L'information au cœur d'un monde numérique

Stockage, diffusion, protection

présenté par Matthieu Finiasz

ENSTA ParisTech

Qu'est-ce que l'information ?

- L'information existe sous de multiples formes :
  - tout ce que vous savez : souvenirs, connaissances...
  - tout ce que vous ressentez : ce que je dis, ce que vous voyez...
  - toutes les autres information qui ne vous parviennent pas : ce qui se passe dans la pièce à côté...

Qu'est-ce que l'information ?

- L'information existe sous de multiples formes :
  - tout ce que vous savez : souvenirs, connaissances...
  - tout ce que vous ressentez : ce que je dis, ce que vous voyez...
  - toutes les autres information qui ne vous parviennent pas : ce qui se passe dans la pièce à côté...

- Comme dans les autres sciences, sans définitions précises on ne peut pas raisonner sur l'information.

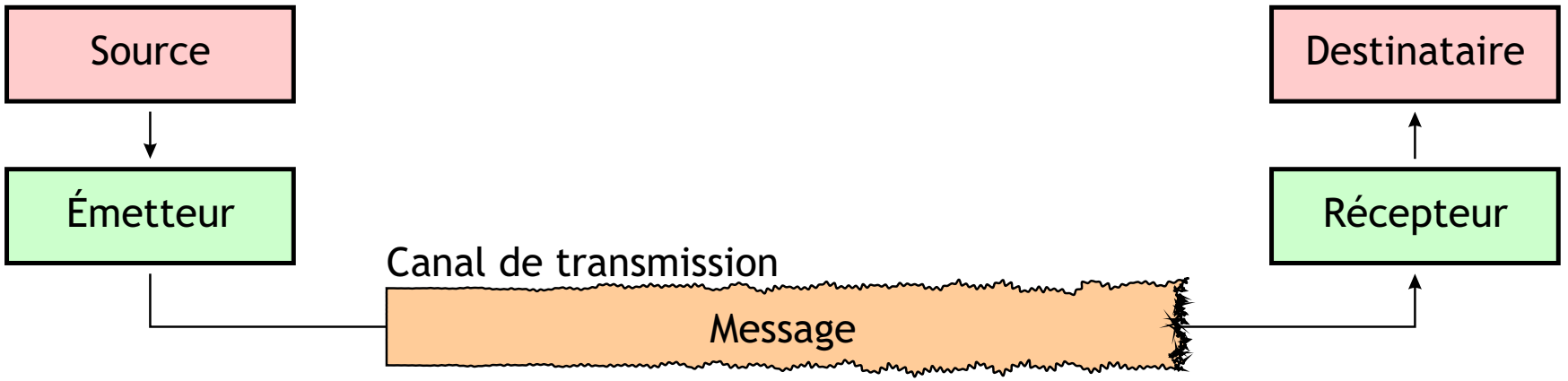
Qu'est-ce que l'information ?

- L'information existe sous de multiples formes :
  - tout ce que vous savez : souvenirs, connaissances...
  - tout ce que vous ressentez : ce que je dis, ce que vous voyez...
  - toutes les autres information qui ne vous parviennent pas : ce qui se passe dans la pièce à côté...

- Comme dans les autres sciences, sans définitions précises on ne peut pas raisonner sur l'information.

Il faut une théorie de l'information.

- Claude Shannon (1916-2001) "invente" la théorie de l'information en 1948 :
  - le bit est l'unité de mesure d'information
  - l'entropie mesure la quantité d'information (similaire à la thermodynamique)
  - donne un modèle de communication :



- Toute information peut être représentée par une séquence de bits,
  - le nombre de bits minimal pour la représenter est défini par l'entropie.

- Toute information peut être représentée par une séquence de bits,
  - le nombre de bits minimal pour la représenter est défini par l'entropie.

⚠ On ne représente pas juste une information, mais une information parmi un ensemble d'informations possibles.

- cet ensemble doit être dénombrable (en général fini),
- cet ensemble est muni d'une loi de probabilité.

- L'information est la réalisation d'une variable aléatoire.

Stockage de l'information

Comment représenter l'information ?

- On définit un alphabet Σ :
  - chaque lettre de Σ est une valeur possible d'une variable aléatoire

    → une information est représentée par une lettre de Σ.

Comment représenter l'information ?

- On définit un alphabet Σ :
  - chaque lettre de Σ est une valeur possible d'une variable aléatoire
  - → une information est représentée par une lettre de Σ.

Problème : l'alphabet Σ peut être immense.

Comment représenter l'information ?

- On définit un alphabet Σ :
  - ▷ chaque lettre de Σ est une valeur possible d'une variable aléatoire
  - → une information est représentée par une lettre de Σ.

Problème : l'alphabet Σ peut être immense.

- On choisit Σ plus petit et une information est décrite par un mot, une suite de lettres de Σ.
  - ▷ Par exemple : les lettres, la ponctuation et l'espace.

Codage binaire de l'information

- Il faut une façon standard de représenter l'information :
  - on se ramène à des chaînes de bits.
    - → toute information se manipule de la même façon.

- C'est ce que l'on appelle le codage de source.
- Cela comprend :
  - la transcription en binaire,
  - la compression des données.

- Supposons que Σ soit notre alphabet à 26 caractères.
  - on veut coder chaque lettre en binaire,
  - avec n bits on peut coder 2^n caractères
    → il faut 5 bits par caractère.

a	00000	n	01110
b	00001	o	01111
c	00010	p	10000
⋮	⋮	⋮	⋮
l	01100	y	11001
m	01101	z	11010

- Un mot de 10 lettres est codé en 50 bits.

- Supposons que Σ soit notre alphabet à 26 caractères.
  - ▷ on veut coder chaque lettre en binaire,
  - ▷ avec n bits on peut coder 2^n caractères
    - → il faut 5 bits par caractère.
- Cela marche bien, mais ce n'est pas optimisé :
  - ▷ certaines séquences de 5 bits ne sont pas utilisées,
  - ▷ on aurait pu coder 6 caractères de plus !
- On aurait donc pu coder plus d'information dans la même place.
  - → on aimerait avoir un alphabet d'exactement 2^n caractères.

- C'est ce que fait le codage ASCII des ordinateurs,
  - au départ sur 7 bits : 95 caractères + 33 de contrôle

USASCII code chart

b_7 b_6 b_5 →	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
Bits b_4 b_3 b_2 b_1 ↓	Column →	0	1	2	3	4	5	6	7
Row ↓									
0 0 0 0	0	NUL	DLE	SP	0	@	P	`	p
0 0 0 1	1	SOH	DC1	!	1	A	Q	a	q
0 0 1 0	2	STX	DC2	"	2	B	R	b	r
0 0 1 1	3	ETX	DC3	#	3	C	S	c	s
0 1 0 0	4	EOT	DC4	$	4	D	T	d	t
0 1 0 1	5	ENQ	NAK	%	5	E	U	e	u
0 1 1 0	6	ACK	SYN	&	6	F	V	f	v
0 1 1 1	7	BEL	ETB	'	7	G	W	g	w
1 0 0 0	8	BS	CAN	(	8	H	X	h	x
1 0 0 1	9	HT	EM	)	9	I	Y	i	y
1 0 1 0	10	LF	SUB	*	:	J	Z	j	z
1 0 1 1	11	VT	ESC	+	;	K	[	k	{
1 1 0 0	12	FF	FS	,	<	L	\	l	|
1 1 0 1	13	CR	GS	-	=	M	]	m	}
1 1 1 0	14	SO	RS	.	>	N	^	n	~
1 1 1 1	15	SI	US	/	?	O	_	o	DEL

- C'est ce que fait le codage ASCII des ordinateurs,
  - au départ sur 7 bits : 95 caractères + 33 de contrôle

- En informatique, le stockage se fait par octets → 8 bits
  - divers codages sur 8 bits (128 caractères de plus) :
    - ISO-8859-1 (latin 1) avec les accents européens
    - ISO-8859-5 avec les caractères cyrilliques
    - ISO-8859-15 (latin 9) avec œ, š, ž, ÿ, l'euro...
  - des codage sur plusieurs octets :
    - UTF-8 avec tous les caractères asiatiques, arabes...
    - d'autres standards moins courants.

- Hormis les problèmes de standards, cela fonctionne bien.

- En revanche ce n'est pas très efficace...
  - on peut coder un hiéroglyphe aussi bien qu'un 'e',
  - on se retrouve avec des codages sur 16 ou 32 bits,
  - le codage sur 5 bits était presque suffisant...
    → on utilise 3 à 6 fois plus de bits !

- Hormis les problèmes de standards, cela fonctionne bien.

- En revanche ce n'est pas très efficace...
  - on peut coder un hiéroglyphe aussi bien qu'un 'e',
  - on se retrouve avec des codages sur 16 ou 32 bits,
  - le codage sur 5 bits était presque suffisant...
    → on utilise 3 à 6 fois plus de bits !

- Si on connaît les probabilités de chaque lettre, on peut coder plus efficacement.
  → codage court pour les lettres les plus fréquentes.

- On suppose que l'alphabet Σ contient s caractères σ_i pour $i \in [0, s-1]$.
  - Le caractère σ_i apparaît avec probabilité p_i.
  - Le taux d'entropie moyen par caractère :

$$T_H = -\sum_{i=0}^{s-1} p_i \log_2(p_i).$$

- C'est le nombre de bits d'information que donne, en moyenne, chaque caractère.

- On suppose que l'alphabet Σ contient s caractères σ_i pour $i \in [0, s-1]$.
  - Le caractère σ_i apparaît avec probabilité p_i.
  - Le taux d'entropie moyen par caractère :

$$T_H = -\sum_{i=0}^{s-1} p_i \log_2(p_i).$$

- Si $s = 2^n$ et $p_i = \frac{1}{2^n}$ (distribution uniforme) on trouve :

$$T_H = -\sum_{i=0}^{2^n-1} \frac{1}{2^n} \times (-n) = n.$$

- Cela correspond au maximum possible pour $s = 2^n$.

Fréquence des lettres en français

Un exemple de taux d'entropie

A	9,42%	B	1,02%	C	2,64%	D	3,39%	E	15,87%
F	0,95%	G	1,04%	H	0,77%	I	8,41%	J	0,89%
K	0,00%	L	5,34%	M	3,24%	N	7,15%	O	5,14%
P	2,86%	Q	1,06%	R	6,46%	S	7,90%	T	7,26%
U	6,24%	V	2,15%	W	0,00%	X	0,30%	Y	0,24%
Z	0,32%								

- Cette distribution donne un taux d'entropie $T_H = 2{,}63$.
  - une répartition uniforme donnerait un taux de 4,7.

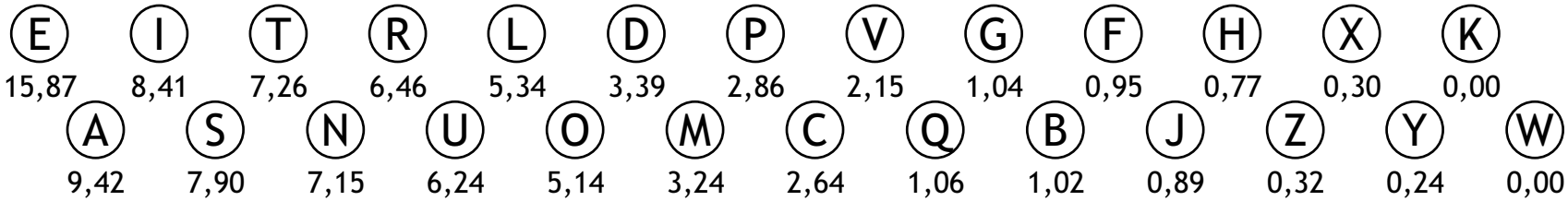
Fréquence des lettres en français

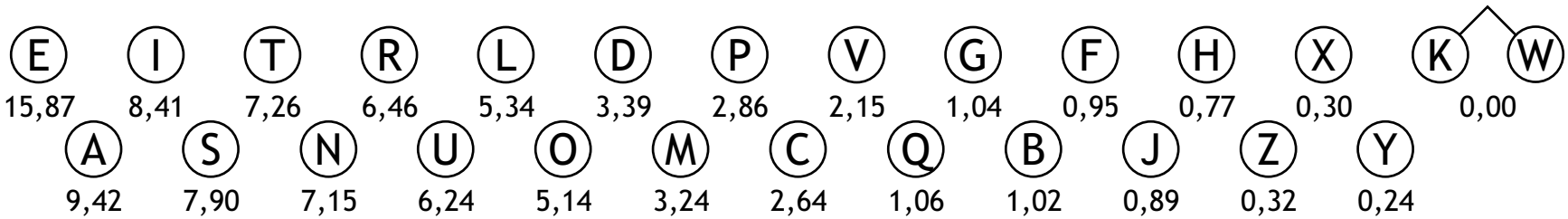
Un exemple de taux d'entropie

A	9,42%	B	1,02%	C	2,64%	D	3,39%	E	15,87%
F	0,95%	G	1,04%	H	0,77%	I	8,41%	J	0,89%
K	0,00%	L	5,34%	M	3,24%	N	7,15%	O	5,14%
P	2,86%	Q	1,06%	R	6,46%	S	7,90%	T	7,26%
U	6,24%	V	2,15%	W	0,00%	X	0,30%	Y	0,24%
Z	0,32%								

- Cette distribution donne un taux d'entropie $T_H = 2{,}63$.
  - une répartition uniforme donnerait un taux de 4,7.
- On cherche un codage qui utilise en moyenne 2,63 bits par caractère d'entrée.

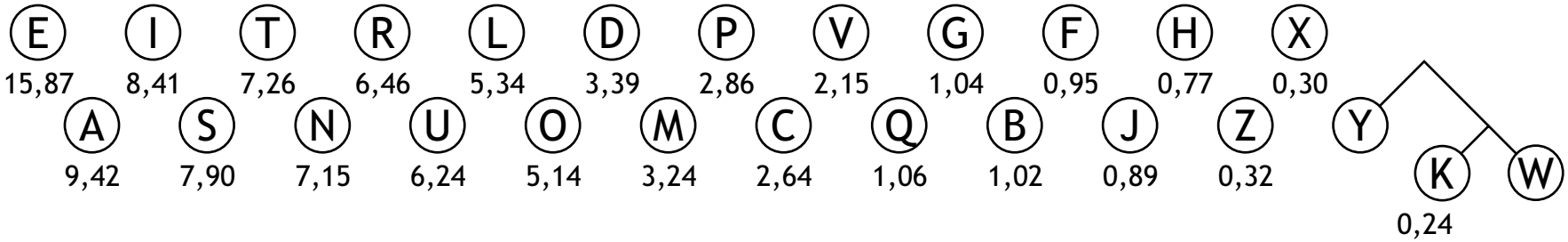
- On va ranger les lettre dans un arbre en fonction de leur probabilité :
  - plus une lettre est rare, plus elle loin dans l'arbre,
  - plus elle est courante, plus elle sera proche de la racine.
- Le codage d'une lettre dépend de sa position dans l'arbre.


E I T R L D P V G F H X K
15,87 8,41 7,26 6,46 5,34 3,39 2,86 2,15 1,04 0,95 0,77 0,30 0,00
A S N U O M C Q B J Z Y W
9,42 7,90 7,15 6,24 5,14 3,24 2,64 1,06 1,02 0,89 0,32 0,24 0,00


E I T R L D P V G F H X K W
15,87 8,41 7,26 6,46 5,34 3,39 2,86 2,15 1,04 0,95 0,77 0,30 0,00
A S N U O M C Q B J Z Y
9,42 7,90 7,15 6,24 5,14 3,24 2,64 1,06 1,02 0,89 0,32 0,24

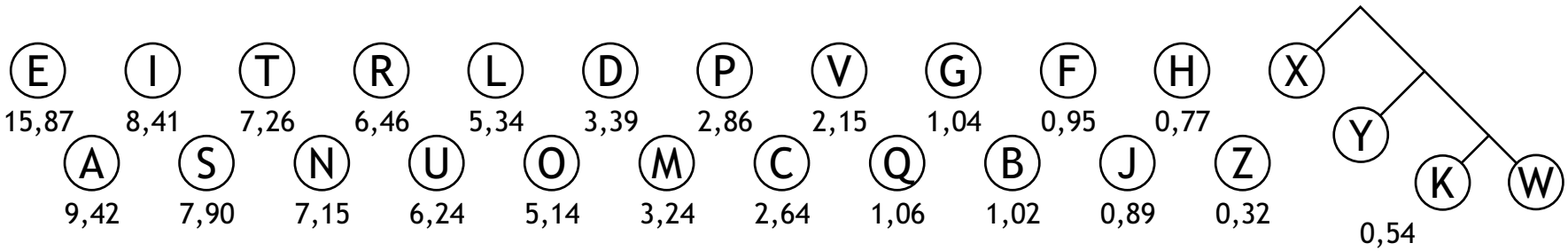
Les arbres de Huffman

Construction



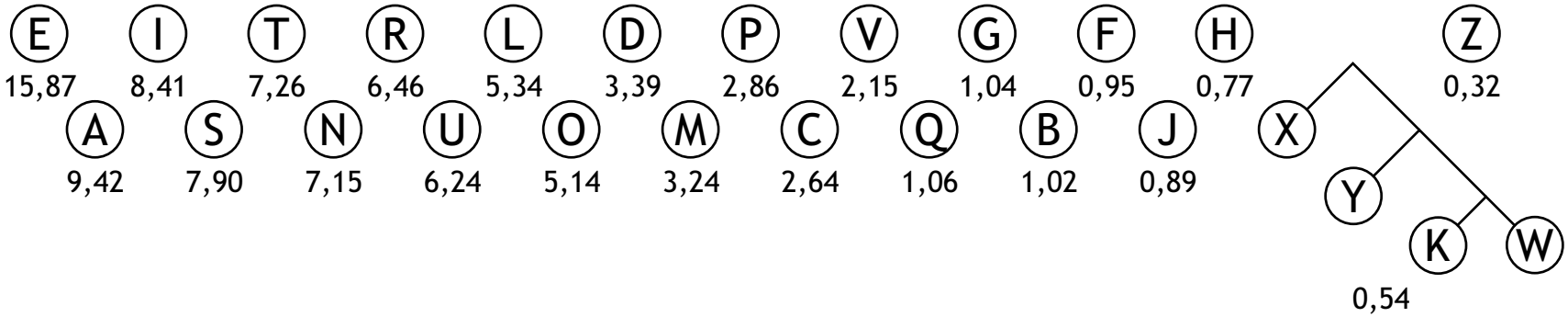
Les arbres de Huffman

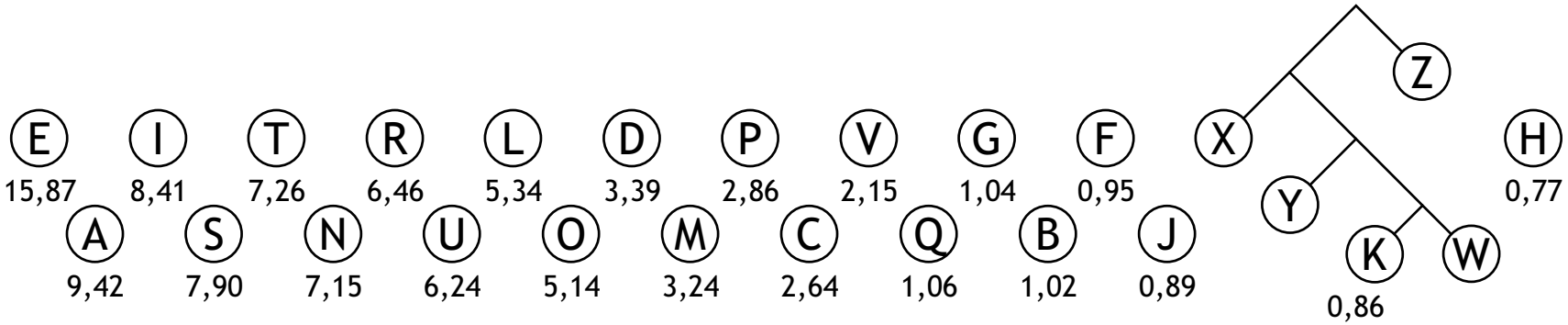
Construction



Les arbres de Huffman

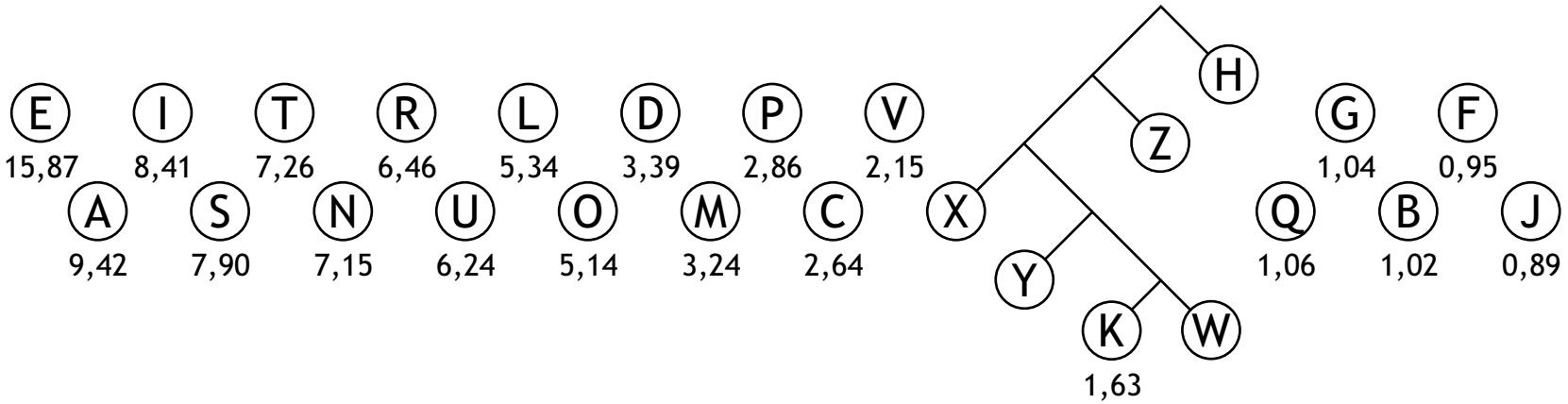
Construction




E
15,87
I
8,41
T
7,26
R
6,46
L
5,34
D
3,39
P
2,86
V
2,15
G
1,04
F
0,95
A
9,42
S
7,90
N
7,15
U
6,24
O
5,14
M
3,24
C
2,64
Q
1,06
B
1,02
J
0,89
X
Z
H
0,77
Y
K
0,86
W

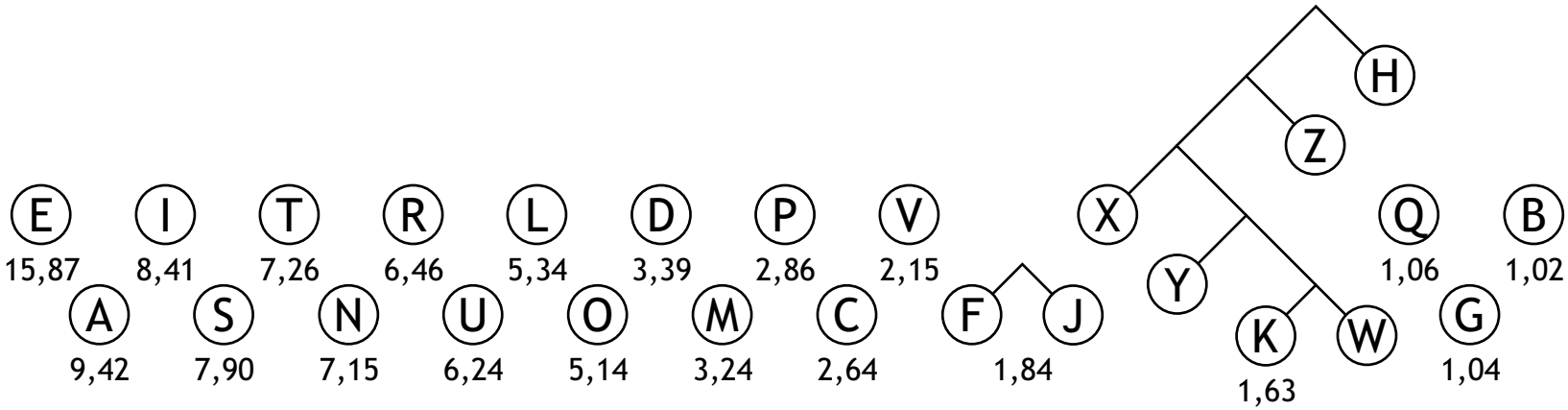
Les arbres de Huffman

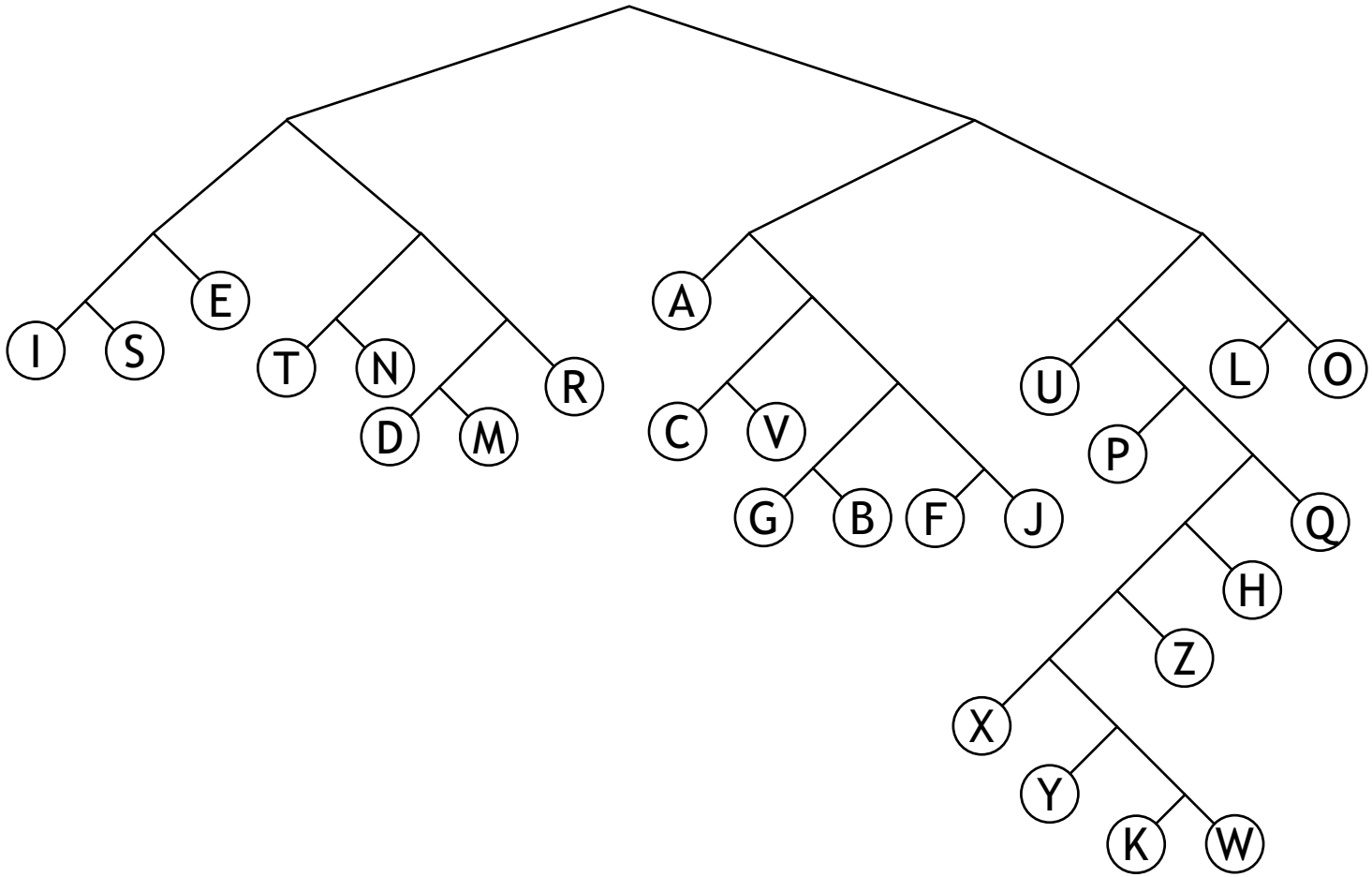
Construction



Les arbres de Huffman

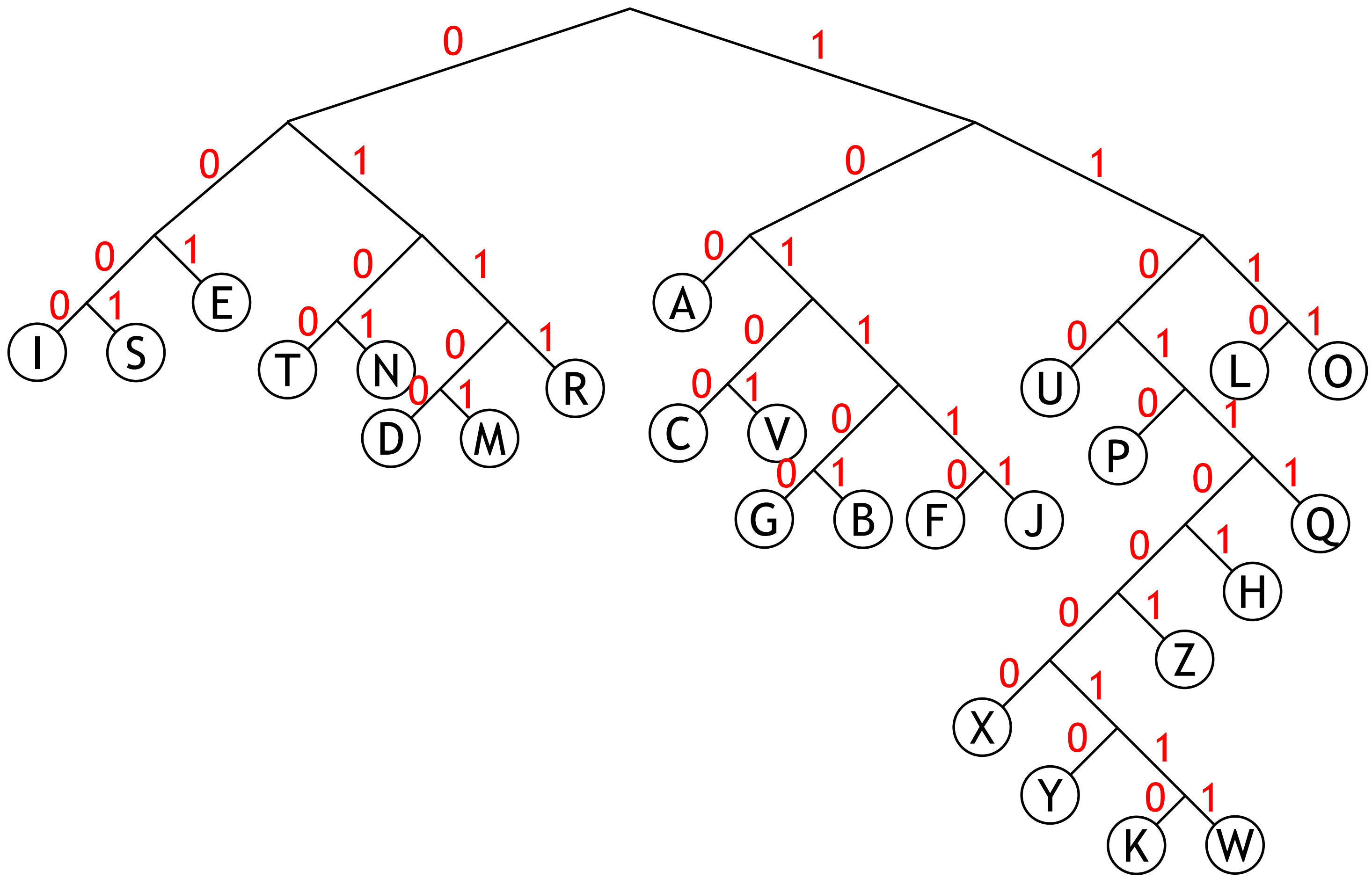
Construction



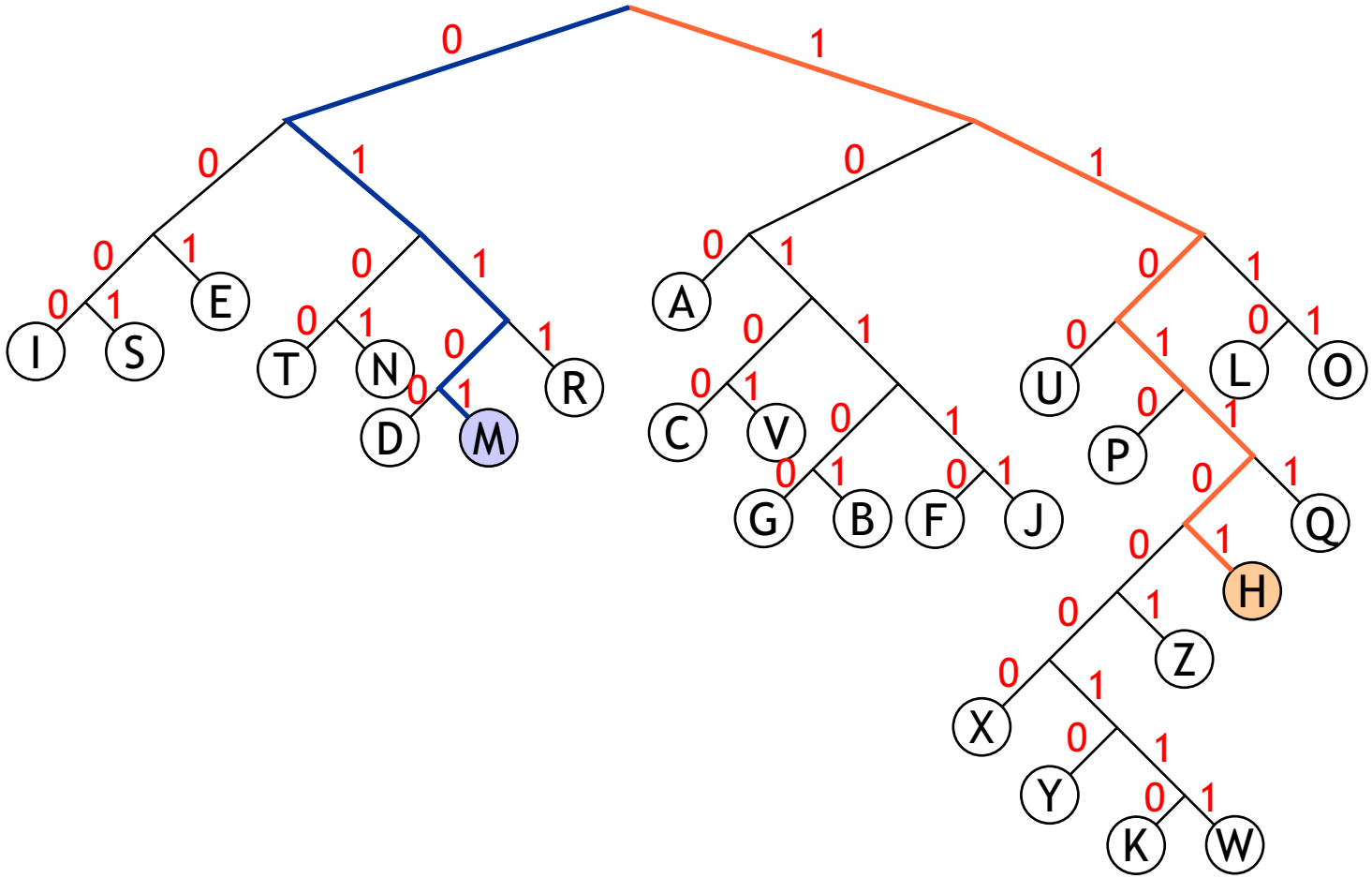


- L'arbre est construit après 25 étapes.

Les arbres de Huffman
Utilisation



- On attribue ensuite un bit à chaque arrête
  → cela détermine le codage.



- M → 01101, H → 1101101, E → 001...

- Cela donne un codage binaire optimal de l'alphabet,
  - le nombre de bits moyens n'est égal à T_H que si tous les p_i sont de la forme $\frac{1}{2^j}$.
  - dans notre cas, on arrive environ à 4 bits en moyenne.

- L'arbre donne aussi un décodeur efficace :
  - on lit les bits reçus un par un en descendant dans l'arbre.

- Cela permet de stocker un texte sur moins de bits que le codage direct, mais cette solution n'est pas encore parfaite...

- On peut regarder plusieurs caractères à la fois :
  - en conservant des probabilités par caractère
    → l'arbre est plus gros, mais le codage meilleur.
  - en regardant la probabilité réelle de chaque séquence
    → on obtient un taux d'entropie plus faible
    → on obtient aussi un meilleur codage.

⚠ La formule de taux d'entropie n'est vraie que si les caractères sont mutuellement indépendants !

- On peut regarder plusieurs caractères à la fois :
  - en conservant des probabilités par caractère
    → l'arbre est plus gros, mais le codage meilleur.
  - en regardant la probabilité réelle de chaque séquence
    → on obtient un taux d'entropie plus faible
    → on obtient aussi un meilleur codage.

⚠ La formule de taux d'entropie n'est vraie que si les caractères sont mutuellement indépendants !

- Mesurer la quantité d'information dans un texte n'est pas simple :
  - savoir si un codage est bon ne l'est pas non plus...

Algorithmes de compression génériques

- On considère maintenant un alphabet binaire et un message quelconque de n bits.
- Si le message est un texte codé en ASCII on doit aussi pouvoir utiliser un arbre de Huffman
  - ▷ on construit un arbre spécifique au message,
  - ▷ on stocke l'arbre en début de message.

- ◇ Choisir une taille de bloc,
- ◇ couper le message en blocs,
- ◇ compter les occurrences de chaque bloc,
- ◇ construire un arbre de Huffman et coder les blocs avec,
- ◇ stocker le tout.

Algorithmes de compression génériques

- On considère maintenant un alphabet binaire et un message quelconque de n bits.
- Si le message est un texte codé en ASCII on doit aussi pouvoir utiliser un arbre de Huffman
  - ▷ on construit un arbre spécifique au message,
  - ▷ on stocke l'arbre en début de message.

- Pose plusieurs problèmes :
  - ▷ il faut essayer plusieurs tailles de bloc,
  - ▷ mal adapté aux messages dont la distribution évolue,
  - ▷ utilise une seule taille de bloc à la fois.

Les algorithmes de Lempel-Ziv

Publiés en 1977 et 1978

- Deux idées différentes :
  - LZ77 : utilise une fenêtre coulissante
    → stocker la longueur et la position de la dernière fois que l'on a vu une séquence,
  - LZ78 : utilise un dictionnaire
    → ajouter au dictionnaire chaque fois que l'on rencontre une nouvelle séquence.

- Le codage dépend de ce qui a déjà été lu :
  → le décodeur apprend à décoder en décodant.

- Il existe de multiples variantes des algorithmes LZ :
  - LZW, LZMA, LZO...
    - → toutes basées sur les mêmes idées, mais optimisées.
  - DEFLATE : mélange de LZ77 et de Huffman.

- Utilisés dans de nombreux formats :
  - données : zip, gzip, rar...
  - images : gif, png

La transformée de Burrows-Wheeler

Publiée en 1994

- Réordonne simplement les caractères d'un message :
  - ▷ permet de faciliter la compression du message
  - ▷ est inversible !

- La transformation rapproche des motifs similaires, mêmes distants
  - → les algorithmes LZ fonctionnent mieux

- Malheureusement, cela prend du temps :
  - ▷ utilisé dans bzip2 avec un codage de Huffman.

Un paradoxe de la compression

Un problème de taille !

- Considérons tous les messages de n bits notés m_i pour $i \in [0, 2^n - 1]$.

- Si f est une fonction de compression $|f(m_i)|$ est la longueur de m_i une fois compressé.
  - ▷ les 2^n messages compressés $f(m_i)$ sont différents :
    - → certains font plus de n bits de long.
  - ▷ on peut même prouver qu'en moyenne $|f(m_i)| \geq n$.

- En moyenne un zip est plus gros que le fichier source...
  - ▷ heureusement ça n'arrive jamais en pratique.

Compression de fichiers multimédia

- Audio, image, vidéo...
  - ce sont des données de grande taille
    - → il faut les compresser efficacement.
- Ce sont justement les cas où les algorithmes classiques marchent mal
  - les données sont trop proches de l'aléa

Compression de fichiers multimédia

- Audio, image, vidéo...
  - ce sont des données de grande taille
    - → il faut les compresser efficacement.
- Ce sont justement les cas où les algorithmes classiques marchent mal
  - les données sont trop proches de l'aléa
- Deux solutions :
  - des algorithmes spécifiques
    - → FLAC pour l'audio, utilise de la prédiction
  - supprimer (discrètement) de l'information
    - → compression à perte.

Compression à perte

jpeg, mp3, mpeg...

- Un fichier multimédia contient beaucoup d'information
  - peut-on l'assimiler entièrement ?
  - quelle information supprimer ?
- Tous les bits n'ont pas la même importance.
- En général on passe par une transformée de Fourier :
  - représentation spectrale (en fréquence)
    - → on simplifie le spectre
- Compromis entre le taux de compression et la fidélité par rapport à l'original :
  - les premiers bits se remarquent moins que les derniers.

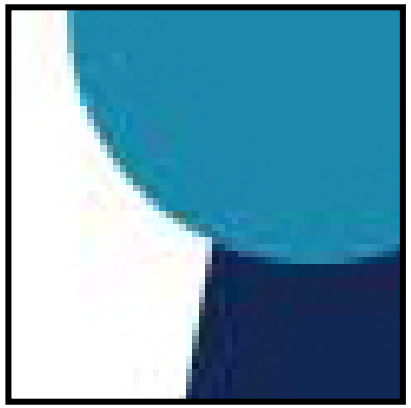
Exemple de compression à perte

Compression d'image en jpeg

55 kB

15 kB

9 kB



Exemple de compression à perte
Compression d'audio en GSM

- Les téléphones portables peuvent transmettre une quantité d'information très limitée :
  - la voix est compressée avec beaucoup de pertes
  - pour une seconde de son :
    - CD $\rightarrow$ 1 411 200 bits de données
    - GSM $\rightarrow$ 13 000 bits de données
    - MP3 $\rightarrow$ $\sim$160 000 bits de données.
- La compression GSM date de 1990 :
  - elle doit être très simple...
  - supprime les fréquences plus hautes que la voix
  - simplifie le spectre en donnant des priorités
    $\rightarrow$ certains sons (sirène, cloche) écrasent la voix.

Diffusion de l'information

Diffusion de l'information

- Pour envoyer une information à un destinataire :
  - la source code l'information dans un message,
  - le message est envoyé au destinataire,
  - le destinataire extrait l'information du message.
- Problème : on veut pouvoir garantir que le destinataire reçoit bien toute l'information.
  → Comment faire ?

Principe général de la correction d'erreur

- Si on modifie des lettres d'une phrase, on arrive encore à la comprendre :

  "La tléorie de l'informttizn c'esr amusaft."

Principe général de la correction d'erreur

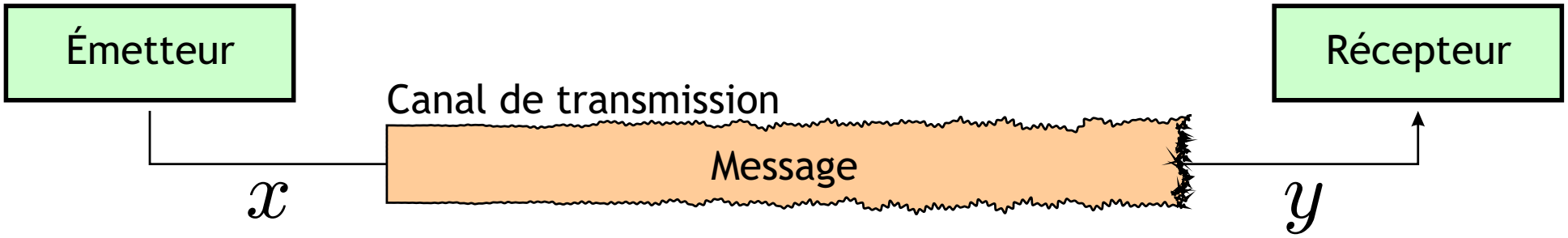
- Si on modifie des lettres d'une phrase, on arrive encore à la comprendre :

  "La tléorie de l'informttizn c'esr amusaft."

  - La phrase est plus longue que la quantité d'information qu'elle contient.
    → le français crée des redondances.
- C'est grâce à cela que l'on comprend :
  - un texte mal écrit à la main,
  - quelqu'un qui parle avec du bruit.

Le cerveau parvient à corriger les erreurs pour extraire l'information.

- Pour formaliser cela, on définit un canal :



- Quand x est envoyé, on reçoit y,
  - ▷ s'il y a du bruit, $y \neq x$.
- Le canal est définit par l'ensemble des probabilité $P(y|x)$ de recevoir y quand x est envoyé.
- Par exemple, le canal binaire symétrique :
  - ▷ $P(0|0) = P(1|1) = 1 - p$ et $P(0|1) = P(1|0) = p$.

Quelques exemple de canaux

- À chaque modèle de bruit correspond un canal :

  - canal Gaussien $\rightarrowtriangle$ $x, y \in \mathbb{R}$, $y - x$ suit une distribution Gaussienne centrée en 0.

  - canal à effacement $\rightarrowtriangle$ $x \in \{0, 1\}$, $y \in \{0, 1, e\}$ et soit $y = x$, soit $y = e$.

  - canaux à mémoire $\rightarrowtriangle$ les probabilités évoluent en fonction de ce qui est transmis
    $\longrightarrow$ paquets d'erreur : la probabilité d'erreur est plus grande si une erreur vient d'avoir lieu.

Capacité d'un canal

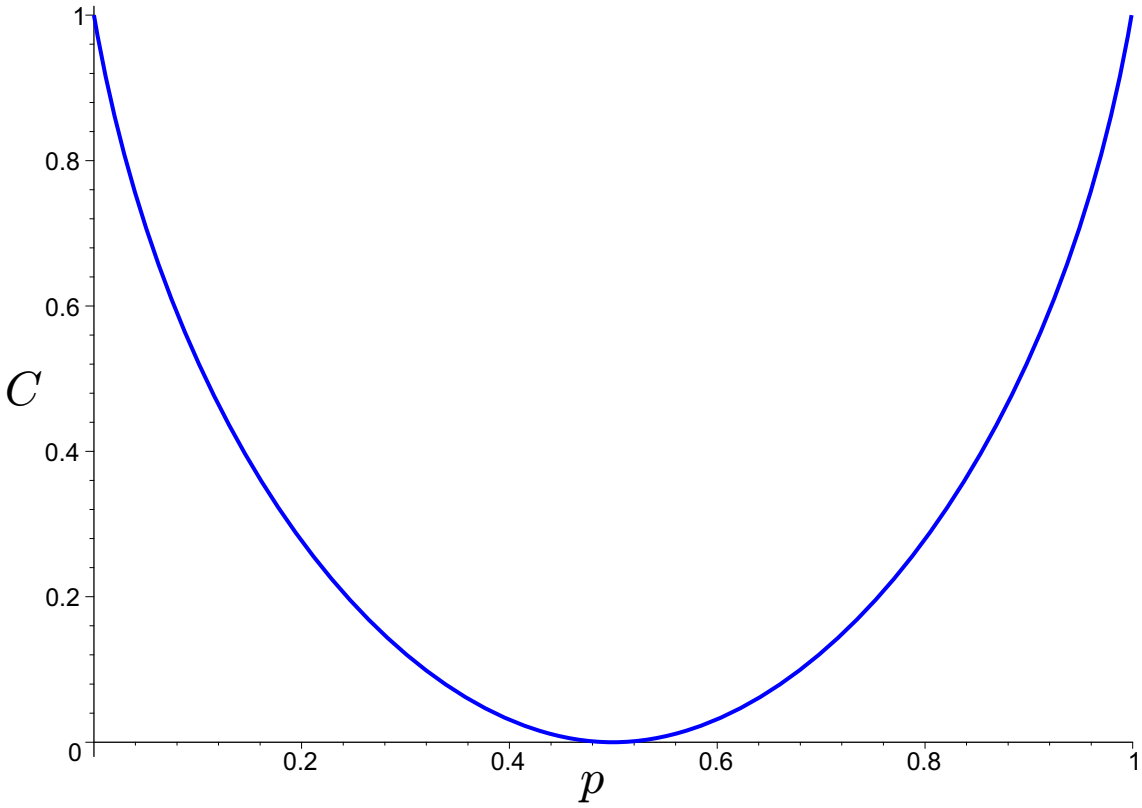
- On définit la notion de capacité de canal
  - cela correspond à la quantité maximale d'information qui peut transiter par la canal (par symbole transmis)
  - c'est la quantité d'information que l'on a sur x en recevant y

    → l'information mutuelle entre x et y.

$$C = \sum_{x,y} P(y|x)P(x)\log_2 \frac{P(y|x)}{P(y)}.$$

Capacité du canal binaire symétrique

- $C = 0$ pour $p = \frac{1}{2}$
- $C = 1$ pour $p = 0$ ou $p = 1$



$$C(p) = 1 + p \log_2 p + (1-p) \log_2 (1-p).$$

- Pour un canal de capacité C et un taux d'information $R = \frac{k}{n}$ tels que $R < C$, il existe une méthode pour coder k bits d'information sur n bits, transmettre les n bits sur le canal et retrouver (avec probabilité 1) les k bits d'information à partir de la sortie du canal.

⚠ Cela ne nous donne pas la méthode à utiliser...

- Réciproquement, si $R > C$, quel que soit le codage utilisé, on ne pourra jamais retrouver les k bits d'information envoyés.

Les codes correcteurs d'erreurs

- On veut coder k bits en n bits :
  - ▷ on part de k bits d'information (un texte compressé...),
  - ▷ on construit $n - k$ bits de redondance,
  - ▷ on transmet le tout sur le canal,
  - ▷ on décode les n bits bruités reçus.

- C'est le travaille des codes correcteurs d'erreurs :
  - ▷ définir comment calculer la redondance,
  - ▷ donner un algorithme de décodage (efficace et fiable).

- Un code bien adapté à un canal permet de corriger toutes les erreurs.

Quelques exemple de codes simple

- Le code à répétition d'ordre 3 : $R = \frac{1}{3}$
  - → 0 ⇀ 000 et 1 ⇀ 111.
  - ▷ corrige une erreur par bloc de 3 bits transmis,
  - ▷ se trompe s'il y a deux erreurs.

- Le code de parité de longueur 8 : $R = \frac{7}{8}$
  - → 0110110 ⇀ 01101100 ou 1001100 ⇀ 10011001
    le bit ajouté est la somme des 7 autres (modulo 2).
  - ▷ détecte une erreur, mais ne corrige rien,
  - ▷ utilisable si on peut demander une réémission.

- Une extension est le CRC (Cyclic Redundancy Check)
  - ▷ utilisé par exemple pour vérifier les paquets Ethernet.

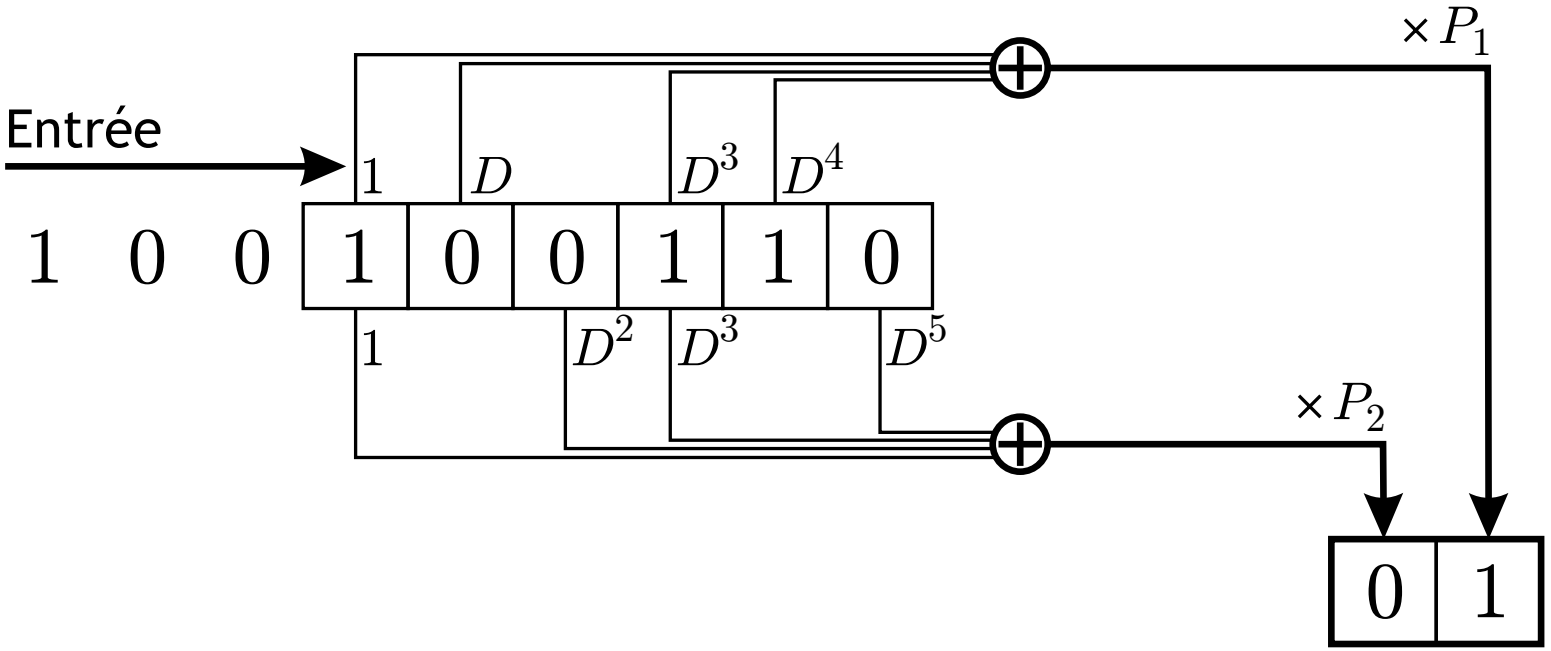
Les grandes familles de codes

- Les codes convolutifs :
  - l'information est codée bit par bit,
  - le codeur a une mémoire interne.

- Les codes en blocs :
  - le message est coupé en blocs,
  - les blocs sont codés indépendamment.

- Les codes concaténés :
  - le message est d'abord codé avec un code en bloc,
  - il est recodé avec un code convolutif.

- Un registre à décalage :
  - un bit rentre à chaque étape,
  - 2 (ou plus) bits de sortie sont déduits de l'état.



Les codes convolutifs

- Un registre à décalage :
  - un bit rentre à chaque étape,
  - 2 (ou plus) bits de sortie sont déduits de l'état.

- Nécessitent peu de calculs :
  - idéaux pour les systèmes embarqués,

- Corrigent bien les erreurs isolées
  - jusqu'à 3-4 % de bruit pour un taux $\frac{1}{2}$,
  - mais on ne sait pas si on se trompe.

Les turbocodes

Berrou, Glavieux et Thitimajshima - 1993

- Variante de codes convolutifs de taux $\frac{1}{2}$:
  - on code l'information d'une part,
  - on code une permutation de l'information d'autre part,
  - la sortie est un entrelacement des deux.
- Parmi les meilleurs codes binaires à l'heure actuelle :
  - corrigent jusqu'à 10 % d'erreurs pour un taux $\frac{1}{2}$,
  - la limite de Shannon pour un canal binaire symétrique et un taux de transmission $\frac{1}{2}$ est 11 %.
- Utilisés dans la plupart des standards récents :
  - les transmissions spatiales, la 3G des téléphones portables...

- Le code est défini par une matrice génératrice G :
  - la matrice est de taille $k \times n$,
  - le message est coupé en blocs de taille k,
  - chaque bloc m est multiplié par G pour obtenir un mot de code de taille n : $c = m \times G$.

- Ce sont des élément d'un sous-espace vectoriel de dimension k :
  - on définit une matrice de parité du code H de taille $(n-k) \times n$ orthogonale à cet espace :

$$H \times G^T = 0$$

→ $H \times c^T = 0$ si et seulement si c est dans le code.

Décodage d'un code linéaire

- On transmet $c = m \times G$ et on reçoit $c' = c + e$.
  - le but du décodage est de retrouver c à partir de c'
  - on cherche l'élément de l'espace qui avait la plus forte probabilité de donner c'
    $\longrightarrow$ le mot de code "le plus proche".
- Pour le canal binaire symétrique on utilise la distance de Hamming : le nombre de bits changés.
- Décodage par force brut :
  - regarder tous les mots du code,
  - mesurer leur distance à c' et choisir le plus proche.
- Trop long $\longrightarrow$ on utilise un code structuré.

- Définit par sa matrice de parité :

$$H = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix}$$

- Définit par sa matrice de parité :

$$H = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix}$$

- On en déduit une matrice génératrice :

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$

- Définit par sa matrice de parité :

$$H = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix}$$

- On en déduit une matrice génératrice :

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$

- Par exemple : $m = 0110 \rightarrow c = 0110010$.
  - Si $c' = 0010010$ on calcule $c' \times H^T = \begin{pmatrix} 0 & 1 & 0 \end{pmatrix}$
    $\rightarrow e = 0100000$ et $c = c' + e = 0110010$ et $m = 0110$.

Distance minimale d'un code

- Pour qu'un code soit bon il faut :
  - ▷ un décodeur efficace : comme Hamming, ou répétition
  - ▷ une bonne capacité de correction :
    - ⟶ pouvoir décoder un taux d'erreurs élevé.
- La capacité de correction est liée à la distance minimale du code notée d :
  - ▷ la plus petite distance entre deux mots de code.
    - ⟶ mieux vaut des mots espacés.
  - ▷ aussi, le plus petit poids d'un mot de code non nul.
- Pour le code de Hamming, $d = 3$,
  - ⟶ on dit que c'est un code $(7, 4, 3)$.

Bornes sur la distance minimale

- On sait que pour un code (n, k, d) on a toujours :

$$d \leq n - k + 1.$$

- La borne de Gilbert-Varshamov dit qu'un code (n, k, d) sur $GF(q)$ existe si :

$$q^k \leq \frac{q^n}{\sum_{i=0}^{d-1} \binom{n}{i} (q-1)^i}.$$

- Mais c'est une borne non constructive...
  - ▷ cela représente les meilleurs codes possibles.

- Un code corrigeant t erreurs vérifie toujours $t \leq \frac{d-1}{2}$,
  - ▷ sinon le décodage ne peux pas toujours être unique.

Codes de Reed-Solomon

Inventés en 1960

- Le message est un polynôme de degré $< k$ sur $GF(q)$
  - ▷ espace de dimension k.
- Le mot de code associé est son évaluation en n points $(x_1, ..., x_n)$ de $GF(q)$ appelés support du code.

- C'est un code linéaire :
  - ▷ combinaison linéaire des évaluations des X^i.
- Sa distance minimale est $d = n - k + 1$
  - ▷ un mot de code de poids $\leq n - k$ posséderait k zéros
    - → c'est le mot nul.

- Le décodage utilise l'algorithme de Berlekamp-Massey :
  - avec k évaluation on peut interpoler un polynôme,
  - avec n dont $\frac{n-k}{2}$ fausses on peut encore...
- Le décodage est unique et efficace jusqu'à $\frac{d-1}{2}$.

- On peut aussi utiliser l'algorithme de Guruswami-Sudan
  - inventé en 1999,
  - décodage en liste : pas nécessairement unique,
  - décode efficacement jusqu'à $n - \sqrt{nk}$ erreurs.
- Décode plus d'erreurs, mais est plus compliqué.

Utilisation pratique des RS

- Utilisés dans beaucoup d'applications :
  - ▷ les CD, DVD et Blue-Ray,
  - ▷ en codes concaténés : TNT, Mars Pathfinder...
- Leur structure sur un $GF(q)$ fait qu'ils résistent au bruit en "paquets".

Exemple : le Reed-Solomon $(15, 7, 9)$ sur $GF(16)$

- ▷ corrige 4 erreurs $\longrightarrow$ 4 erreurs isolées parmi 60 bits.
- ▷ chaque erreur peut porter sur 4 bits consécutifs $\longrightarrow$ jusqu'à 16 erreurs consécutives.
- Convient bien pour des taches sur un CD...

Autres familles de codes linéaires

- Il existe un grand nombre de familles de codes linéaires :
  - les codes aléatoires : atteignent GV en moyenne,
    → mais on ne sait pas les décoder...
  - les codes de Goppa binaires, pour $n \leq 2^m$
    → codes $(n, n - mt, 2t + 1)$ corrigeant t erreurs.
  - les codes LDPC (Low Density Parity Check)
    → décodage basé sur les équations de parité.
- Ont fait l'objet de beaucoup de recherches car on peut bien analyser ce qu'il se passe.

Pourquoi a-t-on besoin de meilleurs codes ?

- Les codes permettent de corriger le bruit du canal,
  - pourquoi ne pas plutôt directement améliorer le canal ?

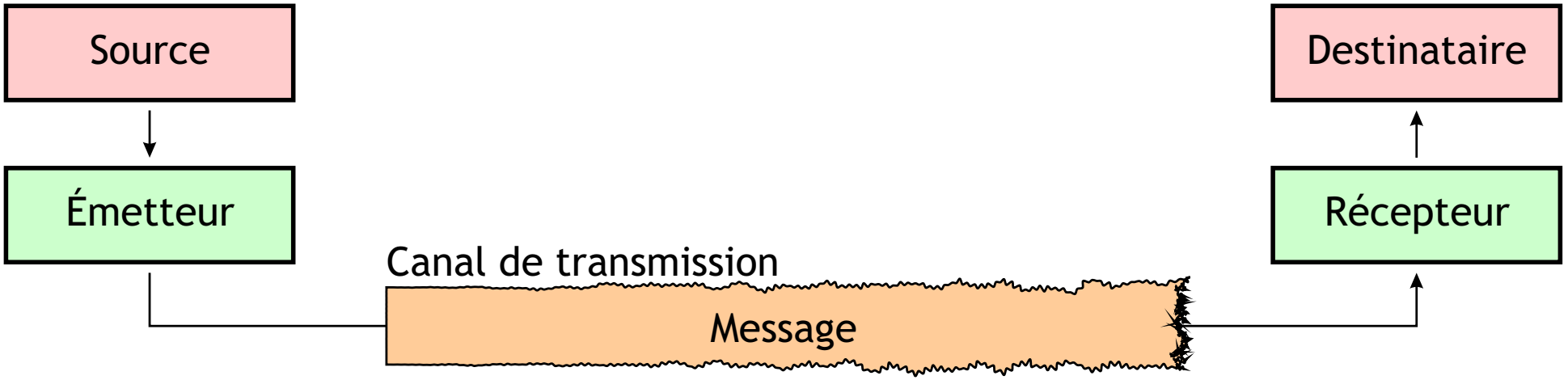
- En communication spatiale améliorer le canal signifie :
  - agrandir les antennes → plus lourd, plus cher
  - augmenter la puissance → plus grands panneaux...

- Le code coûte aussi en puissance :
  - émettre plus longtemps consomme plus,
  - compromis puissance d'émission/taux d'expansion.

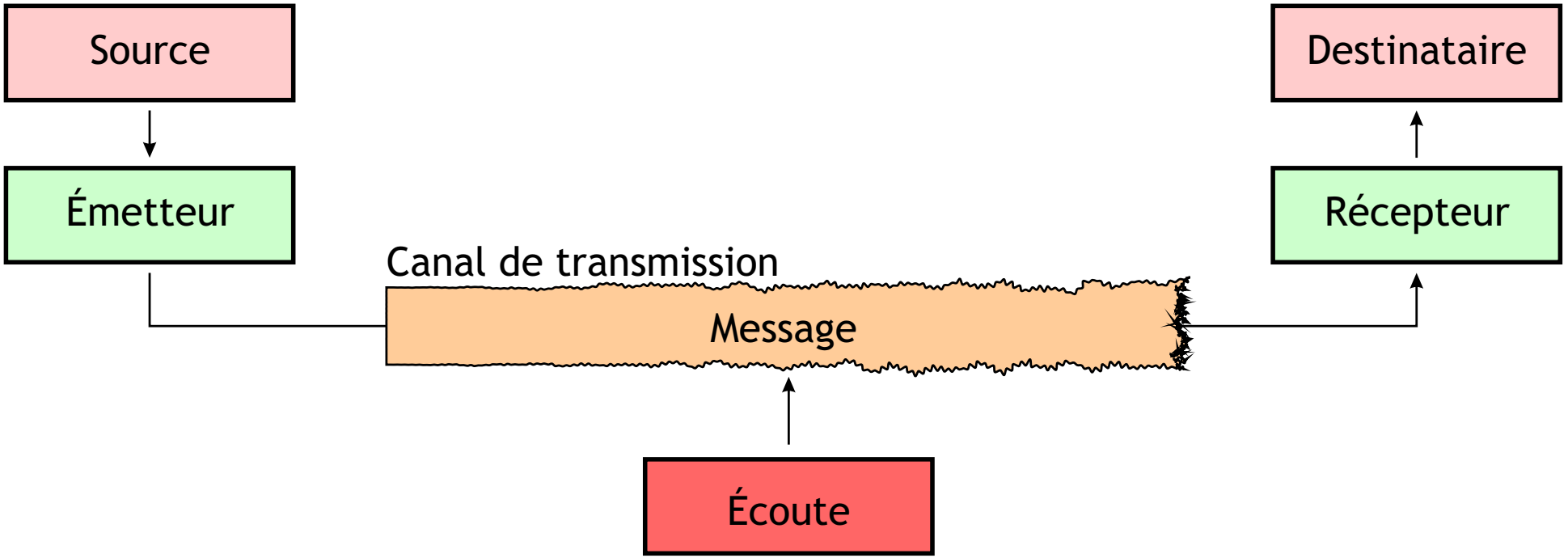
- Clairement, pour un taux donné, il faut la meilleure capacité de correction possible.

Protection de l'information

- Les codes correcteurs protègent l'information du bruit,
  - mais un canal a d'autres mauvaises propriétés...



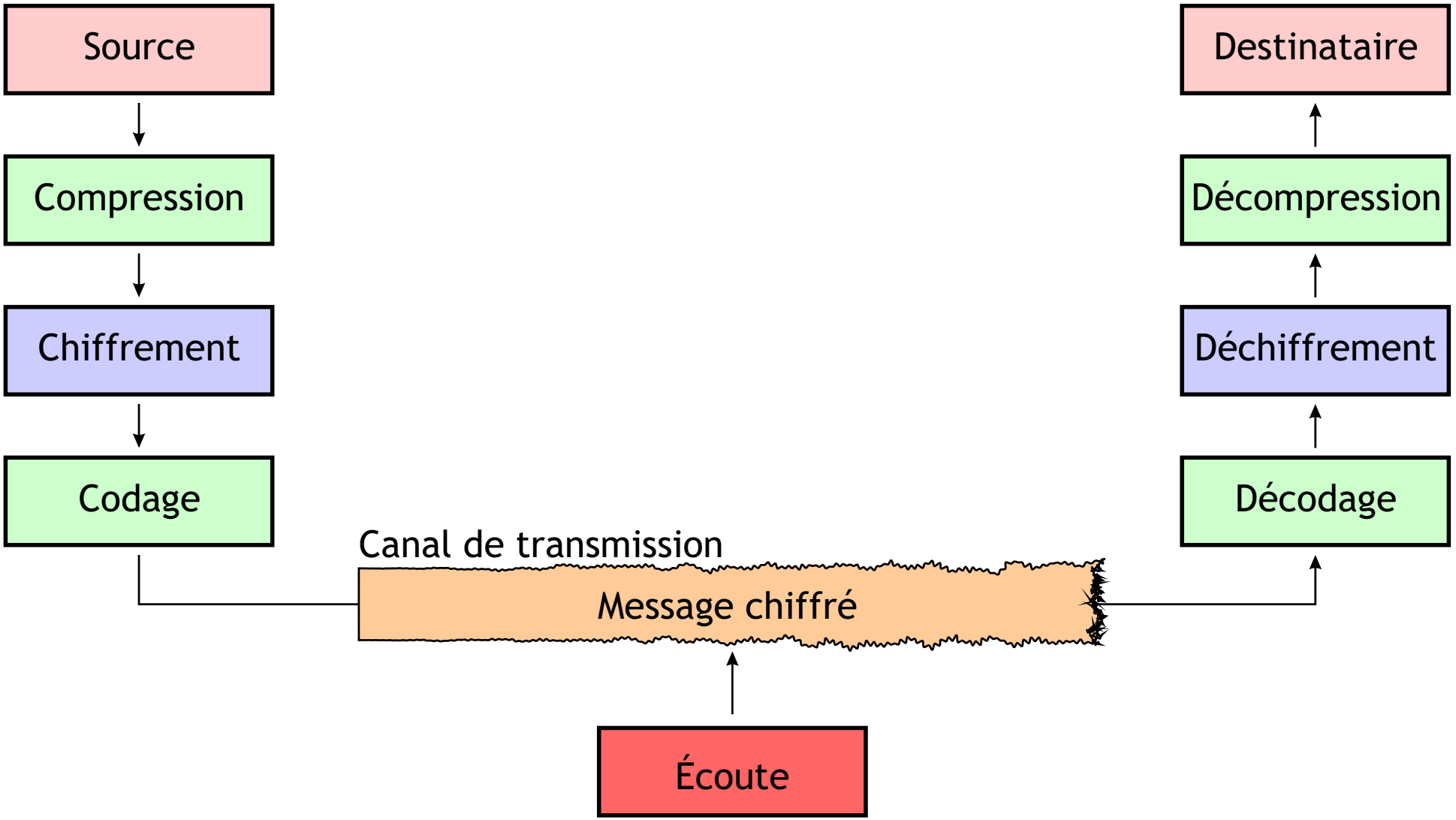
- Les codes correcteurs protègent l'information du bruit,
  - mais un canal a d'autres mauvaises propriétés...



- On ne peut pas toujours garantir que le canal soit sûr.

Modèle de communication chiffrée

- On modifie un peu le modèle de communication.



Que doit faire un bon chiffrement ?

- Un chiffrement est parfait si connaître le chiffré ne donne aucune information sur le message.
  - ▷ on s'intéresse à l'information mutuelle entre les deux.

- Notons m le message et $C(m)$ le chiffré de m.

$$I(m; C(m)) = \sum_{m,c} P(c|m)P(m) \log_2 \frac{P(c|m)}{P(c)}$$

- $P(c|m) = 1$ si $c = C(m)$ et 0 sinon
  - ▷ on trouve donc :

$$I(m; C(m)) = \sum_{m} P(m) \log_2 \frac{1}{P(m)} = H(m).$$

  → $C(m)$ contient toute l'information de m !

Que doit faire un bon chiffrement ?

- Un chiffrement doit donc utiliser un clef :
  - ▷ un chiffré doit pouvoir venir de plusieurs messages.
  - ▷ si on a une clef, un chiffré donné se déchiffre en autant de messages différents qu'il y a de clefs.
- Tous les chiffrements de l'antiquité sont à jeter :
  - ▷ permutation des lettres,
  - ▷ substitutions de lettres...
    - → aucun n'utilisait de clef.
- La sécurité du système doit reposer sur la connaissance de la clef utilisée,
  - ▷ l'algorithme utilisé doit lui pouvoir être public.

Chiffrement symétrique

- Un système de chiffrement symétrique consiste en :
  - un espace M de messages possibles,
  - un espace C de chiffrés possibles,
  - un espace K de clefs possibles,
  - une famille de fonctions de chiffrement, indexée par une clef k : $E_k : M \to C; m \mapsto E_k(m)$,
  - une famille de fonctions de déchiffrement, indexée par une clef k : $D_k : C \to M; c \mapsto D_k(c)$.
- Il faut aussi que : $D_k(E_k(m)) = m$.
- On appelle cela chiffrement symétrique, car source et destinataire utilisent la même clef k.

Existe-t-il un chiffrement symétrique parfait ?

- Pour cela il faut :

$$I(m; E_k(m)) = \sum_{m,c} P(c|m)P(m) \log_2 \frac{P(c|m)}{P(c)} = 0$$

- Si $P(c|m) \neq 0$ il faut $P(c|m) = P(c)$,
  - si un chiffré c peut être obtenu, ce doit l'être avec la même probabilité depuis n'importe quel message.
  - tous les messages peuvent donner ce chiffré,
  - le déchiffrement $D_k(c)$ est unique pour chaque clef,
    → nécessairement $|K| \geq |M|$

- Pour des messages binaires et des clefs binaires :
  → la clef au moins aussi longue que le message.

Exemple de chiffrement parfait

Le chiffrement de Vernam

- Très simple, inventé en 1917 :
  - $M = C = \{0,1\}^n$
  - $k \in \{0,1\}^n$
  - $E_k(m) = m \oplus k$,
  - $D_k(c) = c \oplus k$.
- Le chiffrement est le XOR du message avec la clef.

- Problème : la clef fait la taille du message,
  - on ne peut pas transmettre la clef sur le canal,
  - il faut un échange de clef “physique”,
  - ensuite on peut échanger un message aussi long.

Les chiffrements symétriques en pratique

- On veut utiliser des clefs courtes
  → impossible d'avoir une sécurité inconditionnelle.
- On se contente d'une sécurité calculatoire :
  - le temps de calcul nécessaire pour obtenir de l'information est trop grand pour être réalisable,
  - il suffit que $|K|$ soit assez grand et que l'attaquant n'ait pas d'information sur k.
    → l'attaquant doit essayer toutes les clefs de K.
- Les records de calculs sont autour de 2^{60} opérations :
  - $|K| = 2^{80}$ doit être suffisant pour être sûr,
  - on utilise des clefs de 128 bits pour avoir de la marge.

Les chiffrements symétriques en pratique

- Il existe 2 grandes familles de chiffrements symétriques.
- Les chiffrement à flot : A5/1, RC4...
  - la clef sert à générer une suite chiffrante,
  - cette suite est XORée au message.
- Les chiffrements par blocs : DES, AES...
  - le message est coupé en petits blocs
  - chaque bloc est chiffré avec la clef.
- Un très grand nombre de schémas de chiffrement existe.

- Mise au point en 1976 par Diffie et Hellman,
  - puis en 1977 avec RSA de Rivest, Shamir et Adleman.
- L'idée est de communiquer de façon sûre sans avoir à partager une clef secrète.
- Si Alice veut envoyer un message à Bob :
  - Bob génère une paire de clef publique/privée,
  - il publie la clef publique et cache la clef privée,
  - Alice utilise la clef publique pour chiffrer un message,
  - elle envoie le message chiffré à Bob,
  - seul Bob connaît la clef privée qui permet de déchiffrer.

- Pour qu'un tel schéma fonctionne il faut que les deux problèmes suivants soient difficiles :
  - retrouver la clef privée à partir de la clef publique,
  - inverser la fonction de chiffrement sans la clef privée.

- Cela oblige à avoir des systèmes très structurés :
  - la sécurité repose sur la difficulté de problèmes bien identifiés,
  - les tailles de clefs sont déduites du coût des meilleurs algorithmes connus pour résoudre ces problèmes.

- Bob choisit p et q deux nombres premiers et $e < pq$
  - il calcule $N = pq$ et (e, N) est la clef publique,
  - p, q et d tel que $ed = 1 \mod (p-1)(q-1)$ sont privés.
- Pour chiffrer un message m on calcule $c = m^e \mod N$.
- Pour déchiffrer un chiffré c : $m = c^d \mod N$.
- Le choix de d fait que $(m^e)^d \mod N = m$.
- Les deux problèmes difficiles sont :
  - retrouver d à partir de $e \iff$ factoriser N en p et q,
  - calculer $\sqrt[e]{c} \mod N$.

- Aucun algorithme efficace n'extrait la racine e-ième.
- Pour attaquer RSA il faut factoriser N :
  - N de 1024 bits donne un coût estimé à 2^{80} opérations,
  - N de 2048 bits donne un coût estimé à 2^{112} opérations.

- Les clefs sont grosses :
  - chiffrement et déchiffrement sont lents,
  - on préférerait utiliser un chiffrement symétrique,
  - on utilise du chiffrement hybride.

Chiffrement hybride

- Fonctionne en deux temps :
  - un cryptosystème asymétrique sert à échanger une clef,
  - cette clef est utilisée dans un chiffrement symétrique.

- C'est ce qui est utilisé dans le protocole SSL (https) :
  - le site web envoie un certificat $\longrightarrow$ clef publique signée,
  - le client choisit une clef k et la chiffre,
  - le client envoie la clef au site qui la déchiffre,
  - tout le reste de la communication est chiffré avec k.

Conclusion

Conclusion

La théorie de l'information c'est :

- un ensemble de définitions indispensables,
- des outils pour mesurer l'information,
- des algorithmes de codage de source/compression,
- des codes correcteurs et des bornes pour les évaluer,
- des systèmes de chiffrement sûrs et efficaces,
- les bases pour construire un monde numérique.