1873653 – DALPHOND, Jean-Olivier
1928777 – SARWAT, Mariam

INF8770 - RAPPORT DE TRAVAIL PRATIQUE #1
Comparaison et caractérisation de méthodes de codage

Travail présenté à M. Hugues Perreault

École Polytechnique de Montréal
28 septembre 2020

Table des matières

Question 1

Hypothèse #1 :

La méthode de codage arithmétique est une solution tout-usage, qui performe mieux que la méthode par paires d'octets de façon générale. Le codage par paire d'octets est probablement plus efficace dans les cas où on reconnaît des séries de caractères, comme dans des phrases réelles, puisqu'il s'agit d'une méthode de type dictionnaire. Toutefois, pour des cas plus généraux, nous croyons que le codage arithmétique est préférable.

Hypothèse #2 :

La méthode de codage par paires d'octets est plus efficace lorsqu'on a des symboles qui se répètent souvent, donc par exemple lorsqu'on a des images en noir et blanc ou des longs textes. Dans le cas d'une image avec des couleurs plus variées (telle que les photographes), la méthode arithmétique serait une meilleure option. Effectivement, les couleurs dans une photographie, par exemple, ne se répètent pas très souvent comme chaque pixel présente habituellement une nouvelle variation de la couleur, même cette variation est faible.

Hypothèse #3 :

La méthode de codage par paires d'octets est plus rapide à exécuter que la méthode de codage arithmétique. Les décisions que doit prendre l'algorithme sont plus simples et plus répétitives, alors que les données à traiter par le codage arithmétique sont plus larges et demandent plus de bits pour les représenter.

Hypothèse #4 :

La notion de distribution des symboles est prise en compte dans le codage arithmétique, ce qui suggère que cette méthode sera plus efficace dans des cas où il y a, par exemple, beaucoup d'occurrences d'une petite quantité de symboles, ainsi que quelques symboles plus rares. Dans des cas où la distribution est davantage équilibrée, l'encodage par paires d'octets devrait s'avérer plus efficace.

Question 2

Expérience hypothèse 1 :

Pour tester l'hypothèse #1, nous comparerons la performance, d'une façon plus générale, des deux méthodes en utilisant une multitude de formats de fichiers, déjà compressés ou non : des séries de caractères (des phrases réelles), des images Bitmap monochromes et avec couleurs, un fichier audio WAV et un PDF. Nous vérifierons le taux de compression (formule présentée dans l'expérience hypothèse 2). Pour comparer nos résultats, des tableaux seront utilisés.

Expérience hypothèse 2 :

Cette expérience consiste à tester et de comparer l'efficacité les deux méthodes d'encodage à l'étude. Pour ce faire, nous allons comparer le taux de compression. Ce dernier est calculé en utilisant la formule suivante :

$$1 - \frac{\text{Taille compressée}}{\text{Taille initiale}}$$

Pour confirmer ou infirmer l'hypothèse #2, nous allons répéter cette expérience sur différentes images, sur des séquences de caractères répétitives et sur des textes hétérogènes. Par la suite, les taux seront comparés entre eux à l'aide d'un diagramme afin de pouvoir tirer une conclusion.

Expérience hypothèse 3 :

Afin de confirmer ou infirmer l'hypothèse #3, nous allons comparer le temps d'exécution que chaque algorithme prend pour compresser différents types de données. Pour être capables de généraliser nos résultats, nous allons faire l'expérience avec les types de données suivantes : des messages « génériques » (tel que vu en classe), des paragraphes de texte réel, des séquences de lettres générées aléatoirement, des images multicolores, des images en noir et blanc, un fichier audio WAV et un vidéo MOV.

Nous avons modifié les codes trouvés afin de calculer le temps demandé par la compression pour chaque méthode. Chacun des deux algorithmes recevra les mêmes types de données que pour l'expérience 1. Afin d'avoir des temps le plus précis possible, le chronomètre sera lancé au début de la fonction d'encodage, et il sera arrêté lorsque la compression sera terminée. Pour comparer les résultats, un diagramme ainsi qu'un graphique seront utilisés. Afin de confirmer l'hypothèse, nous cherchons des résultats positifs pour 70% des fichiers utilisés.

Expérience hypothèse 4 :

Pour tester l'hypothèse #4, nous utiliserons des documents de texte avec des combinaisons de caractères intéressantes. D'abord, nous utiliserons un document qui contient énormément de quelques caractères très semblables, avec une minorité d'autres caractères, puis un document qui, à l'inverse, contient une distribution de caractères plus

équilibrée. Afin de confirmer ou d'infirmer l'hypothèse en question, nous allons comparer le taux de compression (en pourcentage). La formule utilisée est démontrée dans la section « expérience hypothèse 2 ». Pour présenter les résultats en lien avec cette hypothèse, nous utiliserons également un tableau qui présentera les tailles avant la compression, les tailles après la compression de chaque algorithme ainsi que le taux de compression.

Question 3

Spécification du code

Nous avons choisi d'utiliser du code en C et en C++, trouvés sur le site www.drdobbs.com (voir les références). Nous avons trouvé plusieurs références à cette source sur d'autres sites web, et nous trouvons que le code qu'on y trouve en référence est concis et bien exécuté.

En ce qui concerne l'algorithme BPE, il se résume à un seul fichier de code C. D'abord, on lit le fichier fourni en entrée par la ligne de commande de l'usager qui appelle le programme, caractère par caractère (*unsigned char*) jusqu'à remplir un tampon ou jusqu'à ce que le fichier soit complètement lu. La méthode *fileread* retourne 0 si le tampon est plein, mais que le fichier n'est pas lu complètement, ou 1 si le fichier est lu complètement (caractère EOF). Pour traiter les données dans un bloc, on analyse chaque paire de caractères afin de trouver la paire la plus fréquente dans le bloc. On stocke le résultat en mémoire, on remplace (encode) les paires et on ajuste le décompte pour ne pas traiter en double les données. S'il ne reste plus de symboles pour encoder ou si la compression est maximale, on quitte la boucle. Bien sûr, on sauvegarde ensuite le fichier de sortie.

L'algorithme d'encodage arithmétique, quant à lui, est composé de 3 fichiers codés en C++. Tout d'abord, similairement à l'algorithme BPE, deux arguments sont fournis en entrée à l'aide de la ligne de commande au programme, soit le nom du fichier à compresser et le nom du fichier en sortie (le fichier compressé). Le programme lit (caractère par caractère) le fichier fourni en entrée et encode de façon incrémentale chaque caractère. Effectivement, lorsque chaque caractère du fichier est codé, quelques bits sont ajoutés au message encodé. Alors, à la fin de l'exécution de l'algorithme, on aura, en sortie, le message codé, soit un nombre à virgule flottante qui se retrouve entre 0 et 1. Plus précisément, l'encodage se fait de la manière expliquer dans les prochaines phrases. Premièrement, un modèle sera défini et aura pour but de fournir des probabilités, entre 0 et 1, d'un caractère donné dans un message. Comme mentionné auparavant, ce processus est incrémental, alors la différence entre bas et haut est en train de devenir plus étroit avec chaque caractère encodé. En d'autres mots, la valeur du haut est constamment en train de diminuer, la valeur du bas est en train d'augmenter et la valeur du haut reste toujours plus grande que la valeur du bas. Cette plage, qui commence à [0,0; 1,0], aura comme valeur finale [bas, haut] lorsque le processus d'encodage aura terminé. Le programme, par la suite, écrit dans le fichier en sortie, spécifié par l'usager, un nombre à virgule flottant correspond à la moyenne de la plage finale retrouvée (donc $\frac{(bas+haut)}{2}$). Il est important de mentionner qu'afin de pouvoir atteindre le niveau de précision requis pour l'encodage, les variables utilisées dans l'encodage seront des binaires de longueur arbitraire. Pour que le programme puisse gérer ces variables d'une longueur arbitraire, le bit sera lu, les calculs nécessaires seront exécutés et finalement après avoir sorti le bit, il sera supprimé.

Nous avons modifié les codes pour intégrer la notion de temps d'exécution, les deux de la même façon, via la libraire standard <time.h>. Autour de la fonction qui est appelée pour effectuer la compression, par opposition aux fonctions d'affichage ou d'ouverture ou écriture de fichiers, nous avons d'abord recensé l'horloge interne avec la fonction clock(), qui renvoie une valeur de type clock_t. Après la compression, nous avons simplement calculé la différence entre la nouvelle valeur renvoyée par clock() et celle prise avant la compression, puis convertie en temps en divisant par la valeur CLOCKS_PER_SEC rapportée par la librairie. Avec un peu d'analyse dimensionnelle, on peut voir qu'on obtient effectivement une valeur en secondes après cette simple manipulation, impliquant des tics d'horloge et des secondes.

$$\frac{tic}{tic/_{sec}} = sec$$

Cette valeur calculée est ensuite envoyée à la sortie console du programme, de façon à être affichée par le terminal tout juste avant de quitter.

Résultats

Les tableaux ainsi que les graphiques suivants présentent les résultats obtenus lors des différentes expériences.

Hypothèse 1 :

Tableau 1 : Taille initiale des échantillons, taille de ces derniers après avoir appliqué l'algorithme d'encodage par paires d'octets et le taux de compression en pourcentage.

Format - Description	Taille initiale (octets)	Taille compressée (octets)	Taux de compression (%)
WAV - Fichier audio non compressé	827618	486478	41,22
BMP - Image d'un damier (noir et blanc)	1000138	14232	98,58
BMP - Image d'une plage (en couleur)	14745722	9211326	37,53
TXT - Roman	704143	380685	45,94
TXT - Texte hétérogène	2146	1559	27,35
PDF - Document	1366213	1365408	0,06

Tableau 2 : Taille initiale des échantillons, taille de ces derniers après avoir appliqué l'algorithme d'encodage par paires d'octets et le taux de compression en pourcentage.

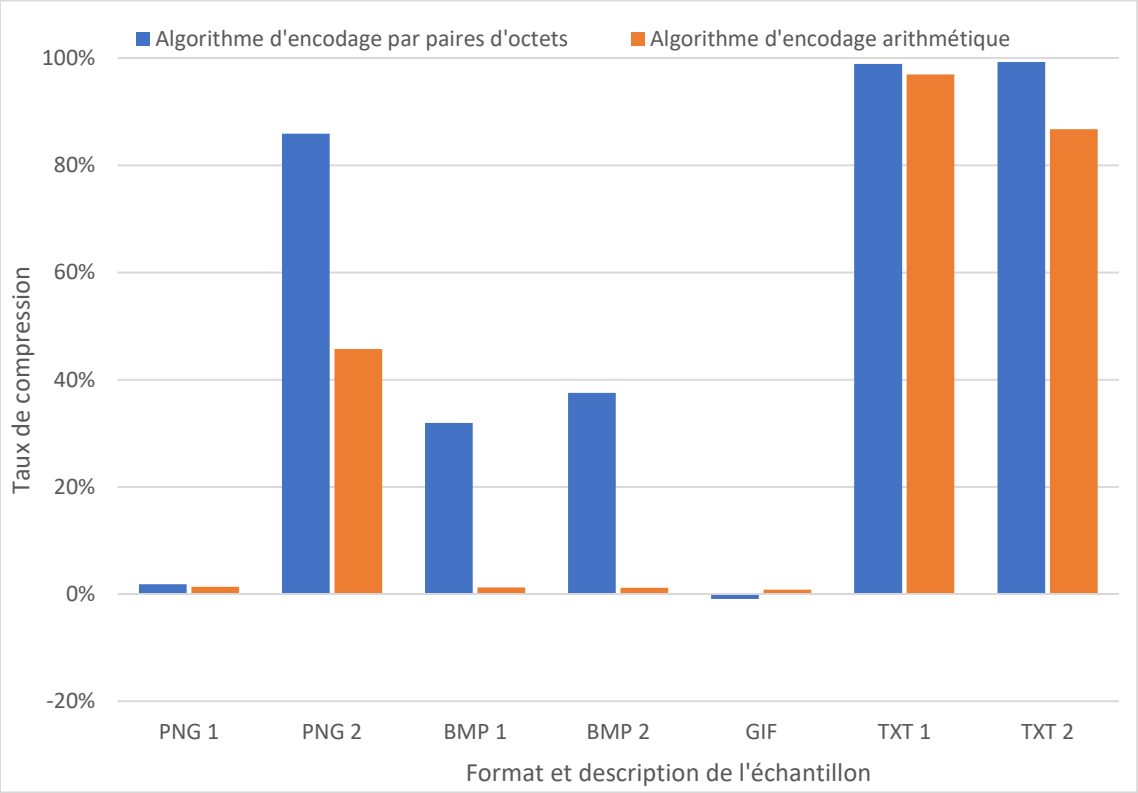
Format - Description	Taille initiale (octets)	Taille compressée (octets)	Taux de compression (%)
WAV - Fichier audio non compressé	827618	504661	39,02
BMP - Image d'un damier (noir et blanc)	1000138	125447	87,46
BMP - Image d'une plage (en couleur)	14745722	14570166	1,19
TXT - Roman	704143	396772	43,65
TXT - Texte hétérogène	2146	1614	24,79
PDF - Document	1366213	1347864	1,34

Hypothèse 2 :

<u>Tableau 3 :</u> Taux de compression pour l'algorithme encodage par paires d'octets et arithmétique selon l'échantillon en question.

Format - Description	Tag graphique	Taux de compression BPE	Taux compression arithmétique
PNG - Image horaire de Poly	PNG 1	1,81%	1,32%
PNG - Image d'un damier (noir et blanc)	PNG 2	85,88%	45,72%
BMP - Image de la grille de couleurs	BMP 1	31,90%	1,24%
BMP - Image d'une plage (en couleur)	BMP 2	37,53%	1,19%
GIF - Image animée	GIF	-0,93%	0,79%
TXT - Texte homogène	TXT 1	98,92%	96,95%
TXT - Texte abababab	TXT 2	99,27%	86,69%

<u>Graphique 1 :</u> Le taux de compression des algorithmes d'encodage par paires d'octets et arithmétique en fonction d'un échantillon.

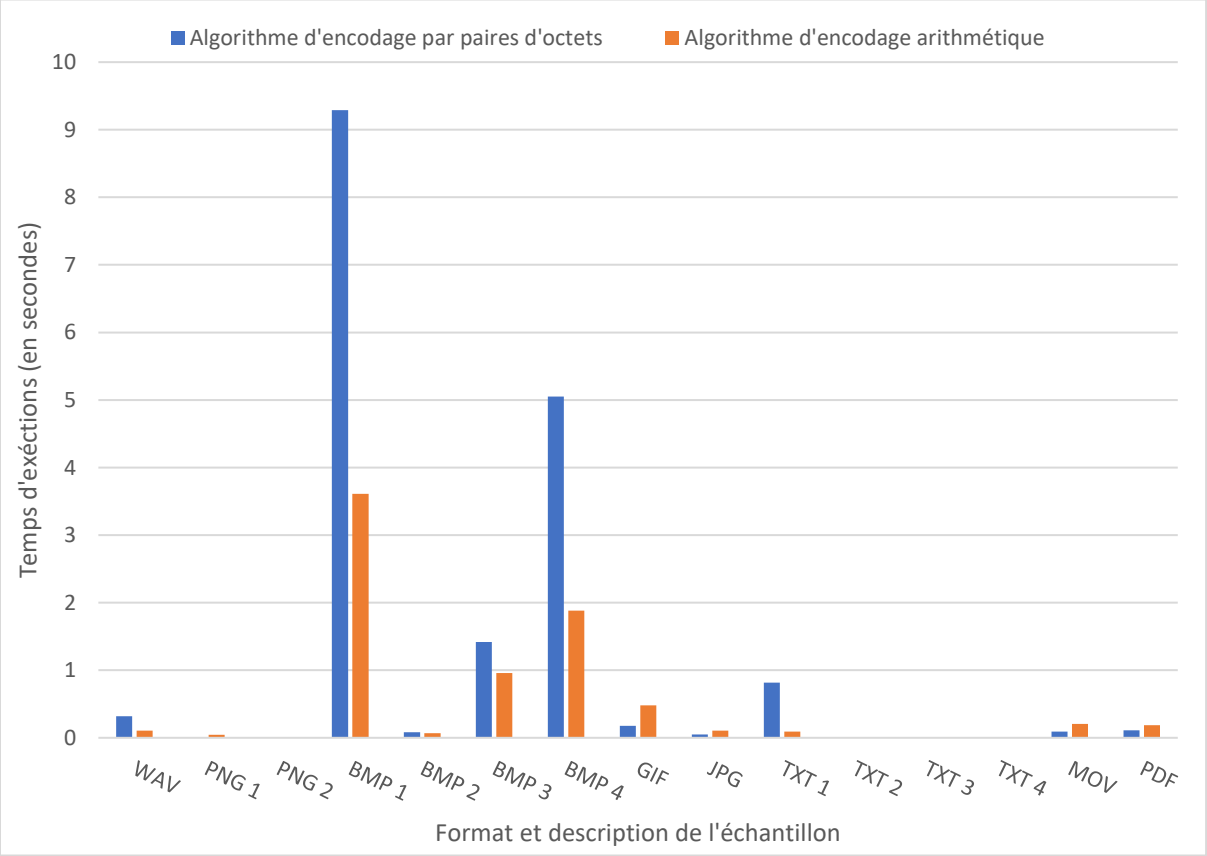


Hypothèse 3 :

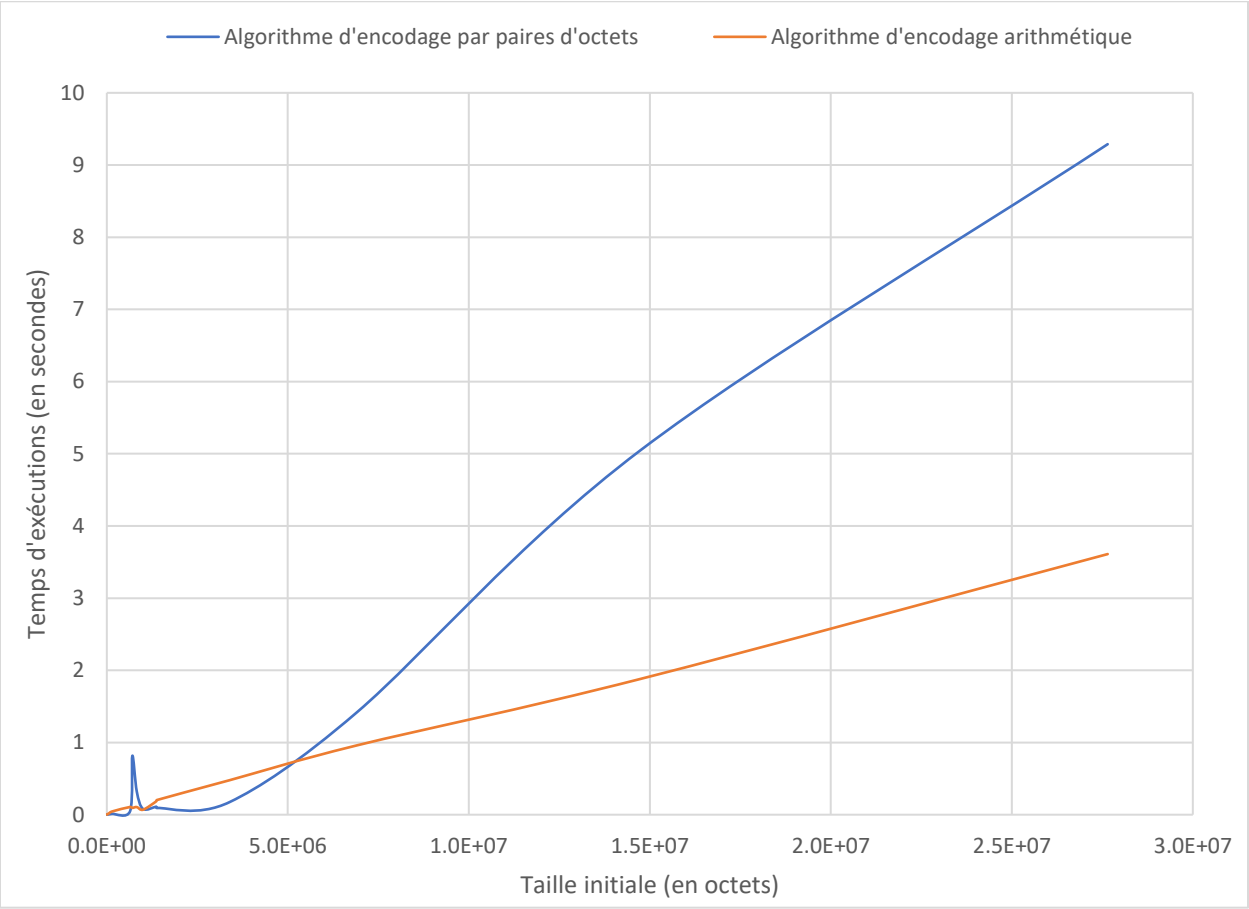
<u>Tableau 4 :</u> Temps d'exécution (en secondes) des algorithmes d'encodage par paires d'octets et arithmétique en fonction d'un échantillon.

Format - Description	Tag graphique	Temps exécution BPE	Temps exécution arithmétique
WAV - Fichier audio non compressé	WAV	0,3216	0,1055
PNG - Image de l'horaire Poly	PNG 1	0,0119	0,0445
PNG - Image d'un damier (noir et blanc)	PNG 2	0,0005	0,0022
BMP - Image d'une grille de couleurs	BMP 1	9,2887	3,6102
BMP - Image d'un damier (noir et blanc)	BMP 2	0,0803	0,0689
BMP - Image d'un fond d'écran HP	BMP 3	1,4201	0,9609
BMP - Image d'une plage	BMP 4	5,0499	1,8813
GIF - Image animée	GIF	0,1756	0,4784
JPG - Image personnelle en couleurs	JPG	0,0515	0,1070
TXT - Roman	TXT 1	0,8148	0,0942
TXT - Texte homogène	TXT 2	0,0011	0,0080
TXT - Texte abababab	TXT 3	0,0030	0,0109
TXT - Texte hétérogène	TXT 4	0,0010	0,0017
MOV - Vidéo personnelle	MOV	0,0926	0,2036
PDF - Document	PDF	0,1111	0,1853

<u>Graphique 2 :</u> Le taux de compression des algorithmes d'encodage par paires d'octets et arithmétique en fonction de l'échantillon.



<u>Graphique 3 :</u> Le temps d'exécution des algorithmes d'encodage par paires d'octets et arithmétique en fonction de la taille initiale de l'échantillon.



Hypothèse 4 :

Tableau 5 : Taille initiale des échantillons, taille de ces derniers après avoir appliqué l'algorithme d'encodage par paires d'octets et le taux de compression en pourcentage.

Description	Taille initiale (octets)	Taille compressée (octets)	Taux de compression (%)
Roman	704143	380685	45,94%
Texte homogène	11409	123	98,92%
Texte abababab	36961	270	99,27%
Texte hétérogène	2146	1559	27,35%

Tableau 6 : Taille initiale des échantillons, taille de ces derniers après avoir appliqué l'algorithme d'encodage arithmétique et le taux de compression en pourcentage.

Description	Taille initiale (octets)	Taille compressée (octets)	Taux de compression (%)
Roman	704143	396772	43,65%
Texte homogène	11409	348	96,95%
Texte abababab	36961	4919	86,69%
Texte hétérogène	2146	1614	24,79%

Question 4

Les résultats seront analysés afin de confirmer ou infirmer chacune de nos quatre hypothèses.

Retour sur les hypothèses

Retour hypothèse 1 (voir Tableau 1 et Tableau 2) :

Notre première hypothèse stipule que de façon générale la méthode d'encodage arithmétique performe mieux que la méthode par paires d'octets. Cependant, le codage par paires d'octets est plus efficace lorsqu'on a des séries de caractères comme dans des phrases réelles. En ce qui est des séries de caractères, nous avons choisi d'utiliser deux fichiers de textes : un roman et un texte hétérogène. On observe que l'algorithme d'encodage par paires d'octets permet, dans les deux cas, un taux de compression plus élevé (~46% vs ~44% dans le cas d'un roman et ~27% vs ~45% dans le cas d'un texte hétérogène). En ce concerne le document PDF, on observe un taux de compression de 0,06% pour l'algorithme par paires d'octets et un taux de 1,34% dans l'autre cas. À la suite de ces résultats, on peut constater que la méthode par paires d'octets s'avère généralement plus efficace lors de la compression de séries de caractères. Cependant, on s'aperçoit que dans le cas d'un PDF, l'algorithme arithmétique est plus performant, mais dans les deux cas le taux de compression n'est pas réellement significatif. Les images Bitmap, quant à elles, avaient un taux de compression équivalent à ~99% et ~38% pour la méthode par paires d'octets ainsi qu'un taux de ~87% et 1,19% pour arithmétique. De plus, le fichier audio WAV présente des taux de compression égale à ~41% pour paires d'octets et ~39% pour arithmétique. Ces résultats indiquent que la méthode d'encodage par paires d'octets permet une compression plus performante dans trois cas précédents.

Notre hypothèse originale contient 2 parties. La première partie est infirmée comme 80% de nos tests concluent que l'encodage par paires d'octets est plus performant de façon générale. En ce qui concerne la deuxième partie de notre hypothèse, on a pu conclure que l'algorithme de paires d'octets est légèrement plus performant dans le cas de la compression d'une série de caractères. Cette dernière est alors confirmée.

Retour hypothèse 2 (voir Tableau 3 et Graphique 1):

Notre deuxième hypothèse énonce que la méthode de codage par paires d'octets est plus efficace lorsqu'on a des symboles qui se répètent souvent. Pour ce qui est des différentes images à l'étude, nous avons utilisé deux images PNG, deux Bitmaps et un GIF. Tout d'abord, pour les images PNG, nous avons employé une image en couleur (PNG 1) et une en noir et blanc (PNG 2). Ces images avaient respectivement comme taux de compression ~1,81% et ~86% pour la méthode par paires d'octets ainsi qu'un taux de ~1,32% et ~46% pour l'encodage arithmétique. Les images Bitmap, quant à elles, avaient un taux de ~32% (BMP 1) et ~38% (BMP 2) pour l'encodage par paires d'octets

et un taux de 1,24% et 1,19% pour la méthode arithmétique. Finalement, dans le cas d'un GIF, on observe une compression de 0,79% pour la méthode arithmétique et une décompression de 0,93% au niveau de l'encodage par paires d'octets. Le taux de compression est négatif comme la taille de l'échantillon compressée était plus grande que la traille initiale de celle-ci. Bref, on observe une hausse significative du taux de compression pour la méthode d'encodage par paires d'octets dans la plupart des cas précédents. Dans le cas de l'image GIF, il semble que le stockage du dictionnaire dans le fichier compressé par l'encodage par paires d'octets requière davantage d'espace, et on observe même une taille « compressée » plus grande que l'originale. Avec l'algorithme arithmétique, il y a un léger gain. C'est donc dire que le format GIF semble être déjà très compressé à la base.

On se trouve à infirmer partiellement notre hypothèse. Effectivement, notre deuxième hypothèse stipule que la méthode arithmétique s'avère plus efficace dans le cas d'une image avec des couleurs plus variées, telles que les photographies. Ceci n'est cependant pas le cas selon nos résultats. De plus, nous avons effectué cette expérience avec un texte homogène et un texte contenant uniquement une série d'a et b. Dans le premier cas, on observe un taux de compression de ~99% pour l'encodage par paires d'octets et un taux de ~97% pour la méthode arithmétique. En ce qui concerne le deuxième cas, nous avons eu un taux de ~99% et ~87% respectivement. Bref, ces deux résultats s'alignent avec notre hypothèse. En effet, on stipule que l'encodage par paires d'octets est plus efficace lors de la compression de textes.

Bref, les résultats obtenus lors de cette expérience nous permettent de confirmer, en bonne partie, cette hypothèse. Cependant, à la suite de nos résultats, il faut infirmer la partie de notre hypothèse où on mentionne que dans le cas d'une image avec des couleurs plus variées, la méthode arithmétique se présente comme une meilleure option.

Retour hypothèse 3 (voir Tableau 4, Graphique 2 et Graphique 3) :

Notre troisième hypothèse stipule que la méthode de codage par paires d'octets est plus rapide à exécuter que la méthode de codage arithmétique. Pour cette expérience nous avons exécuté des tests sur 15 échantillons afin de mesurer le temps d'exécution des deux algorithmes pour compresser un fichier. Afin de confirmer cette hypothèse on cherche à avoir au moins 70% des échantillons qui démontre que l'encodage par paires d'octets est plus vite à exécuter. En regardant à nos résultats, on observe que 9 des 15 échantillons testés ont été compressés plus rapidement par l'algorithme d'encodage par paires d'octets, ce qui donne un taux de 60%. On doit alors infirmer cette hypothèse comme ce taux ne dépasse pas notre seuil de 70%. Cependant, on observe que dans le cas des textes (sauf le roman) ainsi que du document PDF, la méthode par paires d'octets s'avère plus rapide lors de la compression ce qui concorde avec nos notes de cours. Effectivement, étant une méthode de type dictionnaire, l'encodage par paires d'octets est surtout utilisé pour des textes, car des séquences de caractères qui se retrouvent à plusieurs reprises sont plus susceptibles de survenir.

Afin de pousser notre analyse, le graphique 3 démontre le temps d'exécution des algorithmes d'encodage par paires d'octets et arithmétique en fonction de la taille initiale des échantillons. On observe, naturellement, que dans le cas des deux algorithmes, le temps d'exécution augmente en fonction de la taille initiale du fichier à compresser. Cependant, dans le cas de l'encodage par paires d'octets, cette augmentation se fait beaucoup plus rapidement. Effectivement, pour un fichier avec une taille initiale d'environ $3x10^7$ octets prendra environ 4 secondes pour se compresser avec l'algorithme arithmétique et environ 10 secondes avec l'algorithme par paires d'octets. Alors pour compresser le même fichier ça nous prendrait environ deux fois plus de temps avec la méthode d'encodage par paires d'octets.

Retour hypothèse 4 (voir Tableau 5 et Tableau 6) :

Notre quatrième hypothèse indique que, dans des cas où la distribution est davantage équilibrée, l'encodage par paires d'octets devrait s'avérer plus efficace. L'expérience consistait à calculer le taux de compression pour 4 échantillons de format TXT. En ce qui concerne le roman, on observe un taux de compression plus élevé pour l'encodage par paires d'octets avec un taux de ~46%. Le texte homogène, quant à lui, avait un taux de compression de ~99% pour la méthode par paires d'octets et un taux de ~97% pour la méthode arithmétique. Ensuite, pour un texte contenant uniquement des a et des, on observe un taux de ~99% pour l'encodage par paires d'octets et ~87% pour arithmétique. Finalement, pour un texte hétérogène, ces taux descendent à ~27% et ~25% respectivement.

En bref, ces résultats démontrent que dans les quatre cas, la méthode par paires d'octets s'avère plus efficace en termes de taux de compression. Nous constatons un meilleur taux de compression lorsqu'il y a une petite quantité de caractères, et un moins bon taux lorsqu'on compresse des textes hétérogènes, ce qui tend donc à infirmer notre hypothèse.

Considérations supplémentaires

Nous avons également choisi de faire la batterie de tests en appliquant successivement les deux algorithmes. Nous avons constaté qu'appliquer l'encodage arithmétique en premier et ensuite l'encodage par paires d'octets n'améliorait pas le taux de compression, en aucun cas. Cependant, le contraire, soit l'encodage par paires d'octets suivi de l'encodage arithmétique, s'est avéré plus efficace dans la plupart des cas. Les cas où ce n'était pas avantageux sont les cas de l'échiquier noir et blanc en format PNG, les images BMP des couleurs, du fond d'écran avec un petit éventail de couleurs, et de la photo de plage, l'image JPG en couleurs, puis la vidéo en format MOV. Dans certains cas, nous avons observé un gain significatif, jusqu'à près de 50% dans le cas de l'échiquier noir et blanc en format BMP.

En faisant nos recherches, nous avons constaté que les algorithmes que nous avons étudiés sont utilisés dans différentes sphères technologiques. Notamment, l'encodage arithmétique est utilisé dans l'interprétation machine des langages naturels. (Jaswal,

2019) Par ailleurs, nous avons également constaté que l'application successive d'algorithmes d'encodage est une pratique courante, car nous avons trouvé des références à ce type d'utilisation en faisant nos recherches sur internet. En fait, nous avons même trouvé un travail de recherche qui combinait, précisément comme nous avons fait, l'encodage par paires d'octets suivi de l'encodage arithmétique. (Gandhi, 2014)

Le tableau suivant présente nos données de tailles, soit la taille initiale, celles obtenues après l'encodage par paires d'octets, l'encodage arithmétique et les deux combinaisons possibles. Toutes les valeurs présentées dans le tableau sont en octets.

Tableau 7 : Tailles obtenues après encodages (en octets)

Format - Description	Taille initiale	BPE	Arithmétique	Ari→BPE	BPE→Ari
WAV - Fichier audio non compressé	827618	486478	504661	512293	450294
PNG - Image d'un horaire Poly	142730	140150	140842	142924	140010
PNG - Image d'un damier (noir et blanc)	2196	310	1192	1216	312
BMP - Image d'une grille de couleurs	27648122	18829111	27305531	27638064	19022465
BMP - Image d'un damier (noir et blanc)	1000138	14232	125447	97859	8604
BMP - Image d'un fond d'écran HP	6912122	4555337	8770273	6419134	4640490
BMP - Image d'une plage (en couleur)	14745722	9211326	14570166	14750536	9558614
GIF - Image animée	3395028	3426597	3368039	3410766	3402474
JPG - Image personnelle en couleurs	644587	652409	649891	659900	655676
TXT - Roman	704143	380685	396772	402925	354525
TXT - Texte homogène	11409	123	348	354	104
TXT - Texte abababab	36961	270	4919	4986	198
TXT - Texte hétérogène	2146	1559	1614	1644	1487
MOV - Vidéo personnelle	1384914	1248251	1327655	1313521	1250903
PDF - Document	1366213	1365408	1347864	1368442	1355927

Références

Gage, P. (1994, Février 1er). *A New Algorithm for Data Compression*. Récupéré sur Dr Dobbs - The World of Software Development: https://www.drdobbs.com/2441/DrDobbs/a-new-algorithm-for-data-compression/184402829?pgno=3

Gandhi, J. D. (2014, Mars). Achieving Better Compression Applying Index-based Byte-Pair Transformation before Arithmetic Coding. *International Journal of Computer Applications (0975 –8887) Vol. 90*. Récupéré sur Library.net: https://1library.net/document/zke36lpz-achieving-better-compression-applying-index-transformation-arithmetic-coding.html#reference-content

Glen G. Langdon, J. (1984, March). An Introduction to Arithmetic Coding. *IBM J. Res. Develop. Vol 28 No 2*, pp. 135-149.

Jaswal, A. S. (2019, Novembre 22). *Byte Pair Encoding — The Dark Horse of Modern NLP*. Récupéré sur Towards Data Science: https://towardsdatascience.com/byte-pair-encoding-the-dark-horse-of-modern-nlp-eb36c7df4f10

Nelson, M. (2014, Novembre 2014). *Data Compression with Arithmetic Encoding | Dr Dobbs*. Récupéré sur Dr Dobbs - The World of Software Development: https://www.drdobbs.com/cpp/data-compression-with-arithmetic-encodin/240169251?pgno=1&fbclid=IwAR2HTvbbuuYpM_WOoL4ByMgjHufiy01TK2_LM4oAMV0lj8rLJpHqO1LEhmo

Shibata, Y. &. (1999). *Byte Pair Encoding: A Text Compression Scheme That Accelerates Pattern Matching*. Récupéré sur Research Gate: https://www.researchgate.net/publication/2310624_Byte_Pair_Encoding_A_Text_Compression_Scheme_That_Accelerates_Pattern_Matching

Vitter, P. G. (1992). Practical Implementations of Arithmetic Coding. *Image and Text Compression*, pp. 85-112.