standard.

Compilation de chaque code source avec le compilateur ***icc***.

Edition de liens entre les codes objet et les librairies en utilisant ***ilink***.

Configuration du réseau de transputers

Il s'agit de définir quel processus sera exécuté sur quel transputer. Ceci est réalisé grâce à l'outil de développement ***icconf***.

Boot Strap

Pour faire tourner le programme sur un transputer, un module de lancement appelé *bootstrap* doit être ajouté. C'est une portion de code qui est ajoutée au début du code programme et qui contient les instructions nécessaires pour la réinitialisation du transputer, le chargement et le lancement du programme. Une fois le programme chargé, le *bootstrap* est effacé. Le *bootstrap* est ajouté au programme grâce à l'outil de développement ***icollect***.

Exécution du programme

Pour charger un programme dans les transputers on utilise le programme ***iserver***. Ce programme est installé sur l'ordinateur de commande qui peut par ce biais, mettre ses ressources à la disposition du transputer à savoir, les entrées/sorties clavier/écran, l'utilisation des fichiers systèmes.

Débogage

Les programmes pour systèmes multiprocesseurs peuvent être débogués au niveau symbolique par l'utilisation du débogueur ***idebug***. Le débogage par points d'arrêt permet l'exécution interactive de programmes alors que le débogage post mortem permet aux programmes arrêtés d'être débogués à partir du contenu de la mémoire des transputers.

4. Implémentation d'algorithmes rapides de QV sur des systèmes multi-transputers

Nous présentons dans cette partie un exemple d'implémentation d'algorithme de QV sur un système multi-transputers. L'algorithme choisi est l'algorithme rapide de quantification vectorielle par arrangement progressif du dictionnaire présenté au paragraphe 2.8.3 [NYE-92]. Les programmes complets relatifs à l'algorithme proposé pour une implémentation sur un système à quatre transputers sont fournis en annexes.

4.1. Méthode de parallélisation

Afin d'optimiser les charges de calculs, le choix de la méthode d'exploitation du parallélisme est primordial : il s'agit de prendre en compte le problème de l'affectation des données à chaque processeur et/ou le problème de la parallélisation de l'algorithme de traitement [HIR-90] , [PIN-90].

- Partitionnement des données

C'est l'opération de distribution des données aux différents processeurs travaillant en parallèle dans le réseau. Chaque processeur exécute le même algorithme sur des données différentes. Les communications inter-processeurs sont peu nombreuses. Les communications nécessaires sont la diffusion des données sur chaque processeur avec une répartition des charges équitable et la réception des résultats sur le processeur hôte pour l'analyse.

- Parallélisme algorithmique

Chaque processeur dispose d'une partie du programme global. Pour être totalement traitée, une donnée va donc être exploitée par tous les processeurs. Deux approches sont souvent retenues dans ce type de parallélisme [HIR-90] :

la vectorisation : un nombre donné d'opérations identiques est fait en parallèle simultanément avec N processeurs sur un groupe de N données fonctionnellement reliées.

la séquence : un algorithme est décomposé en une séquence de N sous-algorithmes dont chacun est ensuite pris en charge par un processeur (structure pipeline). Pour ce type de parallélisme, un bon équilibrage des charges est nécessaire. Une inégalité de répartition des charges contribuerait à la formation d'un goulot d'étranglement et donc à la perte d'efficacité du programme parallélisé.

Pour la quantification vectorielle des images, l'algorithme de recherche choisi nécessite un traitement identique sur tous les vecteurs. Pour obtenir un équilibrage optimal des charges de calcul sur chaque transputer, il faut utiliser un mode de fonctionnement du type SPMD (Single Program Multiple Data streams). On affecte donc le même programme à chaque processeur, les données sont également réparties équitablement sur chaque processeur et traitées simultanément.

4.2. Configuration du réseau de transputers

Afin de mettre en évidence l'intérêt de la configuration du réseau de transputers, nous prendrons comme exemple un système à deux transputers. Le problème de quantification vectorielle peut alors être divisé en deux processus : un processus maître et un processus esclave (Fig 3.1).

Le processus maître partitionne l'image en deux parties, transmet une partie de l'image au second transputer, traite l'autre partie de l'image, et reçoit l'image traitée par le second transputer.

Le processus esclave reçoit du premier transputer une partie de l'image, il la traite et transmet l'image quantifiée au premier transputer.

Les processus maître et esclave sont deux programmes écrits en INMOS ANSI-C. Ces deux programmes sont compilés indépendamment et portés sur les transputers correspondants grâce à l'outil de configuration icconf. Le programme de configuration pour un système à deux transputers de la famille T800 est du reste présenté ci-dessous. Le programme complet de configuration pour un réseau de quatre transputers est fourni en annexes.

Déclaration de l'architecture du réseau : type de transputers, taille mémoire associée à chaque transputer, dénomination logicielle de chaque transputer.

```
T800  (memory = 1M)  root ;
T800  (memory = 1M)  Slave ;
```

Description des connexions entre transputers

```
connect root.link[0] , host ;          Le transputer maître est relié à l'ordinateur hôte par le lien 0
connect root.link[3] , Slave.link[0];  Connection physique entre transputers maître et esclave
```

Description de l'interface entre les processus de QV et les autres processus du système. Les paramètres concernant la taille de la pile et du tas sont transmis par les canaux de communication

```
fs ------>      PROCESSUS      -----> ToWorker
ts <-----       MAITRE         <----- FromWorker

FromMaster ------>        PROCESSUS
ToMaster   ------>        ESCLAVE
```

```
process(stacksize = 1k, heapsize = 50k,
interface(input fs, output ts, input FromWorker, output ToWorker))  Master ;

process(stacksize = 1k, heapsize = 50k,
interface(input FromMaster, output ToMaster))  Worker ;
```

Rattachement des canaux de communication sotfware sur les processus présents sur les transputers

```
input from_host ;
output to_host ;

connect Master.fs , FromHost ;                    Connexion du processus maître au processeur hôte
connect Master.ts , ToHost ;
connect Master.ToWorker , Worker.FromMaster ;  Connexion des processus maître et esclave
connect Master.FromWorker , Worker.ToMaster ;
```

Entrer le code obtenu après compilation et édition de liens

```
use "master.lku" for Master ;      Code correspondant au processus maître
use "worker.lku" for Worker ;      Code correspondant au processus esclave
```

Implémentation des processus sur les processeurs et les canaux software sur les liens physiques des transputers

```
place Master on root ;
place Worker on Slave ;

place FromHost on Host ;
place ToHost on Host ;

place Master.fs on root.link[0] ;
place Master.ts on root.link[0] ;

place Master.ToWorker on root.link[3] ;
place Master.FromWorker on root.link[3] ;

place Worker.ToMaster on Slave.link[0] ;
place Worker.FromMaster on Slave.link[0] ;
```

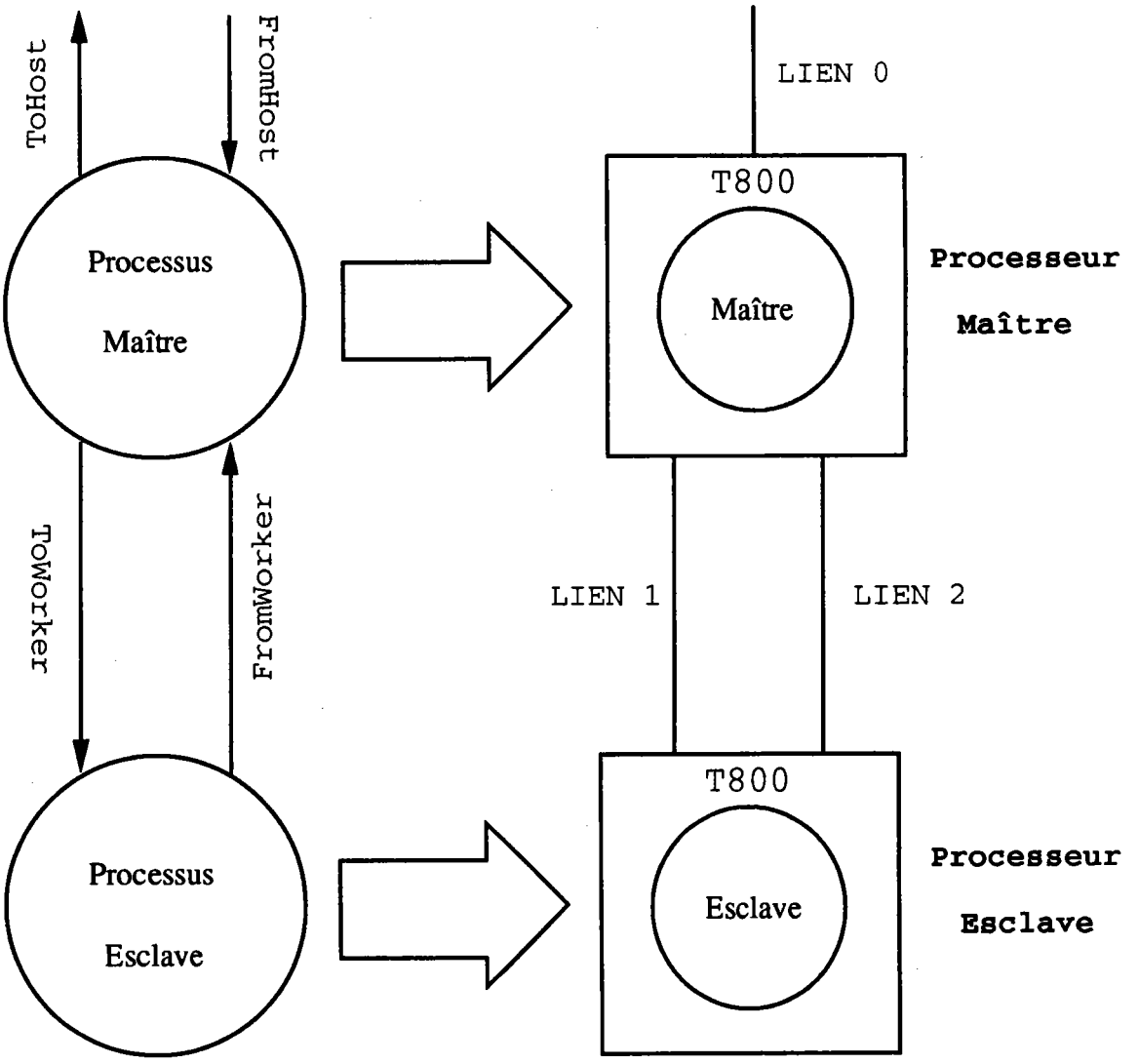


Fig 3.1 Configuration d'un système à deux transputers

4.3 Résultats expérimentaux

La puissance de calcul du transputer utilisé a été dans un premier temps comparée à celle d'un processeur classique. Nous avons à cet effet effectué des opérations arithmétiques usuelles sur des réels en simple précision, en utilisant d'une part un processeur 386 avec son coprocesseur 387 à 25 MHz, et d'autre part un transputer T800 cadencé à 25 MHz. Les algorithmes ont été implémentés sur le processeur 386 en C AIX (version IBM du système d'exploitation Unix) d'une part, et en Microsoft C 5.1 pour MS DOS d'autre part. Sur le T800, les algorithmes sont implémentés en INMOS ANSI-C, l'ordinateur hôte étant un compatible PC IBM fonctionnant sous MS DOS. Les programmes de test ont été successivement compilés sans, et avec les options d'optimisation du code exécutable, à savoir :

```
CL /AL /F A000 /FPc87 [/Opt] %1.c                    pour Microsoft C 5.1

cc [-O] -o $1 $1.c -lm                               pour IBM AIX-C

icc %1.c /T8 [/KS]
ilink %1.tco /f startup.lnk /t /T8
icollect %1.lku /t [/S 512]                          pour INMOS ANSI-C
iserver /sb %1.btl /se
```

Les résultats obtenus sont présentés ci-dessous :

	386-87 25 MHz C AIX	386-87 25 MHz MSC 5.1 (MS DOS)	T800 25 MHz ANSI-C
10^7 Additions	42 sec	137 sec	21 sec
10^7 Multiplications	44 sec	139 sec	24 sec
10^7 Valeurs absolues	90 sec	277 sec	58 sec
10^7 Comparaisons	35 sec	129 sec	11 sec

Table 1
Comparaison des performances sans optimisation des options de compilation.

	386-87 25 MHz C AIX	386-87 25 MHz MSC 5.1 (MS DOS)	T800 25 MHz ANSI-C
10^7 Additions	06 sec	054 sec	12 sec
10^7 Multiplications	07 sec	056 sec	13 sec
10^7 Valeurs absolues	88 sec	164 sec	40 sec
10^7 Comparaisons	33 sec	046 sec	06 sec

Table 2
Comparaison des performances avec optimisation des options de compilation.

Les opérations arithmétiques présentées précédemment sont celles généralement utilisées pour le calcul de distances dans un processus de quantification vectorielle. En se référant à la table 1, on peut constater que le T800 cadencé à 25 MHz est environ deux fois plus rapide qu'un 386-87 à la même fréquence fonctionnant sous AIX, et six fois plus rapide que ce même processeur fonctionnant sous MS DOS, pour des calculs sur des réels en simple précision.

Pour l'implémentation de l'algorithme rapide de QV sur un système multi-processeurs, nous avons successivement utilisé un réseau de deux, puis trois, puis quatre transputers. Compte tenu de la méthode de parallélisation utilisée (SPMD), les performances sur le système multi-transputers sont pratiquement multipliées par le nombre d'éléments du réseau.
Cette évolution quasi-linéaire des performances est due aussi au nombre réduit de transputers utilisés, ce qui réduit d'autant les échanges entre transputers maître et esclaves. Par ailleurs, la taille des images utilisées pour la quantification étant relativement petite (256x256 pixels sur 8 bits), on ne peut redouter d'être pénalisé par le volume de données à transmettre, la vitesse de transfert au niveau de chaque lien pouvant aller jusqu'à 20 Mbits/s.

Ainsi, pour un réseau de quatre transputers, l'algorithme de quantification vectorielle utilisé est pratiquement 8 fois plus rapide que sur un processeur 386-87 à 25 MHz fonctionnant sous AIX, et près de 30 fois plus rapide que sur un 386-87 à 25 MHz fonctionnant sous MS DOS, les algorithmes étant implémentés en langage C , les opérations s'effectuant sur des réels en simple précision, et les options de compilation retenues étant conformes à celles utilisées dans la table 1.

5. Références bibliographiques

[BAN-91] **Jean-Pierre Banâtre**, "La programmation parallèle - Outils, méthodes et éléments de mise en oeuvre", Eyrolles, Paris, 1991.

[HIR-90] **Ernest Hirsh**, "Les transputers - Application à la programmation concurrente", Eyrolles, Paris, 1990.

[INM-89] **INMOS** Limited, "The transputer databook", first/second edition, 1989.

[INM-90a] **INMOS** Limited, "ANSI C toolset language reference", August 1990.

[INM-90b] **INMOS** Limited, "ANSI C toolset user manual", August 1990.

[INM-91] **INMOS** Limited, "The transputer development and *i*q systems databook", second edition, 1991.

[KER-90] **B. W. Kernighan et D. M. Ritchie**, "Le langage C", 2ème édition, Masson, Paris, 1990.

[NYE-92] **A. Nyeck, H. Mokhtari and A. Tosser-Roussey**, "Fast adaptive search algorithm for vector quantization by progressive codebook arrangement", to be published in Pattern Recognition, Pergamon Press, Oxford.

[PAS-91] **O. Pasquier and J. P. Calvez**, "Utilisation du transputer pour les applications temps réel", La lettre du transputer et des calculateurs distribués, N° 10, Juin 1991, pp. 7-34.

[PIN-90] **A. Pinti, N. Schaltenbrand, M. Toussaint, J. Gresser, R. Luthringer, R. Minot et J. P. Macher**, "Etude d'un réseau de neurones multi-couches pour l'analyse automatique du sommeil sur T-Node", La lettre du transputer et des calculateurs distribués, N° 8, Déc. 1990, pp. 21-32.

[PRE-82] **Kendall Preston Jr and Leonard Uhr**, "Multicomputers and image processing - Algorithms and programs", Academic Press Inc., London, 1982.

Conclusion générale

Le problème majeur posé par la technique de quantification vectorielle est l'énorme quantité de calculs nécessaires pour coder chaque vecteur et aussi pour construire un dictionnaire optimal.

Pour l'élaboration du dictionnaire, nous avons utilisé l'algorithme LBG. Cet algorithme converge vers un optimum local, et la validité du dictionnaire construit dépend étroitement du dictionnaire choisi pour initialiser l'algorithme. Comme il n'existe pas de méthode d'initialisation universelle qui assure l'optimalité certaine du dictionnaire, nous avons pour notre part proposé une méthode d'initialisation efficace appelée méthode du maximum d'entropie. Le but de cette méthode est d'éliminer les vecteurs du dictionnaire les moins représentatifs de l'espace à coder et de sélectionner les éléments du dictionnaire de telle sorte qu'ils aient des fréquences d'occurrence pratiquement identiques (maximum d'entropie). Les résultats obtenus en utilisant cette méthode montrent que l'algorithme de classification converge en un minimum d'itérations vers un dictionnaire d'entropie maximale tout en générant une erreur de quantification minimale.

Un autre problème posé par l'algorithme LBG est la nécessité d'utiliser une séquence d'entraînement très longue pour construire un dictionnaire optimal. Suivant les paramètres du dictionnaire, les coûts de calculs et de mémoire associés deviennent alors prohibitifs. Typiquement, c'est la recherche de la partition optimale qui est pénalisante par comparaison avec le temps de remise à jour des représentants. Si on considère que le calcul de la distorsion nécessite K opérations arithmétiques élémentaires, le coût calcul pour la construction d'un dictionnaire de N représentants à partir d'une séquence d'entraînement de M vecteurs nécessitant I itérations est de l'ordre de KxNxMxI. Selon la taille du dictionnaire et de la séquence d'entraînement, l'élaboration du dictionnaire prend plusieurs heures de calculs sur une station *SPARC1*, d'où l'intérêt de mettre au point des algorithmes rapides. Nous avons à cet effet proposé plusieurs nouveaux algorithmes rapides de quantification vectorielle pour le codage d'images. Ces algorithmes exploitent les propriétés métriques de la distance utilisée ainsi que la redondance spatiale de l'image. Les résultats des simulations montrent qu'on peut atteindre une vitesse 30 fois supérieure à un algorithme de recherche exhaustive sur tout le dictionnaire. Par ailleurs l'implémentation de ces algorithmes sur des machines multi-processeurs à base de transputers nous permet d'envisager l'utilisation de la quantification vectorielle pour les applications de codage d'images en temps réel.

Le codage d'images néanmoins ne peut être réduit uniquement à la quantification vectorielle. S'il est vrai qu'un dictionnaire optimisé est indispensable pour restituer une image de bonne qualité, le système de codage doit tenir compte de l'aspect subjectif de l'image. Ceci peut être réalisé par un choix judicieux de la mesure de distorsion utilisée, cette mesure de distorsion reproduisant fidèlement la perception subjective du signal traité, en d'autres termes, une forte distorsion indiquant une mauvaise qualité et une faible distorsion une bonne qualité. Mais dans le cas du codage d'images, il est difficile d'envisager une mesure de distorsion modélisant idéalement la perception subjective de l'oeil. Aussi, pour améliorer la qualité de l'image, il est plus raisonnable d'introduire un traitement supplémentaire du signal avant codage, un traitement basé sur une transformation orthogonale type transformée en cosinus discrète étant bien indiqué à cet effet.

ANNEXES

Architecture du transputer IMS T800 1

Développement d'une application sur un système multi-transputers principales étapes et fichiers conventionnels en INMOS ANSI-C 2

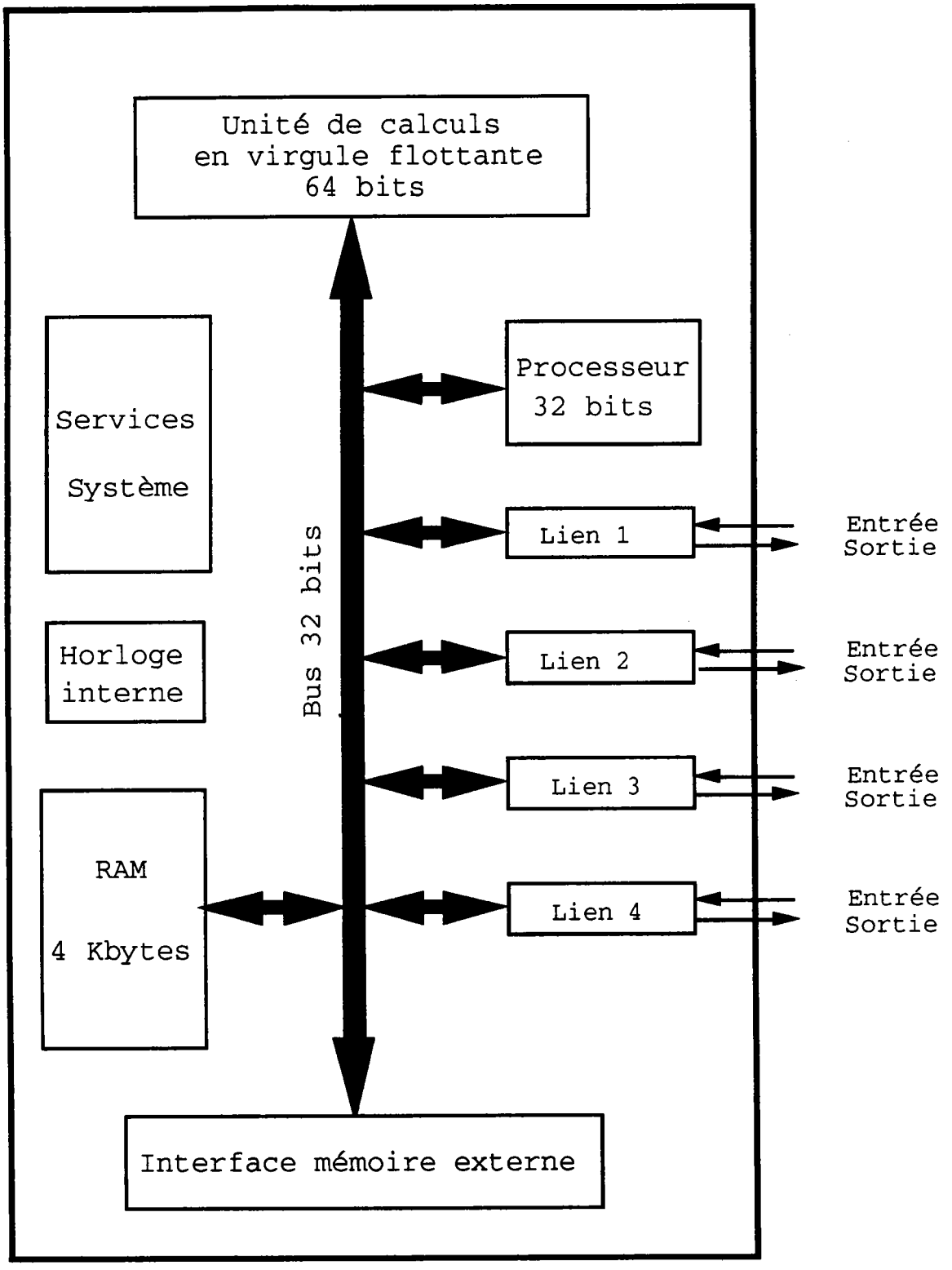
Configuration adoptée pour un réseau de 4 transputers IMS T800 - 1 MO - 25 MHz 3

Programme de configuration du réseau (qv.cfs) 4

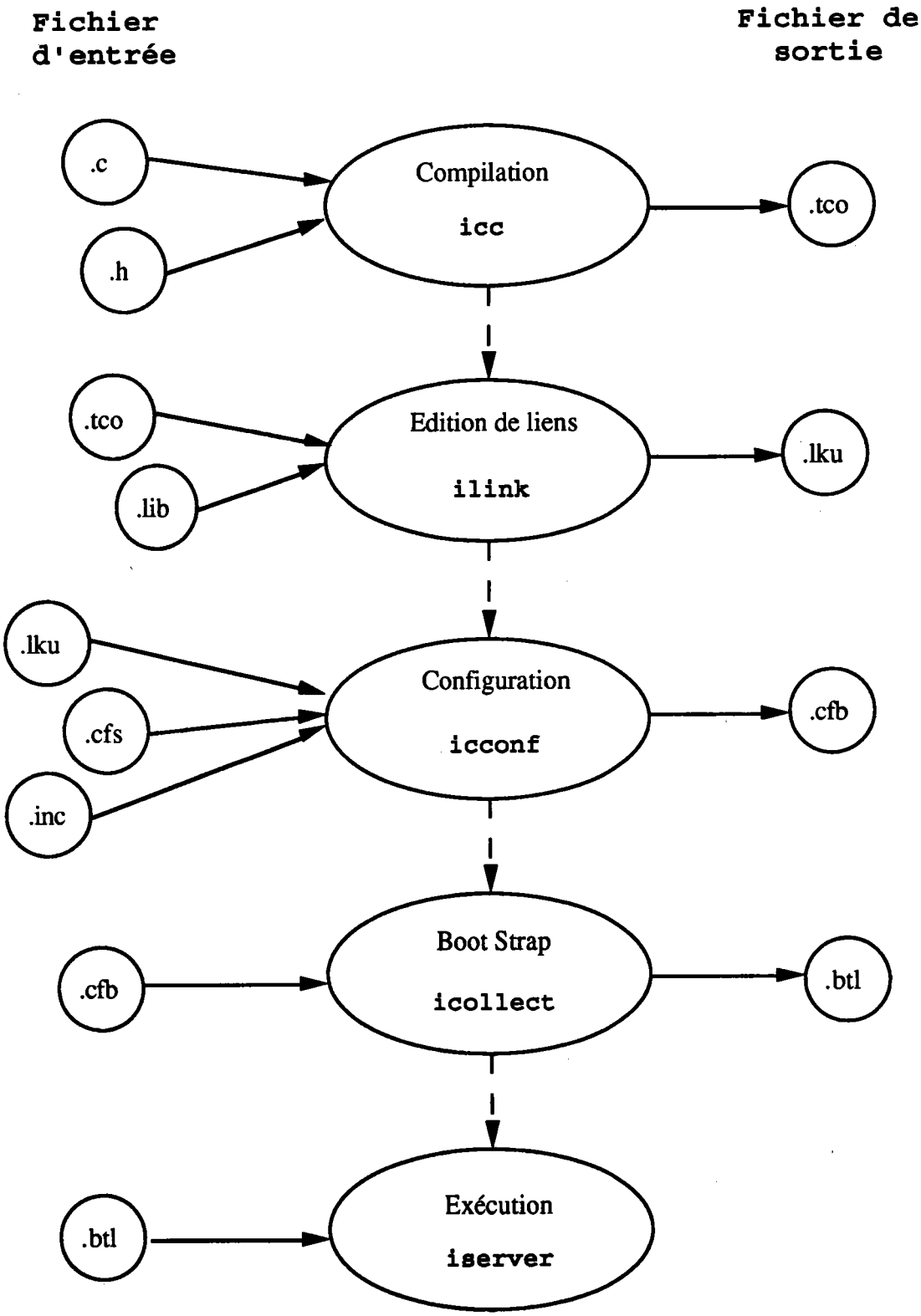
Partitionnement de l'image en vue d'un traitement par le réseau de 4 transputers en parallèle 6

Programme INMOS ANSI-C correspondant au traitement effectué par le transputer maître (masterqv.c) 7

Programme INMOS ANSI-C correspondant au traitement effectué par chaque transputer esclave (workerqv.c) 12



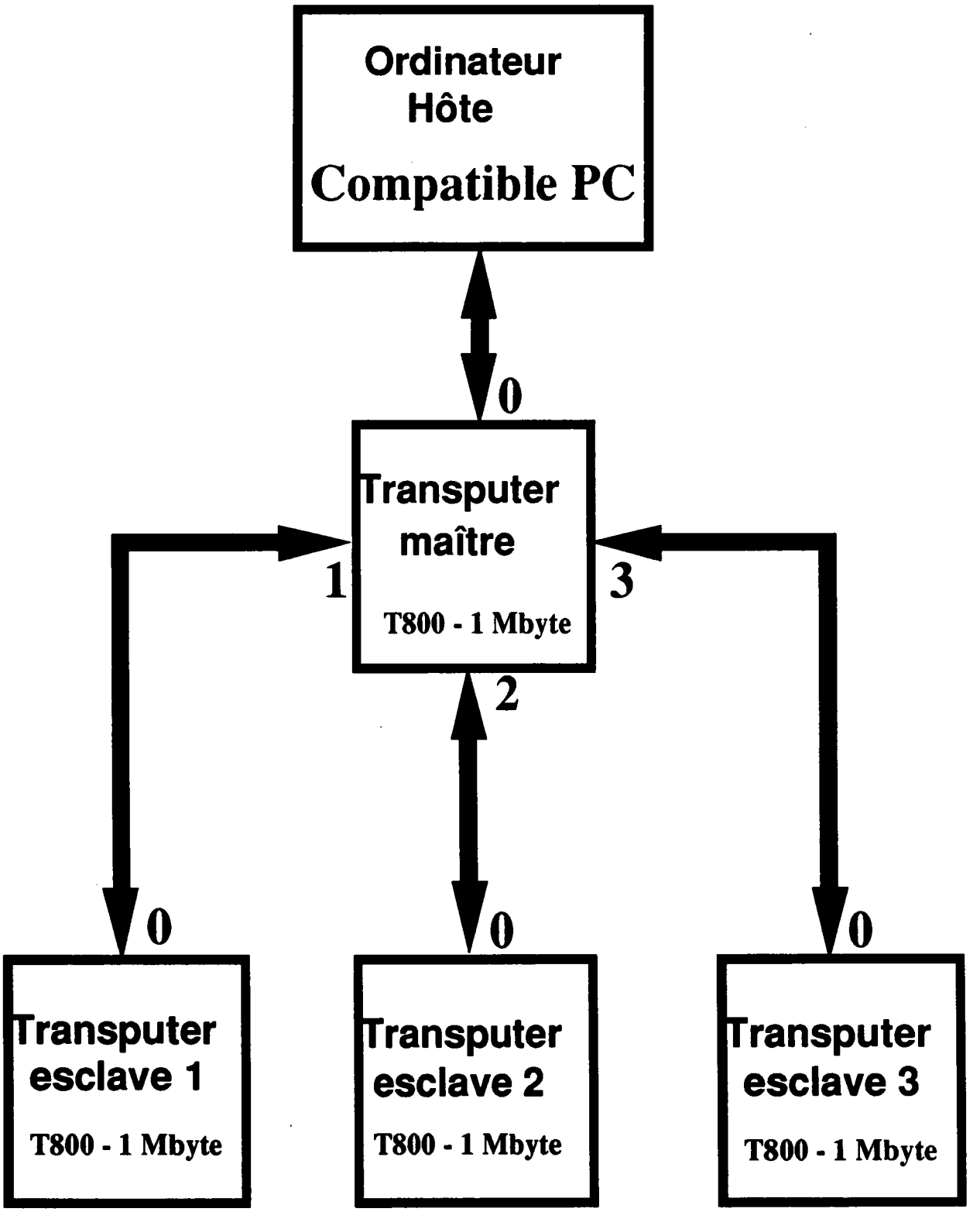
Architecture du transputer (Famille T800)



Développement d'une application sur un système multi-transputers

Principales étapes et fichiers conventionnels en INMOS ANSI-C

CONFIGURATION du RESEAU
4 transputers IMS T800 - 25 MHz



/* **qv.cfs : PROGRAMME DE CONFIGURATION DU RESEAU (4 IMS T800 - 1MO - 25 MHz)**

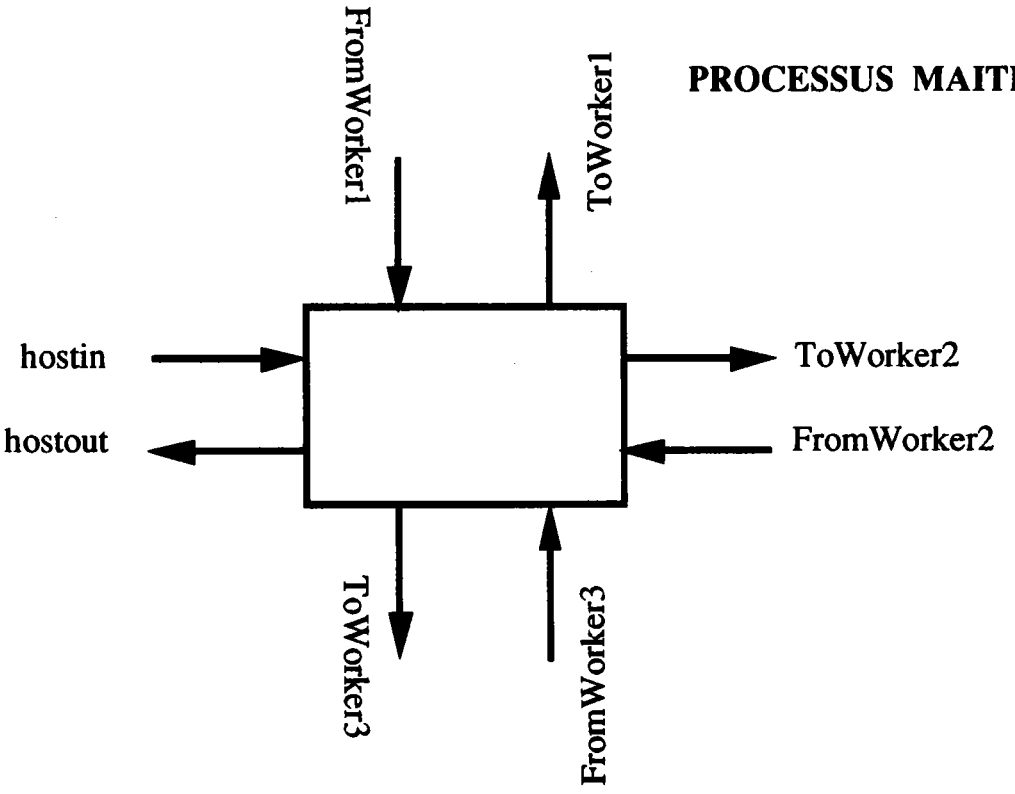
Alain NYECK - TDF / CERLOR - Juillet 1991 */

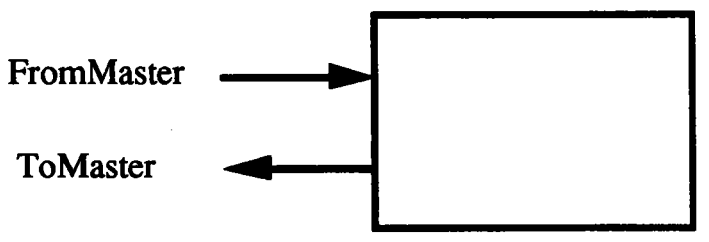
```
/* Définition de l'architecture matérielle  - 4 x T800 - 1 MO */

T800  (memory = 1M)   Maitre , Esclave[3];

connect  Maitre.link[0]  to  host;     /* Lien entre transputer maître et ordinateur hôte  */
rep  i = 0  for  3
        connect Maitre.link[i+1] to Esclave[i].link[0];    /* Liens entre transputer maître
                                                              et transputers esclaves      */

/*---------------------- Description de la structure logicielle du système ----------------------------
```






```
----------------------------------------------------------------------------------------------------*/
```

```
process (stacksize = 20K, heapsize = 50K);
rep i = 0 for 3
        process (interface (input FromMaster, output ToMaster)) Worker[i];
process (interface (input hostin, output hostout,
                    input FromWorker[3], output ToWorker[3])) Master;


/* Définition des interconnexions logicielles */

input hostinput;
output hostoutput;
connect Master.hostin to hostinput;
connect Master.hostout to hostoutput;
rep i = 0 for 3
{
        connect Master.FromWorker[i] to Worker[i].ToMaster;
        connect Master.ToWorker[i] to Worker[i].FromMaster;
}

/* Prise en compte du code obtenu après compilation et édition de liens */

use "masterqv.lku" for Master;
rep i = 0 for 3
        use "workerqv.lku" for Worker[i];

/* Transfert des processus sur les transputers */

place Master on Maitre; /* Exécution du processus maître sur le transputer maître */
rep i = 0 for 3
        place Worker[i] on Esclave[i]; /* Exécution des processus esclaves sur les
                                          transputers esclaves */
place hostinput on host;
place hostoutput on host;

place Master.hostin on Maitre.link[0];
place Master.hostout on Maitre.link[0];

rep i = 0 for 3
{
        place Master.ToWorker[i] on Maitre.link[i+1];
        place Master.FromWorker[i] on Maitre.link[i+1];

        place Worker[i].ToMaster on Esclave[i].link[0];
        place Worker[i].FromMaster on Esclave[i].link[0];
}
```

PARTITIONNEMENT de L'IMAGE EN VUE d'un TRAITEMENT par 4 TRANSPUTERS en PARALLELE

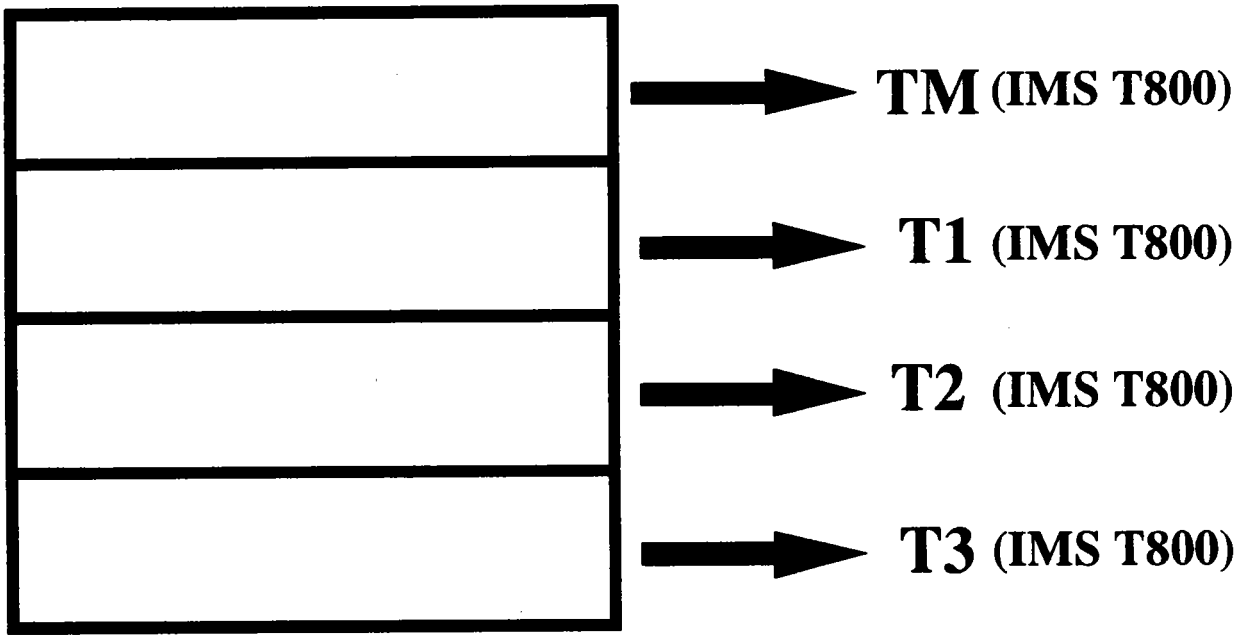


Image à coder
256x256 pixels sur 8 bits

Parallélisation en mode SPMD

(Single Program Multiple Data streams)

```
/*-------------------------------------------------------------------------
                masterqv.c  (version INMOS ANSI-C)

Codage de fichiers image par QUANTIFICATION VECTORIELLE.

Traitement effectué par le transputer maître TM connecté aux
transputers esclaves T1 , T2 et T3 par les liens 1, 2 et 3.
Le transputer maître envoie 1/4 des données à traiter à chacun des
transputers esclaves, ainsi que le dictionnaire, puis traite le dernier
1/4 de données restantes. Après quantification, TM reçoit les codes
retenus par T1, T2 et T3.

            Alain NYECK  -  TDF / CERLOR  -  Juillet 1991
-------------------------------------------------------------------------*/


#include    "globvar.h"                 /* Fichier de variables globales */

int main()
{
FILE            *fichier;

Channel         *from_worker1, *to_worker1;
Channel         *from_worker2, *to_worker2;
Channel         *from_worker3, *to_worker3;

unsigned int    NbVectMax, NbVectWork, NbVectDico;
unsigned int    deb, debIma, i, j, k, l, x;
int             distMink(unsigned char *,  unsigned char *,  int *);
unsigned int    deb1, Code;
unsigned char   Vecteur[dimVecteur], Centre[dimVecteur];
unsigned char   Indice[256], IndTmp[256];
int             Rang[256];

int             Ind, ind, alpha, min, dmin, min2[256];
unsigned int    Size_Image, Size_Ima_Work, Size_Dico, Taille;


        NbVectMax = 8000;               /* Nombre total de vecteurs à traiter      */
        NbVectWork = 2000;              /* Nombre de vecteurs traités par chaque
                                           transputer  (T1, T2, T3 et TM)          */
        NbVectDico = 256;               /* Nombre de vecteurs du codebook          */

        Size_Image = (int)dimVecteur*NbVectMax;
        Size_Ima_Work = (int)dimVecteur*NbVectWork;
        Size_Dico = (int)dimVecteur*NbVectDico;
        Taille = Size_Ima_Work + Size_Dico;   /* Taille des données transmises à
                                                 T1, T2 et T3  par le transputer maître TM */
```

```
from_worker1 = (Channel *)get_param(3);
to_worker1 = (Channel *)get_param(6);
from_worker2 = (Channel *)get_param(4);
to_worker2 = (Channel *)get_param(7);
from_worker3 = (Channel *)get_param(5);
to_worker3 = (Channel *)get_param(8);

if ((fichier = fopen("training.ima", "rb")) == NULL)
{
        printf("Le fichier à traiter doit être <training.ima> \n");
        abort();
}
fread((unsigned char *)image_source, sizeof(char), Size_Image, fichier);
fclose(fichier);

if ((fichier = fopen("training.dic", "rb")) == NULL)
{
        printf("Le fichier dictionnaire doit être  <training.dic> \n");
        abort();
}
fread((unsigned char *)dictionnaire, sizeof(char), Size_Dico, fichier);
fclose(fichier);

/*-----------------------------------------------------------------------------------------
   TM transmet à chacun des transputers  T1, T2 et T3  une partie des vecteurs
    à traiter. Le reste des vecteurs sera traité par  TM
-----------------------------------------------------------------------------------------*/

memcpy((unsigned char *)image_Work,
          (unsigned char *)&image_source[Size_Ima_Work], Size_Ima_Work);
memcpy((unsigned char *)&image_Work[Size_Ima_Work], dictionnaire, Size_Dico);

ChanOut(to_worker1, (unsigned char *)image_Work, Taille);

memcpy((unsigned char *)image_Work,
          (unsigned char *)&image_source[2*Size_Ima_Work], Size_Ima_Work);
memcpy((unsigned char *)&image_Work[Size_Ima_Work], dictionnaire, Size_Dico);

ChanOut(to_worker2, (unsigned char *)image_Work, Taille);

memcpy((unsigned char *)image_Work,
          (unsigned char *)&image_source[3*Size_Ima_Work], Size_Ima_Work);
memcpy((unsigned char *)&image_Work[Size_Ima_Work], dictionnaire, Size_Dico);

ChanOut(to_worker3, (unsigned char *)image_Work, Taille);
```

```
/*-----------------------------------------------------------------
Calcul des distances entre tous les vecteurs du codebook
  en vue de l'élimination par inégalité triangulaire.
-----------------------------------------------------------------*/

k = 0;
for (i = 0; i < NbVectDico-1; i++)
{
        for (x = 0; x < dimVecteur; x++)
                Centre[x] = dictionnaire[k++];
        l = k;
        for (j = i+1; j < NbVectDico; j++)
        {
                dmin = 255;
                for (x = 0; x < dimVecteur; x++)
                        Vecteur[x] = dictionnaire[l++];
                x = distMink(Centre, Vecteur, &dmin);
                DicoDist[i][j] = DicoDist[j][i] = dmin;
        }
        DicoDist[i][i] = DicoDist[j][j] = 0;
}


for (i = 0; i < NbVectDico; i++)
{
        Indice[i] = IndTmp[i] = i;
        Rang[i] = (int)dimVecteur*i;
}

for (i = 0; i < 256; i++)
        min2[i] = i+i;


/*-----------------------------------------------------------------
  Recherche du ppv de chaque vecteur traité par  TM
-----------------------------------------------------------------*/

j = debIma = 0;

while (j < NbVectWork)
{
        Ind = k = 0;
        ind = Indice[k];
        dmin = 255;
        deb = Rang[ind];

        for (i = 0; i < dimVecteur; i++)
                Vecteur[i] = image_source[debIma++];
```

```
        for (i = 0; i < dimVecteur; i++)
                Centre[i] = dictionnaire[deb++];


        min = distMink(Vecteur, Centre, &dmin);
        Code = ind;

        while (++k < NbVectDico)
        {
                ind = Indice[k];
                x = DicoDist[Code][ind];
                if (x < min2[min])
                {
                        deb = Rang[ind];
                        for (i = 0; i < dimVecteur; i++)
                                Centre[i] = dictionnaire[deb++];
                        alpha = distMink(Vecteur, Centre, &dmin);
                        if (alpha < min)
                        {
                                min = alpha;
                                Code = ind;
                                Ind = k;
                        }
                }
        }

        CodeVect[j] = Code;

/*-----------------------------------------------------
  Réorganisation dynamique du dictionnaire.
  Le vecteur retenu est placé en tête de liste
  -----------------------------------------------------*/

        if (Ind > 0)
        {
                deb = Ind;
                deb1 = deb+1;
                memcpy(&IndTmp[1], Indice, deb);
                IndTmp[0] = Code;
                memcpy(Indice, IndTmp, deb1);
        }

        ++j;
}
```

```
        /*-------------------------------------------------------------------------------
          Réception des codes correspondant aux vecteurs traités respectivement
          par les transputers  T1, T2 et T3
        -------------------------------------------------------------------------------*/

        ChanIn(from_worker1, (unsigned char *)&CodeVect[NbVectWork], NbVectWork);
        ChanIn(from_worker2, (unsigned char *)&CodeVect[2*NbVectWork], NbVectWork);
        ChanIn(from_worker3, (unsigned char *)&CodeVect[3*NbVectWork], NbVectWork);

        if ((fichier = fopen("training.cod", "wb")) == NULL)
        {
                printf("Le fichier de codes  <training.cod>  ne peut être ouvert \n");
                abort();
        }
        fwrite((unsigned char *)CodeVect, sizeof(char), NbVectMax, fichier);
        fclose(fichier);

        exit_terminate(EXIT_SUCCESS);
}



/* ------------------------------------------------------------
  Distance de Minkowski partielle entre 2 vecteurs
 ------------------------------------------------------------ */

int distMink(X, Y, distmin)
unsigned char        X[dimVecteur], Y[dimVecteur];
int                  *distmin;
{
int     i;
int     dist, max;

        dist = (int)X[0] - (int)Y[0];
        max = abs(dist);
        for (i = 1; (i < dimVecteur) && (max < *distmin); i++)
        {
                dist = (int)X[i] - (int)Y[i];
                dist = abs(dist);
                if (dist > max)
                        max = dist;
        }
        if (max < *distmin)
                *distmin = max;
        return(max);
}
```

```
/*-----------------------------------------------------------------------
                    workerqv.c  (version INMOS ANSI-C)

Codage de fichiers image par QUANTIFICATION VECTORIELLE.

Traitement effectué par chacun des transputers esclaves T1 , T2 et T3
connectés au transputer maître TM  par les liens 1, 2 et 3.
Le transputer esclave reçoit les données à traiter du transputer maître, ainsi
que le dictionnaire.  T1, T2 et T3  effectuent le même traitement sur des
données différentes (parallélisation en mode SPMD "Single Program
Multiple Data streams").  Après quantification, les codes retenus par le
transputer esclave sont transmis au transputer maître pour le décodage.

                Alain NYECK  -  TDF / CERLOR  -  Juillet 1991
-----------------------------------------------------------------------*/


#include     "varglob.h"              /* Fichier de variables globales  */

int main()
{

Channel          *from_master, *to_master;

unsigned int      NbVectMax, NbVectDico;
unsigned int      deb, debIma, i, j, k, l, x;
int               distMink(unsigned char *,  unsigned char *,  int *);
unsigned int      deb1, Code;
unsigned char     Vecteur[dimVecteur], Centre[dimVecteur];
unsigned char     Indice[256], IndTmp[256];
int               Rang[256];

int               Ind, ind, alpha, min, dmin, min2[256];
unsigned int      Size_Image, Size_Dico, Taille;


        NbVectMax = 2000;         /* Nombre de vecteurs à traiter   */
        NbVectDico = 256;         /* Nombre de vecteurs du codebook  */
        Size_Image = (int)dimVecteur*NbVectMax;
        Size_Dico = (int)dimVecteur*NbVectDico;
        Taille = Size_Image + Size_Dico;

        from_master = (Channel *)get_param(1);
        to_master = (Channel *)get_param(2);

        /* Réception des vecteurs à traiter envoyés par le transputer maître TM  */

        ChanIn(from_master, (unsigned char *)image_source, Taille);
```

```
k = Size_Image;
memcpy(dictionnaire, &image_source[k], Size_Dico);

/*-----------------------------------------------------------------
Calcul des distances entre tous les vecteurs du codebook
en vue de l'élimination par inégalité triangulaire.
-----------------------------------------------------------------*/

k = 0;
for (i = 0; i < NbVectDico-1; i++)
{
        for (x = 0; x < dimVecteur; x++)
                Centre[x] = dictionnaire[k++];
        l = k;
        for (j = i+1; j < NbVectDico; j++)
        {
                dmin = 255;
                for (x = 0; x < dimVecteur; x++)
                        Vecteur[x] = dictionnaire[l++];
                x = distMink(Centre, Vecteur, &dmin);
                DicoDist[i][j] = DicoDist[j][i] = dmin;
        }
        DicoDist[i][i] = DicoDist[j][j] = 0;
}


for (i = 0; i < NbVectDico; i++)
{
        Indice[i] = IndTmp[i] = i;
        Rang[i] = (int)dimVecteur*i;
}

for (i = 0; i < 256; i++)
        min2[i] = i+i;


/*-----------------------------------------------------------
  Recherche du ppv de chaque vecteur de la source
 -----------------------------------------------------------*/

j = debIma = 0;

while (j < NbVectMax)
{
        Ind = k = 0;
        ind = Indice[k];
        dmin = 255;
        deb = Rang[ind];
```

```
        for (i = 0; i < dimVecteur; i++)
                Vecteur[i] = image_source[debIma++];

        for (i = 0; i < dimVecteur; i++)
                Centre[i] = dictionnaire[deb++];

        min = distMink(Vecteur, Centre, &dmin);
        Code = ind;

        while (++k < NbVectDico)
        {
                ind = Indice[k];
                x = DicoDist[Code][ind];
                if (x < min2[min])
                {
                        deb = Rang[ind];
                        for (i = 0; i < dimVecteur; i++)
                                Centre[i] = dictionnaire[deb++];
                        alpha = distMink(Vecteur, Centre, &dmin);
                        if (alpha < min)
                        {
                                min = alpha;
                                Code = ind;
                                Ind = k;
                        }
                }
        }

        CodeVect[j] = Code;

/*-------------------------------------------------------
  Réorganisation dynamique du dictionnaire.
  Le vecteur retenu est placé en tête de liste
 -------------------------------------------------------*/

        if (Ind > 0)
        {
                deb = Ind;
                deb1 = deb+1;
                memcpy(&IndTmp[1], Indice, deb);
                IndTmp[0] = Code;
                memcpy(Indice, IndTmp, deb1);
        }

        ++j;
}
```

```
        /* Transmission des codes sélectionnés au transputer maître  TM */

        ChanOut(to_master, (unsigned char *)CodeVect, NbVectMax);
}



/* ------------------------------------------------------------
   Distance de Minkowski partielle entre 2 vecteurs
 ------------------------------------------------------------ */

int distMink(X, Y, distmin)
unsigned char   X[dimVecteur], Y[dimVecteur];
int             *distmin;
{
        int     i;
        int     dist, max;

        dist = (int)X[0] - (int)Y[0];
        max = abs(dist);
        for (i = 1; (i < dimVecteur) && (max < *distmin); i++)
        {
                dist = (int)X[i] - (int)Y[i];
                dist = abs(dist);
                if (dist > max)
                        max = dist;
        }
        if (max < *distmin)
                *distmin = max;
        return(max);
}
```

Résumé

Des algorithmes rapides de quantification vectorielle sont proposés pour le codage d'image. Ces algorithmes exploitent les propriétés métriques de la distance aussi bien que l'intercorrélation entre les blocs image adjacents pour augmenter la vitesse de codage des vecteurs de l'espace à quantifier d'une part, et accélérer la convergence de l'algorithme de classification LBG d'autre part. L'implémentation de ces algorithmes sur des machines multi-processeurs à base de transputers permet d'envisager l'utilisation de la quantification vectorielle pour des applications de codage d'image en temps réel.

Mots clé

Quantification vectorielle, algorithmes rapides, classification, codage d'image, transputer.

Abstract

Fast search algorithms for image coding vector quantization are proposed. The presented algorithms exploit the intercorrelation between neighbouring image blocks as well as the geometrical properties of the used distance to eliminate the necessity of performing many distortion calculations so that partial distances need to be computed for only a few remaining candidate codevectors. The proposed algorithms can therefore achieve a fast dynamic search by progressive approximation to the nearest prototype and speed up the clustering classification LBG algorithm. The implementation of the proposed fast search algorithms on multitransputer systems provide direct support for real time image coding using vector quantization applications.

Key words

Vector quantization, fast search algorithms, classification, image coding, transputer.