Codage de canal
MA331

Nicolas Barbot

nicolas.barbot@esisar.grenoble-inp.fr

2016-2017

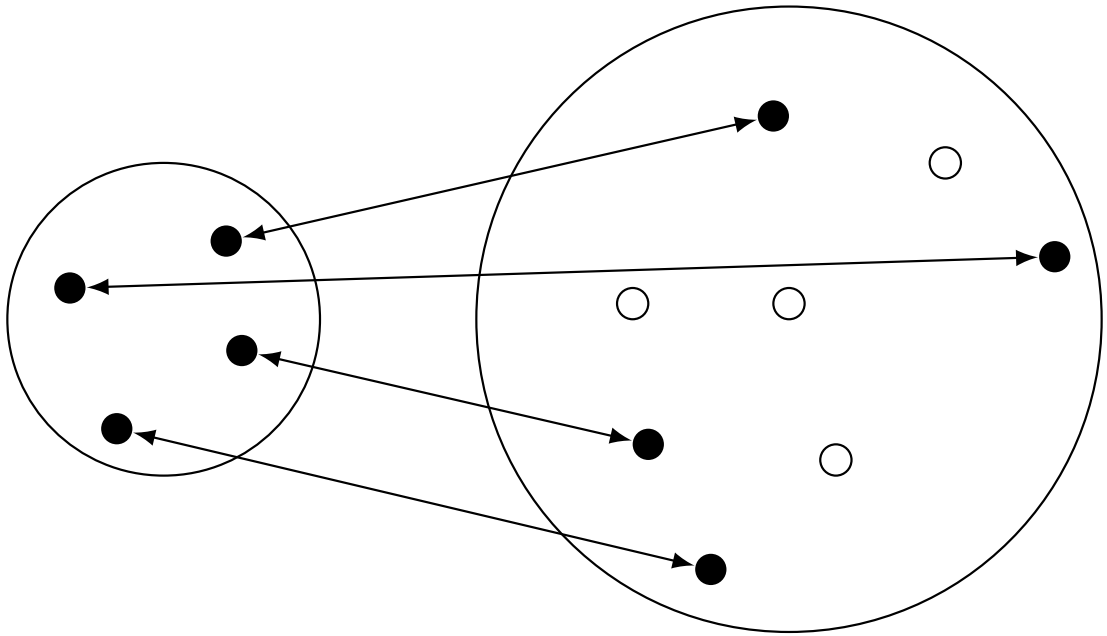
Introduction

Avant la transmission sur le canal il est nécessaire d'ajouter de la redondance pour lutter contre les perturbations du canal. L'ajout de redondance implique une augmentation de la longueur du message. Cette opération est réalisée par le codeur (situé dans l'émetteur).

En réception, cette redondance est exploitée par le décodeur (situé dans le récepteur) pour compenser (ou détecter) les perturbations apportées par le canal.

Codes en bloc

- Un message s est une séquence binaire de taille k
- Un mot de code c est une séquence binaire de taille $n > k$
- Il existe une bijection entre les messages et les mots de code



Codes en bloc

- Il y a 2^k messages possibles
- L'espace du code contient 2^n points dont seulement 2^k sont des mots de codes
- Un code est un sous ensemble de 2^k vecteurs de taille n
- Il y a une correspondance unique entre chaque message et chaque mot de code
- Le rendement du code est défini par $R = k/n$

Exemple: Code de parité sur $n = 3$ bits

$$s \in \{00, 01, 10, 11\}$$

$$c \in \{000, 011, 101, 110\}$$

Distance de Hamming

Le poids de Hamming d'une séquence binaire est défini comme le nombre de 1 dans la séquence

$$w([101]) = 2 \quad w([11101]) = 4$$

La distance de Hamming entre deux séquences binaires x et y est le nombre de places où elles diffèrent

$$\begin{array}{rl} x = & [110101] \\ y = & [110011] \end{array} \Rightarrow d(x,y) = w(x+y) = 2$$

Inégalité triangulaire

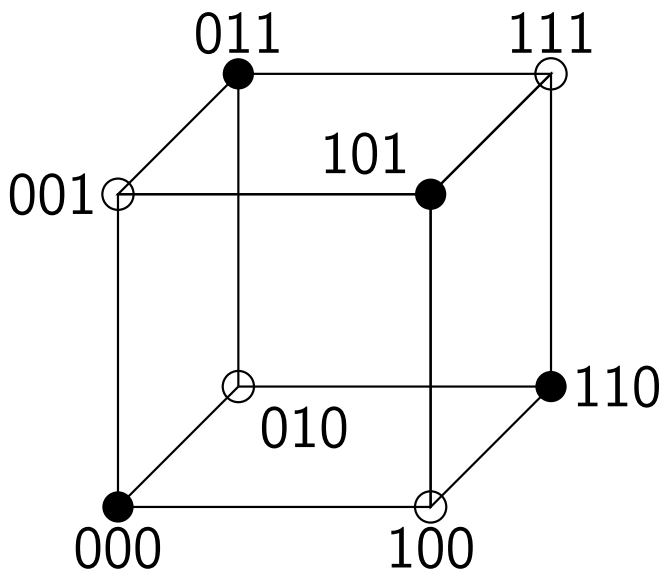
$$d(x,y) + d(y,z) \geq d(x,z)$$

Distance minimale d'un code

$$d_{min} = \min_{v,w} d(v,w) \text{ où } v, w \in C \text{ et } v \neq w$$

Pour le code de parité $n = 3$ bits

$$\{00, 01, 10, 11\} \Rightarrow \{000, 011, 101, 110\}$$



Codes correcteur et détecteur d'erreur

Pour un même code, on peut toujours appliquer deux algorithmes différents:

Code détecteur d'erreur

Le décodeur vérifie simplement que le mot reçu est un mot de code valide (le décodeur renvoie 0 ou 1).

Code correcteur d'erreur

Le décodeur renvoie le mot de code permettant de maximiser une certaine métrique. Le mot de code renvoyé contient en général moins d'erreur que le mot reçu.

Le second décodeur est beaucoup plus complexe que le premier. La théorie de l'information s'intéresse uniquement au second cas.

Décodeur Maximum Likelihood (ML)

Le décodeur ML renvoie le mot de code qui maximise la probabilité:

$$\hat{c}_{ML} = \text{argmax}_{c \in C} p(r|c) \tag{1}$$

Ce décodeur renvoie le mot de code le plus proche du mot reçu

Décodeur Maximum A Posteriori (MAP)

Le décodeur ML renvoie le mot de code qui maximise la probabilité:

$$\hat{c}_{MAP} = \text{argmax}_{c \in C} p(c|r) = \text{argmax}_{c \in C} \frac{p(r|c)}{p(r)} p(c) \tag{2}$$

Si la source est sans redondance, alors pour r donné, $\hat{c}_{ML} = \hat{c}_{MAP}$.

Performance minimale d'un code

Lors de la transmission d'un mot de code c sur le canal, le mot reçu r peut être différent du mot de code envoyé. Pour le canal binaire symétrique $r = c + e$ où e est appelé motif d'erreur.

Pour un code de distance minimale d_{min}:

- pour un algorithme détecteur d'erreur, il est toujours possible de détecter un motif d'erreur de poids $t \leq d_{min} - 1$. Il est toutefois possible de détecter des motifs d'erreurs de poids supérieur.
- pour un algorithme correcteur d'erreur, il est toujours possible de corriger un motif d'erreur de poids $t \leq \lfloor (d_{min} - 1)/2 \rfloor$. Il est toutefois possible de corriger des motifs d'erreur de poids supérieur.

Codes aléatoires

On choisit aléatoirement 2^k mots de code parmi les 2^n vecteurs possibles que l'on stocke dans un grand tableau.

Pour encoder un message s_i de taille k, on récupère la valeur c_i stockée dans le tableau à l'indice s_i.

Pour le décodage, le décodeur parcourt le grand tableau pour trouver le mot de code $\hat{c}_i$ le plus proche du mot reçu. Le message décodé $\hat{s}_i$ correspond à l'indice de ce mot de code.
Si seule la détection d'erreur est demandée, on parcourt le tableau pour déterminer si le mot reçu est un mot de code.

Codes aléatoires

Avantage

- Les codes aléatoires sont "bons": pour n assez grand, on peut prouver en moyenne que ces codes atteignent la capacité du canal ($p_b \to 0$ si $R = k/n < C$)

Inconvénients

- Nécessité de stocker tous les mots de code (pour E et R).
- Nécessité de comparer le mot reçu avec tous les mots de code (pour R)

Exercices : Calculez la taille de la mémoire nécessaire si $n = 100$ et $k = 50$. Déterminez le nombre de comparaisons pour le décodage d'un mot de code.

Codes linéaires

- Les codes linéaires permettent de résoudre le problème de l'encodage.
- Un code linéaire est un sous-espace vectoriel de dimension k d'un espace vectoriel de dimension n sur un corps fini (généralement $\mathbb{F}_2$).
- Il existe une application linéaire permettant de déterminer le mot de code à partir du message
- Les codes linéaires représentent la quasi totalité des codes utilisés en pratique

Matrice génératrice

L'application d'encodage d'un code linéaire C est une application linéaire injective de $\mathbb{F}_2^k$ dans $\mathbb{F}_2^n$.

$$G = \begin{bmatrix} g_{1,1} & g_{1,2} & \cdots & g_{1,n} \\ g_{2,1} & g_{2,2} & \cdots & g_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ g_{k,1} & g_{k,2} & \cdots & g_{k,n} \end{bmatrix}$$

Le mot de code peut alors être obtenu par:

$$c = sG \tag{3}$$

La matrice génératrice est une base du code C et est constituée de k mots de code.

Code équivalent

- Il existe plusieurs matrices génératrices pour définir un même code
- On peut obtenir une matrice équivalente en permutant deux lignes ou en remplaçant une ligne par une combinaison linéaire de deux autres lignes (si C est un code linéaire et si $c_1, c_2 \in C$ alors $c_1 + c_2 \in C$).
- Tous les mots de code restent identiques, seul le mapping change

Code systématique

Tout code linéaire possède une matrice génératrice systématique. Cette matrice assure que les k premiers (ou derniers) bits c_i de chaque mot de code sont identiques aux bits du message s_i et est définie par:

$$G = \begin{bmatrix} 1 & 0 & \cdots & 0 & p_{1,k} & p_{1,k+1} & \cdots & p_{1,n} \\ 0 & 1 & \cdots & 0 & p_{2,k} & p_{2,k+1} & \cdots & p_{2,n} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 & p_{k,k} & p_{k,k+1} & \cdots & p_{k,n} \end{bmatrix}$$

Le mot de code peut alors être obtenu par

$$c = s\ G = [s\ sP] \tag{4}$$

Exercices

Soit la matrice génératrice

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

Encodez le message $s = [1101]$

Le code de parité $n = 3$ bits est un code linéaire.
Trouvez la matrice génératrice du code

Matrice de Parité

Pour tout code linéaire, il existe une application linéaire surjective de $\mathbb{F}^n$ dans un espace de dimension $n-k$ ayant pour noyau exactement le code:

$$c\ H^T = 0 \tag{5}$$

- H est de dimensions $(n-k) \times n$
- Il existe plusieurs matrices H pour un même code
- On peut obtenir une matrice équivalente en permutant deux lignes ou en remplaçant une ligne par une combinaison linéaire de deux autres lignes
- H correspond à la matrice génératice du code dual

Matrice de parité d'un code systématique

Soit un code systématique définit par la matrice génératrice G

$$G = [I_{k\times k} P_{k\times n-k}]$$

Alors la matrice de parité H du code peut être obtenue par:

$$H = [P^T_{n-k\times k} I_{n-k\times n-k}] \tag{6}$$

Un code non systématique peut être transformé en code systématique de manière à obtenir la matrice de parité. Cette matrice de parité peut aussi être utilisée avec le code non systématique.

Décodage par syndrome (détection)

- On reçoit le mot $r = c + e$
- On calcule le syndrome de r: $S = rH^T$
- Si $S = 0$ alors le mot reçu est un mot de code et le décodeur déduit qu'il n'y a pas eu d'erreur de transmission
- Si $S = 0$ une erreur de transmission est détectée

Décodage par syndrome (correction)

- On reçoit le mot $r = c + e$
- On calcule le syndrome de r: $S = rH^T = (c + e)H^T = eH^T$ (le syndrome ne dépend que du motif d'erreur)
- Le récepteur possède une table de correspondance entre les syndromes et le motif d'erreur
- Pour chaque syndrome, le décodeur retrouve le motif d'erreur et inverse les bits erronés

Construction de la table des syndromes

- On génère tous les motifs d'erreur de poids 1 et on calcule les syndromes correspondants. On stocke les résultats dans la table.
- On génère tous les motifs d'erreur de poids 2 et on calcule les syndromes correspondants. On stocke le résultat seulement si le syndrome n'a pas été découvert (si le syndrome a déjà été découvert, on jette le motif d'erreur).
- On répète ces opérations jusqu'a ce que tous les syndromes soient découverts.

Exercice: Combien existe-t-il de syndromes différents. Calculez la taille de la table pour un code linéaire avec $n = 100$ et $k = 50$

Distribution des distances

Pour un code linéaire, on définit la distribution des distances du code $A(w)$ comme le nombre de mots de code de poids w

Cette distribution est indépendante du mot de code considéré

Pour le code de parité $n = 3$ bits:

$$A(0) = 1 \quad A(1) = 0 \quad A(2) = 3 \quad A(3) = 0$$

Ainsi, pour un code linéaire, la distance minimale est égale au poids du mot de code le plus creux (à l'exception du mot de code nul).

Probabilité de non détection

Lors de la détection d'erreur, une erreur est indétectable si elle est égale à un mot de code (si $e \in C$ alors $rH^T = (c + e)H^T = 0$).

Par conséquent la probabilité de non détection d'un code linéaire est égale à la probabilité d'obtenir un motif d'erreur de poids i multiplié par le nombre de mots de codes de poids i, pour toutes les valeurs de i:

Probabilité de non détection

$$P_u = \sum_{i=1}^{n} A(i) f^i (1 - f)^{n-i} \tag{7}$$

Lors de la correction d'erreur, les motifs d'erreurs de poids $t \leq \lfloor (d_{min} - 1)/2 \rfloor$ peuvent toujours être corrigés.
La probabilité d'erreur bloc (WER) p_B est définie par:

$$p_B = p(\hat{c} \neq c) < \sum_{i=t+1}^{n} \binom{n}{i} f^i (1-f)^{n-i} \tag{8}$$

La probabilité d'erreur bit (BER) p_b est la probabilité qu'un bit soit erroné après le décodage. Le BER dépend du code et de la matrice G utilisée à l'encodage
De manière générale, p_B et p_b sont évaluées par des simulations de Monte Carlo

Codes de répétition R_3

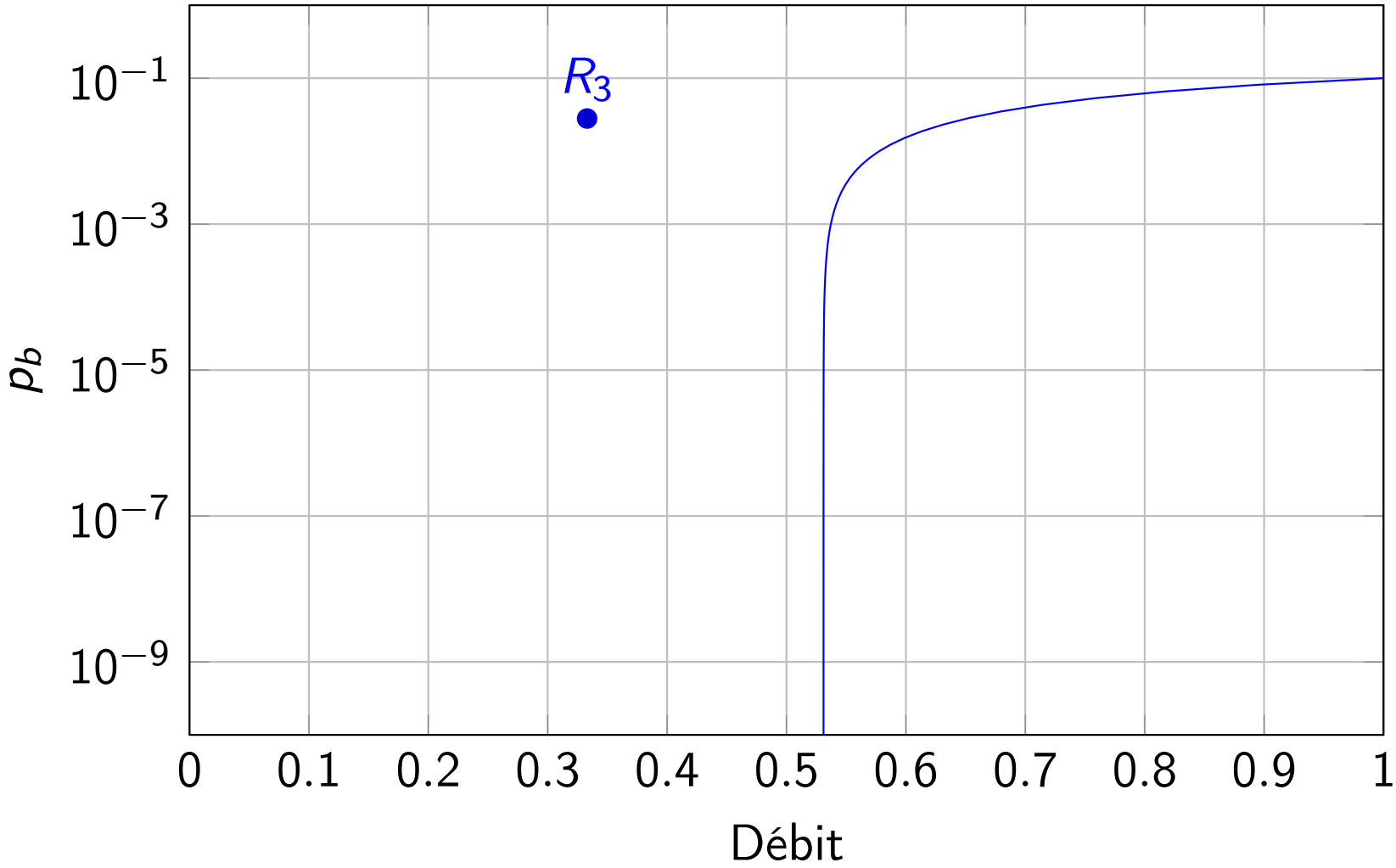
On considère le code R_3 définit par: $R_3 = \{000, 111\}$

- Sachant que R_3 est un code linéaire, déterminez G et H
- Encodez le message $s = [0010110]$

La transmission est réalisée sur un BSC avec $f = 0.1$

- Effectuer le décodage lorsque $r = [000001111000010111000]$
- Calculer p_B ainsi que p_b

Performances du code R_3



Codes de répétition R_n

On considère le code R_n définit par une répétition de n bits identiques et égaux à la valeur du bit à transmettre s_i.

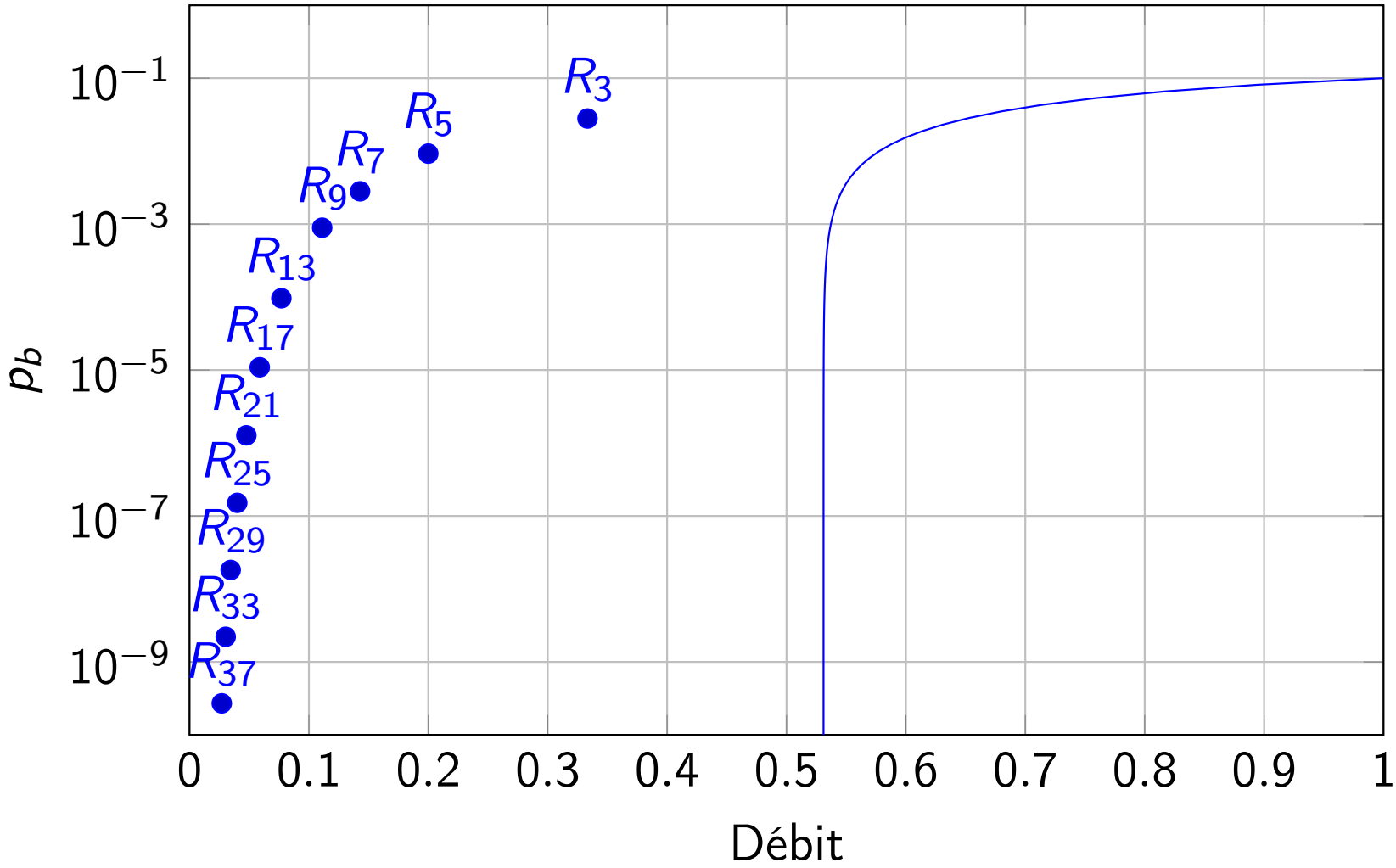
La probabilité d'erreur peut être obtenue en sommant la probabilité d'obtenir un motif d'erreur de poids supérieur à $(n+1)/2$.

$$p_b = \sum_{i=\lceil N/2 \rceil}^{n} \binom{n}{i} f^i (1-f)^{n-i}$$

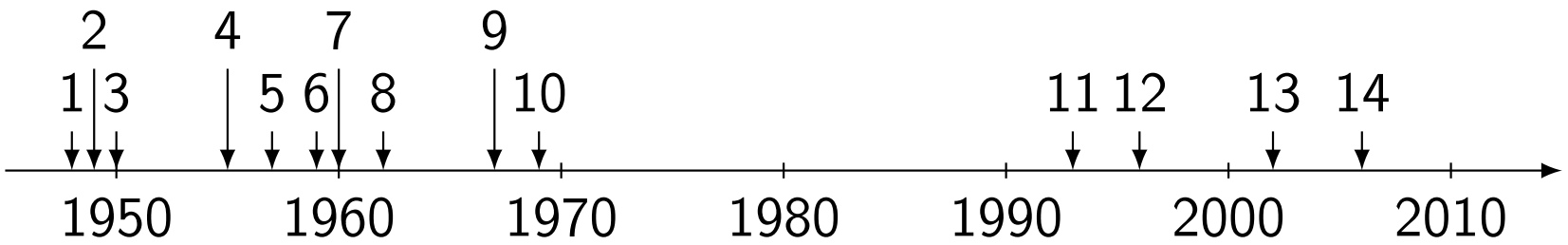
Ce qui implique:

$$\lim_{n\to\infty} p_b = 0 \quad \text{et} \quad \lim_{n\to\infty} R = \frac{k}{n} = 0$$

Performances des code R_n



Historique du codage de canal



1:	Bases de la théorie de l'information	1948
2:	Codes de Golay Code parfait corrigeant 3 erreurs	1949
3:	Codes de Hamming Codes parfaits corrigeant 1 erreur	1950
4:	Codes Convolutifs inventés par Elias	1955
5:	Codes Cycliques découverts par Prange	1957
6:	Codes BCH par Hocquenghem, Bose et Chaudhuri	1959
7:	Codes de Reed Solomon	1960
8:	1ère découverte des codes LDPC par Gallager	1962
9:	Algorithme de Viterbi	1967
10:	Algorithme de Berlekamp Massey	1969
11:	Turbocodes par Berrou, Glavieux et Thitimajshima	1993
12:	Redécouverte des codes LDPC par MacKay et Neal	1996
13:	Codes LT découverts par Luby	2002
14:	Codes Raptor par Shokrolahi	2006

Borne de Hamming

Soit un code $C(n, k)$ est composé de 2^k mots de code de taille n.
Si le code est t correcteur alors il est possible de partitionner l'espace du code en 2^k sphères de rayon t (sinon le code n'est pas t correcteur).
Pour un code binaire, une sphère de rayon t contient $\sum_{i=0}^{t} \binom{n}{i}$ mots. Le volume des 2^k sphères doit toujours être inférieur à 2^n.
Ainsi l'égalité suivante doit être respectée:

Borne de Hamming

$$2^k \leq \frac{2^n}{\sum_{i=0}^{t} \binom{n}{i}} \tag{9}$$

En cas d'égalité, le code est qualifié de parfait

Codes de Hamming

Pour les codes $C(n, k)$ pouvant corriger $t = 1$ erreur, la borne devient:

$$k \leq 2^m - m - 1 \tag{10}$$

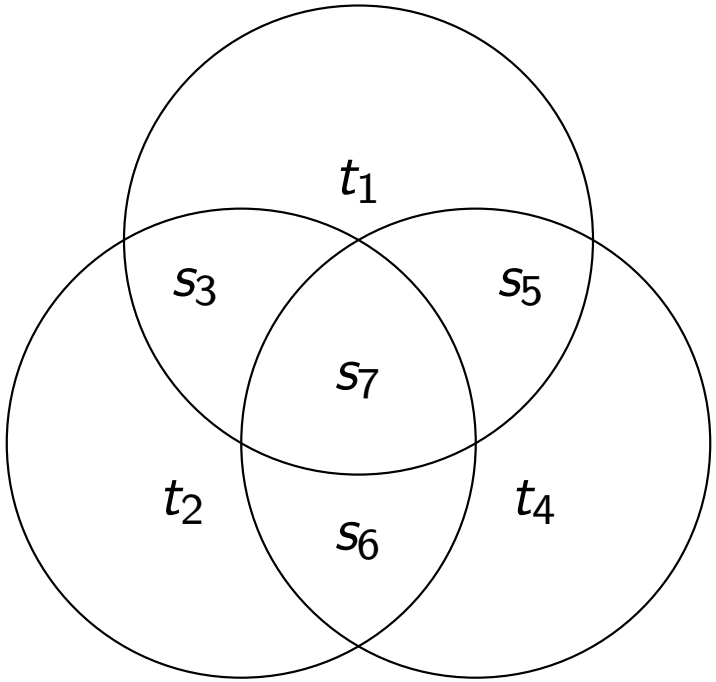
où $m = n - k$ est le nombre de bit de parité

Les codes de Hamming sont des codes parfait permettant de corriger 1 erreur (avec la quantité minimale de redondance).

Les codes de Hamming sont donc: (3,1), (7,4), (15,11)...

Code de Hamming (7,4)

- Numéroter les bits du mot de code de 1 à $N = 7$
- Les bits correspondant à des puissances de 2 (1, 2 et 4) sont des bits de parité, t_i les autres sont des bits de données, s_i
- Un bit de donnée s_i est relié à un bit de parité t_i si sa décomposition en puissances de deux fait intervenir t_i



Code de Hamming (7,4)

Pour encoder un message s, on place les 4 bits du message sur les positions s_i, on calcule ensuite la valeur des bits t_i pour assurer la parité dans chaque cercle. On transmet finalement le mot de code.

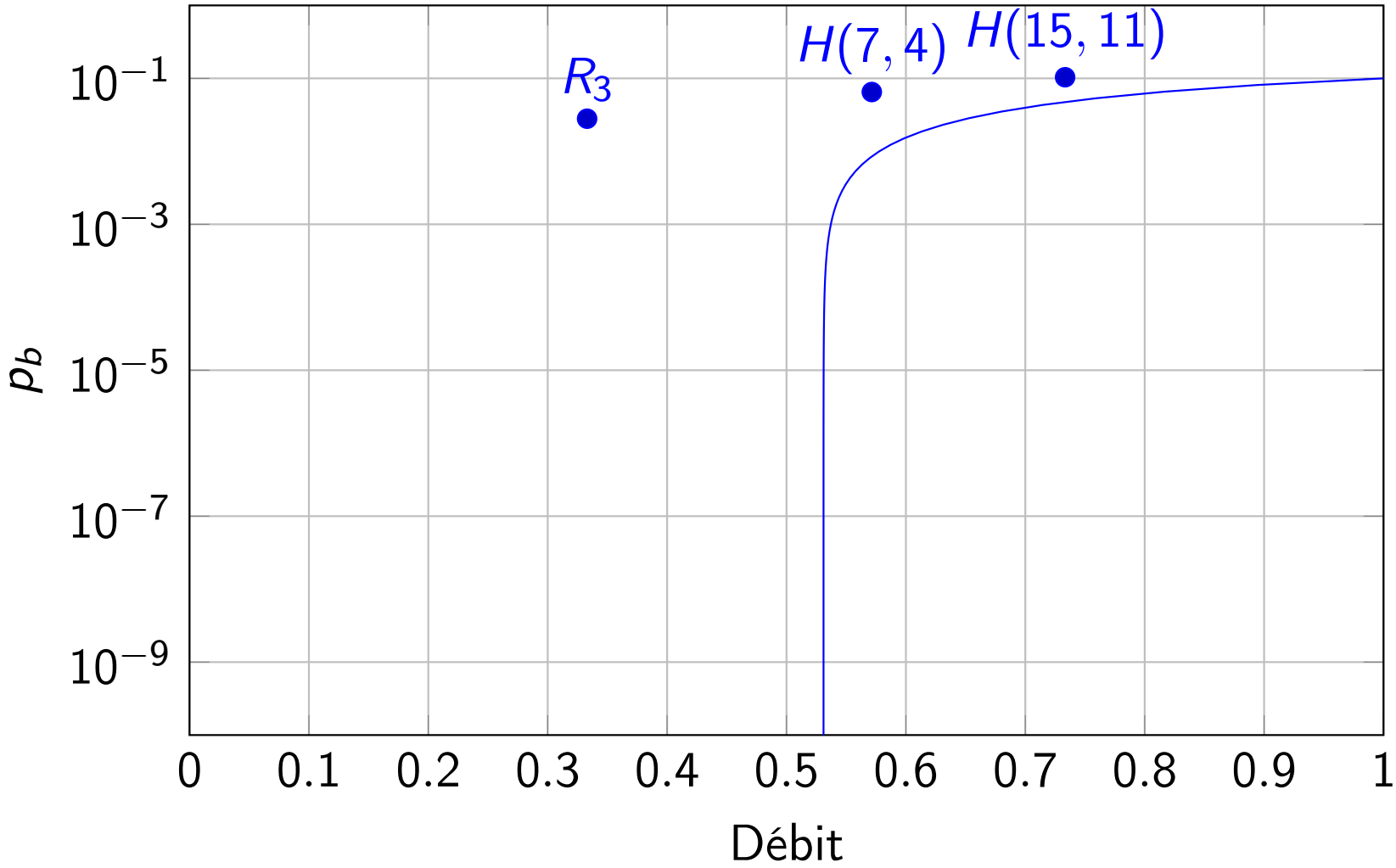
Pour décoder un mot reçu, on place les bits sur les positions s_i et t_i. Si toutes les parités sont respectées, le mot reçu est un mot de code sinon, on change la valeur d'un seul bit pour que la parité soit respectée dans les trois cercles.

L'encodage et le décodage peuvent aussi être effectués à partir des matrice G et H du code (les performances du code sont évidement identiques).

Exercices

- Encodez le message $s = [0110101001011101]$
- Décodez le message $r = [1101011101101101]$
- Ajouter une erreur sur un bit de donnée et décodez.
- Ajouter une erreur sur un bit de parité et décodez.
- Ajouter 2 erreurs et décodez.

Performances du code de Hamming (7,4)



Codes cycliques

Un code C est dit cyclique s'il est linéaire et si la permutation circulaire d'un mot de code engendre un autre mot de code:

$$(c_0 c_1 c_2 \dots c_{n-1}) \in C \Leftrightarrow (c_{n-1} c_0 c_1 \dots c_{n-2}) \in C \qquad (11)$$

Exemple: Soit C un code cyclique et [11101000] un de ses mots de code alors les vecteurs suivants sont aussi des mots de codes:
[11101000] [00111010] [10001110] [10100011]
[01110100] [00011101] [01000111] [11010001]

Le code de parité $n = 3$, les codes de répétitions et le code de Hamming sont aussi des codes cycliques

Représentation polynomiale

On associe à chaque mot de code la représentation polynomiale suivante:

$$(c_0 c_1 c_2 \dots c_{n-1}) \leftrightarrow c(x) = c_0 + c_1 x + c_2 x^2 + \dots + c_{n-1} x^{n-1} \quad (12)$$

Un code cyclique forme un idéal dans l'anneau quotient $\mathbb{F}_2[x]/(x^n - 1)$.
Un décalage à droite peut être obtenu en multipliant $c(x)$ par x:

$$\begin{aligned} xc(x) &= c_0 x + c_1 x^2 \dots c_{n-1} x^n + c_{n-1} + c_{n-1} \bmod x^n - 1 \\ &= c_{n-1} + c_0 x + c_1 x^2 \dots c_{n-2} x^{n-1} + c_{n-1}(x^n + 1) \bmod x^n - 1 \\ &= c_{n-1} + c_0 x + c_1 x^2 \dots c_{n-2} x^{n-1} \\ &\Leftrightarrow (c_{n-1} c_0 c_1 \dots c_{n-2}) \end{aligned}$$

Polynôme générateur

Chaque code cyclique C de longueur n sur $\mathbb{F}_2$ possède un polynôme générateur $g(x)$. De plus, ce polynôme est diviseur de $x^n - 1$ dans $\mathbb{F}_2$.

Ce polynôme permet de générer l'ensemble des mots de code de C. Chaque mots de code est obtenu par:

$$c(x) = s(x)\ g(x) \tag{13}$$

Exemple: Soit C un code cyclique comprenant les mots de code: $\{0, 1+x, x+x^2, 1+x^2\}$
Le polynôme générateur du code est $g(x) = x + 1$
On peut vérifier par exemple que $x(x+1) = x^2 + x \in C$

Polynôme générateur

Il existe des tables pour les valeurs de n permettant de trouver les factorisations de $x^n + 1$

n	Facteurs
7	3.15.13
9	3.7.111
15	3.7.31.23.37
17	3.471.727
21	3.7.15.13.165.127
23	3.6165.5343
25	3.37.4102041
27	3.7.111.1001001
31	3.51.45.75.73.67.57
33	3.7.2251.3043.3777
35	3.15.13.37.16475.13627

Construction d'un code cyclique

On veut construire un code cyclique de longueur $n = 7$ sur $\mathbb{F}_2$. On montre que $(x^7 - 1) = (x + 1)(x^3 + x + 1)(x^3 + x^2 + 1)$.

$g_1(x) = x + 1$
$g_2(x) = x^3 + x + 1$
$g_3(x) = x^3 + x^2 + 1$
$g_4(x) = (x + 1)(x^3 + x + 1) = x^4 + x^3 + x^2 + 1$
$g_5(x) = (x + 1)(x^3 + x^2 + 1) = x^4 + x^2 + x + 1$
$g_6(x) = (x^3 + x + 1)(x^3 + x^2 + 1) = x^6 + x^5 + x^4 + x^3 + x^2 + x + 1$

On peut donc générer les codes cycliques suivants: $C_1(7, 6)$, $C_2(7, 4)$, $C_3(7, 4)$, $C_4(7, 3)$, $C_5(7, 3)$ et $C_6(7, 1)$.

Construction d'un code cyclique systématique

Soit s(x) le message à coder avec un code cyclique de polynôme générateur $g(x)$:

1. Calculer $x^{n-k}s(x)$ en décalant $s(x)$ de $n-k$ positions
2. Calculer le reste $r(x)$ de la division polynomiale de $x^{n-k}s(x)$ par $g(x)$
3. Ajouter $x^{n-k}s(x)$ et $r(x)$

$$x^{n-k}s(x) = q(x)g(x) + r(x) \tag{14}$$

$$\Leftrightarrow c(x) = x^{n-k}s(x) + r(x) \tag{15}$$

Détection d'erreur avec un code cyclique

Le récepteur reçoit le polynôme $r(x)$:

$$r(x) = s(x)g(x) + e(x) \tag{16}$$

où $e(x)$ est le polynôme d'erreur
Le décodeur calcule le syndrome $S(x)$ égal au le reste de la division polynomiale de $r(x)$ par $g(x)$.

$$S(x) = r(x) \bmod g(x) = e(x) \bmod g(x) \tag{17}$$

Si $S(x) = 0$, le décodeur fait l'hypothèse que la transmission est correcte, sinon au moins une erreur de transmission est présente.

Exercices

- Encodez le message $x^5 + x^3 + x$ avec le code C_1
- Encodez le message $x^3 + x + 1$ avec les codes C_2 et C_3
- Encodez le message $x + 1$ avec les codes C_4 et C_5
- Encodez le message 1 avec le code C_6
- Déterminez le code auquel appartient le mot de code $x^6 + x^5 + x^4$

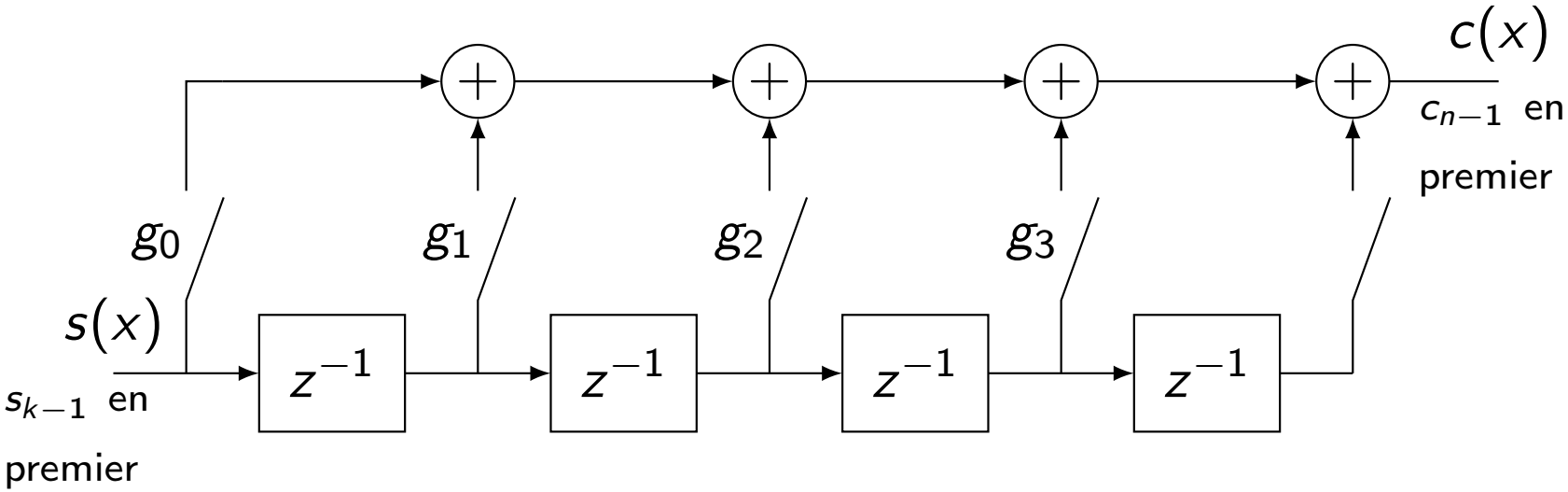
Représentation matricielle

Les codes cycliques, comme tous codes linéaires, admettent une matrice génératice. Il est possible de déduire la matrice G d'un code cyclique à partir de sa représentation polynomiale.

$$\begin{aligned} c(x) &= s(x)g(x) \\ &= s_0 g(x) + s_1 x g(x) + \cdots + s_{k-1} x^{k-1} g(x) \end{aligned}$$

$$c = [s_0 s_1 s_2 \ldots s_{k-1}] \times \begin{bmatrix} g_0 & g_1 & \cdots & g_{m-1} & & & \\ & g_0 & g_1 & \cdots & g_{m-1} & & \\ & & \ddots & \ddots & & \ddots & \\ & & & g_0 & g_1 & \cdots & g_{m-1} \end{bmatrix}$$

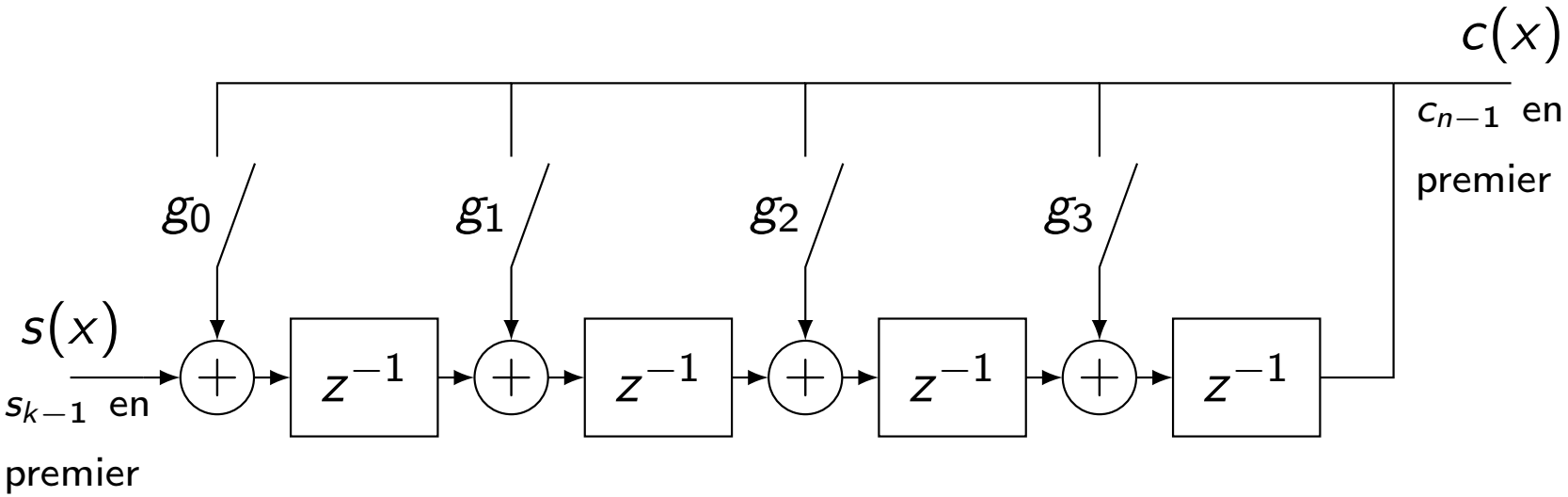
Multiplication polynomiale



- Registres initialisés à 0
- Structure utilisée pour l'encodage non systématique
- Possibilité de transposer la structure (applications haut débit)

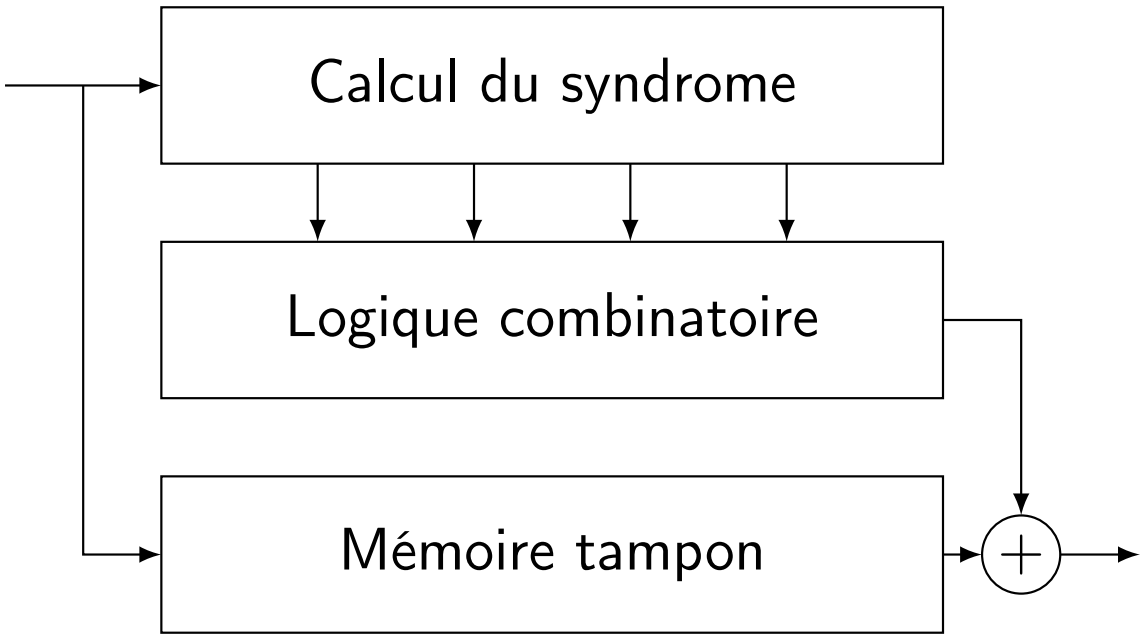
Exercice: Encodez le message $s = [1011]$ sachant que $g = [10111]$

Division polynomiale



- Registres initialisés à 0
- r présent dans le registre après k coups d'horloge
- Structure utilisé pour l'encodage systématique et le décodage

Décodeur de Meggitt (correction d'erreur)



Inconvénients:

- Logique combinatoire souvent complexe
- Sous optimal

Cyclic Redundancy Check

- Codes cycliques binaires et systématiques classiquement utilisés pour la détection d'erreur
- Rendement supérieur à 95%
- Utilisés dans la plupart des protocoles (Ethernet, 802.11, LTE...)

Exercices:
Calculez le rendement du code utilisé dans Ethernet pour une trame de 1500 octets
Le CRC-5 normalisé par l'ITU pour le standard G.704 est défini par $g(x) = x^5 + x^4 + x^2 + 1$. Calculez le CRC pour la trame fictive $s = [1001101110101]$.

Code de Golay

Le code de Golay est un code parfait de paramètres $n = 23$, $k = 12$ et $d_{min} = 7$ permettant de corriger 3 erreurs.

Ce code admet une représentation sous la forme d'un code cyclique de paramètre:

$$g(x) = x^{11} + x^{10} + x^6 + x^5 + x^4 + x^2 + 1$$

Les codes de répétition R_n (dont le rendement tend vers 0), les codes de Hamming (dont le rendement tend vers 1) et le code de Golay ($r \approx 1/2$) sont les seuls codes parfaits connus.

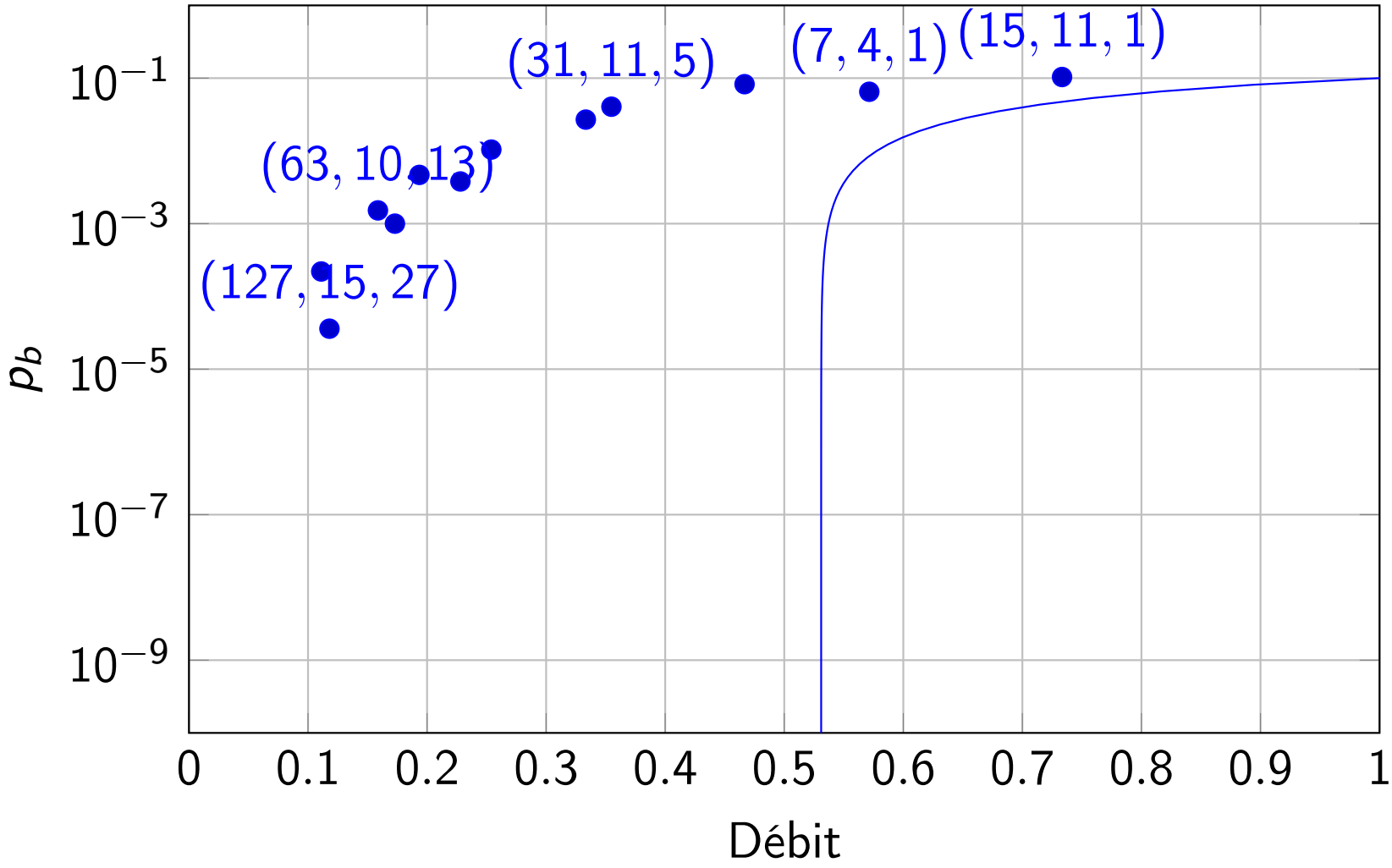
Codes BCH

- Inventés par Bose, Chaudhuri et Hocquenghem
- Codes cycliques binaires t correcteur par construction

n	k	t	$g(x)$
7	4	1	13
15	11	1	23
	7	2	721
	5	3	2467
31	26	1	45
	21	2	3551
	16	3	107657
	11	5	5423325
	6	7	313365047

n	k	t	$g(x)$
63	57	1	103
	51	2	12471
	45	3	1701317
	39	4	166623567
	36	5	1033500423
	30	6	157464165547
	24	7	17323260404441
	18	10	1363026512351725
	16	11	6331141367235453

Performances des codes BCH



Conclusion

Les codes algébriques ne peuvent pas atteindre la capacité du BSC:

- Algorithme de décodage conçu pour être $t = \lfloor (d_{min} - 1)/2 \rfloor$ correcteur (et pas plus). Un code pour atteindre la capacité doit corriger au delà de cette limite
- Complexité trop élevée pour considérer des longueurs de bloc de plusieurs milliers.

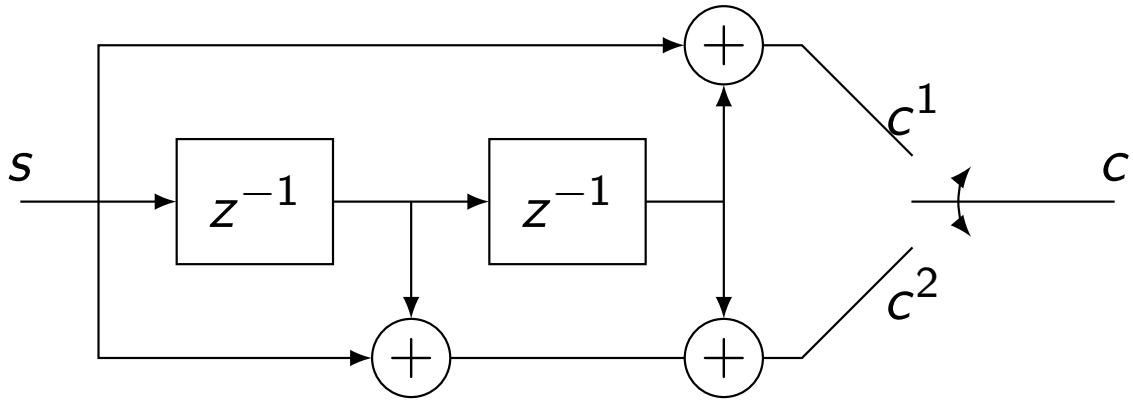
Les codes algébriques sont inefficaces sur le canal AWGN:

- L'algorithme de décodage travaille dans $\mathbb{F}_n$, ce qui nécessite une conversion des échantillons présents en sortie du canal (réels) vers $\mathbb{F}_2$ (décodage hard). Un algorithme efficace doit prendre en compte l'information contenue dans le module (décodage soft).

Codes convolutifs

- Codage linéaire par flux de donnée (différent du codage en bloc)
- Codes utilisés sur les sondes Voyager 1 et 2
- Codes popularisé par l'algorithme de Viterbi
- Décodage hard pour le BSC
- Décodage soft pour le canal AWGN
- Codes utilisés dans la norme 802.11, GSM...

Encodage



Le message s présent à l'entrée du codeur est:

$$s = (\dots s_{-1}, s_0, s_1, \dots)$$

La séquence de sortie est de la forme:

$$c = (\dots c^1_{-1}, c^2_{-1}, c^1_0, c^2_0, c^1_1, c^2_1, \dots)$$

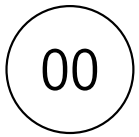
Caractéristiques

- Le code décrit précédemment accepte 1 bit en entrée et génère 2 bits en sortie (le rendement de ce code est donc 1/2)
- Au lancement du décodeur, le registre est initialisé à 0
- La longueur de contrainte l_c est égale aux nombre de délai plus 1 (3 dans ce cas)
- Chaque sortie est caractérisée par un générateur ($g_1 = [101]$ et $g_2 = [111]$).
- Un code convolutif est entièrement décrit par ses générateurs. Dans notre cas, le code est un code convolutif (5,7).

Diagramme d'état

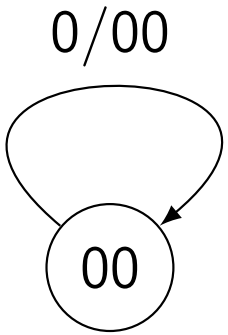
- Diagramme d'état du code $(5, 7)$
- Nombre d'états: $2^{l_c - 1}$
- 2 sorties par état
- État initial: 00

Diagramme d'état



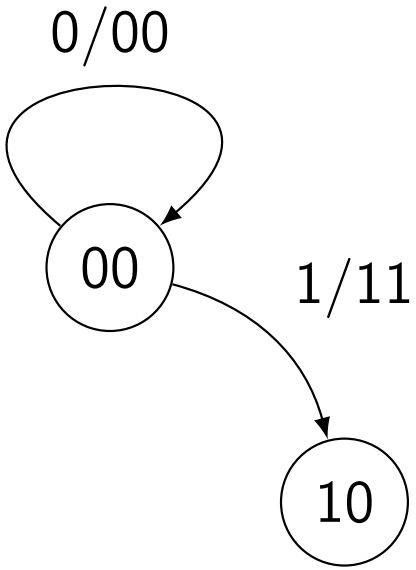
- Diagramme d'état du code $(5, 7)$
- Nombre d'états: $2^{l_c - 1}$
- 2 sorties par état
- État initial: 00

Diagramme d'état



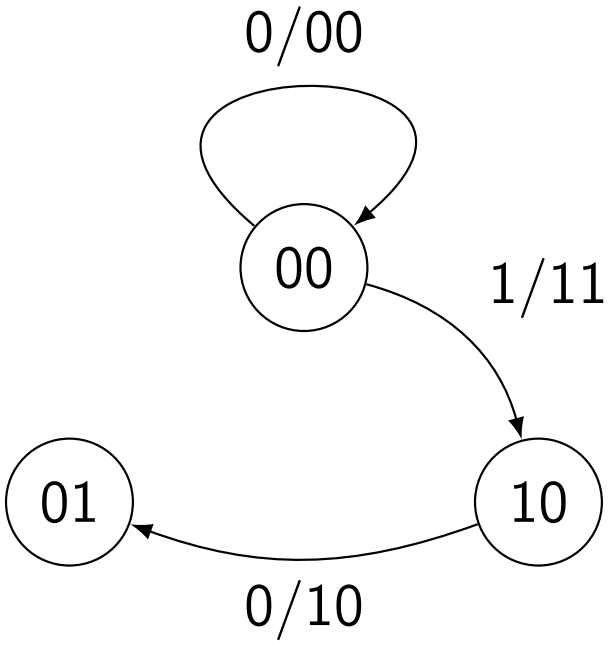
- Diagramme d'état du code $(5, 7)$
- Nombre d'états: $2^{l_c - 1}$
- 2 sorties par état
- État initial: 00

Diagramme d'état



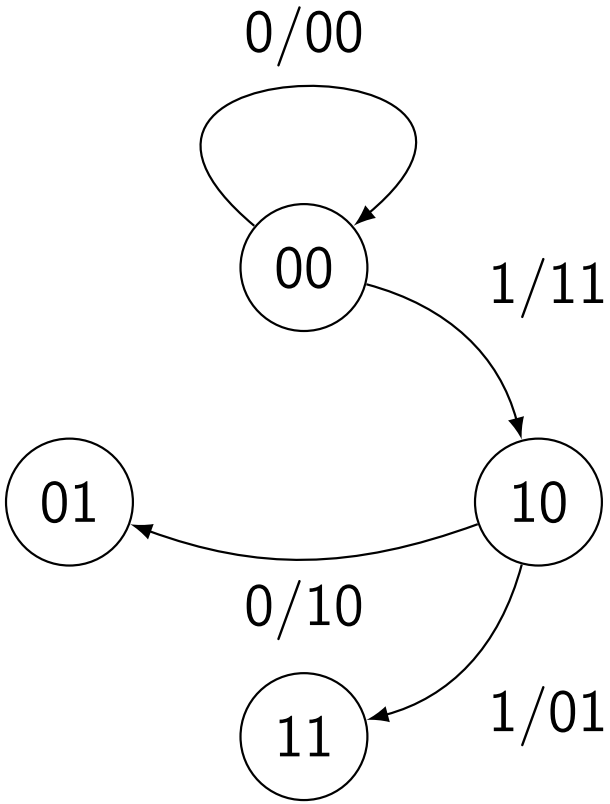
- Diagramme d'état du code $(5, 7)$
- Nombre d'états: $2^{l_c - 1}$
- 2 sorties par état
- État initial: 00

Diagramme d'état



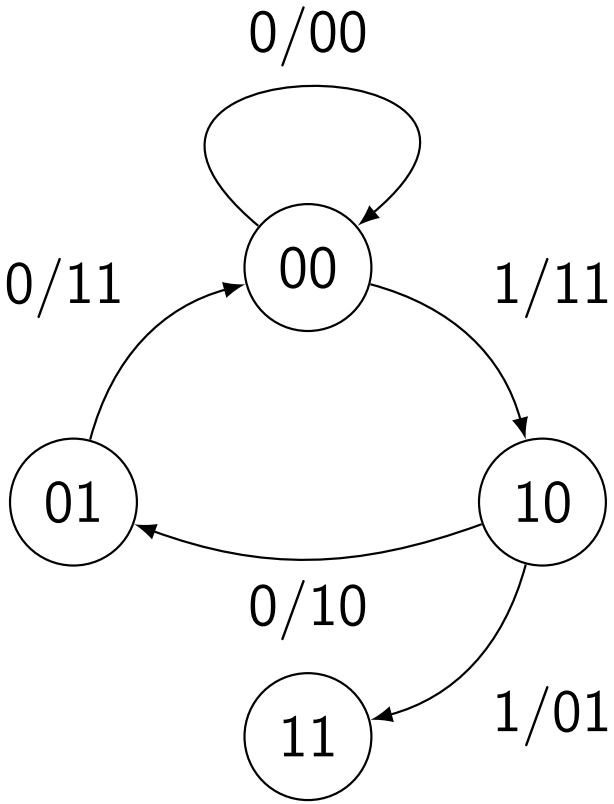
- Diagramme d'état du code $(5, 7)$
- Nombre d'états: $2^{l_c - 1}$
- 2 sorties par état
- État initial: 00

Diagramme d'état



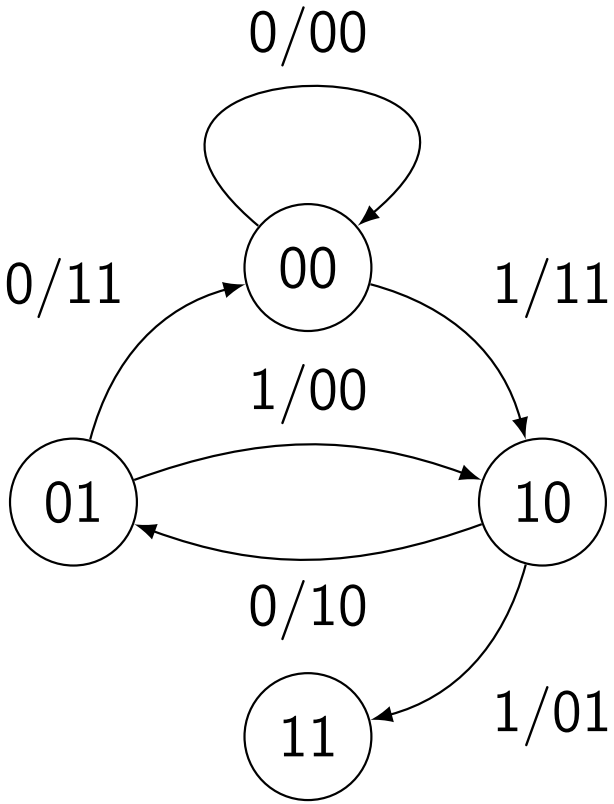
- Diagramme d'état du code $(5, 7)$
- Nombre d'états: $2^{l_c - 1}$
- 2 sorties par état
- État initial: 00

Diagramme d'état



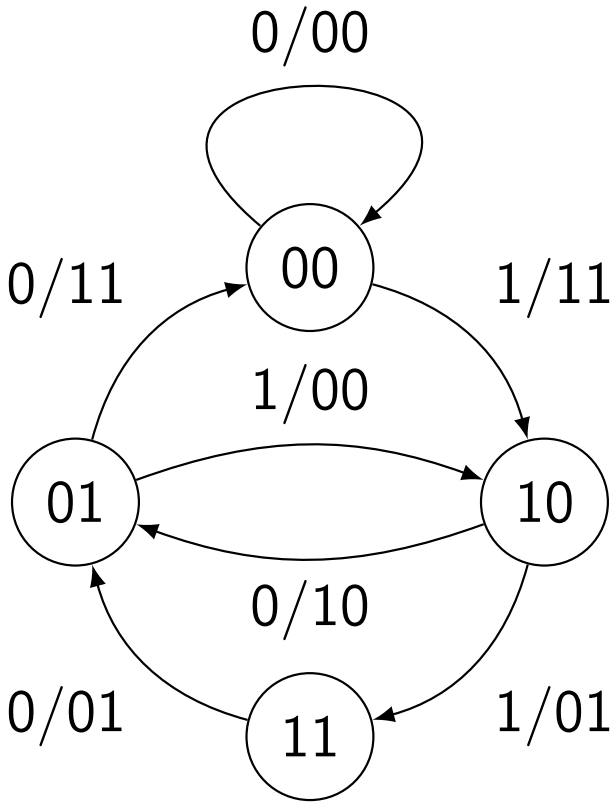
- Diagramme d'état du code $(5, 7)$
- Nombre d'états: $2^{l_c - 1}$
- 2 sorties par état
- État initial: 00

Diagramme d'état



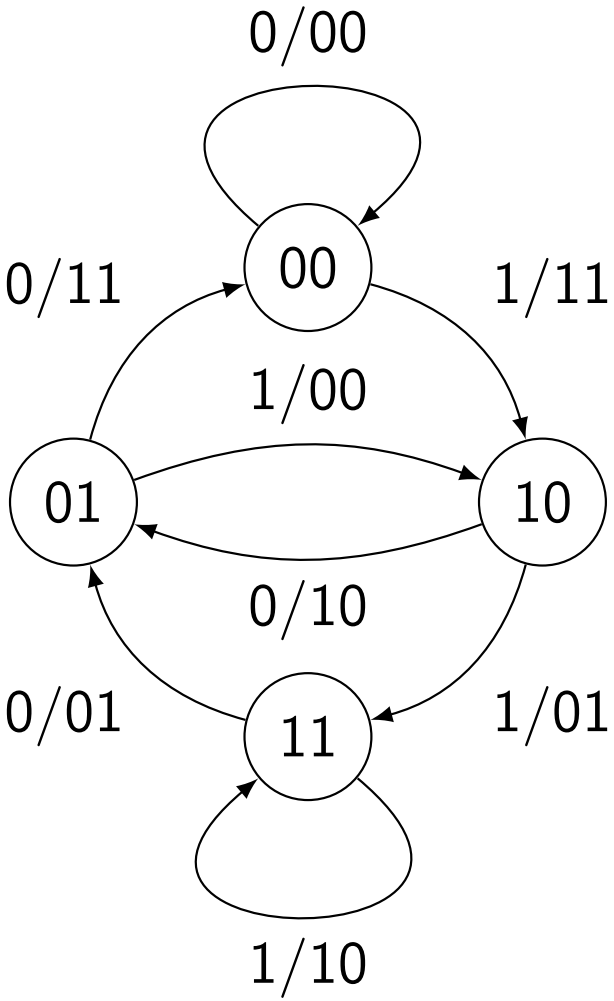
- Diagramme d'état du code $(5, 7)$
- Nombre d'états: $2^{l_c - 1}$
- 2 sorties par état
- État initial: 00

Diagramme d'état



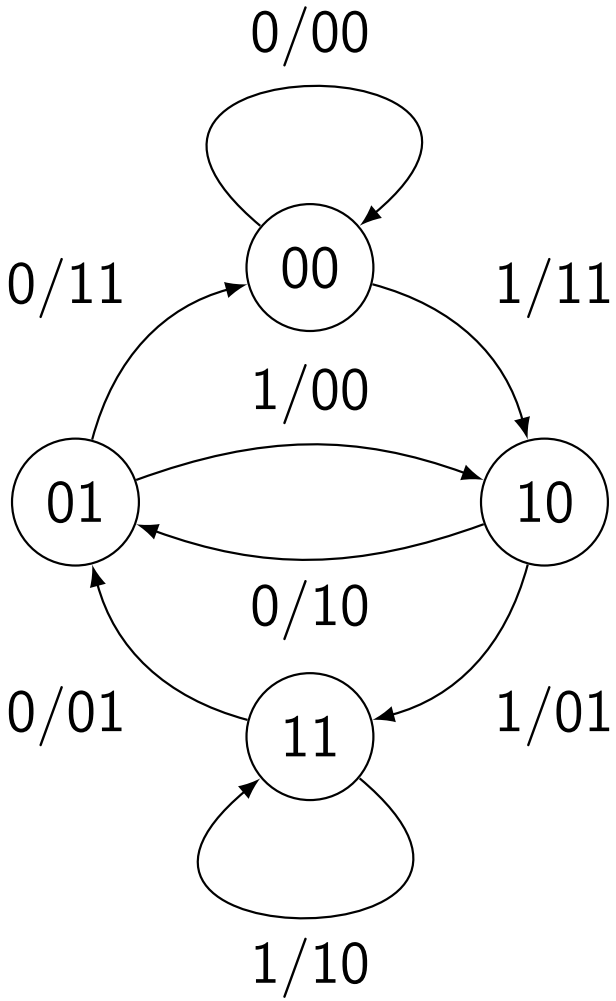
- Diagramme d'état du code $(5, 7)$
- Nombre d'états: $2^{l_c - 1}$
- 2 sorties par état
- État initial: 00

Diagramme d'état



- Diagramme d'état du code $(5, 7)$
- Nombre d'états: $2^{l_c - 1}$
- 2 sorties par état
- État initial: 00

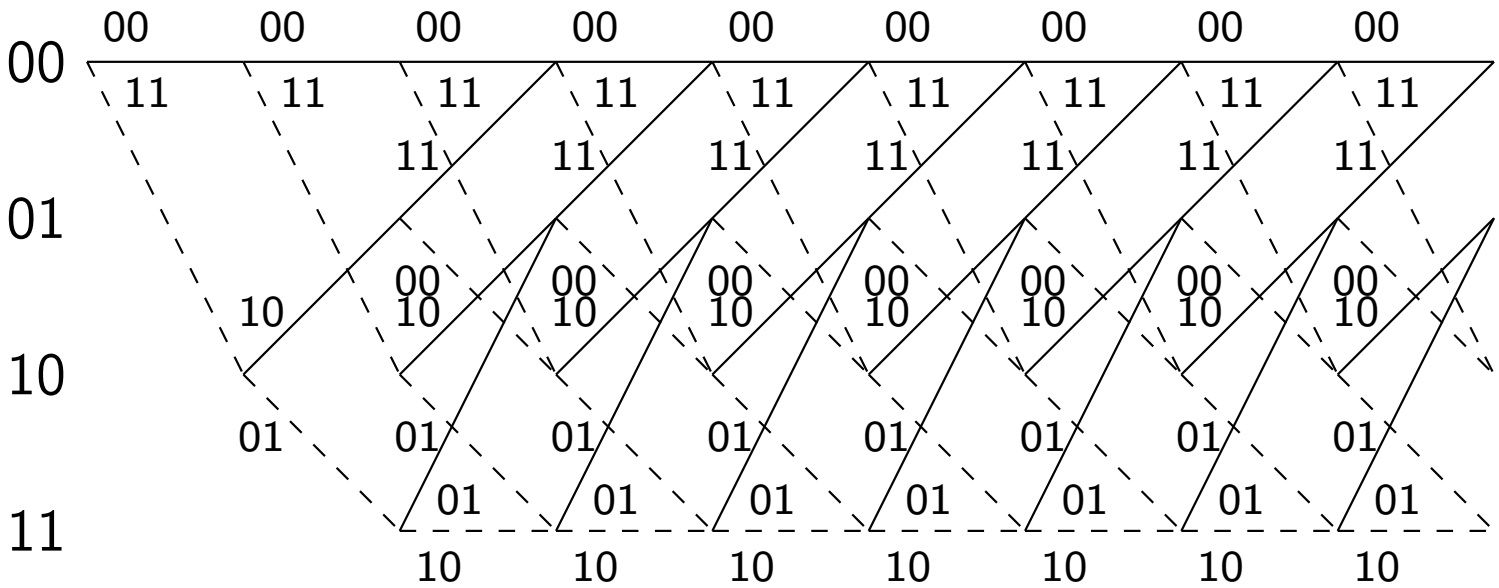
Diagramme d'état



- Diagramme d'état du code $(5, 7)$
- Nombre d'états: $2^{l_c - 1}$
- 2 sorties par état
- État initial: 00

Exercices:
Encodez la séquence $s = [0110100110]$
Décodez la séquence $c = [0001100111]$

Treillis du code



Exercices:

Déterminez le chemin emprunté par les séquences s et c dans le treillis du code (5,7).

Distance libre

Distance libre

La distance libre d_f d'un code convolutif est le poids minimum d'une séquence ayant pour point de départ et d'arrivé, l'état nul (excepté la séquence nulle).

Plus la distance libre d_f du code est importante, plus la probabilité d'erreur est faible.

Pour le code (5,7) la distance libre est obtenu en passant par les états 00, 10, 01, 00 et vaut 5.

Comme pour les codes en bloc, il est possible de déterminer la distribution des distances d'un code convolutif.

Codes de distances libre maximale

Il existe des tables pour déterminer les générateurs permettant d'obtenir une distance libre d_f maximale pour une longueur de contrainte l_c donnée:

l_c	$g(x)$	d_f
3	(5,7)	5
4	(15,17)	6
5	(23,35)	7
6	(53,75)	8
7	(133,171)	10

Rendement: 1/2

l_c	$g(x)$	d_f
3	(5,7,7)	8
4	(13,15,17)	10
5	(25,33,37)	12
6	(47,53,75)	13
7	(133,145,175)	15

Rendement 1/3

Décodage naïf

Pour une séquence reçu r, le décodeur ML doit renvoyer le chemin appartenant au treillis c le plus proche du chemin reçu.

Exercices:
Pour $k = 3$, déterminez toutes les séquences codées différentes.
On reçoit $r = 110011$ sur un canal BSC avec $f = 0.1$, déterminez $p(r|c)$ pour chaque cas.

Décodage naïf

Pour une séquence reçu r, le décodeur ML doit renvoyer le chemin appartenant au treillis c le plus proche du chemin reçu.

Exercices:
Pour $k = 3$, déterminez toutes les séquences codées différentes.
On reçoit $r = 110011$ sur un canal BSC avec $f = 0.1$, déterminez $p(r|c)$ pour chaque cas.

s	c	$p(r\|c)$	s	c	$p(r\|c)$
000	000000	$8.1e-5$	100	111000	$7.3e-4$
001	000011	$6.6e-3$	101	111011	$5.9e-2$
010	001110	$9.0e-6$	110	110111	$5.9e-2$
011	001101	$9.0e-6$	111	110110	$6.6e-3$

Pour un message s de longueur k, il existe 2^k chemins possibles

Fonction de vraisemblance

La vraisemblance d'un chemin r par rapport à un chemin appartenant au treillis c vaut:

$$p(r|c) = \prod_{i=0}^{k-1} p(r_i|c_i) = \prod_{i=0}^{k-1} \prod_{j=0}^{n-1} p(r_i^j|c_i^j)$$

En prenant le logarithme:

$$\log p(r|c) = \sum_{i=0}^{k-1} \log p(r_i|c_i)$$

La vraisemblance logarithmique correspond la métrique chemin et est égale à la somme des métriques branches

Algorithme de Viterbi

L'algorithme de Viterbi permet d'effectuer un décodage ML sans évaluer l'ensemble des chemins possibles.

L'algorithme permet de supprimer un chemin dès qu'il ne peut plus être le chemin qui maximise la vraisemblance.

A chaque fois que plusieurs chemins entrent dans le même état alors seul celui qui possède la vraisemblance la plus élevé est conservé (les autres sont éliminés).

Décodage hard

Sur le canal binaire symétrique de paramètre f, si r_i et c_i ont une longueur de n et différent en d positions alors:

$$p(r_i|c_i) = f^d(1-f)^{n-d}$$

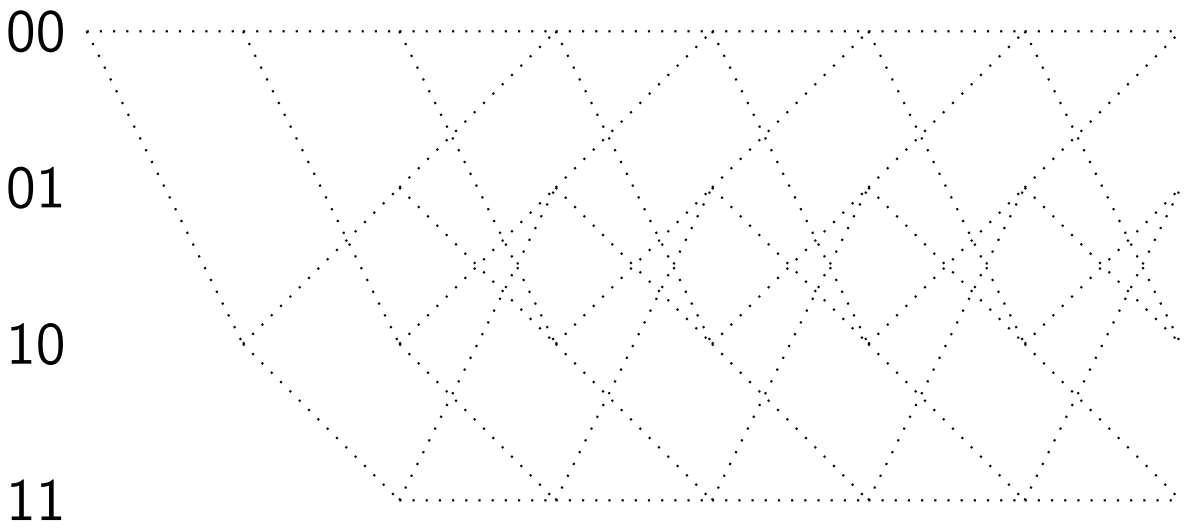
En logarithme:

$$\begin{aligned}\log p(r_i|c_i) &= -d \log \frac{1-f}{f} + n \log(1-p) \\ &\propto -d\end{aligned}$$

La séquence $\hat{c}_{ML}$ est celle qui minimise la distance de Hamming avec la séquence reçue.

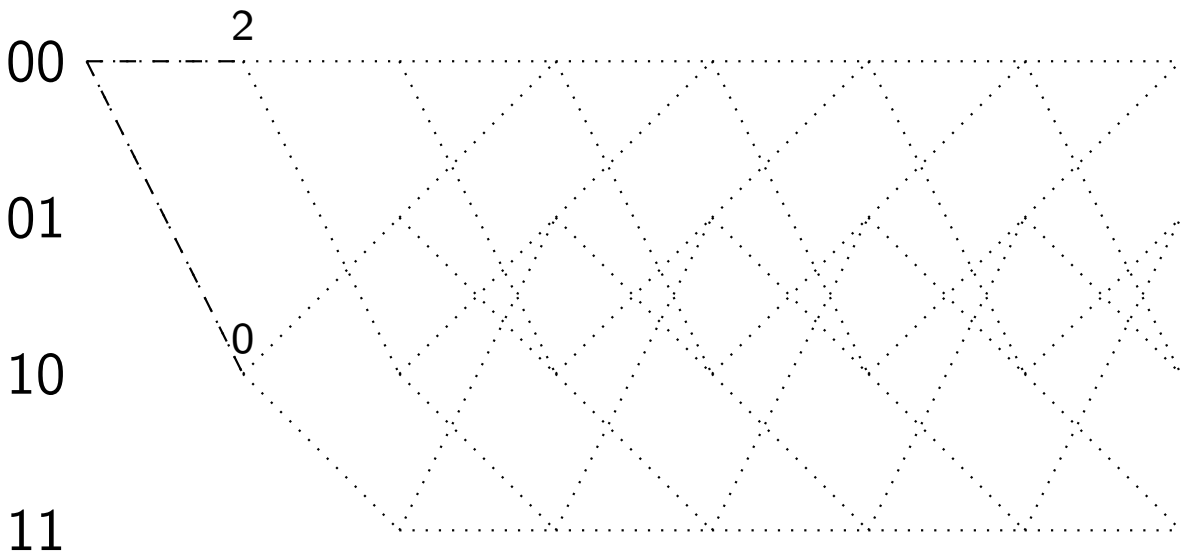
Algorithme de Viterbi (décodage hard)

L'encodeur utilise le code (5,7), on reçoit $r = [11010001001111]$



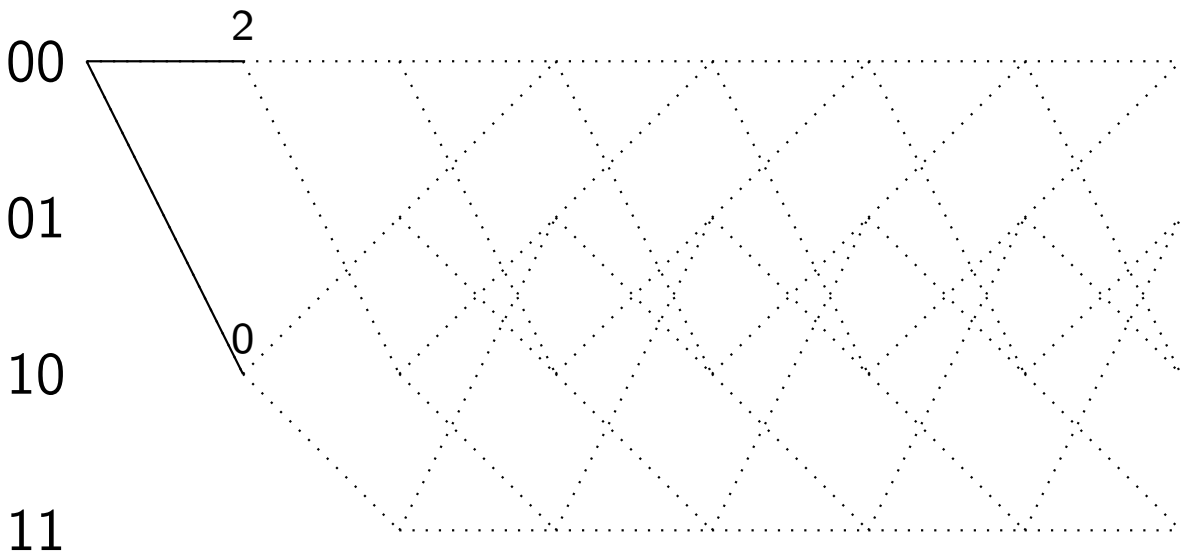
Algorithme de Viterbi (décodage hard)

L'encodeur utilise le code (5,7), on reçoit $r = [11010001001111]$



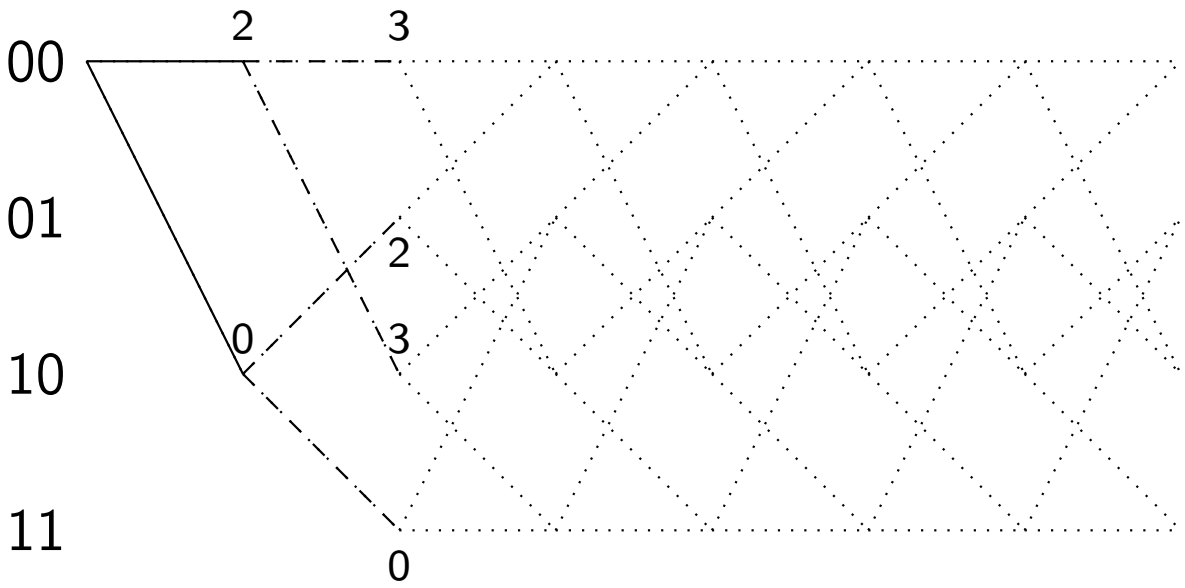
Algorithme de Viterbi (décodage hard)

L'encodeur utilise le code (5,7), on reçoit $r = [11010001001111]$



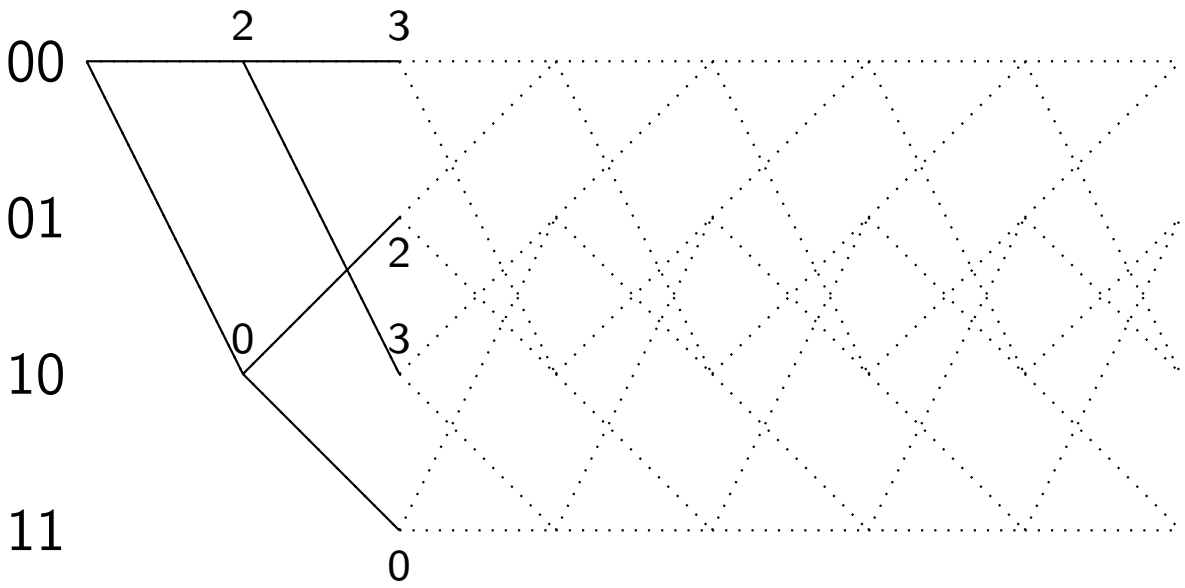
Algorithme de Viterbi (décodage hard)

L'encodeur utilise le code (5,7), on reçoit $r = [11010001001111]$



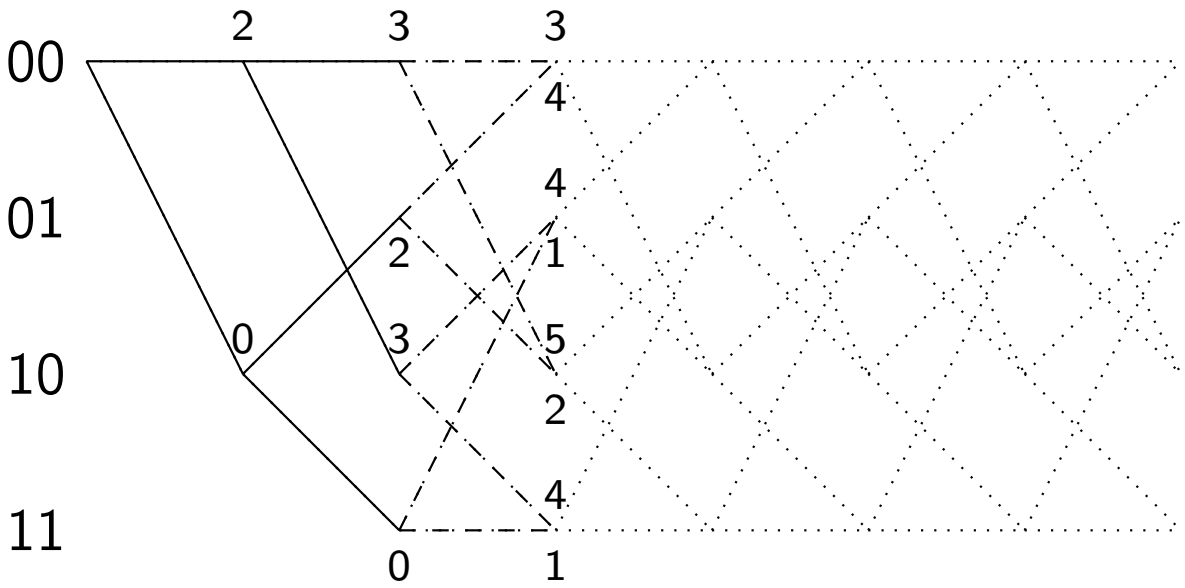
Algorithme de Viterbi (décodage hard)

L'encodeur utilise le code (5,7), on reçoit $r = [11010001001111]$



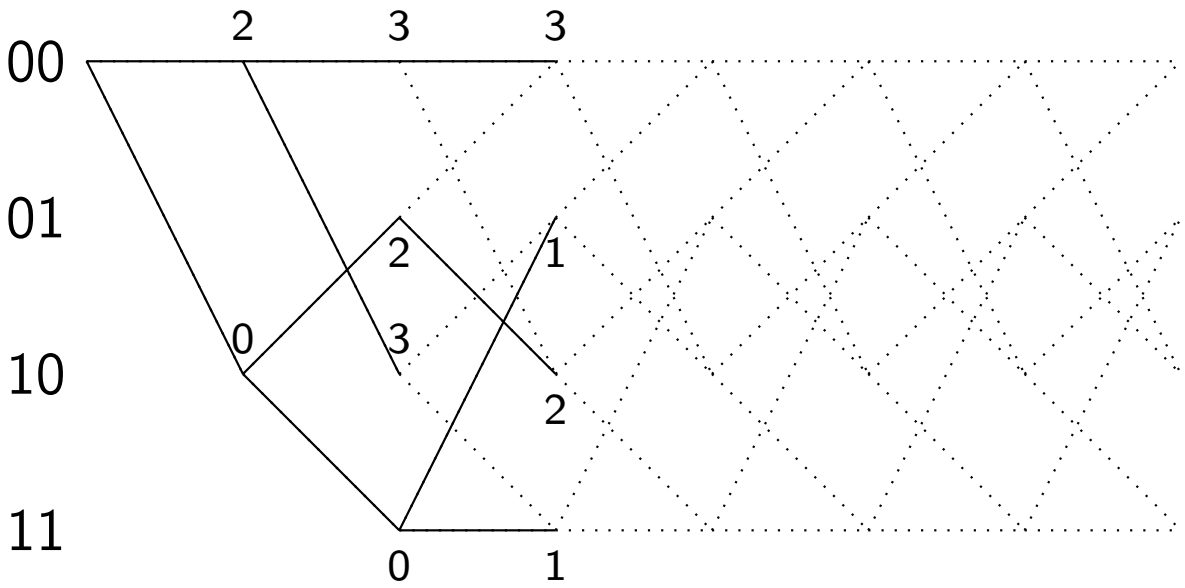
Algorithme de Viterbi (décodage hard)

L'encodeur utilise le code (5,7), on reçoit $r = [11010001001111]$



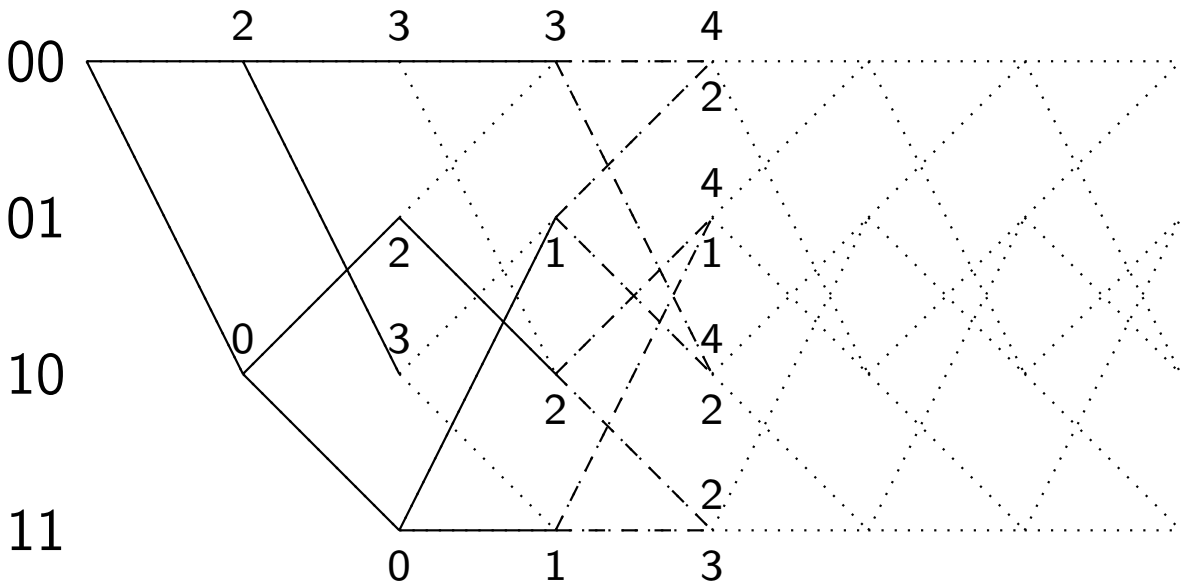
Algorithme de Viterbi (décodage hard)

L'encodeur utilise le code (5,7), on reçoit $r = [11010001001111]$



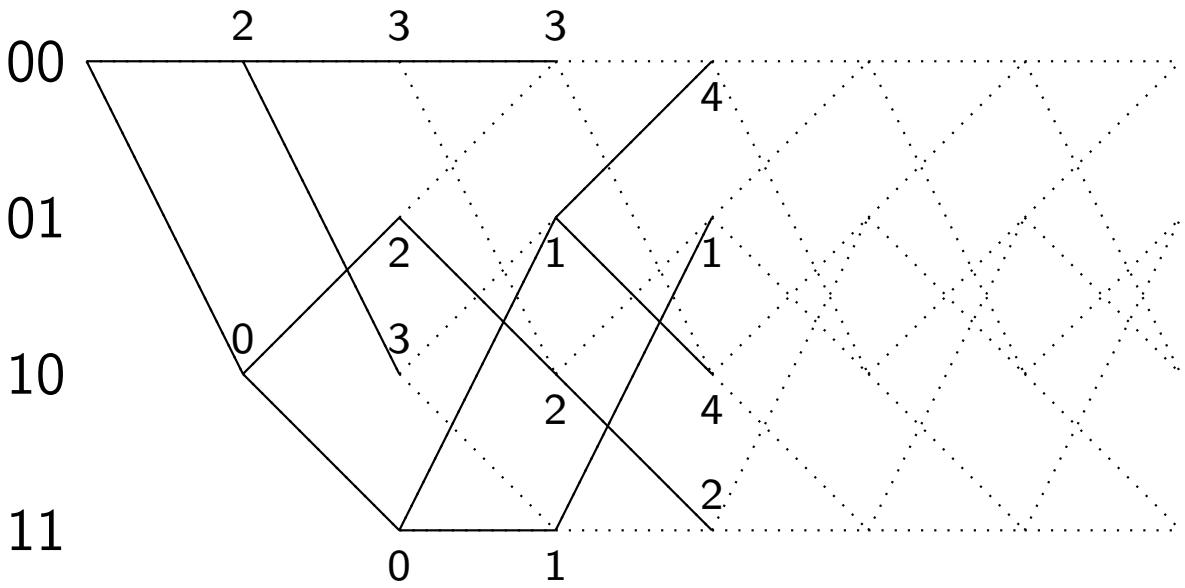
Algorithme de Viterbi (décodage hard)

L'encodeur utilise le code (5,7), on reçoit $r = [11010001001111]$



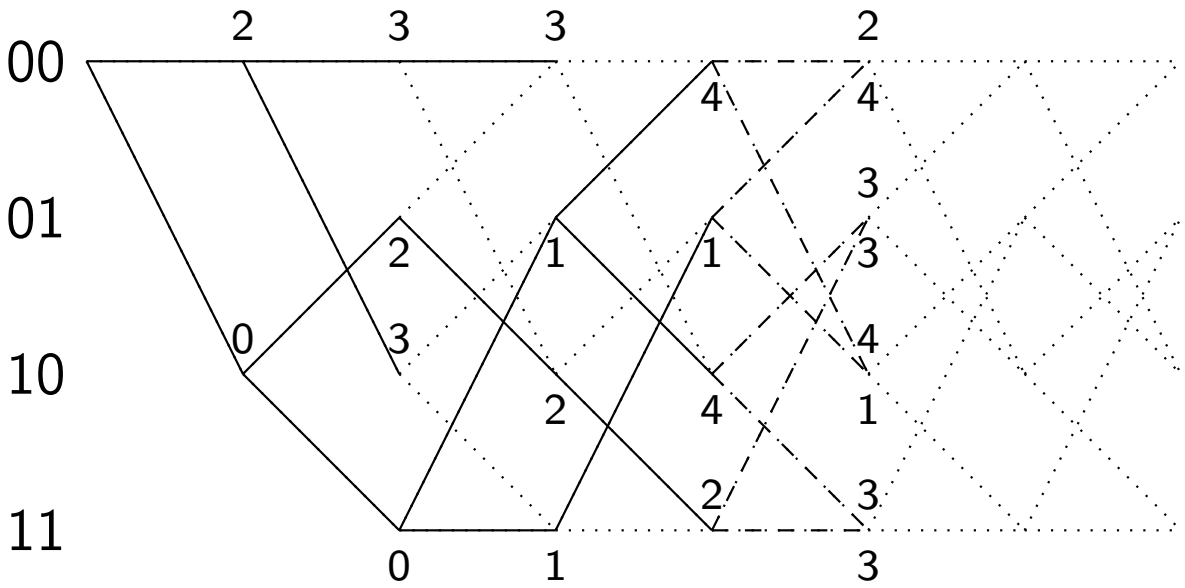
Algorithme de Viterbi (décodage hard)

L'encodeur utilise le code (5,7), on reçoit $r = [11010001001111]$



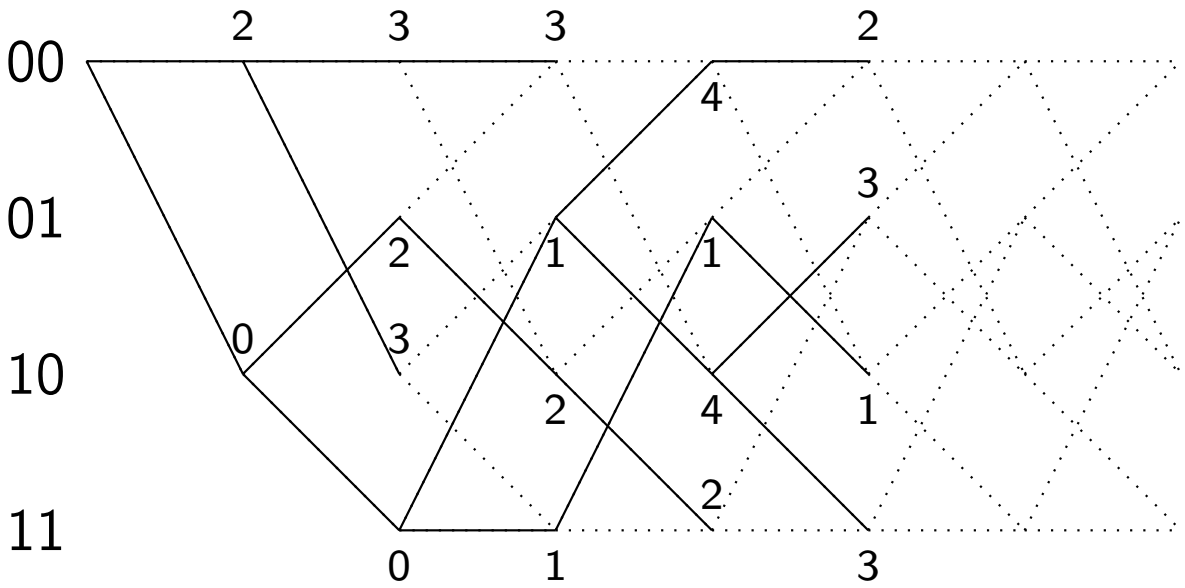
Algorithme de Viterbi (décodage hard)

L'encodeur utilise le code (5,7), on reçoit $r = [11010001001111]$



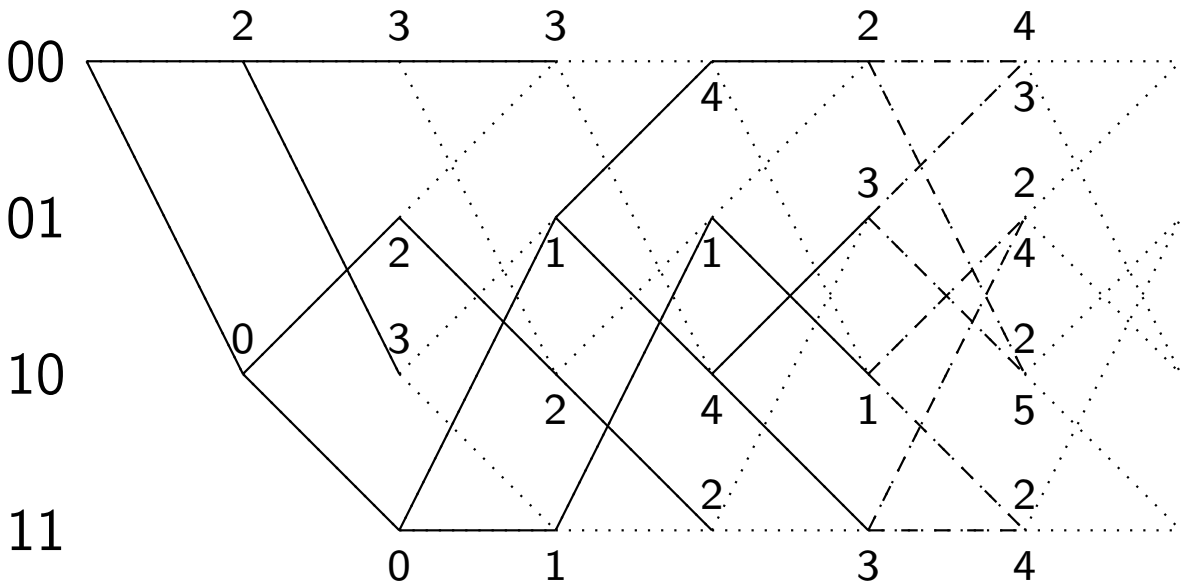
Algorithme de Viterbi (décodage hard)

L'encodeur utilise le code (5,7), on reçoit $r = [11010001001111]$



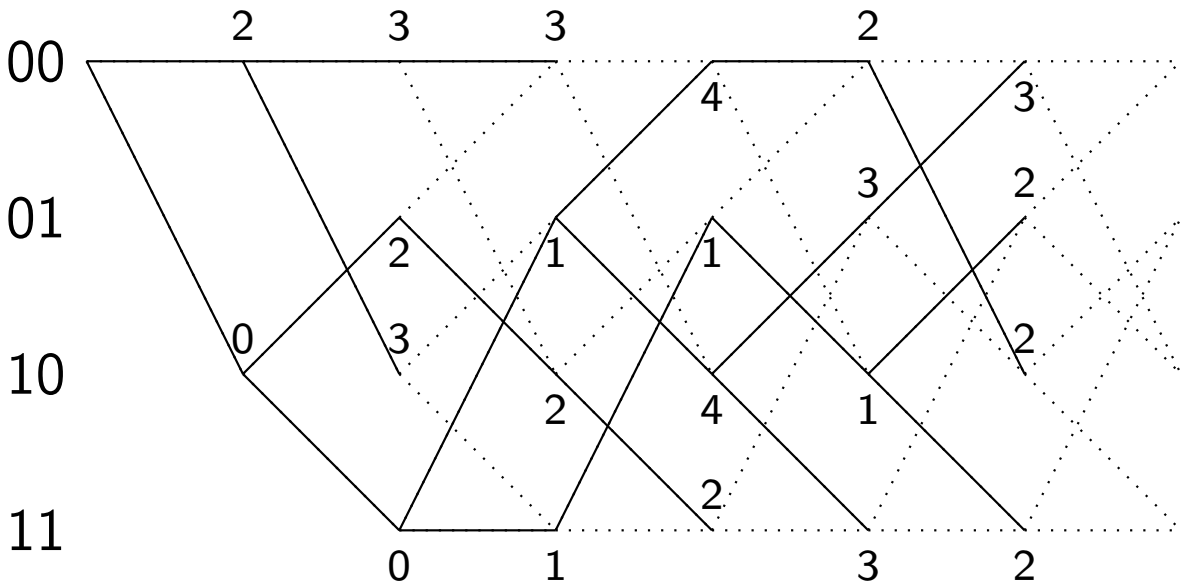
Algorithme de Viterbi (décodage hard)

L'encodeur utilise le code (5,7), on reçoit $r = [11010001001111]$



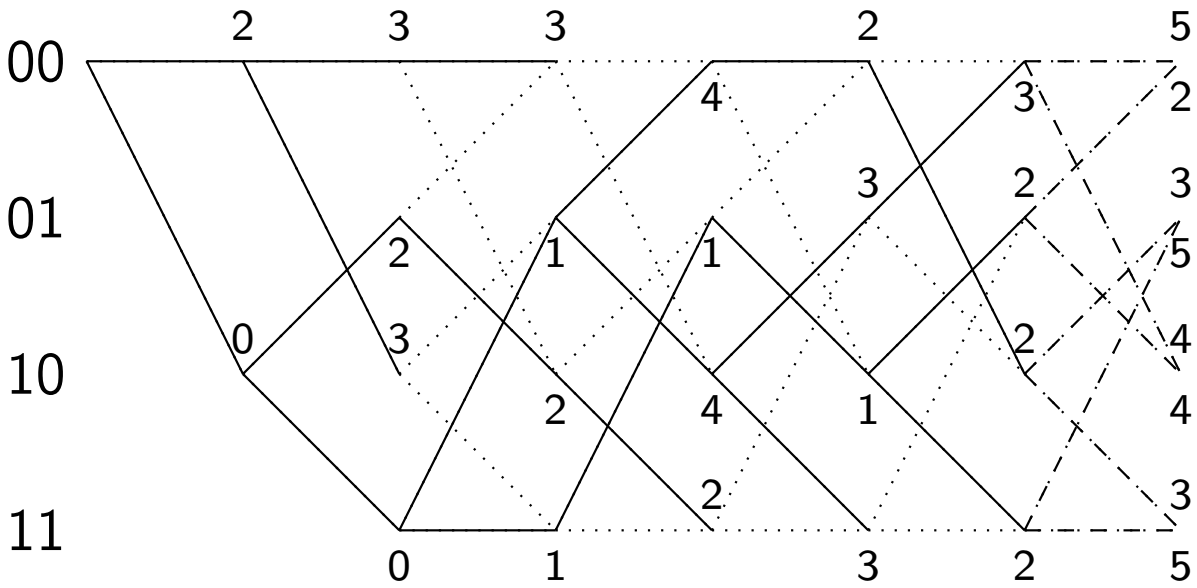
Algorithme de Viterbi (décodage hard)

L'encodeur utilise le code (5,7), on reçoit $r = [11010001001111]$



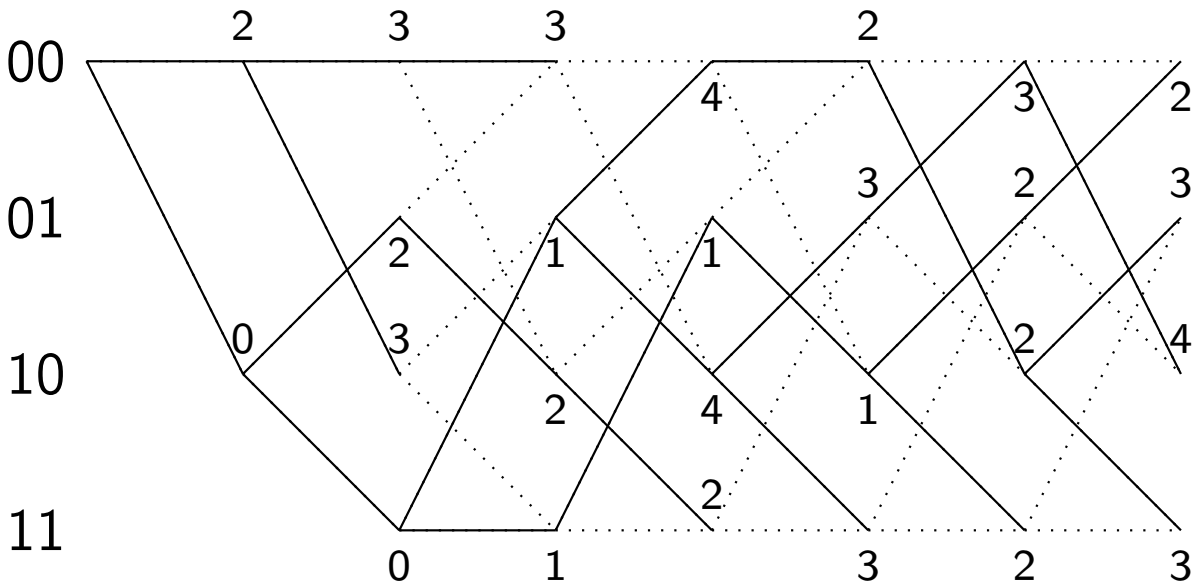
Algorithme de Viterbi (décodage hard)

L'encodeur utilise le code (5,7), on reçoit $r = [11010001001111]$



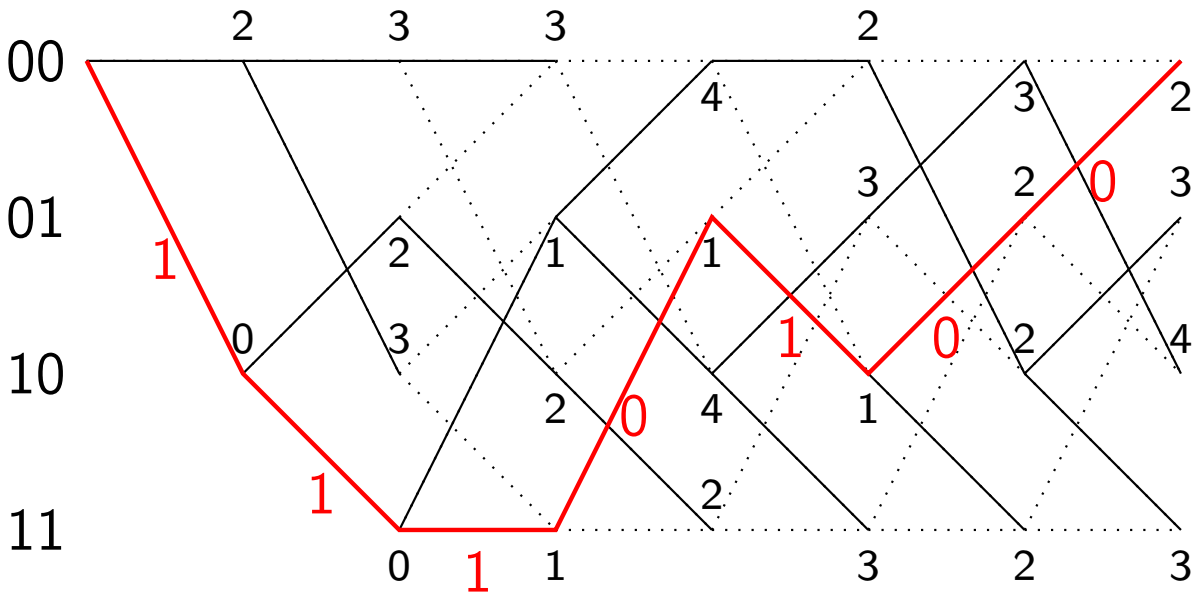
Algorithme de Viterbi (décodage hard)

L'encodeur utilise le code (5,7), on reçoit $r = [11010001001111]$



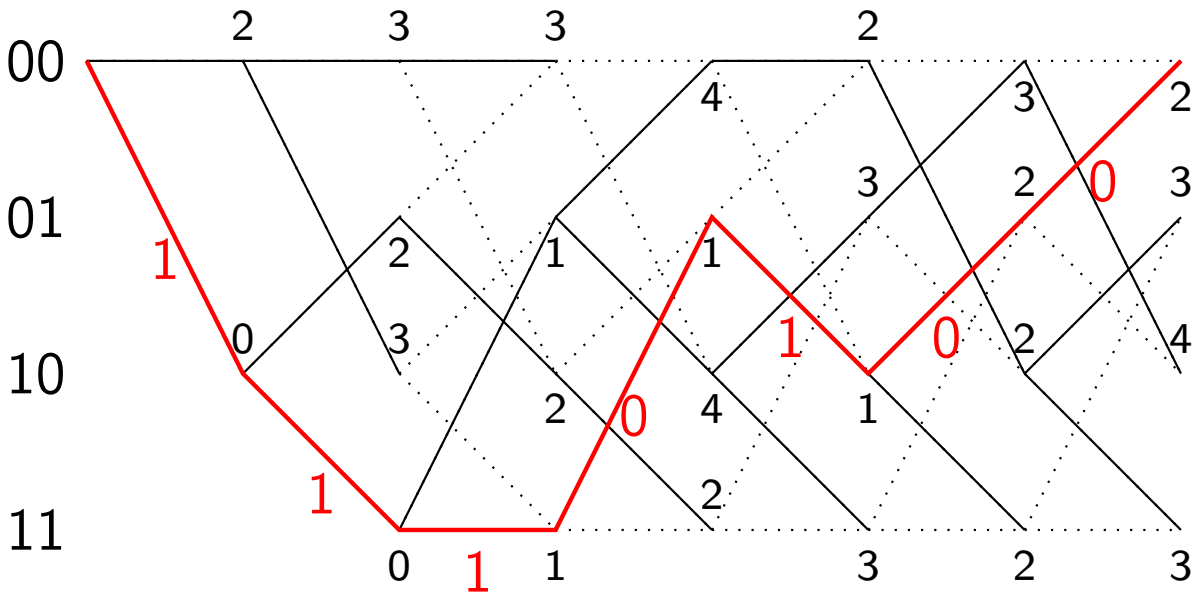
Algorithme de Viterbi (décodage hard)

L'encodeur utilise le code (5,7), on reçoit $r = [11010001001111]$



Algorithme de Viterbi (décodage hard)

L'encodeur utilise le code (5,7), on reçoit $r = [11010001001111]$



Le décodeur détermine alors:

$\hat{c} = [11011001001011]$

$\hat{s} = [1110100]$

Algorithme de Viterbi (décodage soft)

Sur le canal AWGN, si r_i et c_i ont une longueur de n alors:

$$p(r_i|c_i) = \prod_{i=0}^{n-1} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y_i - x_i)^2}{2\sigma^2}}$$

En logarithme:

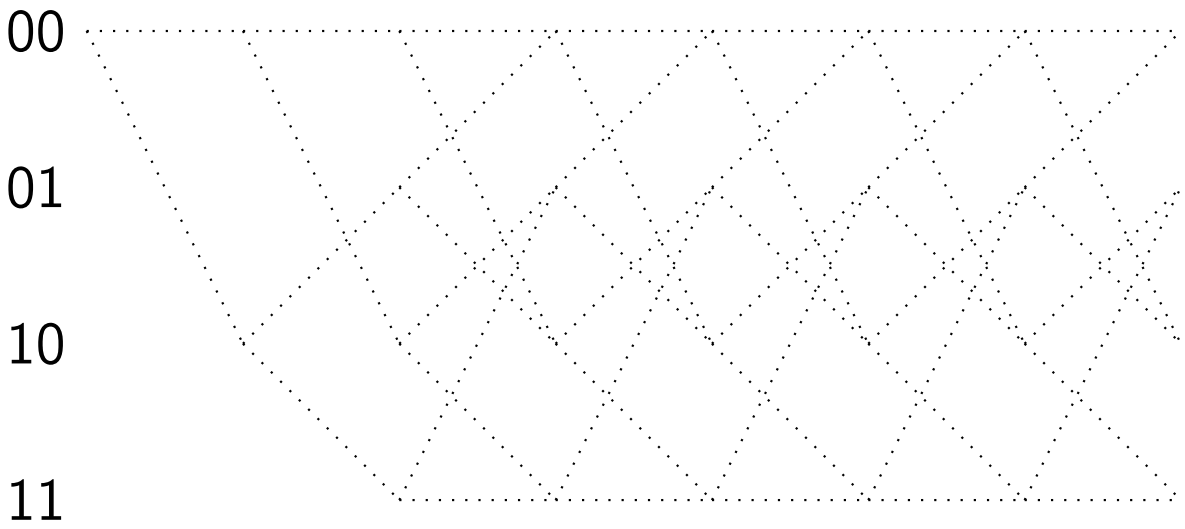
$$\log p(r_i|c_i) = -\frac{1}{2\sigma^2} \sum_{i=0}^{n-1} (y_i - x_i)^2 + n \log \frac{1}{\sqrt{2\pi\sigma^2}}$$

$$\propto -\sum_{i=0}^{n-1} (y_i - x_i)^2$$

La séquence $\hat{c}_{ML}$ est celle qui minimise la distance euclidienne avec la séquence reçue.

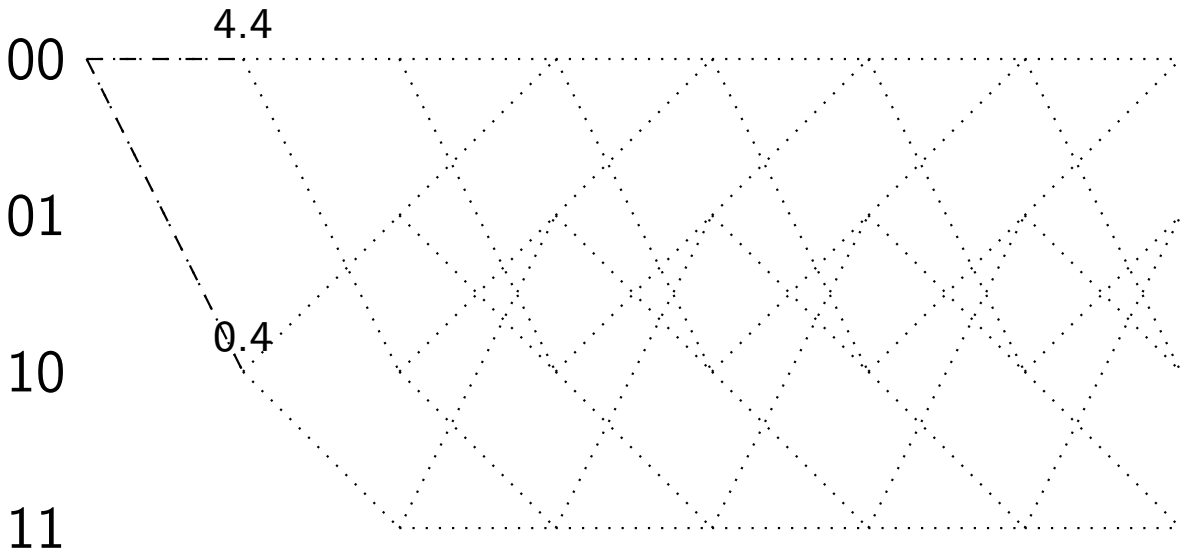
Algorithme de Viterbi (décodage soft)

L'encodeur utilise le code (5,7), on reçoit $r = [-1.1, -1.3, 0.7, 0.2, 1.2, 0.9, -0.5, -0.4, 1.4, -0.1, 0.8, 0, -1.5, -1]$



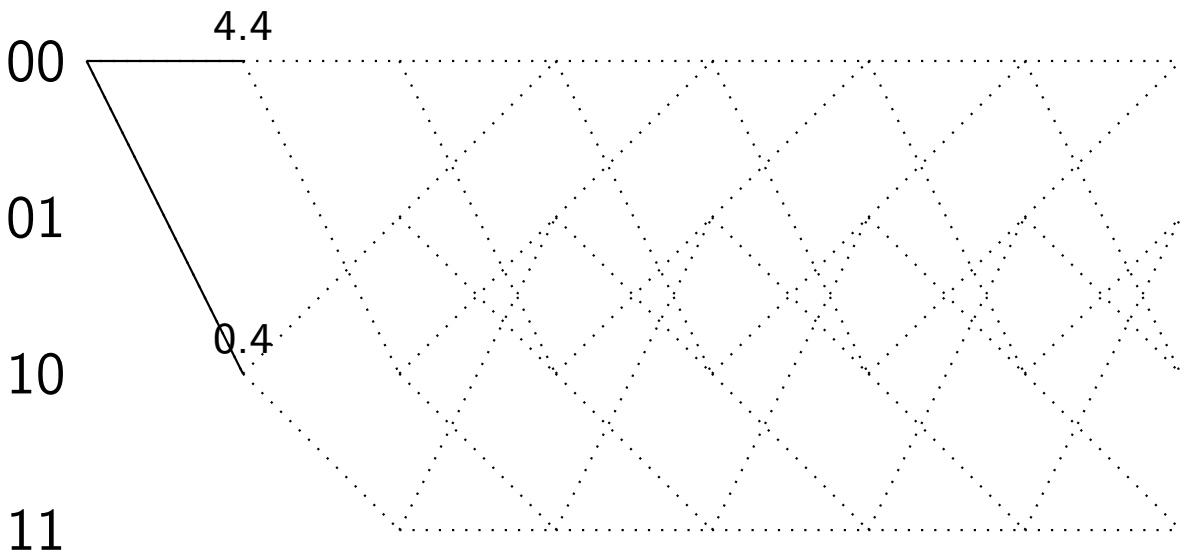
Algorithme de Viterbi (décodage soft)

L'encodeur utilise le code (5,7), on reçoit $r =$
$[-1.1, -1.3, 0.7, 0.2, 1.2, 0.9, -0.5, -0.4, 1.4, -0.1, 0.8, 0, -1.5, -1]$



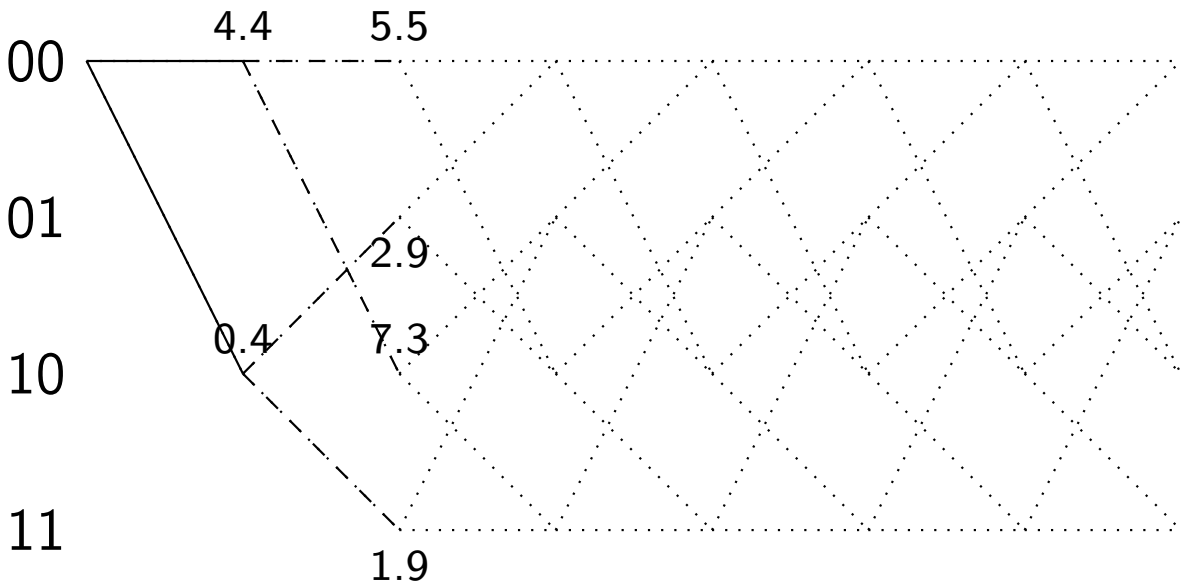
Algorithme de Viterbi (décodage soft)

L'encodeur utilise le code (5,7), on reçoit $r =$
$[-1.1, -1.3, 0.7, 0.2, 1.2, 0.9, -0.5, -0.4, 1.4, -0.1, 0.8, 0, -1.5, -1]$



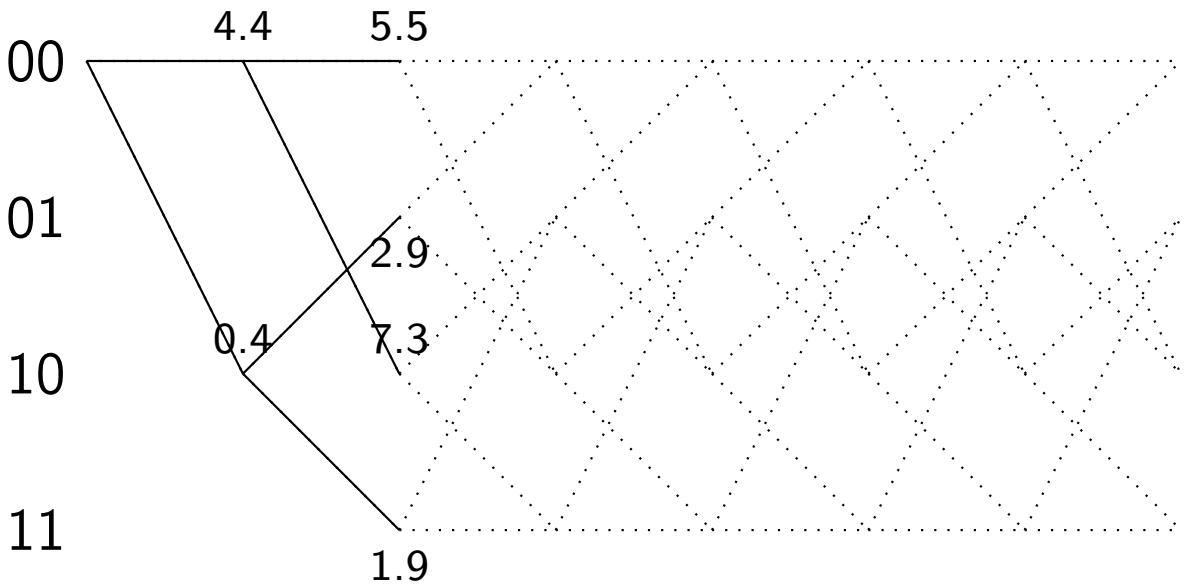
Algorithme de Viterbi (décodage soft)

L'encodeur utilise le code (5,7), on reçoit $r = [-1.1, -1.3, 0.7, 0.2, 1.2, 0.9, -0.5, -0.4, 1.4, -0.1, 0.8, 0, -1.5, -1]$



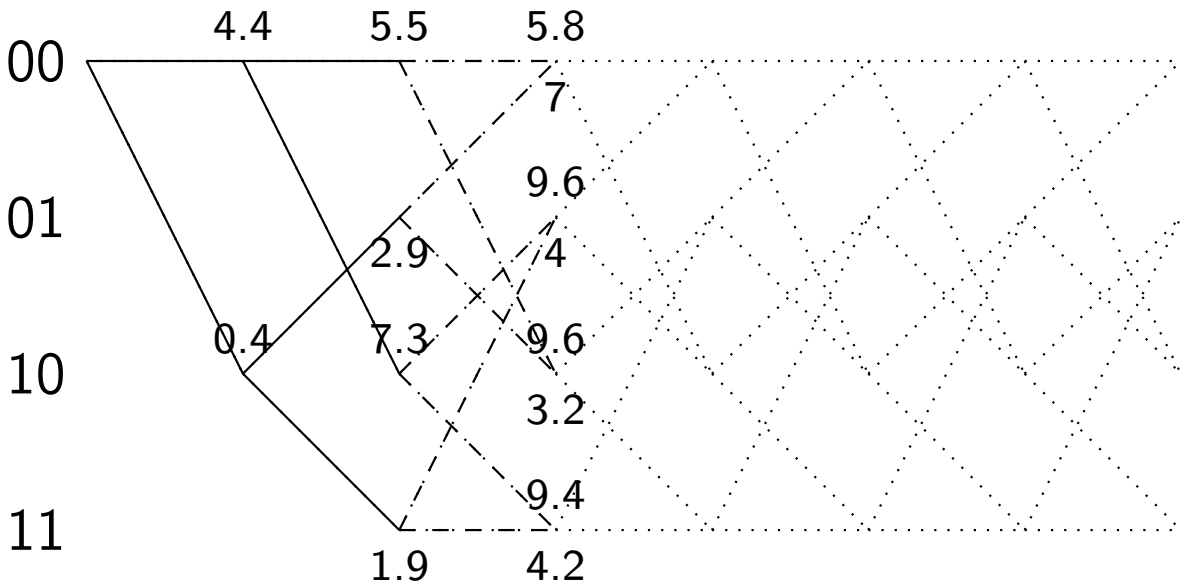
Algorithme de Viterbi (décodage soft)

L'encodeur utilise le code (5,7), on reçoit $r = [-1.1, -1.3, 0.7, 0.2, 1.2, 0.9, -0.5, -0.4, 1.4, -0.1, 0.8, 0, -1.5, -1]$



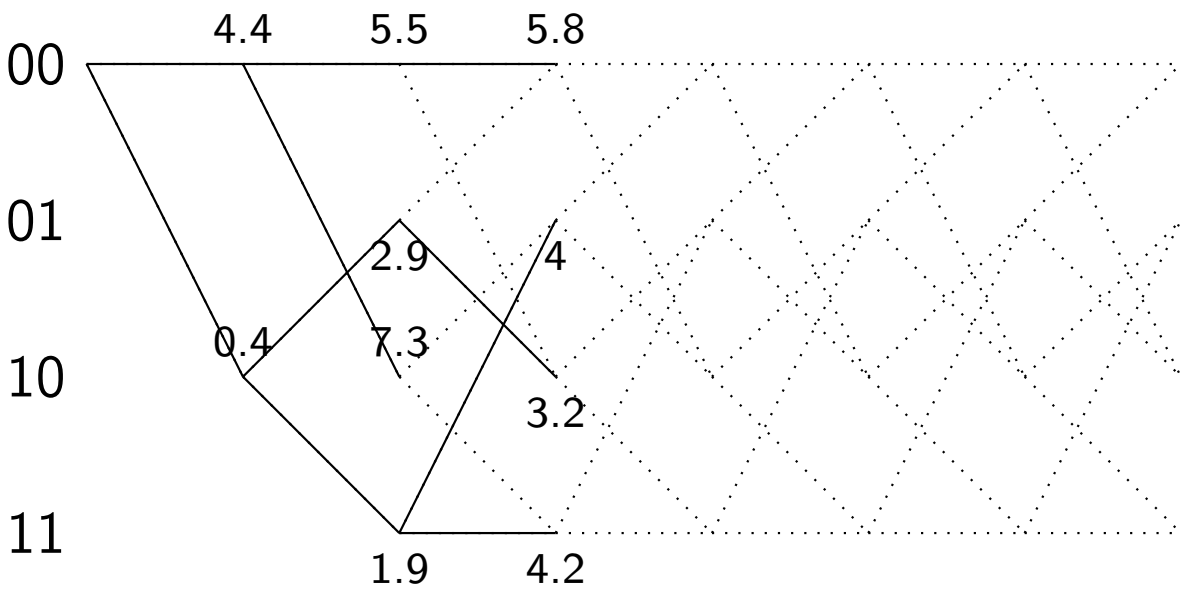
Algorithme de Viterbi (décodage soft)

L'encodeur utilise le code (5,7), on reçoit $r = [-1.1, -1.3, 0.7, 0.2, 1.2, 0.9, -0.5, -0.4, 1.4, -0.1, 0.8, 0, -1.5, -1]$



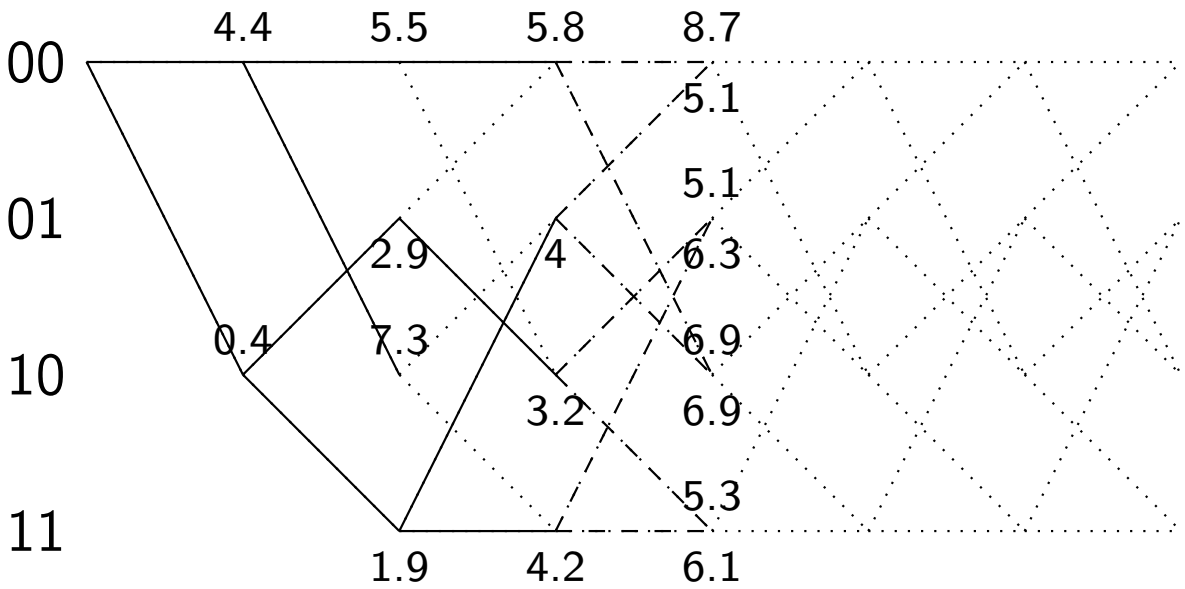
Algorithme de Viterbi (décodage soft)

L'encodeur utilise le code (5,7), on reçoit $r = [-1.1, -1.3, 0.7, 0.2, 1.2, 0.9, -0.5, -0.4, 1.4, -0.1, 0.8, 0, -1.5, -1]$



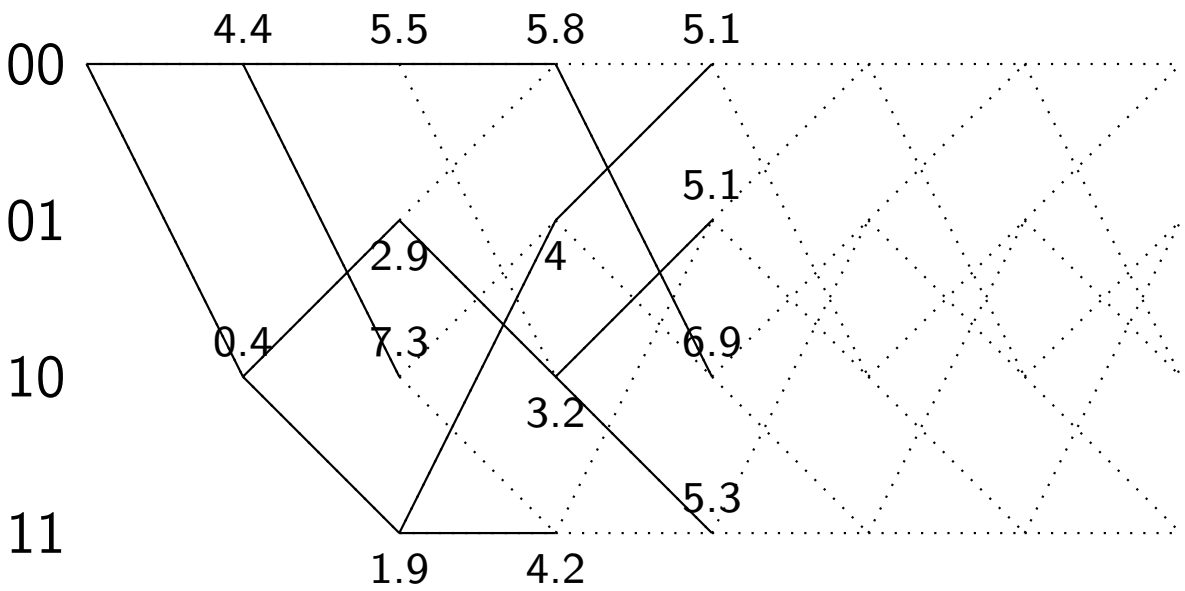
Algorithme de Viterbi (décodage soft)

L'encodeur utilise le code (5,7), on reçoit $r = [-1.1, -1.3, 0.7, 0.2, 1.2, 0.9, -0.5, -0.4, 1.4, -0.1, 0.8, 0, -1.5, -1]$



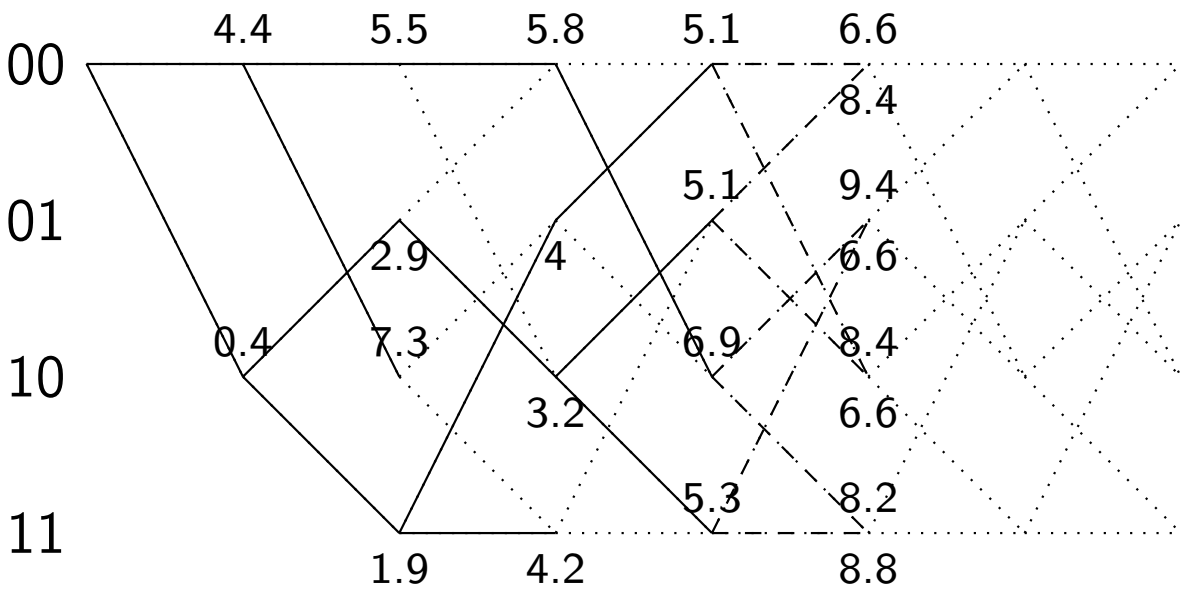
Algorithme de Viterbi (décodage soft)

L'encodeur utilise le code (5,7), on reçoit $r = [-1.1, -1.3, 0.7, 0.2, 1.2, 0.9, -0.5, -0.4, 1.4, -0.1, 0.8, 0, -1.5, -1]$



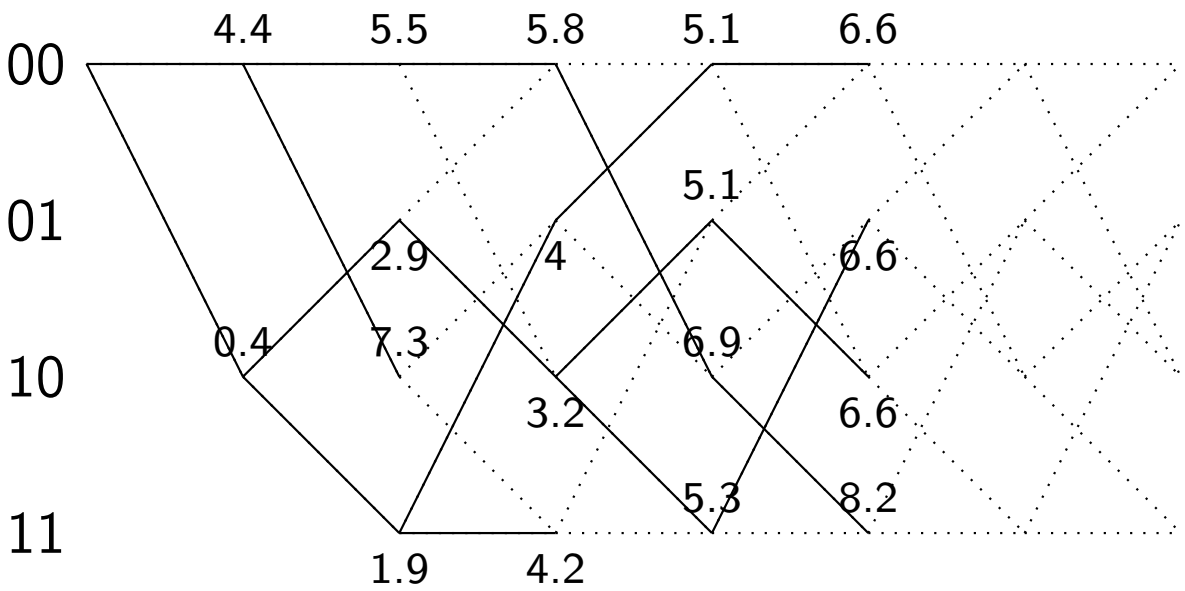
Algorithme de Viterbi (décodage soft)

L'encodeur utilise le code (5,7), on reçoit $r = [-1.1, -1.3, 0.7, 0.2, 1.2, 0.9, -0.5, -0.4, 1.4, -0.1, 0.8, 0, -1.5, -1]$



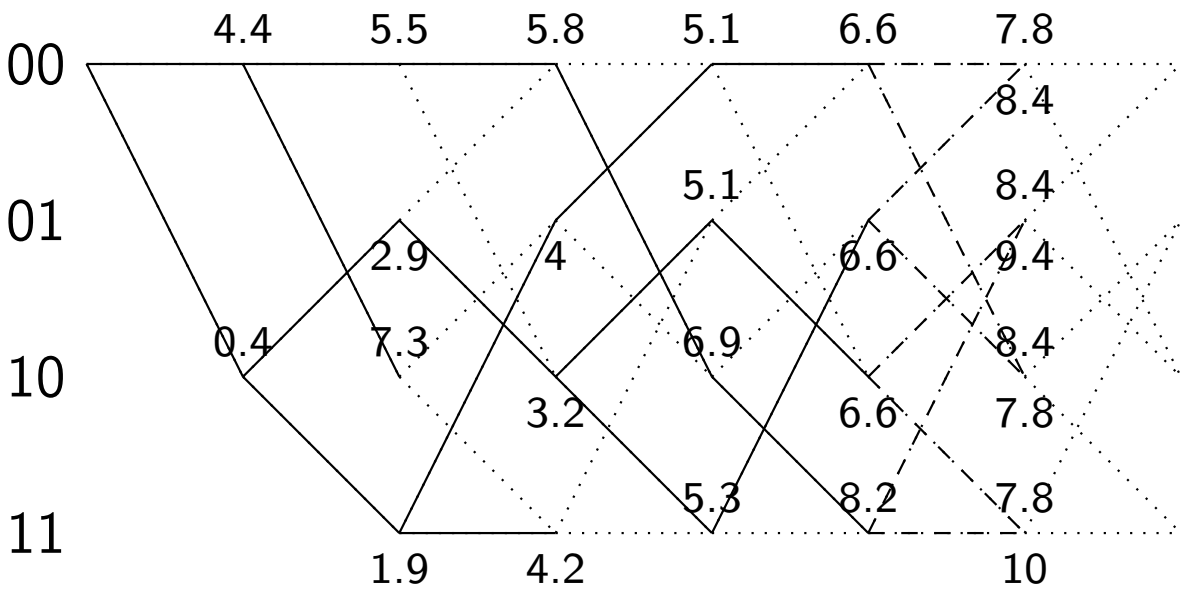
Algorithme de Viterbi (décodage soft)

L'encodeur utilise le code (5,7), on reçoit $r = [-1.1, -1.3, 0.7, 0.2, 1.2, 0.9, -0.5, -0.4, 1.4, -0.1, 0.8, 0, -1.5, -1]$



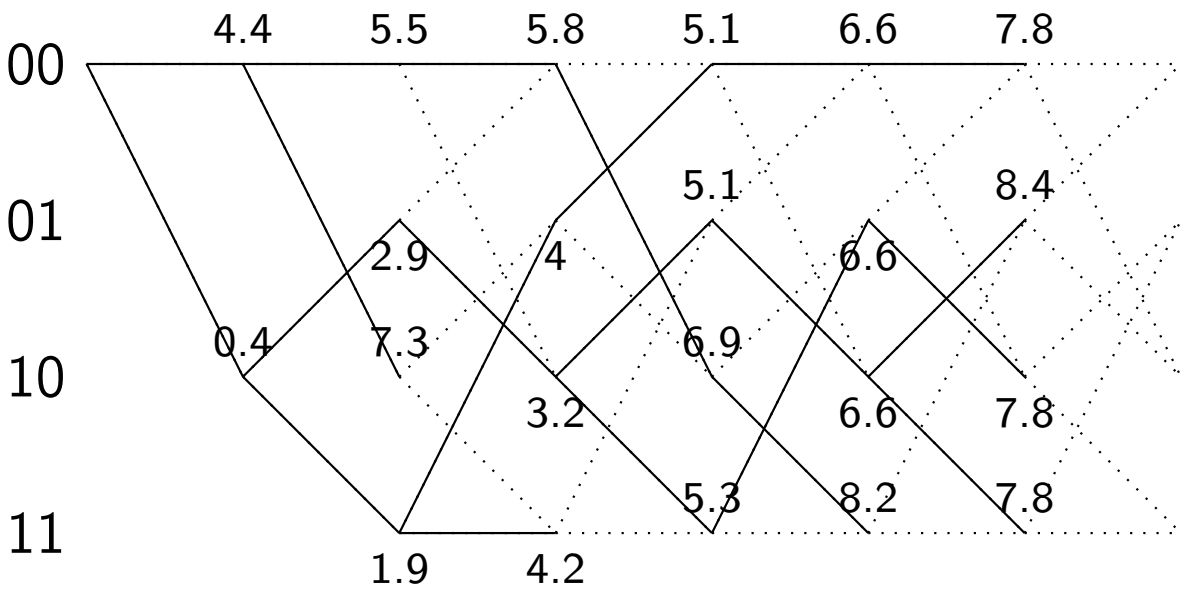
Algorithme de Viterbi (décodage soft)

L'encodeur utilise le code (5,7), on reçoit $r = [-1.1, -1.3, 0.7, 0.2, 1.2, 0.9, -0.5, -0.4, 1.4, -0.1, 0.8, 0, -1.5, -1]$



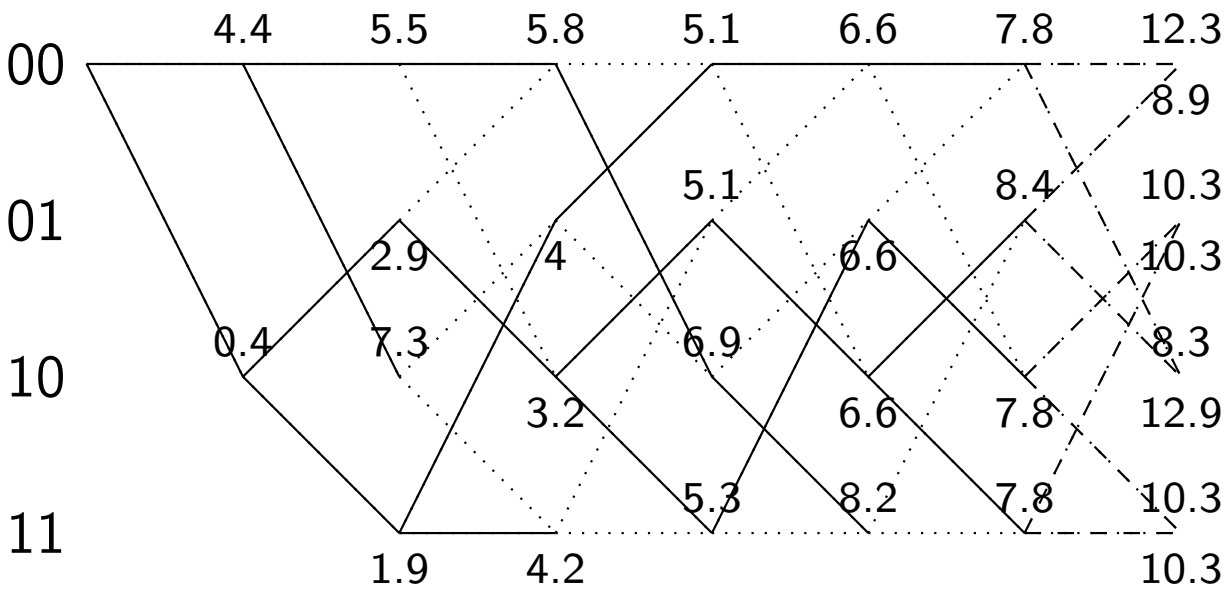
Algorithme de Viterbi (décodage soft)

L'encodeur utilise le code (5,7), on reçoit $r = [-1.1, -1.3, 0.7, 0.2, 1.2, 0.9, -0.5, -0.4, 1.4, -0.1, 0.8, 0, -1.5, -1]$



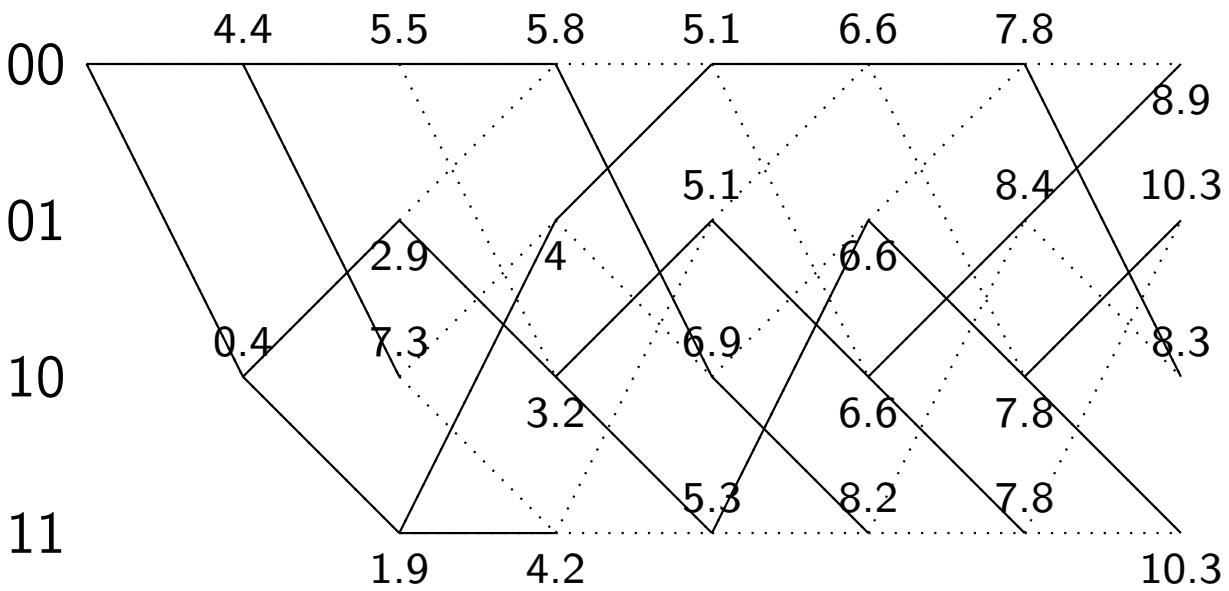
Algorithme de Viterbi (décodage soft)

L'encodeur utilise le code (5,7), on reçoit $r = [-1.1, -1.3, 0.7, 0.2, 1.2, 0.9, -0.5, -0.4, 1.4, -0.1, 0.8, 0, -1.5, -1]$



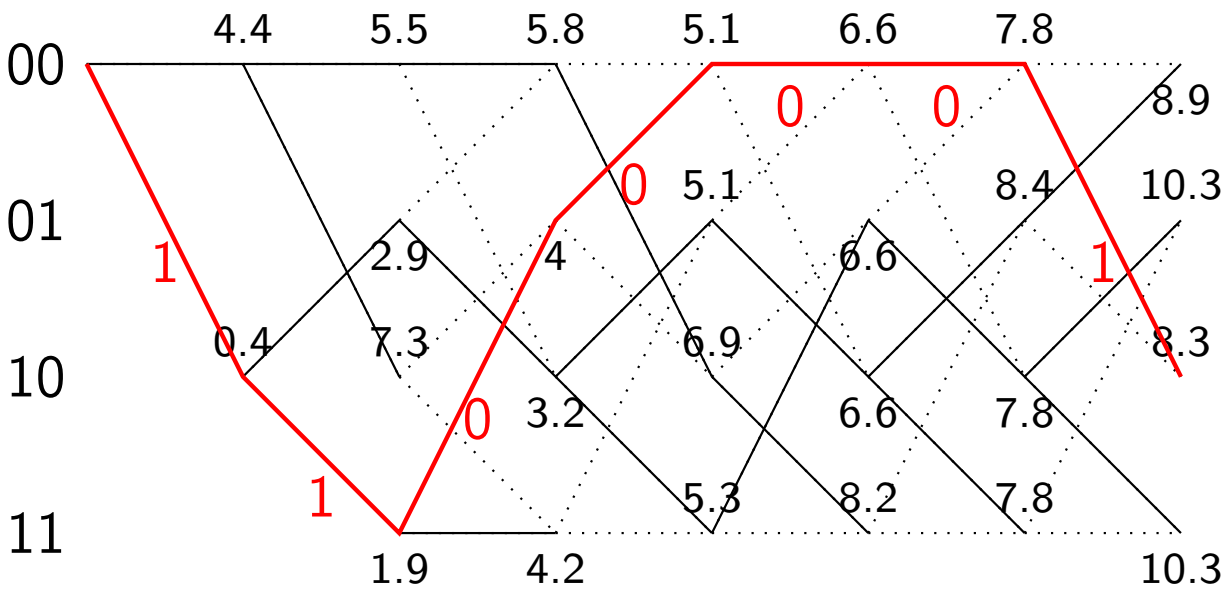
Algorithme de Viterbi (décodage soft)

L'encodeur utilise le code (5,7), on reçoit $r = [-1.1, -1.3, 0.7, 0.2, 1.2, 0.9, -0.5, -0.4, 1.4, -0.1, 0.8, 0, -1.5, -1]$



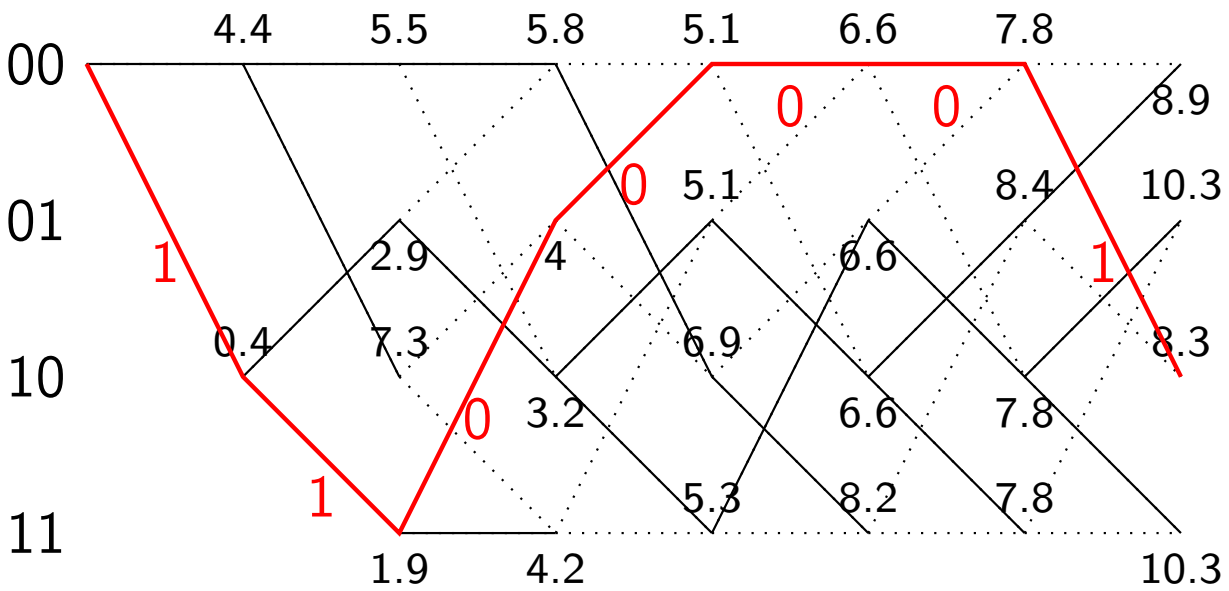
Algorithme de Viterbi (décodage soft)

L'encodeur utilise le code (5,7), on reçoit $r = [-1.1, -1.3, 0.7, 0.2, 1.2, 0.9, -0.5, -0.4, 1.4, -0.1, 0.8, 0, -1.5, -1]$



Algorithme de Viterbi (décodage soft)

L'encodeur utilise le code (5,7), on reçoit $r = [-1.1, -1.3, 0.7, 0.2, 1.2, 0.9, -0.5, -0.4, 1.4, -0.1, 0.8, 0, -1.5, -1]$



Le décodeur détermine alors:
$\hat{c} = [11011001001011]$ et $\hat{s} = [1110100]$

Considérations pratiques

- En théorie, il est nécessaire conserver la métrique de chaque chemin en totalité (de 0 à n) ce qui augmente la latence.
- En pratique, les chemins ne divergent que sur une longueur de $5l_c$. On met en place un fenêtrage permettant de diminuer l'empreinte mémoire et la latence
- De plus, les distances sont quantifiées en virgule fixe sur un nombre de bits donnés ce qui entraîne une (légère) perte de performance

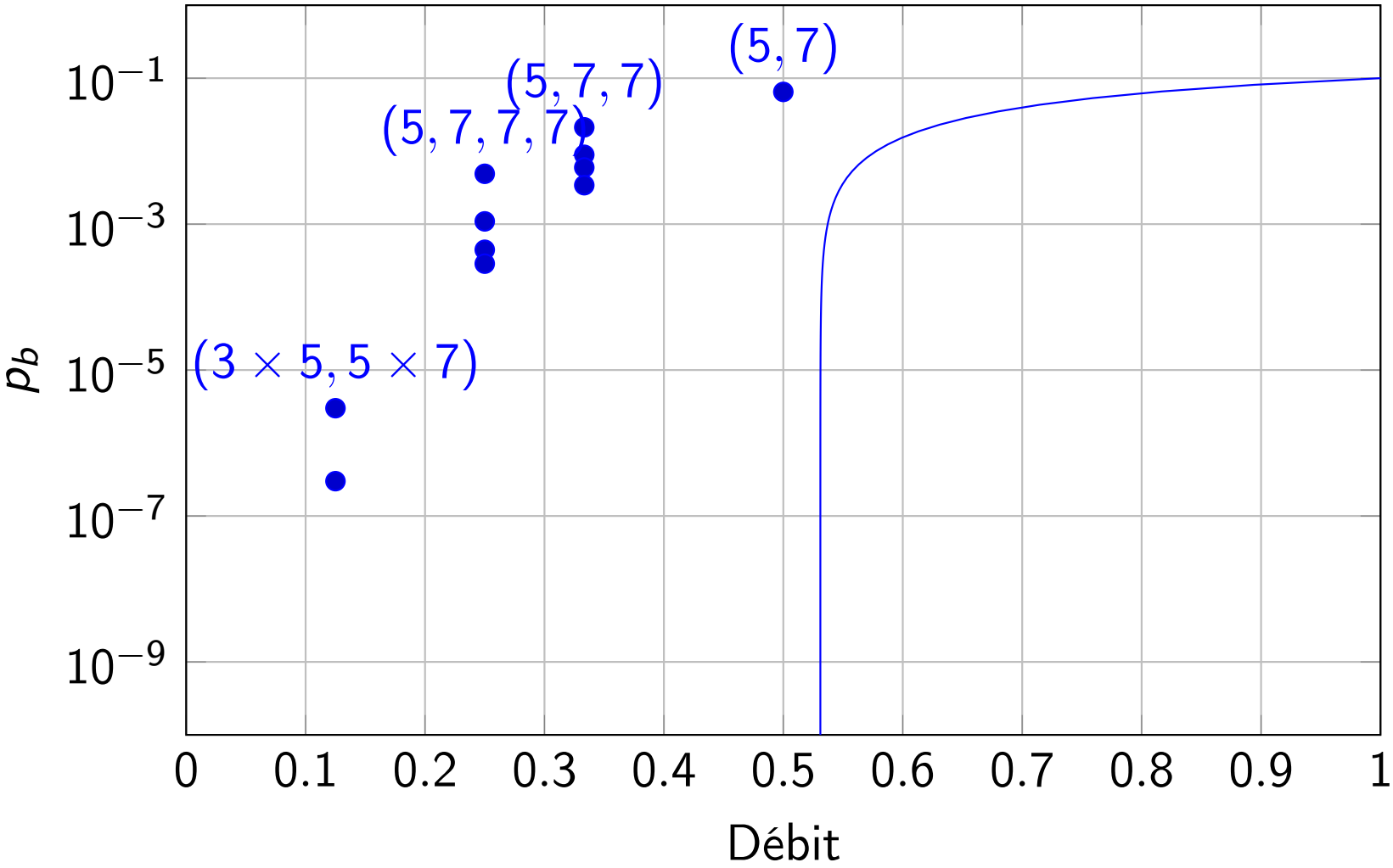
Terminaison du treillis

Dans la plupart des cas, l'état du registre est initialisé à 0 avant la transmission. A la fin de la transmission, l'encodeur ajoute $l_c - 1$ zéros, ce qui permet de ramener l'état du registre à 0 (et diminue légèrement r).

Dans certains cas, le registre n'est par réinitialisé entre chaque message.

Enfin dans d'autres cas, le registre est chargé avec les $l_c - 1$ bits du message s ce qui permet au décodeur de savoir que l'état de départ est le même que l'état final (mais reste inconnu).

Performances des codes convolutifs



Conclusion

Avantages:

- Décodage hard et soft
- Décodage optimal et rapide pour un nombre d'états faibles
- Composants de base des turbocodes

Inconvénients:

- Nécessité d'augmenter le nombre d'état pour atteindre la capacité
- Complexité de l'algorithme de Viterbi devient trop importante
- Parallélisation difficile

Histoire

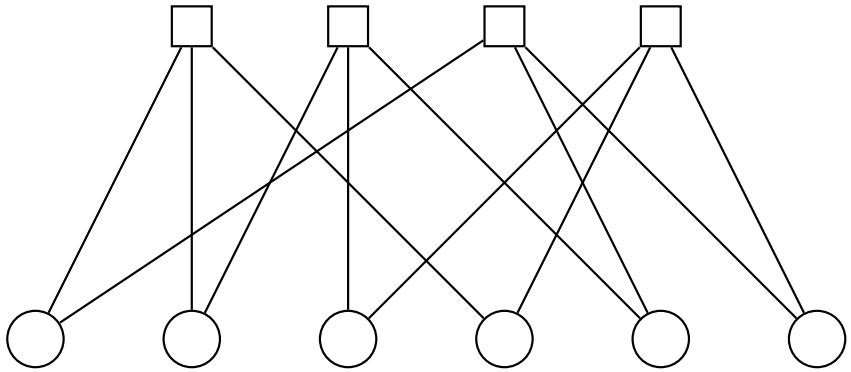
- Découverts par Gallager en 62 puis oubliés...
- et redécouverts par Mackay et Neal en 96.
- Codes en bloc linéaires dont la matrice de parité est creuse
- Codes les plus performants (atteignant la capacité)

Les codes LDPC sont caractérisés par un algorithme de décodage sous optimal mais possédant une complexité extrêmement faible ce qui permet de considérer des longueurs de blocs de plusieurs milliers.

Graphe de Tanner

Toute matrice de parité H peut être représentées sous la forme d'un graphe de Tanner.

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



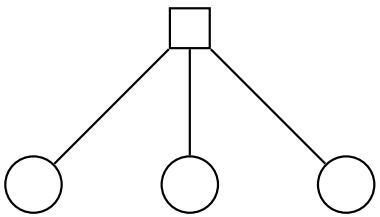
Le graphe contient n nœuds de variable et $m = n - k$ nœuds de contrôle. De plus, le code LDPC est dit régulier s'il contient le même nombre de 1 dans chaque ligne ω_r et le même nombre de 1 dans chaque colonnes ω_c.

Graphe de Tanner

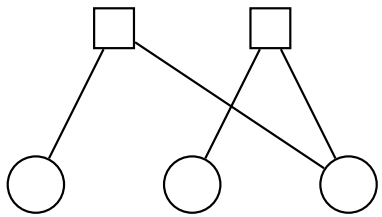
Trouvez les graphes de Tanner du bit de parité $n = 3$, du code R_3 et du code de Hamming $(7, 4)$ et vérifiez que la séquence c=[1000011] est bien un mot de code de $H(7, 4)$

Graphe de Tanner

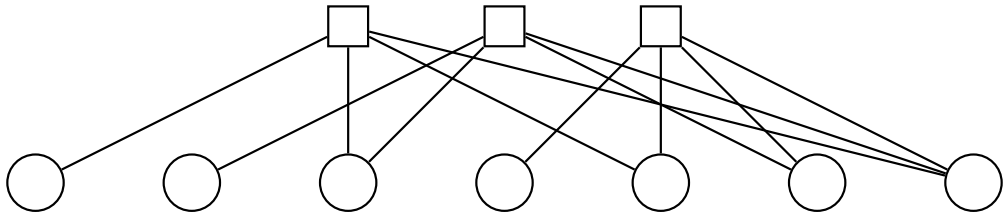
Trouvez les graphes de Tanner du bit de parité $n = 3$, du code R_3 et du code de Hamming $(7, 4)$ et vérifiez que la séquence c=[1000011] est bien un mot de code de $H(7, 4)$



Graphe du bit de parité $n = 3$



Graphe de R_3



Graphe de $H(7, 4)$

Ces codes ne sont pas des codes LDPC car leurs matrices de parité ne sont pas creuses

Algorithme à échange de messages

Tous les algorithmes de décodage des codes LDPC sont itératifs et opèrent sur le graphe de Tanner en échangeant des messages entre les nœuds de variable et le nœuds de contrôle. De plus, il peuvent corriger au delà de la distance minimale du code (qui reste souvent inconnue).

Algorithme bit flipping qui échange des messages binaires (0 ou 1) issue d'une décision hard

Algorithme sum product qui échange des messages réels (LLR) issue d'une décision soft (ou hard)

Échange de message pour le BEC

Décodage bit flipping

On note E_{ji} le message envoyé du nœud de contrôle j vers le nœud de variable i et M_{ij} le message en sens inverse. L'initialisation est réalisée par $M_i = r_i$.

Règle des nœuds de variable

$$M_i = \begin{cases} r_i & \text{si la majorité des } E_{ji} = r_i,\ j \in A_i \\ \bar{r}_i & \text{sinon (majorité +1)} \end{cases} \tag{18}$$

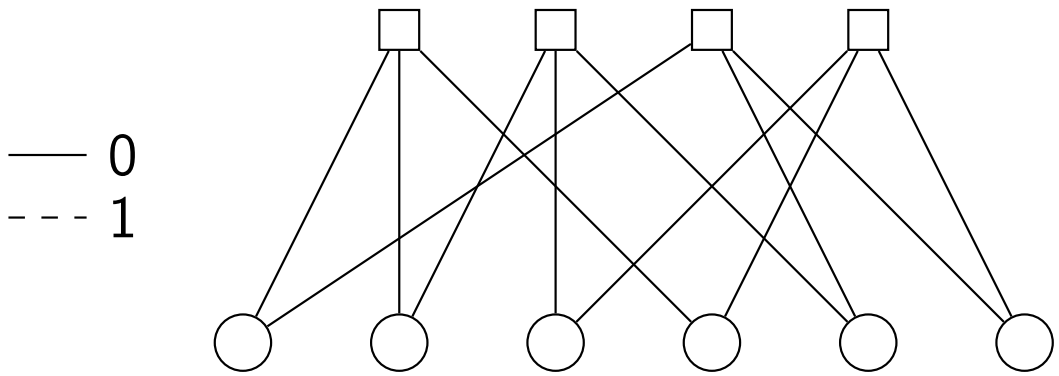
Règle des nœuds de contrôle

$$E_{ji} = \sum_{i' \in B_j,\ i' \neq i} M_{i'} \tag{19}$$

Arrêt lorsque toutes les équations de parité sont vérifiées

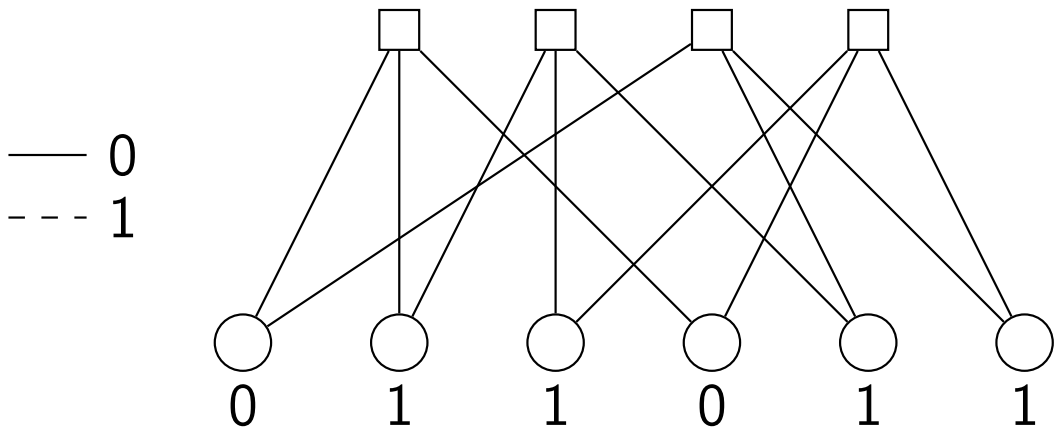
Décodage bit flipping

On reçoit le message $r = [011011]$ codé par un code possédant la matrice de parité régulière présentée précédemment.



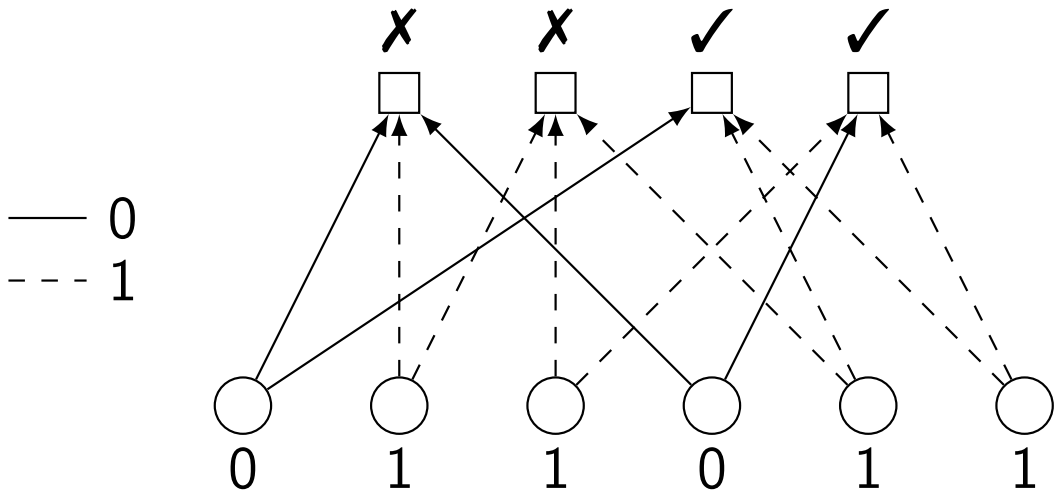
Décodage bit flipping

On reçoit le message $r = [011011]$ codé par un code possédant la matrice de parité régulière présentée précédemment.



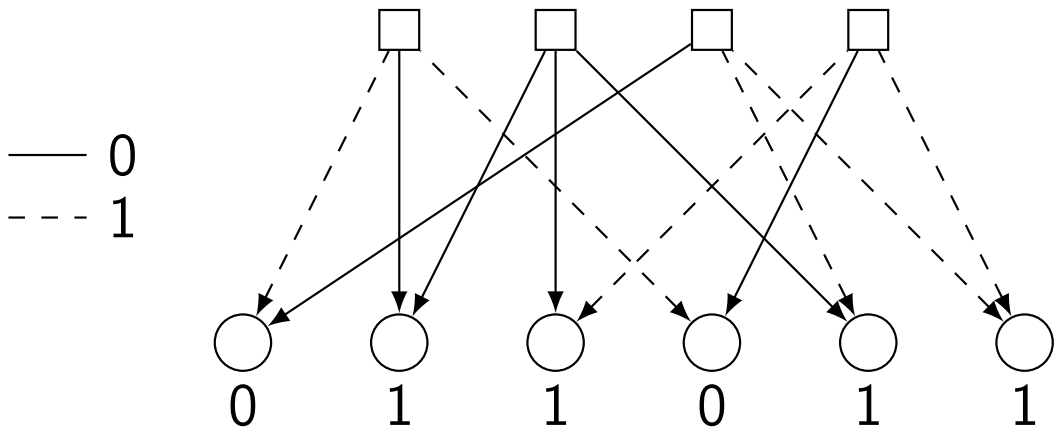
Décodage bit flipping

On reçoit le message $r = [011011]$ codé par un code possédant la matrice de parité régulière présentée précédemment.



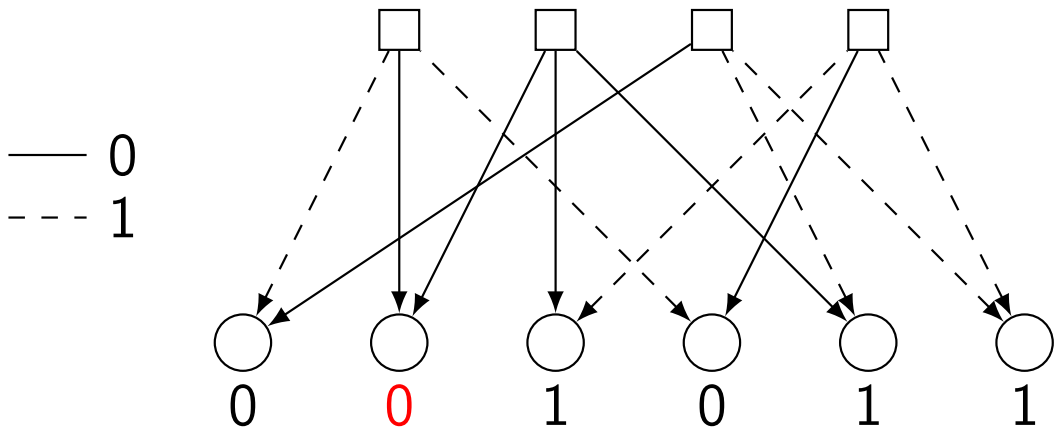
Décodage bit flipping

On reçoit le message $r = [011011]$ codé par un code possédant la matrice de parité régulière présentée précédemment.



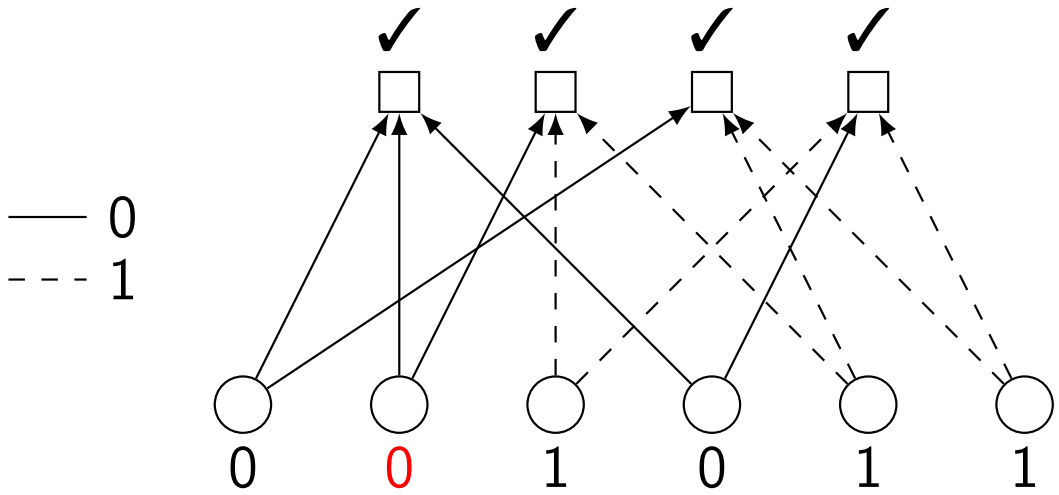
Décodage bit flipping

On reçoit le message $r = [011011]$ codé par un code possédant la matrice de parité régulière présentée précédemment.



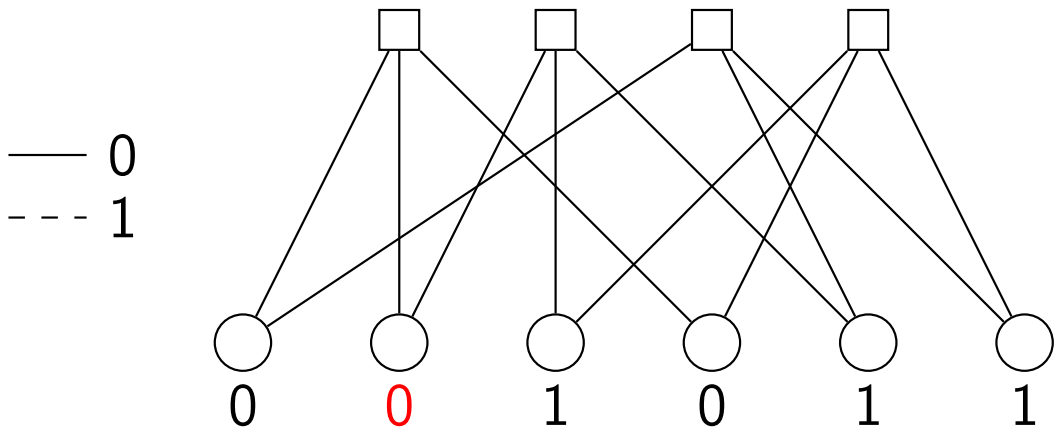
Décodage bit flipping

On reçoit le message $r = [011011]$ codé par un code possédant la matrice de parité régulière présentée précédemment.



Décodage bit flipping

On reçoit le message $r = [011011]$ codé par un code possédant la matrice de parité régulière présentée précédemment.



Le décodeur renvoie alors:
$\hat{c} = [001011]$

Décodage sum product

Le décodeur accepte en entrée le rapport de vraisemblance logarithmique (LLR), R_i provenant du canal (du démodulateur en fait):

$$R_i = \log \frac{p(x=0)}{p(x=1)} \tag{20}$$

Pour le BSC:

$$R_i = \begin{cases} \log \frac{1-f}{f} & \text{si } r_i = 0 \\ \log \frac{f}{1-f} & \text{si } r_i = 1 \end{cases}$$

Pour le canal AWGN:

$$\begin{aligned} R_i &= \frac{2}{\sigma^2} r_i \\ &= 4 \frac{\sqrt{rE_b}}{N_0} r_i \end{aligned}$$

Décodage sum product

L'initialisation est réalisée par $Mji = R_i$

Règle des nœuds de variable

$$M_{ji} = \sum_{j' \in A_i,\ j' \neq j} E_{j'i} + R_i \tag{21}$$

Règle des nœuds de contrôle

$$E_{ji} = \log \frac{1 + \prod_{i' \in B_j,\ i' \neq i} \tanh(M_{ji'}/2)}{1 - \prod_{i' \in B_j,\ i' \neq i} \tanh(M_{ji'}/2)}. \tag{22}$$

Arrêt lorsque les décisions hard de $L_i = \sum_{j \in A_i} E_{ji} + R_i$ vérifient toutes les équations de parité

Considérations pratiques

- Il existe une version où les messages représentent les probabilités (jamais utilisé à cause de l'instabilité).
- Messages quantifiés en virgule fixe sur un nombre de bits donnés
- Règle des nœuds de variable simple (additionneur)
- Simplification de la règle des nœuds de contrôle (min sum):

$$E_{ji} \approx \prod_{i'} \text{sign } M_{j,i'} \min_{i'} |Mj, i'| \tag{23}$$

- Algorithme hautement parallélisable

Encodage

L'encodage nécessite la connaissance de la matrice G du code.
Pour cela, il est nécessaire de mettre H sous la forme:

$$H_s = \begin{bmatrix} p_{1,1} & p_{1,2} & \cdots & p_{1,n-k} & 1 & 0 & \cdots & 0 \\ p_{2,1} & p_{2,2} & \cdots & p_{2,n-k} & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ p_{n-k,1} & p_{n-k,2} & \cdots & p_{n-k,n-k} & 0 & 0 & \cdots & 1 \end{bmatrix}$$

en effectuant des opérations élémentaires sur les lignes.
La matrice G est ensuite obtenue à partir de:

$$G = [I_{k\times k} P^T_{k\times n-k}] \tag{24}$$

Enfin, un mot de code peut alors être obtenu par:

$$c = s \; G = [s \; sP^T] \tag{25}$$

Performances des codes LDPC