POLYTECHNIQUE
MONTRÉAL

INF8770
Technologies multimédias

Travail pratique #1 – Méthodes de codage

Soumis par :
Gaudet, Alexandre – 2090935
Sant'anna, Abdel – 2088865

Le 5 février 2023

Question 1

Contrairement à plusieurs méthodes de compression qui encodent un seul caractère à la fois, le codage LZW excelle en traitant plusieurs caractères simultanément grâce à l'utilisation d'un dictionnaire dynamique. Ce dernier permet de coder des chaînes de caractères répétitives avec un même code. Cette approche de compression se révèle particulièrement performante pour des données comportant des motifs récurrents.

Pour les textes, les numéros 1 et 2 présentent des répétitions assez évidentes de « A » et « B » qui peuvent être bien codées par LZW. Le taux de compression devrait être plus élevé pour le 2, car l'alphabet est plus grand (A, B et C), ce qui implique un code binaire initial plus long. Pour le texte 3, il y a quelque répétitions qui peuvent être bien codées par LZW, tel que les « O », mais quand même moins que le texte 2 par exemple. Pour le texte 4, la distribution des lettres est plus aléatoire et il y a peu de répétitions, ce qui n'en fait pas un bon candidat pour LZW. Pour le texte 5, celui-ci est un peu plus long et il semble y avoir plusieurs mots qui se répètent au fil du texte, ce qui devrait permettre de bien coder les chaînes répétitives avec LZW. Dans l'ensemble, il est important de noter que la véritable force du LZW réside dans sa capacité à encoder de longues sous-chaînes de caractère à la fois. Avec des textes plus étendus, le LZW peut identifier des sous-chaînes plus longues à coder. Or, dans le cas présent, les textes sont relativement courts, ce qui risque de limiter son efficacité. Ainsi, il ne s'agit probablement pas du meilleur choix.

Pour les images, l'algorithme LZW devrait être extrêmement efficace sur les numéros 2 et 5. En effet, les couleurs uniformes (fond noir, cercle rouge) sont un cas idéal pour LZW, car il est possible de coder de très longues sous-chaînes de pixels identiques avec le même code. De façon similaire, LZW est parfait pour le fond blanc de l'image 3. La compression risque d'être un peu limitée par la pomme, qui est plus détaillée, ce qui devrait donner un bon taux, mais plus faible que 2 et 5. Pour l'image 4, la palette de couleur est similaire, cependant le niveau de détail est beaucoup plus élevé, ce qui limite grandement le nombre de motifs récurrents dans l'image. Ainsi, le taux de compression devrait être correct, mais sans plus. Finalement, l'image 1 est un gradient de couleur avec une énorme variation des valeurs RGB et très peu de répétition spatiale. C'est presque un pire cas pour LZW, ce qui risque fortement de causer un taux de compression négatif. Tout compte fait, LZW semble être un bon choix pour les images, sauf pour l'image 1 pour laquelle cela est loin d'être optimal.

Question 2

L'implémentation choisie pour LZW s'inspire largement du cours [1], avec quelques ajustements mineurs visant à optimiser l'ensemble. Le tableau 1 présente les taux de compression obtenus après avoir appliqué LZW sur les données fournies. Comme anticipé, le texte 4 connaît une compression quasi nulle en raison du faible nombre de répétitions. Les textes 2 et 5 affichent les meilleurs taux de compression, ce qui est cohérent. Comme abordé dans la question 1, le texte 5, légèrement plus long et riche en répétitions de mots, permet l'encodage de longues sous-chaînes de caractères. De même, le texte 2 contient de nombreuses répétitions. L'algorithme parvient également à compresser les messages 1 et 3, qui contiennent eux aussi des répétitions. Comme mentionné précédemment, LZW est de toute évidence limité par la longueur des textes. Bien que certains présentent des répétitions évidentes, les taux de compression sont nettement inférieurs à ceux des images, car celles-ci contiennent comparativement beaucoup plus de données.

En ce qui concerne les images, les meilleurs taux de compression sont obtenus avec les images 2 et 5, en accord avec les observations précédentes selon lesquelles l'utilisation de couleurs uniformes se compresse très bien avec LZW en raison de la répétition des mêmes pixels. Comme évoqué, l'image 3 obtient également un bon taux de compression grâce au fond blanc uniforme. Le codage LZW s'est avéré moins efficace sur l'image 4, en accord avec les observations de la question 1. Enfin, l'image 1 affiche un taux de compression désastreux, occupant même plus d'espace que l'originale. Cela n'est pas du tout surprenant, car, comme mentionné précédemment, il s'agit essentiellement d'un des pires scénarios pour LZW, avec d'énormes variations des valeurs RGB et très peu de répétition.

Pour chaque compression, les temps d'encodage ont été mesurés et sont présentés au tableau 1. L'algorithme LZW se révèle manifestement très rapide pour les données fournies. Les temps d'encodage pour les textes sont essentiellement négligeables. Logiquement, l'encodage est plus long pour les images qui contiennent plus de données, mais même pour l'image 4, l'encodage se fait en moins d'une seconde. Cette rapidité s'explique par le fait que LZW n'est pas un algorithme coûteux à implémenter, nécessitant simplement la gestion d'un dictionnaire dynamique et la recherche de la sous-chaîne la plus longue.

Tableau 1. Performance de LZW

Source	Taux de compression	Temps d'encodage (secondes)
texte_1	0,119	0,0010
texte _2	0,266	0,0010
texte _3	0,108	0,0010
texte _4	0,001	0,0010
texte _5	0,266	0,0024
image_1	-0,332	0,2940
image _2	0,969	0,0366
image _3	0,560	0,6989
image _4	0,151	0,9969
image _5	0,986	0,1697

Question 3

L'utilisation de LZW sur les données fournies a connu quelques limitations. Premièrement, pour l'image 1, le taux de compression obtenu est négatif, ce qu'il est assurément souhaitable d'éviter. Deuxièmement, les taux de compression des fichiers textes sont bas, surtout comparativement aux images. Le fait que les textes sont très courts rend plus difficile de trouver de longues sous-chaines et profiter au maximum des avantages de LZW. Il est surement possible de faire mieux à ce niveau.

En tenant compte de ces faiblesses, le codage Huffman, qui est basé sur l'entropie, semble être une alternative intéressante. Cet algorithme permet de coder les caractères les plus fréquents avec de petits codes binaires. Dans le cas d'une distribution parfaitement uniforme, Huffman est équivalent au codage binaire. Cela permet d'éviter d'avoir une compression négative pour l'image 1, qui se rapproche d'une distribution uniforme et cause des problèmes avec LZW en raison du manque de répétition. Aussi, Huffman produit le meilleur code binaire possible basé sur la fréquence de chaque caractère. Son efficacité n'est donc pas limitée par la longueur des textes comme c'était le cas pour LZW. De plus, plusieurs des textes ont des lettres plus fréquentes. Par exemple, le texte 3 contient beaucoup de « O », mais très peu de « B » ou de « D ». Finalement, Huffman devrait aussi bien fonctionner sur certaines des images. Notamment, l'image 3 et l'image 5 qui ont respectivement des fonds uniformes blanc et noir, ce qui implique une haute fréquence de faibles ou hautes valeurs RGB.

L'implémentation choisie est directement basée sur celle utilisée dans le cadre du cours [2]. Le tableau 2 présente les taux de compression obtenus avec Huffman. Comme attendu, l'algorithme a bien fonctionné pour l'image 3 et 5. L'image 1 a aussi un taux de compression positif. Les textes ont un bon taux de compression, autre que le texte 1 pour lequel Huffman ne semble pas être optimal.

Le tableau 2 présente aussi le temps d'encodage pour chaque fichier. Celui-ci est similaire à LZW pour les textes, mais Huffman est généralement plus couteux en termes de temps pour les images. Contrairement à LZW, il y a un coût supplémentaire pour construire l'arbre et générer les codes préfixes avant de coder les données.

Tableau 2. Performance de Huffman

Source	Taux de compression	Temps d'encodage (secondes)
texte_1	0,000	0,0010
texte _2	0,234	0,0010
texte _3	0,358	0,0010
texte _4	0,188	0,0010
texte _5	0,300	0,0022
image_1	0,084	0,7829
image _2	0,000	0,0173
image _3	0,464	3,4592
image _4	0,031	3,0442
image _5	0,695	0,4126

Question 4

Il convient maintenant de comparer les performances des deux méthodes de compression. La figure 1 présente les taux de compression obtenus par chacun des algorithmes à titre de comparaison. Pour les fichiers textes, le codage Huffman obtient un meilleur taux de compression pour les numéros 3, 4 et 5, alors que LZW a un meilleur taux pour les numéros 1 et 2. La méthode la plus constante est certainement Huffman qui obtient un taux supérieur à 0,15 pour quatre textes, contre seulement deux pour LZW. Le tableau 3 présente le gain moyen de compression, qui est obtenu en faisant la moyenne des différences entre les taux de compression. Ainsi, en moyenne, l'utilisation de Huffman au lieu de LZW aboutit en un gain moyen de 0,064 pour les fichiers utilisés. Considérant ces facteurs, Huffman est la méthode la plus performante pour les textes. Ceci n'est pas surprenant, considérant que LZW est quelque peu limité par la

longueur des textes. En effet, sur des textes plus courts, il est moins possible d'encoder de longues sous-chaines à la fois. À l'inverse, Huffman n'est pas limité par ce facteur et peu pleinement tirer avantage des fréquences plus élevées de certaines lettres. La seule limitation de Huffman est sur le texte 1, car le petit alphabet (2 caractères) fait que Huffman est équivalent au code binaire classique.

Pour les images, Huffman obtient seulement un meilleur taux de compression pour l'image 1, alors que LZW est meilleur pour le reste. C'est aussi LZW qui a le plus d'images avec un taux supérieur à 0,15, avec quatre comparativement à deux pour Huffman. De plus, en moyenne, le gain de compression est négatif en passant de LZW à Huffman pour les images, tel que présenté au tableau 3. En considérant ces facteurs, LZW semble être le meilleur choix pour les images. Cependant, il faut noter que le taux de compression de LZW est nettement négatif pour l'image 1. Si c'est quelque chose qu'il faut absolument éviter, il pourrait être plus intéressant de considérer Huffman.

La grande efficacité de LZW sur les images fournies peut s'expliquer par la capacité à coder de très longues sous-chaines de pixels qui se répètent lors de l'utilisation de couleurs uniformes. Notamment, sur les images 2 et 5, le fond noir et le cercle rouge sont extrêmement bien codés par LZW qui atteint d'impressionnants taux de compression supérieurs à 0,95. Bien que Huffman soit efficace sur l'image 5 due aux répétitions plus fréquentes de certains pixels, il est limité par le fait de coder un seul symbole à la fois et ne peut donc pas atteindre des taux aussi élevés que LZW, qui en code plusieurs ensemble. Cependant, LZW a aussi la limitation d'être très mauvais dans le cas où il y a très peu de répétition spatiale et produit un taux de compression négatif, il est fortement dépendant de l'axe par lequel on décide de l'utiliser afin de favoriser la redondance. À l'inverse, Huffman est équivalent au code binaire classique dans le pire cas (image 2).

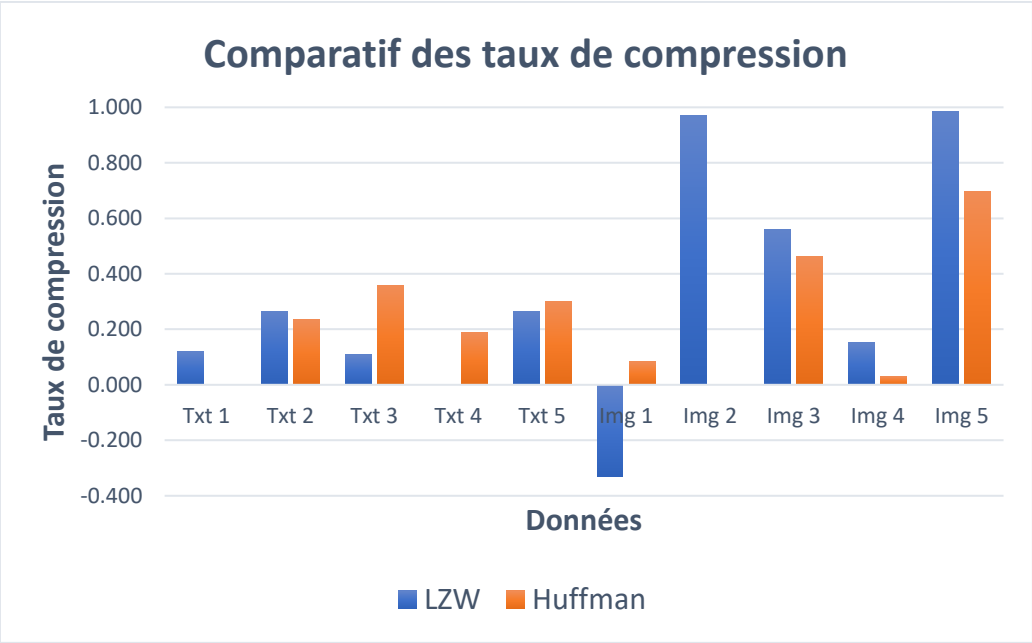

Figure 1. Comparatif des taux de compression

Tableau 3. Gain moyen de compression avec Huffman

Type de données	Gain moyen de compression
Textes	0,064
Images	- 0,212

Références

[1] Bilodeau, G (2020) Codage LZW [Code source].
https://github.com/gabilodeau/INF8770/blob/master/Codage%20LZW.ipynb

[2] Bilodeau, G (2024) Codage Huffman [Code source].
https://github.com/gabilodeau/INF8770/blob/master/Codage%20Huffman.ipynb