Travail pratique de Compression des Données sur le codage arithmétique

1. Présentation

Codage entropique sans perte, l'algorithme de codage arithmétique s'impose depuis des années comme étant un algo de compression de l'information non pas optimal mais asymptomatiquement optimal, et cela peut se prouve par sa capacité à se rapprocher de plus en plus de l'entropie.

Ici, on part du principe que le code obtenu est unique pour toutes séquences et ce, quelque soit la longueur de la séquence.

2. Avantage

Simple, à comprendre on a mécanisme avec mémoire et, peut importe le choix de l'intervalle qu'on affecte à un caractère, l'ordre se fait en générale par rapport à l'alphabet (voir même arbitraire).

3. Inconvénient et difficulté

On est buté au problème de la précision car on doit à chaque fois qu'on se rapproche des bornes multiplier la borne ce qui lui confère une gestion du calcul de la précision pas facile à prendre en main.

En attribuant aux un petites intervalles les symboles fréquents on perd en précision (décimale) et, les moins probables quant à eux se voient attribuer des intervalles plus grands mais seulement, lorsque nous sommes équiprobables, nous avons du mal à compresser.

4. Constat

Pour le code, on ne se soucie pas des normes (commencer par 1 ou 0) et du cumul des fréquences, ***on préfère plutôt s'intéresser à ce qui se passe sur le code, lorsqu'on utilise des fréquences équiprobables,*** autre que des fréquences normales. On aurait alors, à peu près le même nombre de normalisation pour tous les symboles à partir du moment où, tous les symboles sont équiprobables, car il n'y en aurait pas 1 seul, qui dépenserai beaucoup plus en terme de normalisation. Nous y reviendrons plus bas.

5. Principe de l’algorithme du Codage

On récupère la séquence de texte, et on calcul la probabilité de chaque symbole.
Lorsque on commence à travailler avec un symbole, tout le traitement qui se fera sera borné dans cet intervalle (et donc pas forcément [0,1]) dans le quel, chaque autre symbole possédera une probabilité proportionnelle à l’échelle. On effectue ensuite de façon incrémental le codage tout en gérant le redimensionnement et, la perte de l’information sur le MSB ce dernier qui n’est rien d’autre que le bit le plus significatif qui est le même que celui renvoyé à la suite des redimensionnements. Notons alors que un redimensionnement correspond à un décalage vers la gauche.

6. Principe de l’algorithme du décodage

Dans le décodage on fait le processus inverse, c’est à dire que, l’échelle précédent (t-1) c’est l’échelle écoulée (t-2) aussi, l’une des conditions cruciale à respecter est de veiller à ne jamais perdre la table des intervalles. Nous avons aussi vu que dans ce mécanisme de décompression, le décalage se fait vers la droite, ce qui nous paraît tout à fait logique (d’où son nom décodage incrémental)

7. Résultats obtenus

Chapitres	Taille Code (bit)	Taille Code (octet)	Bit par symbole (Arithmétique)	Entropie	Bit par symbole (Huffman)
1	26942	3367.75	4.3497	4.3474	4.3493
2	100501	12562.625	4.3559	4.3552	4.39
3	80045	10005.625	4.4019	4.4011	4.4321
4	64001	8000.125	4.3905	4.3895	4.4195
5	101677	12709.625	4.3965	4.3958	4.4258
6	43111	5388.875	4.4054	4.4037	4.4337
7	59765	7470.625	4.4110	4.4098	4.4098
8	75623	9452.875	4.3449	4.3440	4.3740
9	57468	7183.5	4.3719	4.3707	4.4007
10	46362	5795.25	4.3614	4.3599	4.4037
11	67518	8439.75	4.4204	4.4194	4.4026
12	73752	9219.0	4.3947	4.3938	4.4238
13	72928	9116.0	4.4094	4.4085	4.4385
14	96904	12113.0	4.4268	4.4261	4.4561
15	61345	7668.125	4.4440	4.4429	4.4729

On peut constater que le codage de Huffman est moins bon que le codage Arithmétique, ce dernier est donc mieux car on se rapproche beaucoup plus de l’entropie et ça,

ça nous arrange. En effet, Huffman qui a besoin de plus en plus de symbole pour atteindre une longueur de code très proche de l'entropie implique que cette croissance exponentielle la rend complexe et de ce fait, on va préférer utiliser l'algorithme de codage Arithmétique.

Nous l'aurons donc compris, si on a peu de symbole de probabilité, dans ce contexte on peut considérer que la condition de borne supérieure du codage de huffman est trop exigeante dans ce cas de figure car, non seulement il sera difficile et coûteux de connaître les probas.

8. Performance du codeur Arithmétique

La principale force de cet algorithme, est principalement axée sur le codage par blocs afin de nous donner en sortie un code unique et propre à la chaîne codée.

On pourrait donc représenter l'intervalle et le code compressé avec une valeur de cette intervalle qui ne sera rien d'autre que la borne inférieur. Par exemple, dans le cas du décodage, on sait que lorsque les bornes d'un intervalle se rapprochent, on va forcément trouver un code qui nous permettra d'identifier le premier caractère et, plus on découvre ce premier plus on connaît l'échelle du premier caractère qu'on multiplie par le code et enfin, au fur et à mesure que l'on multiplie les échelles et que l'on divise par le code, on parvient finalement à déduire le code dont il est question.

9. Analyse les performances du codeur arithmétique avec une loi de probabilité uniforme

Comme cela a été précisé plus haut (dans le constat), si la distribution de probabilités est uniforme alors on se retrouve dans un contexte équiprobable car tous nos intervalles de même longueur auront la même probabilité et pourtant lorsqu'on se place dans un référentiel d'équiprobabilité, on aura du mal à compresser convenablement car on va par exemple compresser avec perte et, on aura à peu près le même nombre de normalisation pour tous les symboles ce qui nous placerai beaucoup plus en position de perdant.

10. Conclusion

L'objectif de ce travail était de mettre en œuvre l'algorithme de Codage Arithmétique, d'établir un parallélisme entre résultats obtenus en nombre de bit par symbole entre Huffman et Arithmétique ainsi que l'entropie et, de conclure sur les performances de ce même codeur, avant de présenter une analyse détaillée sur l'influence de la distribution de probabilité des symboles.