INF8770 - Technologies Multimédia

Travail Pratique 1

07/02/2022

Groupe 01

Soumis par :

Mohamed-Ali Messedi – 1946597

Thomas Caron – 1944066

Question 1 : *Formulez une liste d'hypothèses à tester pour caractériser la performance des deux méthodes. Expliquez vos différentes hypothèses. Exemple : La méthode 1 donnera un code plus compact que la méthode 2 si le message...*

Dans le cadre de ce premier travail pratique, nous devons caractériser deux méthodes de codage (par plage et LZ77) et déterminer quelle méthode de codage est meilleure dans quelles circonstances. Ainsi, pour caractériser la performance des deux méthodes, nous pouvons poser les hypothèses suivantes:

Hypothèse 1: Pour deux sources d'entropies différentes, la méthode de codage par plage va donner une meilleure compression pour une entropie basse.

Hypothèse 2: Pour deux chaînes avec deux nombres de symboles variables, la méthode LZ77 va donner une meilleure compression pour la chaîne ayant un plus grand nombre de symboles. Ceci est dû au fait que la méthode de codage par plage est plus efficace lorsqu'on a moins de symboles.

Selon la matière vue en cours, la méthode de codage par plage est efficace si:

- Peu de symboles sont utilisés (en général cette méthode est utilisée sur les images en noir et blanc).
- Les symboles sont consécutifs.

Aussi, la méthode de Codage LZ77 efficace si :

- série de caractères se répète,
- séries de caractères chaînés ou par trop loin les une des autres

Ainsi, nous allons pouvoir effectuer une série de tests pour montrer que nos hypothèses sont valides. Dans le cas de LZ77, nous allons utiliser une taille de dictionnaire fixe (plus on augmente la taille du dictionnaire, plus celui-ci prend de temps avant de terminer). En gardant une taille de dictionnaire fixe, on s'assure d'ailleurs que ce n'est pas un facteur qui va venir influencer nos différentes hypothèses. Pour ce qui est du codage par plage, on utilise une taille de compteur de 8 bits, ce qui est la taille typique pour cet algorithme.

Question 2 : *Décrivez en détails les expériences que vous allez réaliser pour vérifier les hypothèses formulées (bases de données utilisées, contenu des messages, critère d'évaluation, code informatique utilisé, etc.). Les expériences doivent être menées sur deux des trois (3) types de données suivantes : Fichier WAV, Fichier texte (Chaîne de caractères significativement longue) ou Images de formats quelconques (JPEG, PNG, ...)*

2.1 : Choix des types de fichiers pour l'étude

Pour réaliser les hypothèses formulées dans la section précédente, nous avons réalisé différentes expériences sur plusieurs types de données, soit le format WAV, des fichiers texte (format .TXT) ainsi que des images de format BMP. Nous expliquerons donc comment nous avons généré nos données ainsi que les différents critères utilisés lors de nos tests.

Pour ce qui est de la génération de données, nous avons utilisé différentes méthodes pour obtenir des données testables.

2.2 : Méthodes utilisées pour générer les échantillons

Pour les fichiers de texte, nous avons utilisé la commande suivante pour générer une chaîne de taille variable avec une série caractères limités:

$ cat /dev/urandom | tr -dc {start}-{end} | head -c 5000

Les variables **start** et **end** représente le début et la fin de l'intervalle de caractères. Par exemple, si on souhaite générer une chaîne contenant seulement des A, il suffirait de seulement mettre **"A"** alors que si nous voulons un dictionnaire allant de A à C, il faudrait mettre **"A-C"**. L'utilisation du générateur de nombre aléatoire du système nous assure d'obtenir des données complètement aléatoires, donc seul le nombre de caractères dans le dictionnaire.
Pour les tests d'entropie, nous avons créé un script python nous permettant de définir une série de caractères, ainsi qu'un poids attribué à celui-ci. Ceci nous permet donc de plus ou moins prédire l'entropie du résultat que nous allons obtenir avec la formule suivante:

$$L(C) = - \Sigma_{p(x) > 0} p(x) log_2 p(x)$$

Pour ce qui est des images, nous avons opté d'utiliser le format Bitmap (.BMP) qui est utile car celui-ci garde directement la valeur de chaque pixel en mémoire et ne fait aucune optimisation comparé à d'autres formats plus populaires (PNG, JPG). Nous avons donc utilisé plusieurs sources en ligne [1] [2].

Pour le format WAV, nous n'avons pas généré les fichiers, ils ont été récupérés d'une banque de sons [3]. Nous avons testé différents types de sons, avec des séquences sonores ayant des fréquences à complexité variable. En effet, certaines séquences étaient composées d'oscillations répétitives, tandis que d'autres avaient des motifs sonores moins prévisibles pour mettre nos méthodes de compression à l'épreuve.

Le type d'échantillons choisis sera détaillé dans la partie suivante.

2.3 : Type d'échantillon choisis

Pour le texte, la première étape a été d'identifier les catégories de chaînes qui allaient être choisis pour cette étude : il s'agit d'identifier un échantillon pertinent permettant de relever la variation du taux de compression en fonction de deux critères, soit l'entropie et le nombre de symboles générés par la source.

Pour l'entropie, nous avons décidé de former des chaînes à partir d'une source ayant un dictionnaire de quatre symboles telle que S ∈ {A,B,C,D}. Toutes les chaînes ont une taille commune de 500 caractères, afin que la longueur ne soit pas une variable d'intérêt.

Parmi les images choisies, nous avons différents niveaux de couleurs ainsi que de degrés de bruit. Le choix évident a d'abord été de traiter une image de couleur unie (blanche dans notre cas). Ensuite, nous avons choisi d'autres cas, comme divisé en quatre couleurs unies, et des images avec des bruits différents. La première est uniquement en noir et blanc (similaire à un code QR) tandis que les autres ont plus de nuances dans les pixels.

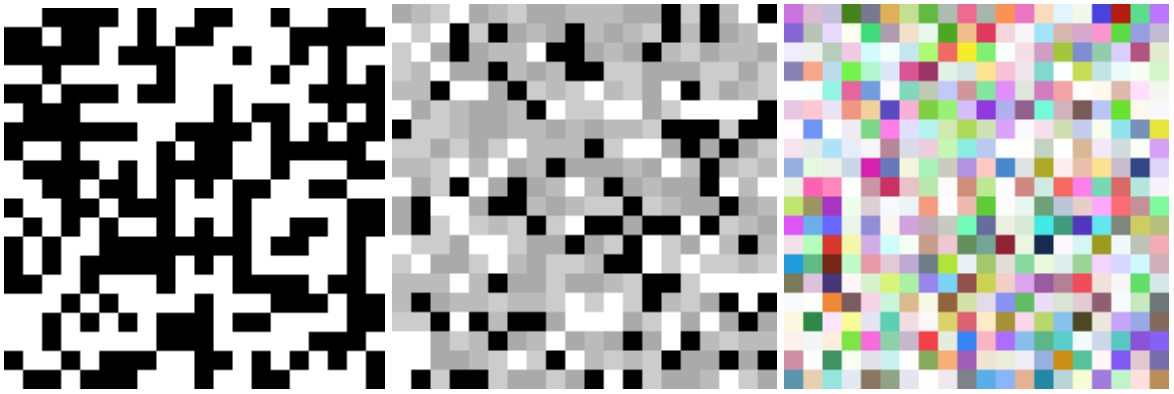

Figure 1 : Exemples d'images utilisées

Enfin pour le son, nous avons opté d'utiliser trois bruit colorés différents. Ces bruits ont différents spectres sonores, et vont nous permettre de vérifier si la densité de puissance sonore à un impact sur la taille de compression.

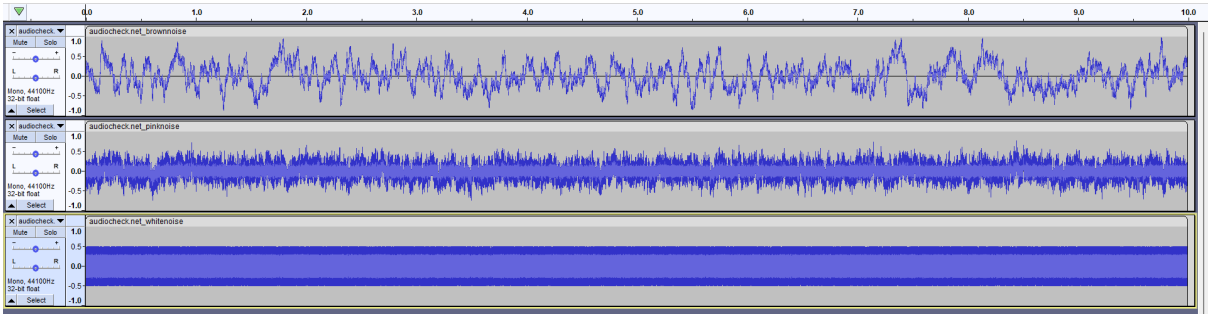

Figure 2 : Exemple d'ondes sonores que nous avons utilisé

Nous avons choisi par la suite, un extrait de Cantina Band #1 de John Williams, qui apparaît notamment dans la saga de films populaires Star Wars. Une séquence musicale n'était pas uniquement composée d'ondes sonores ¨propres¨, elle est plus difficile à prédire car moins répétitive.

Notre deuxième fichier est plus court et contient le bruit d'un seul applaudissement. Lors du choix des données, nous étions assez limités dans la taille des fichiers choisis, car nos algorithmes pour LZ77 et codage par plage n'étaient pas assez performants pour traiter des fichiers de taille significative. Le format WAV étant un format "lossless" (sans pertes), nous avons dû prendre des très petits extraits.

Nous avons d'ailleurs décidé d'ajouter d'autres fichiers audio par la suite pour nous permettre de mieux tester. Notamment différentes "couleurs" de son, ainsi qu'un fichier contenant une onde carrée, et un autre avec une onde sinusoïdale.

Question 3 : *Décrivez les codes informatiques utilisés pour réaliser les expériences. Décrivez, si applicable, comment vous avez adapté les codes informatiques pour réaliser les expériences décrites à la question 2. Donnez les résultats obtenus pour les expériences décrites à la question 2 sous un format approprié.*

Pour l'algorithme du LZ77, nous avons utilisé l'algorithme provenant du dépôt GitHub de l'utilisateur Manassra [4]. Le principe de l'algorithme est le même que celui que nous avons étudié en classe, mais l'enjeu était de trouver un algorithme qui utilise un "look ahead buffer" afin de rendre le processus plus rapide et le calcul plus performant.

Le code comprend deux fonctions principales. Une permettant de compresser le fichier désiré, et l'autre de le décompresser.

Nous n'avons pas modifié les fonctions de compression/décompression, car elles prennent en paramètres un tableau d'octets, ce qui correspond parfaitement à nos besoins.

La modification apportée a permis de créer une partie dans le programme permettant de faire en sorte que le fichier passé en paramètres puisse être converti en tableau d'octets afin de pouvoir traiter ce tableau avec l'algorithme.

La fonction de compression permettait de valider que la taille finale était inférieure à la taille d'origine, et la fonction de décompression était un moyen de vérifier qu'en inversant le processus, le fichier revenait à sa taille originale (pas de données corrompues en chemin)

Pour ce qui est de l'algorithme du codage par plage (Run Length Encoding), nous avons fait notre propre implémentation en se basant sur l'algorithme détaillé dans les notes de cours. Nous utilisons un dictionnaire contenant les valeurs nécessaires, et une taille de compteur de 8 bits, qui est la taille la plus commune selon les diapositives du chapitre.

Nous avons ajouté une fonctionnalité permettant d'écrire nos valeurs encodées dans un fichier, ce qui nous permet de comparer la taille du fichier original avec la taille du fichier encodé. Ceci nous permet d'ailleurs de plus facilement comparer l'algorithme LZ77 avec notre algorithme de codage par plage.

Nous avons utilisé Julia pour compiler les résultats et les traduire en tableau et en graphiques dans le but de visualiser les données. (Voir partie 4).

Question 4 : *Analysez les résultats obtenus et mettez-les en relation avec les hypothèses de la question 1. Est-ce que les hypothèses sont supportées par les résultats ?*

4.1 : Résultats pour les fichiers textes

4.1.1 : Analyse avec entropies variables

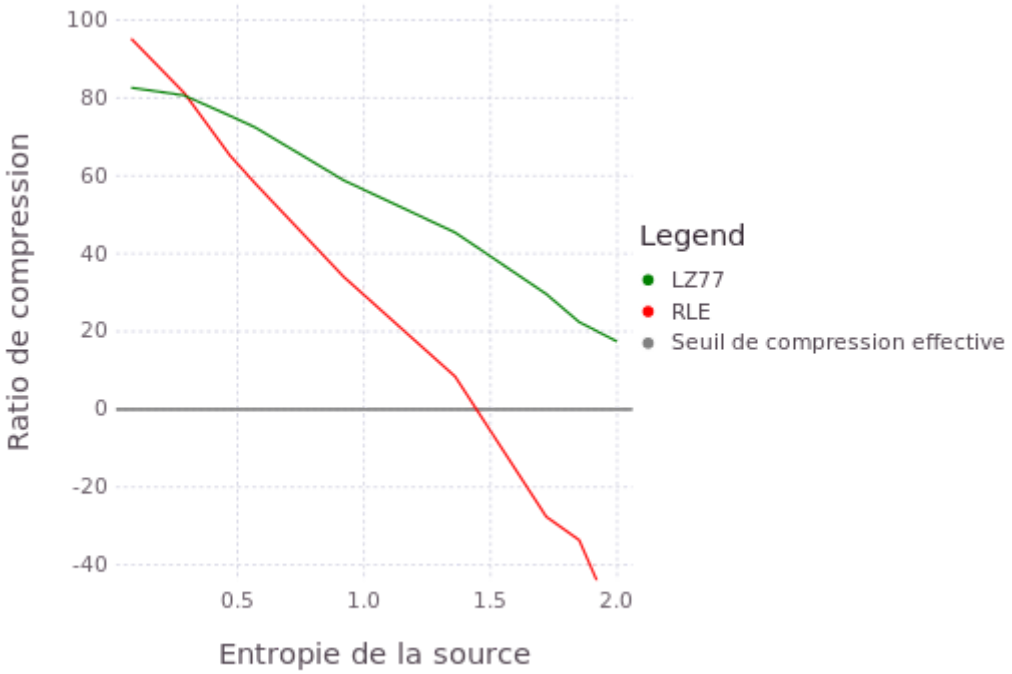

Figure 3 : Ratio de compression en fonction de l'entropie : LZ77 vs RLE

On peut voir ici que pour de très faibles entropies, l'algorithme de compression RLE est extrêmement efficace, avec des ratios presque proches de 100. Par exemple, avec une entropie de 0.08, on a une compression de 24 caractères pour une chaîne de 500 caractères, soit un ratio de compression de 95.2%.

Cependant, à mesure que l'entropie de la source augmente, on voit que les ratios de compression du RLE diminuent significativement, jusqu'à tomber en dessous de la valeur de compression effective : un ratio de compression négatif signifie que la chaîne compressée **est plus longue que la chaîne d'origine** !

L'algorithme LZ77 est plus stable dans l'ensemble : même si le ratio de compression est légèrement plus bas que RLE pour de faibles entropies, on voit qu'il se stabilise à mesure que l'entropie augmente.

Entropie, conclusion : Le RLE est à favoriser pour des chaînes ayant un petit nombre de symboles (par ex. des séquences de bits avec plusieurs 0 et 1 se suivant) tandis que le LZ77 peut être utilisé lorsque l'entropie augmente, mais a tout de même certaines limites.

Valeurs limites expérimentales : Nous avons tout de même essayé d'augmenter l'entropie (valeur de 3.32) en ajoutant des symboles au dictionnaire pour voir si LZ77 allait passer le seuil de compression effective, et il s'avère que c'est également le cas. Cela permet de cerner les limites de cette méthode, étant donné que notre échantillon expérimental est limité par des valeurs d'entropies allant de 0 à 2.

On peut supposer que c'est la raison pour laquelle les deux algorithmes ne sont jamais utilisés seuls, mais plutôt combinés avec d'autres méthodes de compression pour garantir plus d'efficacité.

4.1.2 : Analyse avec nombre variable de symboles de la source

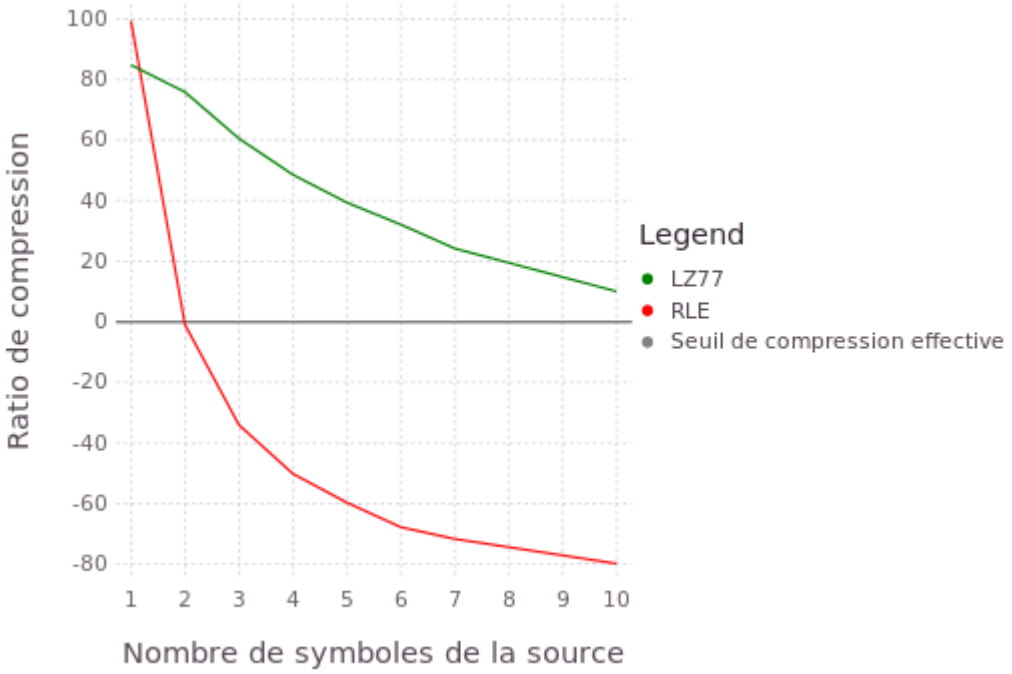

Figure 4 - Ratio de compression en fonction du nombre de symboles : LZ77 vs RLE

Ici, on étudie le ratio de compression pour des chaînes de même longueur, mais au nombre de symboles variable. Il faut également noter que nous avons généré les symboles de façon complètement aléatoire, ce qui permet d'affirmer que l'entropie est maximale pour toutes les chaînes.

On voit que pour une entropie maximale, l'algorithme RLE est très efficace pour un seul symbole, avec une valeur de compression proche de 100%. Cependant, on voit que dès qu'on passe à deux symboles, le ratio de compression devient nul.

En effet, avec une entropie maximale, il y a très peu voire pas de symboles qui se répètent, ce qui veut dire que le principe de l'algorithme RLE n'est plus utile.

Avec LZ77, on voit que la compression est non négligeable même pour une entropie maximale avec 10 symboles. On peut supposer que cela est dû au fait que même avec une entropie maximale de chaque symbole, certaines combinaisons de symboles sont redondantes, ce qui permet à l'algorithme de les exprimer plus efficacement grâce à son dictionnaire.

La comparaison basée sur le nombre de symboles nous permet d'appuyer celle émise par rapport à l'entropie. En effet, l'algorithme de RLE n'est efficace que si deux conditions sont

respectées : un faible nombre de symboles et une faible entropie. Cela montre que cet algorithme est très limité dans ses cas d'utilisation.
L'algorithme LZ77 sera donc plus efficace que le RLE pour des chaînes avec de plus grandes entropies et un nombre de symboles élevé.

4.2 : Résultats pour les fichiers BMP

Row	Image	LZ77_Compression_rate	RLE_Compression_rate
	String31	Float64	Float64
1	White	0.848179	0.992128
2	Black and white noise	0.847735	0.494735
3	Greyscale noise	0.833015	0.175116
4	Color noise	0.755749	-0.945279
5	Dots	0.780521	0.733265

Figure 5 - Ratio de compression en fonction du nombre de symboles : LZ77 vs RLE

Les résultats sont prévisibles : on voit que pour les images ayant très peu de répétitions (White et Dots) , la compression par RLE est optimale.

Elle domine en particulier pour l'image unie blanche comparée à l'algorithme LZ77. Par contre, elle devient très mauvaise quand il s'agit d'images plus complexes comme les images pixelisées ou là il les couleurs se répètent beaucoup moins selon une suite logique.

LZ77 en revanche est beaucoup plus stable au niveau du ratio de compression, on voit que même pour des images ou les couleurs sont plus nombreuses et ou l'ordre d'apparition est plus aléatoire, le ratio de compression reste tout de même convenable.

Cela montre une fois de plus que l'algorithme RLE est extrêmement efficace pour un faible nombre de symboles et une répétition successive élevée de ces symboles.

4.3 : Résultats pour les fichiers WAV

Row	WAV_File	LZ77_Compression_rate	RLE_Compression_rate
	String31	Float64	Float64
1	sine.wav	-0.0413923	-0.965193
2	square.wav	-0.113319	-0.987235
3	brown-noise.wav	-0.113398	-0.990106
4	cantina-band.wav	-0.0570625	-0.981638
5	clap.wav	0.031933	-0.959446
6	pink-noise.wav	-0.121489	-0.990442
7	white-noise.wav	-0.122034	-0.990358

Figure 6 - Ratio de compression en fonction des fichiers WAV : LZ77 vs RLE

Pour ce qui est des fichiers WAV, les résultats sont quand même surprenants. On remarque que pour ce qui est de RLE, nous avons souvent un taux de compression clairement négatif. Le fichier comprimé est presque deux fois plus grand que le fichier original.

Pour l'algorithme LZ77 cependant, les résultats sont moins pires, mais on n'obtient toujours pas une compression satisfaisante.

En regardant la Figure 2 qui se trouve plus haut, on remarque que les signaux des fichiers WAV semblent assez aléatoires, ce qui nous indiquerait que ces fichiers ont une haute entropie.

Ceci nous permet donc de confirmer notre première hypothèse qui indique donc que LZ77 va offrir une meilleure compression pour des fichiers avec une entropie plus élevée. De plus, en utilisant la librairie "scipy" qui nous permet de lire les valeurs de l'onde du fichier WAV, on remarque que ces valeurs obtenues ont souvent un intervalle très grand (ex. [-30000,3000]).

4.4 : Décision sur les hypothèses émises en début de TP

Après avoir effectué les tests sur différents types de données, nous avons remarqué que ces algorithmes n'étaient pas idéales pour certains types de fichiers (le fichier compressé était plus grand que le fichier original), ce qui est notamment vrai pour nos fichiers audio.

Cependant, en regardant nos résultats, il nous est possible de confirmer nos hypothèses de départ, soit que l'algorithme de codage par plage offre une meilleure compression pour un fichier avec une basse entropie, et que le codage par plage est plus efficace si on possède moins de symboles dans le fichier.

En effet, on remarque que pour les fichiers choisis qui avaient une plus haute entropie (les fichiers audio, l'image avec un bruit de couleurs, etc…), l'algorithme LZ77 semble nous offrir un meilleur taux de compression que la méthode de codage par plage, même ce taux n'est pas toujours idéal. Le contraire semble aussi être vrai (si l'entropie est basse, on a un meilleur taux de compression avec le codage par plage), l'image blanche est un très bon exemple de cela.

Pour ce qui est des fichiers avec un grand nombre de symboles, on remarque que pour nos sources ayant très peu de symboles, l'algorithme RLE est extrêmement efficace. La Figure 4 se trouvant plus haut illustre parfaitement ce principe. On remarque d'ailleurs que le contraire est aussi vrai, car l'algorithme LZ77 semble diminuer linéairement alors que le codage par plage semble diminuer de manière exponentielle.

Références:

- South Carolina Department of Mathematics (2011) “BMP Files Microsoft Bitmap files” [En Ligne]. Disponible: https://people.math.sc.edu/Burkardt/data/bmp/bmp.html.
- Online Random Tools (s.d.) “Random Bitmap Generator” [En Ligne]. Disponible: https://onlinerandomtools.com/generate-random-bitmap
- FindSounds (s.d.) “Search the Web for Sounds” [En Ligne. Disponible: https://www.findsounds.com/ISAPI/search.dll?keywords=sine+wave
- W. Manassra (2019) “Python LZ77 Compressor” [En Ligne]. Disponible: https://github.com/manassra/LZ77-Compressor