TP1 INF8770

Valentin BOUIS

Benjamin HEINEN

Question 1

Hypothèse 1 :

On suppose que moins il y aura de teintes de couleur différentes (exemple : image en noir et blanc), plus l’algorithme de codage par paire d’octets sera efficace. En effet il sera capable de remplacer les répétitions de couleurs par un autre symbole. Dans le cas d’une image en noir et blanc, les répétitions cote a cote de noir et de blanc seraient alors remplacées par un autre symbole, ce qui diminuerait grandement la taille de l’image. Au contraire pour l’algorithme de codage prédictif, l’objectif est d’obtenir le plus de zéro possible dans l’erreur. Ainsi pour une image en noir et blanc on essayerait de transformer les 1 (blanc) en 0 (noir), mais les valeurs initialement à 0 risquent également d’être modifié. On pense donc que le codage par paire d’octets compressera mieux une image avec peu de teintes de couleurs que le codage prédictif.

Hypothèse 2 :

Plus l’image aura de couleurs distinctes qui ne se répètent pas selon un motif, plus le codage prédictif compressera mieux que le codage par paire d’octets. En effet le codage par paire d’octets ne sera pas capable de simplifier des duo de pixels s’il ne trouve pas de motif répété.

Voici un exemple :

Code : 0 1 2 3 4 5

Version paire d’octets : pas de simplification possible (pas de paire identique)

Version prédictif : 0 0 0 0 0 0 dans le cas ou x(n) = x(n-1) - 1

On voit clairement que dans le cas extreme ou aucun motif n’apparait l’algorithme de codage prédictif est bien meilleur que celui par paire d’octets.

Hypothèse 3 :

On pense que le temps d’exécution de l’algorithme de codage par paire d’octets sera plus lent que l’algorithme de codage prédictif lors de la compression. En effet le codage par paire d’octets nécessite un algorithme récursif ainsi que d’établir une map des combinaisons de 2 pixels possibles ainsi que leur nombre d’occurrences. En revanche le codage prédictif ne nécessite seulement que le calcul de l’erreur.

Hypothèse 4 :

Le dernier critère serait la taille occupée dans la mémoire lors de la compression. On pense que celui-ci serait plus faible pour l’algorithme de codage prédictif (en admettant qu’on combine celui-ci à l’algorithme de Huffman) car le dictionnaire nécessaire à la compression par paire d’octets sera plus grand puisque il y a plus de duo de symboles (chaque symbole apparait dans 2 duo) que de symboles.

Question 2

Test hypothèse 1 :

Pour tester l’efficacité de compression, nous allons tout simplement comparer la valeur du quotient de compression pour chaque algorithme. Celui-ci est égal à :

Taille de l’image initiale/Taille de l’image compressée

On regardera alors ce quotient pour chaque exemplaire d’image afin de voir si l’algorithme de codage par paire d’octets est bien plus efficace que celui prédictif selon les situations.

Test hypothèse 2 :

De la même manière que pour l’hypothèse 1 ce sera le quotient de compression que nous comparerons ici aussi.

Test hypothèse 3 :

Pour tester le temps d’exécution de chacun des algorithmes, nous allons tout simplement comparer le temps pris par chacun pour compresser une image. Pour cela nous utiliserons le module Time de python. Nous donnerons la même image à traiter aux deux algorithmes. Nous lancerons le timer au commencement de l’algorithme (après l’import de l’image) et l’arrêterons quand celui-ci aura fini de compresser l’image.

Test hypothèse 4 :

Afin de comparer la taille prise en mémoire par chaque algorithme, nous allons utiliser la librairie guppy de python. Celle-ci nous permet d’obtenir de nombreuses informations liées à la consommation en mémoire du programme.

Question 3

Tout d’abord, nous avons réalisé l’algorithme par paire d’octets en nous inspirant de celui que nous avions à notre disposition sur le Github du cours. Néanmoins, nous l’avons modifié afin de pouvoir traité une image. Afin de simplifier les calculs, nous avons fait le choix de choisir une image uniquement composée de teintes de gris, ce qui permet de réduire la taille de l’image par trois. En effet, celle-ci n’est composé que d’un octet par pixel, au lieu des trois que l’on a classiquement avec le modèle RGB.

Par la suite, nous avons fait le choix de choisir une image de petite taille (64x64 pixel) plutôt qu’une grande image. Ce choix a été fait de façon expérimental, lorsque l’on c’est rendu compte que l’algorithme par paire d’octets prenait beaucoup de temps.

Ensuite, nous avons commencé à développer l’algorithme prédictif. Cela fut assez rapide en reprenant l’exemple pris sur le Github du cours. Néanmoins, il ne serait pas raisonnable de comparer le codage prédictif avec le codage par paire d’octet. Ainsi, afin de les rendre comparable, nous avons appliqué le codage d’Huffman au codage prédictif. Pour ce faire, nous avons également fait le choix de récupérer l’implémentation du codage Huffman présente sur le Github du cours.

De plus, afin de diversifié les tests possible, nous avons codé des images directement en python.   
Ainsi, une image composé d’une alternance de blanc et de noir sera un tableau de deux dimensions composés de zéro et de uns, pour lequel on change de nombre à chaque index du tableau. De même, une image composé de blanc sur la première moitié puis de noir sur la seconde moitié sera une image avec une première moitié avec que des zéros, puis une seconde moitié avec que des uns.

Tableau 1 : Tigre, taille 33024

|  |  |  |
| --- | --- | --- |
|  | Predictif + huffman | Pair Octets |
| Taille compressée | 17669 | 8724 |
| Temps exécution | 0,4 | 1880,54 |
| Mémoire vive utilisée (Mo) |  |  |

Tableau 2 : Noir et blanc (alterné), taille 33020

|  |  |  |
| --- | --- | --- |
|  | Predictif + huffman | Pair Octets |
| Taille compressée | 33020 | 8 |
| Temps execution | 1,34485 | 0,2498 |
| Mémoire vive utilisée (Mo) |  |  |

Tableau 3 : Noir et blanc (alterné), taille 90

|  |  |  |
| --- | --- | --- |
|  | Predictif + huffman | Pair Octets |
| Taille compressée | 90 | 5 |
| Temps exécution | 0,22321 | 0,00178 |
| Mémoire vive utilisée (Mo) |  |  |

Tableau 4 : Noir et blanc (continu), taille 33020

|  |  |  |
| --- | --- | --- |
|  | Predictif + huffman | Pair Octets |
| Taille compressée | 33020 | 16 |
| Temps execution | 1,17671 | 0,60958 |
| Mémoire vive utilisée (Mo) |  |  |

Tableau 5 : Noir et blanc (continu), taille 90

|  |  |  |
| --- | --- | --- |
|  | Predictif + huffman | Pair Octets |
| Taille compressée | 114 | 10 |
| Temps execution | 0,23677 | 0,005 |
| Mémoire vive utilisée (Mo) |  |  |

Tableau 6 : Dégradé, taille 49152

|  |  |  |
| --- | --- | --- |
|  | Prédictif + Huffman | Pair Octets |
| Taille compressée | 18807 | 8352 |
| Temps exécution | 0,38237 | 2490,66 |
| Mémoire vive utilisée (Mo) | 5,97 | 5,69 |

Question 4

Dans cette partie, nous analyserons les résultats obtenus selon trois catégorie : la compression du message, le temps d’exécution de l’algorithme de codage et la mémoire utilisé. Ensuite, nous comparerons les résultats obtenus à nos hypothèses de la première question.

**Compression du message**

Tout d’abord, dans le tableau 1, nous pouvons constater que pour une image composé de motif (un lion dans notre image), l’algorithme par paire d’octet compresse deux fois mieux le message que l’algorithme prédictif mélangé au codage Huffman, ce qui s’avère très intéressant.

Dans les tableaux 2 et 3, qui affichent les résultats pour une image avec une alternance de noir et de blanc, on se rend compte de quelque chose de particulièrement intrigant. Si le codage par paire d’octets réduit considérablement la taille du message, passant de 33020 caractère à 8, le codage prédictif mélangé au codage Huffman ne réduit absolument pas la taille du message. En réalité, ce résultat semble très naturel. En effet, compte tenu du fait que notre message ne contient que des 0 et des 1, même si le codage prédictif permettait, avec une formule optimal, d’avoir que des 0, chaque symbole du message tiendrait sur un bit, comme pour le message original. Ainsi, le codage Huffman ne serait pas capable d’améliorer la compression. En effet, le codage Huffman permet d’optimiser le nombre nécessaire de bits à chaque symbole. Comme chaque symbole tient déjà sur un bit, cela ne pourrait pas améliorer la compression du message.

Par la suite, on constate dans les tableaux 4 et 5 que le codage prédictif réagit exactement de la même façon que dans les deux tableaux précédent. En revanche, dans le tableau 5, qui ne comporte que 90 éléments, on peut tout de même remarquer que la compression échoue totalement, augmentant ainsi la taille du message. Cela pourrait être améliorer en choisissant une formule plus approprier pour déterminer l’erreur. En effet, la formule actuel produit trois symboles différents dans le message, ce qui s’encode en deux bits. Cela augmente donc légèrement la taille du message.

Enfin, dans le tableau 6, on se rend compte que le codage prédictif, toujours couplé au codage de Huffman, diminue la taille du message d’un facteur proche de 2,60, tandis que le message par paire d’octets diminue la taille du message par un facteur d’environ 5,90.

**Temps d’exécution**

Tout d’abord, dans le tableau 1, qui permet de coder une image composé d’un motif représentant un lion, on se rend compte que le codage par pair d’octet prend un temps considérable par rapport à au codage prédictif, couplé au codage Huffman. En effet, le codage par pair d’octet prend environ 4700 fois plus de temps.

En revanche, dans le cas d’une image composé d’une alternance de noir et de blanc, on peut voir constater, comme présenté dans les tableaux 2 et 3, que le codage par pair d’octet est le plus rapide, et ce qu’importe la quantité d’information. En revanche, le facteur diminue proportionnellement à la longueur du message. Par conséquent, il se peut qu’à partir d’une certaine taille, l’utilisation d’un codage prédictif couplé à un codage Huffman soit plus rapide. Avec les données que nous avons récolté, on se rend compte que lorsque l’on a 33020 données, le codage par pair d’octet est environ 5 fois plus rapide. De plus, quand le message a une longueur de 90, il est 125 fois plus rapide.

Ce constat se reproduit dans les tableaux 4 et 5, qui permettent d’analyser les données pour des images composées d’une moitié blanche et d’une autre moitié noire.

**Mémoire vive utilisé**