INF8770 – Technologies multimédias

Automne 2024

Travail pratique #1 – Comparaison et caractérisation de méthodes de codage

Groupe 02 – Équipe A37

2221053 – Thomas Rouleau

2149244 – Justin Lefrançois

Soumis à : M. Yassine Achour

2024-10-02

Question 1 (/4)

Hypothèse 1 : Taux de compression

Puisque le codage arithmétique est une méthode probabiliste basée sur la théorie de l'entropie, il est efficace pour représenter l'information de manière compacte même lorsque celle-ci est très entropique[1]. En comparaison, la méthode LZ77 présente d'excellentes performances lors du codage de données répétitives[1]. Ainsi, nous formulons l'hypothèse que le codage arithmétique produira en moyenne un taux de compression supérieur à celui de LZ77 pour des images complexe. À l'inverse, LZ77 performera mieux pour des textes dans des langues structuré comme l'anglais.

Hypothèse 2 : Temps de compression

Étant donné que la méthode LZ77 repose sur des opérations telles que la recherche dans une fenêtre glissante et l'encodage basé sur un dictionnaire, tandis que le codage arithmétique utilise des calculs complexes d'estimation probabiliste des symboles et d'encodage entropique, ce dernier est limité dans sa capacité à coder plusieurs symboles répétés simultanément[2]. Par conséquent, nous supposons que la méthode LZ77 nécessitera moins de temps pour compresser des données, en moyenne, comparativement au codage arithmétique.

Hypothèse 3 : Utilisation des ressources (Mémoire vive et CPU)

Le codage arithmétique repose sur de nombreux calculs probabilistes, ce qui implique une demande élevée en ressources de calcul (CPU). Ce type de codage utilise des techniques avancées pour représenter les symboles en fonction de leurs probabilités, entraînant une plus grande complexité algorithmique[3]. En revanche, la méthode LZ77 se base sur une fenêtre glissante et un dictionnaire pour identifier et encoder les séquences répétées de manière plus simple. Cela peut engendrer une utilisation accrue de la mémoire, particulièrement pour gérer les données dans la fenêtre et le dictionnaire, en fonction de leur taille[3]. Ainsi, nous supposons que la méthode de codage arithmétique exigera une plus grande capacité CPU que la méthode LZ77, tandis que LZ77 demandera une plus grande utilisation de la mémoire pour le processus de compression[2].

Question 2 (/4)

Pour vérifier les hypothèses formulées concernant les caractéristiques de performances des méthodes de compression LZ77 et codage arithmétique, nous nous concentrerons sur deux types de données : les fichiers texte et les images. Ces deux catégories offrent une variété de structures de données permettant de vérifier les performances des algorithmes.

Les fichiers texte choisis comporteront des chaînes de caractères significativement longues et représentant bien des textes anglophones standard. Ces fichiers proviennent du Canterbury Corpus collection et sont connus pour leur résultat de compression « typique », ce qui les rend idéaux pour tester les algorithmes de compression. Nous utiliserons les fichiers alice29.txt, asyoulik.txt et plrabn12.txt pour ces tests.

Les images utilisées seront en couleurs (RGB) avec une profondeur de 24 bits et sous format TIFF. Le format TIFF (Tagged Image File Format) est un format de fichier d'image flexible, adapté à une variété d'applications et est un format de stockage sans perte de données. Les images proviennent de la base de données de l'University of Southern California SIPI (Signal and Image Processing Institute). Nous utiliserons 15 images, dont 8 de format 256x256 pixels (figure 1) et 7 de formats 512x512 pixels (figure 2). Cette variété de tailles et de couleurs permet de tester les méthodes de compression sur des types de données avec des caractéristiques distinctes. Ces images sont présentées dans les figures 1 et 2 suivantes :

Figure 1 : Images de format TIFF de 256x256 pixels utilisées pour les tests de compression

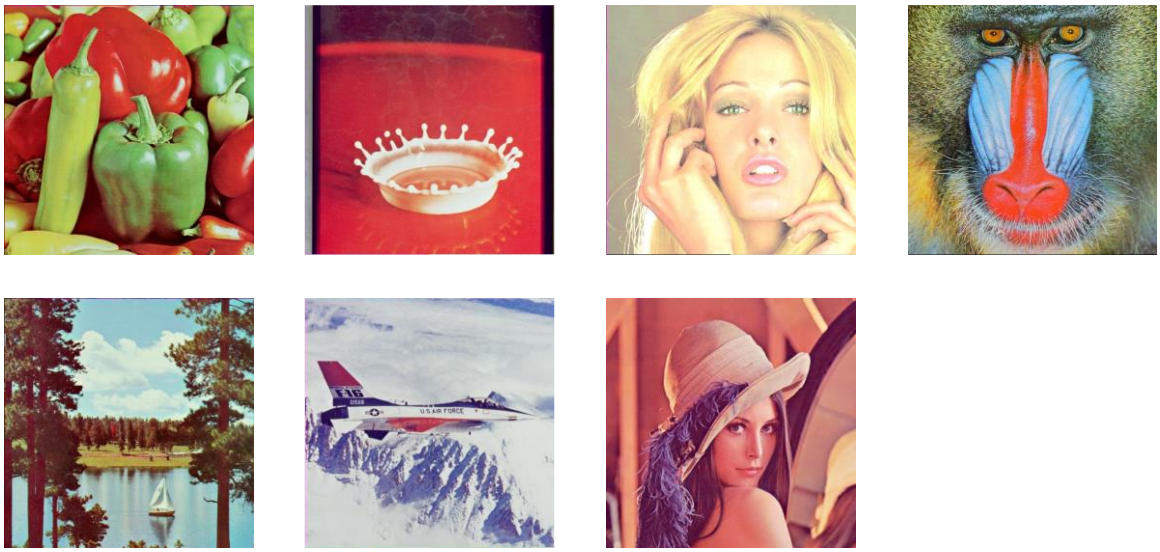

Figure 2 : Images de format TIFF de 512x512 pixels utilisées pour les tests de compression

Pour évaluer la qualité de la compression, nous comparerons les méthodes LZ77 et codage arithmétique en mesurant la taille des fichiers originaux et compressés. Nous calculerons le taux de compression en utilisant la formule suivante indiquant le taux d'espace économisé par l'algorithme de compression :

$$Taux\ de\ compression = 1 - \frac{Taille\ compressée}{Taille\ originale}$$

Le temps de compression sera mesuré pour chaque fichier. Nous utiliserons la librairie Python « psutil.cpu_times() » pour mesurer le temps CPU nécessaire à la compression, sans prendre en compte le temps d'écriture ou de lecture dans un fichier, mais en prenant en compte tout calcul préliminaire tel que le calcul de fréquence de l'algorithme arithmétique. De plus, nous mesurerons l'utilisation de la mémoire et du CPU pour chaque fichier compressé afin de déterminer l'utilisation maximale de chaque algorithme de compression. Nous utiliserons la librairie Python « psutil » pour obtenir une moyenne d'utilisation des ressources lors de la compression des données.

Les mêmes fichiers texte et images seront utilisés pour tous les tests afin de garantir leur cohérence. Nous utiliserons des implémentations des algorithmes LZ77 et codage arithmétique en Python pour effectuer ces compressions ainsi qu'un script permettant d'effectuer des tests automatisés. Chacun des trois fichiers texte sera compressé 10 fois par méthode de compression afin d'obtenir une moyenne pour chaque fichier et une moyenne totale en fonction de l'algorithme de compression. Chacune des 15 images sera compressée 3 fois pour obtenir une répartition des résultats similaire à celle des fichiers texte, soit une moyenne par image en fonction de la compression utilisée et une moyenne totale en fonction de la compression. Les tests seront effectués sur Windows Subsystem for Linux (WSL) avec une image Ubuntu 22.04. L'ordinateur utilisé pour les tests possède un processeur Intel i9-13900H de 14 cœurs et 32 GB de mémoire vive**.**

Question 3 (/4)

Les algorithmes de compression sont définis dans des classes Python afin de faciliter leur utilisation. L'algorithme de compression arithmétique a été développé en s'appuyant sur plusieurs sources mentionnées dans le code. L'algorithme LZ77, quant à lui, a été tiré du web et a été modifié pour s'adapter à nos entrées et sorties. L'objectif de cette étude est de vérifier les performances de ces algorithmes, plutôt que de les développer complètement, étant donné qu'ils ont déjà été largement implémentés. Des méthodes de décompressions ont également été implémentés afin de confirmer la validité des compressions.

Algorithme de compression arithmétique

L'algorithme de compression arithmétique est implémenté dans la classe ArithmeticCompressor. Le code est basé sur pyae.py et Codagearithmetique.ipynb. Cette classe contient des méthodes pour la compression et la décompression des fichiers texte et des images. Voici une description des principales méthodes et leur utilité :

Classe ArithmeticCompressor :

- **compute_symbol_probabilities** :
  - **Description** : Cette méthode calcule les fréquences et les probabilités des symboles dans les données.
  - **Fonctionnement** :
    - Parcourt les données pour compter la fréquence de chaque symbole
    - Calcule la probabilité de chaque symbole en divisant sa fréquence par la taille totale des données
  - **Entrées** : Une chaîne de caractères ou une liste de symboles.
  - **Sorties** : Un dictionnaire des probabilités des symboles et un dictionnaire des bornes inférieures cumulatives des symboles.
  - **Objectif** : Les probabilités des symboles sont nécessaires pour la compression arithmétique, car elles déterminent les intervalles de codage.

- **arithmetic_compression** :
    - **Description** : Cette méthode effectue la compression arithmétique en utilisant les probabilités calculées.
    - **Fonctionnement** :
        - Initialise les bornes inférieures et supérieurs de l'intervalle de codage
        - Pour chaque symbole dans les données, réduit l'intervalle de codage en fonction de la probabilité du symbole
        - Génère un bitstream en fonction des bornes inférieures et supérieures de l'intervalle de codage
    - **Entrées** : Les données à compresser, les probabilités des symboles, et les bornes inférieures cumulatives des symboles.
    - **Sortie** : Un bitstream compressé.
    - **Objectif** : La compression arithmétique permet de réduire la taille des données en utilisant des intervalles de codage basés sur les probabilités des symboles.
- **write_bitstream_to_file** :
    - **Description** : Cette méthode écrit le bitstream compressé dans un fichier.
    - **Fonctionnement** :
        - Écrit les informations sur les probabilités des symboles et les bornes inférieures cumulatives dans le fichier.
        - Écrit le bitstream compressé dans le fichier.
    - **Entrées** : Le bitstream compressé, les probabilités des symboles, les bornes inférieures cumulatives des symboles, la taille totale des symboles, et éventuellement des informations sur l'image (largeur, hauteur, mode).
    - **Sortie** : Un fichier contenant le bitstream compressé.
    - **Objectif** : Écrire le bitstream dans un fichier permet de stocker les données compressées pour une utilisation ultérieure.
- **Méthode read_bitstream_from_file** :
    - **Description** : Cette méthode lit le bitstream compressé à partir d'un fichier.
    - **Fonctionnement** :
        - Lit les informations sur les probabilités des symboles et les bornes inférieures cumulatives à partir du fichier.
        - Lit le bitstream compressé à partir du fichier.
    - **Entré** : Le chemin du fichier contenant le bitstream compressé.
    - **Sorties** : Le bitstream compressé, les probabilités des symboles, les bornes inférieures cumulatives des symboles, la taille totale des symboles, et éventuellement des informations sur l'image (largeur, hauteur, mode).
    - **Objectif** : Lire le bitstream à partir d'un fichier permet de récupérer les données compressées pour la décompression.

- **Méthode arithmetic_decompression** :
    - **Description** : Cette méthode effectue la décompression arithmétique.
    - **Fonctionnement** :
        - Initialise les bornes inférieures et supérieures de l'intervalle de codage.
        - Lit les bits du bitstream compressé pour déterminer les symboles en fonction des probabilités des symboles.
        - Réduit l'intervalle de codage en fonction des symboles décompressés.
    - **Entrées** : Le bitstream compressé, les probabilités des symboles, les bornes inférieures cumulatives des symboles, et la taille totale des symboles.
    - **Sortie** : Les données décompressées.
    - **Objectif** : La décompression arithmétique permet de récupérer les données originales à partir du bitstream compressé.

LZ77Compressor

Pour l'algorithme de compression sans perte LZ77, nous avons utilisé les fonctions disponibles sur ce repo GitHub : LZ77 sous python/compress.py. Voici les méthodes fournis par ce repo ainsi que leur description :

- **Méthode compress()** :
    - **Description** : Cette méthode effectue la compression LZ77 des données d'entrée.
    - **Fonctionnement** :
        - Convertit les données d'entrée en un bytearray pour faciliter la manipulation des octets.
        - Utilise une fenêtre coulissante pour rechercher les séquences répétitives.
        - Pour chaque octet de données d'entrée, calcule la longueur et l'offset de la plus grande séquence répétitive trouvée dans la fenêtre.
        - Ajoute à la sortie un triplet (offset, length, char) représentant les données compressées.
    - **Entrées** : input_data (données d'entrée sous forme de bytes), max_offset (taille maximale de la fenêtre), max_length (longueur maximale des séquences).
    - **Sortie** : Une liste de triplets (offset, length, char) représentant les données compressées.
    - **Objectif** : La compression LZ77 permet de représenter des séquences répétitives de manière plus compacte pour réduire la taille des données.

- **Méthode to_bytes()** :
  - **Description** : Cette méthode convertit la représentation compressée en un tableau d'octets.
  - **Fonctionnement** :
    - Parcourt la liste des triplets (offset, length, char) pour les convertir en une série d'octets.
    - Assure que les valeurs d'offset et de longueur respectent le nombre de bits spécifié.
    - Combine les valeurs d'offset et de longueur, puis ajoute les caractères correspondants au tableau d'octets.
  - **Entrées** : compressed_representation (liste de tripletscompressés), offset_bits (nombre de bits pour représenter l'offset), length_bits (nombre de bits pour représenter la longueur).
  - **Sortie** : Un bytearray contenant la version compressée des données.
  - **Objectif** : Convertir la structure des données compressées en un format binaire pour pouvoir les écrire dans un fichier ou les transmettre efficacement.
- **Méthode best_length_offset()** :
  - **Description** : Cette méthode trouve la meilleure combinaison longueur/offset pour les séquences répétitives dans la fenêtre coulissante.
  - **Fonctionnement** :
    - Coupe la fenêtre à la taille maximale (max_offset) pour limiter la recherche.
    - Recherche la séquence la plus longue dans la fenêtre correspondant aux données actuelles.
    - Retourne l'offset et la longueur de la plus grande séquence trouvée.
  - **Entrées** : window (fenêtre coulissante sous forme de bytearray), input_array (données restantes à compresser), max_length (longueur maximale des séquences), max_offset (taille maximale de la fenêtre).
  - **Sortie** : Un couple (length, offset) représentant la longueur et l'offset de la meilleure séquence trouvée.
  - **Objectif** : Identifier les séquences répétitives pour minimiser la taille de la représentation compressée.
- **Méthode repeating_length_from_start()** :
  - **Description** : Cette méthode calcule la longueur maximale de répétition de l'entrée à partir du début de la fenêtre.
  - **Fonctionnement** :
    - Compare les octets du début de la fenêtre avec ceux de l'entrée.
    - Continue tant que les octets sont identiques et que la longueur maximale n'est pas atteinte.
    - Retourne la longueur totale des octets identiques.
  - **Entrées** : window (fenêtre coulissante sous forme de bytearray), input_array (données restantes à compresser).
  - **Sortie** : Un entier représentant la longueur de la séquence répétée.

    - **Objectif** : Aider à déterminer la meilleure longueur d'une séquence répétée pour la compression.
- **Méthode compress_file()** :
    - **Description** : Cette méthode lit un fichier, compresse son contenu, puis écrit le résultat dans un fichier de sortie.
    - **Fonctionnement** :
        - Ouvre et lit les données d'entrée en mode binaire.
        - Compresse les données en utilisant la méthode compress.
        - Convertit la représentation compressée en octets grâce à to_bytes.
        - Écrit les données compressées dans un fichier de sortie en mode binaire.
    - **Entrées** : input_file (chemin du fichier d'entrée), output_file (chemin du fichier de sortie).
    - **Sortie** : Aucun (écrit les données compressées dans le fichier de sortie).
    - **Objectif** : Permettre la compression et le stockage de données à partir d'un fichier pour réduire l'espace de stockage nécessaire.

Autres fonctions et modifications

Pour réaliser les expériences décrites, plusieurs modifications et ajouts de fonctions ont été effectués :

1. **Surveillance des Ressources** : Les méthodes monitor_resources et measure_resources ont été ajoutées pour surveiller l'utilisation du CPU et de la mémoire pendant la compression et la décompression. C'est dans ces méthodes que le temps de compression et l'utilisation des ressources sont mesurés et calculés à l'aide des fonctions time.time(), psutil.Process(), etc. psutil fournit la fonction cpu_percent qui indique directement le pourcentage de CPU requis par un processus quelconque et la fonction memory_info permettant de mesurer la mémoire utilisée par un processus. L'utilisation de process.cpu_times() permet de mesurer uniquement le temps CPU alloué à la compression afin d'assurer un résultat uniforme et spécifique.
2. **Automatisation des tests** : Un script (compression_benchmark.py) a été ajouté pour automatiser les tests de compression et de décompression. Ce script parcourt les dossiers spécifiés, effectue de multiples compressions et enregistre les résultats dans un fichier. Pour les tests effectués, chacun des trois fichiers texte a été compressé 10 fois par méthode de compression afin d'obtenir une moyenne pour chaque fichier et une moyenne totale en fonction de l'algorithme de compression. Chacune des 15 images a été compressée 3 fois pour obtenir une répartition des résultats similaire à celle des fichiers texte, soit une moyenne par image en fonction de la compression utilisée et une moyenne totale en fonction de la compression.
    a. monitor_resources : Surveille l'utilisation du CPU et de la mémoire pendant la compression.

b. measure_resources : Mesure le temps de compression, l'utilisation du CPU et de la mémoire.
c. perform_compression : Effectue la compression en utilisant les méthodes de surveillance des ressources et enregistre les résultats.
d. run_compression : Exécute la compression pour les fichiers texte et image en utilisant l'algorithme de compression arithmétique.
e. run_lz77_compression : Exécute la compression pour les fichiers texte en utilisant l'algorithme LZ77.
f. main : Parcourt les dossiers spécifiés pour les fichiers texte et image, effectue les compressions, et enregistre les résultats dans un fichier.

3. **Calcul du Taux de Compression** : La méthode calculate_compression_ratio a été ajoutée pour calculer et afficher le taux de compression. L'utilisation de la fonction os.path.getsize() permet de récupérer les informations concernant la taille des fichiers afin d'effectuer le calcul.

Résultats

Le script compression_benchmark.py a été utilisé pour obtenir tous les résultats. Son fonctionnement a été décrit dans la section précédente. Il est à noter que le pourcentage CPU obtenu représente un CPU ou cœur par tranche de 100%. Ainsi, une valeur de 200% représente l'utilisation totale de 2 CPU ou de 2 cœurs. Tous les résultats sont disponibles dans le fichier compression_results.txt.

Le tableau 1 présente les taux de compression théorique par entropie et la moyenne de 10 itérations des taux de compression, du temps de compression, du maximum de mémoire utilisé et de l'utilisation maximale du CPU en fonction du fichier texte compressé par codage arithmétique obtenues à l'aide du script compression_benchmark.py.

Fichier	Taux de compression	Taux de compression théorique (entropie)	Temps de compression (s)	Maximum de mémoire utilisé (MiB)	Utilisation maximale du CPU (%)
Alice29.txt	0,44	0,44	0,22	91,43	181,53
Asyoulik.txt	0,39	0,40	0,19	92,64	178,55
plrabn12.txt	0,45	0,44	0,67	105,65	228,58
Moyenne	**0,43**	**0,43**	**0,36**	**96,58**	**196,22**

Tableau 1 : Résultats de la compression des textes par arithmétique

Le tableau 2 présente la moyenne de 10 itérations des taux de compression, du temps de compression, du maximum de mémoire utilisé et de l'utilisation maximale du CPU en fonction du fichier texte compressé par codage LZ77 obtenues à l'aide du script compression_benchmark.py.

Fichier	Taux de compression	Temps de compression (s)	Maximum de mémoire utilisé (MiB)	Utilisation maximale du CPU (%)
Alice29.txt	0,44	9,99	93,72	255,89
Asyoulik.txt	0,40	8,74	93,63	262,14
plrabn12.txt	0,37	35,69	122,63	262,68
Moyenne	**0,40**	**18,14**	**103,33**	**260,24**

Tableau 2 : Résultats de la compression des textes par LZ77

Le tableau 3 présente les taux de compression théorique par entropie et la moyenne de 3 itérations des taux de compression, du temps de compression, du maximum de mémoire utilisé et de l'utilisation maximale du CPU en fonction du fichier d'image de format TIFF compressé par codage arithmétique obtenues à l'aide du script compression_benchmark.py. Les fichiers débutant par 4.1 sont de format 256x256 pixels et les fichiers débutant par 4.2 sont de format 512x512 pixels.

Fichier	Taux de compression	Taux de compression théorique (entropie)	Temps de compression (s)	Maximum de mémoire utilisé (MiB)	Utilisation maximale du CPU (%)
4.1.01.tiff	0,12	0,12	0,46	106,54	178,07
4.1.02.tiff	0,20	0,20	0,44	106,83	168,83
4.1.03.tiff	0,23	0,30	0,41	107,32	178,00
4.1.04.tiff	0,05	0,09	0,46	107,13	165,57
4.1.05.tiff	0,10	0,19	0,40	107,09	177,13
4.1.06.tiff	0,04	0,09	0,43	107,29	217,83
4.1.07.tiff	0,16	0,28	0,39	118,81	272,57
4.1.08.tiff	0,13	0,22	0,41	118,78	228,00
4.2.01.tiff	0,09	0,09	1,69	182,34	247,77
4.2.02.tiff	0,19	0,17	1,50	185,56	247,83
4.2.03.tiff	0,02	0,08	1,74	192,90	273,07
4.2.04.tiff	0,03	0,07	1,78	194,36	266,67
4.2.05.tiff	0,16	0,16	1,63	187,04	267,63
4.2.06.tiff	0,02	0,06	1,82	194,18	257,10
4.2.07.tiff	0,04	0,05	1,81	193,61	230,77
Moyenne	**0,11**	**0,15**	**1,03**	**147,32**	**225,12**

Tableau 3 : Résultats de la compression des images par arithmétique

Le tableau 4 présente la moyenne de 3 itérations des taux de compression, du temps de compression, du maximum de mémoire utilisé et de l'utilisation maximale du CPU en fonction du fichier d'image de format TIFF compressé par codage LZ77 obtenues à l'aide du script compression_benchmark.py. Les fichiers débutant par 4.1 sont de format 256x256 pixels et les fichiers débutant par 4.2 sont de format 512x512 pixels.

Fichier	Taux de compression	Temps de compression (s)	Maximum de mémoire utilisé (MiB)	Utilisation maximale du CPU (%)
4.1.01.tiff	-0,13	5,59	119,37	261,63
4.1.02.tiff	0,00	40,80	117,40	259,97
4.1.03.tiff	0,21	33,64	114,11	262,10
4.1.04.tiff	-0,26	51,46	121,39	283,57
4.1.05.tiff	-0,15	47,66	119,80	303,83
4.1.06.tiff	-0,30	53,51	121,81	262,87
4.1.07.tiff	0,19	33,88	126,36	262,93
4.1.08.tiff	0,06	39,23	128,40	263,23
4.2.01.tiff	-0,15	196,51	203,88	302,70
4.2.02.tiff	-0,19	206,41	212,83	285,07
4.2.03.tiff	-0,68	286,66	240,42	305,10
4.2.04.tiff	-0,52	263,11	232,11	327,33
4.2.05.tiff	-0,08	199,46	206,64	327,67
4.2.06.tiff	-0,50	266,00	230,56	327,77
4.2.07.tiff	-0,55	284,53	233,55	325,97
Moyenne	**-0,20**	**136,56**	**168,58**	**290,78**

Tableau 4 : Résultats de la compression des images par LZ77

Question 4 (/8, 4 points par qualité)

Discussion

Dans ce rapport, notre objectif était de comparer les performances de deux méthodes de compression de données : le codage arithmétique et la méthode LZ77. Nous avons mesuré et analysé les taux de compression, les temps de compression et l'utilisation des ressources (CPU et mémoire) pour des textes structurés en anglais et des images TIFF 24 bits. Les résultats obtenus ont permis de vérifier partiellement les hypothèses formulées.

D'abord, il convient de mentionner que les valeurs des taux de compression et des temps de compression des images par le codage LZ77, ainsi que les temps de compression pour les textes par le codage LZ77, semblent ne pas être représentatifs des capacités de ce codage. Non seulement le taux de compression est négatif dans le cas des images, mais il est également significativement inférieur au taux de compression par entropie (moyenne de -0,20 contre 0,15). Le temps de compression est également significativement plus long que celui de l'encodage arithmétique. Par exemple, le temps de compression pour le texte est plus de 40 fois plus élevé (0,43 secondes pour le codage arithmétique et 18,14 secondes pour le codage LZ77), et pour les images, il est plus de 132 fois plus élevé (1,03 secondes pour le codage arithmétique et 136,56 secondes pour le codage LZ77). Ces résultats peuvent être expliqués de plusieurs manières. Il est possible que la fenêtre glissante utilisée ne fût pas de taille appropriée pour les données, bien que les taux de compression étaient idéaux pour le texte, mais les temps étaient tout de même élevés. Une autre cause possible est l'utilisation de structures de données et de bibliothèques peu efficaces pour la gestion de la compression LZ77, ou encore une mauvaise implémentation générale de ce type de codage, surtout considérant que le code n'a pas été écrit par notre équipe.

En effectuant la compression de chacun des trois textes sélectionnés 10 fois et de chacune des 15 images trois fois, nous avons déterminé les taux de compression, les temps de compression, l'utilisation maximale de mémoire et de CPU moyen en fonction des fichiers compressés, ainsi que la moyenne globale pour chaque méthode de codage. Ainsi, le taux de compression moyen pour les textes est de 0,43 pour le codage arithmétique, de 0,40 pour le codage LZ77 et de 0,43 pour la valeur théorique par entropie. Le codage arithmétique et LZ77 ont donc des taux de compression similaires (différence de 0,03) pour les textes structurés en anglais. Le taux de compression moyen pour les images 24 bits de format TIFF est de 0,11 pour le codage arithmétique, de -0,20 pour le codage LZ77 et de 0,15 pour le taux de compression théorique par entropie. Le codage arithmétique a un taux de compression supérieur à celui de LZ77 (différence de 0,31). Notre première hypothèse est donc partiellement confirmée par ces résultats. En effet, comme prévu, le codage arithmétique a produit en moyenne un taux de compression supérieur à celui de LZ77 pour des images complexes. Cependant, les taux de compression pour les textes structurés en anglais ont été similaires pour le codage arithmétique et LZ77, avec un léger avantage pour le codage arithmétique,

ce qui ne supporte pas notre hypothèse sur la performance supérieure de LZ77 sur ce type de données.

En ce qui concerne le temps de compression, le codage arithmétique a montré une performance nettement supérieure. Pour les textes, le temps de compression moyen est de 0,36 secondes pour le codage arithmétique et de 18,14 secondes pour LZ77. Pour les images, le temps de compression moyen est de 1,03 secondes pour le codage arithmétique et de 136,56 secondes pour LZ77. Ces résultats contredisent notre deuxième hypothèse selon laquelle LZ77 nécessiterait moins de temps pour compresser des données.

Dans le cas de l'utilisation des ressources, le codage arithmétique a démontré généralement de meilleures performances. Pour les textes, l'utilisation maximale du CPU moyen est de 196,22% pour le codage arithmétique et de 260,24% pour LZ77 (différence de 64,02%), tandis que le maximum de mémoire utilisé moyen est de 96,58 MiB pour le codage arithmétique et de 103,33 MiB pour LZ77 (différence de 6,75 MiB). Pour les images, l'utilisation maximale du CPU moyen est de 225,12% pour le codage arithmétique et de 290,78% pour LZ77 (différence de 65,66%), tandis que le maximum de mémoire utilisé moyen est de 147,32 MiB pour le codage arithmétique et de 168,58 MiB pour LZ77 (différence de 21,16 MiB). Ces résultats contredisent notre troisième hypothèse selon laquelle le codage arithmétique exigerait une plus grande capacité CPU, mais confirment notre considération que la méthode LZ77 utiliserait une plus grande utilisation de la mémoire dans son processus de compression.

Conclusion

Notre objectif était de comparer les méthodes de compression de données arithmétique et LZ77 en mesurant et comparant leurs taux de compression, leurs temps de compression et leurs utilisations des ressources (CPU et mémoire). Pour ce faire, nous avons développé l'algorithme de compression arithmétique et obtenu un algorithme de compression LZ77 en ligne, puis nous avons effectué de nombreuses compressions sur des textes structurés en anglais et des images au format TIFF afin d'obtenir des moyennes représentatives. Les résultats obtenus montrent que le codage arithmétique est plus efficace en termes de taux de compression pour les images complexes et est plus rapide pour les textes et les images. Contrairement aux hypothèses initiales, le codage arithmétique utilise moins de CPU pour la compression des textes et des images. Ces résultats suggèrent que le codage arithmétique est une méthode plus performante pour la compression de données complexes et structurées, tandis que LZ77 peut être plus adapté pour des données répétitives simples.

Plusieurs sources d'erreurs peuvent influencer les résultats obtenus. En premier lieu, le code de compression arithmétique a été obtenu en ligne, ce qui peut le rendre inadapté à notre objectif, causant une différence significative entre les résultats obtenus et les résultats attendus. De plus, l'ordonnanceur de l'ordinateur et le compilateur du code peuvent causer des différences dans l'utilisation du CPU et de la mémoire des méthodes de compression.

Bibliographie

[1] M. Ignatoski, J. Lerga, L. Stankovic, and M. Dakovic, "Comparison of Entropy and Dictionary Based Text Compression in English, German, French, Italian, Czech, Hungarian, Finnish, and Croatian," *Mathematics,* vol. 8, p. 1059, 07 2020, doi: 10.3390/math8071059.

[2] Z. Jing, "Real-Time Lossless Compression of SoC Trace Data," 2015. [Online]. Available: https://api.semanticscholar.org/CorpusID:54881141.

[3] C. Shuai, S. Li, and H. Liu, "Comparison of compression algorithms on vehicle communications system," 2015.