République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Mouloud Mammeri, Tizi-Ouzou
Faculté de Génie Electrique et d'Informatique

Département Informatique.

MEMOIRE DE MASTER

En Informatique

Option : Système informatique

Thème :

Implémentation et évaluation d'une nouvelle transformée pour la compression de données sans perte.

Proposé et dirigé par :

M^{r} : SADOU Samir

Réalisé par :

M^{elle} : KHERBACHE Yamna

M^{elle} : MAMERI Chahrazed

Promotion : 2012/2013

Remerciement

Remerciement

Nous remercions DIEU le tout puissant, qui nous a donné la force et la patience pour l'accomplissement de ce travail.

Nous remercions chaleureusement notre promoteur, Mr SADOU. Samir pour nous avoir proposé ce sujet, en nous faisant confiance, ainsi que pour avoir dirigé notre travail avec ses orientations, ses précieux conseils , ses remarques constructives et surtout son sérieux et sa disponibilité .

Un grand merci aux membres de jury pour l'honneur qu'ils nous ont attribué pour évaluer et juger notre travail.

Merci à nos familles et amis pour leurs soutiens et leurs encouragements.

Et à tous ceux qui de prés ou de loin nous ont aidés à l'aboutissement de cette quête.

Merci infiniment à tous.

Dédicace

DEDICACE

Je dédie ce modeste travail :

A la mémoire de mon cher père que DIEU le garde dans son vaste Paradies.

A mon adorable mère que j'adore pour son soutien et son dévouement tout en long de mes études.

A mon cher frère que j'aime énormément : Ahmed

Et à mes adorables sœurs : Ghania, Zahia, Kenza, Nadira.

A mes grands-parents.

A mes très chers oncles : Arezki, Ahcen, Nadir, Seddik, Hakim, Hamid.

A mes adorables tantes : Taous, Nadira.

A mes cousins(es).

A ma meilleure amie Malia et à toute sa famille.

A tout mes amis(es).

A mon binôme Chahrazed.

A Sofiane.

Je remercie tous ceux et celles qui m'ont aidé à réaliser ce travail.

Yamna.

DEDICACE

Je dédie ce modeste travail :

A ceux qui me pérenne la dignité, l'honneur et la joie de ma vie mes très chers parents Mouloud et Ouiza.

A la mémoire de mes grands parents

A Nouredine qui m'a toujours encouragé et soutenu

A mes très chers frères Ali, Sofiane et Farid

A mon cher grand frère Mohamed et sa femme Samah et leurs adorables petits enfants Lina, Maria-Farah et notre ange Abdou

A mes chères sœurs Nacéra, Lynda et Dihia

A toute la famille TALEB

A mon binôme Yamna et toute sa famille

A mes amies: Malika, Sabrina, Nadia, Amel, Aziza

A Sofiane, Farida

A toutes les personnes qui nous ont aidées de prés ou de loin

Chahrazed

Liste des figures

Chapitre I : Généralités sur la compression de données.

Chapitre III : Analyse et conception de la méthode proposée.

Chapitre IV : Implémentation et évaluation.

Liste des tableaux

Chapitre I : Généralités sur la compression de données.

Chapitre II : La méthode proposée.

Chapitre IV : Implémentation et évaluation

Introduction générale

Introduction générale

Le développement technologique et les exigences des utilisateurs entrainent le traitement d'une quantité importante de données, ainsi, depuis les débuts des technologies de l'information, le problème de l'exploitation optimale des voies de communication et des capacités de stockage est toujours resté un sujet d'actualité. C'est donc dans cette optique que la compression de données est devenue un enjeu crucial.

La compression de données consiste à réduire la taille de l'information pour son stockage et son transport, pour que le temps de transfert des fichiers soit plus court et donc à moindre coût.

De multiples études ont été menées sur les méthodes par transformation qui consistent à effectuer des modifications étudiées sur le fichier d'entrée à la compression et cela conduit à des taux de compression très important, l'idée de base est de changer la structure du contenu du fichier pour augmenter le niveau de redondance et par conséquent diminuer l'entropie de ce dernier pour le soumettre en fin à un algorithme de compression entropique tels que Huffman , RLE, … etc. Une fois compressées, les données ne sont plus directement accessibles et il est nécessaire de les décompresser pour qu'elles redeviennent intelligibles.

Pour mener à terme notre travail et à fin de rendre la démarche compréhensible, le présent mémoire est structuré de la manière suivante :

- Le premier chapitre, présentera des généralités sur la compression de données.
- Le second chapitre, nous allons expliquer les transformées dans la compression de données ainsi que la méthode proposée.
- Le troisième chapitre est consacré pour l'étude et la conception de la méthode décrite dans le deuxième chapitre.
- Le dernier chapitre détaillera l'implémentation ainsi le résultat des différents tests effectués sur cette méthode.

Chapitre I :

Généralités sur la compression de données.

I. Introduction

Les données informatiques utilisent un espace disque non négligeable, de même lorsqu'on souhaite transmettre des données (flux vidéo, sonore…) sur le réseau Internet, alors la bande passante allouée pour le transfert est très limitée en termes de débit et de capacité de gestion de gigantesque volume de paquets de données, ce qui diminue considérablement la qualité du service Internet. C'est pour ces raisons que la compression de données est presque toujours utilisée, pour réduire autant que possible la quantité de données à stocker ou à transmettre.

II. La Théorie de l'information [1]

La théorie de l'information est due à Shannon (vers 1948), avec l'influence des grands théoriciens de l'informatique (Turing, Von Neumann, Wiener). Elle est la première grande théorie systématique (et même mathématique) de la communication. Elle définit celle-ci comme un transfert d'*information* entre un *é*metteur et un *r*écepteur à travers un *canal* de communication. Cette information ne peut être véhiculée que sous la forme d'un *code* commun aux deux parties. Les défauts de transmission de l'information par le canal sont appelés ***bruit.***

La théorie de l'information fournit une mesure quantitative de la notion d'information apportée par un message (ou une observation). Elle s'appuie non seulement sur les communications mais aussi sur l'informatique et la statistique.

II.1.Concepts de la théorie de l'information [2]

La théorie de l'information a pris sa dimension avec l'élaboration de sa théorie mathématique par SHANON et WEAVER en définissant les concepts de quantité d'information, entropie et redondance.

II.1.1. Quantité d'information

La définition mathématique de la quantité d'information repose sur une théorie probabiliste. La quantité d'information contenue dans un flot de données est liée à sa probabilité d'apparition.

Soit *S*, une source d'information (alphabet) $S=(X_1, X_2, X_3, \ldots\ldots, X_k)$ de *K* symboles. Et soit $P(X_i)$ la probabilité d'apparition deX_i.

La quantité d'information (L_i) contenue dans un flot de données est une fonction de l'inverse de la probabilité d'apparition de ce symbole, elle est donnée par la relation suivante:

$$\boldsymbol{L_i = \log_2\left(1/P(X_i)\right) = -\log_2(P(X_i))}$$

L'unité de la quantité d'information est le Shannon.

-L'intérêt de cette quantité d'information :

Lors de la transmission par un canal, nous souhaitons récupérer l'information sans distorsion, autrement dit, l'alphabet de sortie du canal doit être le même que celui de l'entrée.

II .1.2. Entropie

C'est une notion empruntée par la thermodynamique. En théorie d'information, l'entropie mesure le degré du hasard dans un flot de données (fichier).Un fichier qui a un faible taux d'entropie, dont les éléments sont tous prévisibles à l'avance, sera beaucoup compactable.

Par exemple un fichier rempli de zéros, la quantité d'information qu'il contient est nulle, le hasard inexistant, l'entropie est minimale. Au contraire un fichier zippé (déjà compacté) contient des octets apparemment aléatoires sans aucune relation les uns avec les autres. La quantité d'information y est maximale, presque par définition puisqu'il s'agit d'un fichier déjà compressé, donc dont on a à priori supprimé toutes les redondances. Il contient par ailleurs une dose de hasard maximale, soit une entropie maximale.

L'entropie H d'un fichier est l'information moyenne contenue par chaque symbole, elle est donnée par la relation suivante.

$$\boldsymbol{H = \sum_{i}^{n} L_i\, P(X_i) = -\sum_{i}^{n} P(X_i)\ \log_2(P(X_i))}$$

Avec n : le nombre maximum de symboles (Si) existant.

- ✓ L'unité de l'entropie est le bit (binary) ou le Shannon(Sh).
- ✓ L'entropie représente aussi le nombre de bits par symbole qui sont nécessaires pour coder ce symbole. Ce nombre de bits est la longueur L du code **L≥H.**
- ✓ **Propriétés**
  - $0 \leq H \leq \log_2(n)$.
  - $H = 0$ lorsque $P(X_i) = 1\ et\ P(X_j) = 0, i \neq j$.
  - $H = \log_2(n)$ lorsque $P(X_i) = {}^{1}/_{n}$, $\forall\, i$.

II.1.3. La redondance [3]

Mesure la quantité d'information significative dans le débit de décision généré par une source, il est définit par : **R=D-H**

La compression de données se base sur la détection et l'élimination des redondances dans un flot de données. Et sachant que l'information informatique quel que soit son format (texte, son, image, ….) est logiquement représentée par une combinaison de 256 octets différents (si on considère l'octet comme unité de base) appelé alphabet ; cela implique que chaque flot de

données qui a une taille supérieure a celle de l'alphabet (nombre de symboles) présente nécessairement une redondance.

✓ **Exemple:** Soit *A* l'alphabet et *F* le flot de données.

A= {0,1} de taille 2 symboles, F=1001…10.F présente la redondance des symboles 0 et 1 dès que F dépasse la taille de 2. De ce fait, on peut dire que les flots de données de grande taille (plusieurs Mo) ne sont que le résultat de la redondance des symboles de l'alphabet qui est un ensemble borné.

✓ **Types de redondances [4]**

Il ya plusieurs types de redondances qui peuvent exister dans un même flot de données :

➢ **Redondance de caractères :**

Dans un flot de données, certains caractères peuvent avoir une fréquence d'apparition plus élevée que d'autre. Il est donc possible de tirer profit de cette caractéristique pour réaliser une compression en agissant des codes courts aux caractères les plus fréquents et les codes longs aux caractères plus rares.

➢ **Redondance de groupe de caractères :**

Dans l'alphabet de la langue anglaise par exemple le nombre de combinaisons de caractères qui forme un mot est inférieure au nombre de combinaisons possibles, comme les mots « the », « are », « what », « was », peuvent apparaître dans un texte par contre il est presque impossible de rencontrer « zzzzdddd », donc on va associer au groupe de symboles le plus fréquent un code cours et au groupe de symboles rares un code long.

➢ **Corrélation entre les symboles :**

Dans un flot de données, l'apparition d'un symbole peut augmenter la probabilité d'apparition d'un autre symbole après le premier.

Par exemple lorsqu'on rencontre un 'T' dans un texte de la langue anglaise, il est fort probable que la lettre suivante serait un 'H' a cause de la forte corrélation qui existe entre ces deux lettres. En terme de probabilité, la phrase précédente peut s'écrire : P= (H/T) ≈1 qui se lit : « la probabilité d'avoir un H sachant qu'on a déjà un T est proche de 1 ». Ainsi, il est intéressant d'affecter des codes à des groupes de symboles fortement corrélés.

III. Le Codage [5]

III.1. définition

Il s'agit d'une règle permettant de convertir une information sous une forme différente de sa représentation initiale.

✓ Ainsi, on peut scinder les « codes » en plusieurs grandes familles :

- Les codes de communication (Morse, Baudot…..).
- Les codes de représentation (ASCII, Base64,…).
- Les codes de protection (cryptographie,….).
- Les codes de compression (le codage VLC,……).
- Les codes d'identification (Code-barres,…..).

III.2. La propriété d'un code

Pour avoir un codage correct, il faut qu'il vérifie les propriétés suivantes :

- ✓ Tous les mots du code peuvent êtres distingués.
- ✓ Le décodage ne donne lieu à aucune ambigüité.
- ✓ Aucun mot du code n'est un sous mot initial d'un autre.

IV. La compression des données

IV.1. Définition [6]:

La compression de données ou codage de source est l'opération informatique qui consiste à transformer une suite de bits A en une suite de bits B plus courte, en utilisant un algorithme particulier.

Il s'agit d'une opération de codage, c'est-à-dire changer la représentation de l'information, dans le but de rendre la représentation compressée plus courte que la représentation originale. La **décompression** est l'opération inverse de la compression.

- ✓ La théorie de la compression de données est issue de la théorie de l'information.

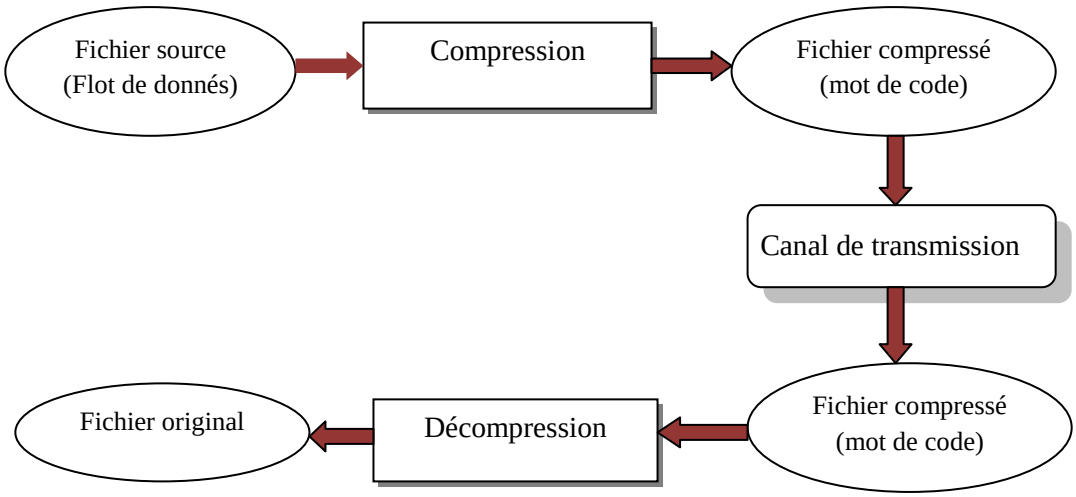

Figure I.1 principe de compression/décompression.

Les différents algorithmes de compression sont basés sur 3 critères :

-Le taux de compression [5] : est le rapport de la taille du fichier compressé sur la taille du fichier initial. Il sert à évaluer la qualité de la compression en faisant l'opération suivante :

$$\textbf{Taux de compression} = \frac{(taille\ originale - taille\ finale) * 100}{taille\ originale}$$

-La qualité de compression : sans/avec pertes (avec le pourcentage de perte).

-La vitesse de compression et de décompression.

Un compresseur utilise un algorithme qui sert à optimiser les données en fonction du type de données à compresser ; un décompresseur est donc nécessaire pour reconstruire les données grâce à l'algorithme dual de celui utilisé pour la compression.

La méthode de compression dépend du type de données à compresser car une image ou un fichier audio ne représentent pas le même type de données.

V. Classification des algorithmes de compression

V.1.Compression symétrique/asymétrique [7]

V.1.1 La compression symétrique

Un algorithme de compression est dit symétrique lorsque le codeur et le décodeur utilisent le même procédé. Le temps d'exécution est donc quasiment égal pour les deux étapes. Par exemple, une application de transmission de données où la compression et la décompression sont les deux faits en temps réels sera généralement implémentée avec un algorithme symétrique si l'on veut atteindre la plus grande efficacité.

V.1.2 La compression asymétrique

Une compression asymétrique est recherchée, lorsque le décodage doit être beaucoup plus rapide que la compression, par exemple, imaginons que nous ayons une base de données où les données seront compressées une seule fois mais décompressées un grand nombre de fois pour la consultation, alors on pourra certainement tolérer un temps beaucoup plus grand pour la compression, dans le but de trouver le taux de compression le plus élevé, que pour la décompression, où là, la vitesse est prédominante. Un algorithme asymétrique qui utilise beaucoup plus de temps CPU pour la compression mais qui est beaucoup plus rapide à la décompression serait un bon choix dans ce cas là. Un exemple classique est l'encyclopédie Encarta de Microsoft.

V.2.Compression physique/logique [8]

La distinction entre compression physique et logique est faite sur la base de comment les données sont compressées ou plus précisément comment est-ce que les données sont réarrangées dans une forme plus compacte.

V.2.1. La compression physique :

La compression physique agit directement sur les données; Cette méthode produit typiquement des résultats incompréhensibles qui apparemment n'ont aucun sens.

Le résultat d'un bloc de données compressées est plus petit que l'original car l'algorithme de compression physique a retiré la redondance qui existait entre les données elles-mêmes.

V .2.2. La compression logique :

La compression logique par contre est effectuée par un raisonnement logique en substituant une information par une information équivalente. Changer "United State of America" en "USA" est un bon exemple de substitution logique car "USA" est dérivé directement de l'information contenue dans la chaîne "United State of America" et garde la même signification.

V.3.Compression statistique/numérique [9]

Pour les algorithmes qui travaillent au niveau statistique, la valeur des motifs ne compte pas. Ce sont les probabilités qui comptent, et le résultat est inchangé par substitution des motifs tandis que pour ceux qui opèrent au niveau numérique, les valeurs des motifs influent sur la compression (Par exemple JPEG), et les substitutions sont interdites.

Enfin le critère de classification le plus pertinent est basé sur la perte des données.

V.4.Compression avec/sans pertes [8]

V.4.1.Compression sans pertes (Lossless)

Une bonne partie des schémas de compression utilisés sont appelés **sans pertes**, cela signifie que lorsque des données sont compressées et ensuite décompressées, l'information originale contenue dans les données a été préservée. Aucune donnée n'a été perdue ou oubliée. Les données n'ont pas été modifiées.

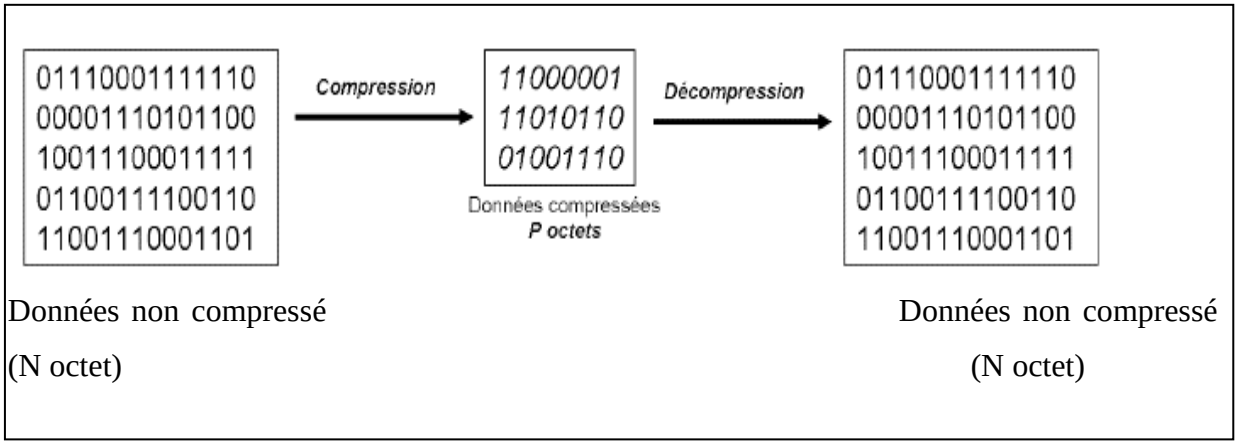

Figure I.2. Principe de la compression sans perte [6].

V.4.1.1.par codage statistique [5]

Le codage statistique a pour but de réduire le nombre de bits utilisés pour le codage des caractères fréquents, et augmenter ce nombre pour des caractères plus rares (exemple : codage de Huffman, Codage de Shannon-Fano).

a) L'algorithme de Huffman [10]

Le codage de Huffman est un algorithme de compression des données sans perte basé sur les fréquences d'apparition des caractères apparaissant dans le document initial. Il a été développé par un étudiant de la MIT, David Albert Huffman en 1952.

Cette technique est largement utilisée car elle est très efficace et nous observons selon le type de données des taux de compression allant de 20% à 90% mais plus généralement entre 30% et 60%.

C'est un codage entropique de type **VLC** (codage à longueur variable) qui consiste à coder les caractères qui apparaissent souvent dans un texte par un code binaire court, et ceux qui apparaissent plus rarement par un code plus long. Par exemple la lettre « e » est très utilisée dans la langue française, tandis que le « z » beaucoup moins.

- **Algorithme de compression :**

1. on cherche la fréquence des caractères.
2. on trie les caractères par ordre décroissant de fréquence.
3. on construit un arbre pour donner le code binaire de chaque caractère.

✓ **Construction de l'arbre**

On relie deux à deux les caractères de fréquence les plus basses et on affecte à ce nœud la somme des fréquences des caractères. Puis on répète ceci jusqu'à ce que l'arbre relie toutes les

lettres. L'arbre étant construit, on met un 1 sur la branche à droite du nœud et un 0 sur celle de gauche.

- **Exemple** : Algorithme de Huffman

Pour illustrer cet algorithme, nous allons coder la phrase suivante : "Le codage est indispensable".

Pour simplifier nous n'allons pas prendre en compte le symbole espace (blanc). Cette phrase est une source de 24 symboles. Tous ces symboles émanent de l'alphabet A.

$A=\{E , S , A , D , I , N , L , B , G , P , T , O , C\}$, Cet Alphabet a N=13 symboles

✓ Le tableau des probabilités d'occurrence est le suivant :

Symbole	**Nombre de fois**	**Probabilités**
E	5	5/24
S	3	3/24
A	2	2/24
D	2	2/24
I	2	2/24
N	2	2/24
L	2	2/24
B	1	1/24
G	1	1/24
P	1	1/24
T	1	1/24
O	1	1/24
C	1	1/24

Tableau I.1. Les probabilités d'occurrences pour la méthode de Huffman.

✓ On applique successivement les étapes de l'algorithme selon le principe de l'algorithme précédemment indiqué. L'arbre de Huffman associé est présenté ci-après.

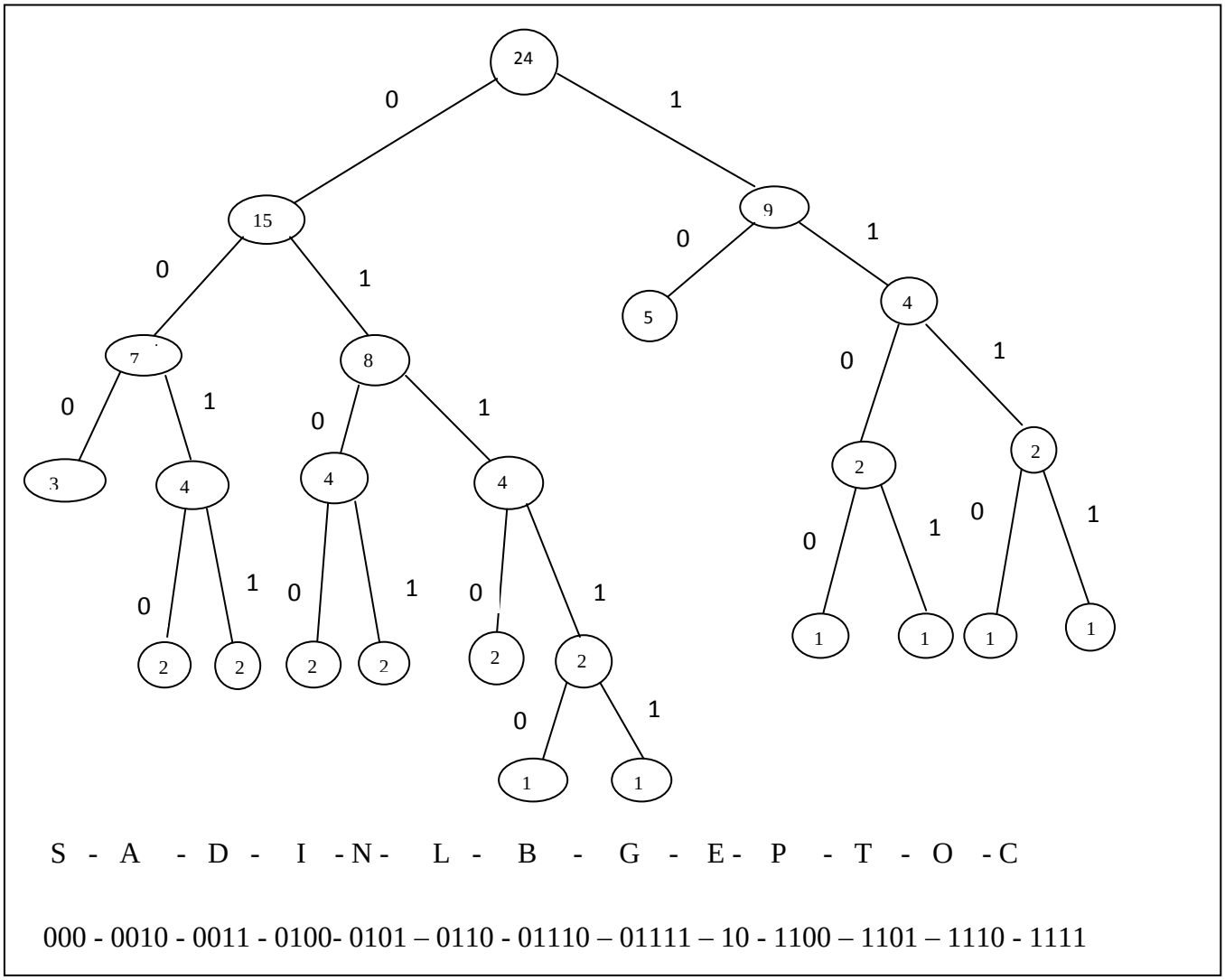

Figure I.3. Arbre de Huffman.

- ✓ Le fichier compressé se compose d'une suite de codes sans séparateur, bien que les codes comportent un nombre variable de bits, car chaque code à la propriété d'unicité de son préfixe.

- **Algorithme de décompression :**

On transmet la bibliothèque de codage, puis on associe les caractères à leur code.

- **Utilité et caractéristiques de compression :**

Cet algorithme permet de compresser aussi bien des images que des textes, on parle de compression statistique ou codage entropique (probabilités d'apparition d'un caractère).

On obtient une compression satisfaisante (50% en moyenne) et un temps de compression assez rapide. En revanche il faut transmettre la bibliothèque en même temps, et il arrive que la taille du fichier soit plus grande que celle du fichier non compressé. De plus il est très sensible

à la perte d'un bit, toutes les valeurs qui suivront ce bit seront fausses lors de la décompression.

b) L'algorithme de Shannon-Fano [10]

Ce procédé est antérieur au codage de Huffman est se base également sur un codage statistique.

- **Algorithme**

1. Construire une table des fréquences d'apparition des symboles triée par ordre décroissant.
2. Diviser cette table en deux parties. Celles-ci doivent avoir une somme de fréquences égale (ou pratiquement égale) à celle de l'autre.
3. Affecter le chiffre 0 à la moitié inférieure, la moitié supérieure prenant la valeur 1.
4. Répéter les opérations 2 et 3 aux deux parties, jusqu'à ce que chaque symbole ne représente plus qu'une partie de la table.

- **Exemple** : Application de l'algorithme de Shannon-Fano

Pour illustrer cet algorithme, nous allons coder la phrase suivante : "Le codage est indispensable".

Pour simplifier nous n'allons pas prendre en compte le symbole espace (blanc). Cette phase est une source de 24 symboles.

Tous ces symboles émanent de l'alphabet A={E , S , A ,D , I , N , L , B ,G , P ,T ,O ,C}, Cet Alphabet a N=13 symboles.

✓ **ETAPE 1**

Symbole	Nombre de fois	Probabilités
E	5	5/24
S	3	3/24
A	2	2/24
D	2	2/24
I	2	2/24
N	2	2/24
L	2	2/24
B	1	1/24
G	1	1/24
P	1	1/24

T	1	1/24
O	1	1/24
C	1	1/24

Tableau I.2. Etape 1 de l'algorithme de Shannon-Fano.

- ✓ **Application successive des étapes 2,3 et 4**

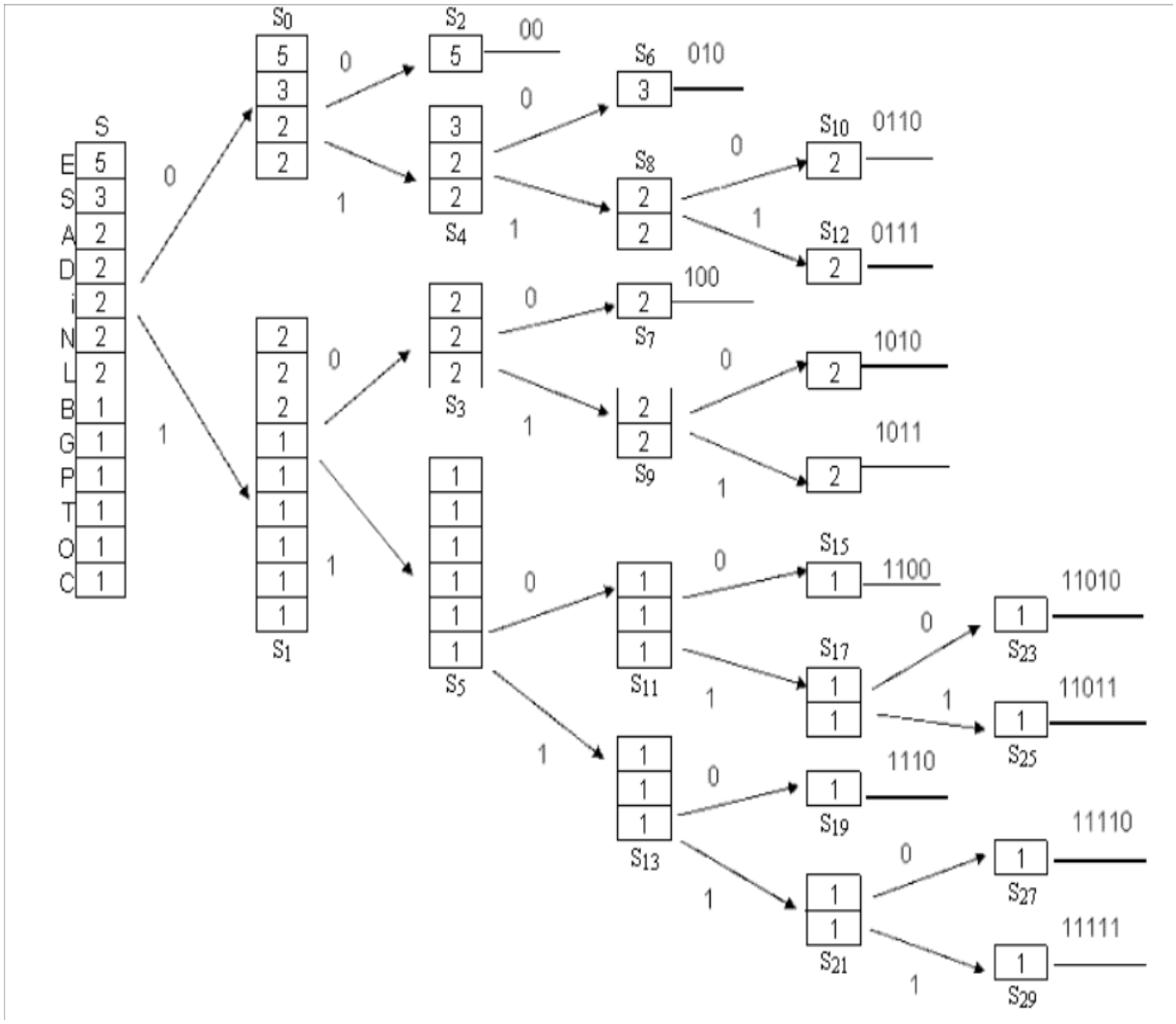

Figure I.4. Exemple algorithme de Shannon-Fano.

V.4.1.2. Par substitution [5] [11] [12]

Le codage de Huffman est efficace quand il y a un petit nombre de types de messages dont quelques-uns couvrent une proportion importante du texte .Quand le nombre de types de messages augmente et que les fréquences sont plus uniformes, le gain est négligeable.

Souvent, le nombre de messages différents est indéterminé à l'avance (par exemple quand il s'agit de mots dans un texte simple) ou trop grand pour justifier un codage à longueur variable (par exemple, le jeu de caractères ASCII). Dans ces cas, on effectue une compression en remplaçant seulement des séquences choisies de texte par des codes plus courts.

Il y a plusieurs façons de procéder, exemples :

– parcourir le texte et remplacer les séquences redondantes par des séquences plus courtes (l'algorithme RLE)

– analyser préalablement le texte pour déterminer les groupes à remplacer et les codes qui les remplacent (l'algorithme LZW).

a) L'algorithme RLE (Run Length Encoding)

RLE est un algorithme de compression de données qui est utilisé par la plupart des formats de fichiers bitmaps tels que TIFF, BMP et PCX. Il a été créé pour compresser n'importe quel type de données sans tenir compte de l'information qu'elle contient.

- **Algorithme de compression :**

- Recherche des caractères répétés plus de n fois (n fixé par l'utilisateur)
- Remplacement de l'itération de caractères par:

1. un caractère spécial identifiant une compression
2. le nombre de fois où le caractère est répété
3. le caractère répété

- **Algorithme de décompression :**

Durant la lecture du fichier compressé, lorsque le caractère spécial est reconnu, on effectue l'opération inverse de la compression tout en supprimant ce caractère spécial.

- **Exemple** : **AAAAARRRRRROLLLLBBBBBUUTTTTTT**

✓ On choisit comme caractère spécial : **@** et comme seuil de répétition : 3

✓ Après compression : **@5A@6RO@4L@5BUU@6T**

✓ Gain : 11 caractères soit 38%

- **Utilité :**

  Essentiellement pour la compression des images (car une image est composée de répétitions de pixels, de couleur identique, codés chacun par un caractère).

- **Caractéristiques de compression :**
  - ✓ algorithme très simple
  - ✓ taux de compression relativement faible (40%)

V.4.1.3. La compression par dictionnaire (substitution de facteurs) [5] [11] [12]

Il consiste à remplacer les séquences (les facteurs) par un code plus court qui est l'indice de ce facteur dans un dictionnaire (exemple : l'algorithme LZW).

a) **L'algorithme LZW (Lempel Ziv Welch)*:***

- **Description de l'algorithme :**

LZW (Lempel-Ziv-Welch) est un algorithme de compression de données sans perte. Il s'agit d'une amélioration de l'algorithme LZ78 inventé par Abraham Lempel et Jacob Ziv en 1978. LZW fut créé en 1984 par Terry Welch d'où son nom.

Cet algorithme de compression consiste à réaliser la construction d'un dictionnaire. Les caractères inférieurs à 256 sont initialement présents dans le dictionnaire. À mesure que l'algorithme examine le texte, il ajoute de nouvelles chaînes de caractère dans de dictionnaire.

- ✓ Ce principe forme la base de la compression LZW (Lempel-Ziv-Welch). La méthode est semblable à la méthode RLE, mais appliquées à des suites d'octets.
- ✓ Cet algorithme réduit la taille des chaînes de caractères (c'est-à-dire les mots) récurrents.

- **Algorithme de compression :**

```
w = Nul;
 tant que (lecture d'un caractère c) faire
   si (wc existe dans le dictionnaire) alors
    w = wc;
  sinon
    ajouter wc au dictionnaire;
   écrire le code de w;
   w = c;
 fin si
fin tant que
écrire le code de w;
```

- **Exemple**

Le tableau suivant illustre le fonctionnement de l'algorithme LZW.

✓ On a la chaine suivante: **TOBEORNOTTOBEORTOBEORNOT**

✓ Le résultat de l'exécution de l'algorithme de compression sur la chaîne précédente donne:

C	W	WC	Sortie	Dictionnaire
T		T		
O	T	TO	T	TO=<256>
B	O	OB	O	OB=<257>
E	B	BE	B	BE=<258>
O	E	EO	E	EO=<259>
R	O	OR	O	OR=<260>
N	R	RN	R	RN=<261>
O	N	NO	N	NO=<262>
T	O	OT	O	OT=<263>
T	T	TT	T	TT=<264>
O	T	TO		
B	TO	TOB	<256>	TOB=<265>
E	B	BE		
O	BE	BEO	<258>	BEO=<266>
R	O	OR		
T	OR	ORT	<260>	ORT=<267>
O	T	TO		
B	TO	TOB		
E	TOB	TOBE	<265>	TOBE=<268>
O	E	EO		
R	EO	EOR	<259>	EOR=<269>
N	R	RN		
O	RN	RNO	<261>	RNO=<270>
T	O	OT		
	OT		<263>	

Tableau I.3. Exemple de l'algorithme LZW.

Après la compression, nous obtenons une séquence de codes de 9 bits sur la sortie: TOBEORNOT.

- **Remarque**

Les programmes de compactage les plus performants combinent la puissance des algorithmes de type LZW et l'efficacité des algorithmes statistiques pour compresser les motifs redondants. La combinaison est simple :

- ✓ Aux algorithmes de type dictionnaire de détecter les redondances dans les fichiers et de constituer les dictionnaires de motifs redondants.
- ✓ Aux algorithmes statistiques de trouver les codages les plus concis (précis) pour les termes contenus dans les dictionnaires.

- **Utilité et caractéristiques de compression :**

  Cette méthode est peu efficace pour les images mais donne de bons résultats pour les textes et les données informatiques en général (plus de 50%).

- **Algorithme de décompression :**

  Lors de la lecture de l'information encodée, les " mots " codés sont remplacés dans le fichier par leur correspondance lue dans le dictionnaire et le fichier d'origine est ainsi reconstitué.

V.4.2.La compression avec perte (lossy) [13]

La compression avec pertes ne s'applique que sur des données perceptuelles (audio, image, vidéo), et s'appuie sur les caractéristiques du système visuel ou du système auditif humain pour ses stratégies de réduction de l'information. Les techniques sont donc spécifiques à chaque média. Ces dernières ne sont pas utilisées seules mais sont combinées pour fournir un système de compression performant et avoir un meilleur taux de compression possible.

V.4.2.1. Méthodes utilisées pour la compression avec pertes :

Voici trois méthodes utilisées pour la compression avec pertes :

- **La transformée en cosinus discrète (DCT)** :

La transformée en cosinus discrète est une fonction mathématique complexe dont il existe plusieurs variantes. Elle est utilisée pour la compression destructrice des données, notamment pour les sons aux formats Vorbis, WMA et MP3 et pour les images et les vidéos suivant les normes JPEG et MPEG.

Lors de l'opération de compression des données dans ces différents formats, l'algorithme transforme les pixels de l'image ou les échantillons de la séquence audio en fréquences. Les

fréquences qui ne correspondent pas à des données pertinentes pour l'œil ou l'oreille humaine sont éliminées par l'algorithme.

- **La compression par ondelettes :**

La compression par ondelettes, aussi appelée DWT (Discrete Wavelet Transform) est une méthode basée sur la théorie mathématique de l'analyse du signal : les ondelettes sont un ensemble de signaux élémentaires à partir desquels on peut reconstruire un signal complexe. La compression par ondelettes consistera donc à décomposer l'image perçue comme un signal en un ensemble d'images de plus petite résolution. Ce procédé, qui repose sur la différence entre zones de contrastes forts et zones de contrastes faible.

- **La compression fractale** :

La compression fractale est un procédé qui s'applique uniquement aux images. Il repose sur la géométrie fractale de Benoît Mandelbrot. Le format des images compressées par ce procédé n'est à l'heure actuelle pas standardisé et n'est donc pas reconnu par les navigateurs.

VI. Les logiciels de la compression/décompression:

- ✓ **Au niveau de compression de fichiers sans perte :**

Le logiciel le plus connu est le **WinZip**. Ce logiciel payant traite de nombreux format de compression/archivage. Les plus connus et les plus fiables sont **ZIP Central**, **FilZip** et **7-Zip** (qui offre en format 7Z de meilleurs résultats de compression).

- ✓ **Au niveau des compressions avec pertes :**

L'existence de plusieurs formats de compression rend particulièrement utiles les logiciels capables de traiter un maximum de codecs, de manière à pourvoir décompresser et compresser à volonté les données dans le format souhaité.

Pour l'audio : une variété de logiciels permettent de compresser dans l'un ou l'autre format spécifique. Mentionnons CDex qui permet de compresser des fichiers en WAV, MP3 ou Ogg Vorbis, et Ogg Drop qui gère spécifiquement ce dernier format. Le logiciel Audacity, entièrement gratuit, permet de mixer différents sons, mais aussi de les compresser dans les différents formats existants une fois le plug-in adéquat installé.

Pour les images: les deux logiciels de traitement les plus connus, Adobe Photoshop et Paint Shop Pro, sont les outils les plus pratiques afin de compresser-décompresser une grande variété de formats, des plus classiques (JPEG, GIF) aux plus rares (JPEG 2000, Pixar).

Pour les vidéos : On peut mentionner le logiciel Ripp-it After Me, gratuit, en français, capable de compresser-décompresser les formats audio et vidéo les plus divers (Mp3, Ogg

Vorbis, AC3 et Real Audio pour l'audio, DivX, Xvid, MP1, MP2, RV10 et d'autres pour la vidéo) et disposant d'un encodeur MPEG4.

VII. Conclusion :

Dans ce chapitre nous avons présenté les différentes approches de la compression de données, ainsi que les différents algorithmes correspondants.

Chapitre II :

La méthode proposée.

I. Introduction:

Les méthodes par transformation sont les techniques les plus utilisées. Elles permettent d'obtenir des taux de compression élevés tout en concervant une bonne qualité du fichier originale (texte, son et image).

Dans ce chapitre nous allons définir les nouvelles méthodes de transformation dans la compression de données.

II. Les transformées dans la compression de données

II.1. Les transformées dans la compression de données avec perte

II.1.1.La transformation en fractale : [14] [15]

La compression fractale est une méthode de compression d'images encore peu utilisée aujourd'hui. Elle repose sur la détection de la récurrence des motifs, et tend à éliminer la redondance d'informations dans l'image.

Il existe plusieurs méthodes (subdivision de triangles, Delaunay etc.) mais la compression par la méthode Jacquin est la plus connue.

Illustration par la méthode de Jacquin :

- La compression fractale consiste tout d'abord à réaliser deux segmentations (appelées aussi pavages, ou partitionnements) sur une image : une segmentation de figures Sources et une segmentation de figures Destinations.
- Il s'agit alors de trouver pour chaque figure Source, quel est le meilleur couple (figure source, figure destination) minimisant une erreur. Cette erreur est généralement calculée en soustrayant les deux figures. Pour réaliser l'opération de soustraction, il est nécessaire d'opérer une transformation de la figure source aux dimensions (et à la géométrie) de la figure destination.

De plus, des règles comme la rotation et les retournements sont possibles.

- Une fois que tous les couples ont été trouvés, le fichier de sortie contient alors les différents couples, ainsi que les différentes transformations effectuées (rotation, réduction de la moyenne etc.).
- Lors de la décompression, l'image est recréée à partir de ces transformations. La convergence est alors garantie par le fait que d'une part il y a une minimisation d'erreur (différence) et une modification des pixels, et d'autre part, que les figures sources sont plus grandes que les figures destinations. La compression fractale utilise la même propriété pour reconstruire l'image.

Partitionnement :

Le partitionnement est l'opération qui consiste à segmenter une image en régions. Dans la compression par la méthode Jacquin, nous avons besoin de 2 partitionnements : Source et Destination. La méthode Jacquin utilise par exemple des figures carrées, mais d'autres formes sont possibles (nids d'abeilles, triangles etc.).

- Un point essentiel dans les partitionnements Source et Destination est que le pavage destination doit être plus petit que le pavage source. En effet, dans le cas contraire, nous serions amenés à faire un agrandissement (et non une réduction) lors de la transposition des figures sources vers les figures destinations. Une fractale possède un motif se répétant à l'infini, en se rétrécissant. Aussi, nous perdons cette propriété si le partitionnement destination est plus grand que le partitionnement source, l'image ne pourra alors pas converger.
- Le partitionnement par la méthode Jacquin est un partitionnement statique. L'utilisation d'un partitionnement adaptatif (qui dépend de l'image à traiter) améliore considérablement le facteur de compression.

Décompression :

La décompression consiste en la lecture du fichier contenant les correspondances figure source-figure destination. Il suffit ensuite d'appliquer les transformations plusieurs fois. Ce procédé de reconstruction itéré, aussi connu sous le nom de système de fonctions itérées, garantit une convergence, relative, vers l'image de départ. La qualité du résultat dépend fortement de la taille des figures de segmentation, plus les figures seront nombreuses, et plus l'image résultante sera de qualité.

II.1.2.La transformée en DCT (Discrete Cosinus Transform): [15]

La transformée en cosinus discrète est une fonction mathématique complexe dont il existe plusieurs variantes. Elle est utilisée pour la compression destructrice des données, notamment pour les sons aux formats Vorbis, WMA et MP3 et pour les images et les vidéos suivant les normes JPEG et MPEG.

La DCT est effectuée sur une matrice carrée N*N de valeurs de pixels et donne une matrice carrée N*N de coefficients de fréquence. Le temps de calcul requis pour chaque élément dans la DCT dépend de la taille de la matrice.

Vu la difficulté d'appliquer la DCT sur la matrice entière, celle-ci est décomposée en blocs.

A la sortie de la matrice de la DCT, la valeur de la position(0,0) est appelée le coefficient continu, cette valeur représente une moyenne de la grandeur d'ensemble de la matrice d'entrée, ce coefficient est plus grand d'un ordre de grandeur à toute valeur dans la matrice de la DCT.

- ✓ La transformée DCT s'exprime mathématiquement par :

$$\mathrm{DCT}(i,j) = \frac{2}{N} C(i)C(j) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} \mathrm{pixel}(x,y) \cos\left[\frac{(2x+1)i\pi}{2N}\right] \cos\left[\frac{(2y+1)j\pi}{2N}\right]$$

- ✓ La constante C vaut:

$$C(x) = \begin{cases} \frac{1}{\sqrt{2}} & \text{pour } x = 0 \\ 1 & \text{pour } x > 0 \end{cases}$$

Le codage par transformation DCT n'effectue aucune fonction de compression mais permet de séparer l'information la plus importante de l'image de celle moins importante.

II.2. Les transformées dans la compression de données sans perte

II.2.1.La transformée de Burrows-Wheeler Transform (BWT) [6]

La transformée de Burrows-Wheeler, couramment appelée BWT (pour Burrows-Wheeler Transform) est une technique de compression de données. Elle fut inventée par Michael Burrows et David Wheeler. Cette technique fut rendue publique en 1994, suite à de précédents travaux de Wheeler en 1983. Il ne s'agit pas à proprement parler d'un algorithme de compression, car aucune réduction de taille n'est effectuée, mais bien d'une méthode de réorganisation des données. Ce qui est intéressant dans cet algorithme par rapport aux autres algorithmes de tri, c'est qu'il est réversible. C'est-à-dire que l'on peut revenir au fichier original à partir du fichier trié.

II.2.1.1.Fonctionnement:

La transformée de Burrows-Wheeler ne compresse pas les données, elle se contente de les réorganiser de manière à obtenir un meilleur taux de compression.

Tout d'abord, la chaîne de caractères à coder doit être copiée dans un tableau carré en décalant la chaîne d'un caractère vers la droite à chaque nouvelle ligne. Ces lignes sont ensuite classées par ordre alphabétique. Nous savons que, grâce au décalage, chaque dernière lettre de chaque ligne précède la première lettre de la même ligne, sauf pour la ligne originale dont on notera

la position. De plus, comme les lignes sont rangées par ordre alphabétique, on peut retrouver la première colonne du tableau grâce à la dernière colonne.

Prenons un exemple : supposons que la chaîne à coder soit « TEXTUEL ». On réalise tout d'abord le tableau. La première lettre est marquée ici par une flèche pour améliorer la lecture du tableau.

```
Chaîne

→T E X T U E L
 L T E X T U E
 E L T E X T U
 U E L T E X T
 T U E L T E X
 X T U E L T E
 E X T U E L T
```

Puis l'on classe ces chaînes par ordre alphabétique :

```
Chaîne          Position

E L T E X T U   1
E X T U E L T   2
L T E X T U E   3
T E X T U E L   4 ← position du texte original
T U E L T E X   5
U E L T E X T   6
X T U E L T E   7
```

Le texte codé est alors constitué de la dernière colonne précédée de la position du texte original, soit : « 4UTELXTE ». La position du texte original sert au décodage.

Cette transformation n'apporte aucun gain de compression immédiat, au contraire, car il est nécessaire de transmettre des informations supplémentaires pour le décodage. Cependant, pour un texte relativement long en langage naturel, qui contient plusieurs fois les mêmes mots, le texte codé comportera de nombreuses répétitions de caractères. En effet, cela revient à trier les caractères par ordre alphabétique puis à chaque fois écrire le caractère qui précédait dans le texte original (Vu qu'elle est construite par rotation, la dernière colonne est composée des caractères précédant). Un exemple flagrant est la transformation de « TEXTUELTEXTUEL » qui donne : « 7UUTTEELLXXTTEE ».

Ainsi, Burrows et Wheeler recommandent d'utiliser ensuite un algorithme de type MTF qui, de par les répétitions de caractères, génèrera beaucoup de 0. Ceci assure avec un algorithme de type codage de Huffman un quotient de compression élevé.

Le décodage consiste à reconstruire le tableau complet à partir de sa dernière colonne (texte codé « UTELXTE ») à partir de laquelle on reconstruit la colonne « suivante », c'est-à-dire, par rotation, la première, dont on sait qu'elle est dans l'ordre alphabétique (« EELTTUX »). On colle alors la dernière colonne juste avant cette première colonne, puis on classe dans l'ordre alphabétique les paires obtenues pour construire les deux premières colonnes. On répète ensuite cette opération jusqu'à constituer le tableau complet dans lequel on retrouve le texte original par son numéro de ligne :

Initiation	**Tri**	**Collage**	**Tri**	**Collage**	**Tri**
U	E	UE	EL	UEL	ELT
T	E	TE	EX	TEX	EXT
E	L	EL	LT	ELT	LTE
L	T	LT	TE	LTE	TEX
X	T	XT	TU	XTU	TUE
T	U	TU	UE	TUE	UEL
E	X	EX	XT	EXT	XTU

Collage	**Tri**	**Collage**	**Tri**	**Collage**	**Tri**
UELT	ELTE	UELTE	ELTEX	UELTEX	ELTEXT
TEXT	EXTU	TEXTU	EXTUE	TEXTUE	EXTUEL
ELTE	LTEX	ELTEX	LTEXT	ELTEXT	LTEXTU
LTEX	TEXT	LTEXT	TEXTU	LTEXTU	TEXTUE
XTUE	TUEL	XTUEL	TUELT	XTUELT	TUELTE
TUEL	UELT	TUELT	UELTE	TUELTE	UELTEX
EXTU	XTUE	EXTUE	XTUEL	EXTUEL	XTUELT

Collage	**Tri**	**Sélection**
UELTEXT	ELTEXTU	1
TEXTUEL	EXTUELT	2
ELTEXTU	LTEXTUE	3

```
LTEXTUE  TEXTUEL ← 4
XTUELTE  TUELTEX   5
TUELTEX  UELTEXT   6
EXTUELT  XTUELTE   7
```

On retrouve bien le texte original à la ligne dont le numéro avait été transmis avec le texte codé.

Vu que la chaine transformée est constituée des caractères qui précèdent dans l'ordre alphabétique. Une autre manière d'appréhender ce décodage consiste à trier par ordre alphabétique et prendre à chaque fois le suivant dans la chaine transformée.

1. On tri les caractères par ordre alphabétique, on obtient : « EELTTUX ».
2. Prendre celui qui correspond à l'indice 4 soit un T.
3. Où était ce T dans le texte transformé? à l'indice 2. C'est le premier T et non celui à l'indice 3 car c'est le premier de la chaine triée.
4. Prendre celui qui correspond à l'indice 2 soit un E.
5. Où était ce E dans le texte transformé? à l'indice 7. Ce n'est pas celui en position 3 car c'est le 2ème E dans la chaine triée c'est donc le deuxième de la chaine transformée.
6. Prendre celui qui correspond à l'indice 7 soit un X.etc.

III. La méthode proposée

III.1.La fonction de codage d'ordre

Avant de présenter la fonction d'ordre, il est important de donner quelques caractéristiques de la relation « Ordre » dans un vecteur de N éléments.

- **Propriétés 1** : Pour N symboles distincts, ils existent N ! (permutation) ordres distincts.

  Exemple1 : soit l'ensemble des symboles E= {A, B} alors on a N=2 éléments par conséquent le nombre d'ordre qu'on peut avoir est 2 ! = 2 ordres.

Permutations	Ordre
A B	Ord1
B A	Ord2

Exemple2 : soit l'ensemble des symboles E= {A, B, C} alors on a N=3 éléments par conséquent le nombre d'ordre qu'on peut avoir est 3 !=6 ordres.

Permutations	Ordre
A B C	Ord1
A C B	Ord2
B A C	Ord3
B C A	Ord4
C A B	Ord5
C B A	Ord6

- **Propriétés 2**:Chaque permutation possède son propre ordre (l'ordre est unique)

La codification de l'ordre d'une permutation donnée passe par les étapes suivantes :

Permutation ⟶ Code intermédiaire ⟶ Code Ordre

Tel que le code intermédiaire est étroitement lié à la relation d'ordre.

Exemple :

Permutation	Code intermédiaire			Ordre
	C3	C2	C1	
A B C	0	0	0	Ord1=0x2!+0x1!+0x0!=0
A C B	0	1	0	Ord2=0x2!+1x1!+0x0!=1
B A C	1	0	0	Ord3=1x2!+0x1!+0x0!=2
B C A	1	1	0	Ord4=1x2!+1x1!+0x0!=3
C A B	2	0	0	Ord5=2x2!+0x1!+0x0!=4
C B A	2	1	0	Ord6=2x2!+1x1!+0x0!=5

Tableau II.1 : Codification de l'ordre.

Chaque digit C_i du code intermédiaire correspond à un symbole S_i de la permutation de façon à indiquer le nombre de symboles à droite de ce dernier qui sont inférieur à S_i.

Dans l'exemple ci- dessus, si on prend la permutation « B C A » alors le calcul de C3, C2 et C1 se fait comme suit :
C3=1 : car il existe 1 seul symbole « A » qui est inférieur à B.

C2=1 : car il existe 1 seul symbole « A » qui est inférieur à C.

C1=0 : car il n'existe pas de symbole après le « A ».

Remarque : le digit C1 est toujours à Zéro car il correspond au dernier symbole du vecteur et il n'existe aucuns symboles à sa droite. Donc on peut garder seulement 2 digits pour le code intermédiaire pour un vecteur de 3 symboles.

III.1.1. Généralisation

Soit $V = \{S_1, S_2, \ldots\ldots S_t\}$ un vecteur de symbole de taille T, alors on a (T !) Permutations possibles qui donnent lieu à (T !) Code d'ordre.

Pour chaque permutation (P_i), il existe un code intermédiaire $C= C_{T-1}C_{T-2}\ldots\ldots C_1$

Tel que C_i représente le nombre de symboles qui sont inférieur au symbole S_i et qui se trouvent à droite de ce dernier.

On note la fonction d'ordre pour une permutation P_i donnée : Ordre(P_i)

Alors on a :

$Ordre(P_i)=C_{T-1} \times (T-1) !+C_{T-2} \times (T-2) !+\ldots\ldots\ldots\ldots C_1 \times 1 !$

III.1.2.Propriétés de la fonction de codage d'ordre

- Unicité : chaque permutation possède un seul et unique code d'ordre.
- Le code Ordre peut être considéré comme l'entropie d'ordre qui permet de mesurer le degré d'ordre d'un vecteur donné.

Pour un vecteur totalement ordonné $H_{min}=0$ et la valeur de l'entropie est maximale $H_{max}= (N !- 1)$ pour un vecteur totalement désordonné.

- La valeur de l'ordre est un entier varie de 0 à (N !-1) selon le degré d'ordre du vecteur.

III.2.Fonction de décodage d'ordre:

Le résultat de la compression est un attribut **Ord** qui représente le code ordre des symboles. Pour retrouver l'ordre des symboles dans le message on doit passer par le processus inverse de la fonction de codage d'ordre, c'est-à-dire à partir du code Ord on retrouve le code intermédiaire par des divisions successives sur (N-1)!, (N-2) !,….1.

Soit l'ensemble des symboles E= {A, B, C} de l'exemple précédant.

Prenant la valeur d'Ord=3 et essayant de retrouver l'ordre des symboles.

D'abord on cherche la valeur du code intermédiaire $C=C_2\ C_1$

On a $Ord= C_2 \times 2! + C_1 \times 1! =3$

$C_2 =3/2!=1$, $C_1 = (3-(1x2!))/1!=1$ ➡ C =11

Il reste le passage du code intermédiaire à l'ordre initial des symboles qui se fera selon l'algorithme suivant :

-Initialiser le message à une chaine vide, M= « ».

-Trier les symboles par ordre croissant dans un vecteur.

A	B	C
0	1	2

-C_2= 1 cela signifie que le premier symbole est celui de la case 1 du vecteur et qui est « B ».

-Retirer le symbole B du vecteur et le concaténer avec le message M= « B »

A	C
0	1

-C_1= 1 cela signifie que le premier symbole est celui de la case 1 du vecteur et qui est « C »

-Retirer le symbole C du vecteur et le concaténer avec le message M= « BC ».

-Il ne reste qu'un seul symbole qui est « A » qu'on concatène avec M ce qui donne le message final M= « BCA ».

III.3.Avantage de la méthode

- ***La simplicité :*** ne nécessite pas de modèles mathématiques compliqués.
- Changer la structure du contenu du fichier permet d'augmenter le niveau de redondance et par conséquent diminuer l'entropie de ce dernier pour le soumettre en fin à un algorithme de compression entropique tels que Huffman, RLE, … etc.

IV. Conclusion

Dans ce chapitre nous avons présenté les transformées DCT, fractale et BWT ainsi que la fonction d'ordre qui est une nouvelle transformée pour la compression de données qui se base sur la codification de l'ordre d'une permutation donnée.

I. Introduction :

Dans ce chapitre nous allons expliquer en premier lieu l'analyse de la nouvelle transformée pour la compression de données qui est basée sur la fonction d'ordre, ensuite présenter l'architecture générale de notre application ainsi que ses différents modules et sous modules en expliquant le processus de codage et de décodage. Ce système doit fournir en plus des fonctionnalités de compression/décompression d'autres fonctions telles que le calcul de temps, le taux de compression et éventuellement les tailles des fichiers.

Enfin nous allons présenter les différents algorithmes nécessaires à la mise en œuvre de notre l'application.

II. Objectif :

Dans notre application nous voulons implémenter une nouvelle approche de compression fondée sur une transformée qui est basée sur la fonction d'ordre et évaluer ses performances (taux, temps…).

III. Analyse de la méthode proposée:

La méthode proposée dans ce projet est une nouvelle transformée dans la compression de données sans perte, qui est basée sur la fonction d'ordre où lors de la décompression on aura le même fichier d'origine.

L'intérêt de cette méthode est de créer des redondances avant de compresser le fichier, pour faciliter l'application de l'algorithme de compression.

III.1.Fonctionnement de la transformée :

Ce processus consiste à lire le fichier source mot par mot de taille variable (2,3,4..8 caractères) puis calculer son code d'ordre et trier chaque mot par ordre alphabétique ensuite sauvegarder ce résultat dans un fichier de sortie (fichier transformé).

Exemple :

Soit M, un extrait d'un fichier avec M= « ….la compression de données », à transformer.

1. Lire le fichier mot par mot de taille soit égal à 8 caractères.

   Mot1= la_compr

   Mot2=ession_d

   Mot3=e_donnee

   Mot4=s_ _ _ _ _ _ _

Le caractère vide est représenté par « _ »

2. Calculer le code ordre pour chaque mot :

Ord(Mot1)=3x7 !+1x6 !+0x5 !+0x4 !+1x3 !+0x2 !+0x1 !=15846

Ord(Mot2)= 2x7 !+5x6 !+5x5 !+2x4 !+3x3 !+2x2 !+0x1 !=14350

Ord(Mot 3)= 2x7 !+0x6 !+0x5 !+4x4 !+2x3 !+2x2 !+0x1 != 10192

Ord(Mot 4)= 2x7 !+0x6 !+2x5 !+4x4 !+2x3 !+2x2 !+0x1 !=35280

3. Trier les mots par ordre alphabétique accompagnés par leurs codes d'ordre :

aclmopr15846_deinoss14350_deeenno10192_ _ _ _ _ _ _ s35280

4. Sauvegarder le résultat dans le fichier de sortie (fichier transformé).

III.2.Fonctionnement de la transformée inverse :

Le processus de la transformée inverse consiste à lire les données du fichier transformé après avoir connu la taille du fichier source et la taille du mot, ensuite appliquer la fonction inverse pour chaque mot en suivant les étapes suivantes :

D'abord on cherche la valeur du code intermédiaire C=C7C6C5C4C3C_2C_1 du Mot1

On a Ord= C_7 x 7! + C_6 x 6! + C_5 x 5! + C_4 x 4! + C_3 x 3! + C_2 x 2! + C_1 x 1! =15846

C_7=15846/7! =3

C_6=15846-(3x7!)/6! =1

C_5=15846-(3x7! +1x6!)/5! =0

C_4=15846-(3x7! +1x6! +0x5!)/4! =0

C_3=15846-(3x7! +1x6! +0x5! 0x4!)/3! =1

C_2 = 15846-(3x7! +1x6! +0x5! 0x4! +1x3!)/2! =0

C_1=15846-(3x7 !+1x6 !+0x5 !0x4 !+1x3 !+0x1 !)/1 !=0

A partir de C=31001000 on reconstitue l'ordre :

_	a	c	l	m	o	p	r
0	1	2	3	4	5	6	7

C_7=3 le caractère se trouve à la position 3 du vecteur trié.

Le caractère est « l », on le retire du vecteur, la valeur actuelle du Mot1 est « l »

_	a	c	m	o	p	r
0	1	2	3	4	5	6

C_6=1 le caractère se trouve à la position 1 du vecteur trié.

Le caractère est « a », on le retire du vecteur, la valeur actuelle du Mot1 est « la »

_	c	m	o	p	r
0	1	2	3	4	5

C_5=0 le caractère se trouve à la position 0 du vecteur trié.

Le caractère est «_», on le retire du vecteur, la valeur actuelle du Mot1 est « la _ »

c	m	o	p	r
0	1	2	3	4

C_4= 0 le caractère se trouve à la position 0 du vecteur trié.

Le caractère est « c », on le retire du vecteur, la valeur actuelle du Mot1 est « la _c »

m	o	p	r
0	1	2	3

C_3=1 le caractère se trouve à la position 1 du vecteur trié.

Le caractère est « o », on le retire du vecteur, la valeur actuelle du Mot1 est « la_co »

m	p	r
0	1	2

C_2=0 le caractère se trouve à la position 0 du vecteur trié.

Le caractère est « m », on le retire du vecteur, la valeur actuelle du Mot1 est « la _com »

p	r
0	1

C_1=0 le caractère se trouve à la position 0 du vecteur trié.

Le caractère est « p », on le retire du vecteur, la valeur actuelle du Mot1 est « la_comp»

r

0

Et le dernier caractère implicite est « r » on l'ajoute à Mot1.

Le résultat final est Mot1= « la_compr » qui correspond parfaitement au Mot1 d'origine.

On suit la même procédure pour le reste des mots.

IV. Conception de la méthode proposée:

La conception consiste à élaborer à partir de la spécification du problème une solution informatique, son principe est de décomposer le problème en sous-problème.

Pour la conception de notre application on a adopté une architecture flexible, basé sur une architecture modulaire.

IV.1. Définition :

- **Les modules** : sont des entités indépendantes intégrées dans une architecture pour produire une application.
- **Système** : l'ensemble des modules utilisés, ainsi que les relations qu'ils entretiennent entre eux.

IV.2. Les méthodes de conception :

Pour construire un système, il faut également une méthode de conception à suivre, pour cela on distingue deux familles de méthodes de conception :

IV.2.1. Les méthodes descendantes (top-down):

L'approche descendante commence par décomposer le problème initial en sous problèmes puis chaque sous-problème en de nouveaux sous-problèmes et ainsi de suite jusqu'aux problèmes que nous pouvons résoudre par des opérations primitives (ou des fonctions simples).

IV.2.2. Les méthodes ascendantes (Bottom-up):

L'approche ascendante construit des opérations primitives que nous assemblons pour obtenir des opérations plus complexes et ainsi de suite jusqu'à une opération globale qui résout le problème initial.

- on utilise les méthodes *descendante* dans le domaine du développement, parce qu'on maîtrise en principe déjà les concepts dans le cadre d'une théorie qu'on met en œuvre, et au

contraire les méthodes ascendantes dans le cadre de la recherche, où on cherche à les faire émerger de la pratique.

IV.3.Conception générale du système:

L'architecture générale de ce nouveau système compression/décompression est composée d'un ensemble de fonctions (obtenues par la méthode descendante) qui se complète pour réaliser l'objectif et la tâche globale visée par le système qui va être présenté dans les paragraphes qui suivent.

IV.3.1.Module principal de la méthode (le noyau) :

Afin de répondre à notre objectif, on va appliquer les concepts vus au préalable pour définir les différents modules constituant le noyau.

Donc on peut décomposer le noyau en cinq sous-modules :

1. Module de transformée ;
2. Module de compression ;
3. Module de décompression ;
4. Module de transformée inverse;
5. Module de calcul et affichage des performances.

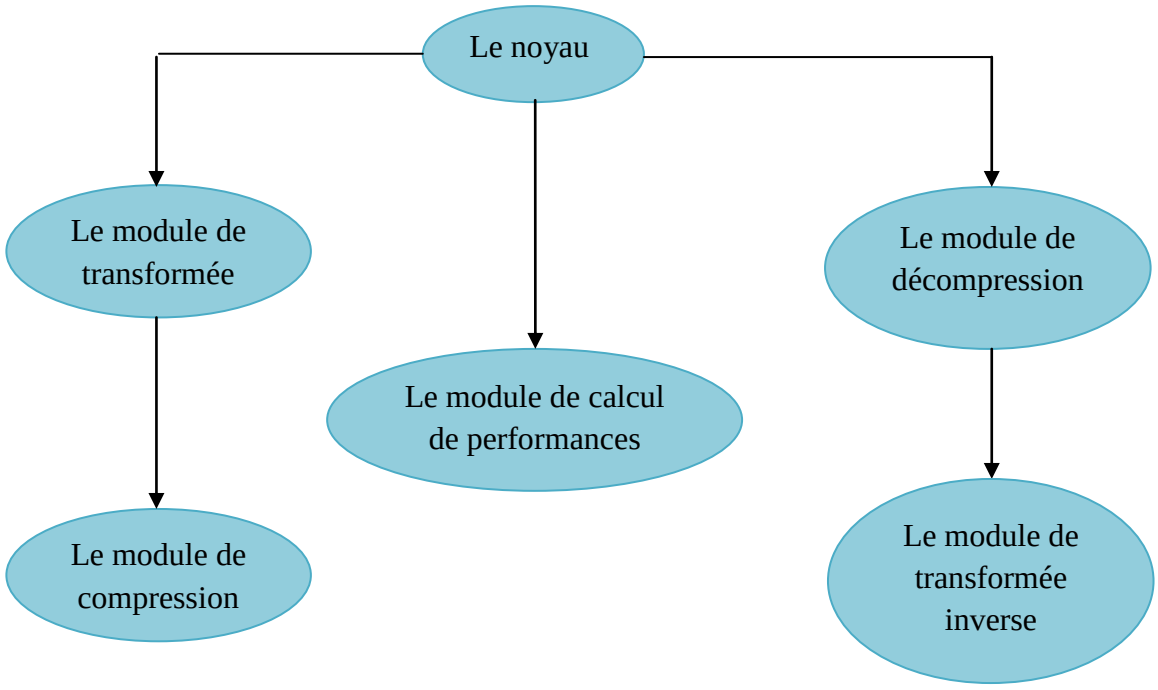

Figure III.1: Décomposition du module principal de l'application.

IV.3.1.1.Module de transformée:

Après avoir spécifié le fichier à compresser, le rôle principal de la fonction de transformée est de lire le fichier source mot par mot de tailles variables (2, 3,4….8 caractères) et de calculer le code ordre pour chaque mot, les trier et les ordonner par ordre alphabétique, enfin, sauvegarder ce résultat dans le fichier transformé (fichier de sortie).

Cette fonction transforme chaque mot de flot de données en un enregistrement de deux attributs M_i,Ord_i tel que M_i représente le mot trié et Ord_i son code ordre.

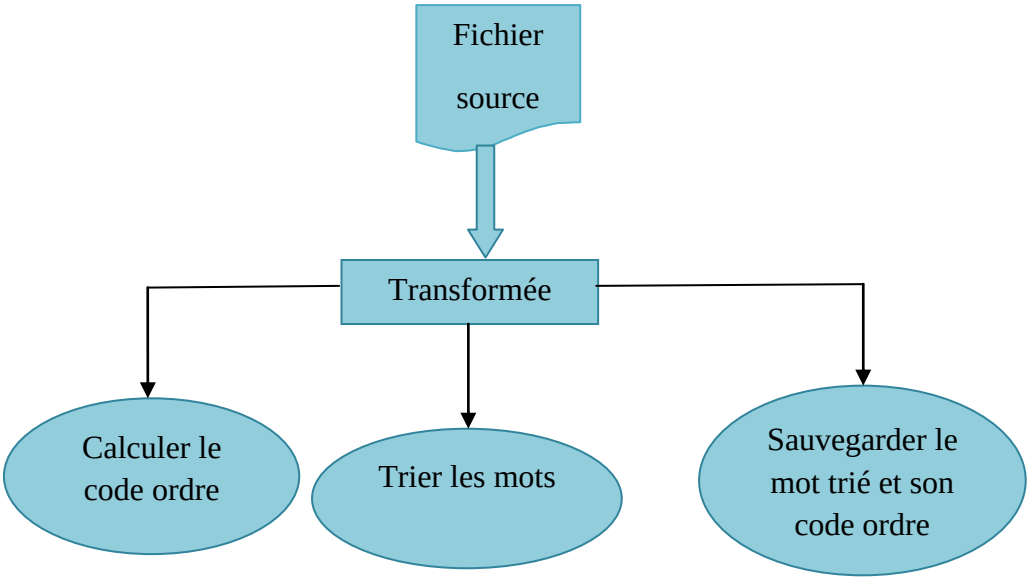

Figure III.2: Schéma général du module de transformée.

IV.3.1.2.Module de compression :

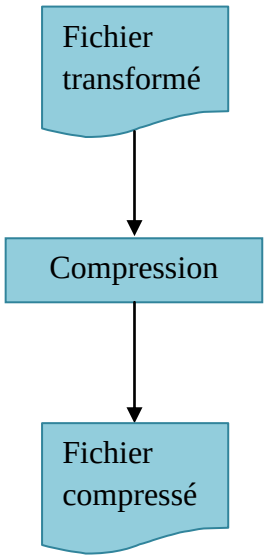

Figure III.3: Schéma général du module de compression.

IV.3.1.3.Module de décompression :

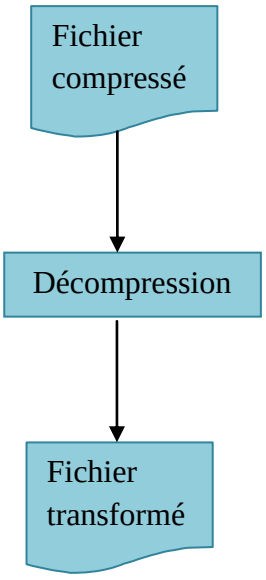

Figure III.4: Schéma général du module de décompression.

IV.3.1.4.Module de transformée inverse :

La transformée inverse consiste à restituer les données initiales du fichier transformé donc elle commence par lire les paramètres nécessaires au décodage des données, comme la taille du fichier source, la taille du mot qui sont enregistrés dans l'en-tête du fichier transformé.

Enfin elle fait le travail inverse c'est-à-dire, retrouver les mots correspondant aux codes ordre et retourner au fichier source.

Le sous module de la transformée inverse est représenté dans la figure suivante :

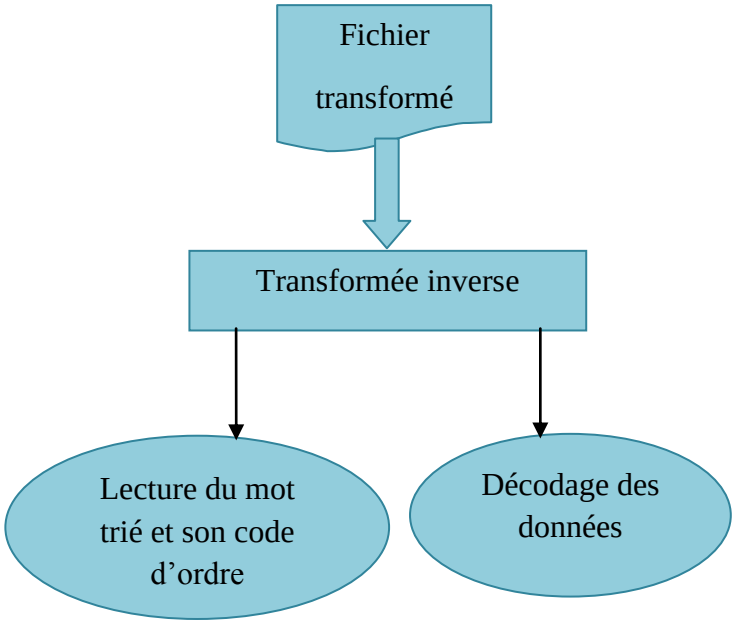

Figure III.5: Schéma général du module de transformée inverse.

IV.3.1.5.Module de calcul et affichage des performances:

- ✓ **Module de calcul et affichage des performances pour la compression:**

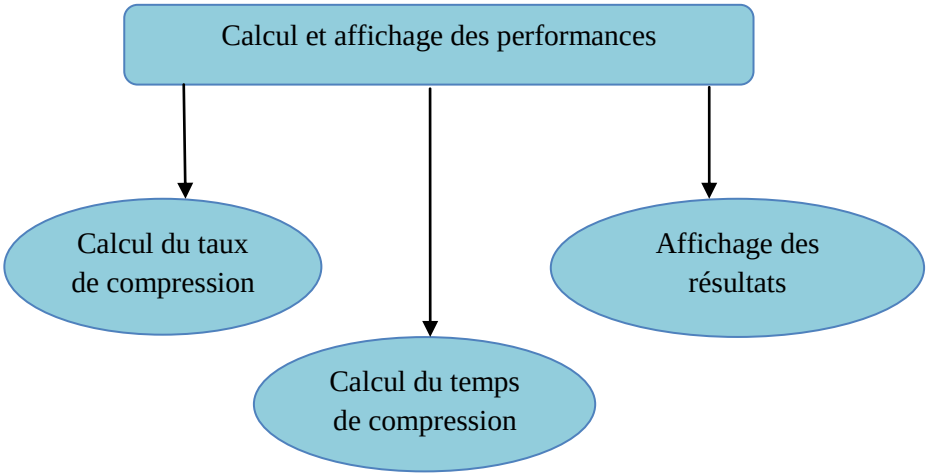

Figure III.6: Schéma général du module de calcul et affichage des performances pour la compression.

✓ **Module de calcul et affichage des performances pour la décompression:**

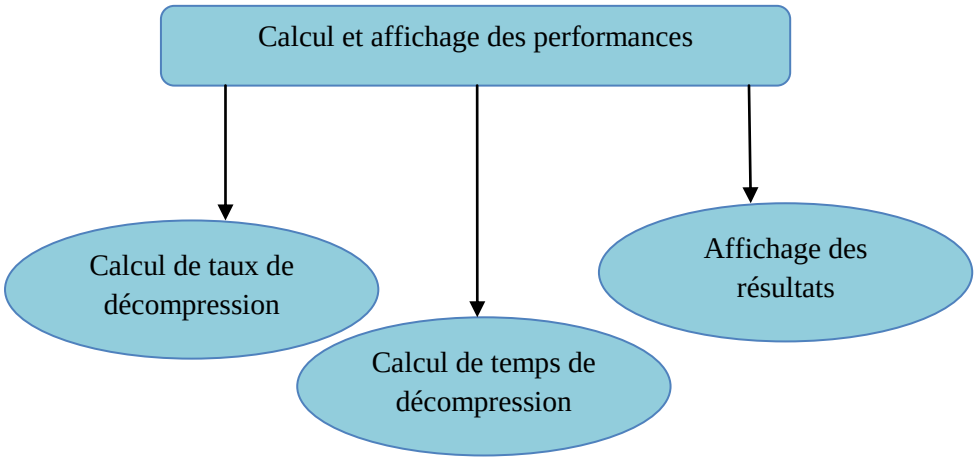

Figure III.7: Schéma général du module de calcul et affichage des performances pour la décompression.

Grace au calcul et affichage des performances de l'algorithme, comme le temps et le taux de compression, que nous pouvons à la fin tester et évaluer cette méthode de compression.

➢ **Le temps de compression**

Pour calculer le temps de compression, ce processus utilise l'horloge système au début de la compression pour lui donner le temps de début de la compression. A la fin de la compression il sollicite une autre fois cette horloge pour lui donner le temps de fin de compression. Et pour obtenir le temps exact de compression il fait la soustraction du temps fin de la compression par le temps début de la compression, comme suit :

Temps de compression = temps de fin de compression – temps de début de compression

➢ **Le taux de compression**

Pour calculer le taux de compression, le processus doit récupérer avant la compression la taille du fichier transformé (source) et le fichier compressé (cible), puis il calcule le taux comme l'indique l'équation suivante :

$$\textbf{Taux de compression} = 1 - \frac{(\boldsymbol{taille\ compressée}) * \mathbf{100}}{\boldsymbol{taille\ originale}}$$

IV.3.2.Organigramme général de la méthode :

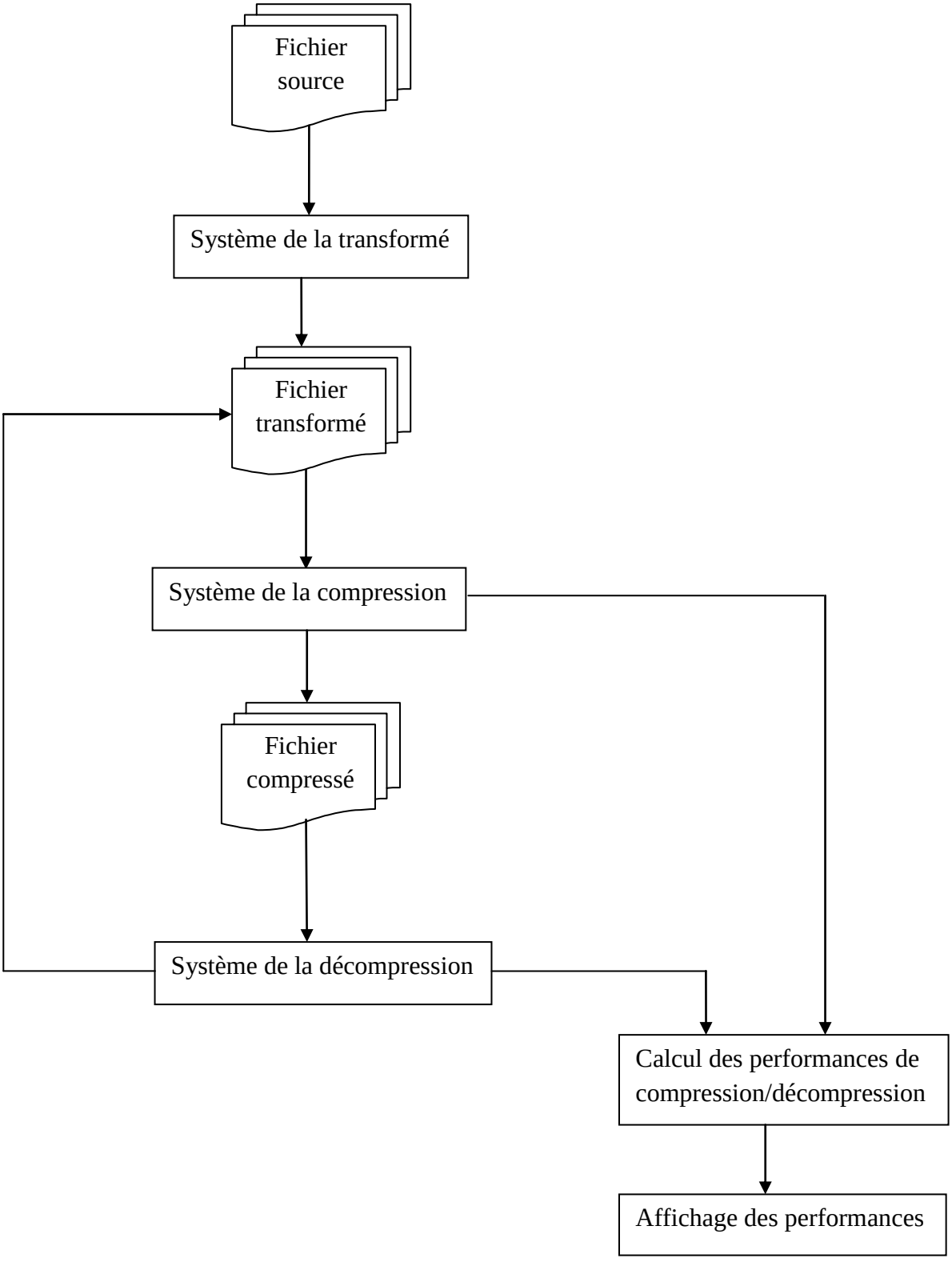

Figure III.8 : Schéma général de la méthode.

V. Les algorithmes :

On présente dans cette partie, les différents algorithmes des modules essentiels qui forment le noyau de l'application, tel que la transformée et la transformée inverse.

V.1. Algorithme de la transformée:

DEBUT

Ouvrir le fichier à compresser ;

Créer un nouveau fichier pour sauvegarder les résultats de la transformée ;

Ecrire dans l'en-tête du fichier transformé les paramètres de la transformée (taille du mot);

Faire

Lire les mots de taille **t** ;

Calculer le code ordre de chaque mot à l'aide de la fonction Fordre() ;

Trier les mots à l'aide da la fonction trier () ;

Sauvegarder les mots triés avec leurs codes ordre dans le fichier crée ;

Tant que (il y'a de caractères) ;

Fermer les deux fichiers source et transformé;

FIN

V.1.1.Organigramme de la transformée :

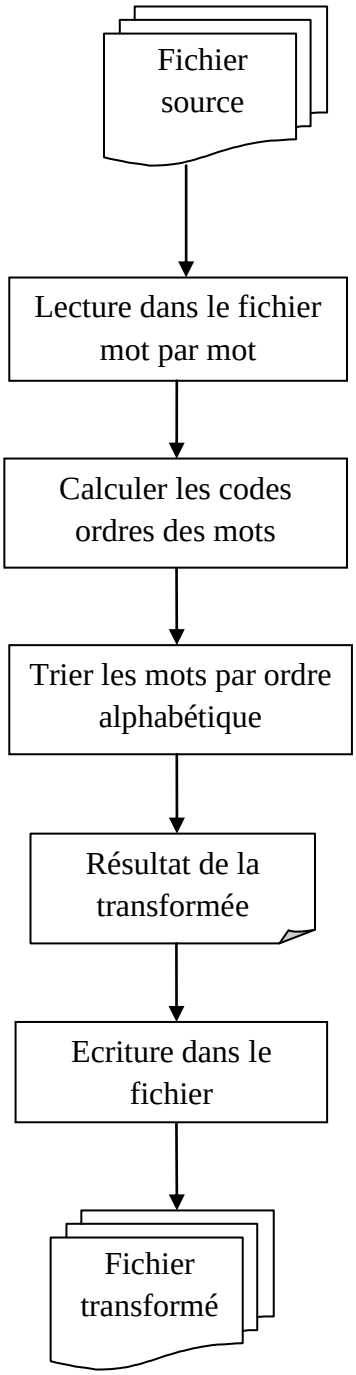

Figure III.9 : Illustration du module de transformée.

V.2.Algorithme de la transformée inverse :

DEBUT

Ouvrir le fichier transformé ;

Créer un nouveau fichier pour sauvegarder le résultat du processus inverse de la transformée ;

Lire l'en-tête du fichier transformé (paramètres de la transformée) ;

Faire

Lire les mots triés de taille t ;

Lire leurs codes ordres ;

Restituer l'ordre initial des mots à l'aide de la fonction Finverse() ;

Tant que (on n'a pas atteint le nombre de caractères spécifiés dans l'en-tête) ;

Fermer les deux fichiers ouverts précédemment ;

FIN

V.2.1.Organigramme de la transformée inverse :

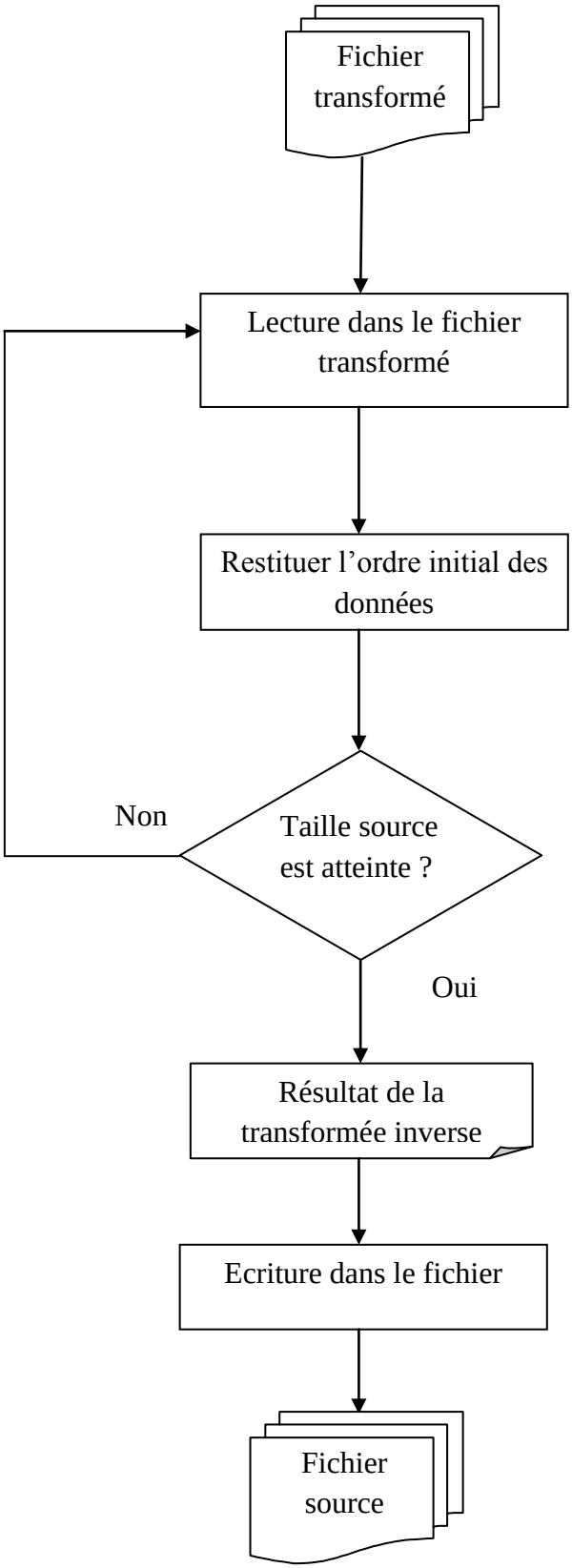

Figure III.10: Illustration du module de transformée inverse.

V.3.Algorithme de calcul du taux de compression :

DEBUT

Récupérer la taille du fichier source Taille_src;

Calculer la taille du fichier compressé Taille_cmp ;

Taux=(1-(Taille_cmp/Taille_src)*100) ;

FIN

V.4.Algorithme de calcul du temps de compression/décompression :

V.4.1.Compression :

DEBUT

Déb=clock() ;

Compression ;

Temps= (Fin-Déb) ;

FIN

V.4.2.Décompression :

DEBUT

Déb=clock() ;

Décompression ;

Temps= (Fin-Déb) ;

FIN

VI. Conclusion :

Dans ce chapitre nous avons spécifié les stratégies qui devront être mise en œuvre pour atteindre notre objectif. Ici nous avons fait l'analyse et la conception de la nouvelle méthode de compression et nous l'avons considéré comme étant un problème (noyau) et décomposé en sous problèmes (modules) et chaque module en sous modules et nous avons illustré tout ça avec des schémas.

Enfin avec ce formalisme assez important, nous sommes en mesure d'aborder l'implémentation qui met en pratique tout ce qui a été présenté dans cette partie, en détaillant davantage les modules et la manière de les implémenter.

Chapitre III :

Analyse et conception de la méthode proposée.

Chapitre IV :

Implémentation et évaluation.

I. Introduction :

Dans ce chapitre nous allons présenter en premier lieu l'environnement de développement de notre application, ainsi le langage de programmation utilisé, en donnant les raisons pour lesquelles on a choisi le langage de programmation C++, puis on va présenter les fonctions principales du noyau, ensuite on va détailler le résultat des différents tests effectués dans le but d'évaluer les performances. A chaque fin d'un test, nous allons comparer les résultats obtenus entre la compression sans et avec utilisation de la méthode proposée.

Les tests de ces deux méthodes de compression sont présentés sous forme des diagrammes formés par l'axe de taux de compression : ces tests ont pour objectif d'évaluer les performances des deux méthodes (avec et sans la transformée).

II. Environnement de développement :

Nous avons choisi pour la réalisation de notre travail le système d'exploitation Microsoft Windows7. Ce choix est fait car les systèmes d'exploitation Windows sont très utilisés grâce à leurs simplicités, efficacités, convivialités et fiabilités.

L'ordinateur sur lequel nous avons fait ces évaluations est doté d'un processeur de fréquence 2.67 GHz et d'une mémoire vive de 4GO.

Pour ce qui concerne la programmation, nous avons choisi le langage C++ et Dev-C++ comme environnement de développement de l'application.

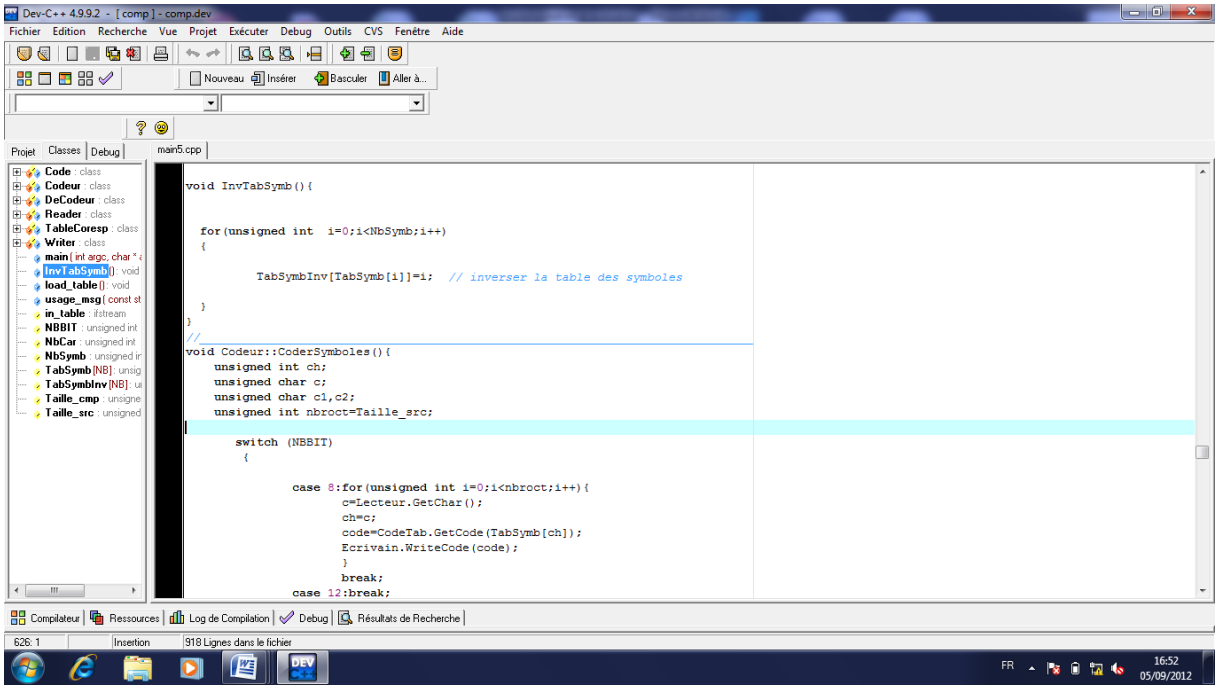

Figure IV.1 : Interface de l'environnement de développement Dev-C++.

II.1. Description du langage C++

II.1.1 Historique [6]:

Le langage C++ a été développé par « Bjarne Stroustrup » au cours des années 1980, alors qu'il travaillait dans le laboratoire de recherche Bell d'AT&T. Il s'agissait en l'occurrence d'améliorer le langage C. Il l'avait d'ailleurs nommé *C with classes* (C avec des classes). Les premières améliorations se concrétisèrent donc par la prise en charge des classes, ainsi que par de nombreuses autres fonctionnalités. En 1989, c'est la sortie de la version 2.0 de C++. Parmi les nouvelles fonctionnalités, il y avait l'héritage multiple, les classes abstraites, les fonctions membres statiques, les fonctions membres constantes, etc….

Comme le langage C++ évoluait, la bibliothèque standard évoluait de concert. La première addition à la bibliothèque standard de C++ concernait les flux d'entrées/sorties qui apportaient les fonctionnalités nécessaires au remplacement des fonctions C traditionnelles telles que *printf* et *scanf*. Ensuite, parmi les additions les plus importantes, il y avait la Standard Template Library.

En langage C, ++ est l'opérateur d'incrémentation, c'est-à-dire l'augmentation de la valeur d'une variable de 1. C'est pourquoi C++ porte ce nom : cela signifie que C++ est un niveau au-dessus du C.

II.1.2. Pourquoi utiliser le C++

- ✓ Il existe de nombreuses bibliothèques C++ en plus de la bibliothèque standard de C++ (*C++ Standard Library*). Par ailleurs, C++ permet l'utilisation de l'ensemble des bibliothèques C existantes.
- ✓ Le C++ est un langage permettant la programmation sous de multiples paradigmes comme la programmation orientée objet qui est très importante dans le développement des applications.
- ✓ Les compilateurs C++ sont actuellement implémentés sur toutes les plates-formes, ce qui fait du langage C++ un outil de programmation très répandu.
- ✓ Le code généré par le compilateur C++ est très optimisé, ce qui rend les exécutables plus compactes et plus rapides.
- ✓ La plupart des implémentations des algorithmes standards sont implémentées à base de langage C++. Il est actuellement le 3^{e} langage le plus utilisé au monde.

III. Les fonctions principales du noyau :

Nous allons présenter dans cette partie les principales fonctions utilisées par notre application, en présentant ainsi le rôle de chacune.

Void transform() : Cette fonction permet de lire le fichier source et d'écrire dans le fichier transformé. Elle fait appel aux fonctions suivantes :

Header_write() :cette fonction est appelée pour écrire l'en-tête du fichier transformé qui contient la taille du fichier source, taille du mot et la taille du code ordre.

Fordre() : retourner le code ordre de chaque mot.

Trier() :Faire le tri pour chaque mot.

Void transform_inv() :son rôle est de lire le corps du fichier transformé(mot trié et son code ordre) et lire l'en-tête avec la fonction Header_read() puis faire appel à la fonction FInverse() et sauvegarder le résultat dans un nouveau fichier de sortie.

Header_read() :elle est appelée pour lire l'en-tête du fichier transformé avant d'effectuer la fonction inverse.

FInverse() :permet de récupérer les données du fichier source.

IV. Evaluation et comparaison :

Dans cette partie on va faire la comparaison entre la compression de données avec et sans la transformée qu'on a présenté précédemment. Pour faire, on choisit « WinRAR ».

Nous allons faire cette comparaison par rapport à trois critères : le type, la taille du mot et aussi la taille du fichier. Dans cette comparaison nous avons pris pour chaque type de fichier (texte, image) un ensemble de fichiers de tailles différentes et nous avons fait les tests sur chacun d'eux, la taille du mot varie entre 2 à 8 caractères.

Ces différents fichiers ont été téléchargé du corpus de Calgary [1] et ont été choisi de façon à être représentatifs des fichiers que nous étions susceptibles de rencontrer.

IV.1. Evaluation du taux de compression :

Pour évaluer le taux de compression nous allons utiliser les résultats donnés par l'application, ces derniers sont calculés par une fonction spéciale.

L'équation pour calculer le taux de compression sous forme de pourcentage est la suivante :

$$\textbf{Taux de compression} = \frac{(taille\ originale - taille\ finale) * 100}{taille\ original}$$

❖ **Les fichiers « texte » :**

Nous commençons par les fichiers texte, pour cela nous avons pris un ensemble de fichiers texte de tailles différentes :

Le nom du fichier	La taille du fichier(Ko)	Taille du fichier transformé/taille mot=2	Taille du fichier transformé/taille mot=3	Taille du fichier transformé/taille mot=5	Taille du fichier transformé/taille mot=8
geo	100	151	134	121	126
obj2	242	362	322	290	302
book2	597	895	796	716	746
world192	2416	3624	3221	2899	3020
bible	3953	5929	5271	4744	4941

Tableau IV.1 : Les tailles des fichiers texte utilisées pour le test d'évaluation.

Taille du mot = 2 :

Les résultats obtenus sont représentés sur la figure suivante :

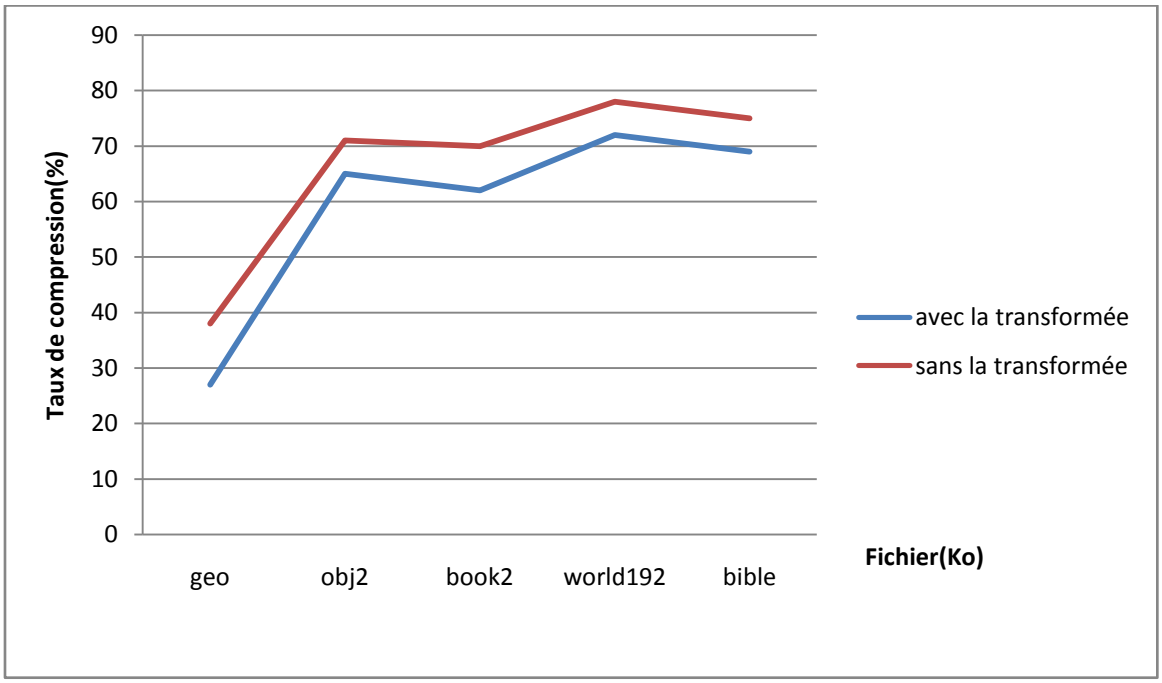

Figure IV.2 : Le taux de compression pour les fichiers texte pour taille du mot=2.

Taille du mot = 3:

Les résultats obtenus sont représentés sur la figure suivante :

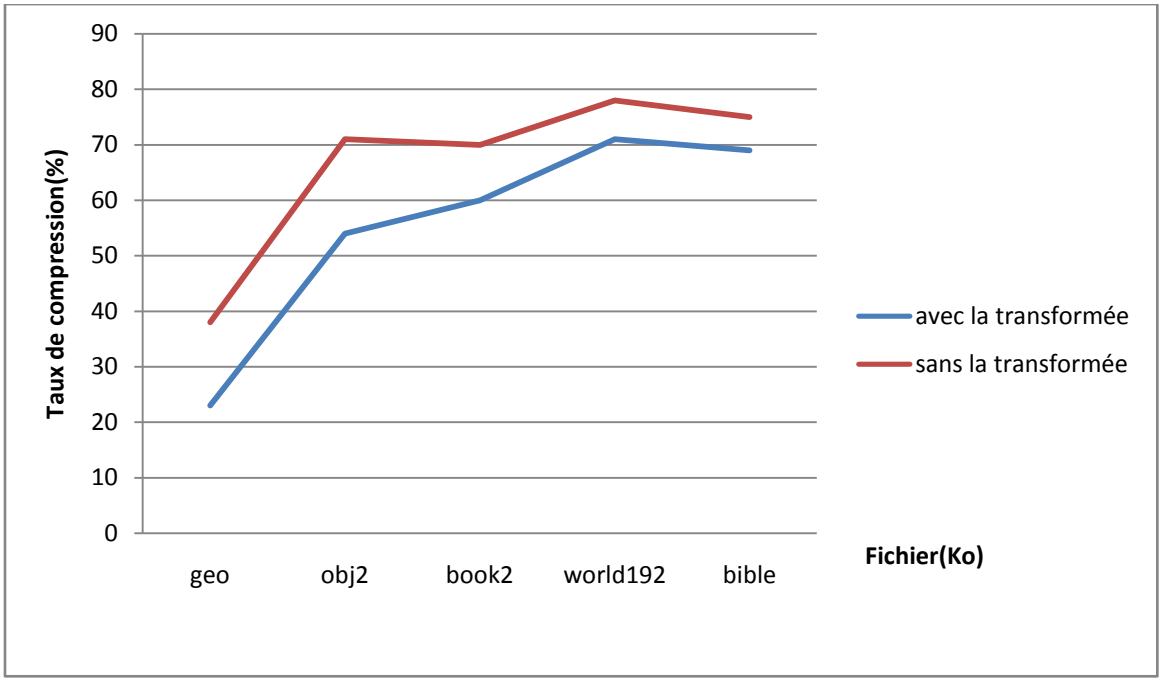

Figure IV.3 : Le taux de compression pour les fichiers texte pour taille du mot=3.

Taille du mot = 5 :

Les résultats obtenus sont représentés sur la figure suivante :

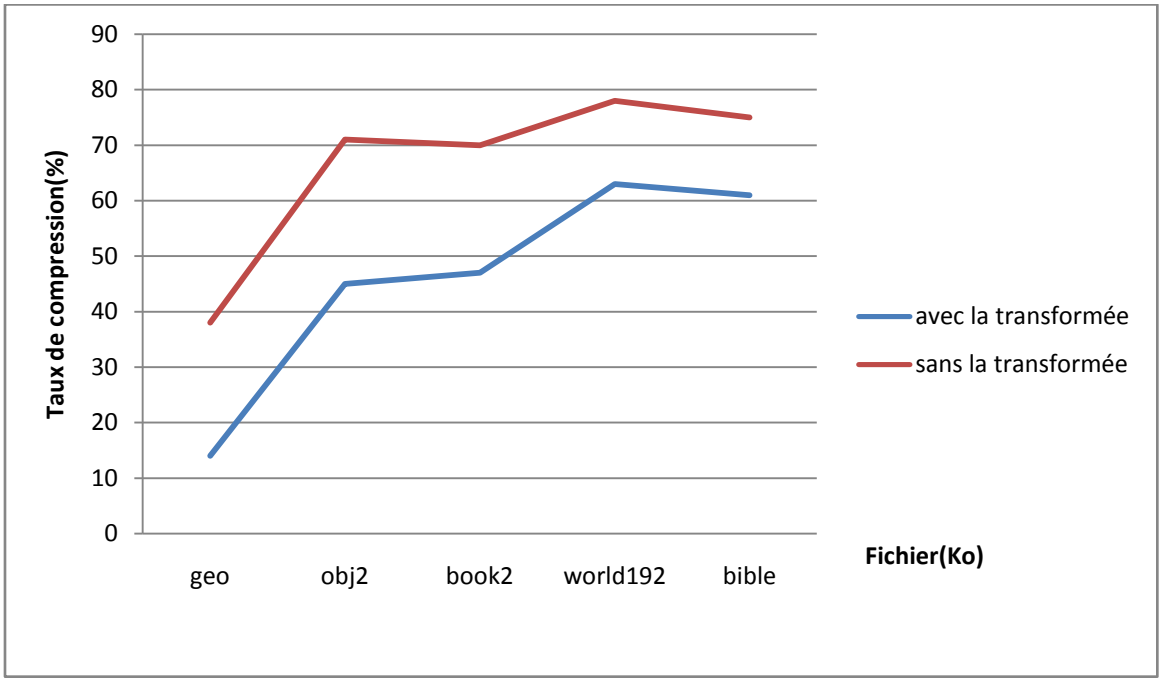

Figure IV.4 : Le taux de compression pour les fichiers texte pour taille du mot=5.

Taille du mot = 8 :

Les résultats obtenus sont représentés sur la figure suivante :

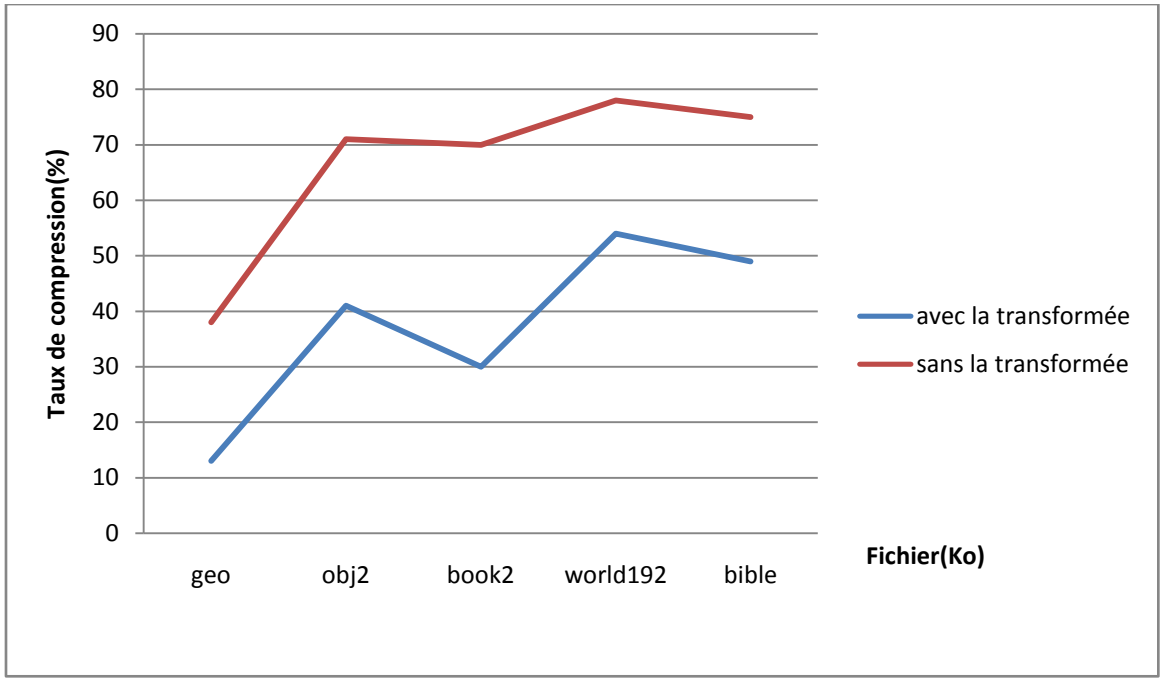

Figure IV.5 : Le taux de compression pour les fichiers texte pour taille du mot=8.

Nous remarquons à partir des figures précédentes que le taux de compression de la méthode de compression sans la transformée est meilleur que celle de la méthode de compression avec la transformée et ceci pour toutes les tailles des mots choisies.

❖ **Les fichiers « image » :**

Pour l'évaluation du taux de compression pour les fichiers image on a pris une collection d'image de deux formats différents (tif, dicom), avec des tailles différentes :

1. ***Extension .tif***

Le nom du fichier	**La taille du fichier(Ko)**	Taille du fichier transformé/taille mot=2	Taille du fichier transformé/taille mot=3	Taille du fichier transformé/taille mot=5	Taille du fichier transformé/taille mot=_
lukas_2d_8_foot_1_t.tif	2184	3275	2911	2620	2729
lukas_2d_8_hand_1_t.tif	2476	3714	3302	2972	3095
lukas_2d_8_head_1_t.tif	2547	3821	3396	3057	3184
lukas_2d_8_thorax_0_t.tif	3455	5183	4607	4146	4319
lukas_2d_8_breast_1_t.tif	3593	5389	4790	4311	4491

Tableau IV.2 : Les tailles des fichiers image (.tif) utilisées pour le test d'évaluation.

Taille du mot =2 :

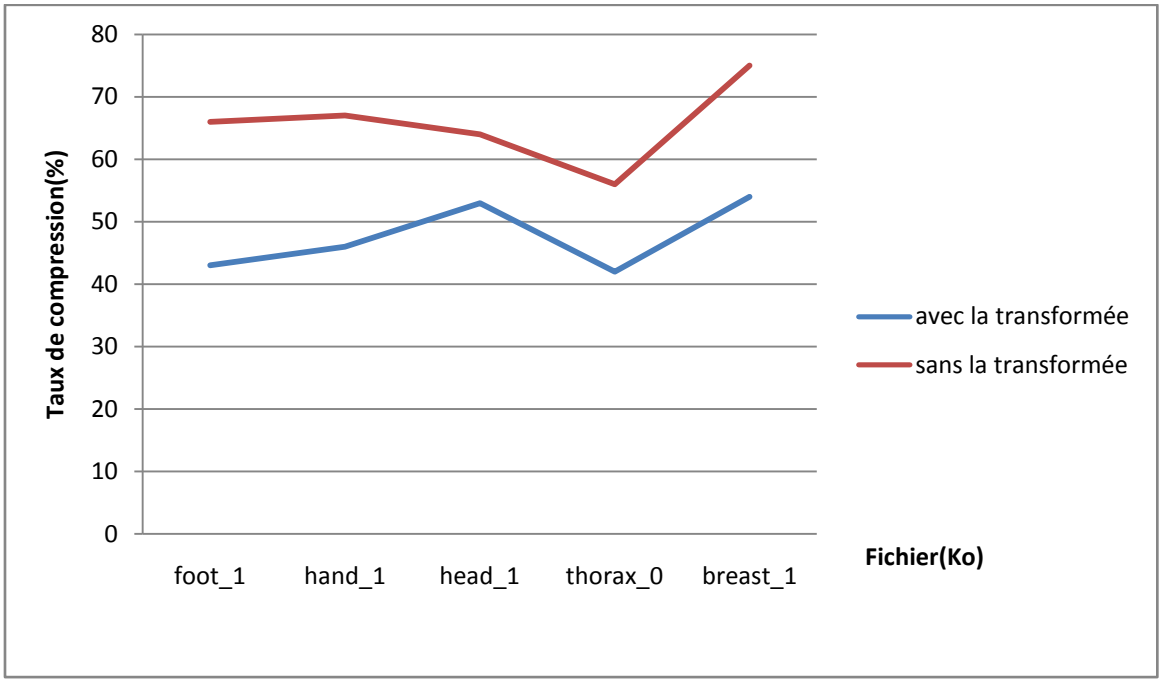

Figure IV.6 : Le taux de compression pour les fichiers image (.tif) pour taille du mot=2.

Taille du mot=3 :

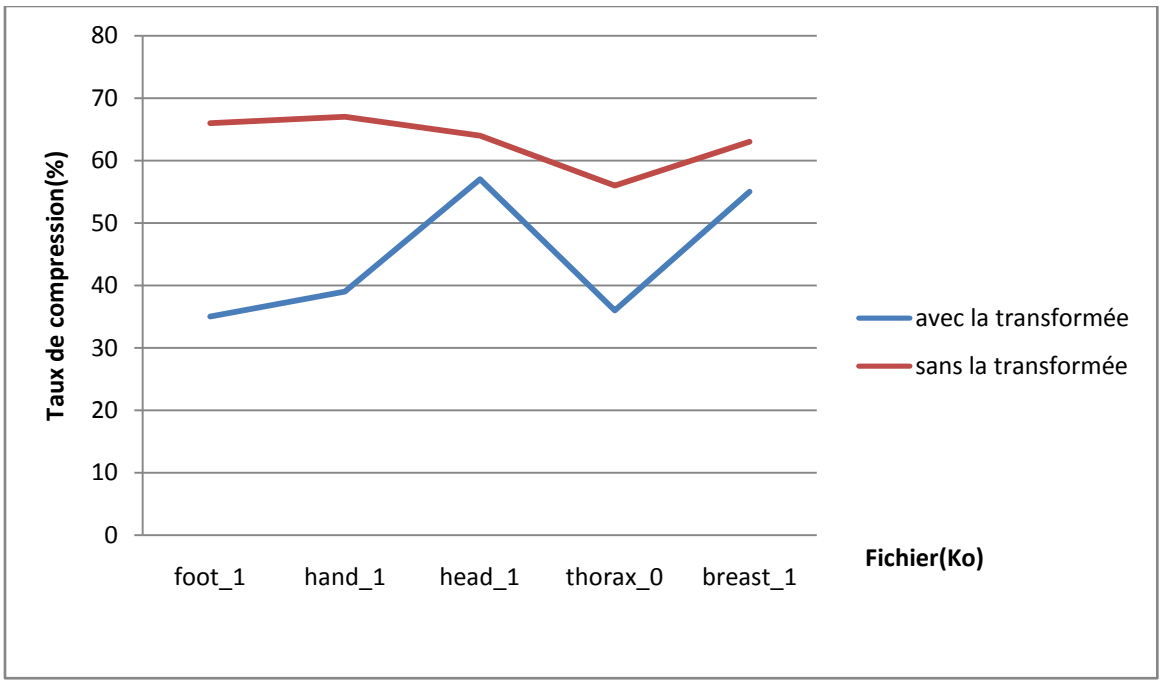

Figure IV.7 : Le taux de compression pour les fichiers image (.tif) pour taille du mot=3.

Taille du mot=5 :

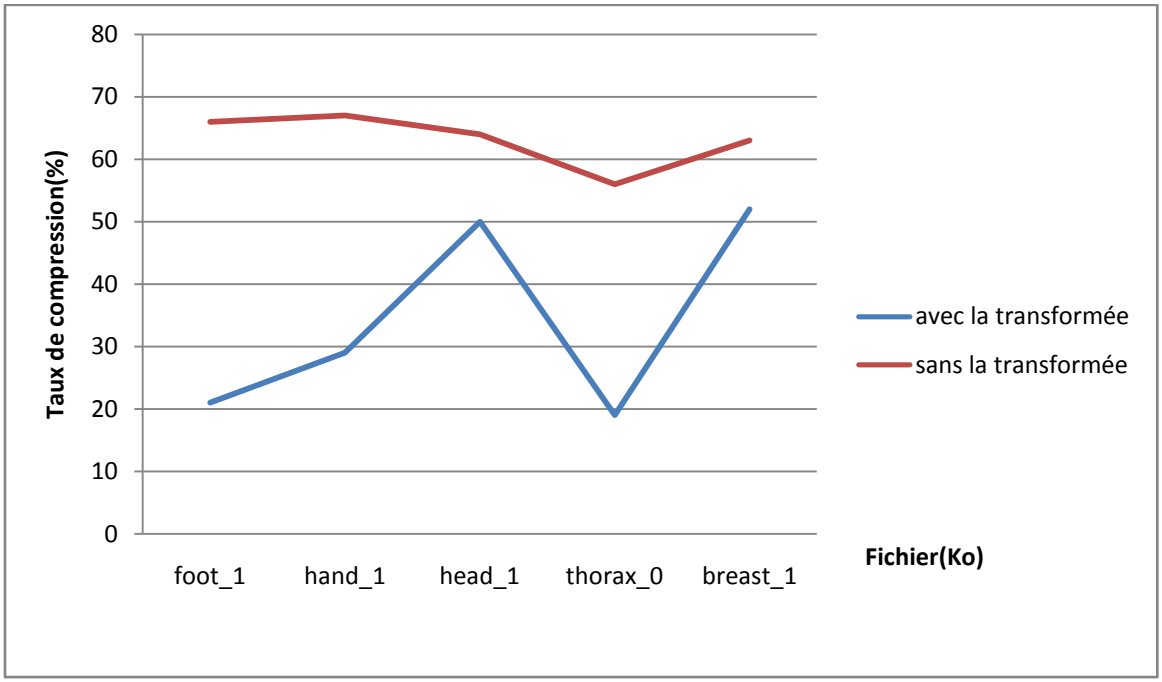

Figure IV.8 : Le taux de compression pour les fichiers image (.tif) pour taille du mot=5.

Taille du mot=8 :

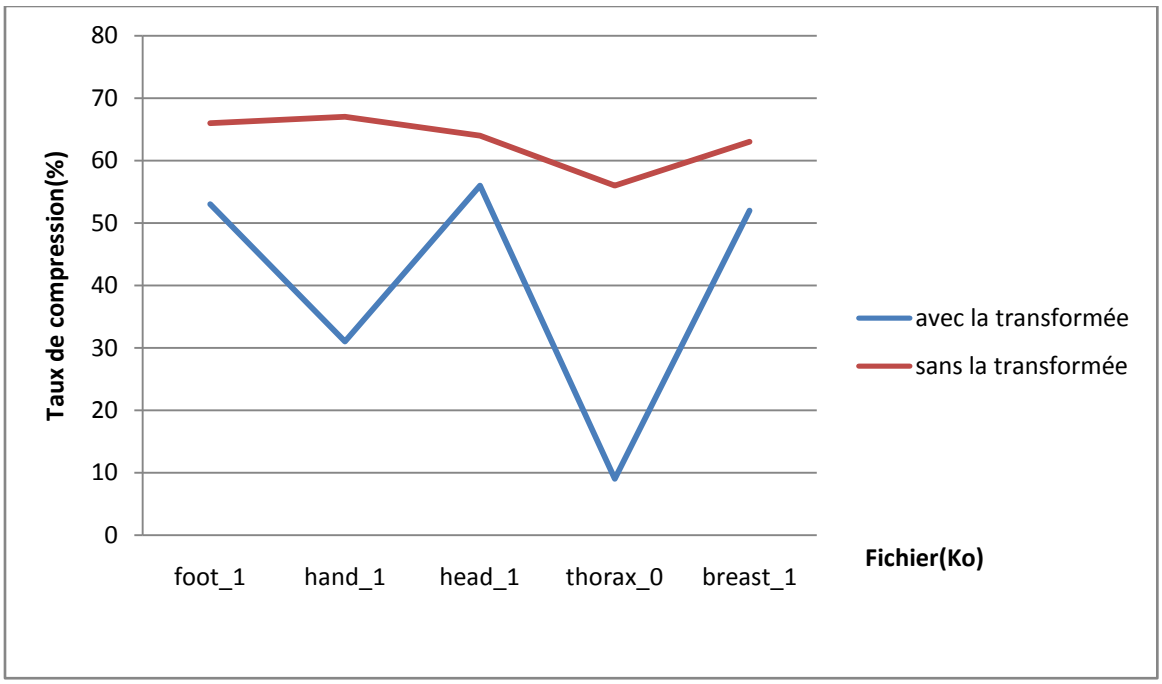

Figure IV.9 : Le taux de compression pour les fichiers image (.tif) pour taille du mot=8.

2. *Extension .dcm*

Le nom du fichier	La taille du fichier(Ko)	Taille du fichier transformé/taille mot=2	Taille du fichier transformé/taille mot= 3	Taille du fichier transformé/taille mot=5	Taille du fichier transformé/taille mot=8
lukas_2d_16_spine_1.dcm	3680	5520	4907	4416	4600
lukas_2d_16_thorax_1.dcm	6052	9078	8070	7263	7565
lukas_2d_16_head_1.dcm	6085	9127	8113	7301	7606
lukas_2d_16_pelvis_1.dcm	7359	11038	9811	8830	9198
lukas_2d_16_hand_1.dcm	8195	12293	10927	9834	10244

Tableau IV.3 : Les tailles des fichiers image (.dcm) utilisées pour le test d'évaluation.

Taille du mot =2:

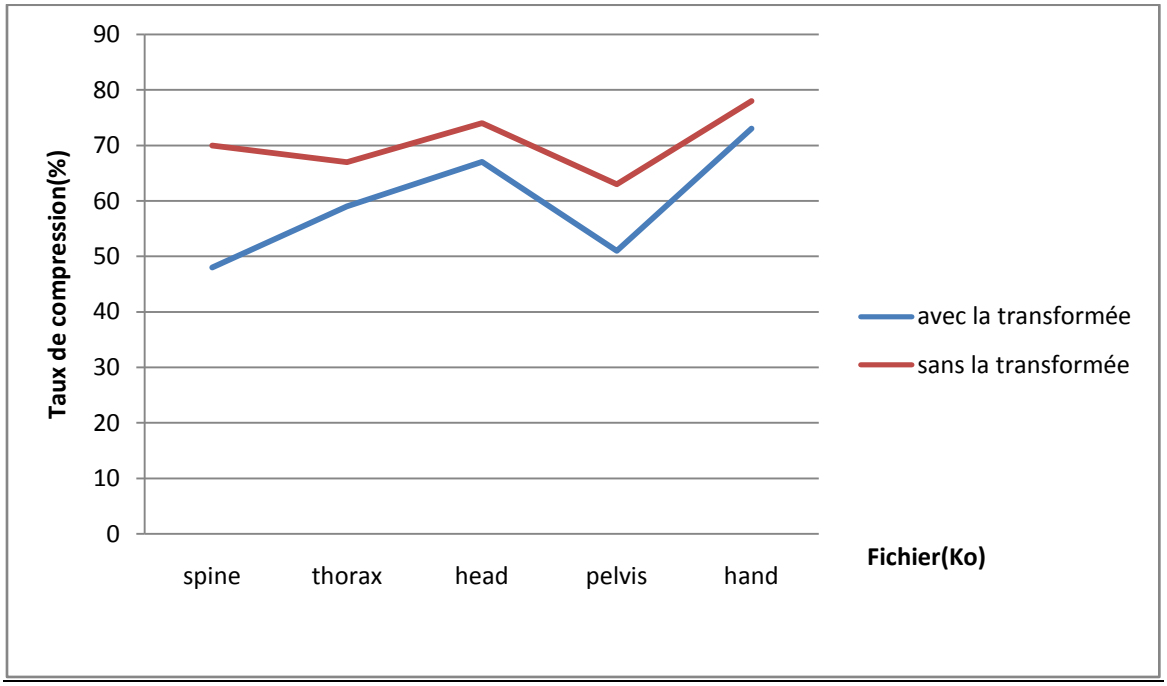

Figure IV.10:Le taux de compression pour les fichiers image (.dcm)pour taille du mot=2.

Taille du mot=3 :

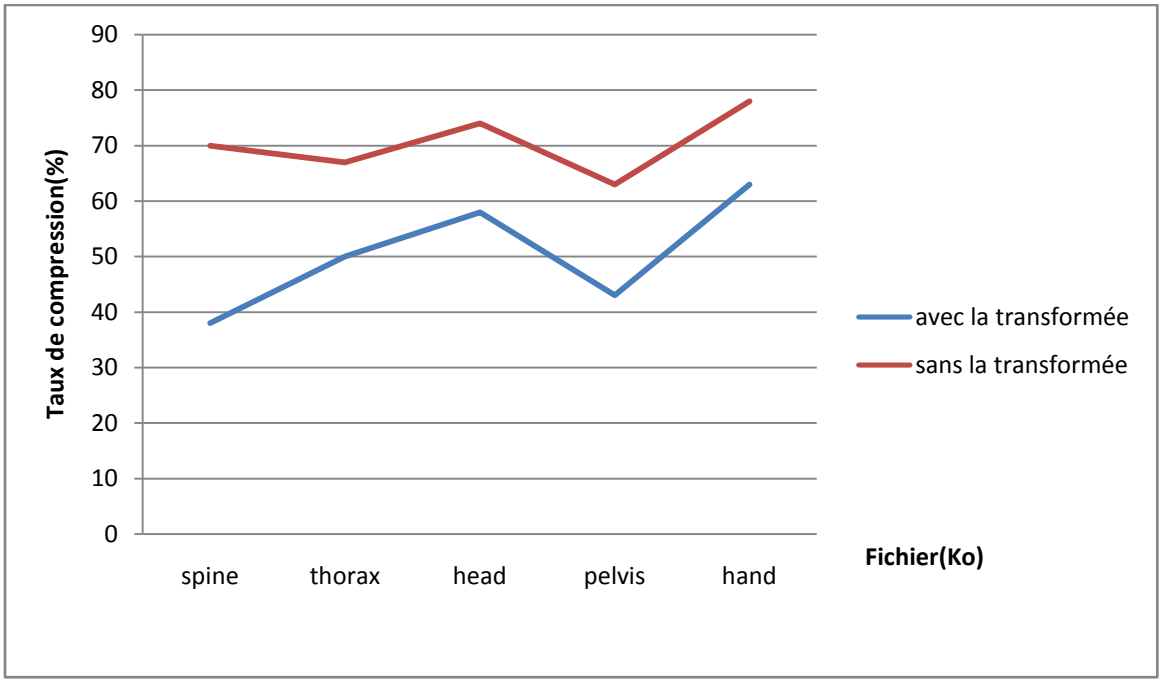

Figure IV.11:Le taux de compression pour les fichiers image (.dcm)pour taille du mot=3.

Taille du mot=5 :

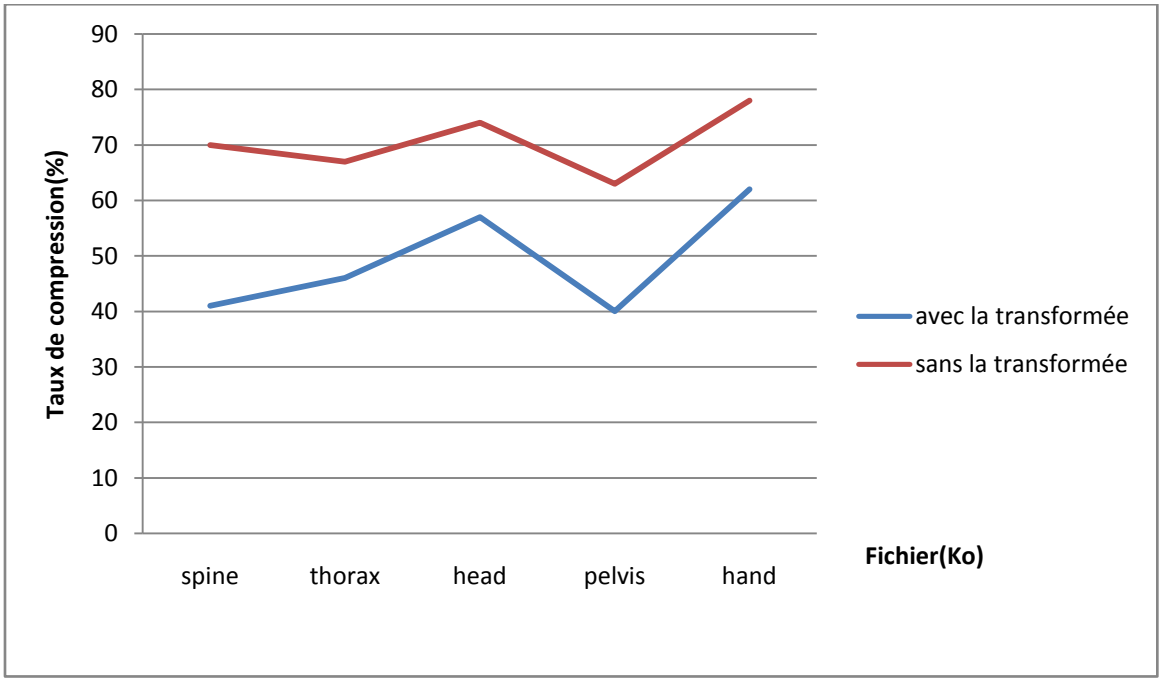

Figure IV.12:Le taux de compression pour les fichiers image (.dcm)pour taille du mot=5.

Taille du mot=8 :

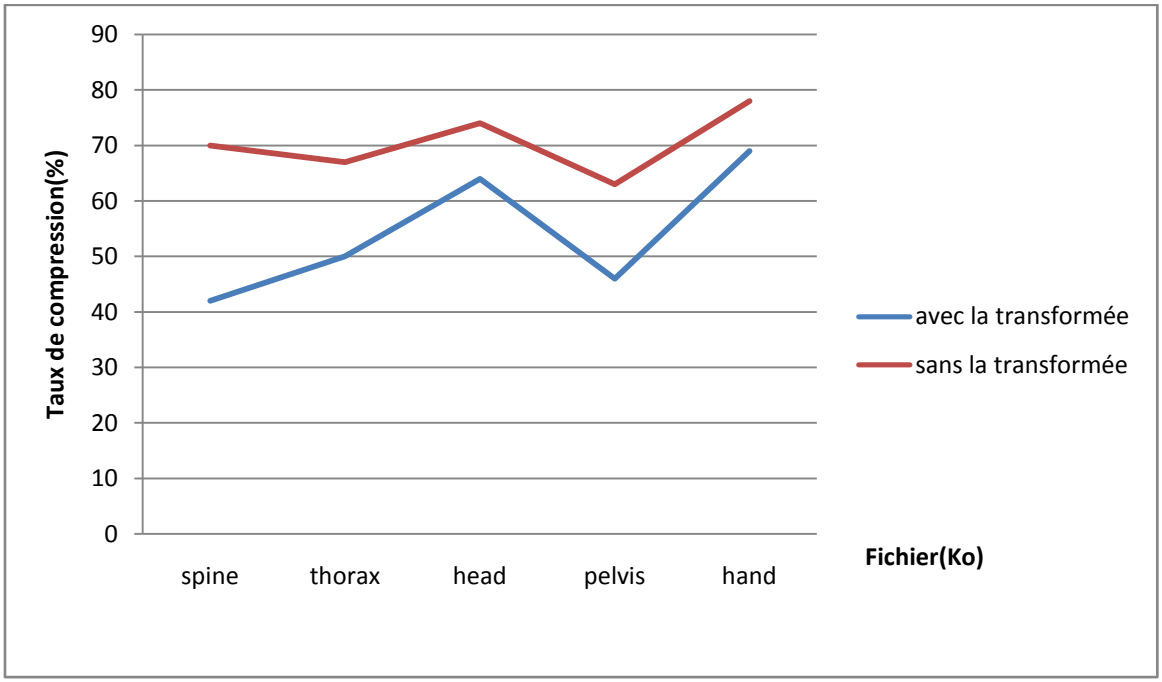

Figure IV.13:Le taux de compression pour les fichiers image (.dcm)pour taille du mot=8.

Dans ces dernières figures, nous remarquons que le taux de compression de la méthode de compression sans la transformée est meilleur, contrairement à la méthode de compression avec la transformée.

- **Fichier « web » :**
  Pour l'évaluation du taux de compression pour les fichiers web, on a pris un ensemble des pages web (d'extension .html) avec des tailles différentes.

Le nom du fichier	La taille du fichier(Ko)	Taille du fichier transformé/taille mot=2	Taille du fichier transformé/taille mot=3	Taille du fichier transformé/taille mot=5	Taille du fichier transformé/taille mot=8
B01	2076	3114	2768	2492	2595
B02	2084	3126	2779	2501	2605
B03	2123	3185	2831	2548	2654

Tableau IV.4 : Les tailles des fichiers web (.html) utilisées pour le test d'évaluation.

Taille du mot=2 :

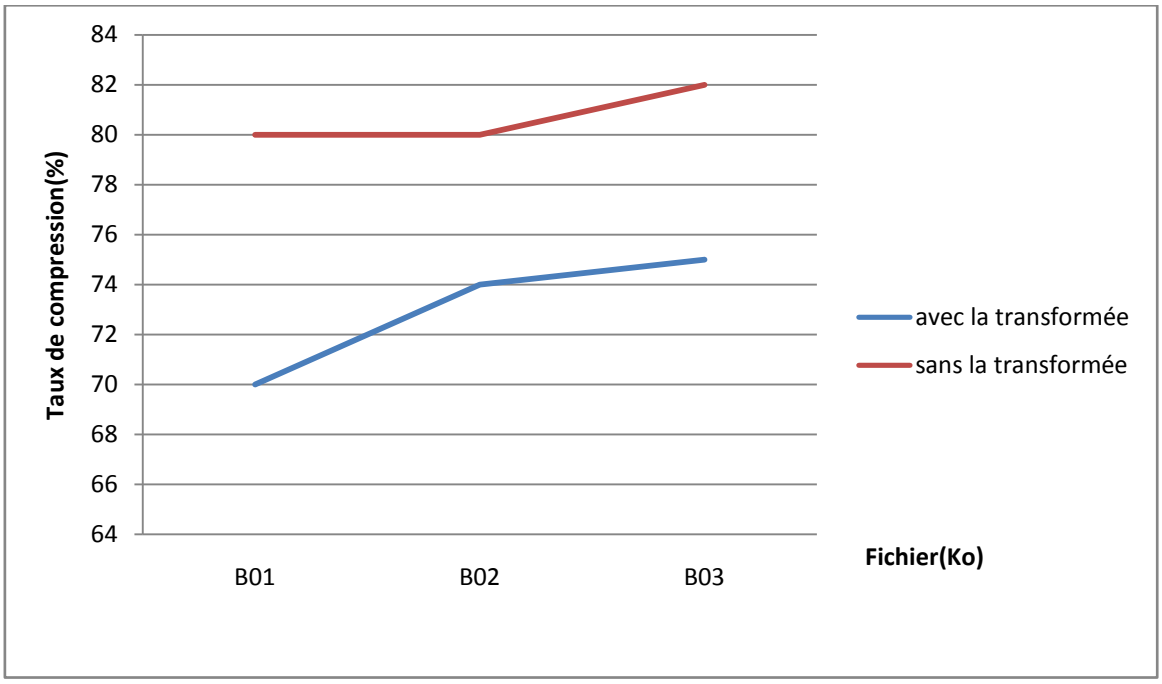

Figure IV.14 : Taux de compression pour les fichiers web (.html) pour taille du mot=2.

Taille du mot=3 :

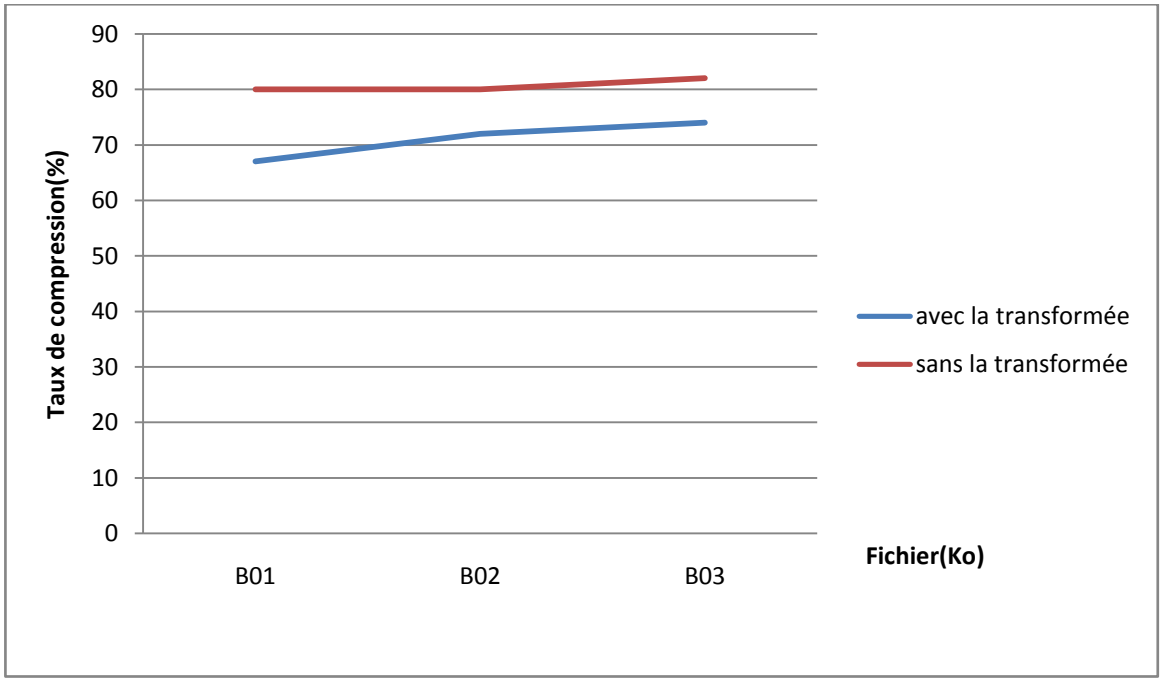

Figure IV.15 Taux de compression pour les fichiers web (.html) pour taille du mot=3.

Taille du mot=5 :

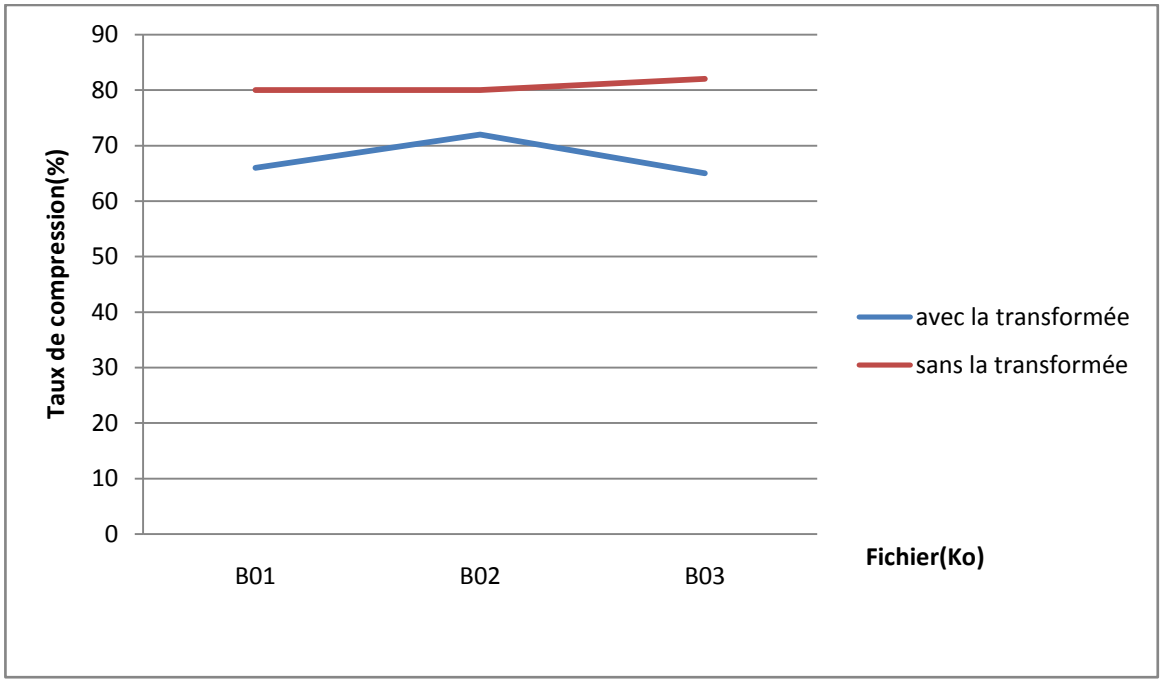

Figure IV.16 : Taux de compression pour les fichiers web (.html) pour taille du mot=5.

Taille du mot=8 :

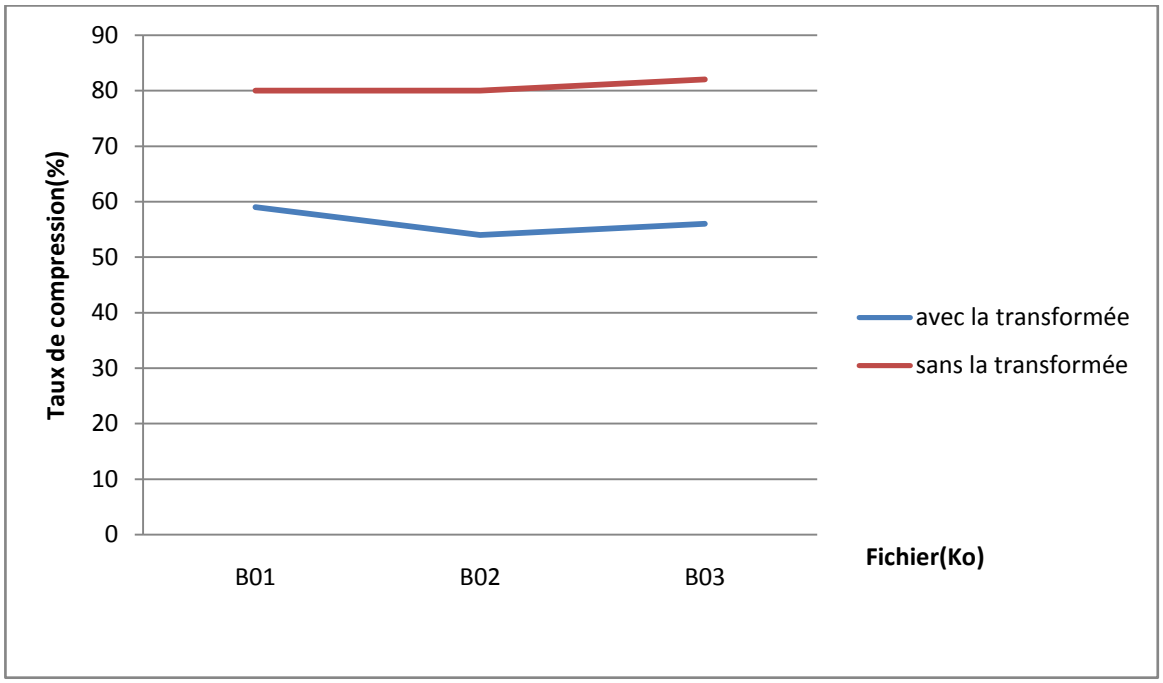

Figure IV.17 : Taux de compression pour les fichiers web (.html) pour taille du mot=8.

On remarque que le taux de compression pour la méthode de compression sans la transformée est un peu mieux que celui de la méthode de compression avec la transformée et ceci pour les tailles choisies.

IV.2.Interprétation des résultats du taux de compression :

Après avoir vu les résultats obtenus des différents tests d'évaluation effectués sur les deux méthodes de compression, on les interprétera dans ce qui suit :

Malgré l'augmentation de la taille du fichier source par l'ajout du code d'ordre pour chaque mot (la taille de code ordre=1bit pour taille du mot =2 et égal à 16bit pour taille du mot =8), nous remarquons que le taux de compression de la méthode de compression avec la transformée pour la taille du mot =2 est beaucoup mieux par rapport aux autres tailles et ceci pour tout type de fichiers choisis.

Pour les fichiers image, le taux de compression de la méthode de compression avec la transformée est proportionnel à la taille du fichier.

Pour les fichiers web, nous avons obtenus des taux de compression élevés pour les deux méthodes et ceci pour toutes les tailles du mot des différents fichiers.

V. Conclusion

Dans ce chapitre on a présenté les différents résultats obtenus par les deux méthodes de compression avec et sans la transformée après les tests effectués avec les interprétations des schémas.

Conclusion générale

Conclusion générale

Ce travail nous a permis, de découvrir le monde de la compression de données en passant en revue par les différentes méthodes et techniques de compression de données, il nous a donné l'idée sur les différences entre ces méthodes où chaque algorithme possède ces propres caractéristiques.

Notre objectif dans ce travail est l'implémentation et l'évaluation d'une nouvelle transformée pour la compression de données sans perte, basé sur la fonction d'ordre, pour ce faire il est nécessaire de comprendre divers concepts dans le monde de la compression de données et de la programmation

A la fin de notre travail, nous avons constaté que cette nouvelle transformée donne des bons résultats de taux de compression pour les fichiers image et web et cela dépond de la taille du mot.

Cette expérience nous a permis donc d'acquérir beaucoup de connaissance dans un monde aussi vaste qui est la compression de données, les différents algorithmes utilisés ainsi de mieux se familiariser avec le langage C ++.

Nous précisons que les différentes parties de notre application peuvent être maintenues et optimisées pour donner encore de bons et de meilleurs résultats.

Enfin, nous espérons avoir parvenu à répondre à l'objectif fixé initialement, à savoir l'implémentation et l'évaluation d'une nouvelle transformée dans la compression de données sans perte basée sur la fonction d'ordre.

Bibliographie

BIBLIOGRAPHIE

[1]: Sciences de l'information et de la communication, Bougnoux Daniel Édition Larousse, collection Textes essentiels, 1994.

[2] : Transmission de l'information, cours de l'EPU de Tours.

[3]: http:// iict/Tcom/cours/PDF/compression.pdf/

[4]: http:// di.ens.fr/pointche/enseignement/enste2/

[5]: data compression, the complete reference, David Salomon, 2004, springer.

[6]: http://www.wikipedia .org/.

[7] : Alexandre THIL- Master1 Informatique –Université de Metz.

[8] : http://www.commentcamarche.com/

[9] : Pereira Vincent-LEPRETTE Franck-HACAULT Vincent : Compression de données, Décembre 2004.

[10]: cours de compression, Dr MVogo Nogono Foseph Master 2.

[11]: Introduction to data compression, Khalid Sayood, 2006, Morgan Kaufmann.

[12]: http:// www.dataligence.com/

[13] :http://www.ulb.ac.be/cours/acohen/travaux_2006_infodoc/CompressionNumerique/AvecPertes.htm.

[14] : les formats de compression d'image, MICHOT JULIEN, 2004.

[15] : T.I.P.E la compression JPEG, FLIEDEL ROMAIN, 2005.