Entropie: quelques rappels

L'entropie d'une séquence X est définie par :

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

où $p_1, \ldots, p_n$ sont les probabilités d'apparition des lettres $a_1, \ldots, a_n$ dans la séquence X.

$H(X)$ ne dépend donc pas de l'ordre des lettres, ni de leurs valeurs!

$$H(BANANA) = H(ANANAS) = H(BONOBO)$$

Information, Calcul et Communication

Algorithmes de Shannon-Fano et Huffman

École polytechnique fédérale de Lausanne

Olivier Lévêque

Compression sans pertes

Principe : Utiliser la redondance présente dans les données (en pratique, abréger ce qui revient souvent)

Exemple : Voici une séquence de 13 lettres (en incluant les deux points d'exclamation):

A B R A C A D A B R A !!

Le but est d'encoder cette séquence de lettres en une séquence de bits, en utilisant le moins de bits possibles et en respectant les règles suivantes :

- A chaque lettre correspond un unique mot de code (ex : A ↔ 01). L'ensemble des mots de code constitue un dictionnaire.
- La séquence de bits ainsi produite doit être décodable de manière unique à l'aide du dictionnaire.

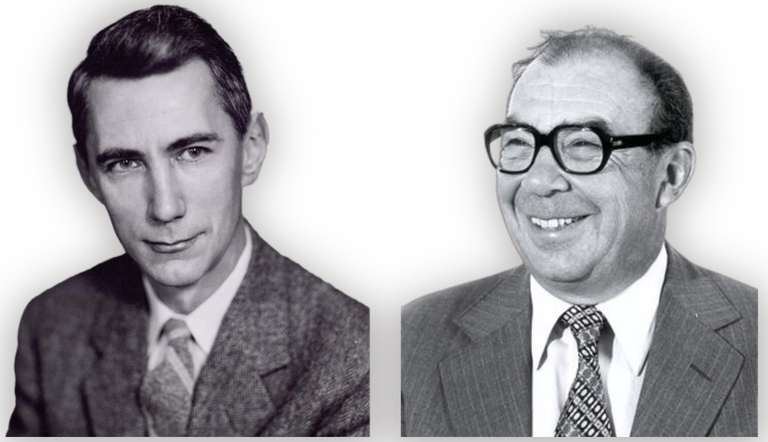
Compression sans pertes

Deux approches :

1. En suivant une version légèrement modifiée du jeu des questions, on obtient l'algorithme de Shannon-Fano.

2. En regroupant les lettres au fur et à mesure selon leurs probabilités d'apparition, on obtient l'algorithme de Huffman.

Algorithme de Shannon-Fano



Vous avez déjà rencontré Claude Shannon...

Voici Robert Fano (1917-2016), prof. au MIT

Reprenons le jeu des questions sur notre nouvel exemple :

A B R A C A D A B R A !!

- Comme précédemment, on commence par classer les lettres de la séquence dans l'ordre décroissant de leur nombre d'apparitions :

Lettre	A	B	R	!	C	D
Nombre d'apparitions	5	2	2	2	1	1

- Quelle est la meilleure première question à poser ici ?

Algorithme de Shannon-Fano

A B R A C A D A B R A !!

Lettre	A	B	R	!	C	D
Nombre d'apparitions	5	2	2	2	1	1

- **Problème :** On ne peut pas diviser l'ensemble des lettres en deux parties strictement égales (tout en respectant l'ordre décroissant).
- **Solution :** On minimise la différence de taille entre les deux ensembles

→ **Q1:** "Est-ce un A ou un B ?" (→ 7 et 6 possibilités, respectivement)

- **Si la réponse est oui,** quelle est la 2ème question à poser ?

  → **Q2:** « Est-ce un A ? » (on n'a pas tellement le choix ici...)

- **Si la réponse est non,** quelle est la 2ème question à poser ?

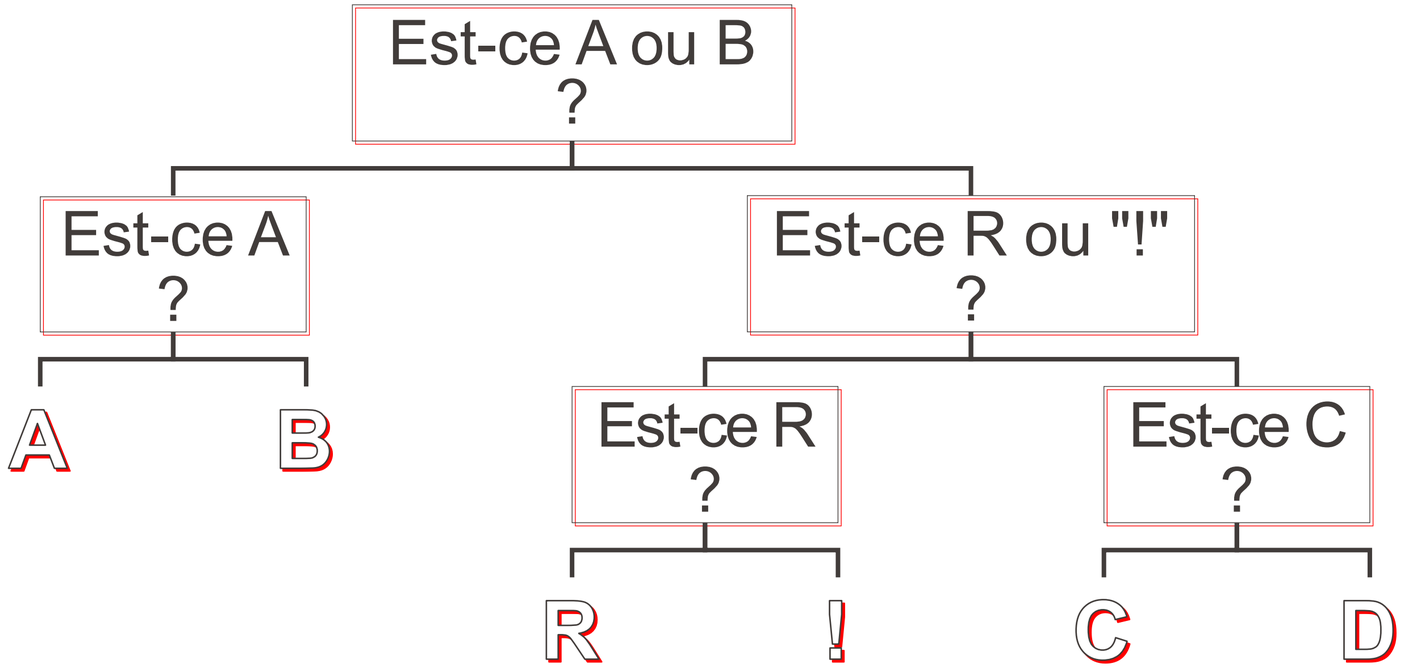
  → **Q2:** « Est-ce un R ou un "!" ? » ou **Q2:** « Est-ce un R ? »

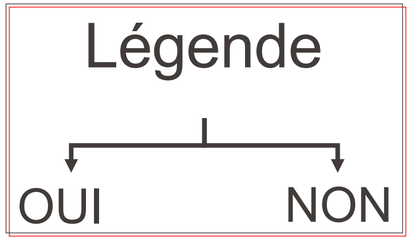
  On a le choix ici ! (mais aucun indice sur quel est le meilleur...)

Algorithme de Shannon-Fano: Option 1

A B R A C A D A B R A !!

Lettre	A	B	R	!	C	D
Nombre d'apparitions	5	2	2	2	1	1





Algorithme de Shannon-Fano: Option 1

Construction du dictionnaire :

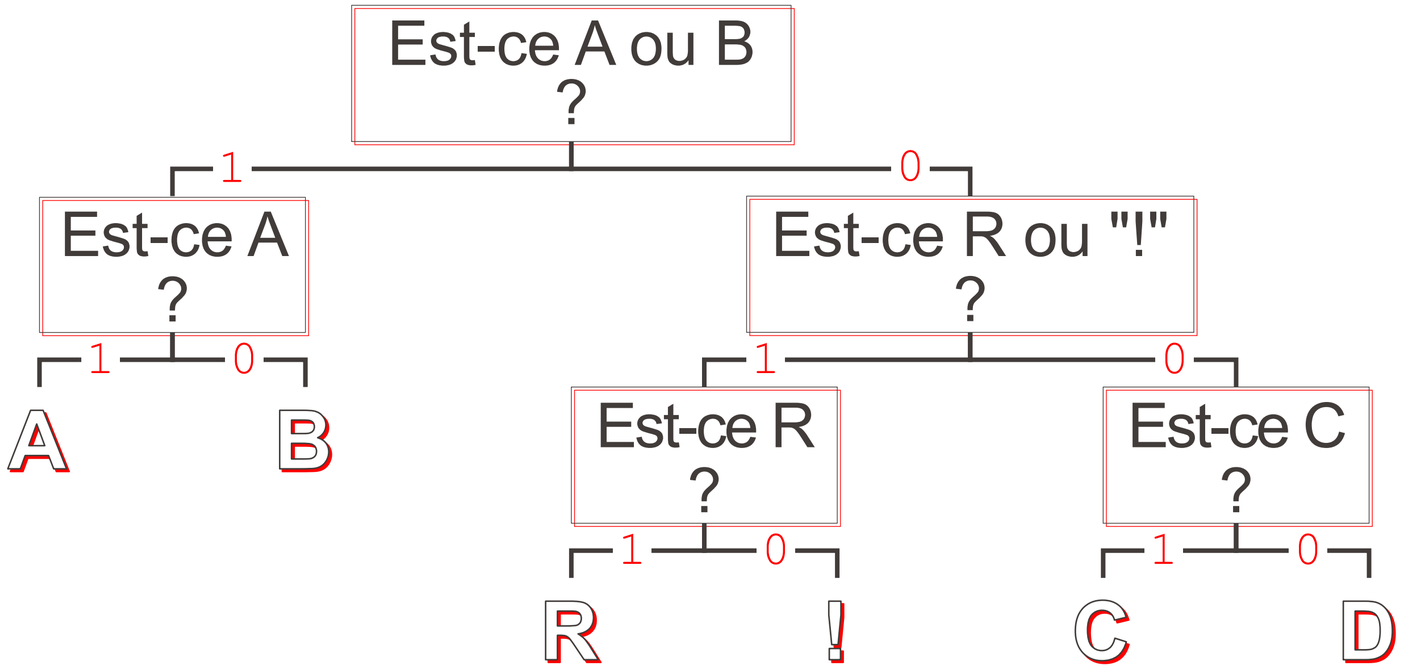
- **Règle n° 1 :** Le nombre de bits attribués à chaque lettre est égal au nombre de questions nécessaires pour la deviner.

- **Règle n° 2 :** Les bits 1 ou 0 sont attribués en fonction des réponses « oui » ou « non » obtenues aux questions.

Algorithme de Shannon-Fano: Option 1

A B R A C A D A B R A !!

Lettre	A	B	R	!	C	D
Nombre d'apparitions	5	2	2	2	1	1
Nombre de questions	2	2	3	3	3	3
Mot de code	11	10	011	010	001	000



Algorithme de Shannon-Fano: Option 1

- Dictionnaire résultant :

Lettre	A	B	R	!	C	D
Nombre d'apparitions	5	2	2	2	1	1
Nombre de questions	2	2	3	3	3	3
Mot de code	11	10	011	010	001	000

- Pour encoder la séquence **A B R A** C A D A **B R A !!**, on utilise donc

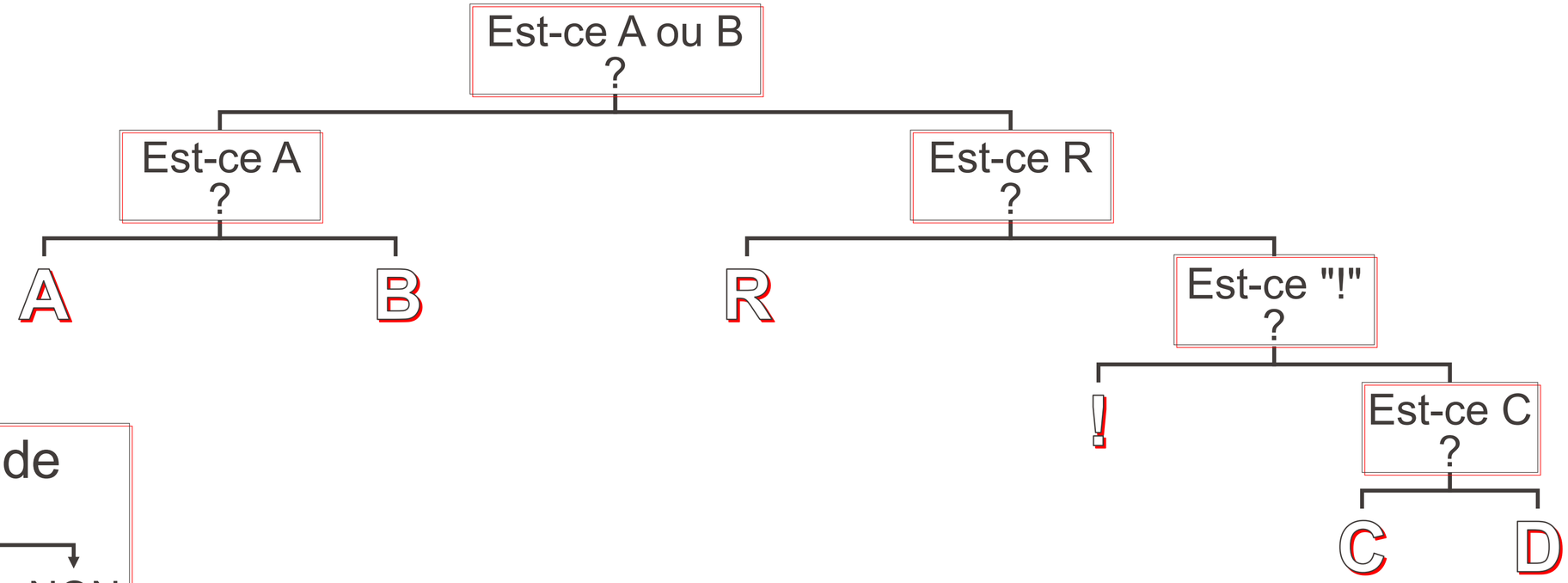
  $2 \cdot 7 + 3 \cdot 6 = 32$ bits au total, ce qui représente $\frac{32}{13} \approx 2.46$ bits par lettre.

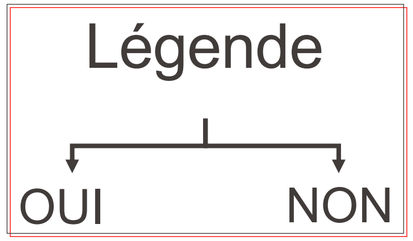
- Et la séquence de bits ainsi produite : 111001111... est parfaitement décodable à l'aide du dictionnaire, si on la lit de gauche à droite (car aucun mot de code du dictionnaire n'est le préfixe d'un autre).

Algorithme de Shannon-Fano: Option 2

A B R A C A D A B R A !!

Lettre	A	B	R	!	C	D
Nombre d'apparitions	5	2	2	2	1	1

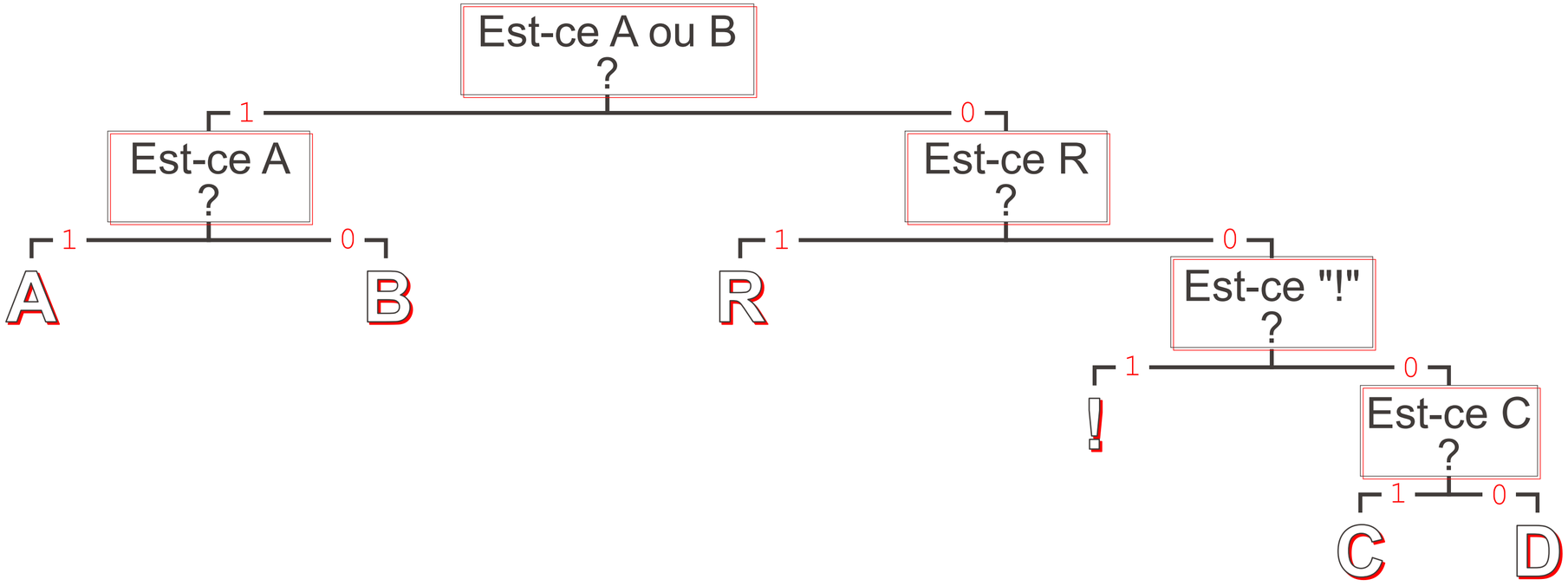




Algorithme de Shannon-Fano: Option 2

A B R A C A D A B R A !!

Lettre	A	B	R	!	C	D
Nombre d'apparitions	5	2	2	2	1	1
Nombre de questions	2	2	2	3	4	4
Mot de code	11	10	01	001	0001	0000



Algorithme de Shannon-Fano: Option 2

- Dictionnaire résultant :

Lettre	A	B	R	!	C	D
Nombre d'apparitions	5	2	2	2	1	1
Nombre de questions	2	2	2	3	4	4
Mot de code	11	10	01	001	0001	0000

- Nombre de bits utilisés pour encoder A B R A C A D A B R A !! :

$2 \cdot 9 + 3 \cdot 2 + 4 \cdot 2 = 32$ bits au total (donc $\frac{32}{13} \approx 2.46$ bits par lettre)

Remarque : Ici, le nombre total de bits utilisés est le même pour les deux options choisies, mais ce n'est pas le cas en général pour l'algorithme de Shannon-Fano.

Algorithme de Huffman

- David Albert Huffman (1925-1999)
  - ingénieur électricien, pionnier de l'informatique
  - élève de Robert Fano...

Repartons avec le tableau des nombres d'apparitions de chaque lettre :

Lettre	A	B	R	!	C	D
Nombre d'apparitions	5	2	2	2	1	1

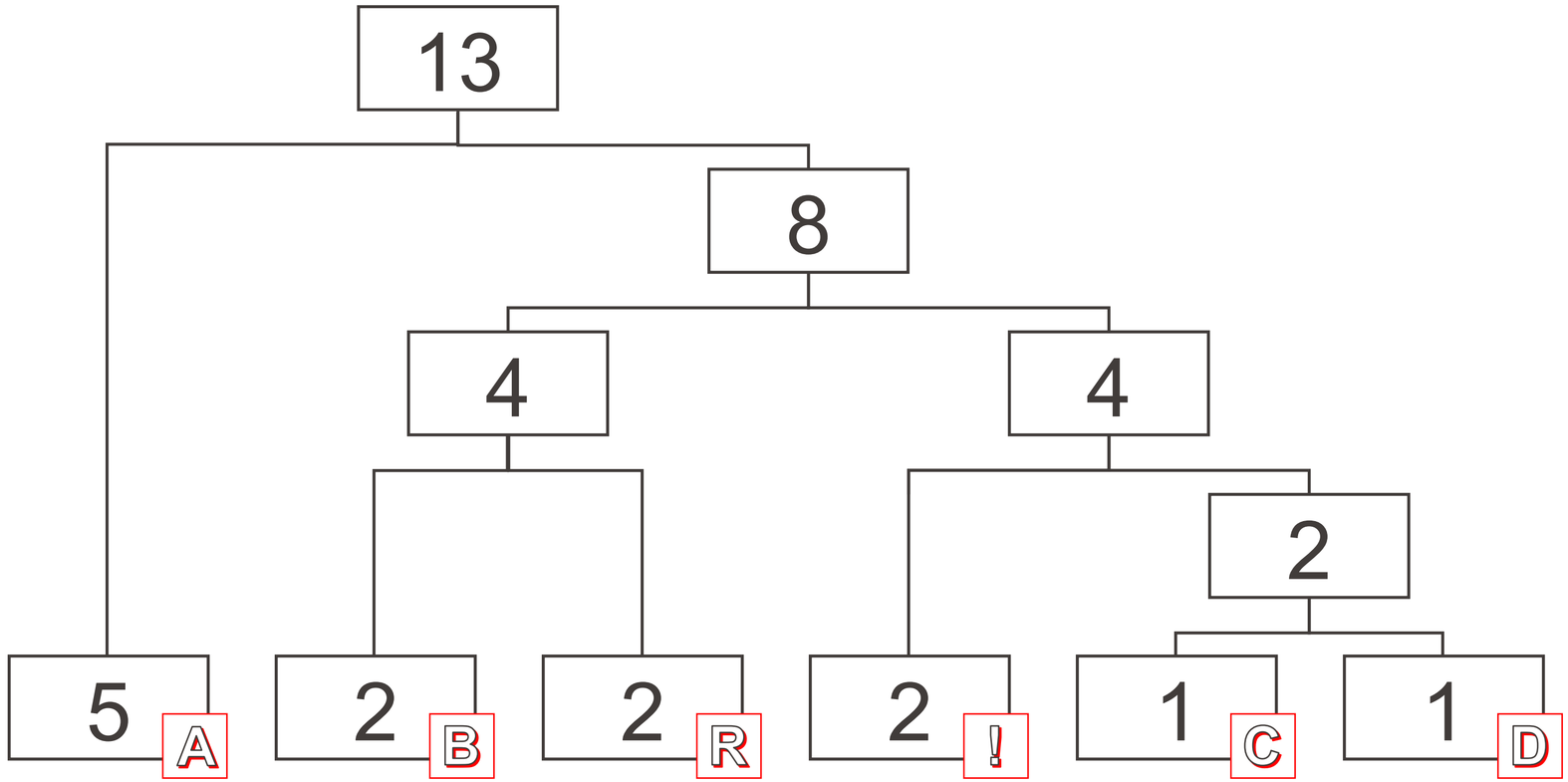
Idée principale : Au lieu de construire l'arbre depuis le haut, construisons-le depuis le bas !

- Voici l'algorithme :
  1. Regrouper les deux lettres les moins probables (ici, C et D)
  2. Considérer ces deux lettres comme une nouvelle "lettre" commune, et faire la somme des nombres d'apparitions (ici, $1 + 1 = 2$)
  3. Recommencer en 1. jusqu'à ce qu'il ne reste qu'une seule "lettre"

Algorithme de Huffman

A B R A C A D A B R A !!

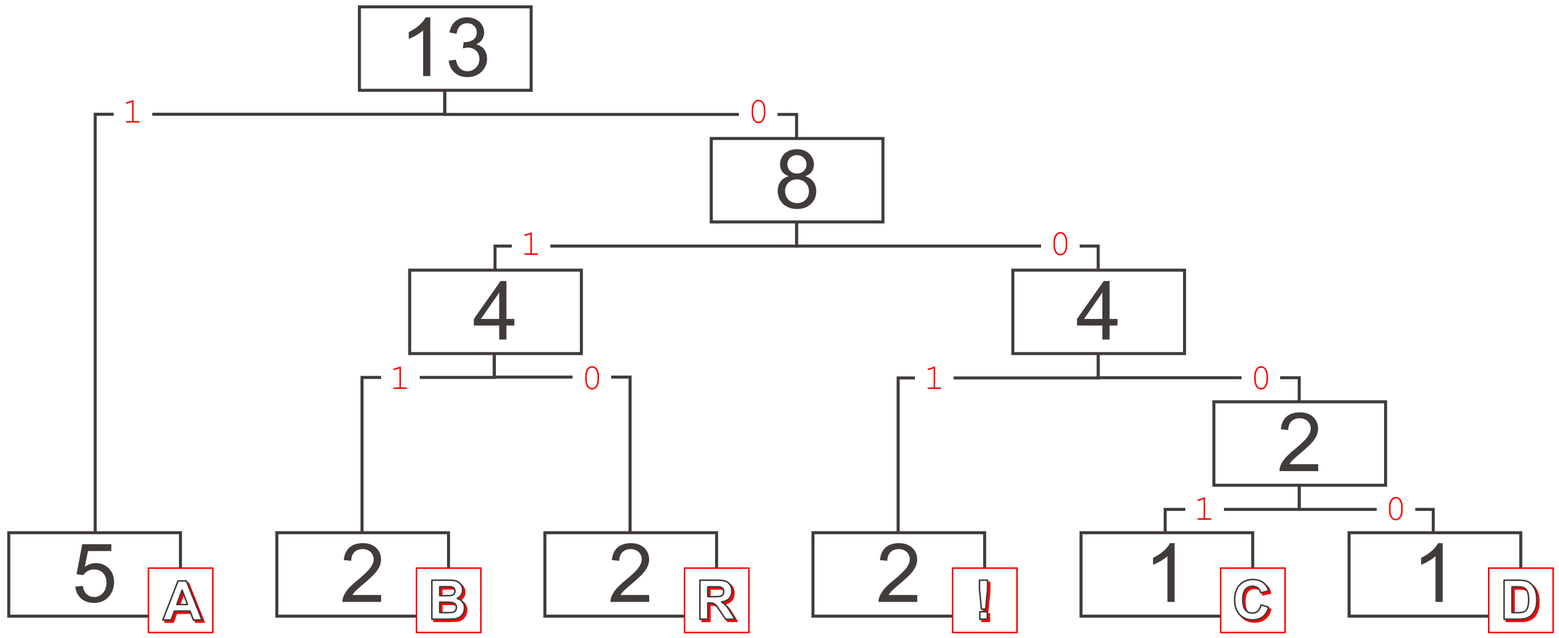
Lettre	A	B	R	!	C	D
Nombre d'apparitions	5	2	2	2	1	1



Algorithme de Huffman

A B R A C A D A B R A !!

Lettre	A	B	R	!	C	D
Nombre d'apparitions	5	2	2	2	1	1
Nombre de bits	1	3	3	3	4	4
Mot de code	1	011	010	001	0001	0000



Algorithme de Huffman

- Dictionnaire résultant:

Lettre	A	B	R	!	C	D
Nombre d'apparitions	5	2	2	2	1	1
Nombre de bits	1	3	3	3	4	4
Mot de code	1	011	010	001	0001	0000

- Nombre de bits utilisés pour encoder A B R A C A D A B R A !! :

$$1 \cdot 5 + 3 \cdot 6 + 4 \cdot 2 = 31 \text{ (donc } \frac{31}{13} \approx 2.38 \text{ bits par lettre)}$$

$$\rightarrow \text{mieux que Shannon-Fano !}$$

De plus :

- L'exécution de l'algorithme de Huffman est aussi plus rapide!
- En général, il y a aussi des choix à faire avec cet algorithme, mais le nombre total de bits utilisés est toujours le même.

Et l'entropie, dans tout ça ?

Lettre	A	B	R	!	C	D
Nombre d'apparitions	5	2	2	2	1	1
Probabilités	$\frac{5}{13}$	$\frac{2}{13}$	$\frac{2}{13}$	$\frac{2}{13}$	$\frac{1}{13}$	$\frac{1}{13}$

- L'entropie de la séquence A B R A C A D A B R A !! vaut :

$$H(X) = \frac{5}{13} \cdot \log_2\left(\frac{13}{5}\right) + 3 \cdot \frac{2}{13} \cdot \log_2\left(\frac{13}{2}\right) + 2 \cdot \frac{1}{13} \cdot \log_2\left(\frac{13}{1}\right)$$

$$= \log_2(13) - \frac{5}{13} \cdot \log_2(5) - \frac{6}{13} \approx 2.35$$

ce qui est légèrement plus petit que le nombre moyen de bits par lettre utilisés par chacun des deux algorithmes (respectivement 2.46 et 2.38).

Nous allons voir que c'est toujours le cas!

Information, Calcul et Communication

Théorème de Shannon

Olivier Lévêque

École polytechnique fédérale de Lausanne

Définitions

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

Exemple : $C = \{\texttt{11}, \texttt{10}, \texttt{01}, \texttt{001}, \texttt{0001}, \texttt{0000}\}$

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

Exemple : $l_4 = 3$

- 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 (si celle-ci se fait de gauche à droite).

Exemple : Avec le code binaire ci-dessus, la séquence `111001110000111` se lit `11,10,01,11,0001,11`

Définitions (suite)

- Le code binaire $C = \{c_1, \ldots, c_n\}$ peut être utilisé comme un dictionnaire pour représenter une séquence X formée avec des lettres tirées d'un alphabet $\mathcal{A} = \{a_1, \ldots, a_n\}$. Chaque lettre a_j est représentée par le mot de code c_j de longueur l_j.

Exemple :

Lettre	A	B	R	!	C	D
Mot de code	11	10	01	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

$$L(C) = p_1 \cdot l_1 + \cdots + p_n \cdot l_n$$

Théorème de Shannon

Quel que soit le code binaire C **sans préfixe** utilisé pour représenter une séquence X donnée, l'inégalité suivante est toujours vérifiée :

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

On voit apparaître ici un seuil (cf. théorème d'échantillonnage) : en-dessous de ce seuil, il n'est pas possible de compresser des données sans faire de pertes.

Inégalité de Kraft

Avant de démontrer le théorème de Shannon, nous avons besoin d'un lemme, **l'inégalité de Kraft :**

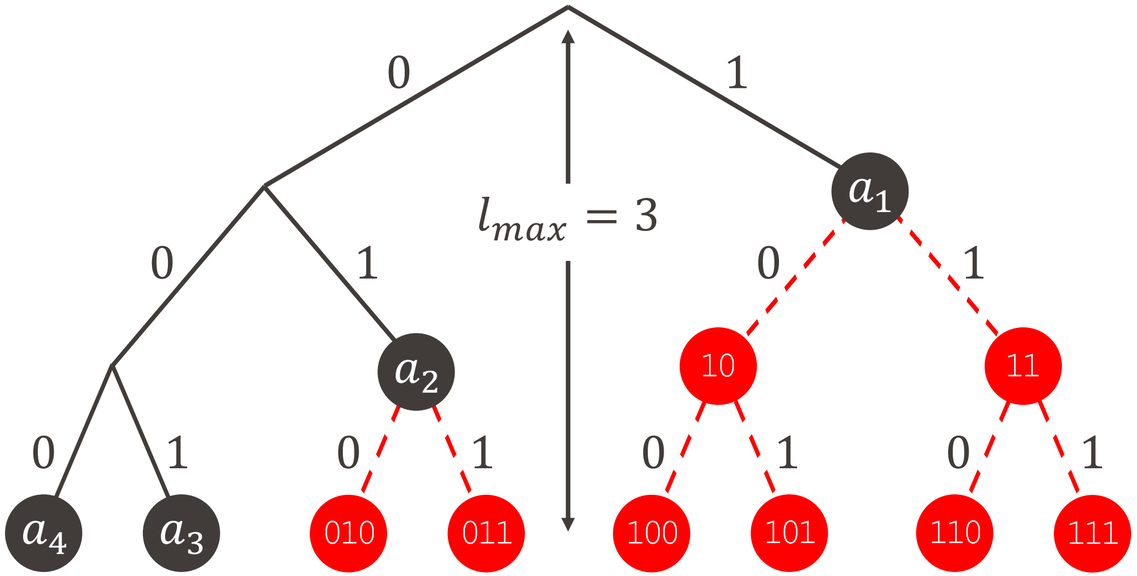
Soit $C = \{c_1, \dots, c_n\}$ un code binaire **sans préfixe**. Alors

$$2^{-l_1} + \cdots + 2^{-l_n} \leq 1$$

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

Démonstration de l'inégalité de Kraft

- Soit l_{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 l_{max}.
- On appelle "descendants" d'un mot de code donné les mots de code situés en dessous de celui-ci dans l'arbre.



Lettre a_j	Code c_j	Longueur l_j
a_1	1	1
a_2	01	2
a_3	001	3
a_4	000	3

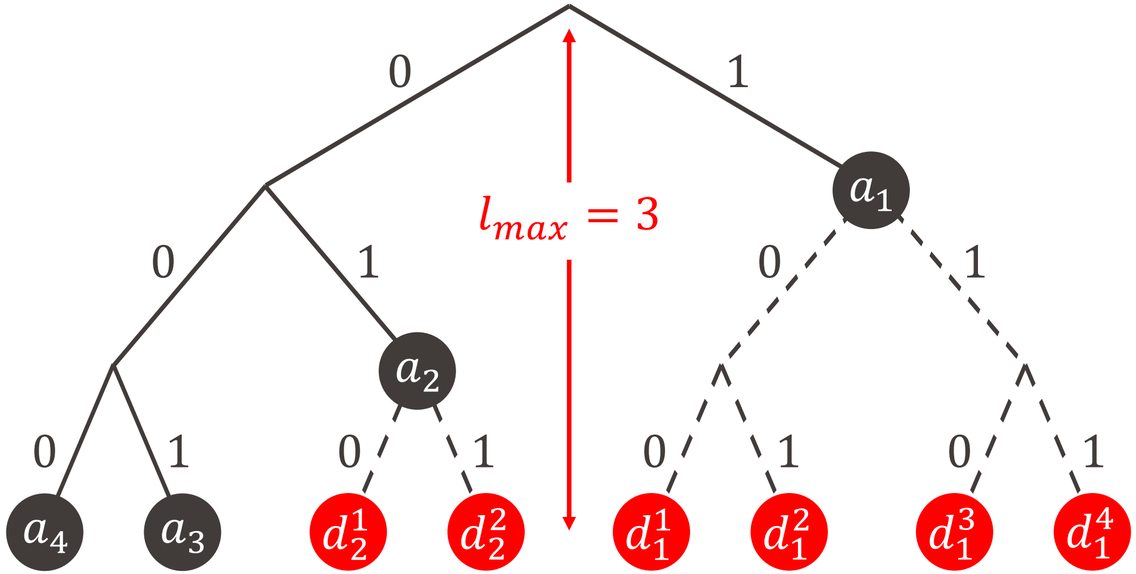
Démonstration de l'inégalité de Kraft

Observations:

- Au niveau l_{max}, il y a $2^{l_{max}}$ nœuds en tout.
- Le mot de code c_j a $2^{l_{max}-l_j}$ descendants d_j^i au niveau l_{max}.
- Les descendants de mots de code distincts sont tous distincts également.

Donc $2^{l_{max}-l_1} + \cdots + 2^{l_{max}-l_n} \leq 2^{l_{max}}$

- En divisant de chaque coté par $2^{l_{max}}$, on obtient l'inégalité de Kraft. ▨



Lettre a_j	Code c_j	Longueur l_j
a_1	1	1
a_2	01	2
a_3	001	3
a_4	000	3

Démonstration du théorème de Shannon

- Par définition,

$$H(X) - L(C) = \left(p_1 \cdot \log_2\left(\frac{1}{p_1}\right) + \cdots + p_n \cdot \log_2\left(\frac{1}{p_n}\right)\right) - (p_1 \cdot l_1 + \cdots + p_n \cdot l_n)$$

$$= p_1 \cdot \left(\log_2\left(\frac{1}{p_1}\right) - l_1\right) + \cdots + p_n \cdot \left(\log_2\left(\frac{1}{p_n}\right) - l_n\right)$$

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

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

$$\text{d'où } H(X) - L(C) = p_1 \cdot \log_2\left(\frac{2^{-l_1}}{p_1}\right) + \cdots + p_n \cdot \log_2\left(\frac{2^{-l_n}}{p_n}\right)$$

Démonstration du théorème de Shannon (suite)

- En utilisant le fait que $f(x) = \log_2(x)$ est une fonction **concave**, on obtient :

$$H(X) - L(C) = p_1 \cdot \log_2\left(\frac{2^{-l_1}}{p_1}\right) + \cdots + p_n \cdot \log_2\left(\frac{2^{-l_n}}{p_n}\right)$$

$$\leq \log_2\left(p_1 \cdot \frac{2^{-l_1}}{p_1} + \cdots + p_n \cdot \frac{2^{-l_n}}{p_n}\right) = \log_2(2^{-l_1} + \cdots + 2^{-l_n})$$

- Par l'inégalité de Kraft : $2^{-l_1} + \cdots + 2^{-l_n} \leq 1$ et le fait que $f(x) = \log_2(x)$ est une fonction croissante, on conclut que

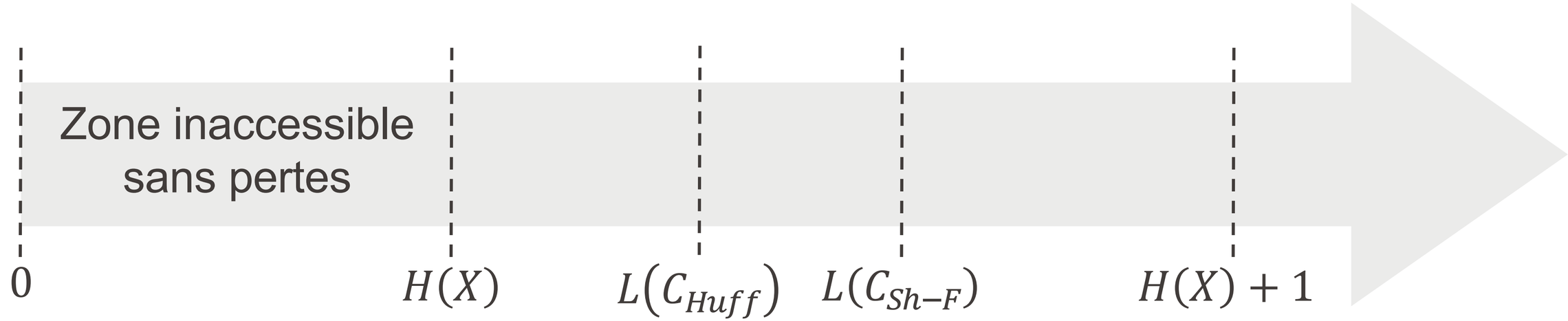
$$H(X) - L(C) \leq \log_2(1) = 0$$

- donc $H(X) \leq L(C)$, ce qui démontre le théorème de Shannon.

Quelques informations supplémentaires

On peut montrer de plus les inégalités suivantes :

- En général, $H(X) \leq L\left(C_{Huff}\right) \leq L(C_{Sh-F}) < H(X) + 1$:



- Pour **n'importe quel code binaire sans préfixe** C, on a de plus

$$L\left(C_{Huff}\right) \leq L(C)$$

ce qui veut dire que le code de Huffman est optimal !

Information, Calcul et Communication

Compression avec pertes

École polytechnique fédérale de Lausanne

Olivier Lévêque

Compression avec pertes

- Pour compresser des données sans pertes, on ne peut donc pas descendre en-dessous de la borne de Shannon.
- Si on essaye malgré tout en utilisant un algorithme du même type, on court à la catastrophe !
- Retournons à notre exemple A B R A C A D A B R A !! et essayons d'utiliser code binaire suivant (**avec** préfixe) :

Lettre	A	B	R	!	C	D
Mot de code	1	0	11	10	01	00

- Avec un tel code, on n'utilise que 19 bits au total, et non 31, mais la représentation binaire du message donne : `10111...`
- En lisant de gauche à droite, on ne peut pas décoder le début de la séquence: est-ce A B A A A ? A B R A ? A C R ? ou encore ! R A ? …

Compression avec pertes

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

1. lorsqu'on désire représenter un nombre réel avec un nombre fixé de bits (l'information comprise dans un nombre réel est "infinie")
2. lorsqu'on désire échantillonner un signal dont la bande passante est infinie
3. 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).

Dans le premier cas, on a vu la représentation en virgule flottante pour les nombres réels.

Comment procéder dans les deux derniers cas ?

- Même principe : appliquer un **filtre passe-bas** pour éliminer les hautes fréquences !

Compression avec pertes : Images

L'oreille humaine ne perçoit pas des sons au-delà de $\sim 20\ kHz$...

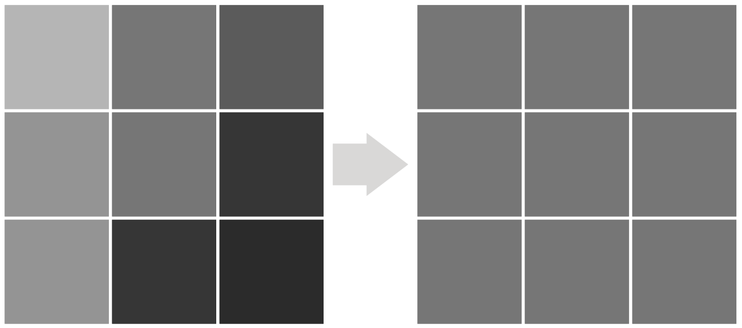
De même, l'œil humain a un pouvoir de résolution d'environ une minute d'arc = $\frac{1}{60} \sim 0.017$ degré, ce qui veut dire qu'il ne distingue pas :

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

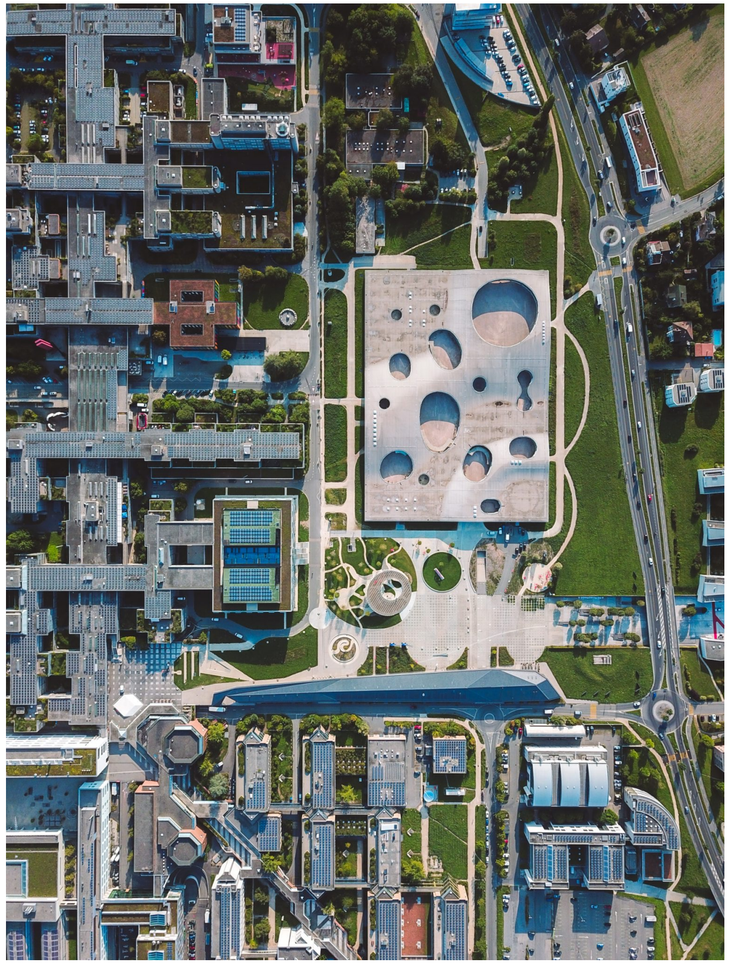
Comment filtrer les hautes fréquences (spatiales) dans une image ?

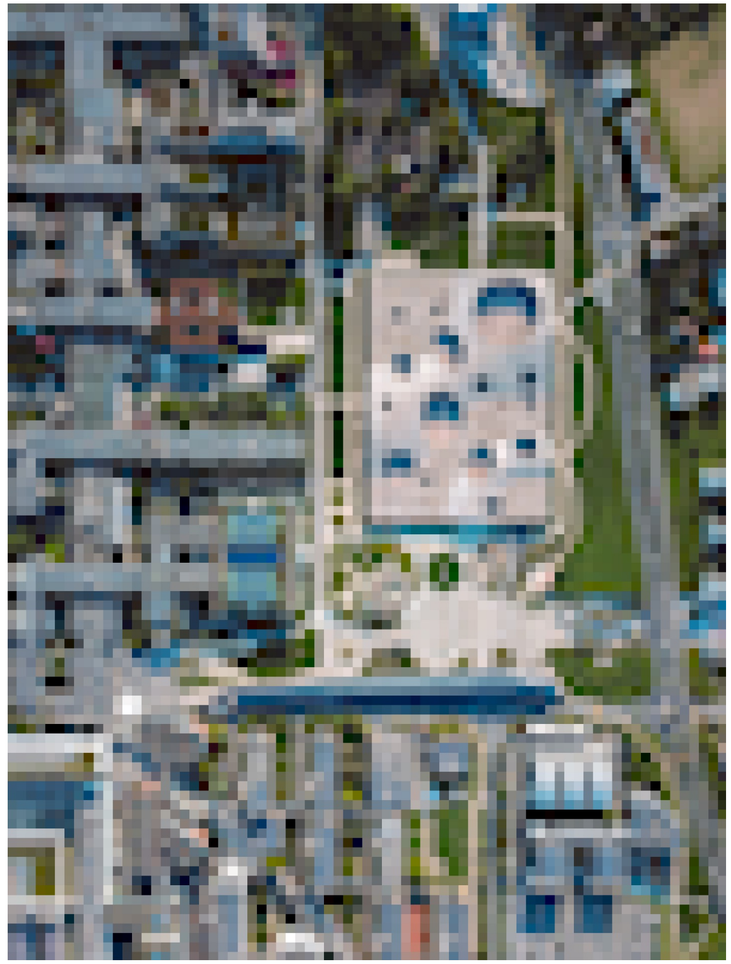
Une façon simple de faire : moyenner les couleurs sur des zones de plus ou moins grande taille.

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 l'image d'origine est déformée : on parle de **distorsion**.

Il existe bien sûr des algorithmes beaucoup plus sophistiqués pour compresser 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

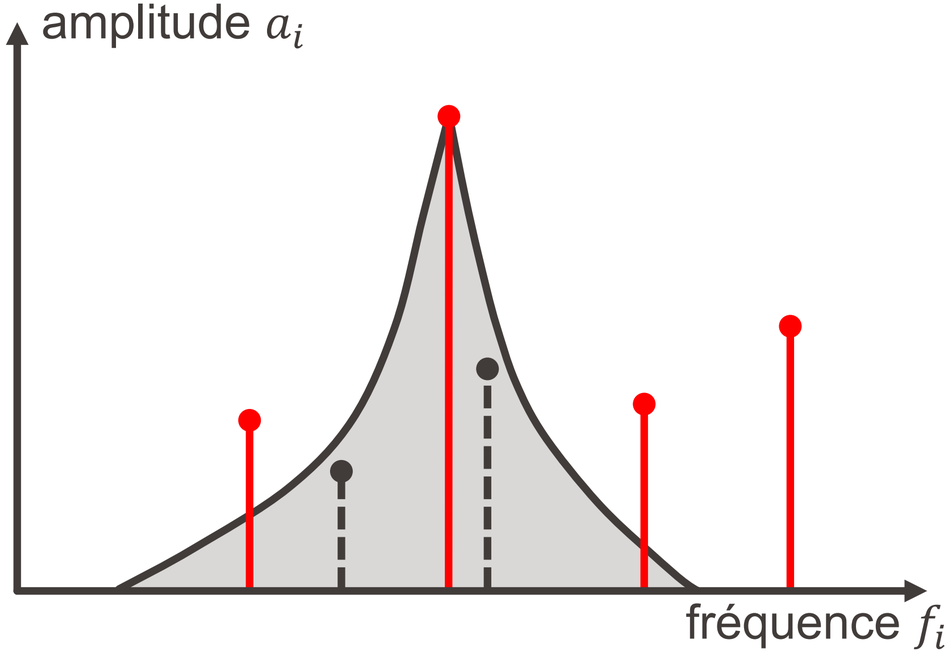
Et pour le son, comment procéder ?

- Pour rappel, le son enregistré sur un CD est d'abord filtré à $22\ kHz$, puis échantillonné à $44.1\ kHz$, et chaque échantillon est encodé sur $2 \cdot 16$ bits (2 canaux pour la stéréo).
- Pour enregistrer une seconde, il faut donc $44'100 \cdot 16 \cdot 2 \approx 1.4$ mégabits.
- Le format MP3 permet d'encoder cette information sur 128 kilobits seulement, ce qui correspond à une réduction d'environ 90% de la taille d'un fichier ! (sans déformation **sensiblement audible** du son)

Comment est-ce possible ?

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 toute façon pas !
- En se basant sur ce principe, Karlheinz Brandenburg et al. ont créé le format **MP3** en 1993.