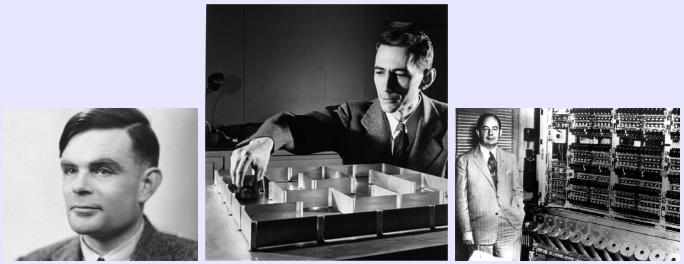


Information, Calcul et Communication

Module 2 : Information et Communication

EPFL ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Information, Calcul et Communication

Leçon 2.4 : Compression de données (2ème partie)

O. Lévêque

EPFL ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Plan

La semaine dernière :

- notion d'entropie
- compression sans perte
- algorithme de Shannon-Fano

Aujourd'hui :

- algorithme de Shannon-Fano (bis) : cas général
- analyse de performance – théorème de Shannon
- compression optimale : code de Huffman
- compression avec pertes

Compression sans pertes : rappel

- L'**entropie** du choix d'une lettre dans une séquence :

$$H(X) = p_1 \log_2\left(\frac{1}{p_1}\right) + \ldots + p_n \log_2\left(\frac{1}{p_n}\right)$$

  mesure le *désordre* / l'« *information* » contenue dans cette séquence :
  - n = taille de l'alphabet
  - p_j = probabilité d'apparition d'une lettre
  - $\log_2\left(\frac{1}{p_j}\right)$ = « nombre de questions binaires nécessaires pour deviner la j^e lettre »
- L'**algorithme de Shannon-Fano** :
  - **Règle n^0 1 :** le nombre de bits attribués à chaque lettre est égal au nombre de questions nécessaires pour la deviner.
  - **Règle n^0 2 :** les bits 0 ou 1 sont attribués en fonction des réponses obtenues aux questions.

  ☞ sur un cas particulier : a permis de représenter une séquence de lettres à l'aide de $H(X)$ bits par lettre en moyenne

Algorithme de Shannon-Fano (bis)

Remarque : La séquence prise en exemple la semaine dernière (IL FAIT BEAU A IBIZA) est particulière, car la probabilité d'apparition p de chaque lettre est une puissance inverse de 2 (c'est-à-dire $1/4$, $1/8$ ou $1/16$).

Donc pour chaque lettre, $\log_2\left(\frac{1}{p}\right)$ est un nombre entier, qui correspond exactement au nombre de questions nécessaires pour la deviner.

En pratique, ce n'est bien sûr pas toujours le cas...

Algorithme de Shannon-Fano (bis)

Considérons par exemple cette autre séquence (12 lettres au total, sans les espaces) :

JE PARS A PARIS

Comment procéder pour représenter cette séquence avec un nombre minimum de bits par lettre ?

A nouveau, reprenons notre jeu des questions et construisons le même tableau qu'avant, en classant les lettres par ordre décroissant du nombre d'apparitions :

lettre	A	P	R	S	J	E	I
nb d'apparitions	3	2	2	2	1	1	1

On remarque ici qu'il n'est pas possible de diviser l'ensemble des lettres en deux parties telles que le nombre d'apparitions à gauche égale le nombre d'apparitions à droite.

Algorithme de Shannon-Fano (bis)

lettre	A	P	R	S	J	E	I
nb d'apparitions	3	2	2	2	1	1	1

Cependant, on peut faire en sorte de *minimiser la différence* entre le nombre d'apparitions à gauche et à droite.

Dans l'exemple ci-dessus, on a en fait deux possibilités pour la première question :

soit « est-ce que la lettre est un A ou un P ? »

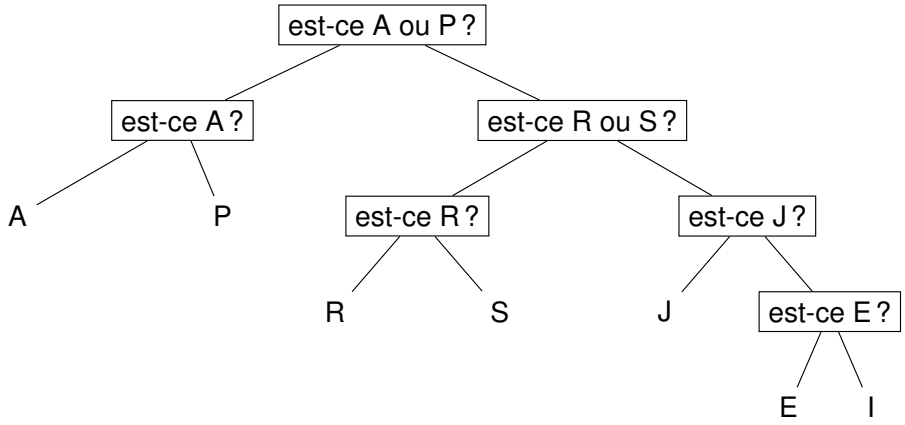
soit « est-ce que la lettre est un A, un P ou un R ? »

Supposons qu'on choisisse la première possibilité ; le jeu continue comme précédemment, avec cette nouvelle règle un peu plus souple...

Algorithme de Shannon-Fano (bis)

Faisons-le ensemble :

lettre	A	P	R	S	J	E	I
nb d'apparitions	3	2	2	2	1	1	1



Algorithme de Shannon-Fano (bis)

Ce qui nous amène (par exemple !) au tableau suivant :

lettre	A	P	R	S	J	E	I
nb d'apparitions	3	2	2	2	1	1	1
nb de questions	2	2	3	3	3	4	4

Donc le nombre moyen de questions à poser pour deviner une lettre est :

$$\frac{5}{12}\times 2+\frac{5}{12}\times 3+\frac{2}{12}\times 4=\frac{10+15+8}{12}=\frac{33}{12}=\frac{11}{4}=2.75$$

La règle d'attribution des mots de code à chaque lettre est exactement la même qu'avant :

- **Règle n^0 1 :** le nombre de bits attribués à chaque lettre est égal au nombre de questions nécessaires pour la deviner.
- **Règle n^0 2 :** les bits 0 ou 1 sont attribués en fonction des réponses obtenues aux questions. Plus précisément, le bit n^0 j est égal à 1 ou 0 selon que la réponse à la j^e question était oui ou non.

Algorithme de Shannon-Fano (bis)

Ce qui donne le dictionnaire suivant :

lettre	A	P	R	S	J	E	I
nb d'apparitions	3	2	2	2	1	1	1
nb de questions	2	2	3	3	3	4	4
mot de code	11	10	011	010	001	0001	0000

(remarquer à nouveau qu'aucun mot de code n'est le préfixe d'un autre, *par construction*)

Un calcul en tout point identique au précédent montre que le nombre de bits utilisés pour représenter la séquence

JE PARS A PARIS

au moyen du code ci-dessus est égal à 33 (et le nombre moyen de bits par lettre est donc égal à $\frac{33}{12}=2.75$)

Algorithme de Shannon-Fano (bis)

Pour finir, calculons l'entropie de la séquence :

lettre	A	P	R	S	J	E	I
nb d'apparitions	3	2	2	2	1	1	1
probabilité p	1/4	1/6	1/6	1/6	1/12	1/12	1/12

(remarquer que $\log_2\left(\frac{1}{p}\right)$ n'est pas nécessairement un nombre entier)

Le calcul de l'entropie donne :

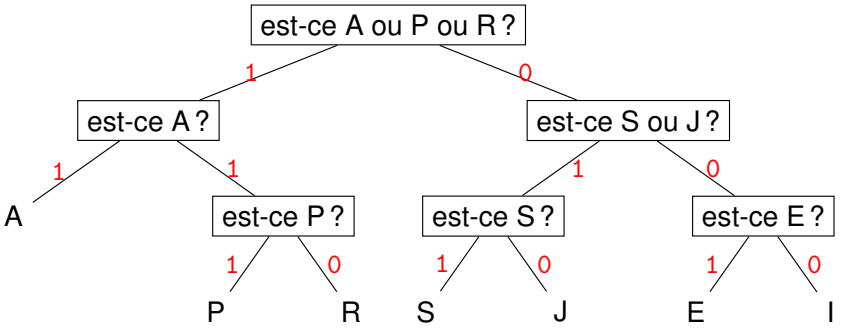
$$H(X)=\frac{1}{4}\log_2(4)+\frac{3}{6}\log_2(6)+\frac{3}{12}\log_2(12)\simeq 2.69\text{ bit}$$

qui est un peu plus petite que la valeur 2.75 trouvée précédemment.

Algorithme de Shannon-Fano (bis)

Et si au départ on avait choisit l'autre question ?

lettre	A	P	R	S	J	E	I
nb d'apparitions	3	2	2	2	1	1	1



Longueur moyenne : $\frac{3}{12}\times 2+\frac{9}{12}\times 3=\frac{33}{12}=2.75$

Analyse de performance

Quelques définitions :

- Un *code binaire* est un ensemble C dont les élements $c_1, \ldots, c_n$ (également appelés *mots de code*) sont des suites de 0 et de 1 de longueur finie.

  Exemple : { 11, 10, 011, 010, 001, 0001, 0000 }

- On note ℓ_j la longueur d'un mot de code c_j.

  Exemple : $\ell_5 = 3$

- Un code binaire C est dit *sans préfixe* si aucun mot de code n'est le préfixe d'un autre. Ceci garantit :
  - que tous les mots de code sont différents ;
  - que tout message formé de ces mots de code peut être décodé au fur et à mesure de la lecture.

  Exemple : avec le code ci-dessus, 1101010011 se lit 11, 010, 10, 011

Analyse de performance

Quelques définitions (suite) :

- Le code $C = \{c_1, \ldots, c_n\}$ peut être utilisé pour représenter une séquence X formée avec des lettres tirées d'un alphabet $A = \{a_1, \ldots, a_n\}$:

  chaque lettre a_j est représentée par le mot de code c_j de longueur ℓ_j

  Exemple :

lettre	A	P	R	S	J	E	I
mot de code	11	10	011	010	001	0001	0000

- Si les lettres $a_1, \ldots, a_n$ apparaissent avec des probabilités $p_1, \ldots, p_n$ dans la séquence X, alors la **longueur moyenne du code** (i.e. le nombre moyen de bits utilisés par lettre) est donnée par

$$L(C) = p_1 \ell_1 + \ldots + p_n \ell_n$$

Analyse de performance

Théorème de Shannon : *Quel que soit* le code binaire C *sans préfixe* (et sans perte) utilisé pour représenter une séquence X donnée, l'inégalité suivante est toujours vérifiée :

$$L(C) \geq H(X)$$

On voit donc également ici apparaître un **seuil** (cf. aussi théorème d'échantillonnage) : en dessous de ce seuil, il n'est pas possible de comprimer des données (de façon non ambigüe) sans faire de perte.

Pour démontrer ce théorème, nous avons besoin de l'inégalité suivante :

Inégalité de Kraft : Soit $C = \{c_1, \ldots, c_n\}$ un code binaire sans préfixe. Alors

$$2^{-\ell_1} + \ldots + 2^{-\ell_n} \leq 1$$

(rappelons que ℓ_j est la longueur du mot de code c_j).

(pour information) **Démonstration de l'inégalité de Kraft :**

Soit $\ell_{\max}$ la longueur du mot de code le plus long dans C.

On peut représenter chaque mot du code C par un nœud dans un arbre binaire de profondeur $\ell_{\max}$.

On appelle « descendants » d'un mot de code donné les mots de code situés en dessous de celui-ci dans l'arbre.

Observations :

- Au niveau $\ell_{\max}$, il y a $2^{\ell_{\max}}$ nœuds en tout.
- Le mot de code c_j a $2^{\ell_{\max} - \ell_j}$ descendants au niveau $\ell_{\max}$.
- A cause de l'hypothèse (code sans préfixe), les descendants de mots de code distincts sont tous distincts également.

Donc

$$2^{\ell_{\max} - \ell_1} + \ldots + 2^{\ell_{\max} - \ell_n} \leq 2^{\ell_{\max}}$$

En divisant de chaque côté par $2^{\ell_{\max}}$, on obtient l'inégalité de Kraft.

QED

Démonstration du théorème de Shannon :

Par définition :

$$H(X) - L(C) = \left(p_1 \log_2 \left(\frac{1}{p_1} \right) + \ldots + p_n \log_2 \left(\frac{1}{p_n} \right) \right) - (p_1 \ell_1 + \ldots + p_n \ell_n)$$

$$= p_1 \left(\log_2 \left(\frac{1}{p_1} \right) - \ell_1 \right) + \ldots + p_n \left(\log_2 \left(\frac{1}{p_n} \right) - \ell_n \right)$$

Remarquer que $-\ell_j = \log_2 \left(2^{-\ell_j} \right)$, donc

$$\log_2 \left(\frac{1}{p_j} \right) - \ell_j = \log_2 \left(\frac{1}{p_j} \right) + \log_2 \left(2^{-\ell_j} \right) = \log_2 \left(\frac{2^{-\ell_j}}{p_j} \right)$$

d'où

$$H(X) - L(C) = p_1 \log_2 \left(\frac{2^{-\ell_1}}{p_1} \right) + \ldots + p_n \log_2 \left(\frac{2^{-\ell_n}}{p_n} \right)$$

Démonstration du théorème de Shannon (suite) :

En utilisant à nouveau le fait que $f(x) = \log_2(x)$ est concave, on obtient :

$$H(X) - L(C) = p_1 \log_2 \left(\frac{2^{-\ell_1}}{p_1} \right) + \ldots + p_n \log_2 \left(\frac{2^{-\ell_n}}{p_n} \right)$$

$$\leq \log_2 \left(p_1 \frac{2^{-\ell_1}}{p_1} + \ldots + p_n \frac{2^{-\ell_n}}{p_n} \right) = \log_2 \left(2^{-\ell_1} + \ldots + 2^{-\ell_n} \right)$$

Par l'inégalité de Kraft :

$$2^{-\ell_1} + \ldots + 2^{-\ell_n} \leq 1$$

et le fait que $f(x) = \log_2(x)$ est croissante, on conclut que

$$H(X) - L(C) \leq \log_2(1) = 0$$

donc $L(C) \geq H(X)$, ce qui prouve le théorème de Shannon. QED

Analyse de performance

On peut montrer également qu'avec l'algorithme de Shannon-Fano,

$$L(C) \leq H(X) + 1$$

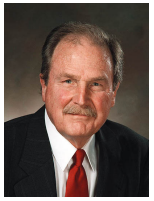
et donc que la performance de celui-ci est proche de la meilleure performance qu'on puisse espérer.

Remarques :

- L'inégalité ci-dessus est en général pessimiste. On a vu dans l'exemple précédent que $L(C)$ peut être plus proche de $H(X)$, et même égal à $H(X)$ dans certains cas particuliers.
- La démonstration de cette inégalité est un peu technique, donc nous ne la ferons pas ici.
- La performance de l'algorithme de Huffman (qui est optimale) est aussi dans ces bornes.

Codes de Huffman

- Du point de vue du taux de compression, l'algorithme de Shannon-Fano n'est pas garanti être optimal, mais il offre cependant une très bonne performance dans la majorité des cas.
- Du point de vue du temps de calcul, il n'est pas optimal non plus.
- Ce n'est donc pas cet algorithme qui est implémenté en pratique, mais une version similaire : l'**algorithme de Huffman**.



- David Albert Huffman (1925 - 1999)
- ingénieur électricien, pionnier de l'informatique,
- spécialiste d'origami...

- Les codes de Huffman sont **optimaux** : (pour une compressions *sans perte*,) on ne peut pas faire plus court !

Codes de Huffman

L'idée est vraiment très simple : **affecter les codes les plus long aux lettres les moins fréquentes**

c.-à-d. mettre les lettres les moins fréquentes les plus en bas de l'arbre de codage

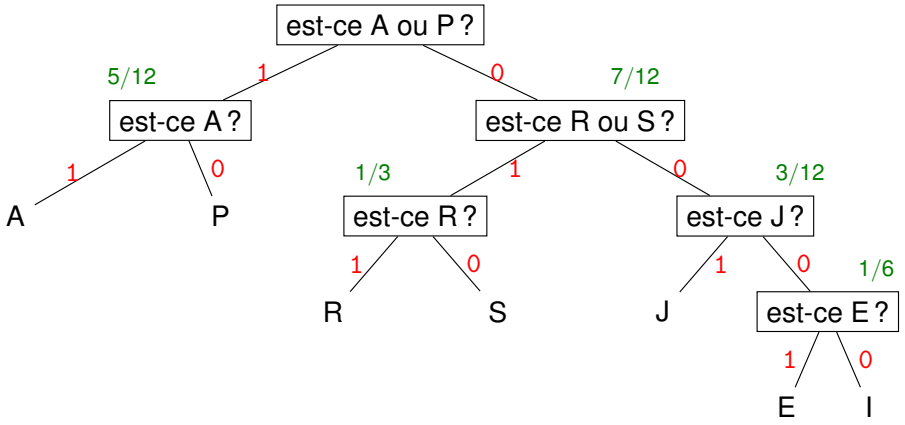
Comme pour l'algorithme de Shannon-Fano, on part du tableau des lettres et de leur nombre d'apparition (ou probabilité).

L'algorithme procède alors itérativement comme suit :

1. Trouver les 2 « lettres » les moins fréquentes et les regrouper sous une « question » qui les distingue.
   En cas d'égalité, en choisir 2 au hasard (ça ne change pas la longueur moyenne du code).
2. Dans le tableau des nombres d'apparitions des lettres, supprimer les 2 dernières lettres considérées et les regrouper comme une seule nouvelle « lettre », avec comme nombre d'apparitions la somme des deux nombres.
3. Recommencer en 1 tant qu'il y a des « lettres ».

Code de Huffman : exemple

lettre	A	P	R	S	J	E	I
nb d'apparitions	3	2	2	2	1	1	1
probabilité p	1/4	1/6	1/6	1/6	1/12	1/12	1/12



Code de Huffman : exemple

Ce qui donne le dictionnaire suivant :

lettre	A	P	R	S	J	E	I
nb d'apparitions	3	2	2	2	1	1	1
mot de code	11	10	011	010	001	0001	0000

Remarques :

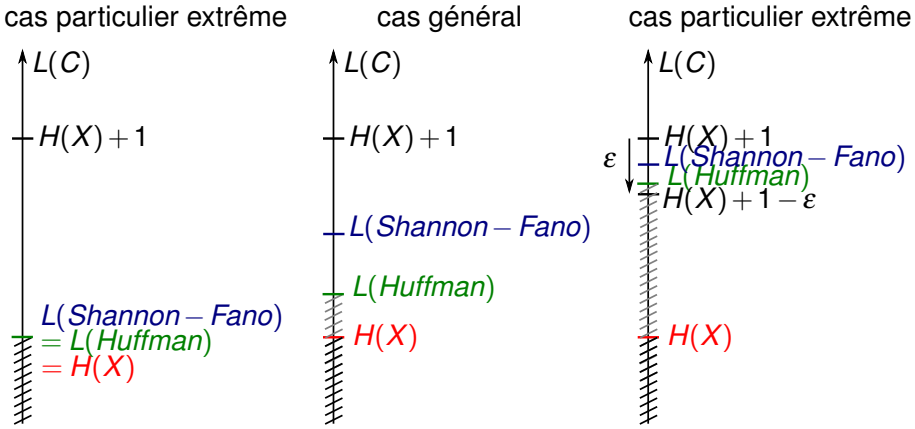
- Un code de Huffman est sans préfixe par construction (arbre).
- Dans le *cas particulier* de l'exemple, on retrouve un code de Shannon-Fano : ce n'est pas forcément le cas en général.
- Notez qu'il faut bien commencer par les deux ***moins* fréquents en premier** et surtout pas par les deux plus fréquents !
  ☞ construction de **bas en haut**

Résumé

Pour un code non ambigu sans perte :

- L'entropie est la borne inférieure à la longueur moyenne de codage.
- Il existe au moins deux codes (Shannon-Fano et Huffman) dont la longueur moyenne de codage est inférieure à l'entropie plus un bit.
- Le code optimal (on ne peut pas faire plus court) est le code de Huffman.

Résumé



Exemple Shannon-Fano $\neq$ Huffman

HOURRA HOURRA HOURRRRA

lettre	R	A	H	O	U
nb d'apparitions	8	3	3	3	3
Shannon-Fano	00	01	10	110	111
Huffman	0	100	101	110	111

20

$L(\text{Shannon-Fano}) = 2.3$ bit

$L(\text{Huffman}) = 2.2$ bit

$H(X) = 2.17$ bit

Plan

La semaine dernière :

- notion d'entropie
- compression sans perte
- algorithme de Shannon-Fano

Aujourd'hui :

- algorithme de Shannon-Fano (bis) : cas général
- analyse de performance – théorème de Shannon
- compression optimale : code de Huffman
- **compression avec pertes**

Compression avec pertes

- Pour comprimer des données sans perte et avec un code non ambigu, on ne peut donc pas descendre en-dessous de la borne de Shannon (= entropie).
- Si on essaye de descendre en-dessous de cette borne en utilisant un code ambigu, on court à la catastrophe !
- **Exemple** avec le message :

  JE PARS A PARIS
- Essayons d'utiliser le code suivant :

lettre	A	P	R	S	J	E	I
mot de code	1	0	11	10	01	00	111

- Avec un tel code, on n'utilise que 18 bits au total, et pas 33, mais la représentation binaire du message donne : 0100...
- Le destinataire ne peut donc pas savoir si le message commence par « JE », « JPP », « PAE », « PAPP » ou encore « PSP »...

Compression avec pertes

On ne va donc pas laisser tomber l'hypothèse de non ambiguïté.

Cependant, on est parfois *obligé* de comprimer en faisant des pertes :

- lorsqu'on désire représenter un nombre réel avec un nombre fixé de bits (l'information comprise dans un nombre réel est « infinie ») ;
- lorsqu'on désire échantillonner un signal dont la plus grande fréquence est « infinie » ;
- lorsqu'on désire télécharger sur un site web l'intégralité de ses photos de vacances (quelques Gigaoctets pour une centaine de photos avec la plus haute résolution).

Comment procéder dans ce dernier cas ?

D'abord filtrer les hautes fréquences !

Compression avec pertes : images

De la même manière que l'oreille humaine n'est pas capable de percevoir des sons au-delà de ~ 20 kHz, l'œil humain a un pouvoir de résolution d'environ une minute d'arc $= 1/60 \sim 0.017$ degré, ce qui veut dire qu'il ne distingue pas :

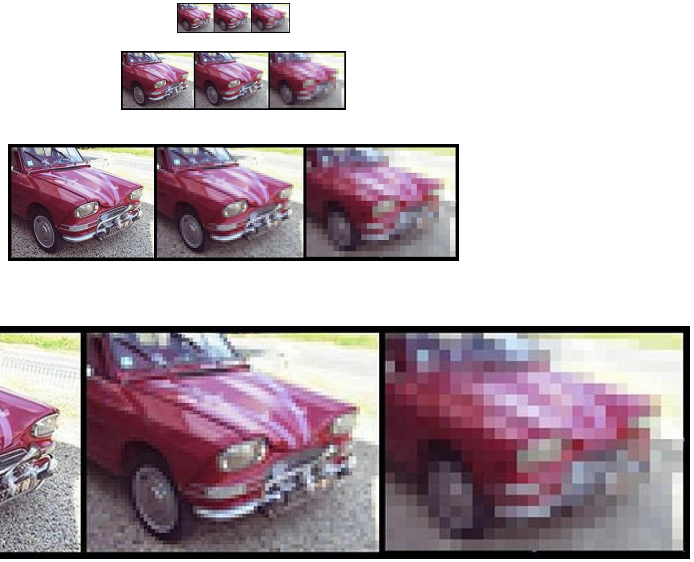
- des cratères sur la lune d'un diamètre inférieur à 100km ;
- des objets de taille inférieure à 1mm situés à 3m de distance ;
- des pixels de taille inférieure à 0.2×0.2mm sur un écran d'ordinateur (à 50cm de distance).

Comment filtrer les hautes fréquences (spatiales) dans une image ?

Une façon simple de faire : moyenner les couleurs sur des zones de plus ou moins grande taille.

☞ c'est un filtre à moyenne mobile !

Compression avec pertes : images



Compression avec pertes : images

On voit apparaître un *compromis* :

- plus on utilise des pixels de grande taille, moins on a besoin d'espace mémoire pour stocker l'image...
- mais plus le signal d'origine est déformé : on parle de **distorsion**.

Il existe bien sûr des algorithmes sophistiqués pour comprimer une image avec pertes :

- format JPEG :
  - analyse les fréquences spatiales présentes dans l'image ;
  - n'en retient que les plus basses ;
  - utilise un algorithme de compression sans pertes par-dessus le tout.
- format JPEG 2000 : la même chose, mais avec des *ondelettes*.

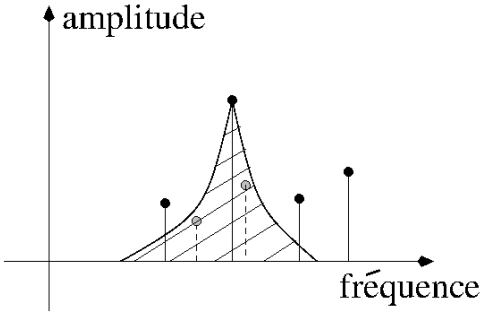
Compression avec pertes : son

Et pour le son, comment procéder ?

- Pour rappel, le son enregistré sur un CD est échantillonné à 44.1 kHz, et chaque échantillon est encodé sur 2×16 bits (2 canaux pour la stéréo).
- Pour une seconde, il faut donc $44100 \times 16 \times 2 \simeq 1.4$ Mib (mégibit).
- Le format MP3 permet d'encoder cette information sur 128 Kib seulement, ce qui correspond-on à une réduction de 90% de la taille d'un fichier ! (sans déformation *sensiblement audible* du son)
- Quel est le « truc » ?

Compression avec pertes : son

C'est (en partie) grâce à *l'effet de masque* : lorsqu'une sinusoïde avec une certaine fréquence est présente avec grande amplitude dans un son, elle cache à l'oreille humaine les autres sinusoïdes de fréquences proches et de moindre amplitude (c'est un effet *psychoacoustique*).



En conséquence, il n'y a pas besoin d'encoder une partie du signal, car on ne l'entend de toutes façons pas !
La même chose est vraie pour des sons rapprochés dans le temps.

Conclusion

Dans ce module, nous avons vu :

- Un aperçu de la théorie de l'*échantillonnage des signaux*.
- Un aperçu de la théorie de la *compression de données*.
- Ces deux théories sont deux facettes d'un seul et même sujet...
  (représenter numériquement de façon optimale et sans perte les informations du monde physique)
- ...avec deux résultats centraux de même nature !
- Si vous ne *deviez* retenir qu'une seule chose de ce module :

Il est toujours possible de stocker un signal ou
des données en économisant de l'espace mémoire !

Il y a cependant des seuils à ne pas
dépasser si on ne veut pas faire de pertes...

Les mesures de ces seuils sont respectivement
(2 fois) la bande passante et l'entropie

Attention à l'utilisation du nom « Shannon »

Claude Shannon a fait beaucoup de choses et son nom est utilisé à plusieurs endroit **différents**.

Ne les confondez pas !
Utilisez les bons noms pour les bonnes choses.

Dans ce module, nous avons vu :

- le théorème de Nyquist-**Shannon** : théorème d'échantillonnage ;
- l'algorithme de codage de **Shannon**-Fano ;
- le « premier théorème » de **Shannon** :
  entropie = limite à la taille moyenne de codes compresseurs ;
  appelé aussi « théorème de codage de source ».