

Leçon 2 : Codage entropique

compression

2018-2019

Pascal Mignot



Introduction

Principe

Codage de
Huffman

Codage
arithmétique

Conclusion

Bibliographie

Le codage de Huffman est un codage à longueur variable basé sur la fréquence d'occurrence des caractères dans la source.

- un caractère fréquent se verra attribuer un code court.
- un caractère rare se verra attribuer un code long.

En conséquence, la compression n'est effective que :

- si les symboles de l'alphabet ont une fréquence sensiblement différentes.
- en raison du comportement moyen : la multiplicité des codes plus courts compense la rareté des codes plus longs.

On reprend l'idée issue de la théorie de l'information.

- on considère un contenant N caractères $\{x_j\}_{j=1\dots N}$ tirés d'un alphabet $S = \{s_1, s_2, \dots, s_n\}$ à n symboles,
- on calcule la fréquence relative p_i de chaque symbole s_i :

$$p_i = \frac{1}{N} \# \{x_j \mid x_j = s_i\}_{j=1\dots N}$$

où $\#E$ est le cardinal de l'ensemble E .

- trouver un code préfixe $C = \{c_1, c_2, \dots, c_n\}$ de longueur $\mathcal{L} = \{\ell_1, \ell_2, \dots, \ell_n\}$ tel que les codes les plus probables (resp. moins probables) se voient affecter un code plus court (resp. long).

Tous les codes se trouvent donc aux feuilles de l'arbre représentant le code préfixe.

Comment construire un tel code ?

Considérons la chaîne "*NEBSTEABLLIB*"

- Liste des symboles : $S = \{B, L, E, I, A, T, S, N\}$.
- Fréquence de symboles : $\mathcal{F} = \{3, 2, 2, 1, 1, 1, 1, 1\}$.
- Le nombre total de symboles est :
$$N = 3 + 2 + 2 + 1 + 1 + 1 + 1 + 1 = 12.$$
- La probabilité des symboles est :
$$\mathcal{P} = \left\{ \frac{3}{12}, \frac{2}{12}, \frac{2}{12}, \frac{1}{12}, \frac{1}{12}, \frac{1}{12}, \frac{1}{12}, \frac{1}{12} \right\}$$

Nous allons voir maintenant comment trouver un code préfixe $C = \{c_1, c_2, \dots, c_8\}$ de longueurs $\mathcal{L} = \{\ell_1, \ell_2, \dots, \ell_8\}$ tel que si $p_i < p_j$, alors $\ell_i \geq \ell_j$.

L'approche est ascendante :

Commencer par trier l'alphabet $\mathcal{S} = \{s_1, \dots, s_n\}$ et les fréquences associées $\mathcal{F} = \{f_1, \dots, f_n\}$ par ordre croissant des fréquences.

- 1 s'il n'y a plus qu'un seul symbole dans \mathcal{S}
alors arrêter
sinon prendre les deux symboles s_i et s_j qui ont les fréquences f_i et f_j les plus petites.
- 2 supprimer s_i et s_j de \mathcal{S} , et f_i et f_j de \mathcal{F} .
- 3 insérer un nouveau symbole combiné (s_i, s_j) dans \mathcal{S} et sa fréquence combinée $f_i + f_j$ dans \mathcal{F} , à la position dans la liste des fréquences correspondant à plus haute position possible afin que les deux listes demeurent triées.

Le symbole obtenu représente alors l'arbre correspondant au codage préfixe (descente à gauche = 0, descente à droite = 1).

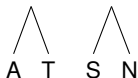
Exemple :

1) B L E I A T S N
3 2 2 1 1 1 1 1

2) B SN L E I A T
3 2 2 2 1 1 1



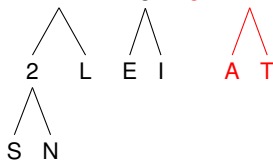
3) B AT SN L E I
3 2 2 2 2 1



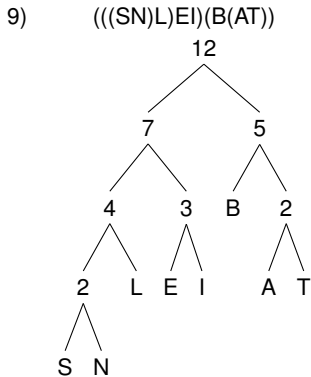
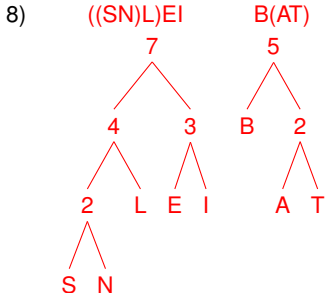
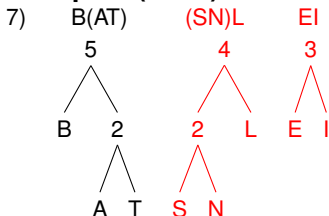
4) EI B AT SN L
3 3 2 2 2



5) (SN)L EI B AT
4 3 3 2



Exemple : (suite)



Exemple : (suite)

Le codage préfixe associé à l'alphabet est par conséquent :

S	0000	I	011
N	0001	B	10
L	001	A	110
E	010	T	111

Liste des symboles :

$S = \{B, L, E, I, A, T, S, N\}$.

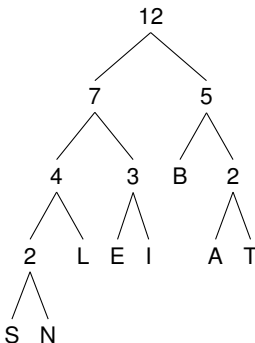
Fréquence de symboles :

$\mathcal{F} = \{3, 2, 2, 1, 1, 1, 1, 1\}$.

Code :

C
 $\{10, 001, 010, 011, 110, 111, 0000, 0001\}$.

$((((SN)L)EI)(B(AT)))$



Le code est bien préfixe et la longueur de chaque code dépend donc bien de sa fréquence.

Algorithmes de compression :

Soit T une suite de caractères t_i à coder.

Initialiser la sortie $C = \{\}$

Parcourir le texte $t = t_1 \dots t_n$ à coder caractère par caractère :

- 1 soit s_j le symbole lu dans le caractère t_i ,
- 2 cumuler le code c_j associé à s_j sur la sortie
($C = C \cup \{c_j\}$).

Le texte compressé est C .

Exemple

Code : $C = \{10, 001, 010, 011, 110, 111, 0000, 0001\}$.

Liste des symboles : $S = \{B, L, E, I, A, T, S, N\}$.

Le texte *NEBSTEABLLIB* est codé comme :

$$\begin{aligned} &0001.010.10.0000.111.010.110.10.001.001.011.10 \\ &= 00010101000001110101101000100101110 \end{aligned}$$

Algorithme de décompression :

Soit B la suite des bits à décoder.

Initialiser la sortie $T = \{\}$

Soit A l'arbre contenant le codage prefixe du code.

Soit a un pointeur positionné à la racine de l'arbre.

Parcourir les bits $b = b_1 \dots b_n$ un à un :

- ❶ déplacer a dans l'arbre en fonction de b_i ($b_i = 0$ aller à gauche, $b_i = 1$ aller à droite).
- ❷ si a est sur une feuille,
 - renvoyer le symbole s_j associé à la feuille,
 - cumuler s_j à T : $T = T \cup \{s_j\}$
 - repositionner a à la racine

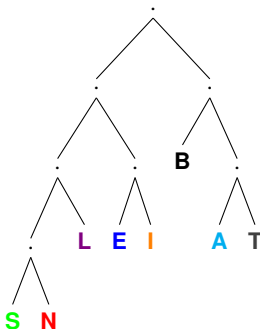
Le texte décompressé est T .

Exemple de décompression :

$$B = b_1 b_2 \dots b_n$$

$$= 0001010100001110101101000100101110$$

$T = NEBSTEABLLIB$



EXERCICE 1: Codage de Huffman

Une source produit les symboles $\{A, D, E, I, L, N, O, R, S, T, U\}$ avec les fréquences suivantes respectives suivantes $\{14\%, 2\%, 14\%, 13\%, 4\%, 9\%, 4\%, 8\%, 14\%, 12\%, 6\%\}$.

- 1 Combien de bits sont nécessaires pour un codage à taille fixe pour cette source ?
- 2 Quelle est l'entropie de cette source ?
- 3 Calculer le codage de Huffman pour cette source.
- 4 Quel est le nombre moyen de bits par symbole pour ce code ?
- 5 Compresser le mot "ASTEROIDE" en utilisant ce code.
- 6 Comparer les longueurs des codes obtenus, et conclure.

Remarques :

Lors de la construction de l'arbre, nous avons utilisé les deux règles suivantes :

- un symbole combiné est placé à la plus haute position dans la liste des symboles permettant de garder la liste des fréquences triée.
- lorsque deux symboles s_i et s_j sont combinés, celui qui a la fréquence la plus haute est placé à droite (i.e. le symbole combiné est (s_i, s_j) si $f_i \leq f_j$ et (s_j, s_i) sinon.

Le code préfixe engendré de cette façon est appelé un code canonique à variance minimale.

On notera aussi qu'il n'y a en général pas un code unique associé à un alphabet (en particulier dès que certaines fréquences sont égales).

Exercice :

refaire le codage avec $S = \{B, L, E, I, A, \mathbf{S}, \mathbf{T}, N\}$ où $\mathcal{F} = \{3, 2, 2, 1, 1, 1, 1, 1\}$ (i.e. avec S et T permutés dans la liste des symboles).

Pourquoi à variance minimale ?

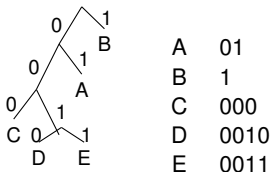
Exemple

Si on prend l'alphabet suivant :

Liste des symboles : $S = \{A, B, C, D, E\}$.

Fréquence de symboles : $\mathcal{F} = \{0.2, 0.4, 0.2, 0.1, 0.1\}$.

Codage C_1 sans priorité



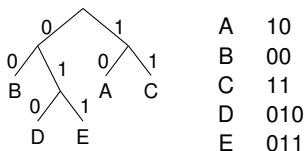
Longueur moyenne : $\bar{\ell} = \sum_i \ell_i \cdot p_i$

$\bar{\ell} = 2.2$ bits/symb

Variance : $\sigma^2 = \sum_i (\ell_i - \bar{\ell})^2 \cdot p_i$

$\sigma^2 = 1,36 \Rightarrow \sigma = 1,17$ bits

Codage C_2 à variance minimale



Longueur moyenne : $\bar{\ell} = \sum_i \ell_i \cdot p_i$

$\bar{\ell} = 2.2$ bits/symb

Variance : $\sigma^2 = \sum_i (\ell_i - \bar{\ell})^2 \cdot p_i$

$\sigma^2 = 0,16 \Rightarrow \sigma = 0,4$ bits

Interprétation de l'efficacité de ces codes :

- la longueur moyenne est la même.

Par la loi des grands nombres, nous déduisons que le codage réalisé par C_1 ou C_2 peut être aussi petit que l'on souhaite, pour peu que le mot codé soit assez long.

- la variance est différente.

Si la variance est faible (resp. forte), que le codage d'un mot m de n symboles produit des chaînes dont la longueur varie peu (resp. beaucoup varier).

Dans l'exemple précédent : $n \leq |C_1(m)| \leq 4n$ et $2n \leq |C_2(m)| \leq 3n$.

Donc, un code à variance minimale aura tendance à compresser mieux en moyenne, y compris lors de l'occurrence ponctuelle de suites d'événements relativement rares.

Rappel : la loi des grands nombres nous dit que la moyenne empirique d'une v.a. X converge en probabilité vers l'espérance de la loi de X (= la moyenne empirique converge vers la vraie moyenne).

Un codage est optimal si :

- 1 Pour tout couple de symboles $s_i \neq s_j$, si $\Pr[s_i] \geq \Pr[s_j]$ alors $|C(s_i)| < |C(s_j)|$.
- 2 Les deux symboles les moins probables ont la même longueur.
Si le code est optimal, comme le codage d'aucun symbole ne peut être plus long que le code de ces deux symboles, alors si ces deux codes n'avaient pas la même longueur, on obtiendrait un code plus long. En conséquence, le code ne serait plus optimal. Donc, cette condition doit être vérifiée.
- 3 Tout noeud possède deux fils.
- 4 Si un codage est optimal et que l'on fusionne dans un noeud intermédiaire l'ensemble des feuilles qui en descendent, alors l'arbre résultant est optimal pour l'alphabet réduit.

Le codage de Huffman C vérifie l'ensemble de ces conditions (la condition 4 correspond à la méthode de construction). Il est donc optimal.

Si le codage de Huffman est optimal, il vérifie la condition (voir premier cours) :

$$H(S) \leq \bar{\ell} \leq H(S) + 1$$

A savoir, le codage moyen des symboles est au plus à 1 bit du codage optimal.

Cas général sur l'optimum

On peut montrer [Gal78] que si p_{\max} est la probabilité maximale du symbole le plus fréquent, alors la borne supérieure pour un codage de Huffman est :

- si $p_{\max} \geq 0.5$, $\bar{\ell} \leq H(S) + p_{\max}$
- si $p_{\max} < 0.5$, $\bar{\ell} \leq H(S) + p_{\max} + 0.086$

En conséquence, si l'alphabet contient peu de symboles, ou si les probabilités entre symboles sont particulièrement biaisées.

- la valeur de p_{\max} peut être importante,
- le codage de Huffman sera inefficace par rapport à l'optimum possible donné par l'entropie.

Si $\bar{\ell} = H(S) + \delta$, alors le codage de n symboles produit $\delta.n$ bits de plus que le minimum nécessaire.

Conditions pour atteindre l'entropie avec un code de Huffman

On a vu qu'un codage optimal doit vérifier que $\ell_i^* = -\log_2 p_i$.

Le codage de Huffman utilise un nombre entier de bits par symbole, la borne inférieure de cet optimum n'est atteignable que si ℓ_i^* est un entier.

Donc, que si p_i peut s'écrire comme une puissance entière négative de 2 (i.e. de la forme 2^{-n}).

Ceci doit avoir lieu en même temps pour tous les p_i , on en déduit que la borne inférieure n'est atteinte que si :

- l'ensemble des probabilités p_i s'écrivent sous la forme 2^{-n_i} , (ce qui conduit à $\ell_i^* = -\log_2 p_i = -\log_2 2^{-n_i} = n_i$).
- la borne supérieure est atteinte dans l'inégalité de Kraft, à savoir : $\sum_i 2^{-\ell_i^*} = 1$ (ce qui revient à dire que la somme des probabilités fait 1).

En conséquence, l'entropie n'est que rarement atteinte par un code de Huffman.

Exemple

Les fréquences $\mathcal{F} = \{0.8, 0.02, 0.18\}$ conduisent au codage de Huffman $C = \{1, 00, 01\}$.

- Longueur moyenne : $\bar{\ell} = 1.2$ bits/symbole.
- Entropie : $H(S) = 0.816$ bits

On dépasse l'entropie de 0.384 bits/symboles.

Le codage de Huffman produit donc 47% de bits en plus que l'optimum théorique.

Cette constatation conduit aux codes de Huffman étendus.

Si certains symboles sont trop fréquents, plutôt que de considérer 1 symbole codé = 1 code, que se passe-t-il si on considérait n symboles = 1 code ?

Exemple

Si on a les symboles $S = \{a_1, a_2, a_3\}$ et que l'on prend des paquets de 4 symboles, on obtient l'alphabet suivant :

$$S^4 = \{a_1 a_1 a_1 a_1, a_1 a_1 a_1 a_2, a_1 a_1 a_1 a_3, a_1 a_1 a_2 a_1, \dots, a_3 a_3 a_3 a_3\}.$$

Evidemment,

- si S contient p symboles, S^n contient p^n symboles (= toutes les combinaisons possibles de p symboles).
- si on suppose que les symboles sont indépendants, la probabilité des symboles de S^n est calculée par leurs produits. A savoir pour tout n -uplet de symboles $(a_{i_1} a_{i_2} \dots a_{i_n})$, on a :

$$\Pr[a_{i_1} a_{i_2} \dots a_{i_n}] = \Pr[a_{i_1}] \cdot \Pr[a_{i_2}] \dots \Pr[a_{i_n}]$$

- s'ils ne le sont pas et que l'on peut évaluer ces probabilités, le codage sera encore plus efficace !

Si on note R^n le nombre de bits nécessaire pour coder n symboles, alors on sait que :

$$H(S^n) \leq R^n \leq H(S^n) + 1$$

En conséquence, le nombre de bits nécessaire pour coder 1 symboles par paquets de n est :

$$\frac{1}{n}H(S^n) \leq R \leq \frac{1}{n}H(S^n) + \frac{1}{n}$$

Or, on peut montrer que $H(S^n) = nH(S)$ (pour des symboles indépendants, voir [Say06] p.53).

Donc,

$$H(S) \leq R \leq H(S) + \frac{1}{n}$$

En conséquence, utiliser un codage de Huffman sur des blocs garantit un taux plus proche de celui de l'entropie.

Néanmoins, si l'on augmente la taille de l'alphabet exponentiellement, le codage de Huffman devient alors difficilement utilisable en pratique. La solution à ce problème est fournie par le codage arithmétique.

EXERCICE 2: Codage de Huffman étendu

On considère une source X utilisant un alphabet constitué de deux symboles $S = A, B$ avec les fréquences respectives suivantes $\mathcal{F} = \{0.25, 0.75\}$

- 1 Effectuer un codage de Huffman (classique) pour ce code.
- 2 Calculer l'entropie $H(X)$ et la comparer au nombre moyen de bits par symbole pour ce code.
- 3 On crée maintenant un alphabet étendu $S_2 = \{AA, AB, BA, BB\}$. Calculer les probabilités associées à chaque symbole.
- 4 Exprimer l'entropie de la source pour l'alphabet étendu $H_2(X)$ en fonction de $H(X)$. On en déduit la valeur de $H_2(X)$.
- 5 En déduire le codage de Huffman pour l'alphabet étendu.
- 6 Calculer le nombre moyen de bits par symbole du code étendu (penser à renormaliser pour tenir compte du fait que chaque symbole de l'alphabet étendu contient deux symboles). On le comparera à l'entropie par symbole pour cette source.

Le codage de Huffman nécessite la connaissance de la probabilité de chaque symbole de la source.

Si cette information n'est pas connue, le codage se fait en deux passes : calcul des fréquences empiriques des symboles, création du code, puis utilisation de code.

Le codage de Huffman adaptatif est une variation de codage de Huffman qui consiste à :

- à la lecture du $k^{\text{ième}}$ caractère, construire dynamiquement les fréquences d'apparition et l'alphabet à partir de l'ensemble des symboles déjà rencontrés.

A savoir, l'alphabet S_k ne contient que les caractères déjà rencontrés, et le calcul des fréquences d'apparition \mathcal{F}_k de S_k ne portent que sur les k premiers caractères.

- mettre à jour dynamiquement le codage C_k à partir de \mathcal{F}_k .
- pour coder le $k^{\text{ième}}$ caractère avec son code dans C_k .

On note $\uparrow\uparrow$ le symbole destiné à rester le moins fréquent.

Algorithme de compression :

- 1 **Initialisation** : la liste des symboles à $\mathcal{S}_0 = (\uparrow\uparrow)$, et la liste des fréquences à $\mathcal{F}_0 = (0)$.
- 2 **Compression des symboles** : lorsque l'on lit le $k^{\text{ième}}$ symbole v_k
 - a si $v_k \in \mathcal{S}_{k-1}$
alors $(\mathcal{S}_k, \mathcal{F}_k) = (\mathcal{S}_{k-1}, \mathcal{F}_{k-1})$ + mettre-à-jour de la fréquence de v_k .
sinon $(\mathcal{S}_k, \mathcal{F}_k) = (\mathcal{S}_{k-1}, \mathcal{F}_{k-1})$ + insérer de $(v_k, 1)$.
 - b retrier conjointement les listes $(\mathcal{S}_k, \mathcal{F}_k)$ par ordre de fréquence.
 - c régénérer le nouveau code de Huffman C_k associé à ces listes.
 - d coder v_k avec son code correspondant dans C_k .

Problème

Comment décompresser le code compressé si l'on ne connaît pas l'ordre des symboles ?

Solution

Lorsque l'on insère un nouveau symbole dans l'arbre, il faut le transmettre avec le code compressé.

A noter que lorsqu'un code est déjà présent, le décompresseur reconnaît le code reçu.

Dans tous les cas, il peut donc mettre à jour l'arbre de façon symétrique au compresseur.

Ceci oblige à :

- 1 Disposer d'une table \mathcal{T} de codes de longueur fixe pour chaque symbole
Typiquement s'il y a p symboles, les coder sur n bits, où n est le plus petit entier tel que $k \geq 2^n$.
 \mathcal{T} n'est pas compressée.
- 2 Lorsqu'un symbole est inséré dans l'arbre, transmettre le code de $\uparrow\uparrow$ (qui indique qu'un nouveau symbole va être inséré) + le code de ce symbole.

Conséquence

Les symboles qui peuvent être transmis sont connus à l'avance par le compresseur et le décompresseur qui disposent chacun de la table \mathcal{T} (non transmise).

Exemple

On veut transmettre le mot **ABBBAC**.




L'alphabet est $S = \{A, B, C\}$.

Les codes de transmission sont $\mathcal{T} = \{00, 01, 10\}$.

Dans la table ci-dessous, le code transmis est issu du code de Huffman de la ligne précédente.

Lecture	S_k	\mathcal{F}_k	Huffman	Sortie
	(\uparrow)	(0)		
ABBBAC	(A, \uparrow)	$(1, 0)$		0 00
ABBBAC	(A, B, \uparrow)	$(1, 1, 0)$		0 01
ABBBAC	(B, A, \uparrow)	$(2, 1, 0)$		11

Exemple (suite)

Lecture	S_k	\mathcal{F}_k	Huffman	Sortie
ABBBAC	(B, A, \uparrow)	$(3, 1, 0)$		1
ABBBAC	(B, A, \uparrow)	$(3, 2, 0)$		01
ABBBAC	(B, A, C, \uparrow)	$(3, 2, 1, 0)$		00 10

La chaîne compressée est donc :

000 001 11 1 01 0010

A noter que dans la transmission ci-dessus :

- la première transmission d'un symbole implique une sortie plus longue,
- sinon, on transmet le code de Huffman (optimal) pour l'ensemble des symboles déjà transmis.

Le nombre de symboles transmis doit donc être suffisamment important afin que la transmission liées aux insertions de symbole soit négligeable.

Le processus de décompression est symétrique puisque :

- l'on dispose de l'alphabet $\mathcal{S} = \{A, B, C\}$ et la table $\mathcal{T} = \{00, 01, 10\}$ des codes de transmission de symbole.
- chaque symbole reçu permet de construire le codage de Huffman identique à celui qui a servi à la compression.

Algorithme de décompression :

- 1 **Initialisation** : liste des symboles à $\mathcal{S}_0 = (\uparrow)$ et $\mathcal{F}_0 = (0)$.
- 2 **Décompression** :
 - a Construire le code de Huffman associé aux listes $(\mathcal{S}_k, \mathcal{F}_k)$.
 - b Lire le code binaire b suivant en utilisant le code de Huffman, et en déduire le symbole s associé.
 - c si $s = \uparrow$
alors | lire le code de transmission du caractère c inséré.
| $(\mathcal{S}_{k+1}, \mathcal{F}_{k+1}) = \text{insérer}(c, 1)$ dans $(\mathcal{S}_k, \mathcal{F}_k)$.
sinon | $(\mathcal{S}_{k+1}, \mathcal{F}_{k+1}) = \text{mettre à jour la fréquence de } c$ dans $(\mathcal{S}_k, \mathcal{F}_k)$
 - d Recommencer au 2 jusqu'à la fin de la chaîne d'entrée.

Exemple



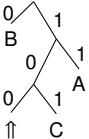
L'alphabet est $S = \{A, B, C\}$.

Les codes de transmission sont $\mathcal{T} = \{00, 01, 10\}$.

On reçoit le code 000001111010010.

Lecture	S_k	\mathcal{F}_k	Huffman	Sortie
	(\uparrow)	(0)	0 \uparrow	
<u>0</u> 00001111010010	(A, \uparrow)	(1, 0)	0 \diagup 1 \uparrow A	A
0000 <u>0</u> 1111010010	(A, B, \uparrow)	(1, 1, 0)	0 \diagup 1 A 0 \diagdown 1 \uparrow B	B
000001 <u>1</u> 11010010	(B, A, \uparrow)	(2, 1, 0)	0 \diagup 1 0 \diagdown 1 B \uparrow A	B

Exemple (suite) :

Lecture	S_k	\mathcal{F}_k	Huffman	Sortie
000001111 <u>0</u> 10010	(B, A, \uparrow)	$(3, 1, 0)$		B
000001111 <u>0</u> <u>1</u> 0010	(B, A, \uparrow)	$(3, 2, 0)$		A
00000111101 <u>0</u> <u>0</u> 1 0	(B, A, C, \uparrow)	$(3, 2, 1, 0)$		C

Le codage de Huffman adaptatif :

- a les mêmes inconvénients que le codage de Huffman classique (il n'atteint pas l'optimal réel sauf dans des cas particuliers), bien qu'il puisse lui-aussi être étendu.
- bien qu'il existe des méthodes rapides permettant de mettre à jour les fréquences, la reconstruction du code de Huffman à chaque symbole peut être quand à elle relativement lente, particulièrement dans le cas où l'alphabet est de grande taille.

Il n'en reste pas moins que le codage de Huffman est l'un des meilleurs algorithmes de compression à longueur variable connu, et reste très utile.

Introduction

Codage de
Huffman

Codage Statique
Codage étendu
Codage adaptatif
Codage de Golomb

Codage
arithmétique

Conclusion

Bibliographie

EXERCICE 3: Codage de Huffman adaptatif

On considère l'alphabet $S = \{A, B\}$, avec les codes de transmissions $\mathcal{T} = \{0, 1\}$ comme dans l'exemple du cours.

- 1 Donner le code de Huffman adaptatif associé à la chaîne *AAAABABB*.
- 2 Décoder le code de Huffman adaptatif suivant :
0011010110101111.

Les codes de Golomb sont une classe de code permettant de coder tous les entiers de \mathbb{N} avec l'hypothèse que plus l'entier est grand, plus faible est sa probabilité d'occurrence.

Ce code permet de travailler avec un alphabet non fini.

L'exemple le plus simple de code de Golomb est le suivant :

- $S = \{0, 1, 2, \dots, n, \dots\}$
- $C = \{0, 10, 110, \dots, \underbrace{11 \dots 1}_n 0, \dots\}$
n chiffres 1

Ce code peut être vu comme un code unaire (une unité = un 1) où 0 est le séparateur.

Ce code est optimal pour la répartition suivante de fréquences :

$$\mathcal{F} = \{2^{-1}, 2^{-2}, 2^{-3}, \dots, 2^{-n}, \dots\}$$

Rappel

$$S_n = \sum_{i=0}^n 2^{-i} = 1 - 2^{-n} \text{ et } \lim_{n \rightarrow \infty} S_n = 1.$$

Nous sommes donc bien dans les conditions de réalisation de l'inégalité de Kraft.

Un code de Golomb d'ordre m est construit de la manière suivante.

Pour coder un entier e :

- calcul $q = e/m$ (division entière) et $r = e \bmod m$.
- coder q : $C_u(q) = \underbrace{1 \dots 1}_{q \text{ chiffres } 1} 0 = q \text{ chiffres } 1 \text{ suivi d'un } 0$.
- coder r : $C_b(r) = r$ codé sur b bits où b est le plus petit entier tel que $m \leq 2^b$ (i.e. $b = \lceil \ln_2 m \rceil$).
- le code de Golomb est $C_u(q)C_b(r)$.

Remarque : Comme r est le reste de la division entière par m , $r < m \leq 2^b$.
Donc, b bits suffisent pour un codage uniforme des nombres entre 0 et $2^b - 1$.

Exemple

Quel est le code d'ordre 6 associé à 52 ?

- $q = 52/6 = 8, r = 52 \bmod 6 = 4$
- $C_u(q) = 111111110$
- $b = \lceil \ln_2 6 \rceil = 3$. On a bien $2^3 = 8 \geq 6$.
- $C_u(q) = 111111110$ et $C_b(r) = 100$.

Donc, son code est 111111110100.

Une source $s = b_1 b_2 \dots b_n$ émet une suite de b bits b_i formant des codes de Golomb d'ordre m .

La partie statique du code a pour longueur de $b = \lceil \ln_2 m \rceil$ bits.

On démarre de $i = 1$:

- ❶ soit $k =$ le plus petit indice $\geq i$ tel que $b_k = 0$.
- ❷ $c_r = b_i \dots b_{k-1}$ (on notera que c_r peut être vide).
Le $r =$ nombre de 1 dans c_r est donc $k - i$.
- ❸ $c_q = b_{k+1} \dots b_{k+r}$.
 $q =$ nombre représenté par le code binaire c_q .
- ❹ le nombre n décodé est $n = q.m + r$.
- ❺ $i = i + (k + r + 1)$
- ❻ tant que l'on a pas atteint la fin de s reprendre en 1.

Exemple

Une source s émet le code de Golomb d'ordre 4 suivant 111000101111111010. Quels sont les entiers émis ?

Pour la partie statique $b = \lceil \ln_2 4 \rceil = 2 \Rightarrow$ partie statique sur 2 bits.

On lit la chaîne dans l'ordre :

- **lecture 1** : 111000 $\Rightarrow q = 3, r = 0, n = 3.4 + 0 = 12$
- **lecture 2** : 1011 $\Rightarrow q = 1, r = 3, n = 1.4 + 3 = 7$
- **lecture 3** : 11111010 $\Rightarrow q = 5, r = 2, n = 5.4 + 2 = 22$

Donc, la source a émis 3 entiers dans l'ordre 12, 7 et 22.

EXERCICE 4: Codage de Golomb

- 1 Donner le code de Golomb d'ordre 8 des entiers suivants : 5, 8, 16, 23.
- 2 Décoder les entiers suivants issus d'un code de Golomb d'ordre 3 : 11010, 010, 1010.
- 3 Est-ce-que toute suite de nombre binaire représente nécessairement un code de Golomb ?

Le code de Golomb est optimal lorsque les nombres suivent une loi géométrique.

Qu'est-ce qu'une loi géométrique ?

Soit p la probabilité de réalisation d'un évènement X (par exemple, la probabilité de faire un six avec un dé).

Alors la loi définie par $\Pr[X = n] = p(1 - p)^{n-1}$ est la probabilité pour que l'évènement X se réalise à la $n^{\text{ème}}$ tentative.

Cette loi s'appelle une loi géométrique (car la fonction de répartition $\Pr[X \leq n] = \sum_{k \leq n} \Pr[X = k]$ est une suite géométrique de raison $1 - p$).

$\Pr[X \leq n] = 1 - (1 - p)^n$ est la probabilité que l'évènement X se réalise si on fait n essais (par exemple, la probabilité de tirer un 6 si l'on lance un dé 10 fois est de $1 - (1 - 1/6)^{10} \simeq 84\%$).

Cette loi convient donc particulièrement bien pour représenter la conjonction de suite d'évènements.

Exemple

Supposons que nous devons compresser un mot contenant des 0 et des 1 tel que $\Pr[X = 1] = p$ et $\Pr[X = 0] = 1 - p$ dans le cas où les suites sont fréquentes.

La probabilité d'avoir une suite de n zéros consécutifs est $(1-p)^n p$ (le $n+1^{\text{ème}}$ n'étant pas un zéro). Cette suite suit donc une loi géométrique.

En conséquence, pour compresser ces suites, l'utilisation d'un code de Golomb est adéquate.

Quel code de Golomb choisir : on peut démontrer que le code de Golomb est optimal pour compresser une loi X si :

- X suit une loi géométrique de paramètre p (où p est la probabilité de réalisation de l'évènement).
- on choisit $m = \lceil -\frac{1}{\ln_2 p} \rceil$.

Pour plus de précisions, voir [SMB10].

Remarque : un code de Golomb pour lequel m est une puissance de 2 assure l'optimalité du code.

L'idée du codage arithmétique est de transformer une suite de symboles en une valeur de l'intervalle de $[0; 1]$, en le découpant récursivement suivant les probabilités associées aux différents symboles.

Soit l'alphabet $\mathcal{S} = \{s_1, s_2, \dots, s_n\}$ dont les probabilités respectives sont $\mathcal{F} = \{p_1, p_2, \dots, p_n\}$ (rappel : $\sum_{i \leq n} p_i = 1$).

On appelle fonction de distribution la fonction définie par : $F_k = \sum_{i \leq k} p_i$.

On associe au symbole s_i l'intervalle $E_i = [F_{i-1}; F_i)$.

Exemple

$$\mathcal{F} = \{0.2, 0.1, 0.05, 0.3, 0.2, 0.15\}$$

Fonction de distribution :

k	0	1	2	3	4	5	6
F_k	0.00	0.20	0.30	0.35	0.65	0.85	1.00

Intervalles associés à chaque symbole :

s_1	s_2	s_3	s_4	s_5	s_6
$[0; 0.2)$	$[0.2; 0.3)$	$[0.3; 0.35)$	$[0.35; 0.65)$	$[0.65; 0.85)$	$[0.85; 1.0)$

2018-2019

Introduction

Codage de
Huffman

Codage
arithmétique

Principe

Unité

Efficacité

Codage

Décodage

Implémentation
entière

Codage adaptatif

Conclusion

Bibliographie

On note les bornes de l'intervalle semi-ouvert $E = [E^{\min}; E^{\max})$.
Pour associer un intervalle à la suite de trois symboles $s_i s_j s_k$, on :

- associe un intervalle $E_i \in [0; 1)$ au symbole s_i
 $E_i = 0 + |[0; 1)| \times [F_{i-1}; F_i)$
- associe un intervalle $E_{ij} \in E_i$ au symbole $s_i s_j$
 $E_{ij} = E_i^{\min} + |E_i| \times [F_{j-1}; F_j)$
- associe un intervalle $E_{ijk} \in E_i$ au symbole $s_i s_j s_k$
 $E_{ijk} = E_{ij}^{\min} + |E_{ij}| \times [F_{k-1}; F_k)$

L'opération $F = a + b \times E$ signifie que l'intervalle $F = [f_0, f_1)$ est construit à partir de l'intervalle $E = [e_0, e_1)$ comme $[f_0, f_1) = [a + b.e_0; a + b.e_1)$.

Interprétation : F = Intervalle E mis à l'échelle b et translaté de a .

Construction itérative pour une suite quelconque $c_1 c_2 \dots c_n$ de symboles :

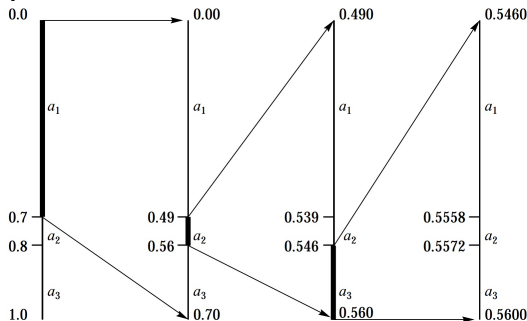
$$E_0 = [0; 1).$$

$$E_n = E_{n-1}^{\min} + |E_{n-1}| \times [F_{c_{[n]}-1}; F_{c_{[n]}})$$

où $c_{[n]}$ est l'indice du symbole c_n dans \mathcal{S} .

2018-2019

Exemple



Calcul de l'intervalle associé à $a_1 a_2 a_3 a_2$:

- $a_1 \in 0 + |[0; 1)| \times [0; 0.7) = [0; 0.7)$
- $a_1 a_2 \in 0 + |[0; 0.7)| \times [0.7; 0.8) = [0.49; 0.56)$
- $a_1 a_2 a_3 \in 0.49 + |[0.49; 0.56)| \times [0.8; 1) = [0.546; 0.56)$
- $a_1 a_2 a_3 a_2 \in 0.546 + |[0.546; 0.56)| \times [0.7; 0.8) = [0.5558; 0.5572)$

Rappel : $E_n = E_{n-1}^{\min} + |E_{n-1}| \times [F_{c_{[n]}-1}; F_{c_{[n]}})$

2018-2019

Introduction

Codage de
Huffman

Codage
arithmétique

Principe

Unicité

Efficacité

Codage

Décodage

Implémentation
entière

Codage adaptatif

Conclusion

Bibliographie

En fait, nous ne sommes pas intéressé spécifiquement par l'intervalle complet associé à une suite de symboles, mais seulement à l'un de ses représentants.

A savoir, pour une suite de n symboles, comme :

- les $|S|^n$ intervalles correspondant à l'ensemble des combinaisons à n symboles sont disjoints,
- ces intervalles recouvrent complètement l'intervalle $[0; 1)$,

alors tout réel r dans $[0; 1)$ est obligatoirement dans un seul de ces intervalles.

En conséquence, un seul réel r représente **la totalité de la chaîne de symboles**.

Le **codage** d'une suite de n symboles sera effectuée en construisant un réel r contenu dans l'intervalle E_n associé à la suite.

Le **décodage** d'une suite de n symboles à partir d'un réel r sera effectué en trouvant l'intervalle E_n qui contient r . La suite de symboles décodée est celle associée à l'intervalle E_n .

Les algorithmes de compression/décompression sont donc :

Compression

Entrée : une suite quelconque $c_1 c_2 \dots c_n$ de symboles.

$$E_0 = [0; 1).$$

pour $k = 1$ à n

$c_{[k]}$ = l'indice du symbole c_k dans S .

$$E_k = E_{k-1}^{\min} + |E_{k-1}| \times [F_{c_{[n]}-1}; F_{c_{[n]}}).$$

Sortie : r = milieu de l'intervalle E_n

Décompression

Entrée : un réel r codant n symboles

pour $k = 1$ à n

trouver l'indice $i \in \{1 \dots |S|\}$ tel quel $r \in [F_{i-1}; F_i)$

$$E_k = [F_{i-1}; F_i)$$

ajouter le symbole s_i à la sortie.

$$r = (r - E_k^{\min})/|E_k| = (r - F_{i-1})/(F_i - F_{i-1})$$

Sortie : $s = s_1 s_2 \dots s_n$.

EXERCICE 5: Principe du codage arithmétique

On souhaite effectuer le codage arithmétique des sources suivantes :

- ① Une source émet deux symboles $S = \{A, B\}$ avec les fréquences suivantes $\mathcal{F} = \{0.8, 0.2\}$.
 - a) Effectuer le codage arithmétique de la suite de symboles AAABA.
 - b) Quelle chaîne de 4 symboles représente le code arithmétique 0,7552.
- ② Une source émet deux symboles $S = \{A, B, C\}$ avec les fréquences suivantes $\mathcal{F} = \{0.4, 0.4, 0.2\}$.
 - a) Effectuer le codage arithmétique de la suite de symboles ABABC.
 - b) Quelle chaîne de 5 symboles représente le code arithmétique 0,14848.

PROPOSITION:

Si r est un nombre dans un intervalle $E \subset [0; 1)$, alors il existe un nombre $c(r)$ unique contenu dans E obtenu en tronquant la représentation binaire de r à :

$$\ell(c(r)) = \left\lceil \log_2 \frac{1}{\Pr(X)} \right\rceil + 1 \text{ bits}$$

où $\Pr(X) = |E|$ = probabilité uniforme de tomber dans E .

Exemple

Soit les symboles $S = \{a_1, a_2, a_3, a_4\}$ et leurs fréquences $\mathcal{F} = \{\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{8}\}$

x	E	$\Pr(X)$	$F(x)$	$c(r)$	binaire	$\ell(c(r))$	code
a_1	$[0; 0.5)$	0.5	0.5	0.25	.010	2	01
a_2	$[0.5; 0.75)$	0.25	0.75	0.625	.101	3	101
a_3	$[0.75; 0.875)$	0.125	0.875	0.8125	.1101	4	1101
a_4	$[0.875; 1)$	0.125	1	0.9375	.1111	4	1111

Remarques :

- notons que $c(r) = E^{\min} + \frac{1}{2} \Pr(X)$ (utile pour la démonstration ci-après).
- $c(r)$ permet donc de représenter tout intervalle E contenant r avec $\ell(c(r))$ bits.

Rappel : l'écriture en binaire de nombre $x \in [0; 1)$ est $.b_1 b_2 b_3 \dots$ où les $\{b_i\}$ sont tels que $x = \sum_i b_i 2^{-i}$.

Notations

- Pour $x \in [0; 1)$, on note $\lfloor x \rfloor_n$ la représentation binaire de x restreinte à n bits après la virgule.
- E_X = intervalle associé à la suite de symboles X .

Démonstration (unicité du code):

Soit X une suite de symboles.

Soit c_X le centre de l'intervalle $E_X = [E_X^{\min}; E_X^{\max})$ associé à cette suite.

Montrons que le code de $\lfloor c_X \rfloor_{\ell(X)}$ est unique dans E_X .

Comme $\lfloor c_X \rfloor_{\ell(X)}$ est tronqué au bit $\ell(X)$, on a nécessairement :

$$0 \leq c_X - \lfloor c_X \rfloor_{\ell(X)} < 2^{-\ell(X)} \quad (\text{rappel : } \sum_{i>n} 2^{-i} < 2^{-n}).$$

Par construction, on a $E_X^{\min} \leq c_X < E_X^{\max}$. Donc, $\lfloor c_X \rfloor_{\ell(X)} < E_X^{\max}$ car $\lfloor c_X \rfloor_{\ell(X)} < c_X$.

Montrons maintenant que $\lfloor c_X \rfloor_{\ell(X)} \geq E_X^{\min}$. Remarquons que :

$$2^{-\ell(X)} = 2^{-\lceil \log_2 \frac{1}{\Pr(X)} \rceil + 1} < 2^{-\log_2 \frac{1}{\Pr(X)} - 1} = 2^{\log_2 \Pr(X) - 1} = \frac{1}{2} \Pr(X)$$

Or, on a vu que $c_X = E_X^{\min} + \frac{1}{2} \Pr(X)$ (voir transparent précédent). En conséquence, $E_X^{\min} + \frac{1}{2} \Pr(X) - \lfloor c_X \rfloor_{\ell(X)} < \frac{1}{2} \Pr(X)$ et $E_X^{\min} < \lfloor c_X \rfloor_{\ell(X)}$.

Donc, $\lfloor c_X \rfloor_{\ell(X)} \in E_X$ permet bien de représenter c_X .

□

Démonstration (décodabilité de manière unique):

Si X et Y sont deux suites de codes distinctes, on sait que c_X et c_Y sont dans des intervalles disjoints ($E_X \cap E_Y = \emptyset, \forall X \neq Y$).

Si on montre que pour toute suite X , l'intervalle $[c_X, c_X + 2^{-\ell(X)})$ est inclus dans E_X , on montrera que le code d'une suite ne peut pas être le préfixe du code d'une autre suite.

On a déjà montré que : $c_X \geq E_X^{\min}$.

Il nous reste à montrer que : $c_X + 2^{-\ell(X)} < E_X^{\max}$.

Or, ceci est vrai $E_X^{\max} - c_X = c_X - E_X^{\min} = \frac{1}{2} \Pr[X]$ (car c_X est le milieu de E_X).

Donc $E_X^{\max} - c_X > 2^{-\ell(X)}$.

Le code X n'a donc pas de préfixe pour tout autre code $Y \neq X$. □

THÉORÈME: Efficacité du codage arithmétique

La longueur moyenne ℓ_A d'un symbole atteint par le code arithmétique d'une suite de longueur m vérifie : $H(X) \leq \ell_A < H(X) + \frac{2}{m}$.

Autrement dit, on peut approcher l'optimum de l'entropie d'aussi prêt que l'on veut, pour peu que la chaîne soit assez longue.

Démonstration :

La longueur moyenne du code pour une suite de longueur m est :

$$\ell_{A^m} = \sum \text{Pr}[X].\ell(X)$$

$$\text{Or } \ell(X) = \left\lceil \log_2 \frac{1}{\text{Pr}(X)} \right\rceil + 1 < \log_2 \frac{1}{\text{Pr}(X)} + 1 + 1 = -\log_2 \text{Pr}(X) + 2.$$

$$\text{Donc } \ell_{A^m} < -\sum P(X) \log_2 P(X) + 2 \sum P(X) = H(X^m) + 2$$

$$\text{En conséquence : } H(X^m) \leq \ell_{A^m} \leq H(X^m) + 2.$$

L'entropie pour la chaîne complète est à moins de 2 bits de l'optimum (borne supérieure !)

L'entropie pour un symbole est donnée par $\ell_A = \frac{1}{m} \ell_{A^m}$ et en se rappelant que $H(X^m) = m.H(X)$ pour des symboles indépendants.

$$\text{D'où } H(X) \leq \ell_A < H(X) + \frac{2}{m}.$$

□

2018-2019

Introduction

Codage de
Huffman

Codage
arithmétique

Principe

Unité

Efficacité

Codage

Décodage

Implémentation
entière

Codage adaptatif

Conclusion

Bibliographie

Pour l'implémentation, nous avons besoin de nombres réels avec une précision illimitée. Or, la représentation des flottants sur un ordinateur a une précision limitée (float : 23 bits, double : 52 bits), ce qui ne permet pas le calcul des intervalles (et de leurs centres) dès que la chaîne de symboles dépasse quelques unités.

Dans un premier temps, cherchons un algorithme itératif qui permette de construire le code binaire associé à un réel $r \in [0; 1)$.

r peut s'écrire $\sum_{i>0} b_i \cdot 2^{-i}$ où $\{b_i\}_{i>0}$ est le code binaire recherché.

Quelle est la valeur de b_1 ?

Comme $\sum_{i>1} 2^{-i} < 2^{-1}$, si $r > 2^{-1}$ alors $b_1 = 1$ sinon $b_1 = 0$.

Quelle est la valeur de b_2 ?

Soit $r_2 = r - b_1 \cdot 2^{-1}$

Avec le même argument, si $r_2 > 2^{-2}$ alors $b_2 = 1$ sinon $b_2 = 0$.

Et en continuant ainsi ...

Soit $r_k = r_{k-1} - b_{k-1} \cdot 2^{-(k-1)}$

Alors si $r_k > 2^{-k}$ alors $b_k = 1$ sinon $b_k = 0$

Tant que $r \neq 0$

Dans la méthode précédente, r_k devient de plus en plus petit. Sur un ordinateur, dès que la précision du flottant utilisé est atteinte, les calculs n'ont plus de sens.

Idee :

Faire une mise à l'échelle par 2 de r_k afin de conserver la précision.

Si le reste est multiplié par deux, alors il faut toujours :

- comparer à $\frac{1}{2}$,
- soustraire $\frac{1}{2}$ au reste

puisqu'à l'étape k , r_k aura été multiplié k fois par 2.

L'algorithme devient

r_0 = le nombre à décomposer en binaire.

$k = 1$

répéter

si $r_k > 2^{-1}$ **alors** $b_k = 1$ **sinon** $b_k = 0$

$k = k + 1$

$r_k = 2.(r_{k-1} - b_{k-1}.2^{-1})$

tant que $r_k \neq 0$

EXERCICE 6: Codage binaire

On utilise dans cette partie la méthode de codage en binaire qui utilise les mises à l'échelle successives pour coder un nombre binaire fractionnaire dans $[0; 1]$ sans nécessité une augmentation de précision arbitraire.

- 1 Quels est la caractéristique partagée par la représentation binaire de tous les nombres strictement inférieurs à 0,5 ?
Même question pour ceux supérieur à 0,5.
- 2 A quoi correspond l'opération de mise à l'échelle sur le mot binaire ?
- 3 Appliquer cet algorithme pour trouver le codage binaire de 0,2 à 5 bits après la virgule.
- 4 Appliquer cet algorithme pour trouver le codage binaire de 0,92 à 5 bits après la virgule.

2018-2019

Introduction

Codage de
Huffman

Codage
arithmétique

Principe

Unicité

Efficacité

Codage

Décodage

Implémentation
entière

Codage adaptatif

Conclusion

Bibliographie

On voudrait appliquer ici cette méthode sur l'intervalle E_k , sachant qu'il est précisé au fur et à mesure de la lecture des symboles.

Le représentant recherché est inclus dans E_k donc le choix d'un bit doit concerner la totalité de l'intervalle.

On sait par ailleurs que pour tout E_i tel que $i > k$, alors $E_i \subset E_k$. Donc, le choix fait pour E_k restera valide lors de la lecture des symboles suivants.

La comparaison par rapport à $\frac{1}{2}$ devient donc :

- ① lire le symbole suivant et mettre à jour E_k .
- ② si $E_k \subset [0; \frac{1}{2})$, alors choisir 0 et doubler l'échelle.
- ③ si $E_k \subset [\frac{1}{2}; 1)$, alors choisir 1 et doubler l'échelle.

où les tests 2 et 3 sont à répéter tant que l'une des conditions est vérifiée.

Conséquences :

- Plusieurs bits peuvent être produits après la lecture d'un seul symbole.
- Réciproquement, la lecture d'un symbole peut ne produire aucun bit.

2018-2019

Introduction

Codage de
Huffman

Codage
arithmétique

Principe

Unicité

Efficacité

Codage

Décodage

Implémentation
entière

Codage adaptatif

Conclusion

Bibliographie

Soit :

L'alphabet $S = \{s_1, \dots, s_n\}$.

Les fréquences des symboles : $\mathcal{F} = \{p_1, \dots, p_n\}$.

La fonction de répartition : $\{F_k\}_{k=0\dots n}$ où $F_k = \sum_{i \leq k} p_i$ et $F_0 = 0$.

Les deux fonctions de mise à l'échelle :

$S_1 : x \mapsto 2x$ (mise à l'échelle $[0; 0.5) \rightarrow [0; 1)$)

$S_2 : x \mapsto 2(x - 0.5)$ (mise à l'échelle $[0.5; 1) \rightarrow [0; 1)$)

Génération du code arithmétique pour une chaîne d'entrée : $c_1 c_2 \dots c_p$

$E_0 = [0; 1)$

Pour k allant de 1 à p

i = indice de c_k dans la table des symboles.

$E_k = E_{k-1}^{\min} + |E_{k-1}| \cdot [F_{i-1}; F_i]$

tant que $(E_k^{\min} \geq 0.5)$ ou $(E_k^{\max} < 0.5)$

si $E_k \subset [0; 0.5)$ alors sortie $\leftarrow 0$ et $E_k = S_1(E_k)$.

si $E_k \subset [0.5; 1)$ alors sortie $\leftarrow 1$ et $E_k = S_2(E_k)$.

Ajouter 1 à la sortie.

Remarque

A la fin de la génération du code, on a $E_k^{\min} < 0.5$ et $E_k^{\max} \geq 0.5$.

Donc $.1 = 0.5$ est toujours un représentant de l'intervalle final et est utilisé comme marqueur de fin de chaîne.

Exemple

Soit $S = \{1, 2, 3\}$ et $\mathcal{F} = \{0.8, 0.02, 0.18\}$.

La fonction de répartition est : $\{F(k)\}_{k=0\dots 3} = \{0, 0.8, 0.82, 1\}$.

Soit la chaîne d'entrée : 1321

L'intervalle initial est $E_0 = [0; 1)$.

Lecture 1321 : $E_1 = E_0^{\min} + |E_0|. [F_0; F_1] = 0 + 1.[0; 0.8) = [0; 0.8)$

Lecture 1321 : $E_2 = E_1^{\min} + |E_1|. [F_2; F_3] = 0 + 0.8.[0.82; 1) = [0.656; 0.8)$

$E_2 \subset [0.5; 1) \Rightarrow$ sortie : 1, échelle : $E_2 = S_2(E_2) = [0.312; 0.6)$

Lecture 1321 : $E_3 = E_2^{\min} + |E_2|. [F_1; F_2] = [0.5424; 0.54816)$

$E_3 \subset [0.5, 1) \Rightarrow$ sortie : 1, échelle : $E_3 = S_2(E_3) = [0.0848; 0.09632)$

$E_3 \subset [0; 0.5) \Rightarrow$ sortie : 0, échelle : $E_3 = S_1(E_3) = [0.1696; 0.19264)$

$E_3 \subset [0; 0.5) \Rightarrow$ sortie : 0, échelle : $E_3 = S_1(E_3) = [0.3392; 0.38528)$

$E_3 \subset [0; 0.5) \Rightarrow$ sortie : 0, échelle : $E_3 = S_1(E_3) = [0.6784; 0.77056)$

$E_3 \subset [0.5; 1) \Rightarrow$ sortie : 1, échelle : $E_3 = S_2(E_3) = [0.3568; 0.54112)$

Lecture 1321 : $E_4 = E_3^{\min} + |E_3|. [F_0; F_1] = [0.3568; 0.504256)$

fin

On ajoute 1 au code ($.1_b = 0.5$) et l'on complète éventuellement par des 0 pour respecter la longueur des mots.

Le code compressé est donc : 1100011_b

Comment commencer à décoder ?

Il faut avoir suffisamment d'informations pour décoder les symboles sans ambiguïté. Pour cela :

① calculer la plus petite probabilité : $p_{\min} = \min_k p_k$.

② soit q_{\min} , le plus grand q tel que $2^{-q} < \frac{1}{2}p_{\min}$

$$\text{i.e. } q_{\min} = \lceil \log_2 \frac{1}{p_{\min}} \rceil + 1$$

à savoir les bits au-delà de q_{\min} ne permettent plus de sortir du plus petit intervalle de la fonction de répartition (= celui de largeur p_{\min}).

On utilise une fenêtre glissante d'au moins q bits au fur et à mesure du décodage.

Exemple

Dans l'exemple précédent, la plus petite probabilité est $p_2 = 0.02$.

On calcule $k = \lceil \log_2 \frac{1}{0.02} \rceil + 1 = 7$.

Les 7 premiers bits du code sont $.1100011_b$, on obtient $r = 0.7734375$.

2018-2019

Introduction

Codage de
Huffman

Codage
arithmétique

Principe

Unicité

Efficacité

Codage

Décodage

Implémentation
entière

Codage adaptatif

Conclusion

Bibliographie

Comment continuer à décoder ?

Soit m la chaîne binaire à décoder.

La fonction $\text{sub}(m, p, q)$ renvoie q bits de m à partir de la position p .

initialisation : $k = 1$, $p = 1$, $q = q_{\min}$, $E_0 = [0; 1)$

répéter (jusqu'à l'arrêt)

r = le réel de $[0; 1)$ représenté par $\text{sub}(m, p, q)$.

$t = (r - E_{k-1}^{\min}) / |E_{k-1}|$ = position de r dans E_{k-1} .

i = l'indice tel que $F_{i-1} \leq t < F_i$

Le symbole décodé est s_i .

Mise à jour de l'intervalle : $E_k = E_{k-1}^{\min} + [F_{i-1}; F_i) \cdot |E_{k-1}|$

si ($E_k^{\min} \leq 0.5$) et ($0.5 < E_k^{\max}$)

alors $q = q + 1$ (augmenter la précision)

sinon

tant que ($E_k^{\min} \geq 0.5$) ou ($E_k^{\max} < 0.5$) */// vérifier $0.5 \in E_k$?*

/// mise à l'échelle et décalage

si $E_k \in [0; 0.5)$ alors $E_{k+1} = S_1(E_k)$ et $p = p + 1$

si $E_k \in [0.5; 1)$ alors $E_{k+1} = S_2(E_k)$ et $p = p + 1$

fin tant que

$q = q_{\min}$ */// réinitialisation de la précision*

$k = k + 1$

fin répéter

2018-2019

Introduction

Codage de
Huffman

Codage
arithmétique

Principe

Unicité

Efficacité

Codage

Décodage

Implémentation
entière

Codage adaptatif

Conclusion

Bibliographie

Comment arrêter de décoder ?

On rappelle qu'à la fin du codage :

- on ajoute un $.1_b$ pour signifier le centre de l'intervalle à l'échelle à laquelle on se trouve à la fin du codage.
note : 0.5 fait toujours partie de l'intervalle terminal.
- ce 1 est complété par un ensemble de 0 (padding sur la fin du mot courant).

La condition d'arrêt indiquée lors du décodage correspond donc à la conjonction de deux conditions :

- on est arrivé à la fin de la chaîne.
à savoir, $p + q$ est strictement supérieur à la longueur de la chaîne à décoder (quand cette condition est réalisée, c'est que l'on est en train de décoder le tout dernier symbole).
- $r = 0.5$

Exemple :

Initialisation : $E_0 = [0; 1)$, $q_{\min} = 7$, $q = q_{\min}$, $p = 1$, $m = .1100011_b$

1 $r = \text{sub}(m, p, q) = \text{sub}(m, 1, 7) = .1100011_b = 0.7734375$
 $t = (r - E_0^{\min})/|E_0| = 0.7734375$

$i = 1$ car $t \in [F_0, F_1) = [0; 0.8) \Rightarrow$ sortie : 1

$E_1 = E_0^{\min} + [F_0; F_1) \times |E_0| = [0; 0.8)$

Augmentation précision : $q = q + 1 = 8$

2 $r = \text{sub}(m, p, q) = \text{sub}(m, 1, 8) = .11000110_b = 0.7734375$
 $t = (r - E_1^{\min})/|E_1| = 0.966796875$

$i = 3$ car $t \in [F_2; F_3) = [0.82; 1) \Rightarrow$ sortie : 3

$E_2 = E_1^{\min} + [F_2; F_3) \times |E_1| = [0.656; 0.8)$

3 **Mise à l'échelle :** $E_3 = S_2(E_2) = [0.312; 0.6)$

Décalage : $p = p + 1 = 2$

Réinitialisation de la précision : $q = q_{\min} = 7$

4 $r = \text{sub}(m, 2, 7) = .1000110_b = 0.546875$
 $t = (r - E_3^{\min})/|E_3| = 0.8155$

$i = 2$ car $t \in [F_1; F_2) = [0.8; 0.82) \Rightarrow$ sortie : 2

$E_4 = E_3^{\min} + [F_1; F_2) \times |E_3| = [0, 5424; 0, 54816)$

Exemple (suite)

5 Mise à l'échelle : $E_5 = S_2(E_4) = [0.0848; 0.09632]$

Décalage : $p=p+1=3$

6 Mise à l'échelle : $E_6 = S_1(E_5) = [0.1696; 0.19264]$

Décalage : $p=p+1=4$

7 Mise à l'échelle : $E_7 = S_1(E_6) = [0.3392; 0.38528]$

Décalage : $p=p+1=5$

8 Mise à l'échelle : $E_8 = S_1(E_7) = [0.6784; 0.77056]$

Décalage : $p=p+1=6$

9 Mise à l'échelle : $E_9 = S_2(E_8) = [0.3568; 0.54112]$

Décalage : $p=p+1=7$

Réinitialisation de la précision : $q = q_{\min} = 7$

10 $r = \text{sub}(m, p, q) = \text{sub}(m, 7, 7) = .1_b = 0.5$

$t = (r - E_9^{\min}) / |E_9| = 0,7769$

$i = 1$ car $t \in [F_0; F_1) = [0; 0.8) \Rightarrow$ sortie : 1

$E_{10} = E_9^{\min} + [F_0; F_1) \times |E_9| = [0.3568; 0,5043]$

Condition d'arrêt atteinte : $r = 0.5$ et $p = |m|$

2018-2019

Introduction

Codage de
Huffman

Codage
arithmétique

Principe

Unité

Efficacité

Codage

Décodage

Implémentation
entière

Codage adaptatif

Conclusion

Bibliographie

EXERCICE 7: Codage arithmétique avec mise à l'échelle

Considérons une source qui émet les symboles $S = \{A, B, C\}$ avec les fréquences suivantes $\mathcal{F} = \{0.4, 0.4, 0.2\}$.

- 1 Effectuer le codage arithmétique de la suite de symboles *ABABC* avec mise à l'échelle.
- 2 Effectuer le décodage du code arithmétique *.0011011* avec mise à l'échelle.

L'exemple précédent est un cas spécifique où les calculs fonctionnent comme attendu, mais les calculs en flottant posent des problèmes spécifiques.

Problème de l'implémentation flottante :

- **rappel** : la représentation flottante d'un représentant de chaque intervalle de la fonction de distribution peut être représentée sur q_{\min} bits. Au cours de l'algorithme, la valeur de q utilisée peut varier, notamment lors des mises à l'échelle.
- comme r = le réel de $[0; 1)$ représenté par $sub(m, p, q)$, l'incertitude sur la position de r est donc de 2^{-q} .
- or, on calcule $t = (r - E_{k-1}^{\min})/|E_{k-1}|$, et donc $r = t \cdot |E_{k-1}| + E_{k-1}^{\min}$.
- on en déduit que l'incertitude en r est donc proportionnelle à celle en t (on a $\delta r = \delta t \cdot |E_{k-1}|$).

Mais t doit se retrouver dans le même intervalle que celui dans lequel il était lorsque l'on a fait la compression.

Comment faire pour s'en assurer ?

Solution pour l'implémentation flottante :

- Il faut donc que la distance de t aux bornes de l'intervalle j dans lequel t se trouve soit supérieure à 2^{-q} , et choisir la valeur de q tel que cela soit le cas. A savoir,
calculer $\delta = \min(t - E_j, E_{j+1} - t) \cdot |E_{k-1}|$
si $\delta > 2^{-q}$, alors tout va bien, t ne peut pas changer d'intervalle.
sinon, mettre à jour la valeur de $q = \lceil \log_2 \frac{1}{\delta} \rceil + 1$, et réitérer le processus.
- Par réitérer le processus, on entend : tant que $\delta < 2^{-q}$, il y a un risque que t change d'intervalle (et donc que le symbole décodé soit mauvais).

Remarque :

comme on peut avoir δ très petit (voir nul), on prendra comme valeur de q le minimum entre la valeur calculée, et le nombre de bits maximum de précision (soit 32bits/64bits si le calcul est en flottant simple/double précision, ou le nombre de bits restant sur le support s'il est inférieur à la précision flottante).

Cas particulier

Supposons que $\mathcal{F} = \left\{\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right\}$.

Que se passe-t-il si l'on tente de compresser la suite $s_2 s_2 \dots s_2 s_2$?

On reste en permanence sur l'intervalle central : les mises à l'échelle S_1 ou S_2 sont donc inopérantes (aucun des sous-intervalles n'est inclus dans $[0; 0.5)$ ou $[0.5; 1)$).

Une perte de précision a rapidement lieu :

choix d'un intervalle de taille 2^{-n} = besoin de n bits de précision en plus.

Il faut alors introduire une mise à l'échelle supplémentaire S_3 sur le centre :

si $x \in [0.25; 0.75)$, utiliser $S_3 : x \mapsto 2(x - 0.25)$

pour contourner ce problème.

Problème : comment signaler au décodeur qu'une mise à l'échelle S_3 a été effectuée sans allonger le code ?

Cas particulier (suite)

On remarque que :

$$S_3 \circ S_1(x) = 2(2(x - 0.25)) = 4x - 1$$

$$S_1 \circ S_2(x) = 2(2x) - 1 = 4x - 1$$

et que :

$$S_3 \circ S_2(x) = 2(2(x - 0.25)) - 1 = 4x - 2$$

$$S_2 \circ S_1(x) = 2(2x - 1) = 4x - 2$$

Donc : $S_3 \circ S_1 = S_1 \circ S_2$ et $S_3 \circ S_2 = S_2 \circ S_1$.

Autrement dit, lorsque l'on applique une mise à l'échelle S_3 , s'en souvenir.

Puis à mise à l'échelle suivante :

si c'est S_1 alors sortie $\leftarrow 01$

si c'est S_2 alors sortie $\leftarrow 10$

Ceci peut se généraliser à des suites $S_3^n = S_3 \circ \dots \circ S_3$ de longueur n . On peut montrer que $S_3^n \circ S_1 = S_1 \circ S_2^n$ et que $S_3^n \circ S_2 = S_2 \circ S_1^n$.

En conséquence, utiliser un compteur n pour se souvenir du nombre de mises à l'échelle S_3 consécutives. Puis à la mise à l'échelle suivante :

si c'est S_1 alors sortie $\leftarrow 0$ suivi de n 1

si c'est S_2 alors sortie $\leftarrow 1$ suivi de n 0

Cas particulier (suite)

L'algorithme de codage devient :

soit $c = c_1 \dots c_n$ l'ensemble des caractères à coder.

$$E_0 = [0; 1)$$

$p = 0$ // *pour compter le nombre de S_3 consécutifs*

Pour k allant de 1 à n

i = indice de c_k dans la table des symboles.

$$E_k = E_{k-1}^{\min} + |E_{k-1}| \cdot [F_{i-1}; F_i)$$

tant que $(E_k \subset [0; 0.5))$ ou $(E_k \subset [0.25; 0.75))$ ou $(E_k \subset [0.5; 1))$

si $E_k \subset [0.25; 0.75)$ alors $p++$ et $E_k = S_3(E_k)$.

si $E_k \subset [0; 0.5)$ alors sortie $\leftarrow 0 + p$, $E_k = S_1(E_k)$ et $p = 0$.

si $E_k \subset [0.5; 1)$ alors sortie $\leftarrow 1 + p$, $E_k = S_2(E_k)$ et $p = 0$.

Ajouter p 0 et 1 à la sortie.

Important

Il faut donc ajouter à l'algorithme de décodage le cas où $0.25 \leq E_k^{\min} < 0.5 \leq E_k^{\max} < 0.75$, effectuer la transformation S_3 et lire le bit suivant.

Voyons maintenant comment effectuer une implémentation utilisant l'arithmétique entière.

On dispose d'un échantillon représentatif $\{x_j\}_{j=1\dots n_e}$ de taille n_e des tirages des symboles de $s_i \in \mathcal{S}$. On en déduit :

- les comptages : $\tilde{n}_i = \#\{x_j \mid x_j = s_i\}$ (donc $\tilde{n}_i/n_e =$ fréquence empirique).
- les comptages cumulés : $\tilde{F}_i = \sum_{j \leq i} \tilde{n}_j$ (donc $\tilde{F}_i/n_e =$ distribution empirique).

Le plus petit intervalle associé à un symbole est de taille $\min_i \tilde{n}_i/n_e$. Cet intervalle doit occuper au moins 2 bits de précision (donc 4 valeurs), afin qu'il soit possible de déterminer s'il est nécessaire de le redimensionner.

Donc, la précision entière à choisir doit permettre de représenter au moins $4.n_e / \min_i \tilde{n}_i$ valeurs (ou $4 / \min_i f_i$, si les fréquences f_i des symboles sont connues).

En conséquence, nous avons besoin d'un minimum de $n_b = \log_2(4.n_e / \min_i \tilde{n}_i)$ bits pour réaliser les calculs en entier.

Nous utilisons maintenant les entiers non signés pour représenter des nombres dans $[0; 1)$. Si le codage est sur n_b bits, alors :

$0 \dots 0_b$ représente 0.

$1 \dots 1_b$ représente 1 (plus précisément $(2^{n_b} - 1)/2^{n_b}$).

Les calculs en entier s'effectuent alors tout simplement en renormalisant l'ensemble des valeurs par 2^{n_b} .

Si s_i est le symbole à coder, les équations de mise à jour des bornes de l'intervalle $E_k = \{E_k^{\min}, E_k^{\max}\}$ deviennent :

$$\begin{aligned} E_k &= E_{k-1}^{\min} + \{\lfloor |E_{k-1}| \cdot F_{i-1} \rfloor, \lfloor |E_{k-1}| \cdot F_i \rfloor\} \\ &= E_{k-1}^{\min} + \left\{ \left\lfloor |E_{k-1}| \cdot \tilde{F}_{i-1} / n_e \right\rfloor, \left\lfloor |E_{k-1}| \cdot \tilde{F}_i / n_e \right\rfloor - 1 \right\} \end{aligned}$$

où $|E_k| = E_k^{\max} - E_k^{\min} + 1$.

2018-2019

Introduction

Codage de
Huffman

Codage
arithmétique

Principe

Unicité

Efficacité

Codage

Décodage

Implémentation
entière

Codage adaptatif

Conclusion

Bibliographie

La valeur du MSB de l'entier associé à l'une des bornes v de l'intervalle permet de connaître sa position dans l'intervalle :

$v_b =$	0xxxxxxx	1xxxxxxx	00xxxxxx	01xxxxxx	10xxxxxx	11xxxxxx
$v \in$	$[0; 0.5)$	$[0.5; 1)$	$[0; 0.25)$	$[0.25; 0.5)$	$[0.5; 0.75)$	$[0.75; 1)$

Remarquons maintenant que :

- $S_1(x) = 2x$ si $x < 0.5$ est équivalent à $x \ll 1$.
- $S_2(x) = 2(x - 0.5)$ si $x \geq 0.5$ est aussi équivalent à $x \ll 1$ (car le MSB=0.5 est perdu par le décalage).
- $S_3(x) = 2(x - 0.25)$ est équivalent à $x \ll 1$ puis à compléter le MSB.

Le LSB (bit ajouté après décalage) sera complété par 0 ou 1 suivant que l'on soit sur la borne haute ou basse de l'intervalle (afin de "coller" aux bornes).

On définit donc les fonctions suivantes :

$\text{MSB}(b_1 \dots b_n) = b_1$ (bit le plus significatif)

$\text{MSB}_2(b_1 \dots b_n) = b_1 b_2$ (2 bits les plus significatifs)

$\text{SMSB}(b_1 \dots b_n) = b_2$ (second bit le plus significatifs).

$\text{Shift}_{12}(b_1 \dots b_n, a) = \overline{b_2} \dots b_n a$ (shift gauche + LSB=a)

$\text{Shift}_3(b_1 \dots b_n) = \overline{b_2} \dots b_n a$ (shift gauche + complément MSB + LSB = a)

A noter que $\text{MSB}(E_k^{\min}) = \text{MSB}(E_k^{\max}) \Leftrightarrow E_k \subset [0; 0.5)$ ou $E_k \subset [0.5; 1)$ et que $(\text{MSB}_2(E_k^{\min}) = 01)$ et $(\text{MSB}_2(E_k^{\max}) = 10) \Leftrightarrow E_k \subset [0.25; 0.75)$ et $0.5 \in E_k$.

Algorithme de codage entier

$E_0 = \{0 \dots 0_b, 1 \dots 1_b\}$

$u = 0$ // *compteur pour S_3*

while ($\neg EOF$) **do**

$x = \text{ReadSymbol}()$

$E_k = E_{k-1}^{\min} + \left\{ \left\lfloor |E_{k-1}| \cdot \tilde{F}_{i-1} / n_e \right\rfloor, \left\lfloor |E_{k-1}| \cdot \tilde{F}_i / n_e \right\rfloor - 1 \right\}$

while ($MSB(E_k^{\min}) = MSB(E_k^{\max})$) *ou* ($MSB_2(E_k^{\min}, E_k^{\max}) = (01, 10)$)

do

if ($MSB(E_k^{\min}) = MSB(E_k^{\max})$) **then**

$b = MSB(E_k^{\min})$

$E_k = \{ \text{Shift}_{12}(E_k^{\min}, 0), \text{Shift}_{12}(E_k^{\max}, 1) \}$

$\text{sortie} \leftarrow b$

while ($u > 0$) **do**

$\text{sortie} \leftarrow \bar{b}$

$u = u - 1$

if ($MSB_2(E_k^{\min}, E_k^{\max}) = (01, 10)$) **then**

$E_k = \{ \text{Shift}_3(E_k^{\min}, 0), \text{Shift}_3(E_k^{\max}, 1) \}$

$u = u + 1$

$\text{sortie} \leftarrow b_1 + u \times \overline{b_1} + b_2 \dots b_n$ où $b_1 \dots b_n = E_k^{\min}$

La fonction entière de décodage est entièrement symétrique :

- 1 on commence à accumuler n_b bits pour construire la valeur de référence.
- 2 ceci permet d'identifier le premier intervalle.
- 3 à partir de cet intervalle, on identifie le symbole associé.
- 4 puis on applique autant de transformation de l'intervalle que nécessaire pour se retrouver avec une valeur de référence au centre.
chaque transformation lit un bit supplémentaire sur l'entrée mettant à jour la valeur de référence.
- 5 recommencer en 3 tant qu'il reste des bits à lire.

On utilisera la fonction $\text{getInterval}(v)$ retourne le numéro de l'intervalle $[F_{i-1}, F_i)$ dans lequel se trouve la valeur de référence $v \in E$:

$\text{getInterval}(v, E)$

```
k = 0  
while ( $\lfloor \frac{(v - E^{\min}) \cdot n_e - 1}{|E|} \rfloor \geq \tilde{F}_k$ ) do k = k + 1  
renvoyer k
```


Algorithme de décodage entier

$E_0 = \{0 \dots 0_b, 1 \dots 1_b\}$

$t = \text{ReadBit}(n_b)$ // lit n_b bits

while (!EOF) **do**

$i = \text{getInterval}(v, E_{k-1})$

 sortie $\leftarrow s_i$ // décode un symbole

$E_k = E_{k-1}^{\min} + \left\{ \left\lfloor |E_{k-1}| \cdot \tilde{F}_{i-1} / n_e \right\rfloor, \left\lfloor |E_{k-1}| \cdot \tilde{F}_i / n_e \right\rfloor - 1 \right\}$

while ($\text{MSB}(E_k^{\min}) = \text{MSB}(E_k^{\max})$ ou $(\text{MSB}_2(E_k^{\min}, E_k^{\max}) = (01, 10))$

do

if ($\text{MSB}(E_k^{\min}) = \text{MSB}(E_k^{\max})$) **then**

$E_k = \{\text{Shift}_{12}(E_k^{\min}, 0), \text{Shift}_{12}(E_k^{\max}, 1)\}$

$b = \text{ReadBit}()$

$t = \text{Shift}_{12}(t, b)$

if ($\text{MSB}_2(E_k^{\min}, E_k^{\max}) = (01, 10)$) **then**

$E_k = \{\text{Shift}_3(E_k^{\min}, 0), \text{Shift}_3(E_k^{\max}, 1)\}$

$b = \text{ReadBit}()$

$t = \text{Shift}_{12}(t, b)$

Note : un exemple sera fait en TD.

2018-2019

Introduction

Codage de
Huffman

Codage
arithmétique

Principe

Unicité

Efficacité

Codage

Décodage

Implémentation
entière

Codage adaptatif

Conclusion

Bibliographie

EXERCICE 8: Codage arithmétique entier

Considérons une source qui émet les 50 symboles $S = \{A, B, C\}$ avec les comptages suivants $\{\tilde{n}_i\} = \{40, 1, 9\}$.

- 1 Calculer les comptages cumulés.
- 2 Calculer le nombre minimum de bits pour réaliser les calculs en entier.
- 3 Effectuer le codage arithmétique de la suite de symboles $ACBA$ avec mise à l'échelle.
- 4 Effectuer le décodage du code arithmétique 1100010010000000 avec mise à l'échelle.

Dans le cas où la distribution n'est pas connue, il est relativement simple de modifier l'algorithme vu afin de le rendre adaptatif :

- 1 on utilise une distribution empirique dont le nombre d'occurrences par symbole est initialisé à 1 pour tous les symboles.
- 2 lors du codage, à chaque symbole, on code le symbole, puis on met à jour la distribution empirique avec l'occurrence de ce symbole.
- 3 lors du décodage, on décode le symbole, puis on met à jour la distribution empirique.

La modification de la distribution empirique n'ayant lieu qu'après le codage/décodage du symbole, la mise à jour est symétrique.

Problème dans le cas de l'implémentation entière

Comme nous l'avons déjà vu, le nombre de bits n_b de codage limite la taille minimale de l'intervalle à 2^{n_b-2} (2 bits de précision sur le plus petit intervalle). En conséquence, lorsque le comptage d'un symbole approche cette valeur, il faut renormaliser la totalité des compteurs symboles avec $\tilde{n}_i = \lceil \tilde{n}_i/2 \rceil$.

En pratique,

- le codage de Huffman est plus simple à mettre en oeuvre et permet de construire des codes à $p_{\max} + 0.086$ du maximum de l'entropie sans passer à des blocs.
- le codage arithmétique est relativement complexe à mettre en oeuvre, mais il est plus performant que le codage de Huffman classique.
- le codage de Huffman par bloc est théoriquement meilleur que le codage arithmétique, mais le nombre de codes étant exponentiel (k^m pour des blocs de taille m sur un alphabet de taille k), il devient très rapidement impraticable et l'optimum ne peut être atteint.
- le codage arithmétique n'a pas besoin de construire un code, et il en existe des implémentations très performantes [RM89].
- en pratique, un codage de Huffman ou arithmétique est généralement utilisé derrière d'autres méthodes permettant spécifiquement de supprimer la redondance spatiale.

Les codes de Tunstall pourraient être ajoutés.

- [Gal78] Robert G. Gallager.
Variations on a theme by huffman.
IEEE Transactions on Information Theory, 24(6) :668–674, 1978.
- [RM89] Jorma Rissanen and Kottappuram Mohammed Mohiuddin.
A multiplication-free multialphabet arithmetic code.
IEEE Transactions on Communications, 37(2) :93–98, 1989.
- [Say06] Khalid Sayood.
Introduction to data compression.
Elsevier, Boston, 2006.
- [SMB10] David Salomon, Giovanni Motta, and David Bryant.
Handbook of data compression.
Springer, London, 2010.