الجمهورية الجزائرية الديمقراطية الشعبية

République Algérienne Démocratique et Populaire

وزارة التعليم العالي والبحث العلمي

Ministère de l'Enseignement Supérieur Et de La Recherche Scientifique

جامعة غرداية

Université de Ghardaïa

كلية العلوم والتكنولوجيا

Faculté des Sciences et Technologies

قسم الآلية والكهروميكانيك

Département d'automatisation et d'électromécanique

N°D'enregistrement :

……

Mémoire présenté en vue de l'obtention du diplôme de

MASTER

Domaine : Automatique et électromécanique
Filière : Automatique
Spécialité : Automatique et système

Thème

Compression de texte

Par : BLIDI Amine et SEKKOUTI Yacine

Devant les jurys :

Nom et Prénom	Grade		
LADJAL Boumedienne	**MAA**	**Ghardaia**	**Examinateur**
BENAOUICHA Karim	**MAA**	**Ghardaia**	**Examinateur**
BITEUR Kada	**MAA**	**Ghardaia**	**Encadreur**

Année Universitaire : 2021 / 2022

Dédicace

Nous tenons à remercie tout d'abord le Dieu puissant qui a écrit dans notre destin d'arriver jusqu'au là et nous remercions nos parents, pour leur soutien constant et leurs encouragements depuis toujours.

Nous tenons à remercier toutes les personnes qui ont contribué au succès dans tous notre parcours éducatifs et surtout les personnes qui nous ont aidés lors de la rédaction de ce mémoire.

On voudrait dans un premier temps remercier, mon encadreur de mémoire M.Biteur, professeur de traitement de signal à l'université de Ghardaia , pour sa patience, sa disponibilité et surtout ses judicieux conseils, qui ont contribué à alimenter ma réflexion.

On remercie également toute l'équipe pédagogique de l'université de Ghardaia et les intervenants professionnels responsables de ma formation, pour avoir assuré la partie théorique de celle-ci.

RÉSUMÉ

La compression des données est la science et l'opération de représenter les informations sous une forme compacte. Depuis des années, la compression des données est considérée comme l'une des technologies importantes pour la révolution multimédia. pour compresser des fichiers de différents formats y en a plusieurs algorithmes de compression de données .dans cette thèse on va présenter les différents algorithmes de base de compression de données sans perte en basant sur différentes techniques tels que Codage de Huffman , Codage arithmétique, Dictionary Techniques et La Transformée de Burrows-Wheeler (BWT).A la fin, on va faire des comparaisons pour avoir un bilan entre deux techniques pour voir les différentes résultats .

Les mots clé :

Compression de données, Codage de Huffman , Codage arithmétique, Dictionary Techniques et La Transformée de Burrows-Wheeler (BWT)

ABSTRACT

Data compression is the science and operation of represent information in a compact form. Since years, data compression has been considered one of the important technologies for the multimedia revolution. To compress files of different formats there are several data compression algorithms. In this thesis we will present the different basic lossless data compression algorithms based on different techniques such as Huffman Coding, Arithmetic Coding, Dictionary Techniques and The Burrows-Wheeler Transform (BWT). At the end, we will make comparisons to have an assessment between two techniques to see the different results.

Key words:

Data compression, Huffman Coding, Arithmetic Coding, Dictionary Techniques and The Burrows-Wheeler Transform (BWT).

ملخص

ضغط البيانات هو علم وعملية تمثيل المعلومات في شكل مضغوط. منذ سنوات ، يعتبر ضغط البيانات أحد التقنيات المهمة لثورة الوسائط المتعددة. لضغط الملفات ذات التنسيقات المختلفة ، توجد عدة خوارزميات لضغط البيانات. في هذه الأطروحة سوف نقدم مختلف خوارزميات ضغط البيانات الأساسية التي لا يمكن ان تفقد معلومات اثناء العملية ، وهذا بناءً على تقنيات مختلفة مثل تشفير هوفمان ، والترميز الحسابي ، وتقنيات القاموس ، وتحويل بوروز ويلر (BWT). في الأخير ، سنجري مقارنات لإجراء تقييم بين تقنيتين لمعرفة النتائج المختلفة.

كلمات مفتاحية:

ضغط البيانات، تشفير هوفمان ، والترميز الحسابي ، وتقنيات القاموس ، وتحويل بوروز ويلر

Table des matières

Liste des tableaux :

Table des figures :

I. Introduction générale :

L’objectif de compression de données est dans l’intérêt de gagner l’espace de mémorisation en réduisant la taille des données, aussi avoir le minimum de période de transmission à travers les réseaux. Cette opération rend la diffusion plus facile grâce à l’utilisation des supports magnétiques de faible capacité tel que les disquettes, elle permet aussi d’améliorer le stockage de données volumineuses.

En parlant de compression de données, la nature des données traitées nous intéresse pas que se soient : texte, son, photo, dessins vectoriels.

Les données à une seule dimension celle des codages des documentations de langue naturelle devaient être traiter par les premiers algorithmes de compression, Peu à peu, on était obligé de faire pareil avec les images, tout d’abord en niveau de gris ensuite en couleurs ; donc ces algorithmes de compression ils ont été développé spécialement pour être adaptés à ces données bidimensionnelles.

En revenant sur les compressions qui doivent traiter les textes l’homme avait besoin de développer les algorithmes classiques en proposant des améliorations pour avoir un taux de compression meilleur.

Fondamentalement, les méthodes de compression de données sont généralement classées en deux catégories qui sont comme suivant :

- Avec pertes (Lossy) :

La compression avec perte ça s'utilise typiquement pour du son, une image, une vidéo. On peut se permettre de perdre de l'information donc pour une image si on ne compte pas toute l'information ce ne sera peut-être pas forcément perceptible pour les humains donc on peut se permettre de ne pas avoir cette information.

- Sans perte (Lossless) :

La compression sans perte dite aussi compactage ça s'utilise typiquement pour les textes ou une Archive Zip c'est à dire qu’on ne peut pas se permettre de perdre de l'information durant l’opération de compression ou la décompression.

Généralement, DC a été utilisé pour compresser du texte, des images, audio, vidéo et certaines données spécifiques à l'application. Cette section passe en revue la plupart des techniques de compression importantes et récentes sous chaque type de données et la hiérarchie de classification. est indiquée dans figure 1.

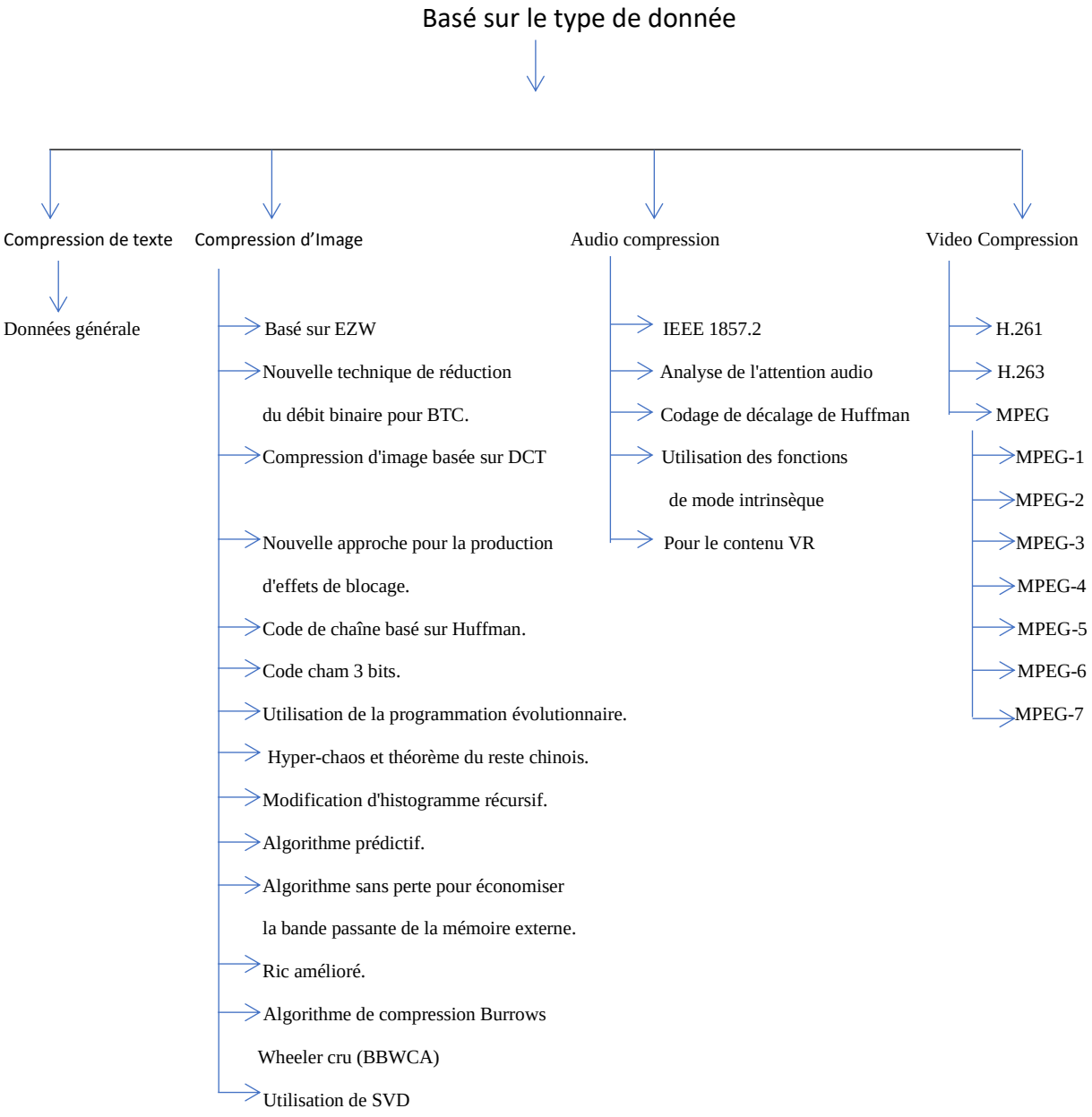


Figure 1 : Classification des techniques de compression par type de donné.

I. Capitre01 : Généralité de compression de donné :

1.1 Introduction :

Aujourd'hui, les médias de plus en plus sont numérisés (images, audio, vidéo...) et archivés ou transmis sous un format numérique. Cela présente plusieurs avantages, et le plus important, c'est que la qualité du support numérique n'est pas dégradée lors de sa transmission ou de son traitement (par exemple, à l'aide d'un logiciel). Les caractères numériques présentent également certains inconvénients inhérents, particulièrement la perte de précision lors de la conversion analogique ou numérique, et inversement.

Pour que le support numérisé soit exactement le même que le signal d'origine, ce dernier doit être numérisé avec une précision infinie, stockant ainsi une quantité infinie d'informations. Ceci est évidemment très théorique et impossible à réaliser. En pratique, les limites de précision nécessaires sont généralement déterminées par le destinataire ultime du support : les yeux pour les images, les oreilles pour le son.

1.2 Qu'est-ce que la compression ?

Aujourd'hui, la puissance du processeur augmente plus rapidement que la capacité de sauvegarde et d'archivage et beaucoup plus rapidement que la bande passante du réseau, car elle nécessite des modifications radicales de l'infrastructure des télécommunications.

Par conséquent, afin de pallier ce manque, en utilisant la puissance du processeur, la taille des données est généralement réduite, plutôt qu'en augmentant la capacité de stockage et de transmission des données.

La compression est faite pour minimiser la taille physique des blocs d'informations. Le compresseur fonctionne grâce à un algorithme qui optimise les données avec des considérations spécifiques au type de données compressées. Pour reconstruire les données d'origine il est donc nécessaire d'avoir un décompresseur, et pour l'avoir il nous faut un algorithme opposé à celui utilisé pour la compression.

Donc le but de compression est de réduire la longueur d'une chaîne et de son stockage sans prendre à son contenu informatif. Donc cela permet à la fois de minimiser les exigences en mémoire et d'augmenter la capacité d'un canal de transmission (théorie de Shannon). Elle est présenté par la formule : $-\log_2 p$ (1)

Si p est la probabilité d'occurrence du message. les probabilités d'occurrence dont l'ensemble X composé de N messages sont données par p_1, ..., p_N . Alors :

$$\sum_{i=1}^{N} pi=1 \quad (2)$$

L'information H associée à ces N messages est définie comme la « surprise moyenne » :

$$\sum_{i=1}^{N} pi \log pi \quad (3)$$

H(x) permet de mesurer l'information. On lui donne le nom d'Entropie.

$$0 \leq H < \infty$$

et est maximale quand $p_1 = p_2 = ... = p_N = \frac{1}{N}$

Longueur moyenne d'un code :

Considérons un codage de message alphabétique de (f) lettres tel qu'il y ait une correspondance unique entre chaque message et son code. Si un message (i) est représenté par une séquence de (L_i) caractères, la longueur moyenne du code est donnée par :

$$\bar{L}=\sum_{i=1}^{N} pi\ li \quad (4)$$

Efficacité et redondance d'un système de codage :

On peut définir l'efficacité d'un système de codage Par le théorème du codage de Shannon.

$$\bar{L} \geq \frac{H(X)}{logf} \quad (5)$$

Donc

$$\frac{H(X)}{\bar{L}\ logf} \leq 1 \quad (6)$$

qui est la formule définissant l'efficacité d'un système de codage.

La redondance est donnée par R = 1 − efficacité.

1.3 Compression avec perte (lossy) :

La compression avec perte s'applique uniquement aux données "perceptibles" (généralement audio ou images). Ces données sont sujettes à des changements potentiellement importants sans perceptibilité humaine. La perte d'informations est irréversible. Après une telle compression c'est impossible de pouvoir retrouver les données d'origine. Par conséquent, la compression avec perte est parfois appelée compression avec perte ou compression non conservatrice.

L'idée derrière cette technique est simple. Seul un sous-ensemble particulièrement restreint de l'ensemble des images possibles (c'est-à-dire celles obtenues, par exemple, en randomisant les valeurs de chaque pixel) ont des caractères utiles et informatifs visibles. Par conséquent, nous nous efforçons d'encoder ces images immédiatement. En pratique, l'œil doit identifier les zones où il y a une corrélation entre pixels adjacents, donc où il y a des zones contiguës de couleur adjacente. Les programmes de compression se concentrent sur la recherche de ces zones et sur leur encodage aussi compact que possible. Par exemple, la norme JPEG 2000 peut généralement encoder des images photographiques à 1 bit par pixel sans aucune perte notable de qualité à l'écran. C'est-à-dire de 24x à 1x de compression.

Étant donné que l'œil ne perçoit pas nécessairement tous les détails d'une image, la quantité de données peut être réduite de sorte que le résultat pour l'œil humain soit particulièrement similaire ou similaire à l'image d'origine. Le problème de la compression avec perte est d'identifier les transformations d'image ou audio qui permettent de réduire la quantité de données tout en préservant la qualité perceptible.

De même, puisque seul un sous-ensemble particulièrement faible de sons possibles est utilisé par l'oreille, une régularité qui crée elle-même des redondances est nécessaire (encoder fidèlement les sons respiratoires est moins important). Par conséquent, les encodages qui éliminent cette redondance et restituent à l'arrivée restent acceptables même si le son restitué ne ressemble pas en tous points au son d'origine.

On peut citer trois grandes familles de compression avec perte :

- par prédiction, par exemple l'ADPCM ;
- par transformation. Les méthodes les plus efficaces et les plus utilisées sont (JPEG 2000,JPEG, la totalité des normes MPEG…) ;
- compression basée sur la récurrence fractale de motifs (Compression fractale).

Le format MPEG est un format de compression avec perte pour les séquences vidéo. C'est pourquoi ils contiennent des encodeurs audios bien connus tels que MP3 et AAC, qui peuvent idéalement être utilisés indépendamment, et bien sûr des encodeurs vidéo et aussi des solutions pour la synchronisation des flux audio et vidéo et leur transport sur différents types de réseaux.

1.3.1 Techniques de compression avec pertes :

La compression avec perte ne se fonctionne que ave les données perceptives (vidéo, images, audio) et sa stratégie de réduction de l'information dépend des caractéristiques du système visuel ou auditif humain. Les techniques sont donc propres à chaque support. Ces technologies ne sont pas utilisées seules, mais en combinaison pour fournir un dispositif de compression puissant. [1]

1.3.2 Sous-échantillonnage :

Le sous-échantillonnage spatial des composants de chrominance est courant dans les images et la vidéo. Le système visuel humain est plus sensible aux changements de luminosité qu'aux changements de couleur, de sorte que la perte d'informations de couleur significatives n'est que légèrement visible. [1]

1.3.3 Quantification :

La quantification c’est la plus importante l'étape dans la réduction de l'information. Il s'agit de la quantification à utiliser lorsque vous essayez d'atteindre un taux cible, principalement en utilisant un modèle taux-distorsion.

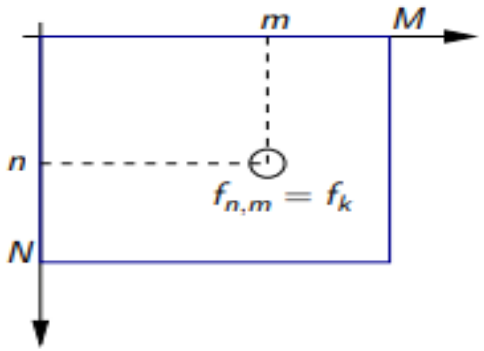
Lors du codage par une transformée (telle qu'ondelette ou DCT), la quantification est appliquée aux coefficients dans l'espace transformé, ce qui entraîne une perte de précision. [1]

1.3.4 Compression d’image :

La compression d'image est une application de compression de données qui encode des images Quelques morceaux de l'original . Les trois formats de compression les plus courants incluent JPEG, GIF et PNG. Compression JPEG, couramment utilisée pour les photos numériques, combinés à un algorithme de compression avec perte pour des couleurs approximatives et élimine les changements de couleur imperceptibles à l'œil humain. Compression GIF réduit la palette d'une image à 256 couleurs ou moins, ce qui permet de représenter efficacement toutes les couleurs de l'image. La compression PNG utilise un algorithme de compression sans perte, filtrage des données d'image et prédiction de la couleur des pixels caractéristiques des autres pixels à proximité. Bien

que chacun de ces algorithmes fonctionne différemment, Ils peuvent être utilisés pour réduire la taille des images non compressées

- Grille discrete, image N × M pixels
- A chaque pixel (m, n), on associe un ordre de traitement k
- Généralement, balayage ligne par ligne unilatéral :
  $k = (n - 1)M + m$
- On notera indifféremment $f_{n,m}$ ou f_k



*F*igure 2:représentation d'une image numérique

1.3.4.1 Comment compresser une Image ?

- Redondance statistique des données
  - Homogénéité des images
  - Similitude entre images successives
- Redondance psychovisuelle
  - Sensibilité aux baisses fréquences
  - Effets de masquage
  - Autres limites du système visuel humain
- Un algorithme de compression (ou codage) doit exploiter
- Au maximum la redondance des données

Les méthodes de réduction sont spatiales ou transformé avec leurs différentes propriétés (statistique ou psychovisuelle, réversible ou irréversible) ; actuellement la frontière entre les 2 méthodes est franchie par l'encodage de sous-bandes ou la compression d'images multi-résolutions.

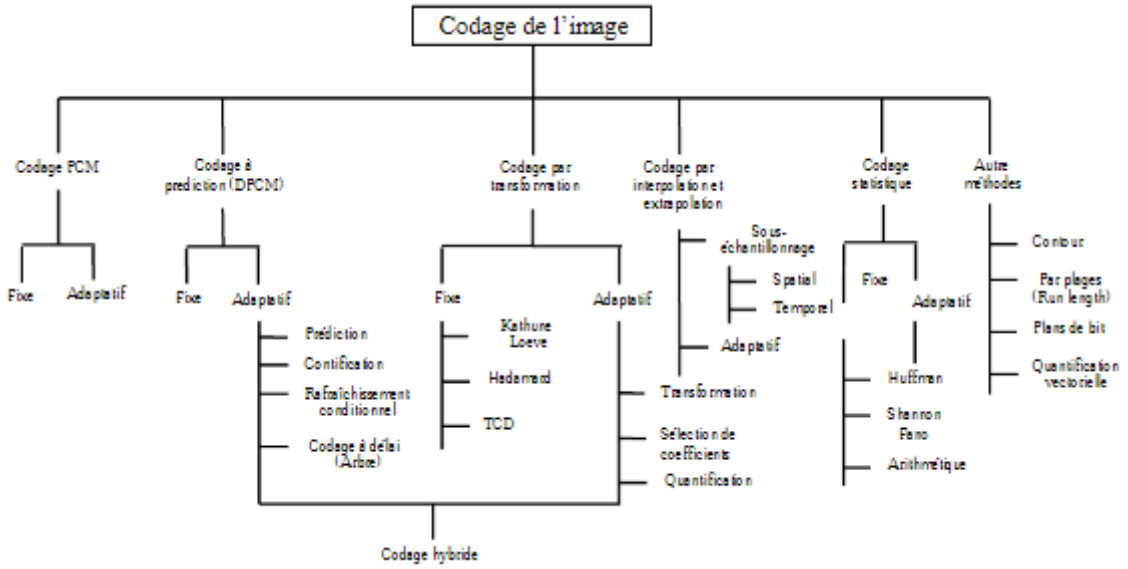


Figure 3:Classification des techniques de codage d'image à débit réduit.[3]

Les méthodes réversibles peuvent compresser les images sans perdre d'informations lors du décodage ; contrairement aux méthodes irréversibles (avec perte (dégradation), à réduction d'entropie) qui s'effectuent toujours sous la contrainte de qualité : (tableau ci-dessous).

Tableau 1:Relation entre le taux de compression et la qualité d'image

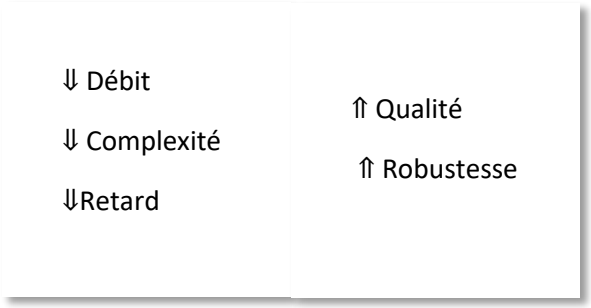
Taux de compression	Qualité de l'image
8	Image visuellement identique à l'originale
22	Image de qualité excellente
64	Image de qualité moyenne
200	Image reconnaissable

Il est intéressant de comparer ces exemples avec le cas de la compression sans perte, qui se traduit par une image reconstruite identique à l'original, mais au détriment d'un taux de compression de seulement 2 à 3.

1.3.4.2 Outils fondamentaux pour la compression :

- ✓ Codage entropique
- ✓ Quantification
- ✓ Transformée
- ✓ Prédiction

1.3.4.3 Critères de performance :



1.3.5 Compression de video :

Les vidéos numériques sont la représentation d'une scène visuelle échantillonnée sous forme numérique. Un flux vidéo numérique f (x, t) qui a été échantillonné temporellement est un groupe d'images qui évoluent dans le temps comme suit :

$$f(x, t) = \sum_{i=1}^{N} fi(x) 1_{[gi,gi+1]}(t)$$

En général, un flux vidéo à 10-20 pps (fps) est considéré comme une fréquence d'images faible, 25-30 pps est une fréquence d'images standard et 50-60 pps est une vidéo élevée. Le tableau suivant montre les différents formats vidéo et leurs résolutions.

Tableau 2:Formats de vidéo et leurs résolutions.

Format	Résolution	Nbr de pixels
Standard Definition (SD)	720×576	141720
720p High Definition(HD)	1280×720	921600
1080HD	1920×1080	2073600
UHD TV	3840×2160	8294400
2160p 4K UHD	4096×2160	8847360
8K UHD	7680×4320	33177600

Pour la compression de vidéo, la plupart des normes de codage vidéo populaires adoptent le codage vidéo hybride basé sur la prédiction de transformation cadre ; par exemple, MPEG-2, H.264/AVC et HEVC. Différent de JPEG, HEVC utilise plus d'intra modes de prédiction à partir de blocs reconstruits voisins dans le domaine spatial au lieu de la prédiction DC. Outre la prédiction intra, plus de gains de codage de la compression de vidéo proviennent de la prédiction inter efficace élevée, qui utilise l'estimation de mouvement pour trouver les blocs les plus similaires comme

prédiction pour le bloc à coder. De plus, HEVC adopte deux filtres de boucle, c'est-à-dire filtre de déblocage et SAO, pour réduire les artefacts de compression séquentiellement.

Les normes de codage image et vidéo basées sur les blocs, la compression est généralement dépendante des blocs et doit être exécuté bloc par bloc séquentiellement, ce qui limite la compression de parallélisme à l'aide d'une plate-forme de calcul parallèle, par exemple, GPU. De plus, la stratégie d'optimisation indépendante pour chaque outil de codage individuel limite également l'amélioration des performances de compression par rapport à la compression d'optimisation de bout en bout. En substance, il existe une autre trajectoire de développement technologique basée sur les techniques de réseau de neurones pour la compression d'images et de vidéos. Avec la résurgence du réseau de neurones.

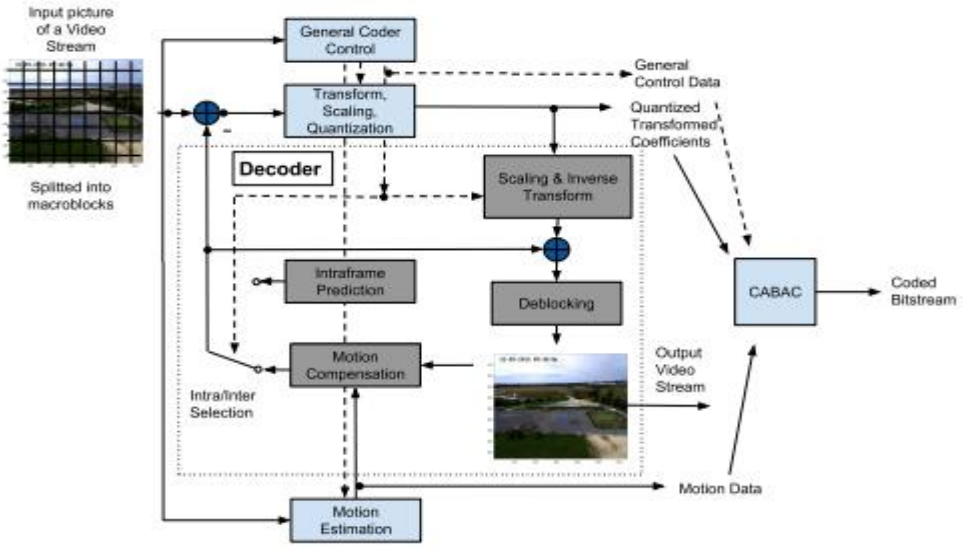


Figure 4 : techniques de réseau de neurones pour la compression d'images et de vidéos

1.3.6 Compression d'audio numérique :

La compression audio est conçue pour réduire la quantité de données qui stocke la musique. Deux formats de compression sont possibles : la compression sans perte et la compression avec perte. Ce dernier nous intéresse car il prend 10 fois moins de place que le premier.

La méthode de compression déprend intrinsèquement du type de données à compresser ; on ne compressera pas de la même façon une image qu'un fichier audio. On se servira d'une compression qui utilise un algorithme destructif pour ce qui concerne un fichier audio, contrairement à un fichier texte qui peut être compressé facilement car dans un fichier audio y en a beaucoup de répétions. Il existe aujourd'hui de nombreuses normes de compression avec perte. Les

plus courants sont MP3, VQF, OGG VORBIS. Mais il existe d'autres formats comme ATRAC, MP3Pro, AAC, WMA...La compression sans perte est Wave et AIFF.

La compression audio numérique utilise une variété de techniques psychoacoustiques afin de n'encoder que les informations utiles. L'une de ces techniques est le masquage. Le seuil auquel l'oreille humaine perçoit le son dépend largement de la fréquence du son. Par exemple, on perçoit plus facilement les sons calmes à 4 kHz qu'à 50 Hz ou 15 kHz. Aussi, à partir de 25 kHz, l'oreille humaine n'entend plus rien, quel que soit le niveau sonore.

Les algorithmes utilisés sont principalement le MPEG (pour le format MP3), l'AAC (MP3Pro), I'ATRAC (Sony Minidisc), le PASC (Philips DCC), et enfin les Dolby AC-1, AC-2 et AC-3.

Il existe également plusieurs formats de compression destructive. Il faut savoir que l'un d'entre eux, le fameux MP3, a été créé et breveté par Thomson Multimedia. En théorie également, toute personne qui crée un fichier MP3 à des fins d'exploitation commerciale devrait payer les droits à l'entreprise.

Un format imposé par Microsoft, Windows Media Audio (WMA), est similaire au MP3. Son format est également breveté, son utilisation est donc en échange des droits à payer à Microsoft.

D'autre part, le format OGG Vorbis résout ces problèmes de brevets. C'est un format "presque identique" au MP3, mais son utilisation est libre de droits. Mais ce format n'est pas aussi courant que MP3 ou WMA.

D'autres formats de compression destructeurs incluent MP3Pro, Real Audio de Real Networks, Yamaha VQF.

Les algorithmes utilisés sont principalement MPEG (pour le format MP3), AAC (MP3Pro), ATRAC (Sony Minidisc), PASC (Philips DCC) et enfin Dolby AC-1, AC-2 et AC-3.

Afin de définir différentes techniques de compression du son, je m'appuierai principalement sur le format MP3, ATRAC, les formats audios utilisent généralement la compression destructive, et donc les limitations psychoacoustiques de l'oreille humaine, pour supprimer certaines parties du son qui sont inaudibles.

1.3.7 Compression non destructive :

Il existe plusieurs algorithmes non destructifs (ZIP, RAR, ACE). D'autres se concentrent davantage sur le son. Notez WAVE (.wav) de Microsoft pour le PC et AIFF (.aif) d'Apple pour le macintosh. Il utilise le mode de compression PCM.

De nombreux autres algorithmes spécifiques à ces médias existent, bien qu'ils soient moins connus. Ceux-ci incluent WavArc (.arc) du format de compression ARC de Dennis Lee, AudioZIP (.zip) du ZIP de Lin Xiao, LPAC de Tilman Liebchen, Monkeys Audio de Matthew T. Ashland et RKAU de Malcolm Taylor.Sound Forge fournit le format propriétaire Perfect Clarrity Audio (son extension de fichier est PCA).

Cependant, malgré la compression, les fichiers audios sont encore trop volumineux pour être utilisés dans certaines applications : lors de l'utilisation de tels fichiers sur internet certes, mais aussi dans les canaux de transmissions numériques comme le câble télévisé

1.4 **Conclusion** :

Dans cette partie on a parlé en bref de deux types majeurs dans la compression des données qui sont les plus utilisables pour la majorité de types de fichiers tels que fichiers textes, audio , vidéos et images.

La compression de données est une opération informatique qui convertit une chaîne de bits A en une chaîne plus courte de bits B contenant les mêmes informations à l'aide d'un algorithme spécifique. La décompression est l'opération inverse de la compression.

Avec les algorithmes de compression sans perte, la séquence de bits résultante après les opérations de compression et de décompression est exactement la même que la séquence de bits d'origine. Les algorithmes de compression sans perte sont utilisés pour de nombreux types de données, y compris les documents, les exécutables et les fichiers texte.

Dans un algorithme de compression avec perte, les séquences de bits obtenues après les opérations de compression et de décompression sont différentes de l'original, mais les informations restent essentiellement les mêmes. Des algorithmes de compression avec perte sont utilisés pour les images, l'audio et la vidéo.

Dans le chapitre suivant on va parler de notre sujet principal de notre mémoire qui se représente à la compression de texte et les techniques disponibles pour effectuer cette opération informatique, et aussi parler des avantages et les inconvénients de chaque méthode et chaque algorithme.

II. Chapitre 02 : Généralité de compression de texte :

1. Introduction

Dans ce deuxième chapitre on va aborder les généralités de compression de texte, tout d'abord nous allons parler sur le type de compression responsable de compacter les textes, puis nous parlerons sur les techniques de compression tels que :codage de huffman, codage arithmétique, codage de dictionnaire.

2 Qu'est-ce qu'un texte :

L'origine du terme « Texte » est le mot latin « textum », du verbe « texere » qui veut dire « tisser ». Le mot s'utilise à l'entrelacement des fibres qui sont dans le tissage. Un texte est une chaine de mots orale ou écrite perçus comme constituant un ensemble cohérent, qui porte un sens et utilise les structures propres d'une langue (conjugaisons, construction et association des phrases…)

En Informatique :

En informatique, la notion de texte s'oppose à la notion de données binaires, donc c'est un :

- Texte brut dans un protocole de télécommunication.
- fichier texte si le programme utilisé permet de formater le texte.
- Texte formaté ou riche lorsque les signes de formats sont données en texte brut (exemple : Rich Texte Format).
- Document texte dans un fichier qui ne contient que des caractères sans mise en forme autre que les sauts de lignes et le choix des caractères.

Le code des caractères le plus utilisé récemment est le code ASCII1, qui crypte un symbole par un octet. Les valeurs de 0 à 127 pour représenter, principalement, les lettres non accentuées (majuscules et minuscules) ou meme les chiffres et les ponctuations sont utilisés par le code **ASCII** standard. Selon les machines et les besoins, pour représenter des caractères accentués (é, ù, ï, à, par exemple), des symboles mathématiques, des caractères étrangers (~n, ˚a, ¸s, par exemple), comme les caractères sérigraphiques ; les valeurs de 128 à 255 qui n'ont pas d'attribution dans le code ASCII standard sont utilisées,. Ainsi, un texte qui contient N symboles est représenté au moyen de N octets, soit $N * 8$ bits. [6]

3 Comment compresser un fichier texte ?

Les fichiers zippés (compressés) occupent moins d'espace de stockage et peuvent être transférés vers d'autres ordinateurs plus rapidement que les fichiers non compressés. Dans Windows, vous pouvez travailler avec des fichiers et des dossiers zippés de la même façon qu'avec des fichiers et dossiers non compressés. Vous pouvez également combiner plusieurs fichiers en un seul dossier zippé pour faciliter le partage d'un groupe de fichiers.

Il faut considérer les symboles un par un et :

– Réduire le nombre moyen de bits par symbole.

– ou considérer certaines suites de symboles, appelés facteurs, et les représenter par un nombre moyen de bits inferieur à huit fois la longueur du facteur considéré.

Remarquons que la première approche peut être considérée comme un cas particulier de la seconde avec des facteurs de longueur 1.

Le fonctionnement d'un algorithme de compression de texte peut se représenter par le schéma suivant :

Figure 5 : Modules d'un compresseur[6]

Le découpage consiste à définir quelle est l'unité de traitement : par exemple, Caractères, syllabes, mots du langage naturel. Nous utiliserons le terme symbole pour spécifier l'unité de traitement.

La factorisation a pour but, regrouper en facteurs des suites fréquentes de symboles, de donner une meilleure compression du texte.

Le codage a pour le rôle d'associer à chaque facteur une représentation binaire la plus courte possible. [6]

Les techniques de compression sans perte, comme leur nom l'indique, n'impliquent aucune perte d'informations. Si Les données ont été compressées sans perte, les données d'origine peuvent être récupérées exactement à partir de la donnée compressée. La compression sans perte est généralement utilisée pour les applications qui ne peuvent tolérer toute différence entre les données originales et reconstruites telle que les Images et les données textuelles.

La compression de texte est un domaine important pour la compression sans perte.

Il est très important que la reconstruction est identique au texte original, car de très petites différences peuvent entraîner des déclarations avec des significations très différentes, considérez les phrases " Ver vert dans un verre vert" et " Vert ver dans un verre vert" Un argument similaire vaut pour les fichiers informatiques et pour certains types de données telles que les relevés bancaires.

Si des données de quelque nature que ce soit doivent être traitées ou "améliorées" ultérieurement pour fournir plus d'informations, il est important que l'intégrité soit préservée. Par exemple, supposons que nous ayons compressé une radiologique image avec perte, et la différence entre la reconstruction et l'original était visuellement indétectable. Si cette image a été ultérieurement retouchée, l'image auparavant indétectable des différences peuvent entraîner l'apparition d'artefacts susceptibles d'induire sérieusement le radiologue en erreur. Parce que le prix de ce genre d'accident peut être une vie humaine, il est logique d'être très attention à l'utilisation d'un schéma de compression qui génère une reconstruction différente de l'original.

Les données obtenues à partir de satellites sont souvent traitées ultérieurement pour obtenir différentes valeurs numériques. Indicateurs de végétation, de déforestation, etc. Si les données reconstruites ne sont pas identiques aux données d'origine, le traitement peut entraîner une « amélioration » des différences. Ce n'est peut-être pas possible de revenir en arrière et d'obtenir à nouveau les mêmes données. Par conséquent, il n'est pas conseillé de permettre à toutes les différences d'apparaître dans le processus de compression. Il existe de nombreuses situations qui nécessitent une compression où nous voulons que la reconstruction être identique à l'original. Il existe également un certain nombre de situations dans lesquelles il est possible d’assouplir cette exigence afin d'obtenir plus de compression. Dans ces situations. [7]

4 Historique :

Fondamentalement, les données textuelles doivent avoir une compression sans perte où la perte d'informations n'est pas autorisée. Pour compresser les données textuelles, En 2005 Abel et Teahan ont développé une méthode de prétraitement universelle. Il intègre cinq algorithmes, notamment la conversion des lettres majuscules, le fin codage de ligne (EOL), remplacement de mots, remplacement de phrases et enregistrement des alphabets. Il n'a aucune dépendance aux langues et n'a pas besoin de dictionnaires. Mais, on constate que le coût de prétraitement s'avère élevé.

En 2012 il a eu une nouvelle compression technique basée sur une approche de programmation évolutive est développée c'est celle Ullah. Dans cet article, une approche évolutive est définie par extraire certaines fonctionnalités de la programmation génétique (GP) et Programmation génétique cartésienne (CGP). Le modèle compressif utilise un algorithme évolutif et le texte d'entrée est converti en nombre de morceaux en fonction du nombre de nœuds. Puis, l'évolution commence sur différentes mesures, y compris les taux de mutation, nombre de descendants et types de fonctions. Une fois les données textuelles sont compressées, elles seront envoyées au décodeur avec le génotype. Le génotype sert de clé et est très utile dans la pression de décompression. La méthode proposée est testée avec des données textuelles. La méthode proposée permet d'obtenir une meilleure compression lorsque le nombre de morceaux augmente.

En 2008, NonPlatoš et Al ont conçu une nouvelle méthode de compression basé sur la minimisation booléenne des données binaires avec BWT codage pour compresser des fichiers texte plus petits. Elle est utilisée dans diverses applications où des textes plus petits doivent être sauvegardé ou transmis comme Facebook, SMS, messagerie instantanée, etc. Les performances de cette méthode est testé à l'aide de fichiers texte moyens et petits et comparé avec des versions de codage Huffman, de codage LZW et BWT en termes de CR et CS. Et en 2015 Kalajdzic et Al ont présenté une technique efficace appelée B64pack pour les messages courts, c'est un algorithme efficace et léger, facile à déployer et interopérable.

Cette technique fonctionne en deux phases telles que la conversion des données d'origine vers un format compressible et transformation. Il s'avère efficace et plus rapide que d'autres méthodes telles que compresse, gzip et bzip2. Robert et Nadarajan (2009) ont introduit une méthode de prétraitement pour le codage de Huffman, le codage arithmétique, les versions de LZ et BWT codage. L'auteur utilise une transformation génétique réversible qui convertit un fichier texte dans d'autres formats avec une taille de fichier inférieure. Cela aide à faire progresser les performances de l'algorithme de compression backend. Une nouvelle technique DC est proposée pour compresser des données générales à l'aide d'une table de vérité logique (Mahmud, 2012) et donc deux bits des données sont représentées par un seul bit. En utilisant cette méthode, les données les bits peuvent être représentées uniquement par leurs demi-bits, c'est-à-dire que les données de 1 Go peuvent être représentées par 512 Mo.

Fondamentalement, il existe deux niveaux pour représenter les données numériques mais la méthode proposée utilise quatre niveaux et est validée par l'utilisation de CR et CF. De Agostino

(2015) a présenté une approche gourmande pour la compression de texte statique afin d'assouplir le préfixe propriété du dictionnaire. Il s'agit d'une machine à états finis (FSM) réalisation d'une compression basée sur la gourmandise avec un dictionnaire arbitraire pour atteindre une vitesse élevée dans les systèmes distribués. En 2015 dans Che et Al y a eu un développement d'un nouveau modèle de compression appelé Comp envoyé, pour la compression des phrases sentimentales en aspect analyse sentimentale. Comp envoyé a l'intention d'éliminer les informations indésirables par compresser les phrases de sentiment complexe à des sentiments courts et plus faciles à analyser. Oswald et Al en 2015 ont utilisé des outils d'exploration de données dans le domaine de la compression de texte. Le codage Huffman est amélioré par la combinaison de l'extraction fréquente d'éléments (FIM). Il est basé sur l'idée d'attribuer des mots de code plus courts à des modèles répétés.

En 2017 Oswald et Al ont utilisé une méthode basée sur des graphes pour extraire la séquence de caractères impliqués dans le processus de compression. Cette méthode exploite tous les modèles qui sont obligatoires pour la compression en une seule passe du graphe et elle construit un graphique en une seule passe du texte. 2017, Oswald et Sivaselvan ont introduit un autre roman basé sur le FIM Techniques de codage de Huffman utilisant une table de hachage (FPH2) pour compresser texte dans le processus de comptage de motifs fréquents. Un ensemble optimal de modèles est utilisé dans FPH2 tandis que l'approche basée sur les caractères est impliquée dans codage Huffman traditionnel. Cette méthode est testée sur un ensemble de 19 ensembles de données de référence et les résultats sont comparés avec gzip, LZW, LZSS et FP Huffman en termes de CR et CT.

5 Techniques de compression du texte :

Les techniques de compression sans perte, comme leur nom l'indique, n'impliquent aucune perte d'informations. Si Les données ont été compressées sans perte, les données d'origine peuvent être récupérées exactement à partir de la donnée compressée. La compression sans perte est généralement utilisée pour les applications qui ne peuvent tolérer toute différence entre les données originales et reconstruites.

La compression de texte est un domaine important pour la compression sans perte. Il est très important que la reconstruction est identique au texte original, car de très petites différences peuvent entraîner des déclarations avec des significations très différentes, considérez les phrases " Ver vert dans un verre vert" et " Vert ver dans un verre vert" Un argument similaire vaut pour les fichiers informatiques et pour certains types de données telles que les relevés bancaires.

Si des données de quelque nature que ce soit doivent être traitées ou "améliorées" ultérieurement pour fournir plus d'informations, il est important que l'intégrité soit préservée. Par exemple, supposons que nous ayons compressé un radiologique image avec perte, et la différence entre la reconstruction et l'original était visuellement indétectable. Si cette image a été ultérieurement retouchée, l'image auparavant indétectable des différences peuvent entraîner l'apparition d'artefacts susceptibles d'induire sérieusement le radiologue en erreur.

Parce que le prix de ce genre d'accident peut être une vie humaine, il est logique d'être très attention à l'utilisation d'un schéma de compression qui génère une reconstruction différente de l'original.

Les données obtenues à partir de satellites sont souvent traitées ultérieurement pour obtenir différentes valeurs numériques. Indicateurs de végétation, de déforestation, etc. Si les données reconstruites ne sont pas identiques aux données d'origine, le traitement peut entraîner une « amélioration » des différences. Ce n'est peut-être pas possible de revenir en arrière et d'obtenir à nouveau les mêmes données. Par conséquent, il n'est pas conseillé de permettre à toutes les différences d'apparaître dans le processus de compression. Il existe de nombreuses situations qui nécessitent une compression où nous voulons que la reconstruction être identique à l'original. Il existe également un certain nombre de situations dans lesquelles il est possible d’assouplir cette exigence afin d'obtenir plus de compression. Dans ces situations, nous cherchons à perdre techniques de compression

5.1 Codage de huffman :

Le codage de Huffman est la méthode de compression de données la plus connue et est couramment utilisé depuis plus de deux décennies. Cet encodage a été développé par David Huffman en 1952, alors qu'il était étudiant au MIT, où il suivait un cours dirigé par Robert Fano.

L'élaboration de ce code lui a également valu une dispense de l'examen final. Comme le code de Shannon-Fano, il s'agit d'un type d'encodage statistique qui utilise les fréquences des symboles qui composent le texte pour construire un code de préfixe. Il a engendré un certain nombre d'œuvres, dont celle de Robert Gallagher, qui a donné une version dynamique ou adaptative qui n'était plus nécessaire Traiter une seule lecture du texte. il n'est plus utilisé directement aujourd'hui texte, mais il est utilisé en combinaison avec d'autres algorithmes, basés sur le codage factoriel, par ex : gzip1 ou l'encodage basé sur des mots. [6]

5.1.1 L'algorithme Huffman statique :

L'algorithme de Huffman, exprimé graphiquement, prend en entrée une liste de valeurs non négatives. Pondère (wi, ..., w,) et construit un arbre binaire dont les feuilles sont étiquetées avec les poids. Lorsque l'algorithme de Huffman sert à construire un code, les poids représentent les probabilités associées aux lettres sources. Au départ, il existe un ensemble de arbres singleton, un pour chaque poids dans la liste. A chaque étape de l'algorithme, les arbres correspondant aux deux plus petits poids, Wi et Wj, sont fusionnés dans un nouvel arbre dont poids est Wi + Wj et dont la racine a deux les enfants qui sont les sous-arbres représentés par Wi et Wj. Les poids Wi et Wj sont supprimé de la liste, et wi + wj est inséré dans la liste. Ce processus continue jusqu'à ce que la liste de poids contienne une seule valeur. Si à tout moment il y a plus d'un chemin Choisissez une paire de poids plus petits, Une telle paire peut être choisie. Chez Hoffmann Papier Le processus commence par une liste de grammages non croissante. Cé détail n'est pas important pour la précision de l'algorithme, Mais il fournit une implémentation plus efficace [Huffman 1952]. Hoffmann L'algorithme est illustré à la figure6.

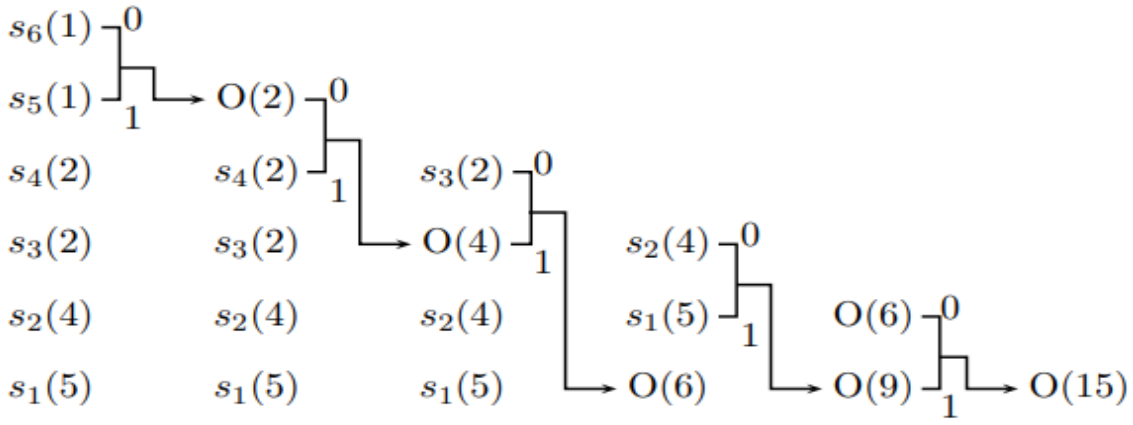


Figure 6 : Construction d'un code.[6]

Algorithme de Huffman pour déterminer longueur du mot de code à mapper Chaque lettre source ai. Avoir De nombreuses alternatives pour spécifier le réel données ; seul le code est requis Possède un attribut de préfixe. La tâche habituelle consiste à étiqueter chaque bord le parent de son enfant gauche avec les numéros 0 et Bord droit de l'enfant avec 1. De Le chiffrement de chaque lettre source est une séquence d'étiquettes le long du chemin Représentation racine des nœuds feuilles lettre.

Exemple :

Appliquons l'algorithme de Huffman `a la distribution de symboles suivante :

$S_1(5)$; $S_2(4)$; $S_3(2)$; $S_4(2)$; $S_5(1)$; $S_6(1)$;

Nous obtenons le processus suivant :

L'arbre est construit comme indiqué à la figure 7 et au code suivant :

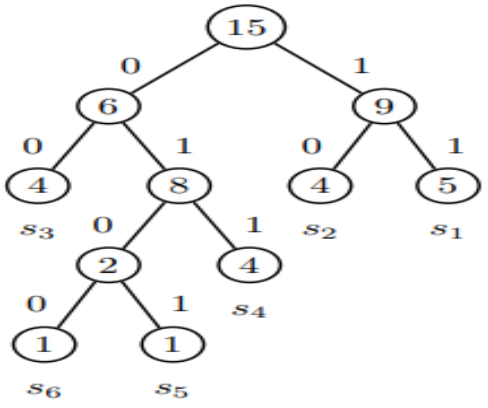


Figure 7 : Arbre de Huffman[6]

Tableau 3 : Code de Huffman

Symbole	Code	Longueur
S_1	11	2
S_2	10	2
S_3	00	2
S_4	011	3
S_5	0101	4
S_6	0100	4

5.1.2 L'algorithme de Huffman adaptatif :

L'algorithme de Huffman utilise deux lectures du texte traité, la première pour calculer les fréquences des symboles, puis leurs codes de Huffman, et la seconde pour remplacer chaque symbole par son code de Huffman.

N. Faller en 1973 et R.G. Gallager en 1978 ont proposé un algorithme adaptatif dans lequel les codes symboliques sont calculés en fonction de leur fréquence dans la portion de texte traitée. Ces algorithmes, mis au point par Knutth et Vitter, permettent de reconstruire une seule branche de l'arbre de Huffman.

Ces algorithmes permettent d'obtenir des taux de compression similaires à ceux obtenus par l'algorithme de Huffman statique. Ils constituent la base de la commande UNIX compact car l'algorithme statique de Huffman est implémenté dans la commande pack. Ils sont peu utilisés aujourd'hui, et ils sont plus populaires que les algorithmes adaptatifs basés sur le codage arithmétique. A noter que si la version adaptative peut lire plus que du texte, l'algorithme statique de Huffman préserve l'accès direct au texte en attribuant à chaque symbole un code unique.

5.2 Codage arithmétique :

Le codage arithmétique est un codage entropique utilisé en compression de données sans perte, est un codage statistique, c'est-à-dire que plus un caractère est représenté, moins il faudra de bits pour le coder. [8]

Il s'agit d'un cousin du codage de Huffman qui cependant reste toujours plus efficace que ce dernier (sauf dans le cas particulier où tous les poids des feuilles/nœuds/racines de l'arbre de Huffman sont des puissances de 2). Il est aussi plus simple à implémenter.

L'avantage que possède le codage arithmétique sur le codage de Huffman est que ce dernier va coder un caractère sur un nombre entier de bits (il ne peut coder sur 1.5 bits) là où le codage arithmétique le peut. Par exemple, si un caractère est représenté à 90%, la taille optimale du code du caractère serait de 0.15 bit, alors que Huffman coderait sûrement ce symbole sur 1 bit, soit 6 fois trop. [9]

L'intérêt principal du codage arithmétique est que chaque symbole peut être codé sur un nombre non entier de bits. Au lieu de coder les données source symbole par symbole, l'algorithme code en paquets de symboles, dont la longueur est définie par la capacité de l'ordinateur à coder des nombres réels plus ou moins grands.

Le codage arithmétique diffère des autres codages source en ce qu'il code le message Dans un paquet de n'importe quelle taille - en théorie - mais en pratique seuls des fragments d'environ 15 symboles peuvent être encodés en moyenne. En raison de l'algorithme de codage proche de l'espace, chaque paquet source est représenté par un nombre à virgule flottante, où Huffman code chaque symbole avec un nombre entier de bits. Le dernier problème avec le codage Huffman est que les caractères avec une forte probabilité d'occurrence seront codés en au moins un bit.

Un autre avantage du codage arithmétique est qu'il est un codage adaptatif pour le cas général d'une source avec mémoire. Plutôt que de supposer connue une fois pour toutes la distribution de probabilité conjointe de la source p(x1,x2,……,xk) ,il est possible d'estimer au fur et à mesure les probabilités conditionnelles p(xi,x1,x2,…,xk).il est noter que le codage arithmétique est optimal dans le cas général des sources avec mémoire.[8]

A. Fonctionnement du codage arithmétique

Le codage arithmétique est inclus avec le L'intervalle [0,1[. Cet intervalle diminue au fur et à mesure que le texte est traité.

La plage actuelle est divisée en autant de plages que de caractères potentiels représenter. La longueur de l'intervalle associé à un caractère est proportionnelle à sa probabilité, Chaque nouveau symbole traité réduit la taille de cet intervalle car le symbole est souvent. Étant donné que la division de l'intervalle peut être modifiée après le traitement de chaque caractère, le codage L'arithmétique est idéale pour exploiter les distributions de probabilités locales. [6]

B. Exemple :

Considérons l'alphabet A ={a,b,c}, dont les probabilités d'apparition des ´éléments sont :

P(a) = 0.6 , P(b) = 0.3 , P(c) = 0.1 .

Procédons au codage du texte T = abc

Le caractère a est représenté par l'intervalle [0,0.6[qui devient le nouvel intervalle de travail, puis on procède au codage du b dans ce nouvel intervalle.

Les ´étapes successives du codage sont illustrées par la figure8.

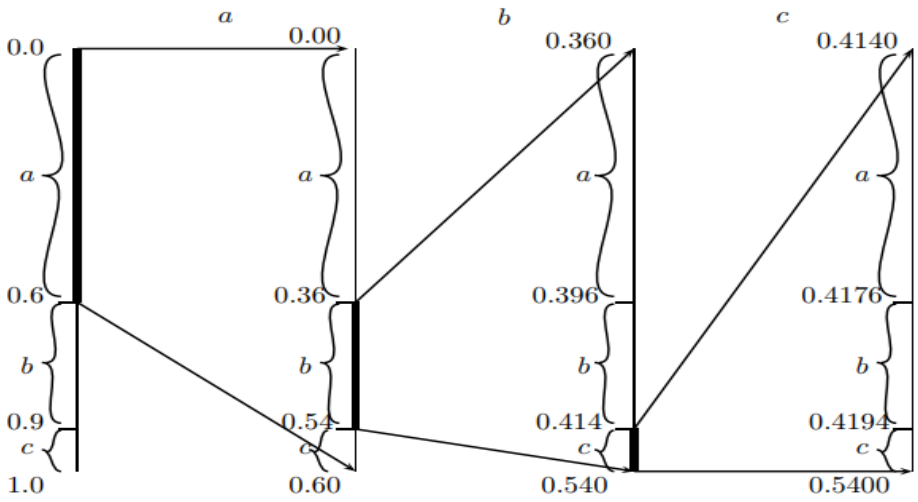


Figure 8 : Les étapes du codage arithmétique [6]

5.3 Dictionary Techniques:

5.3.1 Algorithmes de type LZ77 :

Cette série d'algorithmes n'utilise pas de dictionnaire, mais une fenêtre glissante sur le texte à traiter. Cette fenêtre contient le texte déjà encodé d'une part et le tampon de lecture anticipée d'autre part. L'encodage se fait par recherche dans une fenêtre, le facteur le plus long correspondant au facteur courant du tampon de prélecture. Alors ce facteur est représenté par sa position dans la fenêtre et sa longueur. [6]

a. Principe de fonctionnement de LZ77 :

Le LZ77 est divisé en deux parties utilisant une seule fenêtre. La partie droite, la taille est plus petite que la droite Le reste de la fenêtre s'appelle le tampon et contient les données à compresser. Partie gauche, Appelée fenêtre coulissante ou coulissante, elle peut être assimilée à un dictionnaire. Les octets sont La fenêtre est progressivement introduite à partir de la droite, c'est-à-dire amortir.

L'algorithme compare les données à compresser dans le tampon avec données dans le reste de la fenêtre. Pour ce faire, il évalue les données à compresser Commencez par le tampon de gauche à droite. [11]

Les séquences d'octets à compacter sont codées par 3 informations a, b et c. Si une de ces séquences se trouve déjà dans le reste de la fenêtre alors :

- A est la position de la séquence d'octets équivalente qui se trouve dans le reste de la fenêtre par rapport à la séquence d'octet compactée,
- B est la longueur de la séquence d'octets compactée,
- Celui qui diffère de la séquence d'octets compactée, est le premier octet du tampon qui signifie C

Si la séquence d'octets à compacter ne se trouve pas dans le reste de la fenêtre alors a=0 et b=0. La fenêtre se décale alors de b+1 octet(s) vers la gauche.

b. Exemple :

Message : ‘ Miss␣Mississippi’

Tableau 4 : Obtenir le codage de LZ77

Fenêtre coulissante	Tampon	Fichier compressé
M Mi Mis Miss␣ iss␣Miss Mississi Ississip	Miss iss␣ ss␣M s␣Mi Miss Issi Ppi Pi	(1, 0, M) (1, 0, i) (1, 0, s) (1, 1, ␣) (5, 3, s) (3, 3, i) (1, 0, p) (1, 1, i)

- Message d'origine :
  - 16 caractères (8 bits per symboles)
  - 128 bits (16*8bits)
- Représentation codée :
  - 8 triples (d, l , n)
  - 13 bits per triple (3+2+8 bits)
  - 104 bits (19%-bit savings)

5.3.2 Algorithmes de type LZ78 :

L'algorithme de pression Lempel-Ziv '78, est l'un des multitudes primordiaux algorithmes de diminution génériques. Il a sans tarder été adopté là-dedans de quelques logiciels commerciaux et libres, il est par exemple à la base du format ZIP. Il s'agit d'un algorithme basé sur un dictionnaire : les suites de symboles là-dedans la source sont encodées par un vocabulaire dynamique, qui est compilé au fur et à épaisseur que l'on parcourt la source. Fonctionnent sur une source, telle qu'un fichier, qui est une séquence (ou flux) de symboles appartenant à l'alphabet d'entrée. Ils produisent une sortie, produisant une séquence de symboles pour les lettres.

L'algorithme LZ78 fonctionne sur des sources codées dans des blocs de taille constante ; par exemple, un bloc par octet. Par conséquent, l'alphabet d'entrée se compose de tous les symboles pouvant être représentés sur le bloc. Si nous poursuivons notre exemple, l'alphabet des symboles pouvant être représentés sur un octet est constitué de valeurs comprises entre 0 et 255. Ainsi, dans notre exemple, lorsque nous encodons du texte, les symboles de l'alphabet représenteront un fichier de symboles ASCII, ou la profondeur de couleur (rouge, vert, bleu ou gris lors de l'encodage d'un bitmap avec une profondeur de 8 bits) degrés) (mais notez que l'algorithme LZ78 lui-même ne fait aucune différence entre les deux formats : tout ce que l'algorithme voit est un bloc de 8 bits).

L'alphabet de sortie de LZ78 se compose de paires (préfixe, symbole), où le préfixe est un entier et le symbole est un symbole de l'alphabet d'entrée. Si le codage du symbole est exactement le même que dans l'alphabet d'entrée, le codage du préfixe est de longueur variable, faisant de LZ78 un codage de longueur variable.

A. Principe de fonctionnement de LZ78 :

L’algorithme d’origine émet une suite de couple de la forme (indice, caractère). L’indice se réfère à un facteur déjà présent dans le dictionnaire, le caractère est le premier caractère ne faisant pas partie du facteur reconnu. Le dictionnaire est initialement vide.

Si W est un facteur déjà présent dans la table et a le caractère courant, l’algorithme de codage s’exprime ainsi :

- Si W a est dans le dictionnaire, on lit le caractère suivant avec W = W a
- Si W a n’est pas dans le dictionnaire, on transmet l’indice de W dans le dictionnaire suivi du caractère a. On ajoute W a au dictionnaire et l’on continue la lecture avec W= ε.

B. Exemple :

Considérons le mot u = abaaaabaab, la barre | indique où on en est dans le texte.

Texte	v	alpha	ajout au dico
(Init)			eps->0
|abaaaabaaa	eps	a	a->1
a|baaaabaaa	eps	b	b->2
ab|aaaabaaa	a	a	aa->3
abaa|aabaab	aa	b	aab->4
abaaaab|aab	aa	b	

Donc la sortie sera : (0, a), (0, b),(1, a),(3, b),(3, b).

Pour le d´décompression, il s'agit de reconstituer le dictionnaire exactement de la même façon que l'algorithme de compression. Cela ne présente pas de difficulté particulière.

5.3.3 Algorithmes de type LZW :

L'algorithme LZW vient des noms de ses inventeurs : Lempel, Ziv, et un peu plus tard Welsh, qui l'a amélioré il est actuellement utilisé principalement dans la compression d'images dans les formats existants les plus connus, c'est un algorithme libre de brevets depuis peu d'années pour cette raison sa diffusion est encore assez limitée par rapport aux algorithmes basés sur LZ77. [14]

Il est actuellement utilisé principalement dans la compression d'images dans les formats existants les plus connus, c'est un algorithme libre de brevets depuis peu d'années pour cette raison sa diffusion est encore assez limitée par rapport aux algorithmes basés sur LZ77.

Son principal avantage par rapport aux autres algorithmes est qu'il utilise un dictionnaire qui se construit au fur et à mesure qu'il lit un fichier/flux de données. [15]

La compression LZW fonctionne en lisant une séquence de symboles, en regroupant les symboles en chaînes et en convertissant les chaînes en codes. Comme les codes occupent moins d'espace que les chaînes qu'ils remplacent, nous obtenons une compression. Les caractéristiques de LZW comprennent,

• La compression LZW utilise une table de codes, avec 4096 comme choix courant pour le nombre d'entrées de table. Les codes 0-255 dans la table de codes sont toujours affectés uniques pour représenter des octets du fichier d'entrée.

• Lorsque l'encodage commence, la table de codes ne contient que les 256 premières entrées, le reste de la table étant vide. La compression est obtenue en utilisant les codes 256 à 4095 pour représenter des séquences d'octets.

• Au fur et à mesure que le codage se poursuit, LZW identifie les séquences répétées dans les données et les ajoute à la table de codes.

• Le décodage est réalisé en prenant chaque code du fichier compressé et en le traduisant dans la table de codes pour trouver le ou les caractères qu'il représente. [16]

Pour donner un exemple d'algorithme avec un dictionnaire, on peut imaginer compresser un SMS en remplaçant les mots les plus utilisés par des chiffres.
Par exemple, en utilisant 16 bits, nous pourrions avoir 65535 mots dans notre nouveau dictionnaire compressé, plus que suffisant pour composer un SMS avec un langage simple.

L'économie de cet algorithme imaginaire en termes de données serait considérable, en effet chaque mot ne pourrait occuper même que 2 symboles (en informatique chaque caractère de texte occupe 8bit), avec une telle compression on pourrait arriver à composer des SMS avec un bien supérieur nombre de mots par rapport à un SMS classique. [15]

5.4 La Transformée de Burrows-Wheeler (BWT) :

BWT est une compression sans perte, il utilise le contexte du symbole en cours d'encodage, inventée par Michael Burrows et David Wheeler, elle a été publiée en 1994, à la suite de travaux précédents de Wheeler en 1983, le transformée BWT est un mode de réorganisation des données et non de compression des donnée.

Il transforme une séquence de (n) symboles en une autre séquence de (n) symboles, La justification est que la nouvelle séquence est probablement plus adaptée à d'autres types de compression, tels que le Codage de Huffman et Run Length algorithm, Cette transformation est utilisée notamment dans la compression bzip2 ou en séquençage d'ADN.

L'algorithme peut se résumer comme suit, Soit une séquence de longueur N. on crée N -l autres séquences où chacune de ces N -I séquences est un décalage cyclique de la séquence d'origine, Ces N séquences sont rangées par ordre lexicographique. Le codeur transmet alors la

séquence de longueur N créée en prenant la dernière lettre de chaque triée, décalée cycliquement. Séquence. Cette séquence de dernières lettres L. et la position de la séquence d'origine dans la liste triée. Sont codés et envoyés au décodeur. Comme nous le verrons, Cette information est suffisante pour retrouver la séquence d'origine.

6 **Conclusion** :

La compression sans perte fonctionne de la même manière car elle est moins destructrice. Bien que la suppression des métadonnées soit irréversible, certaines compressions seront réversibles, ce qui en fera un algorithme flexible à de nombreuses fins.

Lossless est l'algorithme de compression de choix pour les fichiers texte et les arts visuels : photographie, conception graphique, art numérique, etc. La combinaison d'algorithmes sans perte avec une profondeur et une résolution appropriée donne une copie presque univoque.

Cependant, il convient de noter que la compression sans perte peut bien servir un domaine spécifique : la gamme d'applications est limitée. Cela réduit sa convivialité globale.

III. Chapitre 03 : La Transformée de Burrows-Wheeler (BWT) :

1. Introduction :

Notre société actuelle, entièrement tournée vers les technologies du numérique, impose de nombreux transferts d'informations. Nous avons donc étudié des techniques de compression de données, de type « sans perte », qui conservent l'intégralité de l'information présente dans un fichier, par le biais de transformations bijectives. Nous nous sommes intéressés à la transformée de Burrows-Wheeler, un algorithme de « précompression », qui modifie une chaîne afin d'améliorer l'efficacité d'autres algorithmes. Notre objectif était donc d'implémenter rigoureusement et efficacement cette transformée, puis de programmer d'autres algorithmes de compression courants qui peuvent lui être appliqués, et analyser son impact en comparant divers critères. **[18]**

2. Définition de la Transformée de Burrows-Wheeler :

Le BWT est un outil très puissant et même les algorithmes les plus simples qui l'utilisent ont étonnamment bonnes performances (le lecteur peut regarder l'algorithme très simple et propre basé sur BWT décrit dans [Nelson 1996] qui surpasse, en termes de taux de compression, le package commercial Pkzip). Suite les compresseurs avancés basés sur BWT, tels que Bzip [Seward 1997] et Szip [Schindler 1997], sont parmi les meilleurs compresseurs actuellement disponibles. Comme on peut le voir à partir des résultats rapportés dans [Arnold et Bell ; Fenwick 1996a] Les compresseurs basés sur BWT atteignent un très bon taux de compression en utilisant des ressources (temps et espace). Considérant que les compresseurs basés sur BWT en sont encore à leurs balbutiements, nous pensons que dans un avenir proche, ils deviendront probablement la nouvelle norme en matière de compression de données sans perte.

Une autre propriété remarquable du BWT est qu'il peut être utilisé pour construire une structure de données qui est une sorte de tableau de suffixes compressés pour la chaîne d'entrée s [Ferragina and Manzini 2000 ; Ferragina et Manzini 2001]. Une telle structure de données consiste en une version compressée de bwt(s) plus o(|s|) bits d'informations auxiliaires. En utilisant cette structure de données, nous pouvons calculer le nombre d'occurrences d'un motif arbitraire p dans s dans O(|p|) temps, et nous pouvons calculer la position en s de chacune de ces occurrences en O(log|s|) temps.

Malgré sa pertinence pratique, aucune analyse théorique satisfaisante du BWT n'est disponible. Bien qu'il est facile de comprendre intuitivement pourquoi le BWT aide à la compression, aucun des algorithmes basés sur le BWT actuellement utilisés n'a été analysé théoriquement. En d'autres termes, ces algorithmes fonctionnent très bien dans pratique, mais aucune preuve n'a été donnée que leur taux de compression est, disons, dans un facteur constant de l'entropie d'ordre zéro de l'entrée.

Dans [Sadakane 1997 ; Sadakane 1998] Sadakane a proposé et analysé trois algorithmes différents basé sur le BWT. En supposant que la chaîne d'entrée est générée par une source de Markov d'ordre fini, il a prouvé que le taux de compression moyen de ces algorithmes se rapproche de l'entropie de la source. Plus récemment, Effros [Effros 1999] a considéré des algorithmes similaires et a donné des bornes sur la vitesse à laquelle le taux de compression moyen se rapproche de l'entropie.

Bien que ces résultats fournissent des informations utiles sur la BWT, elles, ne sont pas totalement satisfaisantes pour plusieurs raisons. Premièrement, ces résultats traitent des algorithmes qui ne sont pas réalistes (et en fait ne sont pas utilisés dans la pratique). Par exemple, certains de ces algorithmes nécessitent la connaissance de grandeurs habituellement inconnues comme l'ordre de la source de Markov ou le nombre d'états dans la source ergodique. Deuxièmement, aucune de ces analyses ne traite de la durée d'exécution l'encodage qui, comme nous le verrons, est une technique fréquemment utilisée en relation avec le BWT. Finalement, le l'hypothèse que l'entrée provient d'une source de Markov d'ordre fini n'est pas toujours réaliste, et les résultats basés sur cette hypothèse ne sont valables qu'en moyenne et non dans le pire des cas. **[19]**

3. Genèse de la transformation Burrows-Wheeler :

Sans doute la dernière percée du 20e siècle en matière d'utilisation sans perte à usage général méthodes de compression était la transformée énigmatique de Burrows et Wheeler, la BWT. David Wheeler avait imaginé la transformation dès 1978, mais ce n'est qu'en 1994 que, avec l'aide de Mike Burrows, l'idée a été transformée en une méthode pratique de compression de données, qui a ensuite été publiée dans un rapport de recherche du Digital Systems Research Center (Palo Alto) (Burrows et Wheeler, 1994). Leur "code de tri par blocs", également surnommé "Burrows Wheeler Transform", a laissé les praticiens de la compression se gratter la tête, car il s'agissait de réorganiser les caractères d'un texte avant l'encodage, puis les arrangeant comme par magie dans leur ordre d'origine dans le décodeur.

Le fait que l'original puisse être recréé est quelque peu étonnant, et leurs premiers travaux ont mis du temps à recevoir la reconnaissance qu'ils méritaient. Dans quelques années, plusieurs auteurs et programmeurs avaient repris l'idée, apparemment principalement à travers les publications de Peter Fenwick (Fenwick, 1995b, c,1996a, b) qui a conduit à l'implémentation de bzip par Julian seward. À peu près au même moment, il y avait un article de Mark Nelson dans le journal du Dr Dobb (Nelson,1996), et le BWT est également apparu par le biais de canaux informels tels que groupes de discussion. **[20]**

Burrows et Wheeler ont d'autres réalisations importantes dans le domaine de l'informatique. David Wheeler (1927-2004) a eu une brillante carrière, ayant a travaillé sur plusieurs premiers ordinateurs, y compris EDSAC qui, en 1949, est devenu le premier ordinateur à programme stocké à être achevé. Wheeler a inventé une méthode d'appel de sous-programmes fermés qui a conduit à avoir une bibliothèque de soigneusement sous-programmes testés, un concept qui a été crucial pour décomposer la complexité en programmation informatique. Avec Maurice Wilkes et Stanley Gill, en 1951, il publie le premier livre sur la programmation informatique numérique2.

Il a également effectué d'importants travaux en cryptographie, notamment le "Tiny Encryption Algorithm » (TEA), un système de cryptage qui pourrait être écrit en seulement huit lignes de code, qui tournaient en dérision les réglementations américaines qui contrôlaient exportation d'algorithmes de chiffrement — celui-ci était assez petit pour être mémorisé ! Wheeler a également conçu et commandé la première version du Cambridge Ring, un système de réseau local expérimental basé sur une topologie en anneau.

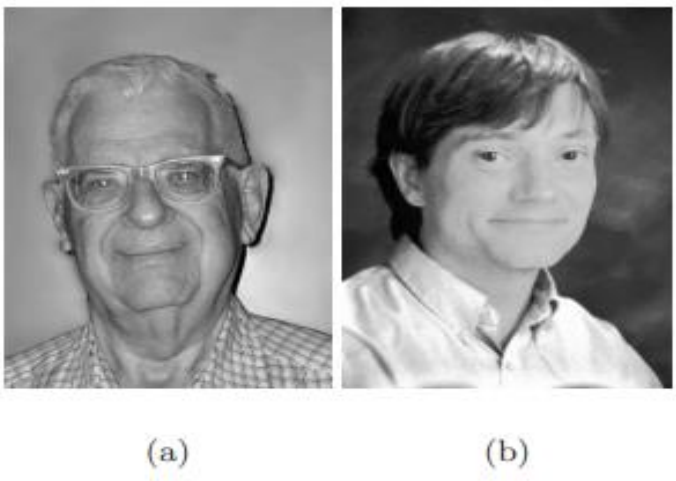


Figure 9:(a) David Wheeler (b) Michael Burrows[20]

Ses travaux sur la compression se sont développés pendant qu'il était consultant en recherche aux Bell Labs (Murray Hill, N.J.) en 1978 et 1983. Il a pris sa retraite en 1994. (La même année que l'article fondateur de BWT a été publié). Ses distinctions notamment être membre de la Royal Society (1981) et membre de l'ACM (1994).

Michael Burrows a également une grande notoriété en dehors de sa contribution à la BWT. Il est l'une des principales personnes qui ont développé le moteur de recherche AltaVista en 1995, qui représentait l'état de l'art avant l'arrivée de Le moteur de recherche de Google. Il a ensuite travaillé pour Microsoft et, en 2007, il est senior chercheur travaillant chez Google sur leur infrastructure distribuée. Les terriers avaient été supervisé par Wheeler au milieu des années 1980 en train de faire un doctorat à Cambridge, et puis est allé travailler chez Digital aux États-Unis. Wheeler avait inventé la transformation en les années 1970, mais ce n'est que lorsqu'il a visité Digital à Palo Alto, puis a travaillé à distance avec Burrows par e-mail en 1990 qu'il a finalement été écrit comme une méthode de compression.

À la fin des années 1990, BWT était encore considéré comme trop lent pour de nombreuses applications, mais ses performances de compression sont devenues bien comprises. Wheeler's Les programmes « bred » (block reduce) et « bexp » (block expand) ont fourni une implémentation publiquement disponible de la méthode BWT qui a prouvé le concept, mais c'est la mise en œuvre efficace de Julian Seward en tant qu'utilitaire à usage général appelé bzip en 1996 qui a établi BWT comme quelque chose qui avait des avantages pratiques utilitaire. Une nouvelle version de l'utilitaire de Seward appelée bzip2 est maintenant largement utilisée car sur le matériel d'aujourd'hui, elle peut compresser des fichiers volumineux à des vitesses assez acceptable pour l'interaction, à une taille plus petite que d'autres général largement utilisé méthodes de finalité.

Par exemple, le fichier « bible.txt » de 4 Mo du corpus Canterbury peut être compressé par bzip2 en 2 secondes environ sur une bande 2,4 GHz ordinateur, et décompressé en 1 seconde environ,L'utilitaire gzip compresse environ trois fois plus rapide(et décompresse un ordre de grandeur plus rapidement), mais le fichier gzip est 40% plus grand que celui de bzip2. Fait intéressant, bzip2 combine l'une des percées les plus récentes en matière de compression (BWT) avec l'une dès le premier (codage de Huffman).

À la fin des années 1990, les chercheurs ont commencé à réaliser que l'approche BWT peut être utile pour plus que la simple compression de texte. Parce que le BWT arrive à "trier" le texte dans l'ordre alphabétique, le texte permuté a l’avantage supplémentaire d'agir comme une sorte de dictionnaire pour le texte original. Traditionnellement, un index et le texte compressé seraient stockés séparément, même bien qu'ils contiennent effectivement les mêmes informations. Dans

cette optique, le BWT est une représentation intermédiaire à mi-chemin entre un texte et un index ; le texte original peut être reconstruit efficacement à partir de celui-ci, mais des listes triées comme celui illustré à la figure 1.1b sont mûrs pour la recherche binaire, donnant très rapidement rechercher des fragments arbitraires dans le texte. **[20]**

4. Exemple pour le BWT :

Dans cette section, nous donnerons un exemple simple de la façon dont un texte est transformé et reconstruit à l'aide du BWT. La méthode décrite ici est pour plus de clarté d'explication.

On traitons l'exemple « BLAN-MENA » :

On crée la liste de toutes les rotations :	On les trie par ordre lexicographique :	On ajoute la chaîne initiale et on note sa position :
ABLAN-MEN	-MENABLAN	-MENABLA**N**
NABLAN-ME	ABLAN-MEN	ABLAN-ME**N**
ENABLAN-M	AN-MENABL	AN-MENAB**L**
MENABLAN-	ENABLAN-M	BLAN-MEN**A** (4)
-MENABLAN	LAN-MENAB	ENABLAN-**M**
N-MENABLA	MENABLAN-	LAN-MENA**B**
AN-MENABL	N-MENABLA	MENABLAN-
LAN-MENAB	NABLAN-ME	N-MENABL**A**
		NABLAN-M**E**

Enfin, on extrait le dernier caractère de chaque rotation triée, et on adjoint la position de la chaîne initiale. On obtient ainsi la chaîne : 4NNLAMB-AE.

La transformation est aussi simple que cela ; en fait, dans la pratique, c'est encore plus simple, car les sous-chaînes ne sont jamais créées, mais sont simplement stockées en tant que références à des positions dans la chaîne d'origine. La taille du texte transformé est identique à l'original, et contient exactement les mêmes caractères mais dans un ordre différent. Cela pourrait semblent n'avoir abouti à rien, mais comme nous le verrons, cela rend le texte beaucoup plus plus facile à compresser car il a rassemblé des caractères qui se produisent dans des contextes liés — c'est-à-dire des caractères qui précèdent les mêmes sous-chaînes.

Il peut sembler que décoder le texte converti serait très difficile ; mais il existe un nombre exponentiel de façons de le faire ?

Pour le décodage : on part de la chaîne « 4NNLAMB-AE »

On trie les couples (caractère, position de ce caractère dans la chaîne) par ordre lexicographique, comme ceci : (- ,7) ; (A,4) ; (A,8) ; (B,6) ; (E,9) ; (L,3) ; (M,5) ; (N,1) ; (N,2)

Puis on débute à la position donnée (4) et on parcourt d'indice en indice : Le couple 4 est (B, 6), il pointe vers le couple 6 (L, 3), qui pointe vers le couple 3 (A, 8), et ainsi de suite jusqu'à retomber sur la position initiale, 4. (4) B →(6) L →(3) A →(8) N →(1) - →(7) M →(5) E →(9) N →(2) A →(4) stop On renvoie bien la chaîne initiale : BLAN-MENA.

5. Comment fonctionne la transformation de Burrows-Wheeler

a) La transformée de Burrows-Wheeler avant :

La transformation directe consiste essentiellement à trier toutes les rotations de l'entrée string, qui regroupe des caractères qui se produisent dans des contextes similaires. La figure 10 (a) montre les rotations A qui se produiraient si la transformée était donnée T= mississippi en entrée1, et la figure 10 (b) montre le résultat du tri A, que nous appellerons As.

mississippi	imississipp
ississippim	ippimississ
ssissippimi	issippimiss
sissippimis	ississippim
issippimiss	mississippi
ssippimissi	pimississip
sippimissis	ppimississi
ippimississ	sippimissis
ppimississi	sissippimis
pimississip	ssippimissi
imississipp	ssissippimi
(a)	(b)

Figure 10 : (a) Le tableau A contenant toutes les rotations du mississippi d'entrée ; (b) obtenu en triant A. La dernière colonne de As (généralement appelée L) est la transformée de Burrows-Wheeler de l'entrée.[20]

Cependant, plutôt que d'utiliser l'espace O(n2) comme suggéré par la figure 10, nous pouvons créer un tableau R [1 . . . n] de références aux chaînes pivotées dans l'entrée texte T. Initialement, R[i] est simplement défini sur i pour chaque i de 1 à n, comme indiqué dans Figure 11(a), pour représenter la liste non triée. Il est ensuite trié à l'aide de la sous-chaîne commençant à T[R[i]] comme clé de comparaison. La figure 11(b) montre le résultat de tri ; par exemple, la position 11 est la première chaîne pivotée dans l'ordre lexical (imiss...), suivi de la position 8 (ippim...) et de la position 5 (issip...); la chaîne de référence finale est R = [11, 8, 5, 2, 1, 10, 9, 7, 4, 6, 3].

Le tableau R indexe directement les caractères de T correspondant aux premières colonnes de As, désignée par F dans la littérature BWT. La dernière colonne de As (appelée L) est la sortie du BWT, et peut être lue comme T[R[i]−1], où i varie de 1 à n (si l'indice de T est 0 alors il fait référence à T[n]). dans ce cas le texte transformé est L = pssmipissii. Nous devons également transmettre un indice a pour indiquer au décodeur quelle position dans L correspond au dernier caractère du texte d'origine (c'est-à-dire quelle ligne de As contient l'original chaîne T). Dans ce cas, l'indice a = 5 est inclus.

Dans la description ci-dessus, la transformation est effectuée en utilisant uniquement l'espace O(n) (pour R). Le temps pris est O(n) pour la création du tableau R, plus le temps nécessaire au tri. Conventionnellement, le tri est considéré comme prenant O (n log n) temps moyen si une méthode standard telle que le tri rapide est utilisée. Cependant, certains les séquences de chaînes peuvent provoquer un comportement proche du pire des cas dans certaines versions du tri rapide, en particulier s'il y a beaucoup de répétitions dans la chaîne et le pivot pour quicksort n'est pas sélectionné avec soin. Cela correspond au traditionnel O(n2) pire cas de tri rapide où les données sont déjà triées - si T contient long du même caractère, le tableau A contiendra de longues séquences triées.

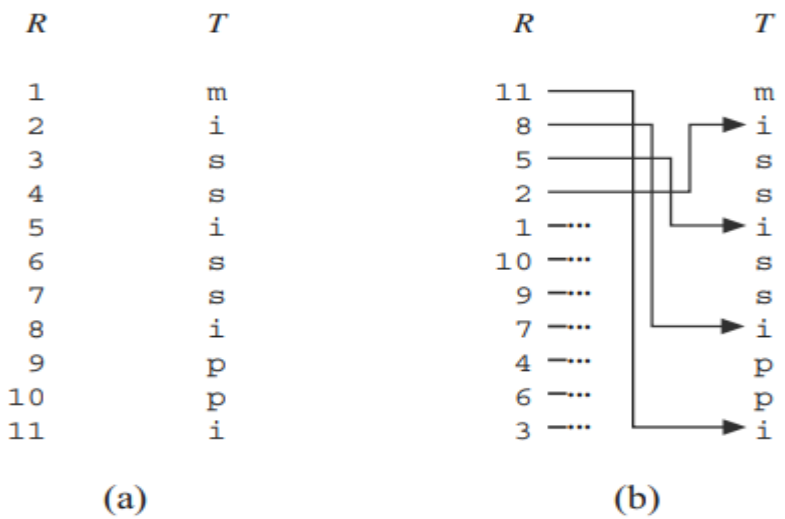


Figure 11:: Le tableau R utilisé pour trier le fichier d'exemple mississippi[20]

Par exemple, la figure 12 montre le tableau A pour l'entrée aaaaaab. C'est déjà trié en raison de la façon dont le b termine la longue séquence de caractères a. Il est possible d'éviter ce pire comportement dans le tri rapide avec des techniques comme la sélection médiane de trois partitions, mais la nature du BWT signifie que des méthodes de tri encore meilleures sont possibles.

Non seulement la liste pré-triée peut entraîner de mauvaises performances dans certaines versions de tri rapide, mais les longs préfixes presque identiques signifient que les comparaisons lexicales nécessitera de nombreuses comparaisons de caractères, ce qui signifie que le temps constant l'hypothèse pour les comparaisons n'est pas valide ; si tous les caractères sont identiques alors cela pourrait prendre O(n) temps pour chacune des O(n2) comparaisons, ce qui serait extrêmement lent, d'autant plus que pour un tel cas, le BWT implique pas de permutations du tout. De longues chaînes répétées peuvent apparaître dans la pratique dans les images qui contiennent de nombreux pixels de la même couleur (comme un scan d'une page en noir et blanc avec peu d'écriture dessus) et dans les données génomiques

où l'alphabet est très petit et les sous-chaînes répétées sont courantes.

aaaaaab
aaaaaba
aaaabaa
aaabaaa
aabaaaa
abaaaaa
baaaaaa

(A)

Figure 12:Le tableau A contenant toutes les rotations de l'entrée aaaaaab.

Il existe plusieurs façons d'éviter ce problème. Burrows et Wheeler ont observé dans leur article original qu'en ayant un caractère sentinelle unique, le problème de tri équivaut à trier tous les suffixes de T, ce qui peut être fait dans le temps et l'espace linéaires à l'aide d'un arbre de suffixes, mais il faut mentionner que le principal inconvénient de cette approche est que bien que l'encombrement soit O(n), le constant facteur peut être important.

Au lieu de cela, Burrows et Wheeler ont proposé une version modifiée de quicksort qui applique un tri par base aux deux premiers caractères de chaque clé de tri. Chaque des seaux à deux caractères doivent maintenant être triés, mais une attention particulière est payée aux compartiments

où les deux premiers caractères sont identiques, car ces sont susceptibles d'indiquer de longues séries du même caractère (généralement nul ou espace caractères), ce qui peut prendre beaucoup de temps pour faire une comparaison lexicale pour tri basé sur la comparaison, mais sont triviaux à trier en raison de la façon dont ils ont été généré. Finalement, le tri rapide n'est appliqué qu'aux groupes de sous-chaînes qui besoin de tri dans des seaux. Par exemple, les chaînes de la figure 12 seraient divisé en trois compartiments pour ceux commençant respectivement par aa, ab et bb. Le tri rapide n'est pas automatiquement appliqué au bucket aa, car les deux premiers caractères sont les mêmes, et en effet dans ce cas le seau arrive à déjà trier, et entraînerait de longues comparaisons entre les chaînes car des préfixes longs des suites de la lettre a.

Une autre approche consiste à éliminer ce problème en codant de longues séries de même caractère en utilisant une technique de codage de longueur de plage, où des séries de répétitions les caractères sont remplacés par un code plus court. Cela peut même parfois avoir un effet positif sur la quantité de compression, bien que le but principal soit d'éviter la mauvaise vitesse de tri qui se produit dans les cas particuliers décrits ci-dessus en éliminant les longues séries du même caractère. Un inconvénient de ceci est que le texte d'origine n'est plus disponible directement dans le BWT, ce qui peut affecter certaines des méthodes de recherche de domaine compressé décrites plus loin dans ce livre. De plus, l'encodage de longueur d'exécution modifiera les informations de contexte qui le BWT utilise, donc l'effet sur la compression n'est pas nécessairement positif.[20]

Un problème inévitable avec le BWT est qu'il nécessite un gros bloc de mémoire pour stocker la chaîne d'entrée (T) et l'index des chaînes en cours trié (R). Si le bloc est trop petit, la compression sera mauvaise, mais si trop grand, il peut utiliser trop de mémoire. Même si la mémoire est disponible, il peut poser des problèmes de mise en cache, et il y a des avantages en termes de performances à conserver blocs dans la taille d'un cache, pas seulement dans la mémoire principale. Le motif d'accès à la mémoire sera aléatoire du fait des opérations de tri cela doit être fait (le même problème se produit également lors du décodage). Sur ordinateurs modernes, il peut y avoir plusieurs couches de mise en cache qui essaieront de deviner les modèles d'accès à la mémoire, et ceux-ci peuvent avoir des interactions complexes avec les accès nécessaires au BWT. Cette préoccupation doit être prise en compte considération lors du choix de la taille du bloc ; s'il tient dans le cache (et pas seulement dans la mémoire principale), il pourrait bien être capable de fonctionner plus rapidement. Sur le d'autre part, à mesure que les machines parallèles avec mémoire embarquée deviennent plus populaires,la méthode BWT peut potentiellement être adaptée pour tirer parti de ce type de architecture, et il est même possible qu'elle ait des avantages de performance dans un environnement parallèle par rapport aux autres méthodes de compression populaires.

L'actuel les performances en pratique dépendront de l'architecture de la machine, de la quantité de mémoire disponible et la conception de tous les caches.

L'annexe B répertorie les sites Web qui fournissent une variété de code source pour exécuter le BWT. Certains sont adaptés pour expérimenter la transformation et tracer le processus, tandis que d'autres sont des systèmes de production qui ont optimisé les détails du codage pour bien fonctionner dans la pratique.

b) La transformée inverse de Burrows-Wheeler :

La transformation inverse prenant un texte permuté BWT et reconstruisant l'entrée d'origine T est un peu plus difficile à mettre en œuvre que la transformation vers l'avant, mais cela peut toujours être fait dans le temps et l'espace O (n) si les soins sont pris. L'exemple de la figure 11 reconstruit le tableau As, mais comme pour l'encodage, il n'est en pratique pas nécessaire de stocker ce tableau O(n2). En général deux tableaux d'indices O(n) seront nécessaires, plus deux tableaux O(|Σ|) pour compter les caractères dans l'entrée. Il existe plusieurs façons de décoder. Le papier original de Burrows et Wheeler produit la sortie à l'envers, bien qu'il ne soit pas difficile de produire la sortie dans l'ordre d'origine. On le fera montrer comment générer des structures de données pour ces deux cas.

	F										*L*
1	i	m	i	s	s	i	s	s	i	p	p
2	i	p	p	i	m	i	s	s	i	s	s
3	i	s	s	i	p	p	i	m	i	s	s
4	i	s	s	i	s	s	i	p	p	i	m
5	m	i	s	s	i	s	s	i	p	p	i
6	p	i	m	i	s	s	i	s	s	i	p
7	p	p	i	m	i	s	s	i	s	s	i
8	s	i	p	p	i	m	i	s	s	i	s
9	s	i	s	s	i	p	p	i	m	i	s
10	s	s	i	p	p	i	m	i	s	s	i
11	s	s	i	s	s	i	p	p	i	m	i

Figure 13: Le tableau Quant au Mississipi ; F et L sont les première et dernière colonne [20]

Nous utiliserons le décodage de la chaîne mississippi comme exemple courant. La figure 13 montre le tableau Comme pour cet exemple, avec les colonnes F et L étiquetées. Comme cela n'est pas stocké explicitement dans la pratique, mais nous l'utiliserons entre-temps pour illustrer comment le décodage peut être fait. Le décodeur peut déterminer F simplement en triant L, puisqu'il contient exactement les mêmes caractères, juste dans un format différent order, chaque colonne de As contient le même jeu de caractères car les lignes sont toutes les rotations de la chaîne d'origine. En fait, F n'a pas besoin d'être stocké, car il peut être généré implicitement en comptant la fréquence d'apparition de chaque caractère dans L.

Regarder As nous aide à voir les informations nécessaires pour effectuer le décodage. Étant donné uniquement F et L, l'étape clé consiste à déterminer quel caractère doit venir après un caractère particulier dans F. Considérez, par exemple, les deux lignes se terminant par un p (lignes 1 et 6). A cause de la rotation, l'ordre de ces deux lignes est déterminé par les caractères qui viennent après l'occurrence respective de p dans T (imi... et pim... respectivement). Ainsi la première occurrence de p dans L correspond à la première occurrence de p dans F, et de même avec la deuxième occurrence. Cela nous permet de travailler sur le texte à l'envers : si on vient de décoder le deuxième p de L, alors il doit correspondre à celui de la ligne 7 de F. En regardant la ligne 7, la colonne L nous indique que le p était précédé d'un i. À son tour, parce que c'est le deuxième i de L, il doit correspondent au deuxième i de F, qui est au rang 2. Nous continuons à parcourir les tableaux L et F de cette manière jusqu'à ce que toute la chaîne soit décodée en sens inverse.[20]

La correspondance aurait également pu être utilisée pour décoder la chaîne dans sa commande d'origine. Par exemple, en regardant le p dans L[6], nous pouvons déterminer qu'il est suivi de F[6], un p. Comme c'est le premier p de F, il correspond au premier p dans L, c'est-à-dire la ligne 1. Ce p est suivi d'un i, et ainsi de suite. C'est marginalement plus simple de décoder la chaîne dans l'ordre inverse, donc généralement la littérature BWT utilise le décodage vers l'arrière.

Un moyen facile de suivre les relations ci-dessus est de numéroter les apparitions des caractères dans F et L. La figure 14 montre les colonnes F et L de Figure 13, mais nous avons numéroté les occurrences de chaque caractère dans l'ordre du premier au dernier en utilisant des indices. Cela rend la chaîne décodée facile à lire à l'arrêt ; par exemple, la quatrième ligne a L [4] = m1, et le F [4] correspondant nous dit qu'il est suivi de i4. Comme i4 est dans L [11], on peut obtenir le suivant caractère de F [11], qui est s4. La chaîne entière est décodée dans l'ordre : m1 i4 s4 s2 i3 s3 s1 i2 p2 p1 i1.

En pratique, le décodeur ne reconstruit jamais As ou F en entier, mais implicitement crée des index pour représenter suffisamment de sa structure pour décoder l'original chaîne de caractères. L est stocké explicitement (le décodeur lit simplement l'entrée et la stocke dans L), mais F est stocké implicitement pour économiser de l'espace et fournir efficacement le genre d'informations nécessaires lors du décodage.

	F	L
1	$\mathtt{i}_1$	$\mathtt{p}_1$
2	$\mathtt{i}_2$	$\mathtt{s}_1$
3	$\mathtt{i}_3$	$\mathtt{s}_2$
4	$\mathtt{i}_4$	$\mathtt{m}_1$
5	$\mathtt{m}_1$	$\mathtt{i}_1$
6	$\mathtt{p}_1$	$\mathtt{p}_2$
7	$\mathtt{p}_2$	$\mathtt{i}_2$
8	$\mathtt{s}_1$	$\mathtt{s}_3$
9	$\mathtt{s}_2$	$\mathtt{s}_4$
10	$\mathtt{s}_3$	$\mathtt{i}_3$
11	$\mathtt{s}_4$	$\mathtt{i}_4$

Figure 14:Utilisation de l'ordre des caractères pour effectuer la transformation inverse[20]

La figure 15 montre trois tableaux auxiliaires utiles pour le décodage. K[c] est simplement un décompte du nombre de fois que chaque caractère c apparaît dans F, ce qui se détermine facilement en comptant les caractères dans L. M[c] localise la première position du caractère c dans le tableau F, donc K et M stockent efficacement ensemble l'information dans F. C[i] stocke le nombre de fois que le caractère L[i] apparaît en L plus tôt que la position i ; par exemple, le dernier caractère de L'est i, et i apparaît 3 fois dans la partie précédente de L. Ces trois tableaux facilitent la traverser l'entrée en sens inverse.

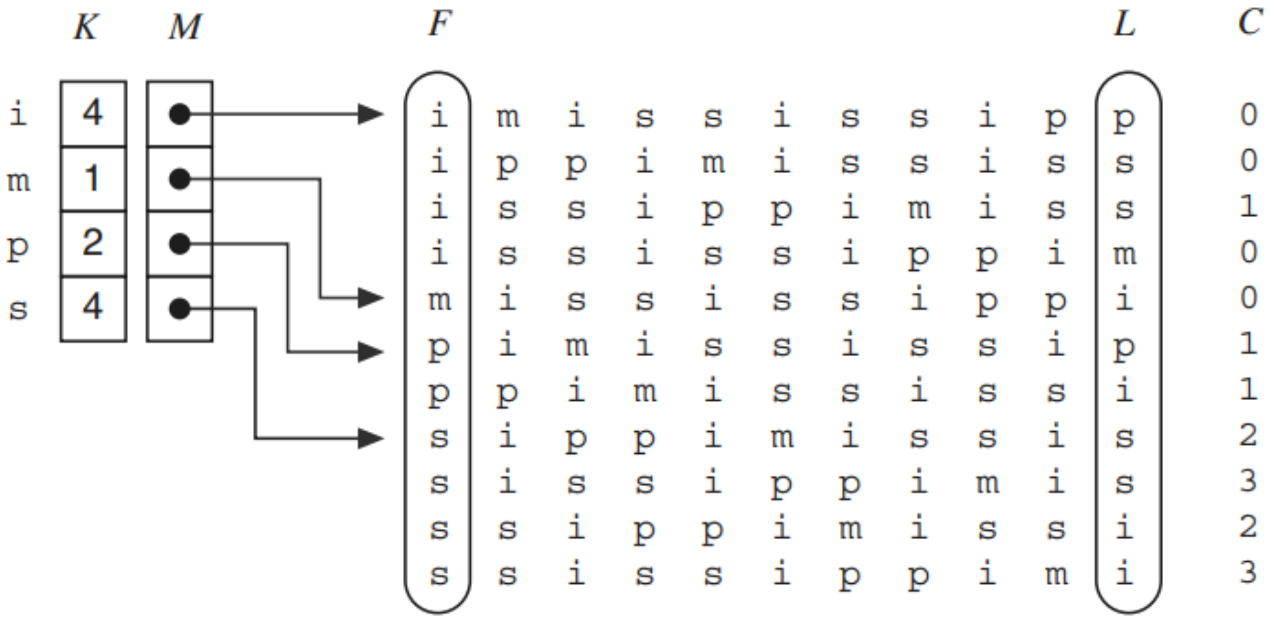


Figure 15:Le tableau (As) qui est reconstruit implicitement pour décoder la chaîne[20]

6 Conclusion :

Les transformées de Burrows-Wheeler sont généralement efficaces sur du texte (syllabes souvent récurrentes), et permettent d'implémenter et d'améliorer (parfois assez significativement) le taux de compression de l'algorithme.

Par contre, le principal inconvénient reste le temps de calcul. Burrows et Wheeler ont rencontré cette difficulté et sont très préjudiciables à une utilisation généralisée. Comme mentionné en annexe, la conversion prend quelques minutes pour des fichiers d'une taille moyenne de 20 Mo.

Une utilisation qui peut être recommandée est d'essayer toutes les combinaisons possibles d'algorithmes et de choisir le plus efficace pour chaque fichier à compresser. En fait, l'étape BW domine largement, et le temps de calcul des autres combinaisons est négligeable.

IV. Chapitre 04 : Résultats et Simulation

1. Introduction :

Dans cette dernière partie de notre thèse , on va essayer de présenter une comparaison entre deux techniques de compression en basant par deux différents codages ou algorithmes , qui sont La Transformation de Burrows Wheeler (BWT) avec Run Lenght (RLE) et l'algorithme de Run Lenght , ensuite , on va aussi présenter les avantages et les inconvénients de chaque algorithme et pour obtenir a la fin des résultats concrètes et réelles , pour déduire l'effet de l'application de BWT pour savoir la meilleure pour faire une compression pour un fichier texte .

2. Simulation et résultats :

On va prendre un exemple pour faire notre étude sur la comparaison entre les deux algorithmes, les exemples sont suivants :

1) Exemple n°1 :

a=

' Plutôt que de se promener sur la rive et regarder le poisson d'un œil d'envie, mieux vaut rentrer chez soi et tisser un filet '

length(a) :125

A. Compression avec Run Lenght (RLE):

Run-Length Encoding c'est le codage par longueur de plage c'est un algorithme de compression de texte qui se repose sur l'idée de signaler le nombre de répétions d'un caractère.

X=

'Plutôt que de se promener sur la rive et regarder le poi2son d'un œil d'envie, mieux vaut rentrer chez soi et ti2ser un filet'

length(x):

```
ans =

   126
```

Transformation de Burrows Wheeler (BWT) avec Run Lenght (RLE) :

a=

‘ Plutôt que de se promener sur la rive et regarder le poisson d'un œil d'envie, mieux vaut rentrer chez soi et tisser un filet ’

length(a) :125

Transformation de Burrows Wheeler (BWT):

C’est un algorithme qu’il s’agit pas de compression mais il consiste à réorganiser une chaine de caractères dans un texte pour faciliter la tache de l’un des codage de compression .et pour plus d’information veuillez voir le chapitre 3.

a=

‘trlneeinrr,eetttaezrtrxnddelgv ruvlsdirmr'rdnsl ih ecovm“ftori iPo ouueeesprs eueeea t p s sii e´eue nuq 'sale inueÅÃt H’

B. Compression de run Lenght et (BWT) :

X=

trln2ein2r,2e3taezrtrxn2delgv4 ruvlsdirmr'rdnsl2 ih ecovm“ftori2 iPo o2u3esprs3 eu3ea2 t p s s2i e´eue nuq 'sale inueÅÃt H'

length(x) :

```
ans =

   123
```

A. Taux de compression :

Taux de compression = τ = 1 - ([Volume final] / [Volume initial])

On a T(A) = 0 et T(B) = 0.1

2) Exemple n°2 :

a=

'LLLLLLLLLLLLhLLLLLLtLLLLLLLhLLLgLLLLLLLLjLLLLLhdLLLgLLL
LoLLNLLLhNLLLLLLLLLtLLhLLLLLLLLgLLLLLLNLLdLLLhLL '

Length(a)

```
ans =

   103
```

A. Compression avec Run Lenght (RLE):

x=

'12Lh6Lt7Lh3Lg8Lj5Lhd3Lg4Lo2LN3LhN9Lt2Lh8Lg6LN2Ld3LhL'

Length(x)

```
ans =

    53
```

Transformation de Burrows Wheeler (BWT) avec Run Lenght (RLE) :

a=

'LLLLLLLLLLLLhLLLLLLtLLLLLLLhLLLgLLLLLLLLjLLLLLhdLLLgLLL
LoLLNLLLhNLLLLLLLLLtLLhLLLLLLLLgLLLLLLNLLdLLLhLL '

Length(a)

```
ans =

   103
```

Transformation de Burrows Wheeler (BWT):

a=

'hLLLLLNhLgLLLtLLgLLLLhLLLLLjLLLLLLLLLgLLLhLddLLNLLLLLoL
NLLLLtLLLLLLLLLLLLLLLLLLLLLLLLLhLLhLLLLLLLLLLLLL '

Compression de run Lenght et (BWT) :

x=

' h5LNhLg3Lt2Lg4Lh5Lj9Lg3LhL2d2LN5LoLN4Lt24Lh2LhL'

Length(x)

```
ans =

    48
```

Taux de compression :

Taux de compression = τ = 1 - ([Volume final] / [Volume initial])

On a T(A) = 0.48 et T(B) = 0.53

Remarque :

Dans les exemples suivants nous avons essayé de compliquer un peu la tâche pour mieux expliquer notre thèse et notre méthode de compression, alors nous avons pris deux images au hasard (comme est montré dans les exemples 3 et 4) et nous avons fait un déchiffrage binaire de 2 images et après nous avons appliqué les 2 méthodes de compression et nous avons eu les résultats en dessous :

Exemple n°3 :

Figure 16

Décryptage binaire de fig16 :

00000000000000000000000000000000000000000

00000000000000000000000000000000000000000

00000000000000000000000000000000000000000

00001111111111111111111111110000000000000

11111111111111111111111111111111111000000

11111111111111111111111111111111111111110

11111111111111111111111111111111111111111

11111111111111111111111111111111111111111

111111111111111111111111111111111111111

111111111111111111111111111111111111111

111111111111111111111111111111111111111

111111111111111111111111111111111111111

111111111111111111111111111111111111110

111111111111111111111111001111111101010

111111111111111111111111111111111111011

111111111111111111111111111111111111110

111111111111111111111111111111111111100

111111111100111111011111111011110111111

111111111100111111011111111101111111111

111111111111111000011111111111111111111

111111111111111000001111111001111111110

111111111111111000000111111001111111110

111111111111111000001111111111111111111

111111111111111110011111111111111111111

111111111111111111111111111111111111111

111111111111111111111111111111111111111

111111111111111110011111111111111111111

A. Compression avec RLE :

124023113034160391027910251208101010371041103812011120610810410171206109102614035150812091015160712091016150371201171201

Length(x)

ans =

121

B. Compression avec RLE BWT :

1111011201201012015019021205150415012010312021302101017108107810481201810221031202012010271059101710151047102410251091070108101610210107105101501016210

Length(x)

ans =

152

Taux de compression :

Taux de compression = τ = 1 - ([Volume final] / [Volume initial])

On a T(A) = 0.85 et T(B) = 0.88

Exemple n°4 :

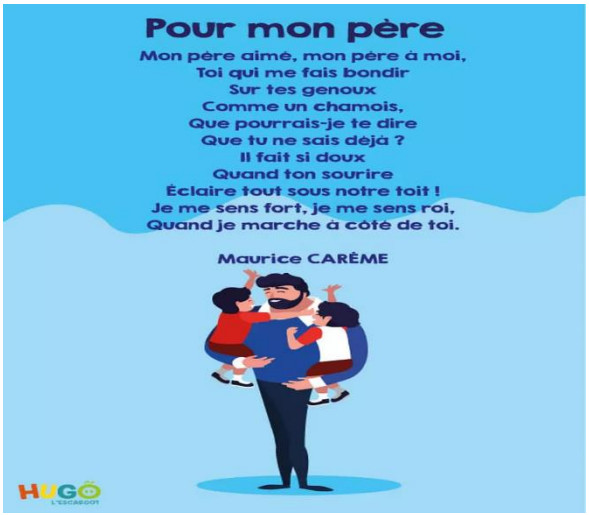


Figure 17

Décryptage binaire de fig 17

11111111111111111111111111

11111110000100001000011111111

11111111001110000000101111111

1111111111111111111111111111

1111111111111111111111111111

1111111111101010100111111111

1111111111101010011111111111

1111111111111111111111111111

1111111111111111111111111111

1111111110000100110111111111

1111111111011010101111111111

1111111111111111111111111111

1111111111111111111111111111

1111111110001010011111111111

1111111100010111011011111111

1111111111111111111111111111

1111111111111111111111111111

1111111011111111111111111111

1111111111111111111111111111

1111111111111111111111111111

1111111111111111111111111111

1111111111110011111111111111

1111111111001101111111111111

1111111111001010111111111111

1111111110001100111111111111

1111111110000010111111111111

1111111110000011111111111111

1111111110000011111111111111

1111111111110000011111111111

1111111111110010111111111111

1111111111110001111111111111

1111111111110001111111111111

1111111111111001111111111111

1111111111111001111111111111

1111111111111001111111111111

1111111111111001111111111111

1111111111111001111111111111

1111111111111001111111111111

1111111111111111111111111111

1111111111111111111111111111

A. Compression avec RLE :

3514014014015120316010741010101202010101207614012021019102101010761301012018130103102107110116120241202102312010102213021202115010211502315025150231201024130251302612026120261202612026120261201

Length(x)

ans =

194

B. Compression avec RLE BWT :

3106120160212014021401201203103120130713031031012012010120101201202101011002109101071021208150312021012010120181091021302102510313107110401022103120261016105120151010101031014160262101410111031204010

Length(x)

ans =

200

Taux de compression :

Taux de compression = $\tau = 1$ - ([Volume final] / [Volume initial])

On a T(A) = 0.826 et T(B) = 0.821

3. Conclusion :

A partir des résultats des quatre exemples, nous concluons que dans la première méthode celle de Run LEngth (A) la compression est moins efficace par rapport à la deuxième méthode celle de BWT avec RLE la méthode (B). Et ça après la comparaison des deux taux de compression.

Quelques remarques concernant les 2 algorithmes :

- Les algorithmes utilisés ont tous une complexité spatiale linéaire en la taille de la chaîne.
- Les algorithmes hors BWT sont en général assez rapides, de l'ordre de quelques dizaines de secondes pour des fichiers de grande taille, à savoir plusieurs méga-octets.
- L'avantage majeur du BWT étendu semble être dans son utilisation d'une plus petite empreinte mémoire, et une étape de tri potentiellement plus rapide en moyenne, puisque les matrices de rotation BWT complètes ne seront pas nécessaires.
- La chose remarquable à propos du BWT n'est pas qu'il génère une sortie plus facilement encodée - un tri ordinaire ferait cela - mais qu'il le fait de manière réversible, permettant au document d'origine d'être régénéré à partir des données de la dernière colonne.
- L'un des inconvénients c'est qu'elle n'est pas toujours efficace puisqu'elle dépend de texte qu'elle traite, par ex : quand un message contient de différents caractères et moins de répétition.
- L'algorithme BWT est lent en cas d'un message long.
- Il est bien connu que le codage Run LEngth (RLE) est une bonne technique pour compresser les données où la même séquence est apparue à plusieurs reprises.
- Un autre avantage du codage Run LEngth (RLE) est qu'il peut être appliqué à chaque dimension de données séparément.
- Lorsque vous utilisez la méthode Run Length (RLE) pour compresser un ensemble de données, sa taille peut augmenter après la compression si les données ne contiennent pas de nombreuses parties répétées.
- Nous n'appliquons la compression que dans le cas où les données peuvent être compressées efficacement.

Conclusion générale :

La compression de données c'est une branche importante dans l'informatique qui nous permis de gagner le stockage et la rapidité de transmission des données. Dans le premier chapitre, nous avons parlé de la compression des données et ses 2 grandes et importantes parties qui sont présentées sous les titres suivants la compression sans perte et la compression avec perte. Et ce dernier il représente la compression de vidéo, audio et la compression d'image.

Dans ce deuxième chapitre qui représente la partie principale de notre mémoire , nous avons pu se détailler dans l'idée de compression de texte, en abordant les différentes techniques de compression de texte tels que Huffman et ses codages statique et adaptatif, arithmétique, et technique de dictionnaire et ses codages LZ77, LZ78 et LZW .et à la fin de chapitre nous avons parlé brièvement de la technique de la Transformation de Burrows Wheeler en préparation au chapitre suivant.

L'algorithme de Transformation de Burrows Wheeler c'est une technique qui facilite la tâche de compression en basant par un codage spécial par la réordination de la chaine de caractère.

Nous avons au long de la dernière partie de notre mémoire présenter et comparer entre 2 techniques de compression de texte (BWT-RLE) et (RLE). La comparaison entre les 2 algorithme à base de la longueur d'un message et son taux de compression (espace et temps) pour chaque codage. Et à la fin nous avons obtenu 2 différents résultats qui se présente dans une amélioration sensible dans le taux de compression.

ANNEXE :

CODE COMPLET

Voici le code de notre programme, en langage MATLAB**.**

Run LEngth Algorithme:

```
clear all
clc
a=input('inter the message','s')
x=' ';
c=a(1);
r=1;
for n=2:length(a)
    if c==a(n)
        r=r+1;
    elseif r==1
   x=[x,c];
   c=a(n);
        r=1;
  else
        x=[x,num2str(r),c];
        c=a(n);
        r=1;
    end


    if n==length (a)
       x=[x,c];
    end

end

x
```

La transformation de Burrows-Wheeler :

```
clear all;
clc;
path=pwd;
[filename, pathname] = uigetfile('*.txt', 'Pick a file');
cd(pathname)
file_open=fopen(filename,'r');
file_read=fread(file_open,'uint8');
fclose(file_open);
cd(path)
disp('precessing..BWT Transform');
a=file_read;
b=zeros(1,2*length(a));
for sort_len=1:length(b)
    if(sort_len>length(a))
    b(sort_len)=a(sort_len-length(a));
else
    b(sort_len)=a(sort_len);
end
end
a=char(a);
b=char(b);
to_sort=zeros(length(a),length(a));
for row_sort=1:length(a)
    to_sort(row_sort,:)=b(row_sort:length(a)+row_sort-1);
end
char(to_sort);
[lexi_sorted_data,ind]=sortrows(to_sort);
char(lexi_sorted_data);
encoded_data=lexi_sorted_data(:,length(a));
primary_index=find(ind==2);
out_data=[encoded_data',primary_index];
file_bwt = fopen('bwt.cmp','w');
fwrite(file_bwt,out_data,'uint8');
fclose(file_bwt);
disp('BWT Transform over');
disp('file written to bwt.cmp');
```

Acronymes

DC : Compression de Données

Mo: Miga Octet

Ko : kilo octet

MPEG : Moving Pictures Experts Group

HEVC : High Efficiency Video Coding

SAO : sample-adaptive-offset

GPU : Graphics Processing Unit

GIF: Graphic Interchange Forma

PNG : Portable Network Graphics

AAC : Advanced Audio Coding

ADPCM: Adaptive Differential Pulse Code Modulation

VQF: Facteur de qualité de voix

OGG : est le nom du principal projet de la fondation Xiph.org dont le but est de proposer à la communauté des formats et codecs multimédias ouverts, libres et dégagés de tout brevet.

VORBIS : est un algorithme de compression et de décompression (codec) audio numérique, sans brevet, ouvert et libre, plus performant sur le plan de la qualité et du taux de compression que le format MP3, mais moins populaire que ce dernier.

WMA : Windows Media Audio

AIFF : Audio Interchange File Format.

AC : Audio Codec

PASC : problèmes d'affectation sous contraintes

DCC : Data Communications Channel

PC : personal computer

PCM : Pulse Code Modulation

RAR : Roshal ARchive

ASCII : American Standard Code for Information Interchange

LZW : Lempel-Ziv-Welch

ACM: Association for Computing Machinery

BWT : La transformée de Burrows-Wheeler

Bibliographie :

[1] : wikilivres /article/compression de donné.

[2] : Frédéric Dufaux Département Traitement du Signal et des Images TELECOM ParisTech 26 janvier 2011.

[3] : Techniques de compression d'images April 2004 Aoued Boukelif University of Sidi-Bel-Abbes.

[4] : Synthèse et étude comparative sur les méthodes de compression vidéo.Réalisé par :Mr. Selmi Mohammed El-Amin Mr. Yala Mohammed.

[5]: Image and Video Compression with Neural Networks: A Review
10 Apr 2019.

[6] : Thèse de Doctorat / Université de Marne-la-Vallée par Claude Martineau.

[7]: livre: Introduction to Data Compression / Khalid Sayood University of Nebraska.

[8]: https://123dok.net/article/codage-arithm

[9] : http://igm.univ-mlv.fr/article/ La compression de données

[10] : http://iphome.hhi.de/schwarz/assets/dc/07-DictionaryCoding.pdf

[11] : https://atika.cohen.web.ulb.be/cours de Compression Numérique

[12] : https://compression.fiches-horaires.net/la-compression-sans-perte/lz78-et-lzw-la-Compression-par-dictionnaire

[13] : coure Compression (M1) – LZ78 & LZW C. Nicau

[14] : boowiki /article/ les algorithmes de compression/lempel-ziv-welch

[15] : : www.oldwildweb.com/article/Implémentation de LZW en C++ par Ugo Cirmignani / Ugo Cirmignani

[16] : Article : Technique de compression LZW /jack sparrow/5/7/2022

[17] :https://www2.ulb.ac.be/cours/acohen/travaux_2006_infodoc/CompressionNumerique/SansPerteLZW.htm#LZ78

[18] : BLANCHÉ Alexandre La Transformée de Burrows-Wheeler université de Lyon

[19] : Giovanni Manzini/ Giovanni Manzini Département d'informatique, Université du Piémont oriental,

[20] : livre: THE BURROWS-WHEELER TRANSFORM: Data Compression, Suffix Arrays, and Pattern Matching -Donald Adjeroh / Amar Mukherjee / Tim Bell-.

[21] :BURROWS Michael, WHEELER David John, A Block Sorting Lossless Data Compression Algorithm