

Daniel BORGES
L3.1 Informatique

Projet de Théorie de l'Information

Huffman, Hamming & Simulation de canal

Sommaire

I. Respect du sujet par rapport à l'implémentation

II. Performances

1. Compression

1. Taille compressée

2. Temps d'exécution

2. Destruction & Correction d'erreurs

III. Conclusion

I. Respect du sujet

Tous les programmes (`huffman`, `hamming`, `canal`) ont été implémentés.

Sur `hamming`, l'information à propos de l'existence ou non d'un bloc en trop (au cas ou, par exemple, avec des blocs de 7 bits, il reste 7 bits vides dans le fichier) n'a pas été implémenté. À la correction, il peut donc y avoir un octet supplémentaire dans le fichier corrigé de valeur indéterminée (surement 0).

Un programme supplémentaire à été ajouté : `bytediff`. Ce programme affiche le nombre d'octets différents entre 2 fichiers. Si un des fichiers donnés est plus petit que l'autre, ce n'est pas précisé à l'affichage et ce n'est pas non plus pris en compte dans le calcul : le programme affichera le nombre de différence entre le plus petit fichier et le début du plus grand.

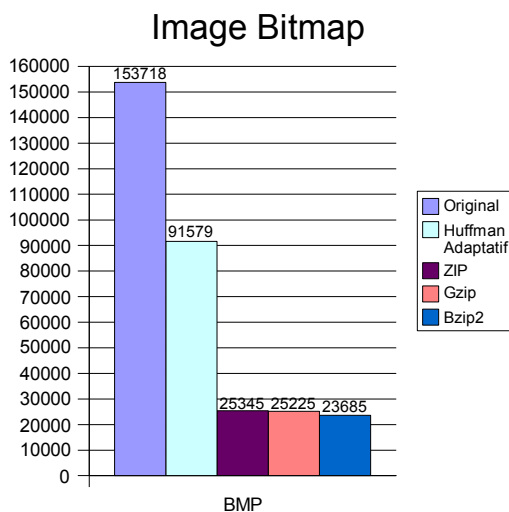
II. Performances

1. Compression

1. Taille compressée

Nous testerons ici les différences de performance en terme de taux de compression entre notre programme (`huffman`) mais aussi différents programmes : ZIP, Gzip & Bzip2.

Toutes les tailles sont données en octets.

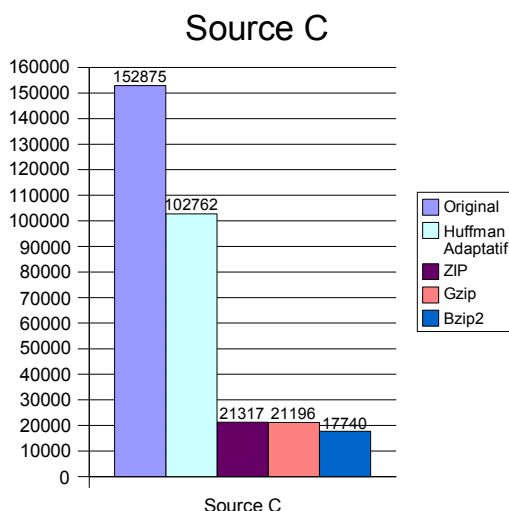


Fichier image bitmap : `moo.bmp`

PC bitmap data, Windows 3.x format, 640 x 480 x 4

Ici, notre algorithme réduit la taille de l'image originale d'un tiers.

Les algorithmes concurrents ont de bien meilleures performances : ZIP et Gzip ont des résultats proches, Bzip2 a un léger avantage sur ces deux derniers.



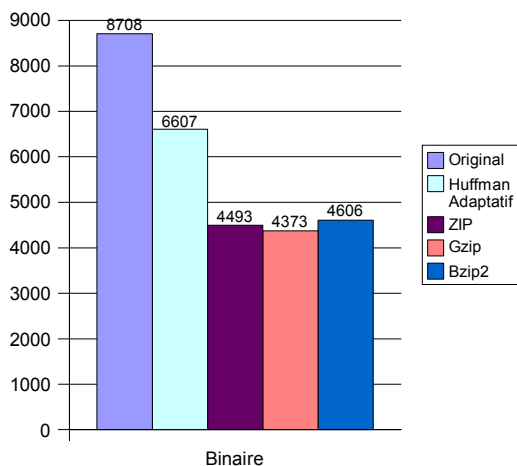
Fichier source C : `drivers/block/DAC960.h`

ASCII C program text

Les résultats sont assez proches par rapport à l'image bitmap. Bzip2 garde son léger avantage sur Gzip et ZIP.

Notre programme compresse moins bien ce source C qu'il ne compresse l'image bitmap précédente alors que pour les programmes concurrents, c'est l'inverse.

Binaire



Fichier binaire : `huffman`

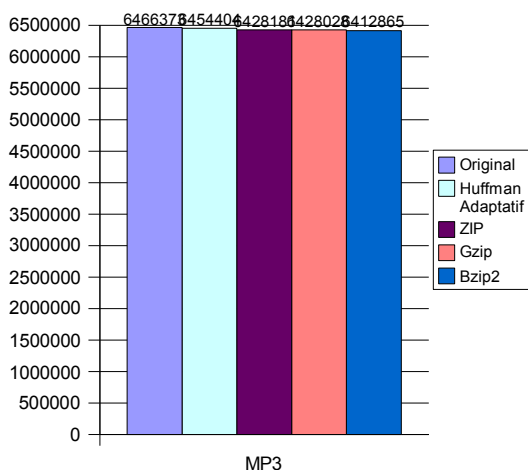
ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), for GNU/Linux 2.6.9, dynamically linked (uses shared libs), stripped

Les résultats sont ici bien différents, le fichier de test étant beaucoup plus petit.

Huffman adaptatif compresse toujours d'environ un tiers mais les concurrents qui compressaient jusqu'à alors à environ 15% ne compressent plus qu'à hauteur d'environ 50%.

On remarque aussi que ce n'est plus Bzip2 qui compresse le mieux mais maintenant Gzip, il est même devenu le moins bon compresseur parmi les 3 concurrents.

MP3



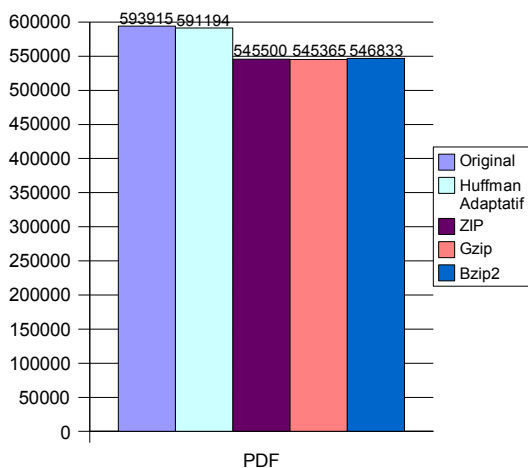
Fichier musical : `09-parallel_universe.mp3`

MPEG ADTS, layer III, v1, 192 kBits, 44.1 kHz, JntStereo

On a affaire ici à la compression d'un fichier déjà compressé, les résultats sont évidemment beaucoup moins bon, quelque soit le compresseur utilisé.

Bzip2 redevient le meilleur compresseur. Gzip et ZIP ont à nouveau des résultats très proches.

PDF



Fichier PDF : `piraterie.pdf`

PDF document, version 1.2

On a ici encore affaire à un fichier dans un format compressé.

Bzip2 perd une deuxième fois son titre de meilleur compresseur. Gzip devient une deuxième fois le meilleur compresseur.

Huffman adaptatif ne compresse que très peu le fichier d'origine.

La différence est cette fois sensible entre les fichiers compressés par les concurrents et les fichiers compressés par Huffman Adaptatif.

Au vu des résultats, il est possible qu'une partie des fichiers PDF ne soit pas compressée, sûrement une en-tête, ce qui expliquerait une meilleure compression sur une seule partie du fichier et donc des résultats proches pour Gzip, ZIP et Bzip2.

2. Temps d'exécution

Machine de test :

Processeur : Intel Pentium-M 1,86Ghz (mode performance)

Mémoire vive : 1024Mo DDR2

Système : Gentoo GNU/Linux (kernel : 2.6.18-gentoo-r4)

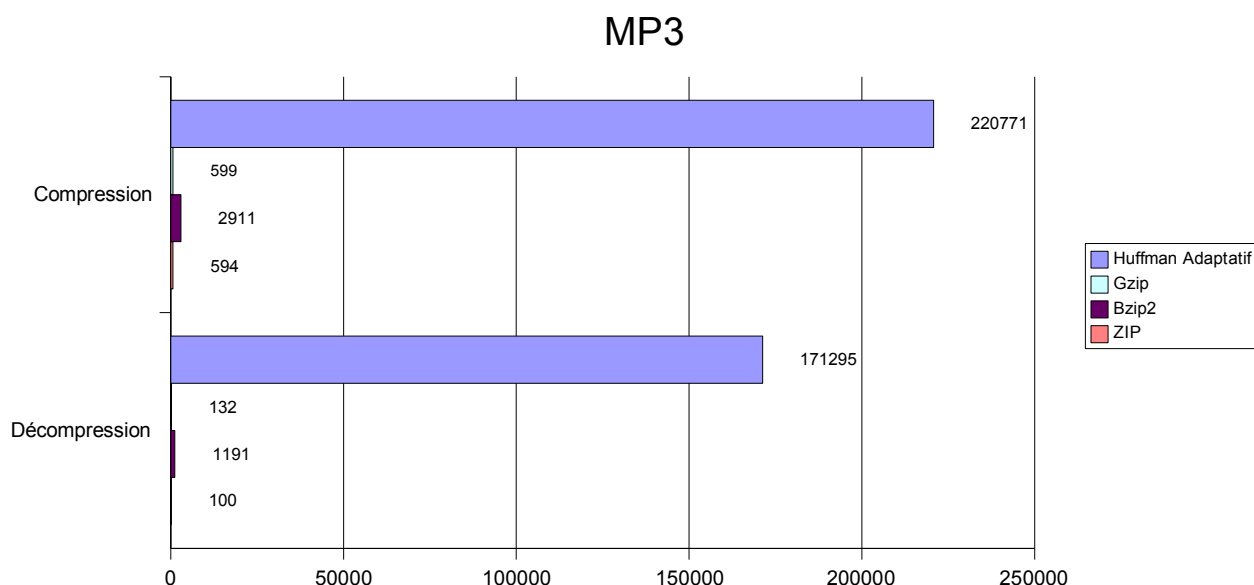
Les programmes de tests ainsi que les différentes bibliothèques utilisées par ces programmes et tout le reste du système ont été compilés avec les options suivantes :

CFLAGS = -march=pentium-m -O3 -pipe -fomit-frame-pointer

Tous les temps sont donnés en millisecondes.

Fichier musical : 09-parallel_universe.mp3

MPEG ADTS, layer III, v1, 192 kBits, 44,1 kHz, JntStereo



Pas besoin de tests supplémentaires, on voit très bien ici que l'algorithme d'Huffman adaptatif est beaucoup plus lent que les algorithmes concurrents. Ceci est probablement dû au fait que dans notre programme, l'arbre d'Huffman est mis à jour à chaque octet lu. Une amélioration possible des performances en temps d'exécution serait de travailler sur des blocs d'octets ou de commencer avec un arbre contenant tous les octets possibles et n'équilibrer cet arbre que tous les X octets lus.

Bzip2 compresses souvent mieux, ceci explique sans doute le temps d'exécution légèrement supérieur à ces deux concurrents.

2. Destruction & Correction d'erreurs

Dans cette section, nous testerons la qualité de la correction avec l'algorithme d'Hamming en faisant passer un fichier généré aléatoirement de 512Ko dans le simulateur de canal. Nous confronterons ses résultats aux corrections effectuées par le programme `par2` sur ces mêmes fichiers.

Pour chaque test, la simulation de canal étant aléatoire, on effectue le test 50 fois, et c'est la moyenne du nombre d'octets différents entre l'original et le fichier passé dans le canal après correction qui est donné.

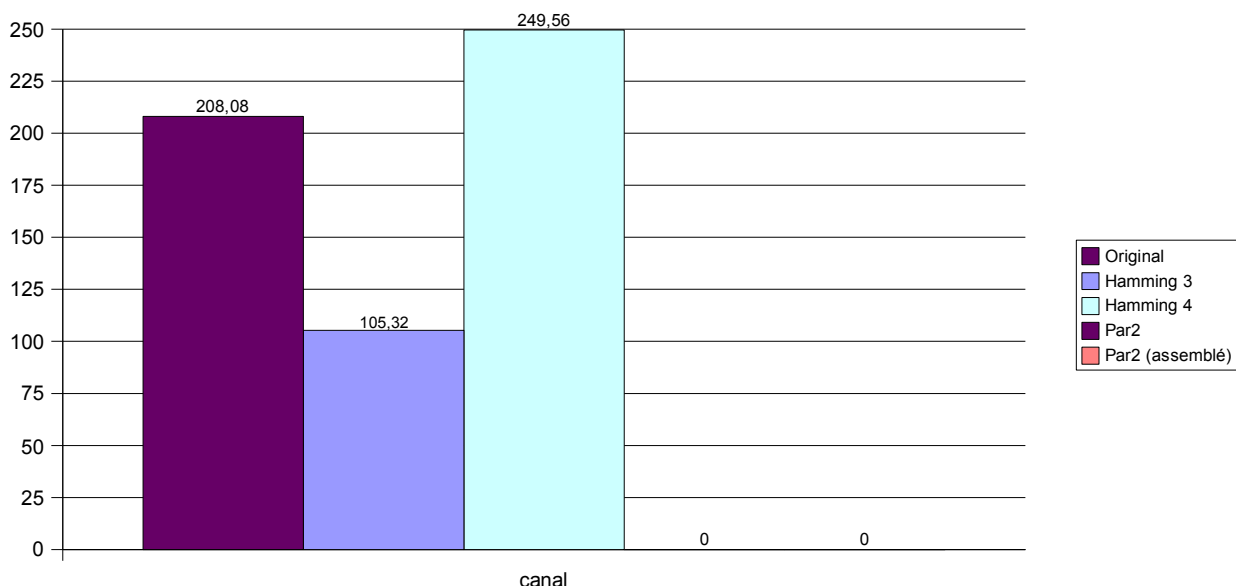
Le programme `par2` générant des fichiers pour la correction d'erreurs, 2 résultats seront donnés, le premier indique la moyenne des octets différents avec l'original après correction à l'aide des fichiers générés par `par2`, le second résultat sera quand à lui fait sur les fichiers corrigés à l'aide des fichiers générés par `par2` mais qui auront aussi été passés dans le canal. Pour cela on assemblera les fichiers de reconstruction avec le fichier d'origine, à l'aide de la commande `tar`.

Note : `par2` est binaire, il ne corrige que s'il peut corriger le fichier entier.

La première valeur donnée (original) donne la moyenne du nombre d'octets différents entre le fichier original et le fichier original passé dans le canal.

Hamming 3 et Hamming 4 signifient hamming lancé avec la valeur $k=3$ et $k=4$ respectivement.

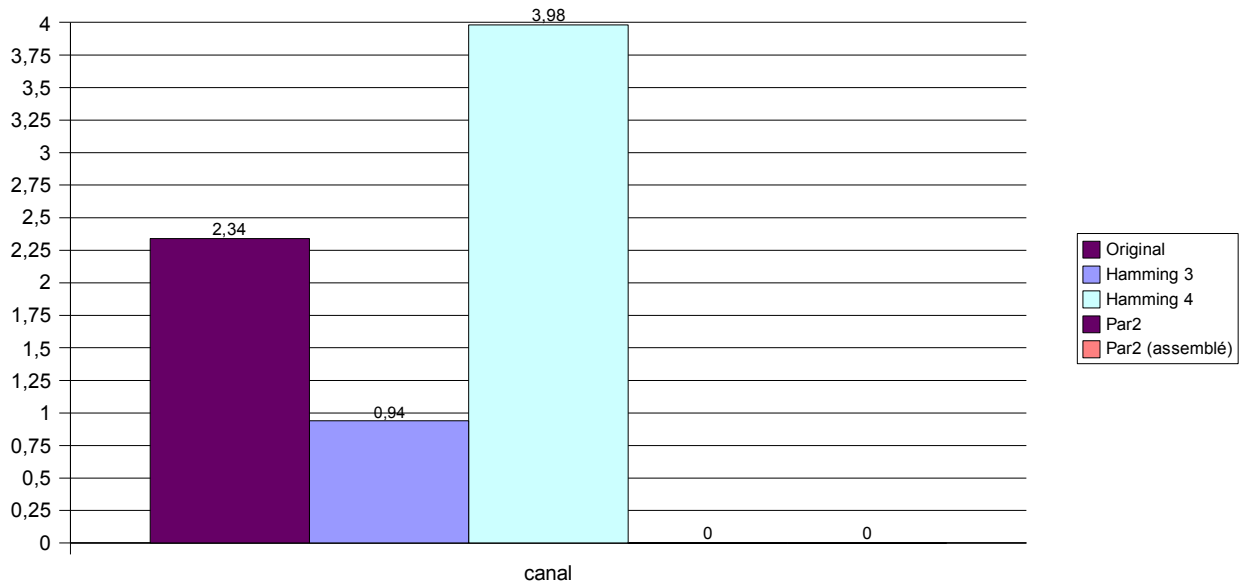
Pour un canal de valeurs $p = 0,0001$ et $q = 0,0001$:



Les résultats pour `par2` sont à 0, non pas parce que les corrections sont parfaites mais parce que `par2` n'a jamais pu corriger.

Hamming 3 apporte suffisamment de bits de corrections pour pouvoir corriger une partie du message mais Hamming 4 est submergé d'erreurs et fini par ajouter des erreurs au lieu de corriger.

Pour un canal de valeurs $p=0,00001$ $q=0,00001$:



Étonnement, on retrouve des résultats proches. Les erreurs étant beaucoup moins nombreuses, hamming 3 corrige toujours aussi souvent, étonnement, hamming 4 crée toujours des erreurs.

Par2 ici corrige toutes les erreurs !

III. Conclusion

Huffman adaptatif est une méthode de compression qui, certes compresse, mais les méthodes plus récentes ont des performances bien supérieures surtout en terme de vitesse de compression.

Hamming est très limité par la possibilité de ne corriger qu'un seul bit par bloc et par le fait qu'il ne peut détecter qu'une seule erreur, il est donc dangereux, puisqu'il peut créer des erreurs. Néanmoins, par rapport à par2 qui ne peut corriger que s'il a suffisamment de données pour corriger, il permet de corriger une partie du message, c'est toujours mieux que rien.