

Transmission numérique

Bureau d'étude n° 3

Mathieu Goutelle

5 décembre 2001

1 Introduction

Nous allons décrire dans ce BE deux méthodes de compression. L'une est une méthode conservative et utilise l'algorithme développé par Lempel et Ziv encore appelé algorithme LZ et ses dérivés. Cette méthode est principalement utilisée pour les textes et les images non photographiques type schéma ou courbe avec le format GIF. La seconde méthode est cette fois non-conservative : il s'agit de la méthode utilisée dans le format JPEG, très efficace pour les images photographiques.

2 Méthode de compression LZW

2.1 Présentation

La méthode de compression¹ mise au point par Lempel, Ziv et Welch est basée sur le constat suivant : lorsqu'une séquence de caractères est répétée dans un texte, il est avantageux de transmettre une adresse pour désigner cette séquence de caractères plutôt que la séquence elle-même, et ce d'autant plus que la séquence est longue.

Cette méthode de compression s'inscrit dans la catégorie des méthodes à dictionnaire. L'un de ses avantages est que la phase de décompression n'a pas besoin du dictionnaire créé lors de la phase de compression. Ceci est particulièrement intéressant car le dictionnaire ne sera pas stocké avec les données et augmente encore les avantages de cette méthode par rapport aux autres méthodes à dictionnaire.

Cette méthode travaille sur les caractères (code ASCII étendu sur 8 bits). Elle va produire une succession d'adresses, codées sur 9 bits ou plus, pointant sur un élément du dictionnaire. Il est important de noter que dans le fichier produit apparaîtront également quatre mots de commandes, représentés par les codes 256 à 259. Ces commandes servent par exemple à spécifier une augmentation de la taille du dictionnaire (le cas échéant).

Les spécificités de l'algorithme LZW par rapport aux autres méthodes de la famille LZ sont :

- Le résultat contient uniquement des mots-codes. Cela signifie que le dictionnaire ne peut être vide au début : il doit contenir toutes les racines qui peuvent apparaître dans le flux d'entrée ;
- Puisque toutes les racines sont déjà présentes dans le dictionnaire, chaque étape de l'encodage commence avec un préfixe de longueur 1. La première chaîne cherchée dans le dictionnaire contiendra donc deux caractères ;
- Le caractère avec lequel le nouveau préfixe commence est le dernier caractère de la chaîne précédente (*C*). Ceci est indispensable pour permettre à l'algorithme de décodage de reconstruire le dictionnaire sans la présence explicite des caractères dans le flux codé.

1. Réf. : <http://www.rasip.fer.hr/research/compress/algorithms/fund/lz/>

2.2 Description de l'encodage

Pour décrire l'algorithme, nous allons utiliser les termes et notations suivantes :

Flux de caractères : la séquence de données à encoder ;

Caractère : l'élément de base du flux de caractères ;

Préfixe : une séquence de caractère qui en précède un ;

Chaîne : le préfixe et le caractère qu'il précède ;

Mot-code : un élément de base du flux codé. Il représente une chaîne du dictionnaire ;

Flux codé : la séquence de mots-codes et de caractères (le résultat de l'algorithme) ;

Dictionnaire : un tableau de chaînes. À chaque chaîne est associé un mot-code en fonction de son index dans le tableau ;

Racine : une racine est une chaîne de caractère de longueur 1 ;

Préfixe courant : le préfixe en train d'être traité par l'algorithme d'encodage. Symbole : P ;

Caractère courant : un caractère déterminé dans l'algorithme d'encodage. Généralement, il s'agit du caractère suivant le préfixe courant. Symbole : C ;

Mot-code courant : le mot-code juste produit par l'algorithme de décodage. Il est représenté par W et la chaîne qu'il représente par $string.W$.

L'algorithme d'encodage peut être décrit de la façon suivante :

1. Au début, le dictionnaire contient toutes les racines possibles et $P = \emptyset$;
2. C est assigné au caractère suivant du flux d'entrée ;
3. La chaîne $P + C$ est-elle présente dans le dictionnaire ?
 - (a) si oui, $P \leftarrow P + C$ (concaténation) ; i
 - (b) sinon,
 - i. le mot-code représentant P est ajouté au flux de sortie ;
 - ii. la chaîne $P + C$ est ajouté au dictionnaire ;
 - iii. $P \leftarrow C$;
4. Le flux d'entrée est-il vide ?
 - (a) si non, il faut retourner à l'étape 2 ;
 - (b) si oui,
 - i. le mot-code représentant P est ajouté au flux de sortie ;
 - ii. FIN.

2.3 Description du décodage

Pour décrire l'algorithme de décodage, définissons deux concepts supplémentaires :

Mot-code courant : le mot-code en train d'être traité. Il est noté cW et la chaîne qu'il représente $string.cW$;

Mot-code précédent : le mot-code qui précède le mot-code courant dans le flux codé. Il est noté pW et la chaîne qu'il représente $string.pW$.

Au démarrage, le dictionnaire contient également tous les caractères contenus dans le flux d'entrée. Lors d'une étape intermédiaire du processus de décodage, l'algorithme se souvient du code précédent (pW) et lit alors le mot-code courant (cW) du flux codé. Il ajoute la chaîne $string.cW$ à la sortie et ajoute au dictionnaire la chaîne $string.pW$ concaténée avec le premier caractère de la chaîne $string.cW$. À cause de ce principe, l'algorithme de décodage est toujours décalé d'une étape en arrière par rapport à l'algorithme d'encodage, en ce qui concerne l'ajout des nouvelles chaînes au dictionnaire.

Un cas particulier survient lorsque cW représente une chaîne vide dans le dictionnaire. Ceci peut arriver, à cause du retard par rapport à l'algorithme d'encodage, lorsque ce dernier lit une chaîne qu'il vient juste d'ajouter au dictionnaire. Au cours du décodage, cette chaîne n'est pas encore présente dans le dictionnaire. Une chaîne peut apparaître deux fois dans le flux de caractères

si et seulement si le caractère de début et de fin sont égaux : la chaîne suivante commence en effet toujours par le dernier caractère de la précédente. Il faut donc ajouter la règle de décodage suivante : la chaîne *string.pW* est concaténée avec son propre premier caractère et la chaîne résultante est ajouté au dictionnaire et au flux de sortie.

On obtient alors l'algorithme suivant :

1. Au début, le dictionnaire contient toutes les racines possibles ;
2. cW est assigné au premier caractère du flux codé ;
3. la chaîne *string.cW* est ajouté au flux de caractères ;
4. $pW \leftarrow cW$;
5. cW est assigné au caractère suivant du flux codé ;
6. La chaîne *string.cW* est-elle présente dans le dictionnaire ?
 - (a) si oui, $P \leftarrow P + C$ (concaténation) ;
 - i. la chaîne *string.cW* est ajouté au flux de sortie ;
 - ii. $P \leftarrow \text{string.pW}$;
 - iii. C est assigné au premier caractère de la chaîne *string.cW* ;
 - iv. la chaîne $P + C$ est ajouté au dictionnaire ;
 - (b) sinon,
 - i. $P \leftarrow \text{string.pW}$;
 - ii. C est assigné au premier caractère de la chaîne *string.cW* ;
 - iii. la chaîne $P + C$ est ajouté au flux de sortie et au dictionnaire ;
 - iv. $\text{string.cW} \leftarrow P + C$;
7. Le flux codé est-il vide ?
 - (a) si non, il faut retourner à l'étape 4 ;
 - (b) si oui, FIN

2.4 Exemple

Nous allons présenter l'encodage d'un flux de caractères. La colonne **Étape** indique le numéro de l'étape d'encodage. Une étape est terminée après l'exécution de l'étape 3.b de l'algorithme. La colonne **Pos.** indique la position courante dans le flux d'entrée. La colonne **Dico.** montre les chaînes qui viennent d'être ajoutées au dictionnaire et son adresse entre parenthèses. La colonne **Sortie** contient les mots-codes qui sont ajoutés au flux de sortie.

La chaîne à encoder est : ABCD_⊔AB_⊔CD_⊔EFFEFEFEG. Le dictionnaire contient au début de l'encodage chaque caractère présent dans la chaîne précédente, *i.e.* _⊔ABCDEFG.

Étape	Pos.	Dico.	Sortie
1	1	(8) AB	(1)
2	2	(9) BC	(2)
3	3	(10) CD	(3)
4	4	(11) D _⊔	(4)
5	5	(12) _⊔ A	(0)
6	6	(13) AB _⊔	(8)
7	8	(14) AB _⊔ C	(0)
8	9	(15) CD _⊔	(10)
9	11	(16) CD _⊔ E	(0)
10	12	(17) EF	(5)
11	13	(18) EFF	(17)
12	14	(19) EFFE	(6)
13	15	(20) FE	(5)
14	16	(21) FEF	(20)
15	18	(22) FEFE	(20)
16	20	(23) FEFEG	(7)

Dictionnaire	
(0) _⊔	(1) A
(2) B	(3) C
(4) D	(5) E
(6) F	(7) G
(8) AB	(9) BC
(10) CD	(11) D _⊔
(12) _⊔ A	(13) AB _⊔
(14) AB _⊔ C	(15) CD _⊔
(16) CD _⊔ E	(17) EF
(18) EFF	(19) EFFE
(20) FE	(21) FEF
(22) FEFE	(23) FEFEG

TAB. 1 – Exemple de compression d'une chaîne de caractères par l'algorithme LZW

La chaîne initiale comptait 20 caractères codés sur 8 bits, soit 160 bits à transmettre ou à stocker. Après compression, on obtient un flux codé contenant 16 adresses codées sur 9 bits, soit 144 bits seulement. La chaîne initiale a bien été compressée, même si elle comportait peu de redondance.

3 Méthode de compression JPEG

3.1 Présentation

La méthode de compression JPEG, sur laquelle est basée le format homonyme, est l'une des méthodes les plus connues et les plus utilisées, surtout pour réduire la taille des photos. Les idées de cette méthode sont également à la base des méthodes de compression de vidéo grâce au format MPEG. Le cœur de la méthode est d'utiliser la transformée de Fourier des données à compresser : au lieu de compresser les données en espace, nous allons travailler sur les fréquences spatiales associées à ces données.

3.2 Schéma de la méthode

L'image originale est représentée par trois tableaux de nombres pour les trois composantes RVB. On transforme alors cette représentation RVB dans la base HSB (*Hue Saturation Brightness* ou Teinte, Saturation et Brilliance ou plus simplement Chrominance et Luminance). Les valeurs entières sont d'abord signées : on retranche 2^{p-1} si p représente le nombre de bits pour coder une composante. Par exemple, pour des couleurs codées sur 24 bits (un octet par composante), on soustrait à chaque valeur $128 = 2^7$.

La méthode JPEG se décompose alors en cinq étapes depuis la donnée de la matrice initiale :

1. Transformation de Fourier ;
2. Quantification à l'aide de tables ;
3. Codage différentiel ;
4. Codage des longueurs de séquence (RLE ou *Run Length Encoding*) en zigzag ;
5. Codage VLE (*Variable Length Encoding*) ou entropique.

Dans l'ensemble des étapes de la méthode, seule l'étape de quantification est non conservative. Les trois dernières étapes réalisent une compression conservative.

3.2.1 Transformation de Fourier

Pour calculer la transformée de Fourier, on découpe l'image en bloc de 8×8 . En réalité, nous allons utiliser la transformée de Fourier discrète en cosinus (DCT) au lieu d'une transformée de Fourier discrète classique. Ce choix a l'avantage de nécessiter moins de calcul, en particulier des calculs complexes. Il vient alors, pour chaque bloc de l'image :

$$X(m, n) = \frac{C(m)C(n)}{\sqrt{2N}} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} x(i, j) \cos \frac{(2i+1)\pi m}{2N} \cos \frac{(2j+1)\pi n}{2N}$$

où $N = 8$ est la taille des blocs et les coefficients C sont définis comme suit :

$$\forall p, \quad C(p) = \begin{cases} \frac{1}{\sqrt{2}} & \text{si } p = 0 \\ 1 & \text{si } p > 0 \end{cases}$$

3.2.2 Quantification

La quantification se fait en effectuant la division entière de chaque valeur du tableau obtenu après le calcul de la DCT par une valeur (un *quantum*) dépendant à la fois de la position dans la matrice et du type de données de la matrice (luminance ou chrominance). En effet, une fois la DCT

calculée, la valeur en haut à gauche de la matrice est la moyenne des valeurs de la matrice précédente : plus l'on s'éloigne de cette valeur, plus on considère « des hautes fréquences ». Les hautes fréquences spatiales peuvent être « coupées » sans perte de qualité visible sur le résultat. D'autre part, l'œil humain est plus sensible aux informations d'intensité (la luminance) qu'aux différences de couleur (la chrominance). Il est donc possible de prendre pour les valeurs de chrominance des *quanta* supérieurs à ceux choisis pour la luminance. Par exemple, on pourra utiliser les tableaux suivants comme matrice de quantification :

Luminance								Chrominance							
16	11	10	16	24	40	51	61	17	18	24	47	99	99	99	99
12	12	14	19	26	58	60	55	18	21	26	66	99	99	99	99
14	13	16	24	40	57	69	56	24	26	56	99	99	99	99	99
14	17	22	29	51	87	80	62	47	66	99	99	99	99	99	99
18	22	37	56	68	109	103	77	99	99	99	99	99	99	99	99
24	35	55	64	81	104	113	92	99	99	99	99	99	99	99	99
49	64	78	87	103	121	120	101	99	99	99	99	99	99	99	99
72	92	95	98	112	100	103	99	99	99	99	99	99	99	99	99

TAB. 2 – Exemples de matrices de quantification

Il est bon d'observer que, étant donné que l'étape de quantification est la seule qui est non conservative, la qualité du résultat et sa taille dépendent beaucoup de ces matrices. Pour obtenir une meilleure qualité au détriment de la taille, il faut choisir un pas de quantification plus petit. Inversement, si l'on souhaite réduire la taille en diminuant la qualité de l'image, il faut augmenter le pas de quantification. Dans l'exemple précédent, la qualité obtenue est acceptable avec une réduction importante de la taille originale de l'image.

Il est également possible de donner une formule permettant de donner la valeur du *quantum* en fonction des indices (m, n) dans la matrice et de trois paramètres :

$$q(m, n) = 1 + F_q (1 + \alpha(m^r + n^r))$$

Un choix possible pour les valeurs des paramètres est : $\alpha = 1$, $r = 1$ et $F_q = 5$. On peut alors obtenir le résultat souhaité (haute qualité ou forte compression) en faisant varier, par exemple, la valeur de F_q (pour facteur de qualité).

3.2.3 Codage différentiel

Les valeurs en haut à droite des blocs², proportionnelles à la valeur moyenne des pixels du bloc, vont être codées à l'aide d'un codage différentiel.

Chacune de ses valeurs $(DC_i)_i$ sera remplacée par sa différence avec la valeur précédente, avec comme valeur de départ (DC_{-1}) prise égale à zéro :

$$\delta_i = DC_i - DC_{i-1}$$

Comme les valeurs $(DC_i)_i$ sont proportionnelles à la valeur moyenne des pixels de chaque bloc et que cette valeur moyenne (dans une image) varie peu d'un bloc à l'autre, il est avantageux de compresser les valeurs δ_i obtenues compressées par codage entropique de Huffman : les valeurs seront réparties en douze catégories notées S , les catégories représentant de fortes différences étant codées sur plus de bits (cf. tableau 3). La règle de regroupements des valeurs est la suivante :

$$\begin{aligned} \delta_i = 0 &\Rightarrow S = 0 \\ 2^{j-1} \leq |\delta_i| < 2^j - 1 &\Rightarrow S = j \end{aligned}$$

2. aussi appelées coefficients DC

Catégorie S	Luminance		Chrominance	
	Longueur	Code	Longueur	Code
0	2	00	2	00
1	3	010	2	01
2	3	011	2	10
3	3	100	3	110
4	3	101	4	1110
5	3	110	5	11110
6	4	1110	6	111110
7	5	11110	7	1111110
8	6	111110	8	11111110
9	7	1111110	9	111111110
10	8	11111110	10	1111111110
11	9	111111110	11	11111111110

TAB. 3 – Définition des catégories pour le codage différentiel

3.2.4 Codage RLE en zigzag

L'étape suivante consiste à coder les autres valeurs (notées AC). Pour cela, on lit les coefficients de la matrice *en zig-zag* à partir de la valeur DC . Cette méthode permet de faire apparaître de longues suites de zéros (dans les fréquences hautes), ce qui convient très bien à la méthode RLE qui code ces séquences par le caractère répété et leur longueur.

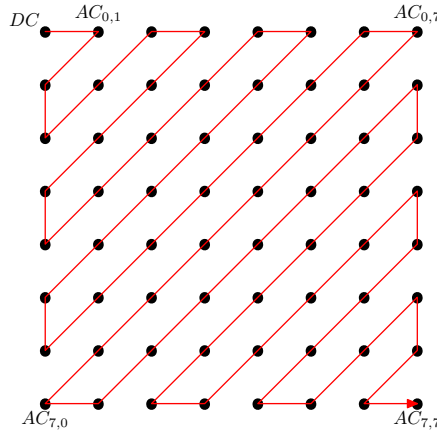


FIG. 1 – Méthode de lecture en zig-zag

3.2.5 Codage VLC ou entropique

La suite de valeurs obtenue précédemment pour chaque bloc est codé grâce à un algorithme de compression entropique (méthode d'Huffman ou de Shannon–Fano), utilisant des codes de longueur variable en fonction de la fréquence d'apparition des différents caractères.

3.3 Exemples

3.3.1 Photographie

Nous avons choisi une image très souvent utilisée dans les tests sur les méthodes de codage d'images. Cette image est accompagnée d'une histoire assez particulière qui lui confère³ une touche d'originalité plus importante encore...

3. Vous pouvez la consulter ici : <http://www-2.cs.cmu.edu/~chuck/lennapg/lenna.shtml>

Si l'on choisit différents taux de compression des images de sortie, on peut comparer la qualité de l'image restituée après codage. La comparaison suivante donne les différents taux de compression et taille obtenue après encodage. On peut remarquer facilement que les faibles taux de compression ne dégradent pas visiblement l'image mais que la qualité se dégrade avec l'augmentation du taux de compression : les transitions marquées de teinte s'estompent (coupure des « hautes fréquences »), les blocs sont de plus en plus visibles, de couleur uniforme égale à la couleur moyenne du bloc (quantification trop importante). Pour un taux de compression de 75%, l'image est presque identique à l'originale. Pour des taux plus faibles (50% et 25%), la dégradation est visible mais peut être tolérée dans le cas d'une image diffusée sur Internet. Les taux encore plus faibles rendent l'image inutilisable, sauf pour des effets voulus (effet « mosaïque » pour masquer les visages par exemple).



FIG. 2 – *Lena* (256×256 256 niveaux de gris, 65 ko)



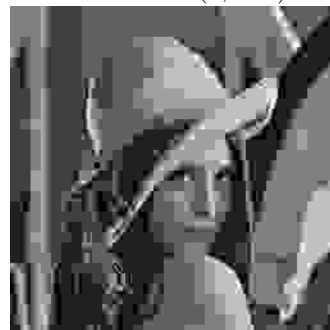
Taux : 75% (11 ko)



Taux : 50% (7,3 ko)



Taux : 25% (4,8 ko)



Taux : 0% (870 o)

FIG. 3 – *Exemples de résultats obtenus par compression JPEG*

3.3.2 Image de synthèse

Considérons une image de synthèse 64×64 codée sur 64 niveaux de gris. La couleur en chaque point de l'image sera donnée par la formule suivante :

$$x(k, l) = 32 \exp \left(-\frac{(k-32)^2 + (l-32)^2}{128} \right) + p(k, l) + \Psi(k, l)$$

où : $p(k, l)$ est le plateau : $p(k, l) = \begin{cases} 10 & \text{si } 0 \leq k \leq 31 \text{ et } 0 \leq l \leq 31 \\ 0 & \text{sinon} \end{cases}$;

$$\Psi(k, l) = \sum_{r=1}^3 2^{4-r} \sin \left[\frac{(k-32)(l-32)}{2^{4-r}} \right].$$

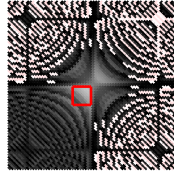


FIG. 4 – Exemple d'image de synthèse

Pour illustrer la méthode JPEG, nous allons effectuer chacune des étapes décrites précédemment sur le bloc 8×8 encadré. Il correspond au tableau de valeurs suivant :

30	33	18	18	24	27	35	38
33	21	18	21	31	29	39	41
18	18	21	30	30	36	41	44
18	21	30	32	33	42	44	46
24	31	30	33	41	44	46	47
27	29	36	42	44	46	49	47
35	39	41	44	46	49	49	46
38	41	44	46	47	47	46	44

La première étape consiste à signer chacune des valeurs. Comme notre image est codée sur 64 niveaux de gris, il faut retrancher 32 à chaque valeur :

-2	1	-14	-14	-8	-5	3	6
1	-11	-14	-11	-1	-3	7	9
-14	-14	-11	-2	-2	4	9	12
-14	-11	-2	0	1	10	12	14
-8	-1	-2	1	9	12	14	15
-5	-3	4	10	12	14	17	15
3	7	9	12	14	17	17	14
6	9	12	14	15	15	14	12

On peut alors calculer les coefficients de la DCT grâce à la formule donnée précédemment :

29	-49	4	2	0	0	-3	0
-49	-4	20	3	4	-1	-3	1
4	20	9	4	2	-2	-4	1
2	3	4	2	-2	-4	-3	1
0	4	2	-2	-5	-5	0	-1
0	-1	-2	-4	-5	0	1	-3
-3	-3	-4	-3	0	1	-1	-3
0	1	1	1	-1	-3	-3	-2

En utilisant la formule donnant les coefficients de quantification ($q(m, n) = 6 + 5(m + n)$), on peut calculer les valeurs quantifiées qui seront ensuite codées comme nous l'avons exposé plus haut.

On peut facilement observer que le tableau de valeurs obtenu est très « creux ». La compression se révélera d'autant plus efficace.

5	-4	0	0	0	0	0	0
-4	0	1	0	0	0	0	0
0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Pour décoder l'image, on réalise les étapes dans l'ordre inverse. Après avoir décodé le fichier, on obtient les blocs de valeurs quantifiées. On multiplie ces valeurs par les coefficients de quantification pour obtenir les valeurs non quantifiées :

30	-44	0	0	0	0	0	0
-44	0	21	0	0	0	0	0
0	21	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

En appliquant une DCT inverse (donnée par la formule suivante) et en ajoutant 32 à chacun des coefficients, on obtient l'image reconstituée :

$$x(i, j) = \frac{1}{\sqrt{2N}} \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} C(m)C(n) X(m, n) \cos \frac{(2i+1)\pi m}{2N} \cos \frac{(2j+1)\pi n}{2N}$$

30	28	25	23	24	28	33	36
28	26	24	24	26	31	36	39
25	24	25	27	31	36	41	44
23	24	27	31	36	41	45	48
24	26	31	36	41	45	47	49
28	31	36	41	45	47	47	47
33	36	41	45	47	47	45	44
36	39	44	48	49	47	44	41

Les valeurs sont, on peut le constater, très proches des valeurs originales : la différence maximale est de 7 sur quelques pixels, la moyenne des différences étant égale 2,22. On a donc finalement peu dégradé l'image originale alors que la quantification était importante : avec un facteur de qualité plus grand ($F_q = 2$), le résultat obtenu est encore plus proche de l'original.