Chapitre 1

Codage et Compression de données

Avant d'introduire la notion de compression, nous allons survoler quelques codes relativement courants. La définition d'un code est un symbole remplaçant un autre. Dans la cadre informatique, il s'agira d'une suite binaire qui remplacera un symbole donné (souvent un caractère).

1.1 Les tables de caractères

1.1.1 ASCII

Le code ASCII (American Standard Code for Information Interchange) est un code de 7 bits (de données) et d'un bit de parité par symbole. Il date de 1961, et fut créé par Bob Bemer. 128 ($= 2^7$) symboles peuvent donc être codés. On y retrouve les lettres majuscules et minuscules, les 10 chiffres, quelques symboles de ponctuation et des caractères de contrôle.

	0	1	2	3	4	5	6	7
0	NUL	DLE	space	0	@	P	`	p
1	SOH	DC1 XON	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3 XOFF	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	del

FIG. 1.1 – Table ASCII basique

On entend régulièrement parler du code ASCII étendu (*extended ASCII*). Ce code n'est en réalité pas clairement déterminé, en ce sens qu'il n'est pas unique, plusieurs variantes non compatibles ayant été publiées. Les plus courants sont le code ASCII étendu OEM, et le code

ASCII étendu ANSI. L'objectif principal était de permettre l'utilisation des caractères accentués, absents de la table ASCII basique. Chaque caractère y est représenté par une chaine de 8 bits.

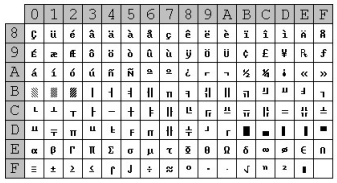


FIG. 1.2 – Table ASCII étendue OEM

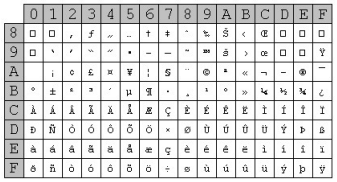


FIG. 1.3 – Table ASCII étendue ANSI

1.1.2 EBCDIC

Il s'agit d'un mode de codage des caractères sur 8 bits créé par IBM à l'époque des cartes perforées. De nombreuses versions existent, bien souvent incompatibles entre elles, ce qui est en partie la cause de son abandon. On l'utilise encore de nos jours, mais presque essentiellement pour des raisons de rétro-compatibilité.

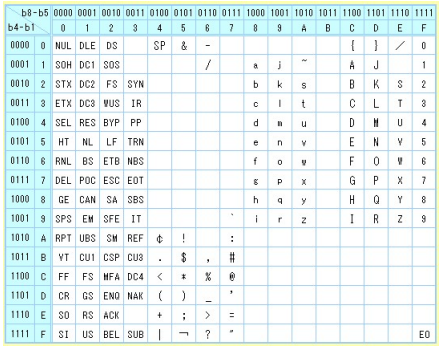
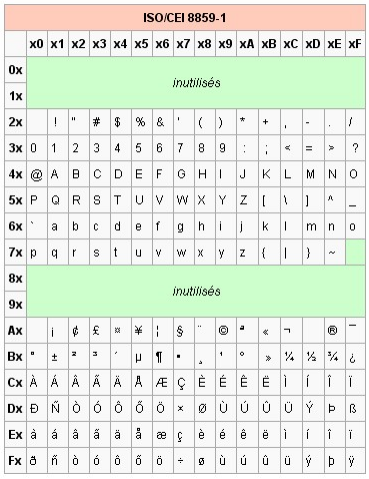


FIG. 1.4 – Table EBCDIC

1.1.3 ISO 8859-1 (et -15)

C'est une norme de l'International Organization for Standardization, souvent dénommée Latin-1. Elle se compose de 191 caractères codés sur un octet. Cette norme recouvre les caractères plus utilisés dans les langues européennes et dans une partie des langues africaines.

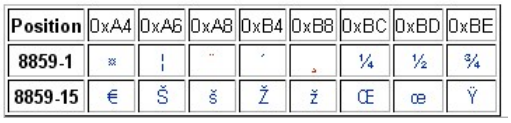


ISO/CEI 8859-1															
	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xA	xB	xC	xD	xE	xF
0x	*inutilisés*															
1x																
2x		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3x	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4x	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5x	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6x	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7x	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	
8x	*inutilisés*															
9x																
Ax		¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬		®	¯
Bx	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
Cx	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
Dx	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
Ex	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
Fx	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

FIG. 1.5 – Table ISO 8859-1

Comme on le voit sur la figure 1.5, certaines plages ne sont pas utilisées : les codes 0x00 à 0x1F et 0x7F à 0x9F ne sont pas attribuées.

Pour permettre l'utilisation du sigle €, il a fallu mettre cette table à jour. La norme ISO 8859-15 est alors apparue. Elle se compose des mêmes éléments que la norme ISO 8859-1, mis à part 8 caractères, illustrés à la figure 1.6.



Position	0xA4	0xA6	0xA8	0xB4	0xB8	0xBC	0xBD	0xBE
8859-1	¤	¦	¨	´	¸	¼	½	¾
8859-15	€	Š	š	Ž	ž	Œ	œ	Ÿ

FIG. 1.6 – Différences entre les normes 8859-1 et 8859-15

1.1.4 Unicode

Ce projet fut initié en 1987 par Becker, Collins et Davis. L'objectif était alors de construire un code universel sur 16 bits, permettant donc 65,536 (2^{16}) codes différents. En 1991 est fondé le consortium Unicode, comprenant des linguistes et autres spécialistes de nombreux horizons. La norme Unicode est, en plus d'un standard de codage de caractères, un immense rapport des

recherches mondiales sur les langues utilisées et ayant été utilisées à travers le monde.

Unicode en est actuellement à la version 5.0, le nombre de caractères identifiées étant sans cesse croissant. La version 3.0 (1999) comptait déjà 49.194 caractères, la limite des 2^{16} étant déjà bien proche. En 2001, la version 3.2 ajouta un peu moins de 45.000 caractères. L'espace des codes Unicode s'étend de 0 à 0x10FFFF. Un code sur 16 bits n'est donc plus suffisant. Le codage sur 4 octets permet la représentation de plus de 4 milliards de caractères (2^{32}).

On y retrouve le code ASCII, auquel on a notamment ajouté les jeux complets de caractères coréens, japonais et chinois, mais aussi les symboles mathématiques ou encore les écritures anciennes telles que l'écriture cunéiforme ou le linéaire B.

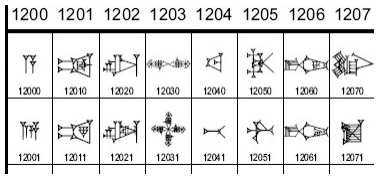


FIG. 1.7 – Ecriture cunéiforme dans une table UNICODE

Plusieurs formats d'encodage sont possibles (UTF - Universal Transformation Format) :
- UTF-8 : 8 bits
- UTF-16 : 16 bits
- UTF-32 : 32 bits

Le standard Unicode définit les caractères comme une série de points de code (*codepoints*). Selon le format utilisé, sa représentation sera différente.

UTF-8

Les codepoints encodés sous ce format se présentent comme un octet ou une suite d'octets. Lorsque le caractère se situe dans la plage 0x00 à 0x7F, il s'agit du code ASCII, ce qui ne pose pas de problème pour le représenter sur un seul octet. Pour les autres valeurs, un premier octet (*lead byte*) suivi d'un nombre variable d'octets (*trailing byte*)(maximum 4 octets au total) représentent conjointement la valeur à encoder.

Dans le cas où on utilise un *lead* byte, celui devra toujours avoir son bit de poids fort à 1, et sera suivi d'autant de bit à 1 que de *trailing* byte.

Format du premier octet	Nombre d'octets utilisés au total
110.....	2
1110....	3
1111....	4

Les bits remplacés par '-' dans le tableau ci-dessus représente les bits disponibles pour la valeur Unicode à encoder. Toutefois, chaque *trailing* byte doit avoir ses deux premiers bits à 10. Ainsi, chaque *trailing* byte possédera 6 bits d'information.

Formats possibles	Nombres de bits qu'il est possible d'encoder
110..... 10.....	8 à 11 bits disponibles
1110.... 10..... 10.....	12 à 16 bits disponibles
1111.... 10..... 10..... 10.....	17 à 21 bits disponibles

La limite de 21 bits est suffisante pour représenter l'ensemble des codepoints définis par Unicode.

Par cette méthode d'encodage, un même caractère peut avoir plusieurs représentations. Par exemple, le caractère '/' peut s'écrire
- 0x2F
- 0xC0 0xAF
- 0xE0 0x80 0xAF
- 0xF0 0x80 0x80 0xAF

Depuis la troisième version d'Unicode, il a été spécifié que seul l'encodage le plus court était accepté.

UTF-16

Les codepoints sont ici exprimés sous la forme de mots de 16 bits. La plupart des caractères courants sont représentables sur 16 bits, mais certains ne le sont toutefois pas (la plage adressable Unicode étant étendue jusque 0x10FFFF).

Similairement à l'UTF-8, il est possible d'utiliser plusieurs mots (maximum 2) de 16 bits afin d'encoder des valeurs nécessitant plus de 16 bits.

Afin de déterminer les mots de 16 bits, on a recours au calcul suivant (U représente la valeur Unicode) :

1. Si U $<$ 0x10000, U est encodé comme un entier non signé sur 16 bits.
2. Soit U' = U - 0x10000. U étant inférieur ou égal à 0x10FFFF (par définition de la plage de validité du standard Unicode), U' sera inférieur ou égal à 0xFFFFF. 20 bits sont donc nécessaires.
3. Initialiser 2 entiers non signés de 16 bits, W1 et W2, aux valeurs 0xD800 et 0xDC00. Chaque entier possédera 10 bits libres, qui fourniront au total les 20 bits nécessaires.
4. Associer les 10 bits de poids forts de U' aux bits libres de W1, et les 10 autres bits de U' (les 10 bits de poids faibles de U') à W2.

Il n'y ici qu'une seule manière de représenter un codepoint donné.

UTF-32

Les codepoints sont ici représentés par une valeur sur 32 bits. Cette taille est suffisante pour représenter tous les codepoints Unicode existants.

1.1.5 Base64

Largement utilisé pour la transmission de messages, le codage Base64 utilise, comme son nom le laisse présager, un alphabet de 64 caractères. Un 65ème caractère existe, mais ne fait pas partie de l'alphabet à proprement parlé. Il est utilisé comme caractère final ('=').

6-bit value	character encoding	6-bit value	character encoding	6-bit value	character encoding	6-bit value	character encoding
0	A	16	Q	32	g	48	w
1	B	17	R	33	h	49	x
2	C	18	S	34	i	50	y
3	D	19	T	35	j	51	z
4	E	20	U	36	k	52	0
5	F	21	V	37	l	53	1
6	G	22	W	38	m	54	2
7	H	23	X	39	n	55	3
8	I	24	Y	40	o	56	4
9	J	25	Z	41	p	57	5
10	K	26	a	42	q	58	6
11	L	27	b	43	r	59	7
12	M	28	c	44	s	60	8
13	N	29	d	45	t	61	9
14	O	30	e	46	u	62	+
15	P	31	f	47	v	63	/
						(pad)	=

FIG. 1.8 – Table BASE64

Le traitement s'opère par groupe de 24 bits (3 octets) et produit en sortie un bloc de 4 caractères (4*6 bits). Chaque groupe de 6 bits sert d'index dans l'alphabet Base64, et le caractère associé sera placé sur le flux de sortie.

Lorsque le nombre de bits en entrée devient insuffisant (le flux d'entrée est composé de moins de 24 bits), seules deux possibilités existent (en sachant que l'entrée ne peut être constituée que d'octets entiers) :

- l'entrée fournit 8 bits (1 octet) : le bloc de 4 caractères en sortie sera composé de 2 caractères de la table base64 suivis de 2 caractères finaux (c'est-à-dire le 65ème caractère).
- l'entrée fournit 16 bits (2 octets) : le bloc de 4 caractères en sortie sera composé de 3 caractères de la table base64 suivis de 1 caractère final.

Exemple Le mot "oui" est traduit sous forme binaire en '01101111 01110101 01101001'. Si on découpe ces 24 bits en blocs de 6 bits, il vient '011011 110111 010101 101001'. Sous forme décimale, on obtient '27 55 21 41', et en substituant, il vient 'b3Vp'.

Lorsqu'on utilise cet encodage dans le logiciel de chiffrement de messages PGP, le résultat est ensuite encodé au format ASCII.

Retour à l'exemple Ainsi, si on met le bit de parité à 0, la correspondance de 'b3Vp' est '01100010 00110011 01010110 01110000'.

1.2 La compression de données

La compression a pour but de réduire la longueur d'une chaîne sans affecter son contenu informatif. Cela permet à la fois de réduire les exigences en mémoire et d'augmenter la capacité d'un canal de transmission (théorie de Shannon).

L'information d'un message peut se définir comme la "surprise" causée par la connaissance de ce message. Elle se calcule par la formule

$$-\log_2 p$$

si p est la probabilité d'occurrence du message.

Soit l'ensemble X composé de N messages dont les probabilités d'occurrence sont données par $p_1, ..., p_N$. Alors

$$\sum_{i=1}^{N} p_i = 1$$

L'information H associée à ces N messages est définie comme la « surprise moyenne » :

$$H(x) = -\sum_{i=1}^{N} p_i \log p_i$$

H(x) permet de mesurer l'information. On lui donne le nom d'**Entropie**.

$$0 \leq H < \infty$$

et est maximale quand $p_1 = p_2 = ... = p_N = \frac{1}{N}$

Indépendance des messages

Soit un ensemble C de messages égal au produit de 2 ensembles A et B indépendants. Alors

$$H(C) = H(A) + H(B)$$

En effet,

$$\begin{aligned}
\sum_i \sum_j p_i q_j \log p_i q_j &= \sum_i p_i \left[\sum_j q_j (\log p_i + \log q_j) \right] \\
&= \sum_i p_i \left[\log p_i + \sum_j q_j \log q_j \right] \\
&= \sum_i p_i \log p_i + \sum_j q_j \log q_j
\end{aligned}$$

Longueur moyenne d'un code

Soit un codage des messages à partir d'un alphabet de d caractères tel qu'il existe une correspondance non ambiguë entre chaque message et son code. Si le message i est représenté par une séquence de l_i caractères, la longueur moyenne d'un code est donnée par la formule suivante :

$$\overline{L} = \sum_{i=1}^{N} p_i l_i$$

Efficacité et redondance d'un système de codage

Par le théorème du codage de Shannon, on peut définir l'efficacité d'un système de codage. En effet, par Shannon,

$$\bar{L} \geqslant \frac{H(X)}{\log d}$$

et donc

$$\frac{H(X)}{\bar{L} \log d} \leqslant 1$$

qui est la formule définissant l'efficacité d'un système de codage.

La redondance est donnée par $R = 1 -$ efficacité.

Il existe un codage dont l'efficacité tend vers 1 ainsi qu'une méthode de construction.

1.2.1 Codage de Huffman

Supposons avoir 8 messages $m_1, ..., m_8$ dont la probabilité d'occurrence est donnée par $p_1 = p_2 = ... = p_8 = 1/8$. Il vient

$$H(X) = -\sum_{1}^{8} p_i \log p_i = 8.\frac{1}{8}.3 = 3$$

Si l'alphabet est équivalent à {0,1}, alors d=2 et

$$\bar{L} \geqslant \frac{3}{1} = 3$$

Donc, le codage à 3 bits [000,001,...,111] est efficace[1].

Quand d=2 (et donc que $\log_2 d = 1$), H(X) donne
– la longueur en bits pour un codage non-redondant, ou encore
– le nombre de choix binaires à faire en moyenne pour identifier un message.

Si on dispose d'une autre distribution P telle que

$$\{p_i'\} = \left\{\frac{1}{2}, \left(\frac{1}{2}\right)^2, \left(\frac{1}{2}\right)^3, \left(\frac{1}{2}\right)^4, \left(\frac{1}{2}\right)^5, \left(\frac{1}{2}\right)^6, \left(\frac{1}{2}\right)^7, \left(\frac{1}{2}\right)^7\right\}$$

	p	codage eff pour p	p'_i	codage eff pour $\{p'_i\}$
m_1=a	$\left(\frac{1}{2}\right)^3$	000	$\frac{1}{2}$	0
m_2=b	$\left(\frac{1}{2}\right)^3$	001	$\left(\frac{1}{2}\right)^2$	10
m_3=c	$\left(\frac{1}{2}\right)^3$	010	$\left(\frac{1}{2}\right)^3$	110
m_4=d	$\left(\frac{1}{2}\right)^3$	011	$\left(\frac{1}{2}\right)^4$	1110
m_5=e	$\left(\frac{1}{2}\right)^3$	100	$\left(\frac{1}{2}\right)^5$	11110
m_6=f	$\left(\frac{1}{2}\right)^3$	101	$\left(\frac{1}{2}\right)^6$	111110
m_7=g	$\left(\frac{1}{2}\right)^3$	110	$\left(\frac{1}{2}\right)^7$	1111110
m_8=h	$\left(\frac{1}{2}\right)^3$	111	$\left(\frac{1}{2}\right)^7$	1111111

FIG. 1.9 – Codage de Huffman

alors $H(X) = \frac{127}{64}$ et on obtient un codage non-redondant où L est inférieure à 2 bits !

Ce codage a en plus le mérite de permettre le décodage instantané (caractère par caractère, sans attendre une séquence entière). On parle de propriété **préfixe** : (aucun code n'est préfixe d'un autre code).

Exemple : 111100111111110 est décodé directement en *eahb*, à partir du codage de la figure 1.9.

Algorithme du codage de Huffman

1. Les messages constituent les feuilles d'un arbre portant chacune un poids égal à la probabilité P d'occurrence du message correspondant
2. Joindre les 2 noeuds de moindre poids en un noeud parent auquel on attache un poids égal à la somme de ces 2 poids
3. Répéter le point 2 jusqu'à l'obtention d'une seule racine à l'arbre (de poids $\sum p_i = 1$)
4. Affecter les codes 0 et 1 aux noeuds descendants directs de la racine
5. Continuer à descendre en affectant des codes à tous les noeuds, chaque paire de descendants recevant les codes L0 et L1 où L désigne le code associé au parent.

Par exemple, soit un ensemble de 3 messages a, b et c de probabilité respective 0.6, 0.3 et 0.1. La construction de l'algorithme est donnée à la figure 1.10.

[1] On dit également "non-redondant".

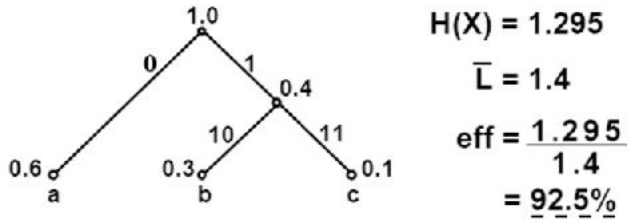


FIG. 1.10 – Codage de Huffman

En codant des séquences de plus en plus longues, l'efficacité tend vers 100% (mais le gain est de moins en moins important, comme on le voit sur la figure 1.11).

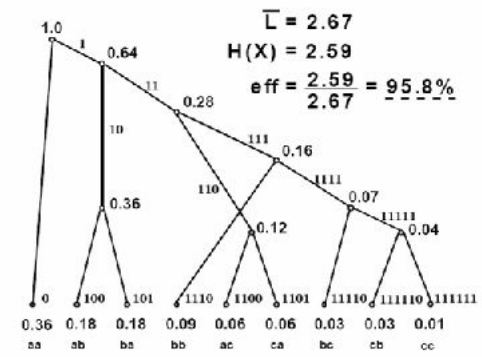


FIG. 1.11 – Codage de Huffman

En résumé, l'algorithme de HUFFMAN donne un codage optimal (car la redondance est minimale) et possède la propriété préfixe (ce qui est intéresssant en cas de transmission sur un canal).

1.2.2 Codage de Shannon-Fano

Ce procédé est antérieur au codage de Huffman est se base également sur un codage statistique.

Algorithme

1. Construire une table des fréquences d'apparition des symboles triée par ordre décroissant.
2. Diviser cette table en deux parties. Celles-ci doivent avoir une somme de fréquences égale (ou pratiquement égale) à celle de l'autre.

3. Affecter le chiffre binaire 0 à la moitié inférieure, la moitié supérieure prenant la valeur 1.
4. Répéter les opérations 2 et 3 aux deux parties, jusqu'à ce que chaque symbole ne représente plus qu'une partie de la table.

	Prob.					Etat final
1.	0.25	1	1			11
2.	0.20	1	0			10
3.	0.15	0	1	1		011
4.	0.15	0	1	0		010
5.	0.10	0	0		1	001
6.	0.10	0	0		0	1 0001
7.	0.05	0	0		0	0 0000

FIG. 1.12 – Codage de Shannon-Fano

Le codage de HUFFMAN est efficace quand il y a un petit nombre de types de messages dont quelques-uns couvrent une proportion importante du texte ($p_i >> p_j$). Quand le nombre de types de messages augmente et que les fréquences sont plus uniformes ($p_i \simeq p_j \forall i \forall j$), le gain est négligeable.

Souvent, le nombre de messages différents est indéterminé à l'avance (par exemple quand il s'agit de mots dans un texte simple) ou trop grand pour justifier un codage à longueur variable (par exemple, le jeu de caractères ASCII). Dans ces cas, on effectue une compression en remplaçant seulement des séquences choisies de texte par des codes plus courts.

Il y a plusieurs façons de procéder :
- parcourir le texte et remplacer les séquences redondantes par des séquences plus courtes (RLE)
- analyser préalablement le texte pour déterminer les groupes à remplacer et les codes qui les remplacent (LZW)
- ...

1.2.3 Codage RLE (Variantes)

Le codage RLE (Run-Length Encoding) est, comme son nom l'indique, un codage de "course", c'est-à-dire qu'il élimine certaines séquences de caractères en les remplaçant par un code spécifique. Chaque "course" de k éléments ($2 < k \leq 9$) est remplacée par un caractère non utilisé (ex : @) suivi de l'entier k et du caractère substitué.

Exemple 1 "AB200003944445260D666@A2" est remplacé par "AB2@40394@445260D@36@@A2".

Si tous les caractères sont utilisés, on en choisit un rarement utilisé comme "indicateur" et on le double quand on rencontre ce caractère rare. Ce principe est illustré dans l'exemple ci-dessus.

On remarque également que dans le cas précis où on se met d'accord pour ne remplacer que les courses d'un caractère donné, il n'est pas nécessaire de fournir le caractère remplacé. Ce type d'utilisation est fréquent dans le cas des transmissions d'image en noir et blanc (type fax).

Exemple 2 "AB20000394000005260D000A2" est remplacé par "AB2@4394@55260D@3A2".

Une condition est nécessaire pour que cette compression soit utile : la séquence répétée doit contenir au moins 4 éléments pour obtenir un gain (3, dans le cas où on est d'accord sur le caractère remplacé).

1.2.4 Utilisation d'une bit-map

Il est facile de laisser tomber des champs entiers d'un enregistrement. Prenons un enregistrement de 4 champs de 8 caractères (32 caractères) pour les noms alors que la longueur moyenne est bien plus courte. On utilise alors la technique du bit-mapping.

Le principe est d'utiliser une bit-map devant chaque enregistrement pour indiquer la présence ou l'absence de valeur dans les différents champs.

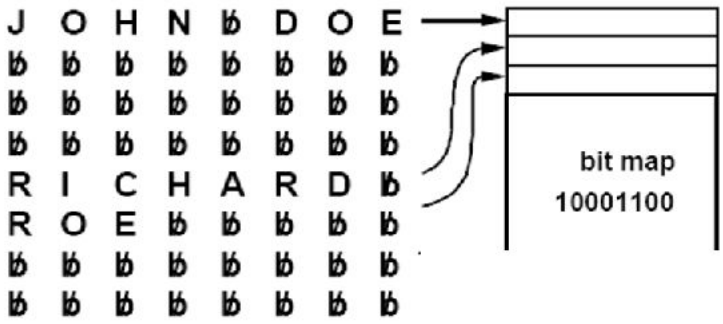


FIG. 1.13 – Utilisation d'une bit-map

1.2.5 Recherche de pattern

Par une étude sur des textes anglais, on a pu déterminer que le nombre de mots distincts utilisés représentait 10 à 15% du nombre total de mots dans le texte. Ce pourcentage est encore plus petit pour les textes liés à des domaines spécialisés (articles scientifiques, documents militaires, médicaux, ...). Dès lors, on pourrait remplacer des mots ou «patterns» préalablement choisis.

Cependant, le gain est limité par 2 observations :
- les mots les plus courants sont courts (the, a, to, in, ...),
- par loi de ZIPF : la fréquence du n^{ieme} mot le plus courant est proportionnelle à 1/n, c'est-à-dire que le gain augmente de plus en plus lentement quand on code de plus en plus de mots.

Il existe tout de même quelques pistes de solution :

- on peut coder des "digrammes", c'est-à-dire remplacer des paires de lettres par un seul caractère. Ainsi, on pourrait remplacer les paires "ES", "TR", "LL",... par un caractère propre. On obtient dans ce cas une compression d'environ 50% en moyenne.

- on peut aussi coder les "patterns" les plus courants : cette technique s'utilise lorsque des mots apparaissent fréquemment (ex : vocabulaire informatique) mais
  1. l'analyse préalable pour les déterminer est coûteuse :
     - il faut tenir compte d'un nombre élevé de candidats, et
     - le choix d'un pattern peut affecter l'utilité des précédents. Par exemple, si la séquence 'ere' est choisie puis également 'here', 'there' et 'where', si on commence par remplacer les séquences les plus longues, le codage de 'ere' devient beaucoup moins utile.
  2. cela peut gêner la recherche :
     Par exemple, si 'ing l' est codé par '#' et 'ing t' par '@', et si 'string' est codé par 'str#' dans le contexte 'string lists' et par 'str@' dans le contexte 'string trees'. La recherche de 'string' conduit à analyser plus d'un pattern.

1.2.6 Les algorithmes à dictionnaire (ou à substitution de facteurs)

Ils consistent à remplacer des séquences (les facteurs) par un code plus court qui est l'indice de ce facteur dans un dictionnaire.

LZ77 (ou LZ1)

Dans cette approche, le dictionnaire n'est qu'une portion du texte encodé précédemment. L'encodeur examine le texte à compresser par l'intermédiaire d'une fenêtre glissante (*sliding windows*).

	sir_sid␣eastman␣	⇒	(0,0,"s")
s	ir␣sid␣eastman␣e	⇒	(0,0,"i")
si	r_sid␣eastman␣ea	⇒	(0,0,"r")
sir	␣sid␣eastman␣eas	⇒	(0,0,"␣")
sir␣	sid_eastman␣easi	⇒	(4,2,"d")

FIG. 1.14 – Encodage par LZ77 [SAL04]

La figure 1.14 montre la fenêtre glissante, composée de deux éléments :

– Le *Search Buffer*, SB, qui contient le texte qui vient d'être compressé,
– Le *Look-ahead Buffer*, LaB, contenant le texte à compresser.
Les tailles de ces buffers ne sont pas imposées, elles dépendent de l'implémentation.

Chaque portion de texte encodé est représentée par un triplet <o,l,cw>, où
– o est l'offset : distance séparant le pointeur du SB de la LaB,
– l est la longueur de la séquence commune aux deux buffers,
– "cw" est l'identifiant du caractère suivant la séquence à compresser dans le LaB.
Si on représente ce triplet sur l'exemple précédent, il vient :

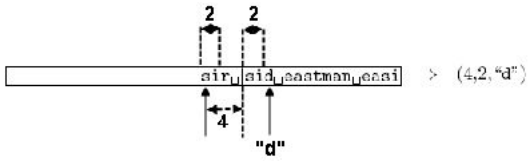


FIG. 1.15 – Encodage par LZ77 - triplet

En ce qui concerne le décodage, il suffit de regarder les valeurs des données du triplet, et le texte peut être reconstruit.

Au point de vue applicatif, les logiciels GZIP, PKZIP, etc. fonctionnent sur une variante du LZ77, portant le nom de DEFLATE[2].

LZ78 (ou LZ2)

L'algorithme LZ77 a subi beaucoup d'améliorations : optimisation de la recherche dans le SB, le codage des triplets, l'élimination du triplet encodant un caractère unique, ...

L'inconvénient majeur de cet algorithme est la manière dont il gère les séquences périodiques plus longues que la taille du SB. Ainsi, imaginons le cas d'un SB de longueur 6. Il suffit que la séquence soit plus longue d'un caractère pour qu'aucun des éléments la composant ne puisse être compressé. Ce cas est illustré à la figure pour la séquence ABCDEFG.

FIG. 1.16 – Problème lié au LZ77

L'algorithme LZ78 résoud ce problème par l'intermédiaire d'un dictionnaire qui devra être construit autant pour le codeur que pour le décodeur.

Les séquences codées ont la forme d'un couple <i,c>, où i est l'index de l'entrée dans le dictionnaire correspondant à la plus longue chaîne semblable à ladite séquence à coder, et c, le

[2]Voir Data Compression, The complete reference, 3rd, 2004.

caractère suivant la séquence dans le flux d'entrée. Si la séquence n'a pas de correspondance dans le dictionnaire, i est égal à 0, et le couple est ajouté au dictionnaire. Un exemple de dictionnaire est donné à la figure .

"sir␣sid␣eastman␣easily␣teases␣sea␣sick␣seals".

Dictionary	Token		Dictionary	Token
0	null			
1	"s"	(0,"s")	8	"a"	(0,"a")
2	"i"	(0,"i")	9	"st"	(1,"t")
3	"r"	(0,"r")	10	"m"	(0,"m")
4	"␣"	(0,"␣")	11	"an"	(8,"n")
5	"si"	(1,"i")	12	"␣ea"	(7,"a")
6	"d"	(0,"d")	13	"sil"	(5,"l")
7	"␣e"	(4,"e")	14	"y"	(0,"y")

FIG. 1.17 – LZ78 - construction du dictionnaire [SAL04]

LZW

La variante la plus populaire du LZ78 est l'algorithme LZW (Lempel, Ziv, Welch) créé en 1984. Le principe est de réduire l'envoi d'informations au décodeur. Au lieu du couple <i,c>, seul l'index <i> sera transmis.

Algorithme

- On initialise la table (le dictionnaire) en y plaçant les codes des caractères ASCII étendu,
- On regarde le caractère à transmettre :
  1. s'il existe déjà dans la table, on regarde le caractère suivant,
  2. si le groupe des deux existe également, on regarde le suivant, etc.
- Lorsqu'un nouveau groupe est découvert, on le définit en l'insérant dans le dictionnaire. Dans un premier temps, on transmet les codes des morceaux qui le composent. La prochaine fois qu'on le rencontrera, on transmettra son code propre. Les mots ajoutés au dictionnaire seront déterminés par l'intermédiaire d'une "fenêtre" évoluant au fil de l'analyse du texte à compresser. Ce concept est explicité dans l'exemple ci-dessous.
- On procède de la sorte jusqu'à la fin de la transmission.
- Lorsque le dictionnaire est plein, soit on procède à son extension, soit on se borne à utiliser les codes déjà existants.

La base du dictionnaire repose sur les 256 caractères du code ASCII étendu. A la suite de ceux-ci, on trouvera différents groupes de lettres rencontrés au fur et à mesure de la compression du texte.

Comment se construit le dictionnaire ?

Le principe est d'utiliser une fenêtre grandissant jusqu'à l'obtention d'un mot inexistant dans le dictionnaire. Auquel cas le mot en question est ajouté, le code du préfixe connu envoyé, et la fenêtre ramenée au dernier caractère analysé.

Pour plus de clarté, soit l'exemple suivant, sachant que les 256 premières entrées du dictionnaire sont initialisées avec les 256 valeurs du code ASCII étendu :

– Imaginons le message "ma maison" a compressé.
  1. Le premier caractère a analysé est "m". Il est dans le dictionnaire.
  2. Le caractère suivant est concaténé avec "m" et forme la chaine "ma". Elle n'est pas dans le dictionnaire : il faut donc l'ajouter. On lui affecte donc l'entrée 256 dans le dictionnaire (on commence à 0). On envoie la valeur correspondante à "m"(109).
  3. Un nouveau mot venant d'être ajouté, on reprend alors au dernier caractère pris en compte (ici "a"). Le caractère suivant est concaténé et forme la chaine "a ". Elle n'est pas dans le dictionnaire : il faut donc l'ajouter. On lui affecte donc l'entrée 257 dans le dictionnaire. On envoie la valeur correspondante à "a"(97).
  4. Un nouveau mot venant d'être ajouté, on reprend alors au dernier caractère pris en compte (ici " "). Le caractère suivant est concaténé et forme la chaine " m". Elle n'est pas dans le dictionnaire : il faut donc l'ajouter. On lui affecte donc l'entrée 258 dans le dictionnaire. On envoie la valeur correspondante à " "(32).
  5. Un nouveau mot venant d'être ajouté, on reprend alors au dernier caractère pris en compte (ici "m"). Le caractère suivant est concaténé et forme la chaine "ma". Cette chaine est déjà dans le dictionnaire. On concatène le caractère suivant et cela forme la chaine "mai". Elle n'est pas dans le dictionnaire : il faut donc l'ajouter. On lui affecte donc l'entrée 259 dans le dictionnaire. On envoie donc la valeur correspondante à "ma"(256).
  6. Un nouveau mot venant d'être ajouté, on reprend alors au dernier caractère pris en compte (ici "i"). Le caractère suivant est concaténé et forme la chaine "is". Elle n'est pas dans le dictionnaire : il faut donc l'ajouter. On lui affecte donc l'entrée 260 dans le dictionnaire. On envoie la valeur correspondante à "i"(105).
  7. Un nouveau mot venant d'être ajouté, on reprend alors au dernier caractère pris en compte (ici "s"). Le caractère suivant est concaténé et forme la chaine "so". Elle n'est pas dans le dictionnaire : il faut donc l'ajouter. On lui affecte donc l'entrée 261 dans le dictionnaire. On envoie la valeur correspondante à "s"(115).
  8. Un nouveau mot venant d'être ajouté, on reprend alors au dernier caractère pris en compte (ici "o"). Le caractère suivant est concaténé et forme la chaine "on". Elle n'est pas dans le dictionnaire : il faut donc l'ajouter. On lui affecte donc l'entrée 262 dans le dictionnaire. On envoie la valeur correspondante à "o"(111).
  9. Un nouveau mot venant d'être ajouté, on reprend alors au dernier caractère pris en compte (ici "n"). Le caractère suivant est concaténé et forme la chaine "n(eof)". Elle n'est pas dans le dictionnaire : il faut donc l'ajouter. Etant donné le caractère spécifique de fin de fichier, il n'est pas ajouté au dictionnaire. On envoie la valeur

correspondante à "n"(110).

– Au final, le flux compressé sera (109)(97)(32)(256)(105)(115)(111)(110).

– Lors de la décompression, le décodage se fera de manière inverse.

Cette compression est efficace pour de gros fichiers. Il n'en est pas de même pour un petit fichier, comme on le voit ci-dessus dans l'exemple : le message est trop court pour pouvoir bénéficier des avantages de la compression LZW.

Application du LZW - la compression sous UNIX

Il s'agit de la commande `compress`. Le dictionnaire a initialement une taille de 512 entrées. Ainsi, la taille des index est limitée à 9 bits ($2^9 = 512$). Une fois le dictionnaire plein, on double sa taille. Les index peuvent maintenant se transmettre sur 10 bits.

La taille maximale des index, notée b_{max}, peu être spécifiée par l'utilisateur entre 9 et 16, 16 bits étant la valeur par défaut. Une fois la taille du dictionnaire égale à $2^{b_{max}}$, `compress` poursuit la compression de manière statique (i.e., seules les séquences déjà présentes dans le dictionnaire peuvent être utilisées pour la compression).

1.2.7 La compression par antidictionnaire

Un antidictionnaire[3] est un ensemble de mots qui n'apparaissent pas dans le texte.

Pour mieux comprendre son fonctionnement, nous allons l'illustrer par un exemple :
– Soit un texte (binaire) que l'on souhaite compresser.
– Imaginons que le mot '1001' soit dans l'anti-dictionnaire.
– Dès lors, on pourra coder la séquence '1000' par '100'.
– En effet, sachant que le mot '1001' est dans l'anti-dictionnaire, c'est donc que ce même mot n'est pas dans le texte à coder. Donc seul un '0' peut suivre la séquence '100' dans le texte.

1.2.8 Catégorisation des compressions

Il existe plusieurs manières de classer les formats de compression. On peut notamment les scinder comme suit :
– Par analyse statistique ou par dictionnaire : Huffman VS LZW
– Avec ou sans perte (= destructive ou non destructive) :
  – Avec perte : des détails sont détruits lors de la compression, et il est impossible de les retrouver par la suite. On utilise les propriétés de l'oreille et de l'oeil humain pour supprimer les informations inutiles. Exemple : Jpeg
  – Sans perte : aucune perte et restitution parfaite après décompression. Exemples : Huffman, RLE, Zip, ...

[3] M.Crochemore, F.Mignosi, A.Restivo, S.Salemi., Data Compression using antidictionnary, 2000.

- Symétrique ou asymétrique :
  - Symétrique : le temps de calcul nécessaire pour la compression ou la décompression est équivalent. Il s'agit par exemple d'algorithmes de transmission de données.
  - Asymétrique : l'une des deux phases est plus rapide que l'autre, tels que les algorithmes d'archivage massif.

1.3 Ressources

Ressources bibliographiques :

Data compression, the complete reference, David Salomon, 2004, Springer.
Introduction to Data compression, Khalid Sayood, 2006, Morgan Kaufmann.

Illustrations :

- Data Compression, the complete reference, David Salomon, 2004, Springer
- Computer and Network Security, 3rd edition, W. Stallings, 2003, Prentice Hall
- Wikipedia.org

Chapitre 2

Le stockage des données

Après codage des informations, les données doivent être stockées sur un support quelconque. Dans la suite du cours, nous expliciterons :

- les supports de type carte magnétique
- les supports optiques
  - CD (laser infrarouge - 780nm)
  - DVD (laser rouge - 650nm)
  - HD-DVD (laser bleu - 405nm)
  - Blu-Ray (laser bleu - 405nm)
- les supports utilisant la technologie holographique

2.0.1 Représentation de quelques grandeurs binaires

Dans la figure 2.1, on compare des tailles en octets à des tailles plus "physiques". L'erreur d'approximation est volontaire, dans un but de représentation.

Kilo-octet (Ko) : 1 000 octets	Petit message
Méga-octet (Mo) : 1 000 000 octets	Petit roman
Giga-octet (Go) : 1 milliard d'octets	La 5ème symphonie de Beethoven
Téra-octet (To) : 1 000 Giga-octets	La totalité des rayons X dans un grand hôpital
Péta-octet (Po) : 1 000 Téra-octets	La moitié du contenu de toutes les bibliothèques universitaires des Etats-Unis
Exa-octet (Eo) : 1 000 Péta-octets	5 Eo = tous les mots prononcés par tous les habitants de la Terre depuis l'origine
Zetta-octet (Zo) : 1 000 Exa-octets	Autant d'information qu'il y a de grains de sable sur toutes les plages du monde
Yotta-octet (Yo) : 1 000 Zetta-octets	Autant d'information qu'il y a d'atomes dans 7 000 êtres humains

FIG. 2.1 – Valeurs représentatives en octets

2.0.2 Le support magnétique

Le principe est d'encoder les informations à partir d'un champ magnétique. La bande magnétique est constituée de pigments (p.ex. oxyde de fer). Par l'intermédiaire d'une tête d'écriture (un électro-aimant), on induit un champ magnétique qui va « marquer » ces pigments et ainsi y retenir une information. On parle de support « coercitif ».

Lors d'une lecture, la tête remarquera ces modifications de champs magnétiques. La tension électrique induite sera traduite et restituera les informations.

Les supports magnétiques sont encore couramment utilisés : disquettes, bandes magnétiques et disquettes haute-densité (Zip, Jaz, SuperDisk, PeerLess,...).

Ce support rencontre toutefois plusieurs problèmes :
- Usure relativement rapide
- Consommation forte en énergie
- Taille de certains supports de stockage

En raison de ces inconvénients, des recherches sont actuellement menées (notamment chez HP). Le type de stockage résultant porterait le nom de "Stockage à résolution atomique".

Stockage à résolution atomique (ARS)

Ce type de support de stockage permettrait 1000 Gbits/in^2 (1 inch = 2.54 cm).
Le principe est d'utiliser un réseau de pointes microscopiques qui écrivent et lisent sur un matériau spécifique. Une partie mécanique déplacera le support d'écriture. Ce matériau a la particularité de posséder deux états selon sa température, l'un servant à l'écriture, l'autre à la lecture.
Une pointe sous tension envoie un faisceau d'électrons qui écrira sur le matériau lorsque la température sera assez élevée, et après refroidissement, une autre pointe lira sur la surface à l'aide d'un faisceau plus faible.

Plusieurs problèmes importants restent à résoudre :
- Le mécanisme de déplacement doit avoir une précision de l'ordre du nanomètre
- Le système doit s'utiliser dans une atmosphère fermée pour éviter la dispersion des électrons à la sortie de la pointe

2.0.3 Support optique

En règle générale, pour les supports optiques, on utilisera le système de fichiers UDF (Universal Disk Format). Celui-ci est propre au stockage de données sur disque optique. Descendant de la norme ISO9660, c'est par son intermédiaire qu'il est possible d'ajouter des fichiers sur un disque après une première gravure (multi-session). Enfin, ce système de fichiers offre une compatibilité entre les systèmes d'exploitation (DOS, Windows, Linux, OS/2, Macintosh et UNIX)

Le Compact Disque

CD-Rom :
Les données sont « incrustées » dans une couche de plastique (polycarbonate) et recouvertes par une couche d'aluminium (ou d'or, ou d'argent). Le tout est recouvert par une couche de vernis et par une couche utilisée pour la présentation du disque (label). Ce type de stockage est durable car aucune partie du lecteur ne touche la surface des données. Il utilise les propriétés de réfraction de la lumière pour identifier les 1 et les 0 d'après les creux et plats (*pits and lands*).

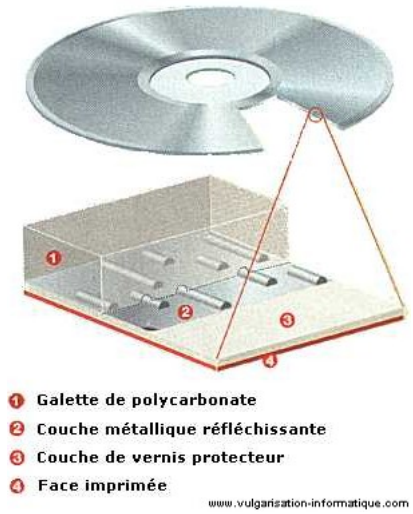


FIG. 2.2 – Structure d'un CD

La structure logique d'un CD se compose de 3 parties :
- La zone Lead-In : il s'agit de la zone la plus proche du centre du disque. Elle contient la TOC (*Table of Content*) permettant au lecteur de situer les données sur le disque.
- La zone de données
- La zone Lead-Out : elle contient des données nulles marquant la fin du disque. La technique d'Overburning permet d'écrire des données dans cette zone.

La capacité d'un disque audio varie de 74 à 99 minutes, de 650 à 870 MB. Ces tailles ont été au départ spécifiée dans des normes : Yellow Book pour le CD-ROM, Red Book pour le cd audio, Orange Book pour les CD-R, Green Book pour les CD-i, etc.

Sur un cd audio, chaque seconde occupe 75 secteurs du disque. Ainsi, un cd audio "plein" possède une capacité totale égale à 74x60x75 = 333.000 secteurs. La taille de ces secteurs dépend du contenu qu'ils renferment, un secteur de données audio nécessitant une correction d'erreurs moins importante (laissant donc place à plus d'information utile) que pour des données classiques (2353 octets contre 2048).

Un disque audio de 74 minutes possède donc une capacité de 783.216.000 octets (746MB) contre 681.984.000 octets (650MB) pour un CD de données.

Des calculs semblables expliquent les différences en termes de capacité pour les disque de 80, 90 et 99 minutes.

CD-R :
Une couche photosensible (entre le polycarbonate et l'aluminium) est brulée par le laser du graveur, ce qui permet de reproduire les « trous » présents dans un CD-Rom classique.

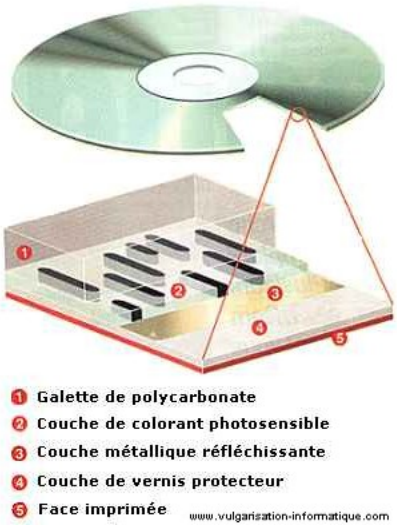


FIG. 2.3 – Structure d'un CD-R

- Le laser utilisé pour bruler la couche photosensible est dix fois plus puissant que celui utilisé pour la lecture.
- La couleur du CD-R varie selon le colorant utilisé pour sa fabrication. Selon ce dernier, la réflexion sera plus ou moins bonne, ce qui influera sur sa qualité et sa durée de vie.
- On trouve une spirale pré-imprimée sur le CD-R. Elle est nécessaire pour guider le laser du graveur.
- Deux zones supplémentaires s'ajoutent aux trois déjà présente sur le CD-Rom. La PCA (Power Calibration Area) permet au graveur de calibrer la puissance du laser pour les phases d'écriture et de lecture.
- La PMA (Power Memory Area) retient la position des sessions écrites mais non finalisées (lorsqu'elles le sont, elles sont placées dans la zone de Lead-in).

CD-RW :
La couche inscriptible est un alliage de plusieurs matériaux (argent, indium, antimoine, et tellure). Deux couches diélectriques sont insérées entre le polycarbonate, cette couche inscriptible et la couche métallique réfléchissante. Suivant la température appliquée, diverses réactions ont lieu au niveau des atomes la composant, ayant pour effet de laisser passer ou non la lumière.

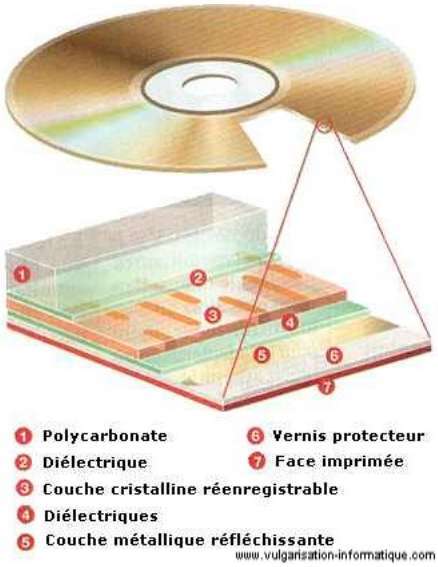


FIG. 2.4 – Structure d'un CD-RW

Le codage de l'information sur un cd :
L'information codée sur un cd-rom utilise le codage EFM, Eight-to-Fourty Modulation, traduisant 8 bits en 14. La contrainte de ce codage est que deux bits à 1 ne peuvent être distants de moins de 2 bits à 0 et ne peuvent être séparés par plus de 10 bits à 0. On note cette contrainte (2,10). Le codage EFM fait partie de la famille des codes RLL (Run Lenght Limited), et on le note ici RLL(2,10).

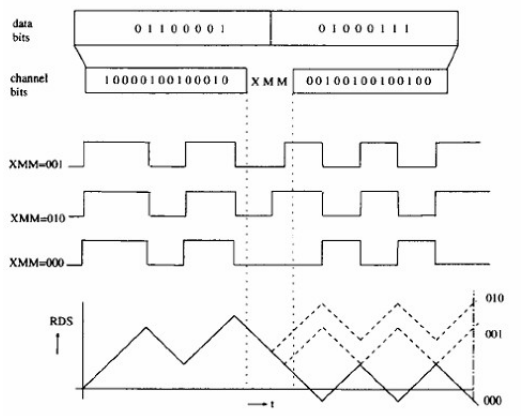


FIG. 2.5 – Evolution de la RDS selon les bits de liaison choisis [Immink, 1990]

Il existe 267 codes EFM respectant la contrainte (2,10), mais lorsqu'on concatène deux codes EFM, il peut arriver que le résultat n'y réponde plus. La solution est d'insérer des bits de liaison (*Merging bits*). Ces bits de liaison seront choisis tels que la valeur absolue de la DSV (Digital Sum Value) ou RDS (Running Digital Sum) soit la plus proche possible de 0. La RDS est un "indicateur de qualité de la réflexion" du disque qui, s'il est trop élevé ou trop bas, provoquera des erreurs de lecture.

Le DVD (Digital Versatil Disc)

Le support de stockage est similaire au CD-Rom, mais le code correcteur d'erreurs est plus évolué et nécessite moins de bits.

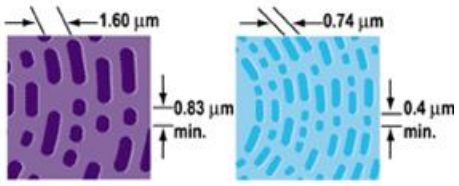


FIG. 2.6 – Comparaison CD - DVD

Le DVD DL (Double Layer)

En jouant sur la longueur d'onde du laser du lecteur (ou du graveur) et de la transparence des couches du disque, on peut lire (graver) plusieurs couches présentes sur le disque.

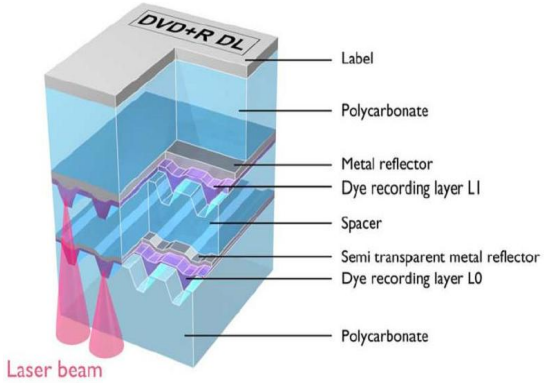


FIG. 2.7 – Structure d'un DVD Double Couche [DaTARIUS Group]

Le codage utilisé porte le nom d'EFMPlus. Légerement plus performant que l'EFM (gain de l'ordre de 6%), la différence conceptuelle principale est qu'il n'utilise que 2 bits de liaison[1].

[1]Voir "EFMPlus : The coding Format of the Multimedia Compact Disc, K.A.S Immink, IEEE, 1995" pour

Le HD-DVD

Au lieu d'utiliser le traditionnel laser rouge des graveurs conventionnels, on utilise ici le laser bleu, dont la longueur d'onde est plus courte.

Le codage utilisé est le ETM (Eight to Twelve Modulation). Il consiste en une table de correspondance entre les différents octets possibles et leur traduction sous la forme d'une suite de 12 bits. La contrainte est ici (1,10).

Le Blu-Ray

Il est lui aussi basé sur la technologie du laser bleu. Il possède dès lors une finesse de gravure égale à celle du HD-DVD. Ce support est plus performant en termes de capacité de stockage que le HD-DVD car le lecteur utilise une lentille plus évoluée.

Le codage des données utilisé par le Blu-Ray porte le nom de 17PP et s'applique tel qu'illustré à la figure 2.8.

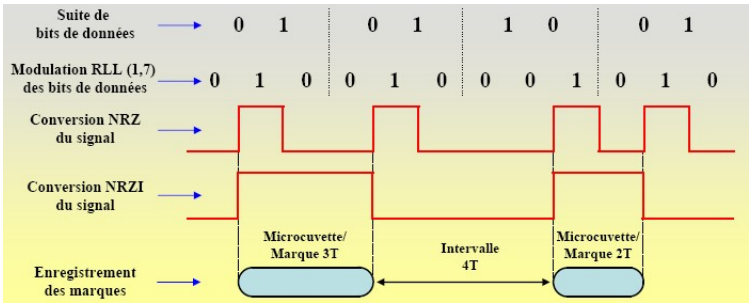


FIG. 2.8 – Codage des informations sur un disque Blu-Ray [Source : Jean-José Wanègue]

Remarques :

Le thème étant en constante évolution, certaines remarques seront peut-être obsolètes dans un laps de temps plus ou moins court.

- A l'heure actuelle, tous les formats sont compatibles.
- Le problème de la cartouche du Blu-Ray fut supprimé grâce à une technologie mise au point par TDK (application d'une couche de polymère protectrice). Le disque étant protégé, il n'était donc plus nécessaire de conserver la cartouche.
- Si on compare les tailles des supports, le hd-dvd permet de stocker 30Go (2 couches de 15Go), alors que le Blu-Ray autorise jusqu'à 50Go (2 couches de 25Go).
- En 2005, Toshiba a annoncé un Hd-DVD pouvant stocker 45 Go (3 couches). Au CES[2] 2006, TDK a présenté un prototype Blu-Ray de 100 Go (4 couches). Selon Sony, des recherches ont lieu pour créer des disques Blu-Ray de 8 couches et ainsi porter la taille du stockage à 200 Go.

plus de précisions

[2]Consumer Electronics Show

- Samsung a annoncé l'arrivée dans le courant 2006-2007 d'un lecteur compatible tout format (cd, dvd, hd-dvd et blu-ray.
- Les chaines de production pour le HD DVD sont sensiblement moins coûteuses.
- Hormis la capacité de stockage, les différences sont minimes (différences au niveau des lentilles utilisées, de la puissance du laser nécessaire, ...). Les deux supports sont compatibles avec les mêmes formats de compression vidéo et utilisent la même longueur d'onde du laser.

Autres supports

Il existe plusieurs autres supports optiques plus ou moins répandus :
- FMD (Fluorescent MultiLayer Disc) : ce disque utilise les propriétés de fluorescence au lieu des propriété de la réflexion. D'un point de vue théorique, il pourrait contenir jusqu'à cent couches.
- DMD (Digital Multilayer Disc) : il est basé sur le FMD. Il peut stocker jusqu'à 21 Go par couche, mais uniquement sur 2 couches (jusqu'à présent). Cependant, il n'y a plus d'évolutions depuis 2004.
- VMD (Versatile Multilayer Disc) : Concurrent des HD-dvd et Blu-ray, il utilise encore le laser rouge et autorise entre 20 et 100 Go par disque. Il permet effectivement jusqu'à 8 couches par face du disque de 4.7 à 6 Go chacune. Il n'a malheureusement pas, pour le moment, de soutien de la part des industries.
- EVD (Enhanced Versatile Disc) : C'est l'équivalent du dvd en Chine. D'une capacité de 8.5 Go, il utilise encore le laser rouge.
- FVD (Forward Versatile Disc) : Semblable au dvd à Taiwan. Il peut renfermer 6 Go et utilise lui aussi le laser rouge.

Recherches en cours : le support holographique

Un nouveau type de cupport fait de plus en plus parler de lui : le support holographique. On parle dès lors de HDSS (Holographic Data Storage System).
Les disques conventionnels offrent une lecture à deux dimensions. Avec ce système, il est possible

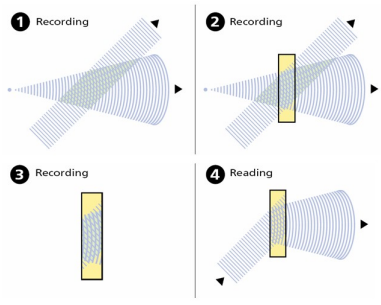


Fig. 2.9 – Ecriture et lecture sur un support holographique [InPhase]

de stocker les données de façon volumiques, c'est-à-dire dans l'épaisseur même du média. Cette technique repose sur une propriété des interférences naissant entre deux ondes lors de leur

passage dans un matériau photosensible. Ces interférences provoquent dans le matériau une série de transformations physiques et/ou chimiques.

L'avantage est que suivant l'angle des rayons utilisés, on peut superposer les informations, et y accéder de manière indépendante. Pour ce faire, on joue sur l'angle d'incidence du rayon de référence (Reference Beam).

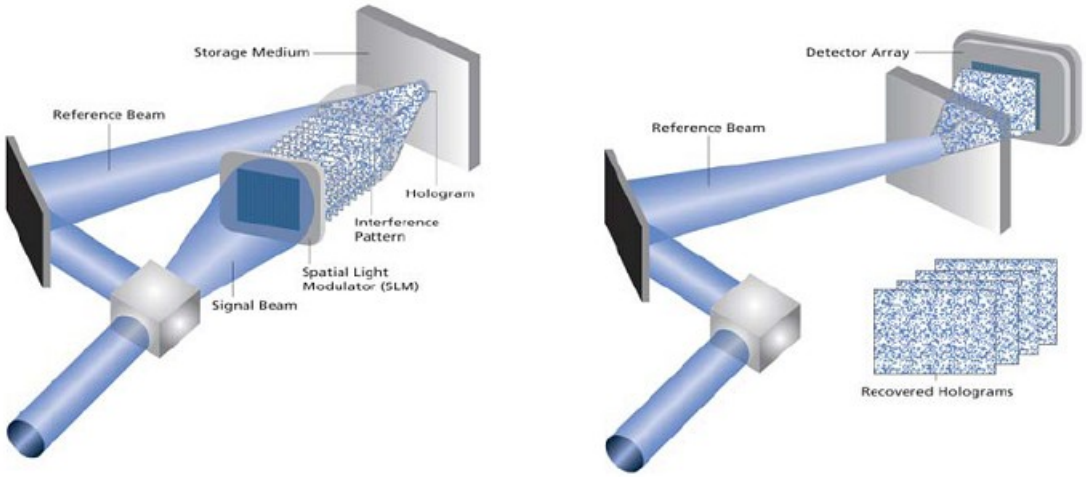


FIG. 2.10 – Ecriture et lecture sur un support holographique (II)[InPhase]

Par ce procédé, il pourrait être possible de créer des éléments de mémoire vive de 25 Go l'unité, des disques durs de 1 To (1000 Go) par plateau, ou encore des data warehouse de 1 Po (1 000 000 Go).

2.1 Ressources

Documents :
Runlength Limited Sequences, K.A.S. Immink, IEEE 1990 EFMPlus : The coding Format of the Multimedia Compact Disc, K.A.S. Immink, IEEE, 1995

Illustrations :

- EMC Annual Report 2000
- DVD Forum, Jean-José Wanègue
- CST-Commission Supérieure Technique de l'image et du son
- In-Phase Technologies

Ressources Internet :

`http://www.dataligence.com/`
`http://www.storagereview.com/`
`http://www.vulgarisation-informatique.com/graveur.php`
`http://www.research.ibm.com/journal/rd/443/ashley.html`
`http://www.optware.co.jp/english/`
`http://www.inphase-tech.com/`
`http://www.nmeinc.com/`