**MASTER EN SCIENCES INFORMATIQUES**

# Techniques de réduction de données

Chevalier, Olivier

*Award date:*
2021

*Awarding institution:*
Universite de Namur

Université de Namur

Faculté d'informatique

Année académique 2020–2021

# Techniques de réduction de données

CHEVALIER Olivier

Promoteur : CLEVE Anthony


Mémoire présenté en vue de l'obtention du grade de
Master en Sciences Informatiques.

# Remerciements

Je tiens à remercier en premier lieu, mon promoteur, le professeur Anthony Cleve pour sa disponibilité, son suivi et ses conseils tout au long de l'élaboration de ce mémoire.

Mes remerciements vont également à mes collègues de classe et en particulier à Caroline Deremiens, Audrey Gilson et Thomas Dardenne qui ont été d'un précieux soutien tout au long de ces années d'études.

Merci également à ma compagne, à mes parents et à toutes les personnes qui m'ont soutenu et encouragé durant ces 4 années.

Mes remerciements s'adressent aussi à tout le personnel de la faculté informatique de l'université de Namur, que ce soit les professeurs, les assistants ainsi que le personnel encadrant, pour les connaissances transmises, leur disponibilité et leur bienveillance.

Enfin, je remercie les membres du jury pour avoir bien voulu examiner et juger ce travail.

# Résumé

Le volume de données numériques créées croît de manière exponentielle dû, entre autres, à la démocratisation des objets connectés.// Afin de stocker ces données, il est nécessaire d'utiliser d'énormes infrastructures de stockage. Toutefois, l'augmentation des capacités de stockage croît moins vite que le volume de données à stocker.

Ce mémoire a pour but de déterminer les différentes techniques de réduction de données existantes afin d'économiser de l'espace de stockage.
Il existe, à ce jour, 2 techniques de réduction de données : la compression et la déduplication.

La compression est une technique populaire qui permet de réduire la quantité de données pour un fichier tandis que la déduplication permet de réduire la quantité de données en éliminant les éléments redondants.

Ce travail décrit et illustre le fonctionnement de ces 2 techniques et propose une comparaison entre elles en matière de similitudes, de différences et de complémentarités.

# Table des matières

# Chapitre 1

# Introduction

Le monde de l'informatique vit, depuis quelques années, une révolution dans la manière de stocker toutes les données que ce soit au travers du disque dur ou au travers du Cloud.

Le Cloud a changé considérablement la manière d'utiliser les ressources informatiques.
En effet, l'espace de stockage est généralement loué le temps de son utilisation contre une compensation financière qui a permis aux grands fournisseurs (comme Google, Amazon, Apple, Microsoft ou Dropbox) de fournir une meilleure maîtrise des coûts d'une entreprise ainsi qu'une certaine flexibilité pour accéder aux données depuis n'importe quel lieu.

Cette révolution permet aux sociétés qui le désirent de ne plus investir dans de grosses infrastructures, mais de payer seulement pour ce dont elles ont besoin. Quant aux particuliers, ils délaissent de plus en plus les disques durs externes au profit du stockage de données dans le Cloud.

Ce changement permet de bénéficier d'un espace de stockage en fonction des besoins ainsi que de garantir davantage la récupération des données (un disque dur externe peut se perdre ou devenir défectueux). Toutefois, il existe une certaine méfiance envers le Cloud et les fuites de données...

Pour limiter la quantité de données et diminuer les coûts liés au stockage, les particuliers, les entreprises ou encore les fournisseurs de service de stockage dans le Cloud, se doivent d'appliquer des méthodes de réduction de volume de données. Parmi ces méthodes, on retrouve la compression et la déduplication de données.

## 1.1 Évolution du stockage de données

Les données sont des informations essentielles, présentes dans tout système intelligent et renfermant toutes nos informations médicales, scientifiques, techniques, administratives, etc.

En informatique, toutes les informations sont codées en binaire.
Le système binaire est un système de codage utilisant la base 2 avec toutes données exprimées sous forme de 0 et de 1.

Un groupe de 8 bits forme un octet et peut disposer de 256 valeurs ($2^8$).
Un Ko (kilo-octet) est défini comme un groupe de $2^{10}$ représentant 1024 octets soit $2^8 * 2^{10} = 262\ 144$ valeurs.
Le Mo équivaut à 1024 Ko, le Go (giga-octet) équivaut à 1024 Mo, le To (téra-octet) équivaut à 1024 Go, le Po (péta-octet) équivaut à 1024 To, etc...

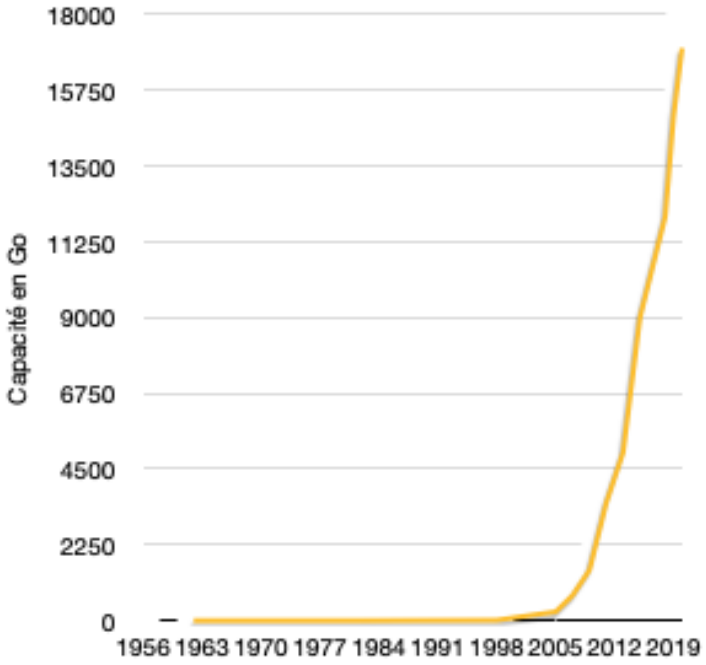
Comme toute donnée numérique requiert un support de stockage électronique, l'évolution des volumes de données a entraîné une croissance en termes de capacité des supports de stockage.

Le ruban perforé et la carte perforée ont été les premiers supports de stockage de données et date du 17$^{ème}$ Siècle. Quant au disque dur, il a été inventé au milieu du 19$^{ème}$ siècle, et s'est vu propulser, quelques années plus tard, sur le devant de la scène suite à une diminution drastique de ses dimensions ainsi que de son prix. Le disque dur a connu une évolution de son prix et de sa capacité de stockage pour le moins impressionnante.

Pour un disque de même taille, il était possible d'enregistrer 5 Mo en 1980 contre 500 Go en 2008.
Le prix de 1 Go est passé de 26 000 000 € en 1956 à 800 € en 1995 et 0,05 € en 2015[12].

La figure 1.1 montre l'évolution des capacités des disques durs depuis les années 1950 jusqu'à ce jour.

Il existe d'autres alternatives au disque dur comme par exemple, les bandes magnétiques qui peuvent offrir une plus grande capacité de stockage ( >180 To) ou encore via la technologie optique utilisée pour le DVD (jusqu'à 15,8 Go) et le Blu-Ray (100 Go).




| Capacité | Année |
|---|---|
| 5 Mo | 1956 |
| 28 Mo | 1962 |
| 1,02 Go | 1982 |
| 25 Go | 1998 |
| 500 Go | 2005 |
| 1 To | 2007 |
| 2 To | 2009 |
| 3 To | 2010 |
| 4 To | 2011 |
| 6 To | 2013 |
| 8 To | 2014 |
| 10 To | 2015 |
| 14 To | 2018 |
| 16 To | 2019 |
| 18 To | 2020 |

FIGURE 1.1 – Évolution des capacités de stockage des disques durs (toutes tailles confondues)[12]

Les disques durs standards sont actuellement menacés par l'émergence de technologie de la mémoire flash.
Cette technologie est beaucoup plus rapide que celle des disques durs mais sa durée de vie reste encore limitée.

Il existe deux grandes familles de mémoires flash :

— D'une part les cartes mémoires destinées aux petits matériels tels que les appareils photos numériques, les téléphones portables (cartes SD, mini SD, etc) ou encore les clés USB,

— D'autre part, les disques électroniques (disques SSD [1]) destinés à remplacer les disques durs standards.

En 2016, les particuliers pouvaient se procurer des disques SSD de 4 To, tandis que les professionnels pouvaient trouver des disques SSD de 16 To au

1. SSD (Solide State Drive) : matériel informatique permettant le stockage de données et constitué de mémoire flash.

prix de 10 000 €. La génération actuelle vise à stocker toutes les données en ligne dans le Cloud.
Ce nom imagé désigne la dernière révolution de l'internet hébergeant des millions d'armoires informatiques conservant des milliards de données.

## 1.2 Évolution du volume des données

La mise au point des nouvelles technologies de stockage entraîne une régression du coût par giga-octet.
Cette diminution peut laisser sous-entendre l'accroissement de l'espace de stockage. Mais la gestion d'énormes infrastructures implique d'autres coûts et le volume des données à stocker croît plus vite que l'augmentation des capacités de stockage. Il est donc plus intéressant de mettre en place des systèmes de compression et de déduplication de données.

Selon le Digital Economy Compass 2019 de Statista[33], le volume annuel de données numériques créé annuellement a été multiplié par plus de 20 au cours de la dernière décennie et devrait s'approcher de 50 zetta-octets (50 000 000 000 To) en 2020. Cette hausse s'explique suite à la démocratisation croissante des objets connectés et l'avènement de la technologie 5G qui sont les principaux moteurs du "big bang" des données numériques.

Le Digital Economy Compass 2019 estime que le volume annuel de données sera multiplié par environ 3,5 tous les 5 ans comme le montre la figure 1.2.

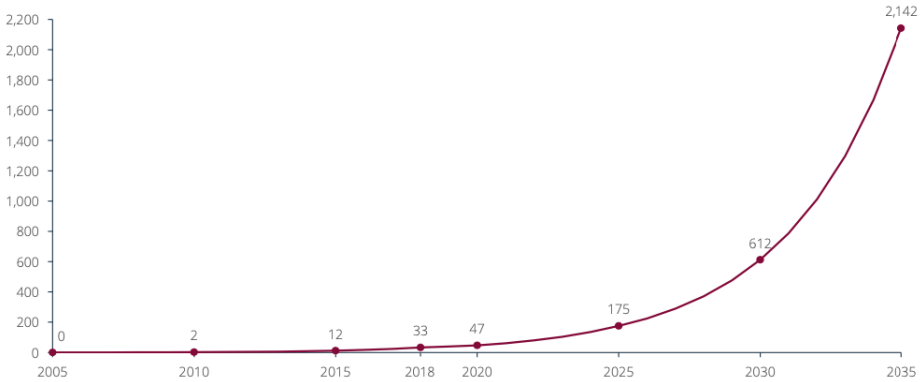



FIGURE 1.2 – Quantité mondiale de données créées par an en zettaoctets[33]

Ce "big bang" des données numériques, souvent redondantes, permet une approche différente pour analyser le monde.

Cette approche est appelée le "Big Data" et consiste à analyser des volumes très importants de données numériques dans le but d'en retirer de l'information. Ce dernier révolutionne de nombreux domaines comme les affaires et la recherche scientifique[30] et fait actuellement l'objet de nombreuses recherches scientifiques.

## 1.3 Structure du travail

Le but de ce mémoire est de décrire et de comparer les différentes techniques de réduction de données, à savoir, la compression et la déduplication de données. La première technique qui sera abordée dans le chapitre 2 est la compression de données.

Après une partie descriptive, différents algorithmes de compression sont passés en revue afin d'en comprendre leurs fonctionnements.
Les performances respectives de ces méthodes sont ensuite comparées sur base d'ensembles de fichiers de différents formats.
Avant de terminer cette partie, une comparaison des logiciels de compression les plus populaires est réalisée.

La deuxième technique de réduction de données est la déduplication de données et sera détaillée dans le chapitre 3.

Après une brève introduction, on discute de l'utilité de la déduplication.
On détaille ensuite l'endroit où s'opère la déduplication ainsi que le moment d'exécution du processus.
Ensuite, c'est au tour de l'architecture d'être analysée avant d'aborder la description de chaque étape du processus de déduplication.
Avant de citer et de détailler des outils de déduplication, on parlera du ratio de déduplication, ainsi que des différents facteurs qui peuvent influencer ce ratio.

Le chapitre 4 sera consacré à ces deux techniques de réduction de données afin de cibler leurs similitudes et leurs différences.

Enfin, le chapitre 5 de ce travail sera consacrée à la conclusion qui donnera une orientation sur les futures recherches dans ce domaine.

# Chapitre 2

# Compression de données

La compression de données consiste à réduire chaque fichier, chaque dossier de manière la plus transparente possible sur le disque. Les logiciels de compression de données sont devenus populaires dès l'apparition des premiers disques durs. Néanmoins, la compression de données a été fortement critiquée à cause de ses temps de latence parfois excessifs. Ainsi, des données sollicitées de manière fréquente, comme celles stockées dans une base de données, ne peuvent pas être compressées.
A contrario, les journaux d'évènements systèmes ou d'applications sont des candidats idéals à la compression.

La compression de données est une opération informatique consistant à transformer une suite de bits A en une suite de bits B plus courte, mais pouvant restituer les mêmes informations en utilisant un algorithme particulier.

A l'heure actuelle, certains systèmes de fichiers tels que NTFS [1] ou ZFS [2] intègrent un système de compression de données.
Le système d'exploitation Windows 10 de Microsoft embarque un algorithme de compression de données pour réduire l'espace alloué aux fichiers systèmes. Cela peut induire une diminution de l'espace de stockage jusqu'à 1,5 Go pour un système 32 bits et jusqu'à 2,6 Go pour un système 64 bits.

La compression de données peut être mise en oeuvre à différents niveaux :

1. NTFS (New Technology File System) : Technologie d'archivage de l'emplacement des fichiers propre à Windows NT.
2. ZFS (Z File System) : système de fichiers open source qui a pour caractéristiques sa très haute capacité de stockage.

— du coté hôte, à l'aide d'application comme WinZip, WinRAR ou encore 7-Zip,
— du côté stockage, avec le produit Data Compression de EMC[36] par exemple.

La compression de données est importante pour une entreprise car elle apporte différents avantages tels que :

— une réduction significative du matériel de stockage et donc de son coût,
— une réduction importante du temps de transfert des données,
— une nette diminution de l'utilisation de la bande passante du réseau.

Lorsque l'on parle de technique de compression ou d'algorithme de compression, on se réfère en réalité à un couple d'algorithmes. Le premier algorithme vise à compresser les données, tandis que le second sert à décompresser ces dernières.

La compression de données peut suivre deux approches distinctes :

— **La première approche** concerne la compression de données avec perte d'information.
Les données compressées via cette première méthode obtiennent généralement un meilleur taux de compression. Pour de nombreuses applications, cette perte d'information n'est pas un obstacle pour la reconstruction des données.
Par exemple, lors de l'enregistrement de musiques, la valeur exacte de chaque échantillon n'est pas essentielle. Suivant la qualité de la musique reconstruite, des quantités variables de pertes d'information peut être tolérées.
— **La seconde approche** est la compression sans perte d'information.
Les données originales peuvent être exactement récupérées à partir des données compressées. Cette approche est utilisée pour des applications ne tolérant aucun perte d'information.
Par exemple, la perte d'information n'est pas acceptable pour un document de type texte. Il est primordial que le document reconstruit soit identique au document original.

Le taux de compression (Tc) est calculé par la division de la taille compressée (Sc) par la taille originale (So).

$$Tc = \frac{Sc}{So}$$

## 2.1 Algorithme de compression

Cette partie a pour but de décrire le fonctionnement de différents algorithmes de compression. Ceux-ci font partie des standards et sont utilisés par la majorité des outils de compression.

Dans un premier temps, on détaillera des algorithmes de compression de données sans perte de données avec, entre autre, le codage de Huffman, le "LZ77" ou encore le "Deflate".
Ensuite, on analysera le "DCT" qui est un algorithme de compression avec perte de données.

Dans la majorité des cas présentés, la phrase suivante sera utilisée à titre d'exemple pour développer les différents algorithmes : "Un problème ou un petit problème est un problème".
Les caractères accentués ainsi que la majuscule ne seront pas pris en compte lors de ces démonstrations.

### 2.1.1 Huffman

Le codage de Huffman est un algorithme de compression de données sans perte[32, 28]. Il utilise un code à longueur variable pour représenter un symbole de la source. Le code est déterminé à partir d'une estimation des probabilités d'apparition des symboles de source.

#### Description de l'algorithme

La première étape consiste à rechercher le nombre d'occurrences de chaque symbole. La table 2.1 représente le nombre d'occurrences des différents symboles rencontrés dans la phrase d'exemple.

La deuxième étape consiste en la construction du dictionnaire à l'aide d'un arbre binaire. La figure 2.1 montre l'arbre binaire correspondant à la table générée lors de la première étape.
L'arbre est construit de sorte que les feuilles les plus proches de la racine contiennent les symboles possédant les plus grands nombres d'occurrences.

| Caractère | Nombre d'occurrence |
|---|---|
| U | 4 |
| N | 3 |
| ESPACE | 8 |
| P | 4 |
| R | 3 |
| O | 4 |
| B | 3 |
| L | 3 |
| E | 8 |
| M | 3 |
| T | 3 |
| I | 1 |
| S | 1 |

TABLE 2.1 – Nombre d'occurrence des symboles de la phrase "un probleme ou un petit probleme est un probleme"

Le symbole "E" (8 occurrences) se situe à 2 noeuds de la racine tandis que le symbole "S" (1 occurrence) se situe à 4 noeuds de la racine.

Le codage de Huffman s'obtient en lisant l'arbre binaire depuis la racine jusqu'au symbole souhaité.
Par exemple, Le symbole "M" est codé "0110" car, pour atteindre ce symbole depuis la racine, il faut d'abord atteindre le noeud se situant à gauche "0" puis, depuis ce noeud, il faut aller à droite "1", ensuite encore à droite "1" et, pour finir, on atteint le symbole en allant vers la gauche "0".

En analysant la table 2.2, qui compare l'encodage via la méthode de Huffman et le codage ASCII [3], on peut calculer qu'il est nécessaire d'utiliser 170 bits pour encoder la phrase exemple à l'aide de la méthode Huffman contre 384 (48*8) bits avec le code ASCII. Une moyenne de 3,54 bits sont nécessaires pour un symbole en utilisant la méthode de Huffman contre 8 bits par le code ASCII. Cette approche nous permet d'obtenir un taux de compression de 44,27 %

$$\frac{170}{384} = 44,27\%$$

3. ASCII (American Standard Code for Information Interchange) : code informatique standardisé pour la représentation des caractères,

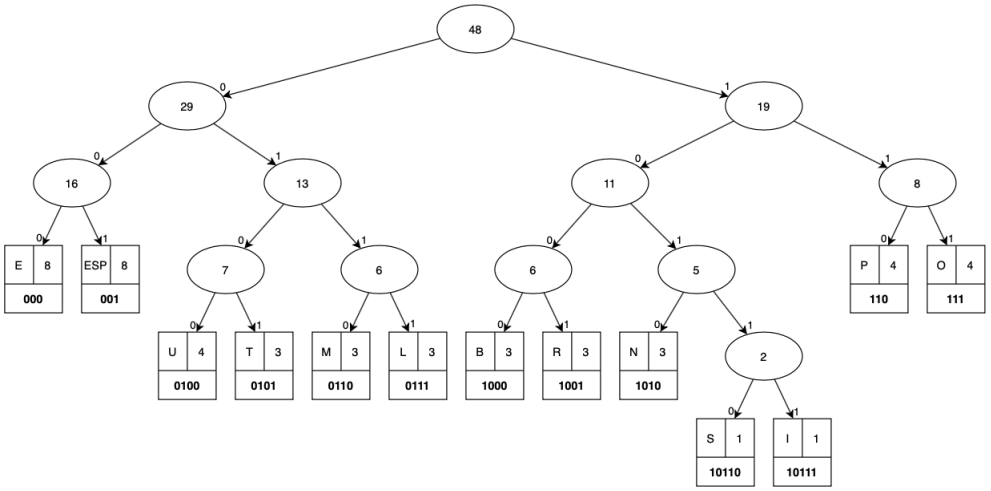



FIGURE 2.1 – Arbre binaire généré

Tous les caractères sont situés sur les feuilles de l'arbre. Par conséquent, le codage d'un symbole ne peut pas être le préfixe d'un autre codage. Dès lors, le décodage (décompression) peut se faire de manière non ambiguë de la gauche vers la droite.

### 2.1.2 Codage par dictionnaire

Le codage par dictionnaire consiste à n'enregistrer qu'une seule fois un mot et, pour chaque répétition de ce mot, d'utiliser une référence vers le mot enregistré. Les mots prennent place dans un dictionnaire, et chacun de ces mots est remplacé par son adresses unique dans le dictionnaire. Le fichier compressé contient les adresses du dictionnaire à la place des symboles.

La phrase exemple retenue ("un problème ou un petit problème est un problème") est composée de 48 symboles, soit 48 octets.
La technique du dictionnaire enregistre chaque nouveau mot, rencontré dans le fichier source, dans une table en lui associant un identifiant unique.
La table 2.3 représente le dictionnaire généré par la phrase d'exemple. On constate que l'enregistrement de ce dictionnaire demande 27 octets. En effet, cette taille découle de l'addition de la taille de chaque ligne du dictionnaire. La taille d'une ligne correspond au nombre de symbole que compose le mot ainsi que de l'identifiant.

Après l'établissement du dictionnaire, l'encodage est généré. Ce ne sont plus les mots que contient le fichier, mais la valeur de l'identifiant du diction-

| Caractère | Nombre d'occurrence | Huffman | ASCII |
|---|---|---|---|
| U | 4 | 0100 | 01110101 |
| N | 3 | 1010 | 01101110 |
| ESPACE | 8 | 001 | 00100000 |
| P | 4 | 110 | 01110000 |
| R | 3 | 1001 | 01110010 |
| O | 4 | 111 | 01101111 |
| B | 3 | 1000 | 01100010 |
| L | 3 | 0111 | 01101100 |
| E | 8 | 000 | 01100101 |
| M | 3 | 0110 | 01101101 |
| T | 3 | 0101 | 01110100 |
| I | 1 | 10111 | 01101001 |
| S | 1 | 10110 | 01110011 |

TABLE 2.2 – Comparaison : Méthode Huffman et codage ASCII

| Identifiant | Mot | Taille |
|---|---|---|
| 1 | un | 3 |
| 2 | ESPACE | 2 |
| 3 | problème | 9 |
| 4 | ou | 3 |
| 5 | petit | 6 |
| 6 | est | 4 |
| | | Total = 27 |

TABLE 2.3 – Exemple de codage par dictionnaire pour la phrase "un problème ou un petit problème est un problème"

naire correspondant.
La phrase retranscrite avec ce dictionnaire donnerait :

12324212523262123

En additionnant la taille de l'encodage qui est de 17 octets et la taille du dictionnaire, on obtient un total de 44 octets. On constate donc une diminution de la taille nécessaire pour enregistrer la phrase.
Dans cette exemple, la différence est minime (4 octets). En revanche, cette technique peut s'avérer très performante pour de plus gros documents.
Dans la pratique, les références du dictionnaire doivent être séparées les unes des autres afin de les différencier, ce qui augmente la taille de l'encodage.

### 2.1.3 LZ77 (Lempel - Ziv 77)

L'idée essentielle de l'algorithme LZ77 est d'utiliser une partie de la donnée d'entrée comme un dictionnaire.
L'algorithme de compression fait glisser une fenêtre de N symboles sur la chaîne d'entrée de la gauche vers la droite[44, 3].

Cette fenêtre est composée de deux parties

- À droite, le tampon de lecture de F symboles dans lequel se trouvent les symboles en attente de compression.
- À gauche, le tampon de recherche de N-F symbole qui constitue le dictionnaire courant des symboles qui ont été lus et comprimés ;

Pour arriver à comprimer les données, l'algorithme recherche dans le tampon de lecture une chaîne de symboles similaires se trouvant dans le tampon de recherche.

Les séquences d'octets à compresser sont codées sous forme d'un triplet (A, B, C) où :

- **A** équivaut à la position de la séquence d'octets dans le tampon de recherche.
- **B** équivaut à la longueur de la séquence compactée.
- **C** équivaut au premier octet qui diffère de la séquence d'octets compactée.

Si aucune séquence d'octets ne peut être compressée, la valeur 0 est associée à A et B tandis que C prend le premier symbole du tampon. La fenêtre se décale de B+1 octet vers la droite de sorte à passer en revue chaque symbole.

La figure 2.2 met en avant le fonctionnement de la compression LZ77 avec la phrase "un problème ou un petit problème est un problème".
Pour cet exemple, la fenêtre est composée de 30 symboles.
La partie, à droite, encadrée de couleur orange représente le tampon de lecture composé de 8 symboles.
La partie, à gauche, encadrée de couleur rouge représente le tampon de recherche et est composé de 22 symboles (30 - 8)

Les tailles des tampons ont été choisies pour mettre en avant le comportement de l'algorithme LZ77.

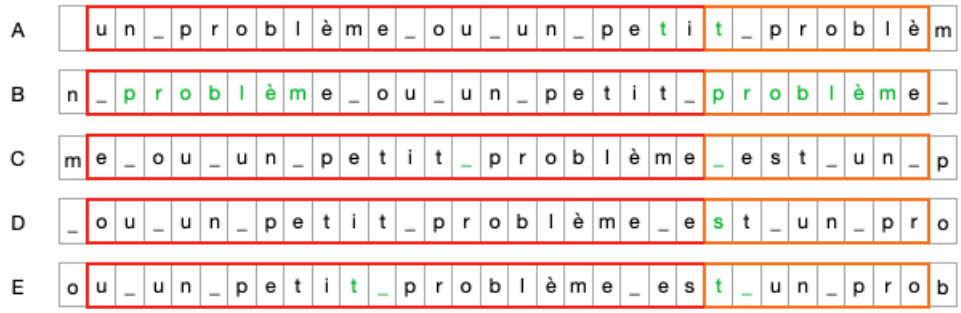



FIGURE 2.2 – Analyse de l'algorithme LZ77

La figure 2.2 représente un morceau de l'algorithme qui a déjà été exécuté en partie lors du traitement, et qui devra être terminé ensuite.
Ce morceau d'algorithme a été choisi afin de mettre en évidence les différentes situations qui peuvent être rencontrées.

Ci-dessous, on reprend le détail des différentes situations rencontrées sur l'exemple décrit sur la figure 2.2.

**Premier cas - ligne A** : Le tampon de lecture lit le premier symbole ("t") et lance une recherche dans le tampon de recherche pour retrouver ce symbole.
La recherche s'effectue de la droite vers la gauche et s'arrête à la première occurrence trouvée.

Dans l'exemple, le "t" se situe a 2 symboles de la fin du tampon de recherche.

Après avoir trouvé le premier symbole, le tampon de lecture lit le deuxième symbole ("_") et recherche dans le tampon de recherche la combinaison de

ces 2 symboles ("t_").

La recherche ne donnant aucun résultat, le triplet remplaçant cette partie est (2, 1, _) et la fenêtre se décale de 2 positions vers la droite.

**Deuxième cas - ligne B** : Le tampon de lecture lit le premier symbole ("p") et lance une recherche dans le tampon de recherche pour retrouver ce symbole. Un "p" est trouvé et est celui du mot "petit".

La recherche s'effectue en ajoutant le deuxième symbole du tampon de lecture ("r").

Les symboles "pr" sont trouvés dans le tampon de recherche. Ce cycle continue jusqu'à la lecture du dernier symbole de tampon de recherche.

Cette fois-ci, pas besoin d'effectuer une recherche car on arrive sur un "cas de base". En effet, le dernier symbole du tampon de lecture ne peut être recherché de la tampon de recherche car celui-ci est utilisé pour générer le triplet.

Le triplet généré est (21, 7, e) et la fenêtre se décale de 8 positions vers la droite.

**Troisième cas - ligne D** : Le tampon de recherche ne trouvant pas le premier symbole demandé par le tampon de lecture, le triplet généré est (0,0,s). La fenêtre se décale de 1 position vers la droite.

Le fichier devient une suite de triplets dont voici une partie du contenu :

$$(0,0,u)(0,0,n)...(2,1,\_)(21,7,e)(9,1,e)(0,0,s)(13,2,u)...(22,3,r)(17,5,e)$$

De nombreuses applications utilisent cet algorithme. On le retrouve, entre autres, dans le système d'exploitation de Microsoft.
Cependant, le LZ77 est confronté à des inconvénients tel que l'incapacité de retrouver une séquence d'octets se trouvant en dehors de la fenêtre ainsi que de consommer plus d'un octet pour représenter une séquence d'octets compressée.
En effet, il est nécessaire de stocker le triplet :

- un premier nombre représentant la position de la séquence dans le tampon de recherche, dans notre cas avec un tampon de longueur de 22, il est nécessaire d'utiliser 5 bits.
- ensuite, pour stocker la longueur de la séquence compactée qui est de la longueur du tampon de lecture -1, ce qui donne 8-1=7 soit 3 bits.
- et enfin il est nécessaire d'utiliser un octet pour encoder le symbole.

Ce qui fait, dans l'exemple, un total de 5 + 3 + 8 = 16 bits soit 2 octets.

L'exemple nous montre que plus la fenêtre est grande, plus il faudra de bits pour coder chaque triplet ce qui impactera le taux de compression.
De plus, plus le dictionnaire est grand, plus la recherche d'une répétition dans le dictionnaire sera coûteuse en temps.
L'utilisation d'une trop grande fenêtre rendrait l'algorithme inutilisable.
Bien que l'algorithme peut être gourmand en temps, suite aux nombreuses comparaisons, le décodage est quant à lui très simple et rapide.
Que ce soit pour l'encodage ou le décodage, les besoins en mémoire sont relativement faibles. En effet, la seule structure conservée en mémoire est la fenêtre qui mesure généralement entre 4 et 64 Ko (kilo-octets).

Pour compresser la phrase d'exemple avec l'algorithme LZ77 et avec les tailles des tampons définis plus haut, il faut stocker 22 triplets.
Chaque triplet ayant une taille de 2 octets, la taille nécessaire est donc de 22*2 = 44 octets, soit 352 bits.

Le taux de compression obtenu est de :

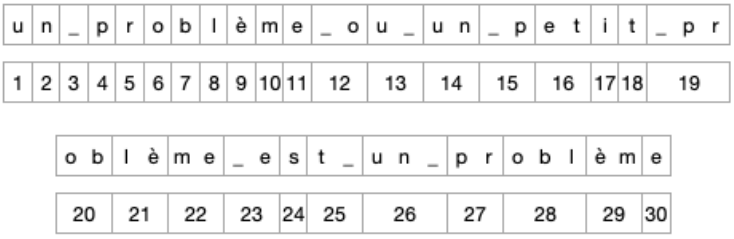
$$\frac{352}{384} = 91,67\%$$

Ce taux de compression n'est pas représentatif, car de manière générale, le taux de compression obtenu par cette méthode est très bon pour de nombreux types de données.

### 2.1.4 LZ78 (Lempel - Ziv 78)

LZ78[45] n'utilise pas, contrairement au LZ77, de fenêtre glissante pour parcourir les séquences d'octets.
Le LZ78 se satisfait de lire les symboles et de former un dictionnaire de manière progressive.
Cet algorithme va lire les symboles de gauche à droite et ajouter toutes les nouvelles parties rencontrées dans le dictionnaire.
Les nouvelles parties sont stockées sous forme de couple, le premier argument référence un mot déjà connu sinon il vaut zéro et le second argument contient le nouveau symbole.
Si le symbole/mot se trouve déjà dans le dictionnaire, l'algorithme va prendre le symbole suivant pour former un groupe de symbole (mot).

La figure 2.3 met en évidence le fonctionnement de l'algorithme LZ78 avec la phrase d'exemple "un problème ou un petit problème est un problème".
La phrase mentionnée dans l'exemple requiert un espace mémoire de 384




(a) Représentation des séquences stockées dans le dictionnaire

| Étape | Dictionnaire | Sortie |
|---|---|---|
| 1 | u | (0,u) |
| 2 | n | (0,n) |
| 3 | _ | (0,_) |
| 4 | p | (0,p) |
| 5 | r | (0,r) |
| 6 | o | (0,o) |
| 7 | b | (0,b) |
| 8 | l | (0,l) |
| 9 | è | (0,è) |
| 10 | m | (0,m) |
| 11 | e | (0,e) |
| 12 | _o | (3,o) |
| 13 | u_ | (1,_) |
| 14 | un | (1,n) |
| 15 | _p | (3,p) |

| Étape | Dictionnaire | Sortie |
|---|---|---|
| 16 | et | (11,t) |
| 17 | i | (0,i) |
| 18 | t | (0,t) |
| 19 | _pr | (15,r) |
| 20 | ob | (6,b) |
| 21 | lè | (8,è) |
| 22 | me | (10,e) |
| 23 | _e | (3, e) |
| 24 | s | (0,s) |
| 25 | t_ | (18,_) |
| 26 | un_ | (14,_) |
| 27 | pr | (4,r) |
| 28 | obl | (20,l) |
| 29 | èm | (9,m) |
| 30 | e | (11,ø) |

(b) Représentation du dictionnaire

FIGURE 2.3 – Principe de fonctionnement de l'algorithme LZ78 avec la phrase "un problème ou un petit problème est un problème"

bits (48*8) sans utiliser de moyen de compression.

L'information est enregistrée sous la forme d'un couple :

— **le premier argument** permet de faire référence à un mot déjà connu dans le dictionnaire et utilise le strict minimum nécessaire de bit pour son encodage.

Par exemple, la première étape ne contiendra pas de bit car le premier symbole lu ne sera jamais présent dans le dictionnaire.
La deuxième étape contiendra 1 bit de référence ce qui permet de référencer, si besoin, la première étape.
La troisième étape contiendra, quant à elle, 2 bits afin d'accéder à la référence des étapes 1 et 2.
La quatrième étape contiendra également 2 bits afin de pouvoir référencer les étapes 1, 2 ou 3 (en binaire 2 bits permet d'écrire les chiffres de 0 à 3). Et ainsi de suite.

— **le deuxième argument** contiendra 1 octet soit 8 bits afin de stocker le nouveau symbole.

Au total, le premier argument prendra l'espace de 119 bits et le deuxième argument occupera un espace de 232 bits (soit 29*8 bits).
Le taux de compression obtenu est de :

$$\frac{232}{384} = 60,42\%$$

L'avantage de cet algorithme par rapport au LZ77 est le nombre réduit de comparaisons de chaînes lors de chaque étape de codage.
En revanche, il a connu moins de succès pour des raisons d'efficacité, mais aussi parce qu'il a été protégé par un brevet logiciel aux États-Unis.

### 2.1.5 LZW (Lempel Ziv Welch)

L'algorithme LZW constitue une évolution de l'algorithme LZ78[38]. Cette méthode a une approche différente puisqu'elle consiste à débuter avec un dictionnaire contenant les 256 codes étendus de la table ASCII[28, 20, 3].
L'utilisation d'un dictionnaire dynamique permet l'ajout de symboles ou de séquences de symboles supplémentaires afin d'augmenter la table ASCII contenue dans ce dictionnaire.

Chaque nouvelle entrée sera ajoutée à la fin du dictionnaire avec un indice incrémenté par rapport à la dernière entrée.
Cependant, dans certains cas comme la compression d'image TIF, les codes 256 et 257 servent de codes de fin et de remise à zéro des séquences apprises. Ainsi, la première entrée disponible est la $258^{\text{ème}}$.

De manière générale, la taille des codes est limitée à 12 bits. Les 8 premiers bits sont utilisés pour les 256 codes de la table ASCII étendue tandis que les

bits suivants sont utilisés afin d'ajouter de nouveaux symboles ou de nouvelles séquences. Ce qui laisse un maximum de $2^{12}$ - $2^8$ = 3840 codes de disponible.

Cet algorithme ne nécessite pas l'enregistrement du dictionnaire. En effet, le dictionnaire est créé lors de la phase de compression mais également lors de la décompression.

La figure 2.4 dévoile le dictionnaire généré pour la phrase "un problème ou un petit problème est un problème".

| Caractère lu | Séquence groupée | Ajout dictionnaire (code, séquence) |
|---|---|---|
| u | u | - |
| n | un | (256, un) |
| _ | n_ | (257, n_) |
| p | _p | (258, _p) |
| r | pr | (259, pr) |
| o | ro | (260, ro) |
| b | ob | (261, ob) |
| l | bl | (262, bl) |
| è | lè | (263, lè) |
| m | èm | (264, èm) |
| e | me | (265, me) |
| _ | e_ | (266, e_) |
| o | _o | (267, _o) |
| u | ou | (268, ou) |
| _ | u_ | (269, u_) |
| u | _u | (270, _u) |
| n | un | - |
| _ | un_ | (271, un_) |
| p | _p | - |
| e | _pe | (272, _pe) |
| t | et | (271, et) |
| i | ti | (272, ti) |
| t | it | (273, it) |
| _ | t_ | (274, t_) |

| Caractère lu | Séquence groupée | Ajout dictionnaire (code, séquence) |
|---|---|---|
| p | _p | - |
| r | _pr | (275, _pr) |
| o | ro | - |
| b | rob | (276, rob) |
| l | bl | - |
| è | blè | (277, blè) |
| m | èm | - |
| e | ème | (278, ème) |
| _ | e_ | - |
| e | e_e | (279, e_e) |
| s | es | (280, es) |
| t | st | (281, st) |
| _ | t_ | - |
| u | t_u | (282, t_u) |
| n | un | - |
| _ | un_ | - |
| p | un_p | (283, un_p) |
| r | pr | - |
| o | pro | (284, pro) |
| b | ob | - |
| l | obl | (285, obl) |
| è | lè | - |
| m | lèm | (286, lèm) |
| e | me | - |

FIGURE 2.4 – Création du dictionnaire LZW pour la phrase "un problème ou un petit problème est un problème"

Sur base de ce dictionnaire, la phrase compressée sera représentée comme ci-dessous. Les caractères présents dans cette phrase sont à remplacer par leur code parmi les équivalents standards de la table étendue ASCII.

un_problème_ou_(256)(258)etit(258)(260)(262)(264)(266)es(274)(271)(259)(261)(263)(265)

Dans cette exemple, ayant moins de 256 nouvelles entrées dans le dictionnaire, l'encodage peut se faire sur 9 bits.
La phrase compressée contient 34 codes de 9 bits soit 34 * 9 = 306 bits.
Pour rappel, la phrase d'exemple non compressée contient 48 symboles soit 48 * 8 = 384 bits.

Le taux de compression obtenu est ainsi de :

$$\frac{306}{384} = 79,69\%$$

### 2.1.6 DEFLATE

Deflate est un format de compression de données sans perte qui couple l'algorithme LZ77 et le codage de Huffman.

L'algorithme Deflate génère une série de blocs dans lesquels est introduit un en-tête de 3 bits.
Le premier bit permet de spécifier si c'est le dernier bloc (1) ou non (0) et les deux suivants servent de paramètres avec de manière générale, pour les données compressibles, les bits "10" qui correspondent à la méthode de codage dynamique de Huffman.
Les instructions pour générer l'arbre de Huffman suivent immédiatement l'en-tête du bloc.

La compression se fait en 2 étapes :

- **Premièrement** l'algorithme LZ77 est utilisé afin de supprimer les redondances, en insérant des références vers la partie identique.
  C'est la partie la plus coûteuse en terme de calcul lors de la compression.
- **Deuxièmement** le codage de Huffman est utilisé afin de remplacer les symboles les plus utilisés par des représentations plus courtes et,

inversement, les symboles les moins utilisés par des représentations plus longues.

### 2.1.7 RLE (Run-Length Encoding)

Le Run-Length Encoding ou en français, le codage par plage, est une forme de compression de données sans perte qui consiste à regrouper les symboles identiques et consécutifs à l'aide d'un compteur de telle sorte qu'il n'y ait plus aucun symbole identique qui se suit[20].

Cet algorithme n'est pas compatible avec la phrase d'exemple car celle-ci ne contient aucun symbole identique et consécutif.
Par contre, il correspond bien à la compression des images jusqu'à 24 bits par pixel.
Notre exemple, en figure 2.5, met en scène une image en noir et blanc de 5 * 5 pixels.

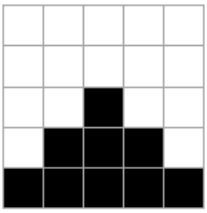


FIGURE 2.5 – Image d'exemple pour l'algorithme RLE"

Pour illustrer cet exemple, un pixel de couleur blanche est représenté par un "B" tandis qu'un pixel de couleur noire est représenté par un "N".
Les pixels de l'image peuvent ainsi être représentés en lisant l'image ligne par ligne de gauche à droite et de haut en bas.
L'image peut donc être stockée de la manière suivante :

BBBBBBBBBBBNBBBNNNBNNNNN

Le RLE permet de réduire la quantité de données stockées en comptant le nombre de symboles identiques qui se suivent.
Grâce au RLE, l'image peut être stockée sous la forme suivante :

12B1N3B3N1B5N

Les 12 premiers pixels de couleur blanche sont représentés par "12B", le pixel suivant, de couleur noire est représenté par "1N" et ainsi de suite.

### 2.1.8 MTF (Move-to-front)

L'algorithme MTF[6] consiste à remplacer chaque caractère par un indice qui est donné par un tableau évoluant de manière dynamique.
Cette technique donne un des meilleurs résultats en l'utilisant conjointement avec la transformée de Burrows-Wheeler[7].
La compression de la phrase d'exemple se réalise en 2 étapes.

La première étape consiste à réorganiser les symboles avec la transformée de Burrows-Wheeler.

La deuxième étape consiste à réaliser la compression avec l'utilisation de l'algorithme MTF.

**Étape 1** : le passage de l'algorithme Burrows-Wheeler.
Il s'agit d'une méthode de réorganisation des données qui permet d'augmenter la probabilité que des caractères identiques initialement éloignés les uns des autres se retrouvent côte à côte dans le résultat.

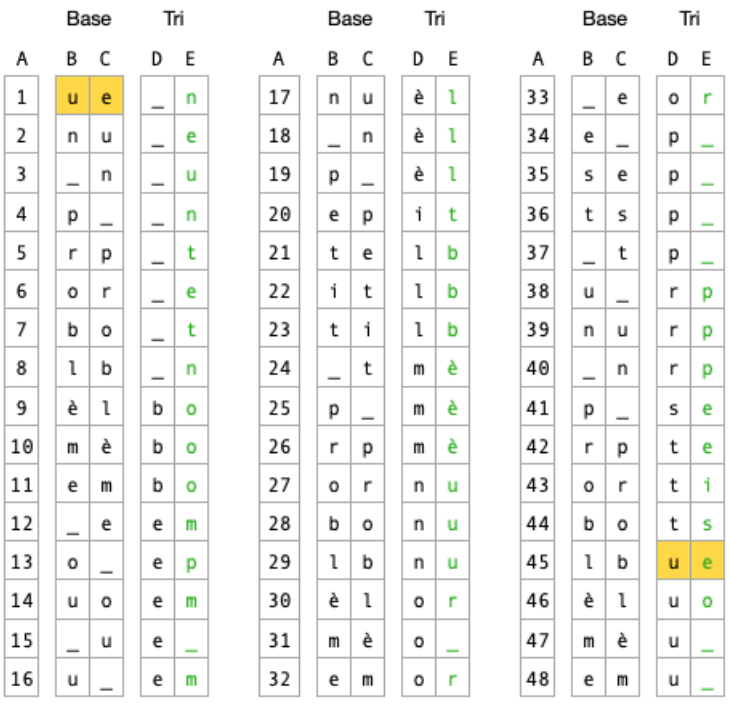



FIGURE 2.6 – Algorithme Burrows-Wheeler avec la phrase "un problème ou un petit problème est un problème"

La figure 2.6 permet de comprendre le fonctionnement de l'algorithme Burrows-Wheeler sur base de la phrase d'exemple "un problème ou un petit

problème est un problème".

La figure présente :

- la colonne "A" représente le numéro de la ligne ;
- la colonne "B" contient la phrase ;
- la colonne "C" contient le symbole correspondant à la ligne N-1 de la colonne "B", avec une exception pour la première ligne ou le symbole correspond au dernier symbole de la colonne "B" ;
- les colonnes "D" et "E" représentent l'équivalent des colonnes "B" et "C" avec un tri appliqué sur la colonne "B".

Le résultat de cette réorganisation se trouve dans la colonne "E". On constate bien des regroupements de symboles identiques avec par exemple les trois "o" qui se retrouvent côte à côte.
Cette colonne contient la phrase réorganisée qui sera utilisée par l'algorithme de compression.

Le numéro de ligne correspondant au début de la phrase des colonnes triées, en jaune dans la figure, est introduit en début de fichier et servira pour retrouver la phrase d'origine lors de la décompression. Dans l'exemple, il est représenté par la ligne 45.

À ce stade, le fichier contient :

45|neuntetnooompm_mllltbbbèèèuuur_rr_____pppeeiseo__

**Étape 2** : la compression avec Move-to-front.
L'algorithme MTF commence par définir l'alphabet en fonction des symboles présents.

Alphabet : beilmnoprstuè_

Ensuite, il lit les symboles un par un de gauche vers la droite, et récupère l'indice correspondant à la lettre de l'alphabet du symbole lu.
Lorsque l'indice est récupéré et enregistré, le symbole est alors déplacé en première position dans l'alphabet (move to front).
Ce remaniement de l'alphabet a pour but de récupérer un maximum d'indices avec des faibles valeurs. Ceux-ci permettent d'économiser des bits grâce à un encodage à longueur variable.

| Étape | Alphabet | | | | | | | | | | | | | | Symbole lu | Output |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | | |
| 1 | b | e | i | l | m | n | o | p | r | s | t | u | è | _ | n | 5 |
| 2 | n | b | e | i | l | m | o | p | r | s | t | u | è | _ | e | 2 |
| 3 | e | n | b | i | l | m | o | p | r | s | t | u | è | _ | u | 11 |
| 4 | u | e | n | b | i | l | m | o | p | r | s | t | è | _ | n | 2 |
| 5 | n | u | e | b | i | l | m | o | p | r | s | t | è | _ | t | 11 |
| 6 | t | n | u | e | b | i | l | m | o | p | r | s | è | _ | e | 3 |
| 7 | e | t | n | u | b | i | l | m | o | p | r | s | è | _ | t | 1 |
| 8 | t | e | n | u | b | i | l | m | o | p | r | s | è | _ | n | 2 |
| 9 | n | t | e | u | b | i | l | m | o | p | r | s | è | _ | o | 8 |
| 10 | o | n | t | e | u | b | i | l | m | p | r | s | è | _ | o | 0 |
| ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ |
| 41 | p | _ | r | u | è | b | t | l | m | o | n | e | i | s | e | 11 |
| 42 | e | p | _ | r | u | è | b | t | l | m | o | n | i | s | e | 0 |
| 43 | e | p | _ | r | u | è | b | t | l | m | o | n | i | s | i | 12 |
| 44 | i | e | p | _ | r | u | è | b | t | l | m | o | n | s | s | 13 |
| 45 | s | i | e | p | _ | r | u | è | b | t | l | m | o | n | e | 2 |
| 46 | e | s | i | p | _ | r | u | è | b | t | l | m | o | n | o | 12 |
| 47 | o | e | s | i | p | _ | r | u | è | b | t | l | m | n | _ | 5 |
| 48 | _ | o | e | s | i | p | r | u | è | b | t | l | m | n | _ | 0 |

FIGURE 2.7 – Récupération du contenu compressé

La figure 2.7 représente les différents outputs obtenus pour chacun des symboles ainsi que le remaniement de l'alphabet.

Le symbole lu lors des étapes 9 et 10 est le "o". Toutefois, les indices correspondant à ce symbole ne sont pas identiques.
En effet, le "o" se trouve en $8^{\text{éme}}$ position lors de le neuvième étape et, suite au remaniement de l'alphabet, il se retrouve à la position zéro pour l'étape suivante.

La représentation visuelle de la phrase compressée est la suivante :

45|52(11)2(11)312800891(13)1(10)006900(13)00(10)00(12)
7101000800(11)0(12)(13)2(12)50

Tous ces numéros d'indice sont encodés en binaire en utilisant un encodage à longueur variable. Plus la valeur de l'indice est faible, moins le nombre de bits nécessaires sera élevé. La réorganisation de la phrase a joué un rôle important afin de diminuer les valeurs des indices et ainsi réduire le nombre de bits nécessaire pour enregistrer la phrase d'exemple.

Toujours pour ce même exemple, la figure 2.8 permet de faire une comparaison avec l'utilisation d'un encodage avec un nombre de bits avec une longueur fixe et avec une longueur variable.
La figure montre le nombre de répétitions pour chaque indice, la représentation binaire de chaque indice ainsi que le nombre de bits nécessaires à l'encodage (nombre de répétition * la longueur du code).

On remarque qu'il faut 72 bits en encodage fixe contre 36 bits en encodage variable pour encoder 18 fois l'indice 0 et, inversement, il faut 12 bits en encodage fixe contre 21 en encodage variable pour encoder 2 fois l'indice 13. On remarque l'importance de récupérer des indices de faible valeur. Sans la transformée de Burrows-Wheeler, cette phrase aurait nécessité un encodage plus long avec un encodage à longueur variable qu'avec un encodage à longueur fixe.

Au final, l'encodage à longueur variable permet d'encoder la phrase en utilisant 8 bits de moins.
La différence n'est pas très grande concernant cette phrase mais cet algorithme est principalement utilisé dans la compression d'images et de vidéos. En effet, avec une image, on retrouve un nombre plus ou moins important de pixels de même couleur côte à côte, ce qui augmente le nombre de cas avec l'indice 0 et, donc, diminue la taille du fichier compressé.

Afin de calculer le taux de compression, il est nécessaire d'ajouter l'alphabet des symboles aux 184 bits précédemment calculés.
Cela représente 13 symboles encodés sur 8 bits, ce qui fait un total de 104 bits auxquels il faut ajouter le numéro de la ligne reçue par l'algorithme Burrows-Wheeler afin de retrouver la phrase d'origine. La phrase faisant 48 caractères, ce nombre est encodé sur 7 bits.
Le nombre nécessaire de bits est donc de 295 et le taux de compression obtenu est de :

$$\frac{295}{384} = 76,82\%$$

| Indice | Nb Répétition | Encodage 4 bit | | Encodage à longueur variable | |
|---|---|---|---|---|---|
| | | Code | Longueur utile total | Code | Longueur utile total |
| **0** | 18 | 0000 | 72 | 00 | 36 |
| **1** | 5 | 0001 | 20 | 01 | 10 |
| **2** | 4 | 0010 | 16 | 100 | 12 |
| **3** | 1 | 0011 | 4 | 1010 | 4 |
| **4** | 0 | 0100 | 0 | 1011 | 0 |
| **5** | 2 | 0101 | 8 | 11000 | 10 |
| **6** | 1 | 0110 | 4 | 11001 | 5 |
| **7** | 1 | 0111 | 4 | 11010 | 5 |
| **8** | 3 | 1000 | 12 | 11011 | 15 |
| **9** | 2 | 1001 | 8 | 111000 | 12 |
| **10** | 2 | 1010 | 8 | 111001 | 12 |
| **11** | 3 | 1011 | 12 | 1110100 | 21 |
| **12** | 3 | 1100 | 12 | 1110101 | 21 |
| **13** | 3 | 1101 | 12 | 1110110 | 21 |
| **Total** | | | **192** | | **184** |

FIGURE 2.8 – Comparaison encodage à longueur fixe et variable

### 2.1.9 DCT (transformée en cosinus discrète)

Le DCT[5, 26] est un algorithme de compression avec perte qui est utilisé massivement pour la compression d'images JPEG, de fichiers musicaux tels que le MP3 ou encore les vidéos comme MPEG.

Avant de décrire le fonctionnement de l'algorithme DCT, il est nécessaire d'expliquer ce qu'est une transformée en cosinus discrète et son fonctionnement.
La transformée en cosinus discrète fonctionne en prenant des données (par exemple d'une image) et en les représentant comme la somme de beaucoup d'ondes.

Sur la figure 2.9, se trouvent 2 ondes cosinus de fréquence différente cos(x) (couleur bleue) et cos(2x) (couleur rouge).
En augmentant le nombre d'ondes, on augmente également le nombre de possibilités de forme de l'onde résultante.
Le résultat de la combinaison de ces 2 ondes obtenu sur cette figure (couleur

verte) représente la moyenne de ces 2 ondes et forme une onde plus complexe.
Ce résultat implique que les 2 cosinus aient la même pondération.
En modifiant les pondérations des ondes, l'onde résultante dessinera des formes différentes.
Chaque onde représente un constituant du résultat.

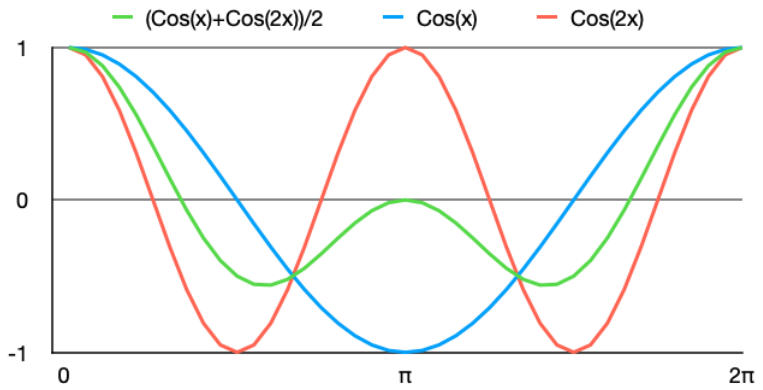



FIGURE 2.9 – Transformée en cosinus discrète : exemple

Afin de décrire le fonctionnement de l'algorithme de compression DCT, une image en noir et blanc est utilisée (voir figure 2.10).
Le choix d'une image en noir et blanc a pour but de simplifier la compréhension.
L'algorithme découpe l'image en groupe de 8x8 pixels et chacun de ces groupes de pixels est codé séparément avec sa propre transformée en cosinus discrète.
Chacun de ces groupes (de 8 par 8 pixels) peut être répliqué à l'identique avec 64 ondes, soit 8 ondes pour la partie horizontale et 8 ondes pour la partie verticale ainsi que toutes les intersections comme le prouve l'article sur la transformée en cosinus discrète[1].

La figure 2.11 montre les 64 ondes cosinus de base qui permettent de produire n'importe quelle image possible de faire en 8x8 pixels.

Si le résultat est exclusivement le premier élément de la matrice, le groupe de pixels aura une couleur unie et sera une variante entre le blanc et le noir.
Si le résultat est exclusivement le deuxième élément de la matrice (première ligne, deuxième colonne), le groupe de pixels résultant sera un dégradé en partant d'une couleur claire vers une couleur foncée ou inversement.

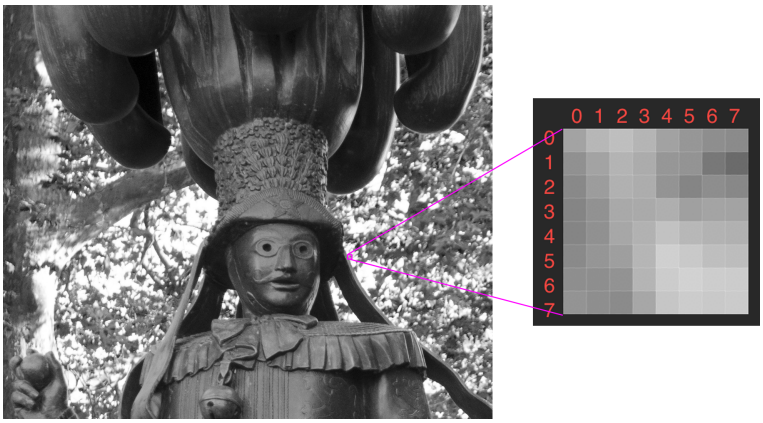



FIGURE 2.10 – Statue gille de Binche : Image d'exemple

Plus on se dirige vers le coin inférieur droit de la matrice des fréquences DCT (figure 2.11), plus on s'oriente vers des éléments de haute fréquence et plus la variation d'intensité de couleurs est importante.

Pour créer n'importe quel type d'image 8x8, il est nécessaire de combiner tout ou une partie des éléments de cette matrice en même temps.
Tous ces éléments sont pondérés en fonction d'un coefficient qui représente la contribution individuelle de cet élément à l'ensemble.
Un élément avec un coefficient de 0,1 aura 100 fois moins d'impact sur l'image résultante qu'un élément ayant un coefficient de 10.

Le travail essentiel de cet algorithme est de calculer les coefficients des différentes ondes.
Pour ce faire, la première étape est de récupérer l'intensité de la couleur RGB de chaque pixel du groupe de pixels à traiter.
Cette valeur se trouve entre 0 et 255 (figure 2.12a) ; 0 signifiant pas de couleur (noir) et 255 l'intensité maximale de la couleur (blanc).

La deuxième étape consiste à centrer les valeurs autour du zéro afin qu'elles correspondent à l'onde du cosinus qui va de -1 à 1.
Cette étape est simple, il suffit de soustraire 128 (256/2) de chacune des valeurs (figure 2.12b).
Ensuite, il faut appliquer la transformée en cosinus discrète afin de calculer la contribution de chacune des ondes cosinus grâce à la formule suivante :

$$DCT(i,j) = \frac{1}{\sqrt{2N}} C(i)C(j) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} P(x,y) cos(\frac{(2x+1)i\pi}{2N}) cos(\frac{(2y+1)j\pi}{2N})$$

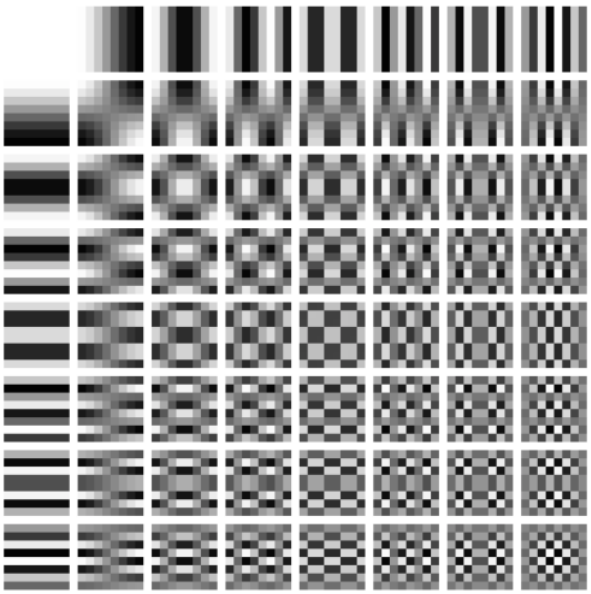


FIGURE 2.11 – DCT fréquence 2 dimensions 8x8[39]

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 0 | 163 | 183 | 191 | 181 | 162 | 151 | 143 | 139 |
| 1 | 145 | 163 | 180 | 172 | 151 | 148 | 120 | 107 |
| 2 | 137 | 157 | 177 | 173 | 148 | 133 | 142 | 136 |
| 3 | 132 | 146 | 169 | 170 | 175 | 160 | 164 | 163 |
| 4 | 133 | 145 | 167 | 178 | 194 | 184 | 179 | 177 |
| 5 | 138 | 149 | 159 | 182 | 208 | 201 | 190 | 189 |
| 6 | 143 | 142 | 181 | 181 | 204 | 210 | 205 | 204 |
| 7 | 148 | 143 | 167 | 167 | 198 | 204 | 201 | 203 |

(a) Matrice correspondant au intensité de couleur RGB

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 0 | 35 | 55 | 63 | 53 | 34 | 23 | 15 | 11 |
| 1 | 17 | 35 | 52 | 44 | 23 | 20 | -8 | -21 |
| 2 | 9 | 29 | 49 | 45 | 20 | 5 | 14 | 8 |
| 3 | 4 | 18 | 41 | 42 | 47 | 32 | 36 | 35 |
| 4 | 5 | 17 | 39 | 50 | 66 | 56 | 51 | 49 |
| 5 | 10 | 21 | 31 | 54 | 80 | 73 | 62 | 61 |
| 6 | 15 | 14 | 53 | 53 | 76 | 82 | 77 | 76 |
| 7 | 20 | 15 | 39 | 39 | 70 | 76 | 73 | 75 |

(b) Matrice correspondant au intensité centrée sur zéro

FIGURE 2.12 – Matrice d'intensité de couleurs correspondant à la figure 2.10

Avec

- N : la largeur du bloc, dans notre exemple 8
- i, j : les indices du coefficient de la DCT
- x, y : les indices du pixel de l'image (dans le bloc)
- DCT(i,j) : la valeur du coefficient à la position (i, j)
- P(x,y) : la valeur à la position x, y de la matrice d'intensité de couleur
- C(x) = $\frac{1}{\sqrt{2}}$ si x = 0, sinon C(x) = 1

On obtient ainsi la matrice des coefficients DCT telle que affichée sur la figure 2.13.
Chaque valeur correspond à la pondération de l'image correspondante de la matrice des fréquences DCT 2.11.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 0 | 307,9 | -49,3 | -77,4 | -18,8 | -6,6 | -2,8 | 8,1 | 11,5 |
| 1 | -85,8 | 122,8 | -8,4 | -30,9 | -12 | 5,9 | -8,1 | -12,3 |
| 2 | 20,1 | 21,4 | 10,5 | 5,3 | -7,9 | 1,0 | 9,5 | -1,9 |
| 3 | 37,0 | -11,1 | -9,1 | 12,3 | 3,8 | -8 | -2,4 | -4,2 |
| 4 | 13,1 | -11,3 | 10,8 | 0,7 | 3,1 | -3,8 | -1,6 | 5,0 |
| 5 | 18,8 | -10,0 | 8,5 | -11,2 | 0,8 | 3,5 | -0,7 | 5,5 |
| 6 | 1,3 | 0,2 | 9,8 | -3,5 | 5,3 | -1,9 | -10,5 | -1,8 |
| 7 | 6,0 | 0,5 | 3,5 | -10,5 | -1,7 | 5,8 | 0,6 | 2,3 |

FIGURE 2.13 – Matrice des coefficients DCT de la figure 2.10

Grâce à cette matrice, il est possible de retrouver l'image d'origine sans perte de qualité.
Pour ce faire, il suffit d'assembler chacune de ces ondes cosinus en fonction de leur pondération.
Plus sa pondération est importante, plus son impact sur l'image sera important.

Habituellement, la première image de la matrice des fréquences DCT à un coefficient élevé parce qu'il n'y a pas de variation d'intensité. Cette valeur représente l'intensité générale de l'image.
De manière générale, les coefficients de la matrice des coefficient DCT se trouvant dans le coin supérieur gauche sont élevés tandis que ceux se situant dans le coin inférieur droit sont faibles.
Cela signifie que la variation importante de l'intensité de couleur, aussi appelée la haute fréquence des ondes cosinus, ne contribuent que pour très peu à l'image. Ces hautes fréquences ont des effets très subtils sur les intensités de couleurs lors de la régénération de l'image.

Par exemple, dans la matrice des coefficients de la figure 2.13, et sans tenir compte de la première valeur (position (0,0)), on observe que les images

les plus importantes de la matrice des fréquences DCT sont aux positions (0,2),(1,0) et (1,1) avec des pondérations de -77,4, -85,8 et 122,8.
A contrario, l'image à la position (6,1) n'a que très peu d'impact pour la reconstruction de l'image.

En observant l'image d'exemple (figure 2.10), on observe facilement que les 3 images de la matrice des fréquences, avec une pondération importante citées plus haut, joue un rôle important.
On peut par ailleurs constater que l'image à la position (1,1) de la matrice des fréquences est assez marquée sur l'image d'exemple avec les coins haut droit et bas gauche plus sombre.

Sachant que certaines ondes ont des effets très subtils sur l'image finale, les enlever ne modifierait que partiellement l'image.

L'étape suivante est le processus de suppression de la quantification des données à haute fréquence[37].
Cela consiste à supprimer les détails qui sont pratiquement invisible à l'oeil nu afin de réduire la quantité de données que représente l'image originale.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 0 | 16 | 12 | 14 | 14 | 18 | 24 | 49 | 72 |
| 1 | 11 | 12 | 13 | 17 | 22 | 35 | 64 | 92 |
| 2 | 10 | 14 | 16 | 22 | 37 | 55 | 78 | 95 |
| 3 | 16 | 19 | 24 | 29 | 56 | 64 | 87 | 98 |
| 4 | 24 | 26 | 40 | 51 | 68 | 81 | 103 | 112 |
| 5 | 40 | 58 | 57 | 87 | 109 | 104 | 121 | 100 |
| 6 | 51 | 60 | 69 | 80 | 103 | 113 | 120 | 103 |
| 7 | 61 | 55 | 56 | 62 | 92 | 92 | 101 | 99 |

FIGURE 2.14 – Matrice standard JPEG définissant le niveau de quantification pour une qualité de 50%[40].

Pour ce faire, il est nécessaire d'avoir une table de quantification, comme par exemple, la table de quantification JPEG standard pour une qualité de 50% représentée sur la figure 2.14.
Cette matrice de quantification, enregistrée dans le fichier compressé, va être

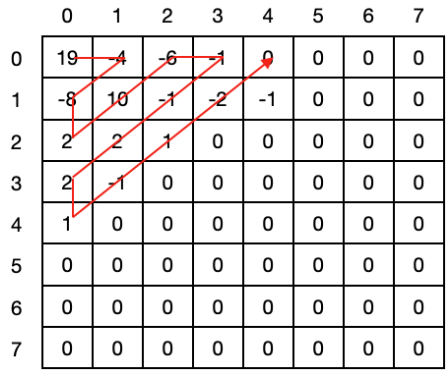
utile pour diviser toutes les valeurs de la matrice de coefficient DCT par la valeur correspondante et ensuite, arrondi à l'entier le plus proche.

On peut observer sur cette matrice de quantification JPEG que les valeurs situées dans la partie inférieure droite sont les plus élevées.
Cela a pour effet de réduire drastiquement les coefficients DCT se trouvant dans cette zone et de les supprimer si leur nouvelle valeur est égale à zéro.

La matrice de coefficient DCT, mise à jour sur la figure 2.15a, permet de constater un nombre important de 0.
Tous ces zéros signifient qu'ils n'ont plus d'utilité pour le reconstruction de l'image.
Seul un nombre limité de coefficients sont nécessaires pour reproduire approximativement l'image originale.
En effet, l'image ne sera pas exactement la même car quelques pixels auront un niveau d'intensité un peu plus élevé ou plus faible.
De manière générale, ces différences sont limitées et sont, pour la plupart d'entre elles, invisibles à l'oeil humain sans l'utilisation d'un zoom. Cela dépend toutefois de la matrice de quantification qui est générée en fonction du taux de qualité souhaité.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 0 | 19 | -4 | -6 | -1 | 0 | 0 | 0 | 0 |
| 1 | -8 | 10 | -1 | -2 | -1 | 0 | 0 | 0 |
| 2 | 2 | 2 | 1 | 0 | 0 | 0 | 0 | 0 |
| 3 | 2 | -1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

(a) Matrice quantifiée




(b) Lecture des données de la matrice en zigzag

FIGURE 2.15 – Matrice résultante de l'algorithme de compression DCT

Toutes les données sont ensuite sérialisées dans le fichier.
Pour ce faire, on commence par l'élément se trouvant à la position (0,0) et on zigzague tout au long de la matrice comme montré sur la figure 2.15b.
Les données enregistrées dans le fichier pour ce bloc sont donc :

19 -4 -8 2 10 -6 -1 -1 2 2 1 -1 1 -2 0 0 -1 0 0 0 0 0 0 0 + 40 zéros.

L'importance de le faire en zigzague est de regrouper un maximum de zéro afin d'utiliser un second algorithme de compression tel que le MTF, RLE ou encore Huffman.

L'image étant maintenant compressée, on va ensuite la reconstruire.

Pour ce faire, le chemin inverse consiste à récupérer la matrice obtenue lors de la compression représentée sur la figure 2.15a et de la dé-quantifier.
Cette étape consiste à multiplier toutes les valeurs de la matrice quantifiée et de la matrice de quantification se trouvant aux mêmes positions.
Cette étape nous donne la matrice des coefficients représentée sur la figure 2.16.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 0 | 304 | -48 | -84 | -14 | 0 | 0 | 0 | 0 |
| 1 | -88 | 120 | -13 | -34 | -22 | 0 | 0 | 0 |
| 2 | 20 | 28 | 16 | 0 | 0 | 0 | 0 | 0 |
| 3 | 32 | -19 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 24 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

FIGURE 2.16 – Matrice des coefficients calculée depuis la matrice quantifiée (figure 2.15a).

L'avant-dernière étape consiste à faire l'inverse de la transformée en cosinus discrète et en utilisant la formule suivante :

$$IDCT(i,j) = \frac{1}{\sqrt{2N}} \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} F(i,j)C(i)C(j)cos(\frac{(2i+1)x\pi}{2N})cos(\frac{(2j+1)y\pi}{2N})$$

Avec
- N : la largeur du bloc, dans notre exemple 8
- i, j : les indices du coefficient de la DCT

— x, y : les indices du pixel de l'image (dans le bloc)
— IDCT(i,j) : la valeur de l'intensité du pixel centrée sur zéro
— F(x,y) : la valeur au position x, y de la matrice des coefficients
— C(x) = $\frac{1}{\sqrt{2}}$ si x = 0, sinon C(x) = 1

On obtient ainsi la matrice centrée sur zéro tel que visible sur la figure 2.17a.

La dernière étape consiste à récupérer l'intensité de la couleur en ajoutant 128 (256/2) ; ces mêmes 128 que l'on avait retiré lors de la deuxième étape.

La figure 2.18b représente l'intensité de chaque pixel ; ce sont ses intensités qui sont utilisées pour recréer l'image visible sur la figure 2.18b.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 0 | 39 | 49 | 57 | 52 | 36 | 17 | 6 | 3 |
| 1 | 28 | 38 | 47 | 45 | 32 | 16 | 6 | 3 |
| 2 | 15 | 25 | 37 | 39 | 31 | 20 | 12 | 9 |
| 3 | 9 | 20 | 34 | 41 | 40 | 34 | 28 | 26 |
| 4 | 10 | 22 | 37 | 49 | 54 | 53 | 50 | 48 |
| 5 | 12 | 23 | 40 | 56 | 66 | 69 | 67 | 64 |
| 6 | 10 | 20 | 38 | 56 | 70 | 75 | 74 | 71 |
| 7 | 6 | 16 | 34 | 54 | 69 | 76 | 74 | 71 |

(a) Matrice reconstruite centrée sur zéro

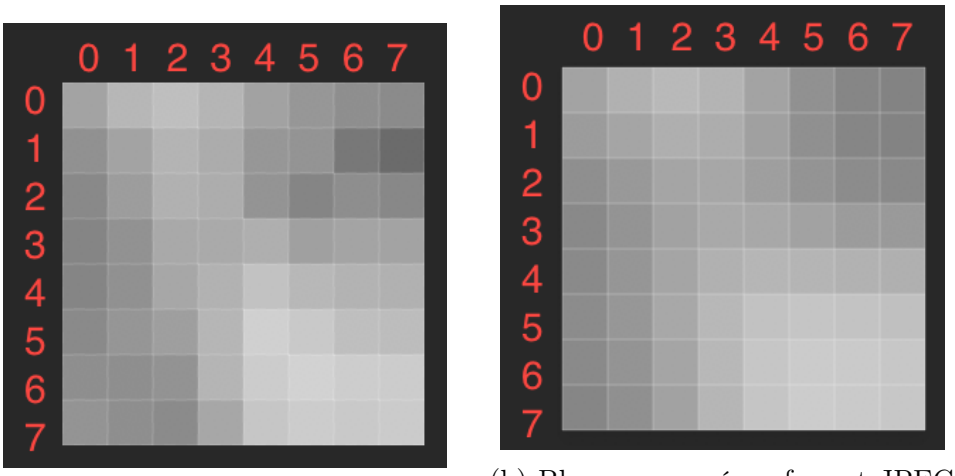
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 0 | 167 | 177 | 185 | 180 | 164 | 145 | 134 | 131 |
| 1 | 156 | 166 | 175 | 173 | 160 | 144 | 134 | 131 |
| 2 | 143 | 153 | 165 | 167 | 159 | 148 | 140 | 137 |
| 3 | 137 | 148 | 162 | 169 | 168 | 162 | 156 | 154 |
| 4 | 138 | 150 | 165 | 177 | 182 | 181 | 178 | 176 |
| 5 | 140 | 151 | 168 | 184 | 194 | 197 | 195 | 192 |
| 6 | 138 | 148 | 166 | 184 | 198 | 203 | 202 | 199 |
| 7 | 134 | 144 | 162 | 182 | 197 | 204 | 202 | 199 |

(b) Matrice reconstruite représentant les intensités de couleurs

FIGURE 2.17 – Matrice résultante de l'algorithme de compression DCT

L'analyse de l'image d'origine (figure 2.18a) et l'image reconstruite (figure 2.18b) montre une diminution du détail de l'image.
L'image compressée a perdu en qualité et est plus lissée dû à l'utilisation de la matrice de quantification (JPEG avec une qualité de 50%).
Ceci a eu pour effet de supprimer les coefficients utiles pour la haute fréquence (en bas à droite de la matrice).

Le traitement a été effectué sur un seul bloc de 8*8 pixels, et doit être exécuté sur tous les autres blocs de l'image. Cette image d'exemple comporte 33040 blocs de 8x8 pixels.




(a) Bloc d'origine au format TIFF

(b) Bloc compressé au format JPEG avec une qualité de 50%

FIGURE 2.18 – Comparaison visuelle du bloc de pixel d'exemple

## 2.2 Lien entre algorithme et format de fichier

La table 2.4 lie les différents algorithmes au format de compression correspondant. Certains de ces formats utilisent plusieurs algorithmes tels que Bzip2 (qui combine Huffman et MTF) ou encore JPEG (qui utilise Huffman, RLE et DCT).
L'algorithme DEFLATE, utilisé entre autre par les formats ZIP et GZIP, est une combinaison de Huffman et de LZW.
L'algorithme DCT, avec perte d'information, est utilisé par les formats MP3 et JPEG, ce qui signifie que lors de la compression des données de ces formats, l'algorithme supprime les données qui sont considérées comme les moins pertinentes ce qui provoque une diminution de la qualité.

## 2.3 Comparaison

Cette partie compare d'une part, les taux de compression obtenus avec différents formats de compression, et d'autre part, les différents logiciels de compression de données.

Il existe différents articles comparant les algorithmes de compression comme, par exemple, l'article "Comparative Study between Various Algorithms of Data Compression Techniques"[3] qui compare les méthodes de compression entre LZW et Huffman.

| Format | Huffman | LZW | DEFLATE | RLE | MTF | DCT |
|---|---|---|---|---|---|---|
| ZIP | | | X | | | |
| GZIP | | | X | | | |
| BZIP2 | X | | | | X | |
| 7-Zip | | X[1] | | | | |
| GIF | | X | | | | |
| BMP | | | | X | | |
| PNG | | | X | | | |
| JPEG | X | | | X | | X |
| MP3 | X | | | | | X |

[1] Une variante de LZW

TABLE 2.4 – Les types d'algorithmes utilisées suivant les formats de compression

Cet article montre les similitudes entre ces deux méthodes comme un taux de compression très proche lorsque les fichiers sont de type texte.
En revanche, lorsque le type de fichier est binaire ou est une image BMP, la méthode LZW donne un meilleur taux de compression contrairement aux images GIF ou JPEG pour lesquelles, ces deux méthodes se montrent inefficaces.
En effet, on peut le constater dans cet article que les taux obtenus sur un lot de 6 images GIF et JPG varient de -43% à -16% avec le LZW et de -9% à -1% avec la méthode de codage Huffman. Les taux négatifs signifient que l'image compressée est plus volumineuse que l'image d'origine.

### 2.3.1 Comparaison des performances

#### Méthodologie

Dans le cadre de la comparaison des performances, 13 types de fichier ont été analysés. Ils correspondent à des fichiers textuels, binaires, images, vidéos et musicaux.
La somme des tailles des fichiers de chaque type est d'environ 5,2 Mo. Il a été choisi d'avoir un taille de fichier équivalant pour les différents types de données. Cela permet de réaliser une analyse sur les performances aussi bien sur un type de fichier en particulier que sur ensemble de données de type confondu.

Les paramètres de compression ont été définis pour obtenir le taux de compression le plus élevé possible dès que l'application le permet.

Du fait de ces paramétrages, le temps nécessaire pour la compression n'a pas été pris en compte.
En effet, en poussant la compression à son maximum, le temps nécessaire au traitement est accru.

| Types | Nombre de fichiers | Taille brute | 7z | Bzip2 | RAR | Zip | Taille compressée médiane | Taux de compression médian |
|---|---|---|---|---|---|---|---|---|
| *.avi | 16 | 5 261 152 | 4 524 067 | 4 720 926 | 4 634 009 | 4 701 064 | 4 645 017 | 11,7 % |
| *.dll | 26 | 5 254 220 | 1 543 179 | 2 095 832 | 1 693 150 | 2 155 923 | 1 872 021 | 64,4 % |
| *.doc | 138 | 5 254 656 | 147 690 | 573 721 | 173 313 | 1 009 814 | 476 135 | 90,9 % |
| *.exe | 24 | 5 254 056 | 3 910 541 | 4 273 885 | 3 948 241 | 4 135 619 | 4 067 072 | 22,6 % |
| *.gif | 246 | 5 246 209 | 4 620 354 | 4 896 084 | 4 639 881 | 5 270 565 | 4 856 721 | 7,4 % |
| *.html | 79 | 5 261 187 | 341 996 | 645 243 | 318 269 | 877 679 | 545 797 | 89,6 % |
| *.jpg | 44 | 5 246 116 | 4 770 061 | 4 743 918 | 4 780 095 | 4 799 508 | 4 773 396 | 9,0 % |
| *.mp3 | 29 | 5 250 432 | 5 053 813 | 5 069 593 | 5 081 085 | 5 101 205 | 5 076 424 | 3,3 % |
| *.mpg | 8 | 5 257 720 | 4 879 067 | 4 888 293 | 4 887 973 | 4 898 961 | 4 888 574 | 7,0 % |
| *.pdf | 36 | 5 257 876 | 4 258 863 | 4 444 829 | 4 258 775 | 4 599 883 | 4 390 588 | 16,5 % |
| *.txt | 8 | 5 253 436 | 1 270 884 | 1 531 448 | 1 318 381 | 1 839 080 | 1 489 948 | 71,6 % |
| *.wav | 1 | 5 256 024 | 3 670 225 | 3 771 508 | 2 657 731 | 4 450 719 | 3 637 546 | 30,8 % |
| *.zip | 19 | 5 262 680 | 5 226 742 | 5 238 677 | 5 202 579 | 5 264 564 | 5 233 141 | 0,6 % |
| **Total** | **674** | **68 315 764** | **44 217 482** | **46 893 957** | **43 593 482** | **49 104 584** | **45 952 376** | **32,7 %** |

Toutes les données sont en octet

FIGURE 2.19 – Comparaison suivant le format d'archive[2]

### Analyse

Le premier point de l'analyse porte sur la dernière ligne de la figure 2.19. Cette ligne donne la taille totale des différents fichiers et de tous types confondus.

En partant avec une taille brute de 68,3 Mo, on arrive, suivant le type de compression, à des tailles variant de 43,6 Mo pour RAR à 49,1 Mo pour Zip. Ce qui donne une différence de 5,5 Mo soit 8,4% entre la meilleure et la moins bonnes des méthodes de compression reprises dans cette figure.

De manière générale, les types de compression 7z et RAR sont les plus performants. À eux deux, ils obtiennent les meilleurs taux de compression à

savoir, 12 fois sur les 13 types analysés et 8 fois pour 7z.
Le moins bon format de compression étant Zip avec le moins bon taux de compression à 12 reprises.
La couleur verte permet de visualiser directement quel type d'archive est le plus performant en fonction du type de fichier en entrée.

Il est nécessaire, ensuite, de discuter des différents taux de compression en fonction des types de fichiers.
Les taux de compression de fichiers déjà compressés sont faibles comme le montre les types MP3 et MPG avec des taux de compression de moins de 10%.
Le format Zip, quant à lui, a un taux de compression moyen de 0,6% et devient même négatif lorsqu'on essaie de compresser un fichier ZIP en ZIP.
Cela s'explique par le fait qu'il y a un ajout de méta-données au fichier qui est nécessaire pour la décompression.
D'un autre côte, il y a des types de fichiers qui se compressent très bien comme les fichiers TXT et HTML avec des taux de compression supérieurs à 70%.

En conclusion, si les données sont de types variées, il est préférable d'utiliser les types de compression 7z ou RAR qui se montrent les plus performants.
Dans le cas de la compression d'une grande masse de données d'un même type, il est utile d'utiliser le format de compression le plus performant pour ledit type.
Par exemple, l'utilisation du type de compression 7z pour archiver les fichiers binaires DLL.

### 2.3.2 Comparaison des logiciels de compression de données

**WinZip** [8] est un logiciel de référence pour la compression des gros fichiers et est particulièrement adapté aux entreprises et aux utilisateurs ayant des besoins importants d'archivage et de compression.
Le logiciel est payant et propose une version d'essai gratuite limitée dans le temps dans laquelle certaines fonctionnalités avancées sont désactivées tel que l'ajout d'un filigrane sur les images et document PDF.
WinZip archive et compresse les fichiers au format ZIP et propose 5 niveaux de compression. Il possède également une interface facile à utiliser.

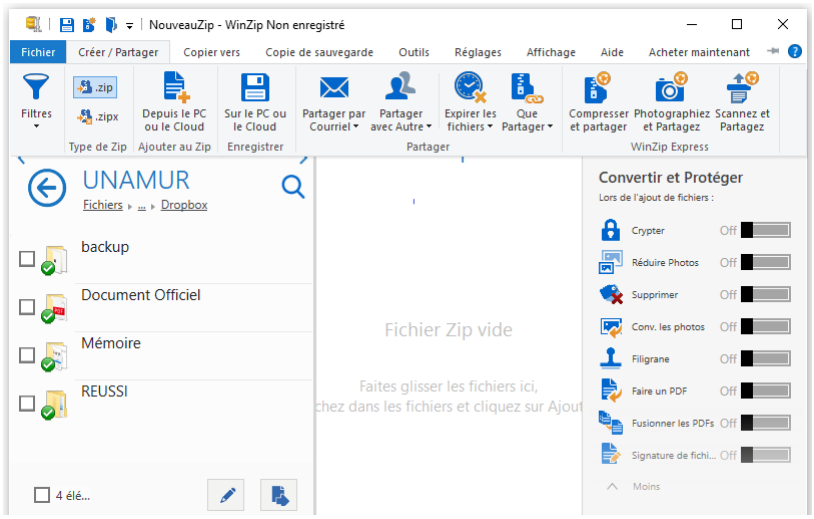



FIGURE 2.20 – Logiciel WinZip

**WinRAR** [11] est l'outil de compression le plus téléchargé et possède plus de 500 millions d'utilisateurs dans le monde.
La caractéristique principale de ce logiciel est l'utilisation de son propre algorithme de compression propriétaire, le format RAR.
Bien que d'autres logiciels peuvent décompresser le format RAR, il est le seul à pouvoir en faire la compression. Cet outils de compression est payant et propose une version d'essai gratuite limitée dans le temps.
Le délai dépassé, le logiciel reste utilisable mais une fenêtre pop-up s'affiche lors de chaque utilisation.
Bien que destiné pour Windows, il existe une version en ligne de commande pour MacOS.

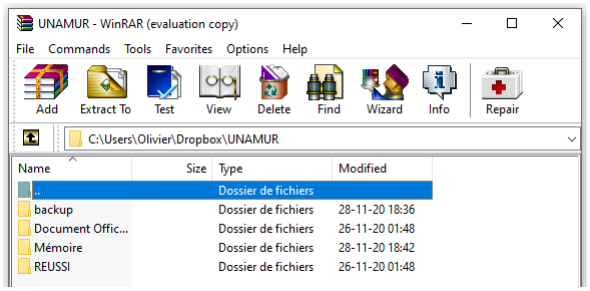



FIGURE 2.21 – Logiciel WinRAR

**7-Zip** [29] fait partie des plus anciens logiciels de compression de données open source et est entièrement gratuit.

En plus de la prise en charge du format ZIP, il dispose de son propre format 7Z qui présente des avantages en matière de taux de compression et de vitesse d'enregistrement.
Contrairement à d'autres logiciels, en général payant, 7-Zip ne dispose pas des fonctionnalités avancées et se concentre sur le strict nécessaire concernant la gestion des archives et de la compression.

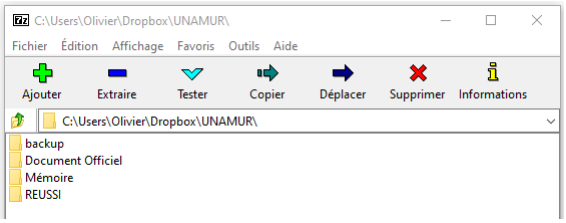



FIGURE 2.22 – Logiciel 7-Zip

**PeaZip** [34] est basée sur certaines technologies provenant de 7-Zip et offre une interface plus moderne avec des menus explicites.
En plus de la version standard, il existe une version portable autonome (à copier sur une clé USB, par exemple).
Ce logiciel, totalement gratuit et open-source, supporte une dizaine de formats pour la compression et un très grand nombre de formats pour la décompression. Ce logiciel permet l'utilisation de scripts pour automatiser, par exemple, des tâches de sauvegarde.

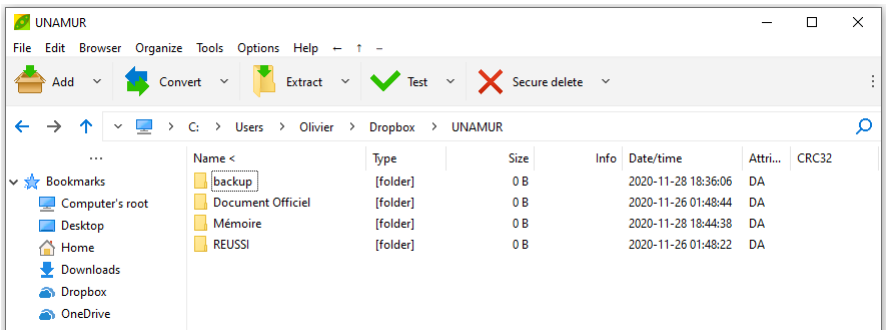



FIGURE 2.23 – Logiciel PeaZip

La figure 2.24 compare les 4 logiciels de compression de données (cités ci-dessus) en se basant sur différents critères comme par exemple, le prix ou les formats de compression et décompression supportés.

Ces quatre logiciels permettent la décompression de tous les types de formats les plus couramment utilisés (Zip, RAR, 7z, Bzip2, Gzip).
Côté compression, les logiciels gratuits proposent plus de types de compression que les logiciels payants qui se limitent à des formats précis.
Néanmoins, il faut garder en tête que les logiciels payant proposent, en général, plus de fonctionnalités que leurs homologues gratuits.

Les quatre logiciels analysés fonctionnent sur Windows.
7-Zip et Winzip possèdent une version complète pour MacOs tandis que WinRAR propose uniquement une version en ligne de commande.
Seuls les deux logiciels gratuits, 7-Zip et PeaZip, sont disponibles pour le système d'exploitation Linux.

Tous ces logiciels sont assez simples d'utilisation et en particulier WinZip qui propose une interface soignée nouvelle génération (figure 2.20) tandis que WinRAR (figure 2.21) et 7-Zip (figure 2.22) possèdent des interfaces plutôt anciennes.

| Logiciel | Système d'exploitation | Prix | Facilité d'utilisation | Protection des archives | Format compression / décompression | Format décompression |
|---|---|---|---|---|---|---|
| **WinZip** | Windows, MacOS | 29,95 € HT** | 5 / 5 | Mot de passe | zip | bzip2, rar, gzip, 7z, tar, cab, z, lha, iso, xz, vhd, vmdk |
| **WinRAR** | Windows, MacOS* | 25,95 € HT** | 4 / 5 | Cryptage donnée | rar, zip | 7z, bzip2, gzip, cab, arj, lzh, tar, uue, iso, z |
| **7-Zip** | Linux, MacOS, Windows | Gratuit | 3 / 5 | Cryptage donnée (7z, zip) | zip, 7z, bzip2, gzip, xz, tar, wim | rar, arj, cab, chm, cpio, deb, dmg, hfs, iso, lzh, lzma, msi, nsis, rpm, tgz, xar, z |
| **PeaZip** | Linux, Windows | Gratuit | 4 / 5 | Mot de passe, Fichier clef | zip, 7z, bzip2, gzip, br, pea, tar, xz, wim, zpaq, lpaq | rar, ace, arj, cab, ohm, cpio, deb, ear, iso, jar, lzma, lzh, nsis, pak, rpm, smzip, u3p, war, xpi, z, zipz |

* Uniquement en ligne de commande
** Version d'essai gratuite

FIGURE 2.24 – Comparatif de logiciels de compression de donnée

# Chapitre 3

# Déduplication

La déduplication de données est une technique qui permet de réduire les besoins de stockage en éliminant les données redondantes ou dupliquées dans un environnement de stockage.
Une seule et unique copie des données est conservée sur le support de stockage et les données redondantes ou dupliquées sont remplacées par un pointeur afin d'avoir une référence vers la copie conservée.

La figure 3.1 montre différents objets reconnaissables par leurs formes et leurs couleurs.
La partie de gauche représente les objets dans leur contexte d'origine, c'est-à-dire qu'il existe plusieurs instances de même objets.
La partie de droite représente le résultat du processus de déduplication.
On constate qu'une seule instance de chaque objet y est stockée tandis que les instances redondantes ne sont plus présentes. Il existe toutefois des pointeurs, non représentés sur la figure, qui permettent de lier les fichiers redondants à l'objet identique.

En conclusion, la déduplication de données consiste à comparer des fichiers ou des blocs d'informations binaires et de supprimer toutes les redondances afin de garder une instance unique.

## 3.1 À quoi sert la déduplication ?

La déduplication de données est une technologie très attirante pour le stockage des données des entreprises car elle permet de leur épargner des coûts importants.

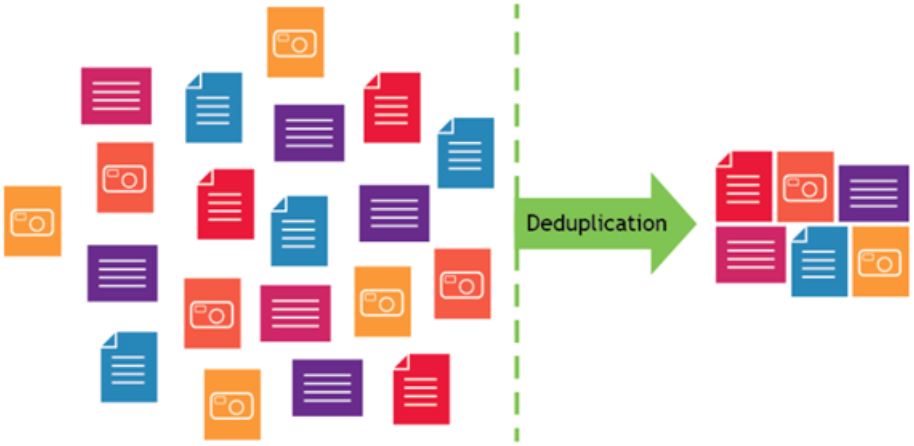



FIGURE 3.1 – La déduplication de données[18]

Pour mettre en lumière l'utilité de la déduplication, l'exemple d'une grande entreprise sauvegardant le contenu de milliers d'ordinateurs sera parlant.

Sur chaque ordinateur, qu'il soit portable ou fixe, on retrouve souvent du contenu identique, comme par exemple, la suite bureautique qui est généralement la même pour tous les employés d'une entreprise.
En effet, de nombreux fichiers identiques, comme par exemple l'exécutable du programme Winword.exe pour la suite de Microsoft Office, sont sauvegardés autant de fois qu'il y a d'ordinateur.
De ce fait, si une entreprise dispose de 1000 ordinateurs, le processus de sauvegarde va enregistrer à 1000 reprises ce même fichier exécutable.

Avec la mise en place de la déduplication, une seule instance de ce fichier sera enregistrée permettant d'obtenir un rapport de 1000 : 1 et ainsi de réduire l'espace de stockage nécessaire à cet effet.
En extrapolant sur tous les fichiers contenus dans les ordinateurs de bureau, les ordinateurs portables et serveurs de toute une entreprise, le gain en termes de coûts est conséquent notamment en réduisant le nombre de disques durs et de bandes magnétiques pour la sauvegarde.
Cela permet également d'épargner un pourcentage important de bande passante du réseau lors de la réplication de données sur une autre infrastructure informatique de l'entreprise pour, par exemple, disposer d'une solution de secours en cas de sinistre grave.

## 3.2 Où s'opère la déduplication ?

La déduplication des données peut être effectuée soit chez le client, soit au niveau de la baie de stockage [1] / VTL [2], ou entre les deux sur une appliance [3].[24, 43]

### 3.2.1 Côté client

La déduplication sur les machines clientes utilise l'approche client-serveur dans laquelle le client communique avec un serveur, une appliance ou directement avec la baie de stockage.
Ce type de déduplication est désigné comme la transmission de données dédupliquées dans laquelle le client traite les données afin d'en extraire les méta-données et les transmet au noeud de stockage.
Ce dernier utilise les méta-données et sa propre banque de données contenant les données dédupliquées afin de déterminer si une redondance existe et transmet le constat au client correspondant.
Dés lors, le client ne transmet que les données uniques sur la Fiber Channel [4] ou sur le réseau IP.

La déduplication côté client offre d'énormes économies en bande passante sur le réseau mais évidemment, ces économies ont un coût.
Premièrement, le processeur de la machine cliente est sollicité pour détecter les doublons.
Deuxièmement, la sécurité peut-être affaiblie puisque tout client peut interroger la baie de stockage avec des méta-données pouvant entraîner un problème de fiabilité.
Finalement, le client, lançant sa déduplication, entraîne, côté serveur, l'exécution d'une tâche permettant la détection de doublons.
Le lancement de cette tâche peut perturber les performances d'autres processus en cours d'exécution sur le serveur.

1. Baie de stockage : équipement de sauvegarde de données informatique composé de disques de stockage, d'un bus qui est l'élément par lequel la baie va communiquer et d'un processeur permettant de traiter toutes les informations.
2. VTL : "Virtual Tape Library" | en français : "bibliothèque de bandes virtuelles"
3. Une appliance est un terme anglo-saxon. C'est un type de matériel similaire à un serveur mais de plus petite taille et qui exécute une tache dédiée comme la déduplication, le filtrage d'accès Internet, le filtrage des mails,...
4. Fibre Channel est un protocole défini par la norme ANSI X3T11 permettant une connexion haut débit entre un ordinateur et un périphérique comme son système de stockage.

### 3.2.2 Côté appliance

Une appliance de déduplication est un système dédié mettant en oeuvre la déduplication.
Typiquement, ces boîtiers de déduplication opèrent à travers deux modes de fonctionnement et sont connectés entre les clients et la baie de stockage.

Le premier, appelé in-band, examine toutes les données entrantes et vérifie l'existence de doublons avant d'écrire les données sur la baie de stockage.

Le second, appelé out-of-band, exécute la déduplication lorsque les données sont écrites sur la baie de stockage.

L'inconvénient de l'utilisation du mode in-band est que l'appliance pourrait être le goulot d'étranglement en raison du traitement supplémentaire requis par le système avant l'écriture des données sur le disque.

### 3.2.3 Côté baie de stockage ou VTL

Les contrôleurs de baie de stockage et les contrôleurs de VTL (bibliothèque de bandes virtuelles) fournissent une très bonne plate-forme pour effectuer la déduplication.

Les contrôleurs de baie de stockage modernes offrent généralement de grandes ressources de calculs et leur proximité avec les données facilite la mise en place de la déduplication.
Afin d'obtenir un meilleur taux de déduplication au sein d'une baie de stockage, cette dernière doit être en mode bloc.

Un VTL est un système informatique incluant un serveur, une grappe de disques et un logiciel capable d'émuler cet espace disque en bande magnétique.
Son utilisation est généralement liée au système de sauvegarde d'une entreprise, ses contrôleurs sont aussi très performants.

## 3.3 Quand s'opère la déduplication ?

La déduplication existe à travers deux modes de fonctionnement[25] : le mode synchrone et le mode asynchrone.

### 3.3.1 Le mode synchrone

Le mode synchrone implique l'exécution de la déduplication lors de toute demande d'enregistrement de données.
En effet, lors de chaque demande d'écriture sur la baie, le système de stockage essaye de dédupliquer les données avant d'écrire les données entrantes sur les disques.


FIGURE 3.2 – Le mode synchrone[25]

La déduplication synchrone peut ajouter une quantité importante de temps de latence sur le système.
Ceci est dû au contrôle de vérification de doublons des données en cours de traitement.
Ce mode de fonctionnement permet également à la partie cliente de surcharger la baie de stockage suite à des demandes intempestives.

### 3.3.2 Le mode asynchrone

Le mode asynchrone consiste à effectuer l'opération de déduplication à intervalles réguliers ou lorsque l'utilisation de l'espace de données dépasse une limite fixée.

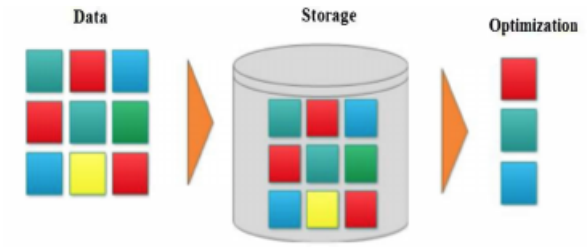



FIGURE 3.3 – Le mode asynchrone[25]

Ce procédé peut s'avérer désirable dans des systèmes où la vitesse d'ingestion est primordiale mais cela engendre un nombre important d'input/output[9]. De ce fait, les données, se trouvant sur une baie de stockage, doivent être entièrement relues afin de détecter des octets redondants.
Cela entraîne un nombre d'écritures plus important que le mode synchrone. Cependant, ce mode nécessite un espace de stockage plus important pour enregistrer les données car elle ne sont pas dédupliquées immédiatement.

La déduplication asynchrone limite les possibilités d'actions.
Elle est moins performante si elle se réalise côté client car les méta-données ne sont pas continuellement mises à jour.

## 3.4 Architecture d'un système de déduplication

L'architecture d'un système de déduplication[13] est illustré sur la figure 3.4.

Lorsqu'un flux de données entre dans le système, il passe par une première couche de service de fichiers tel que NFS qui gère les méta-données et l'espace de nom.
Cette première couche transfère la requête d'écriture vers la couche "Content Store Manager".

Le "Content Store Manager" gère le contenu des données repris dans le flux.
Il fractionne le flux de données en plusieurs segments et fait appel à la couche suivante, le "Segment Store Manager".

Le "Segment Store Manager" a pour rôle d'effectuer le travail de déduplication et d'enregistrer les segments uniques sur un support.
Il peut également faire un appel à un système de compression pour augmenter le gain d'espace, puis écrit les informations compressées dans un espace géré par le "Container Manager".

Ces trois composants se situent en mémoire vive ou sur des disques SSD.

Pour lire un flux de données dédupliquées, le "Content Store Manager" utilise les références des segments dédupliqués afin de délivrer le flux de données souhaitées.

Le "Segment Store Manager" réalise une pré-lecture, décompresse si besoin, et met en cache les segments de données.

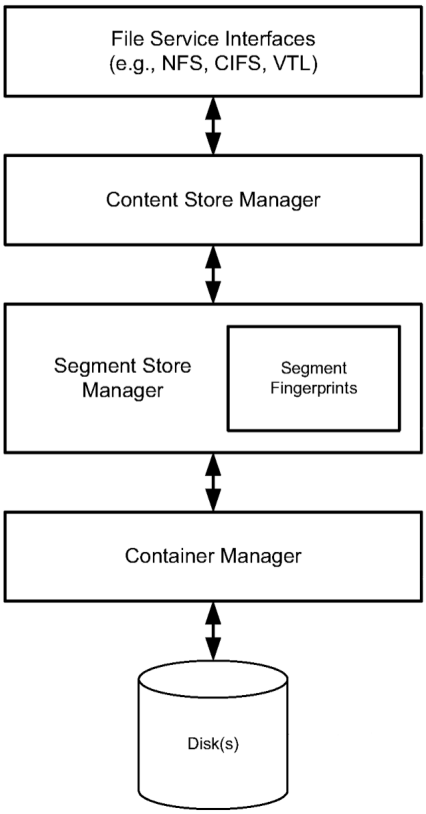



FIGURE 3.4 – Architecture d'un système de déduplication[13]

### 3.4.1 Content Store Manager

Le processus, lié à ce Content Store, doit effectuer plusieurs opérations :

— la première divise le fichier de données en une série de segments de tailles fixes ou variables ;
— la seconde calcule une signature à l'aide d'une fonction de hachage pour chaque segment et crée un descripteur de segment.
Le descripteur de segment contient la signature issue de la fonction de hachage ainsi que la taille du segment ;
— La dernière construit une cartographie à l'aide d'un arbre qui enregistre le lien entre les segments de données et les descripteurs de segments.

Le but est de représenter le fichier de données à l'aide de références et de segments de données.

### 3.4.2 Segment Store Manager

Le "Segment Store Manager" contient essentiellement une base de données de segments indexés par leur descripteur de segment.

Lors de la lecture d'un segment, le "Segment Store Manager" le récupère depuis le container dans lequel il est stocké.

Lors de l'écriture, le "Segment Fingerprints" détermine si un segment est un doublon à l'aide de la signature calculée avec la fonction de hachage[31]. Dans le cas d'un nouveau segment, celui-ci est intégré dans un container.

Le "Segment Fingerprints" est une étape primordiale et doit être efficace pour dédupliquer les données et ainsi limiter les entrées et sorties au niveau des disques.

### 3.4.3 Container Manager

Le "Container Manager" est responsable de l'allocation, la dislocation, la lecture et l'écriture et se doit d'être efficace.
Un container comprend une partie de méta-données comportant des descripteurs de segments sur les segments uniques enregistrés dans celui-ci.

Les containers peuvent être ajoutés ou supprimés mais pas modifiés.
Lorsqu'un Segments Store ajoute un container, le "Container Manager" transmet un identifiant unique.

Les segments uniques ne sont pas directement écrits sur le support de stockage.
En effet, ils ne sont pas assez grands pour atteindre des performances élevées d'écriture.
Ils sont organisés dans un container de taille fixe.
Lorsque le container est plein, il est enregistré sur le support de stockage.
Pour lire un segment, le container correspondant à ce dernier est lu.

## 3.5 Processus de déduplication

Le processus de déduplication s'applique sur des fichiers ou sur des blocs[24, 17].
Les blocs sont utilisés pour stocker des données directement sur les disques durs d'une baie de stockage par exemple, et les fichiers sont utilisés pour stocker des données dans un système de fichiers tel que NTFS ou FAT [5].
Lorsque la déduplication est appliquée au niveau bloc, l'utilisation des données peut-être structurée ou non structurée.

Les données sont structurées lorsqu'elles suivent une organisation permettant de simplifier leurs traitements.
Différentes structures de données existent tels qu'une base de données, un tableau indexé ou encore une structure sous forme de liste.

À contrario, les données écrites dans un fichier sont des données non structurées.
Lorsque la déduplication est appliquée au niveau fichier, il suffit de comparer les fichiers ou le contenu et de supprimer les doublons.

Les différents processus de déduplication proposent des approches différentes ce qui influent sur les performances.
Lorsque cette approche s'applique aux niveaux des fichiers, le traitement est plus rapide mais le gain d'espace est moindre que si elle s'appliquait aux niveaux du bloc de données[27, 23].

### 3.5.1 Par fichiers

Lorsque la déduplication est appliquée au niveau fichier, il suffit de comparer les fichiers ou leurs contenus et de supprimer les doublons.
L'utilisation d'un système de fichiers, pour comparer si deux fichiers sont identiques, est la méthode la plus simple et la moins onéreuse pour éliminer les répétitions.
Il est également facile de déterminer une réplication d'un fichier à l'aide des méta-données.
Celles-ci contiennent, entre autre, le nom du fichier, sa taille, la date de sa création et la date dernière modification.

5. FAT (File Allocation Table) : Table où le système d'exploitation enregistre l'emplacement des différents dossiers et fichiers enregistrés sur le disque dur.

Un exemple de cette méthode est de comparer le nom, la taille, le type et la date de dernière modification de deux fichiers.
Si toutes ces méta-données sont identiques, il existe une forte probabilité pour que ces deux fichiers soient des doublons.
Si le système détecte qu'il s'agit de doublons, il crée un pointeur vers le fichier unique.
Dans le cas contraire, l'instance est enregistrée sur le serveur de stockage.

Cette méthode n'est pas aussi infaillible que les solutions offertes par les outils professionnels de déduplication.
Cependant, cette solution est simple à mettre en place via un petit script et a le mérite d'être gratuite.
Il est également possible d'ajouter une solution de compression de données afin d'augmenter le gain de réduction sur l'espace de stockage.

L'utilisation d'une fonction de hachage, pour déterminer la redondance de fichiers, augmente l'intelligence et l'efficacité de la déduplication.
La fonction de hachage calcule la signature de chaque fichier entrant.
Celle-ci permet au système de la comparer, à l'aide d'une table de hachage, aux signatures déjà présentes dans le but de détecter d'éventuels doublons.

La figure 3.5 explique le déroulement du processus de déduplication par fichier.

Lorsqu'un fichier entre dans le processus de déduplication, sa signature est calculée et comparée aux autres.
Si la signature calculée existe dans la table de hachage, alors le fichier n'est pas enregistré puisqu'il s'agit d'un doublon, et la référence vers l'unique copie de ce fichier est créée.
Dans le cas contraire, le fichier est enregistré et la table de hachage est mise à jour avec la nouvelle signature calculée.

Toutefois, la fonction de hachage peut produire une même signature pour deux fichiers différents.
Lorsqu'un faux positif se produit, le système n'enregistre pas le fichier entrant ce qui entraîne une perte de données.
Néanmoins, il existe une solution de comparaison des méta-données lorsque cet évènement se produit.

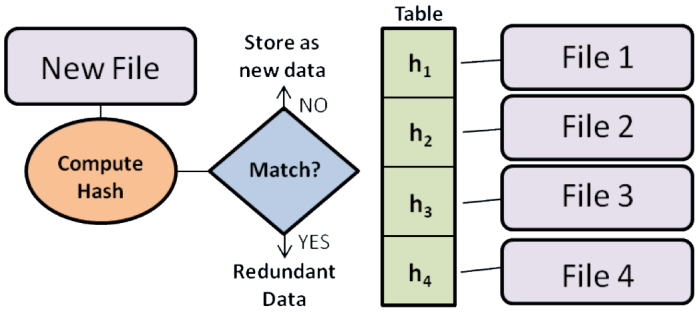



FIGURE 3.5 – Déduplication par fichiers[22]

La déduplication par fichier ne dispose pas d'un taux de déduplication important.
La comparaison est réalisée de manière stricte et uniquement par rapport aux méta-données des fichiers, le contenu des fichiers n'est pas analysé.
Il en résulte que lorsqu'un fichier est modifié, si petite la modification est, les méta-données de ce fichier changeront ce qui entraînera une modification de la signature du fichier.
Suite à cette modification, ce fichier est à nouveau enregistré.

### 3.5.2 Par segments

Lorsque la déduplication est appliquée au niveau des segments[17], il n'est pas nécessaire, contrairement à l'approche de déduplication par fichiers, de disposer des informations sur les fichiers ou même sur le système d'exploitation utilisé.

Il existe 2 types de déduplication au niveau segment. Le segmentation de taille fixe et la segmentation de taille variable.

#### La segmentation de taille fixe

La segmentation de taille fixe consiste à calculer une signature pour chaque segment et de la comparer à l'aide de la table de hachage, quelque soit le type de données.
Si la signature calculée correspond à une existante dans la table de hachage, alors le nouveau segment n'est pas enregistré puisqu'il est considéré comme doublon.

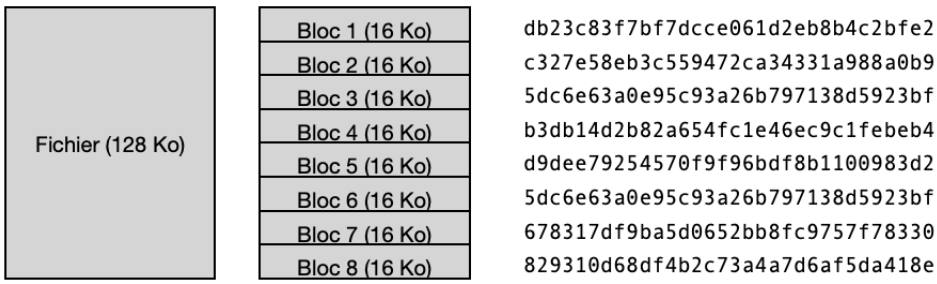



FIGURE 3.6 – Découpe d'un fichier en segments de taille fixe avec la signature des segments

La figure 3.6 présente la segmentation d'un fichier d'une taille de 128 Ko par 8 blocs de données de 16 Ko.
Le processus de déduplication calcule la signature de chaque bloc.

Les signatures sont enregistrées dans la table de hachage permettant de déterminer rapidement si les nouvelles signatures sont uniques.
La figure 3.6 montre l'existence d'une redondance. En effet, les blocs 3 et 6 possèdent une signature identique.
Le processus remplacera donc le bloc 6 par un pointeur vers le bloc 3 afin que les données originales puissent être reconstruites ultérieurement.
Le gain en termes d'espace de stockage pour cet exemple est de 16 Ko sans prendre en considération le pointeur créé.

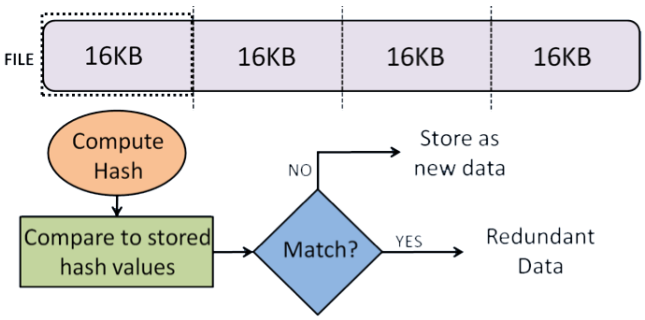



FIGURE 3.7 – Déduplication par segment de taille fixe[22]

L'avantage d'utiliser cette méthode est qu'elle permet d'obtenir un pourcentage plus important de réduction d'espace de stockage.
Toutefois, le nombre de traitement est plus important, par rapport à la méthode par fichier, et entraîne une augmentation du temps d'exécution.

Une problématique de la déduplication par segment de taille fixe survient lors de l'insertion de symboles comme présenté sur la 3ème colonne de la figure 3.8.
Cette insertion créée un décalage de données dans la segmentation et rend impossible la déduplication puisqu'ils ne sont plus identiques.

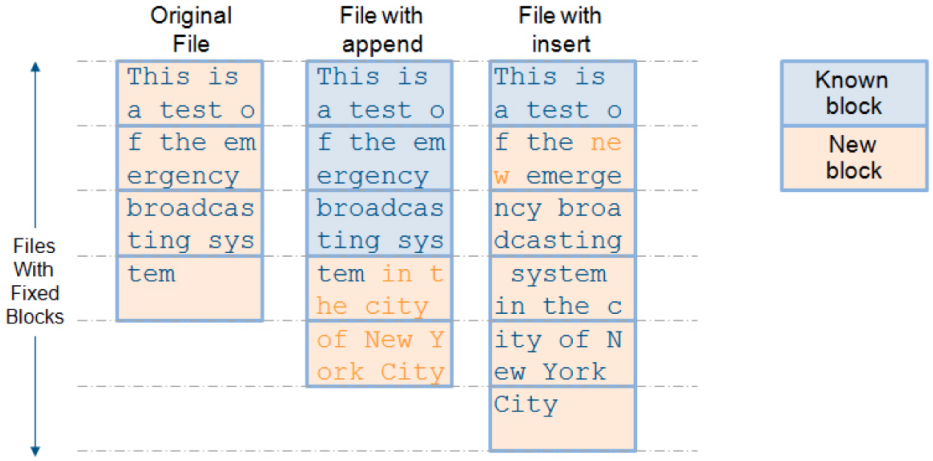



FIGURE 3.8 – Exemple : problématique de la déduplication par segment de taille fixe[35]

### La segmentation de taille variable

La segmentation de taille variable permet de surmonter le problème lié au procédé de déduplication par segments de taille fixe.
L'utilisation de la segmentation de taille variable est nécessaire.
En effet, cette méthode ne découpe pas le flux de données en morceaux de tailles fixes, ni de multiples tailles de longueurs prédéfinies.

Les CDC[6] sont des méthodes de séparation de flux de données de telle sorte que le contenu dupliqué dispose d'une plus grande probabilité d'être découvert, indépendamment de sa position dans le flux de données[21].
Le CDC se doit être évolutif et rapide pour la détection de doublons[16, 42].

6. CDC (Content defined Chuncking techniques) : méthodes servant à séparer un flux de données en des morceaux de tailles variables de façon à ce que le contenu dupliqué soit découvert sans tenir compte de sa position dans le flux de données.

Plus le processus de déduplication utilise des segments de données de petites tailles, plus la génération de méta-données sera importante et plus les performances du système de déduplication diminueront.

L'approche de CDC consiste à délimiter un segment à l'aide de deux tailles définies.
Le mécanisme commute dynamiquement entre ces deux tailles en fonction des données déjà enregistrées.

Comme le montre la figure 3.9, lors de chaque décalage sur le flux de données, le contenu de la fenêtre glissante est analysé et une signature est calculée.
Si la signature est connue, le processus augmente la longueur du segment.
Dans le cas contraire, la signature est ajoutée dans l'index et le segment est enregistré.
Quand le segment atteint sa longueur maximale, il existe une grande probabilité d'obtenir un segment dupliqué.

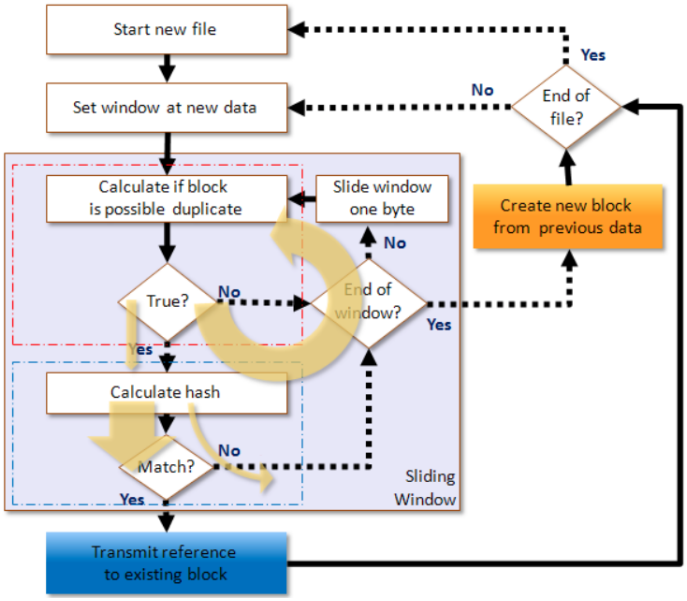



FIGURE 3.9 – Processus par segmentation de taille variable[35]

Pour éviter l'ajout d'un temps de latence supplémentaire dans le processus, la fonction de hachage, permettant de calculer les signatures, doit être efficace puisqu'elle est appelée lors de chaque décalage[16].

Afin d'éviter le calcul entier de la signature lors de chaque décalage, l'utilisation des fonctions de hachage par roulement est conseillée.
Ce mécanisme consiste à prendre en entrée la signature précédemment calculée et de la modifier en fonction des données issues du décalage ce qui lui permet d'être très rapide[31, 16].

## 3.6 Ratio de déduplication

Le ratio de déduplication (R) est la valeur qui mesure l'économie d'espace obtenue grâce aux mécanismes de déduplication.
Ce ratio est calculé en divisant le nombre d'octets entrés avant la déduplication ($O_e$) par le nombre d'octets sortis après la déduplication ($O_s$)[10].

$$R = \frac{O_e}{O_s}$$

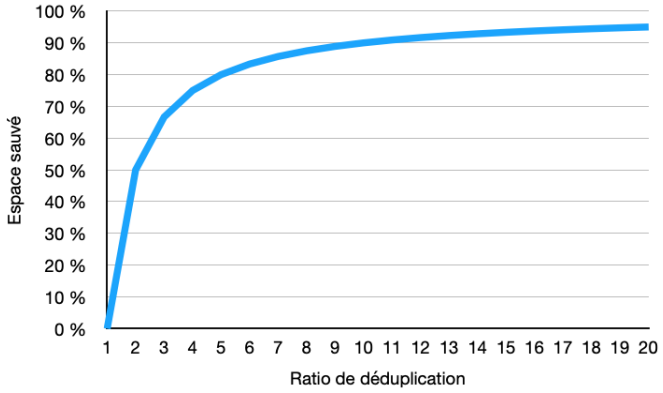
Le ratio de déduplication dépend du nombre d'octets redondants et éliminés à l'aide du processus de déduplication.
Le ratio peut être important ou superflu ; un ratio important indique une quantité importante de données redondantes tandis qu'un ratio superflu signifie qu'il y a peu de données redondantes.

Le ratio est généralement écrit comme ratio :1, ce qui indique que le nombre d'octets entrés a été transformé en 1 octet de sortie à l'aide du processus de déduplication.
Une autre façon d'exprimer l'économie d'espace disque est de transformer le ratio en pourcentage de réduction d'espace (P).

$$P = 1 - \frac{1}{R}$$

Le ratio ne dépend pas seulement de la méthode de déduplication utilisée, mais également d'autres facteurs tels que[10] :

— **Type de données :** Le type de données stockées par l'utilisateur à l'aide de leurs postes de travail est de loin le plus grand impact sur le ratio de déduplication.
En effet, si un groupe d'utilisateurs ne change que quelques parties d'un ensemble de fichiers tous les jours, l'effet de déduplication est certainement beaucoup plus élevé que si ils créent de nouveaux fichiers régulièrement.




| Ratio | Pourcentage |
|---|---|
| 1: 1 | 0 % |
| 2: 1 | 50 % |
| 4: 1 | 75 % |
| 5: 1 | 80 % |
| 10: 1 | 90 % |
| 20: 1 | 95 % |
| 100: 1 | 99 % |
| 500: 1 | 99,8 % |

FIGURE 3.10 – Ratio de déduplication : ratio et pourcentage[10]

— **La modification de données fréquentes :** La vitesse à laquelle les données sont modifiées sur les postes de travail des utilisateurs n'a pas une influence directe sur le ratio de déduplication, mais plutôt sur la probabilité de données redondantes à se produire.
En général, plus il y a de modifications, moins le ratio est élevé.

— **Période de rétention des données :** La durée pendant laquelle les données dédupliquées sont stockées peut affecter le ratio de déduplication.
En effet, plus la durée de rétention des données est importante, plus la quantité de données uniques s'accroît et plus la probabilité de trouver des doublons dans les nouvelles données augmente.

— **Taux de transfert de données élevé :** Lors d'une déduplication inline, des applications requièrent un débit très élevé de données, obligeant le processus de déduplication à traiter le flux de données entrant rapidement.
Pour éviter un goulot d'étranglement au niveau de la déduplication, le processus diminue la charge de calcul en réduisant la capacité de détecter les données redondantes en augmentant la taille des segments de données.
Cette approche entraîne une réduction du taux de déduplication des données.

## 3.7 Outils de déduplication

Cette section décrit 2 outils "populaires" qui utilisent la déduplication : Dell EMC PowerProtect DD et QoreStor.

Il en existe beaucoup d'autres tels que, par exemple, Oracle Greenbytes, Compellent, Veritas NetBackup Appliance, IBM ProtecTIER, Fujitsu Jujitsu Data Deduplication Appliance, etc...
Il existe également des outils open-source comme OpenZFS ou OpenDedup.

### 3.7.1 Dell EMC PowerProtect DD

L'appliance de stockage Dell EMC PowerProtect DD[14] permet aux entreprises de protéger, gérer et restaurer des données à grande échelle.
Elle s'intègre de manière transparente avec les infrastructures existantes et est conçue pour simplifier et optimiser l'efficacité opérationnelle de la protection des données pour les environnements multi-cloud.
Elle supporte la virtualisation et s'exécute dans VMWare, Microsoft Hyper-V, KVM ainsi que dans le cloud avec AWS, AWS GovCloud, VMWare cloud, Azure, Azure Government Cloud et Google Cloud Platform.
Cette appliance fait partie des meilleures du marché, cependant c'est également l'une des plus onéreuse.

Les principales caractéristiques et fonctionnalités de cette appliance sont :

- **Des performances optimisées** qui permettent l'ingestion allant de 7 To à 94 To de données par heure suivant la gamme choisie.

- **Une intégration transparente** afin de faciliter l'utilisation des principales applications de sauvegarde et d'archivage.

- **Une protection multi-cloud** qui permet de hiérarchiser nativement les données dédupliquées vers n'importe quel environnement cloud à des fins de rétention à long terme.

- **Une vérification des données de bout en bout** afin d'éviter la corruption de données en vérifiant l'intégrité des méta-données, l'intégrité des données, l'intégrité de la bande et vérifie également que la somme des contrôles des données lues correspondent à celle des données écrites.

— **Un accès instantané** grâce à de hautes performances pour les machines virtuelles.

— **Une restauration instantanée** grâce au déploiement immédiat de machines virtuelles orientées production au sein de l'appliance elle-même.

### 3.7.2 QoreStor

QoreStor[15] est une plate-forme professionnelle de stockage.
Elle s'appuie sur les technologies de déduplication et de réplication.
Elle est simple à déployer et à administrer.
La plate-forme supporte un large panel de solutions de sauvegarde et de stockage de grands fournisseurs dont Quest, CommVault, Dell/EMC, Veritas, Veeam et IBM.
De plus elle, supporte les grandes plateformes de virtualisation et les fournisseurs de services cloud tels que Azure, AWS, VMWare, Microsoft Hyper-V et KVM.

Les principales caractéristiques et fonctionnalités de QoreStor sont :

— **Un moteur de déduplication du stockage de nouvelle génération** ce qui réduit la capacité de stockage des sauvegardes de 20 :1 et tire profit de la déduplication en bloc variable.

— **Un accélérateur de protocoles intégrés** qui permet l'ingestion de 20 To de données par heure grâce à une technologie optimisée lors des opérations d'écriture côté client.

— **Réplication à distance pour la reprise après sinistre** avec la possibilité de répliquer uniquement les données modifiées sur un site distant, ce qui permet de réduire considérablement les besoins en bande passante.

— **Sauvegarde directe dans le cloud** grâce à la déduplication à la source, seuls les blocs modifiés sont sauvegardés dans le cloud.

— **Connexion sécurisée** qui élimine les sauvegardes incomplètes ou leur mises en échec suite à une déconnexion et applique une reprise automatique de la sauvegarde dès le rétablissement de la connexion.

— **La sécurité des données** par un chiffrement intégré réalisé en dehors des activités de lecture et d'écriture utilisant des clés AES [7] de 256 bits, générées une seule fois ou à intervalles réguliers.

7. Advanced Encryption Standart

# Chapitre 4

# Similitudes et différences entre compression et déduplication

La compression et la déduplication de données utilisent des procédés différents pour épargner l'espace de stockage.
Comme ils ne sont pas liés, ils peuvent être utilisés indépendamment ou, comme vu dans la section 3.4, conjointement pour optimiser le gain.
La déduplication est une forme de compression au niveau des segments et il existe des similitudes entre la compression et la déduplication.

L'algorithme LZ77, utilisé pour la compression, utilise une fenêtre glissante afin de détecter une redondance de symboles. De la même manière, la déduplication de taille variable utilise une fenêtre glissante pour déterminer des doublons à l'aide d'une fonction de hachage.

L'algorithme LZ78 utilise un dictionnaire dynamique reprenant les symboles dupliqués.
Parallèlement, la déduplication utilise une table de hachage dynamique pour détecter les doublons.

Les méthodes de compression utilisent généralement un dictionnaire dynamique grandissant tout au long du processus. La déduplication réalise un procédé similaire en gardant une signature de chaque segment unique.

Le codage de Huffman construit un arbre pour la compression et positionne les symboles les plus fréquemment utilisés proche de la racine permettant ainsi d'utiliser un codage plus court. Ce même principe est utilisé par la

déduplication en plaçant les données les plus fréquentes en mémoire vive afin de diminuer le temps de latence.

La compression utilise des algorithmes pour réduire la quantité d'espace physique que prend un fichier. Elle fait face à plusieurs désavantages :

— la modification des fichiers compressés nécessite plusieurs étapes : la décompression du fichier, la modification et ensuite la compression du fichier modifié,
— l'utilisation importante du processeur tout au long du traitement ce qui peut entraîner un temps de traitement important.

La déduplication réalise une réduction de la charge de stockage lorsqu'il y a une redondance dans le jeu de données. Ce qui signifie que l'ensemble des données doit être composé de plusieurs fichiers ou portions de données identiques. Elle a pour avantage de maintenir les formats de données et d'être transparente pour les utilisateurs.

# Chapitre 5

# Conclusion

L'augmentation du volume des données, générée entre autres par les appareils électroniques connectés, demande de plus en plus de place de stockage dans le Cloud. Ce Cloud est, quant à lui, stocké dans d'énormes centres de données qui doivent mettre en place des techniques afin de garder des hauts débits de transfert et de diminuer l'espace de stockage nécessaire.

À ce jour, il existe deux méthodes de réduction de données : la compression et la déduplication.

La compression est une technique plutôt ancienne, très populaire, qui est utilisée massivement de manière consciente (par exemple le ZIP pour l'envoi de mail) ou inconsciente (par exemple les images JPEG).
Elle permet de réduire le volume des données nécessaires pour un fichier.

La compression de données a fait l'objet du chapitre 2.
Celui-ci a décrit et a illustré le fonctionnement des principaux algorithmes de compression tel que, par exemple, le codage de Huffman, le "LZW" ou encore le "Deflate".

Nous avons analysé la compression d'une image au format JPEG, que nous avons réalisé à l'aide de l'algorithme DCT. Cet algorithme diminue la qualité de l'image afin de réduire la quantité de données nécessaires pour la reconstruction de celle-ci.

Après avoir détaillé les différents algorithmes, nous avons comparé les performances des différents formats de compression. Ensuite, nous avons réalisé une comparaison des logiciels de compression les plus populaires.

La déduplication, quant à elle, est une technique plus récente, elle nécessite une quantité importante de données pour avoir des résultats satisfaisants.
Elle permet de réduire la quantité de données en éliminant les doublons existants.

La déduplication a été détaillée dans le chapitre 3.
Dans ce chapitre, nous avons d'abord défini où et quand s'opère la déduplication, nous avons détaillé l'architecture du système et parcouru les 3 grands axes du processus de déduplication.
Ensuite, nous avons vu ce qu'est le ratio de déduplication et terminé le chapitre en parcourant quelques outils utilisant la déduplication.

Le nombre de recherches scientifiques concernant la déduplication ne cesse d'augmenter depuis quelques années. Cette technique va probablement beaucoup évoluer au cours de cette décennie.

Concernant la compression de données, le nombre d'articles scientifiques reste assez stable d'année en année.
Les recherches actuelles s'orientent, d'une part, vers la rapidité de la compression et, d'autre part, vers l'optimisation de la compression des flux vidéo.

Ces recherches sont dues à l'évolution récente des technologies de production de flux vidéo qui produisent une quantité toujours plus élevée de données.
Afin d'illustrer ceci, nous pouvons prendre l'exemple récent où, en mars 2020, suite à la pandémie du COVID-19, il a été demandé aux plateformes de streaming de diminuer la qualité du contenu afin de libérer de la bande passante.

Le principe de fonctionnement de ces deux méthodes a été comparé, dans le chapitre 4, en matière de similitudes, de différences et de complémentarités.

Ces méthodes de réduction de données permettent de diminuer la quantité de volume de données enregistrées, mais d'autres pistes sont actuellement explorées comme, par exemple, le stockage des données dans l'ADN[4].
Ce support offre de nombreux avantages par rapport aux technologies actuelles comme la capacité d'enregistrer une quantité importante de données dans un volume très faible.
De plus, l'ADN est la base de toute vie sur Terre et, par conséquent, il est fort probable qu'il existera toujours un moyen de consulter les données contrairement à un DVD ou un disque dur.

# Bibliographie


[1] N. Ahmed, T. Natarajan, and K. R. Rao. Discrete cosine transform. *IEEE Transactions on Computers*, C-23(1) :90–93, 1974.

[2] Airelle. Comparatif des performances de différentes méthodes de compression de fichiers, February 2011. http ://rlwpx.free.fr/WPFF/comploc.htm. Consultation : 10/2020.

[3] Mohammed Al-Laham and Ibrahiem MM El Emary. Comparative study between various algorithms of data compression techniques. *IJCSNS International Journal of Computer Science and Network Security*, 7(4) :281–291, 2007.

[4] Fabrice Auclert. Les molécules de notre corps peuvent aussi stocker des données, July 2019. https ://www.futura-sciences.com/tech/actualites/sauvegarde-molecules-notre-corps-peuvent-aussi-stocker-donnees-76700/. Consultation : 12/2020.

[5] Fabio Bellifemine, A Capellino, Antonio Chimienti, Romualdo Picco, and R Ponti. Statistical analysis of the 2d-dct coefficients of the differential signal for images. *Signal Processing : Image Communication*, 4(6) :477–488, 1992.

[6] Jon Louis Bentley, Daniel D. Sleator, Robert E. Tarjan, and Victor K. Wei. A locally adaptive data compression scheme. *Commun. ACM*, 29(4) :320–330, April 1986.

[7] Michael Burrows and David J Wheeler. A block-sorting lossless data compression algorithm. *Citeseer*, 1994.

[8] Corel Corporation. Winzip, 2020. https ://www.winzip.com/win/en/prodpagewz.html. Consultation : 11/2020.

[9] Biplob Debnath, Sudipta Sengupta, and Jin Li. Chunkstash : Speeding up inline storage deduplication using flash memory. In *2010 USENIX Annual Technical Conference (ATC)*. USENIX, June 2010.

[10] Mike Dutch. Understanding data deduplication ratios. In *SNIA Data Management Forum*, volume 7, 2008.

[11] ADC-Soft WinRAR France. Winrar, 2020. https ://www.winrar-france.fr/aide_winrar/index.html. Consultation : 11/2020.

[12] Sébastien Gavois. L'extraordinaire évolution du stockage. In *Magazine #1 de Next INpact*, March 2020.

[13] Windsor W Hsu. Method and apparatus for managing data objects of a data storage system, November 20 2012. US Patent 8,316,064.

[14] Dell Inc. Appliances dell emc powerprotect dd. https ://www.delltechnologies.com/fr-mg/collaterals/unauth/data-sheets/products/data-protection/h17926-dellemc-powerprotect-dd-ds.pdf. Consultation : 11/2020.

[15] Quest Software Inc. Qorestor. https ://www.quest.com/fr-fr/products/qorestor/. Consultation : 11/2020.

[16] Hala Jasim and Assmaa Fahad. Evaluation of two thresholds two divisor chunking algorithm using rabin finger print, adler, and sha1 hashing algorithms. *Iraqi Journal of Science*, 58 :2438–2446, 01 2018.

[17] Amanpreet Kaur and Sonia Sharma. An efficient framework and techniques of data deduplication in cloud computing. *www.ijcst.com*, 8 :27–31, 04 2017.

[18] Gopi Keerthy. Pibytes : Deduplication internals, February 2013. https ://pibytes.wordpress.com/2013/02/02/deduplication-internals-part-1/. Consultation : 10/2020.

[19] Minhaj Ahmad Khan. Evaluation of basic data compression algorithms in a distributed environment. *Journal of Basic & Applied Sciences*, 8(2), 2012.

[20] SR Kodituwakku and US Amarasinghe. Comparison of lossless data compression algorithms for text data. *Indian journal of computer science and engineering*, 1(4) :416–425, 2010.

[21] Erik Kruus, Cristian Ungureanu, and Cezary Dubnicki. Bimodal content defined chunking for backup streams. In *Proceedings of the 8th USENIX Conference on File and Storage Technologies*, FAST'10, page 18, USA, 01 2010. USENIX Association.

[22] Dr. Chiu C. Tan M. Reeve Groman. A framework for testing data deduplication algorithms. Department of Computer & Information Sciences, Temple University.

[23] Jyoti Malhotra and J. Bakal. A survey and comparative study of data deduplication techniques. *2015 International Conference on Pervasive Computing (ICPC)*, pages 1–5, 2015.

[24] Nagapramod Mandagere, Pin Zhou, Mark A Smith, and Sandeep Uttamchandani. Demystifying data deduplication. In *Proceedings of the ACM/IFIP/USENIX Middleware '08 Conference Companion*, Companion '08, page 12–17, New York, NY, USA, 2008. Association for Computing Machinery.

[25] E. Manogar and S. Abirami. A study on data deduplication techniques for optimized storage. *2014 Sixth International Conference on Advanced Computing (ICoAC)*, pages 161–166, 2014.

[26] Colin F Merrick. Selective sampling for compression and effective reconstruction, 2016.

[27] Dutch T Meyer and William J Bolosky. A study of practical deduplication. *ACM Transactions on Storage (ToS)*, 7(4) :1–20, 2012.

[28] Draft Lecture Notes. Compression algorithms : Huffman and lempel-ziv-welch (lzw). *Last Update : February*, 13, 2012.

[29] Igor Pavlov. 7-zip, 2019. https ://www.7-zip.org. Consultation : 11/2020.

[30] C.L. Philip Chen and Chun-Yang Zhang. Data-intensive applications, challenges, techniques and technologies : A survey on big data. *Information Sciences*, 275 :314 – 347, 2014.

[31] M. O. RABIN. Fingerprinting by random polynomials. *Technical Report*, 1981.

[32] Senthil Shanmugasundaram and Robert Lourdusamy. A comparative study of text compression algorithms. *International Journal of Wisdom Based Computing*, 1(3) :68–76, 2011.

[33] Statista. Digital economy compass 2019, July 2019. https ://www.statista.com/study/52194/digital-economy-compass/. Consultation : 09/2020.

[34] Giorgio Tani. Peazip, 2020. https ://peazip.github.io. Consultation : 11/2020.

[35] Western Digital Technologies. Progressive deduplication technology, December 2013.

[36] Nick Trimbee. Next generation storage efficiency with dell emc powerscale in-line data reduction, 2020. Dell Inc.

[37] SV Viraktamath and Girish V Attimarad. Impact of quantization matrix on the performance of jpeg. *International Journal of Future Generation Communication and Networking*, 4(3) :107–118, 2011.

[38] Welch. A technique for high-performance data compression. *Computer*, 17(6) :8–19, 1984.

[39] Wikipedia. Discrete cosine transform, December 2020. https ://en.wikipedia.org/wiki/Discrete_cosine_transform. Consultation : 12/2020.

[40] Wikipedia. Jpeg, December 2020. https ://en.wikipedia.org/wiki/JPEG. Consultation : 12/2020.

[41] W. Xia, H. Jiang, D. Feng, F. Douglis, P. Shilane, Y. Hua, M. Fu, Y. Zhang, and Y. Zhou. A comprehensive study of the past, present, and future of data deduplication. *Proceedings of the IEEE*, 104(9) :1681–1710, 2016.

[42] Wen Xia, Xiangyu Zou, Yukun Zhou, Hong Jiang, Chuanyi Liu, Dan Feng, Yu Hua, Yuchong Hu, and Yucheng Zhang. The design of fast content-defined chunking for data deduplication based storage systems. *IEEE Transactions on Parallel and Distributed Systems*, PP :1–1, 04 2020.

[43] Benjamin Zhu, Kai Li, and Hugo Patterson. Avoiding the disk bottleneck in the data domain deduplication file system. In *Proceedings of the 6th USENIX Conference on File and Storage Technologies*, FAST'08, USA, 2008. USENIX Association.

[44] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3) :337–343, 1977.

[45] J. Ziv and A. Lempel. Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory*, 24(5) :530–536, 1978.