Codificação de Canal: Códigos corretores de erros. Códigos de bloco. Códigos convolucionais.

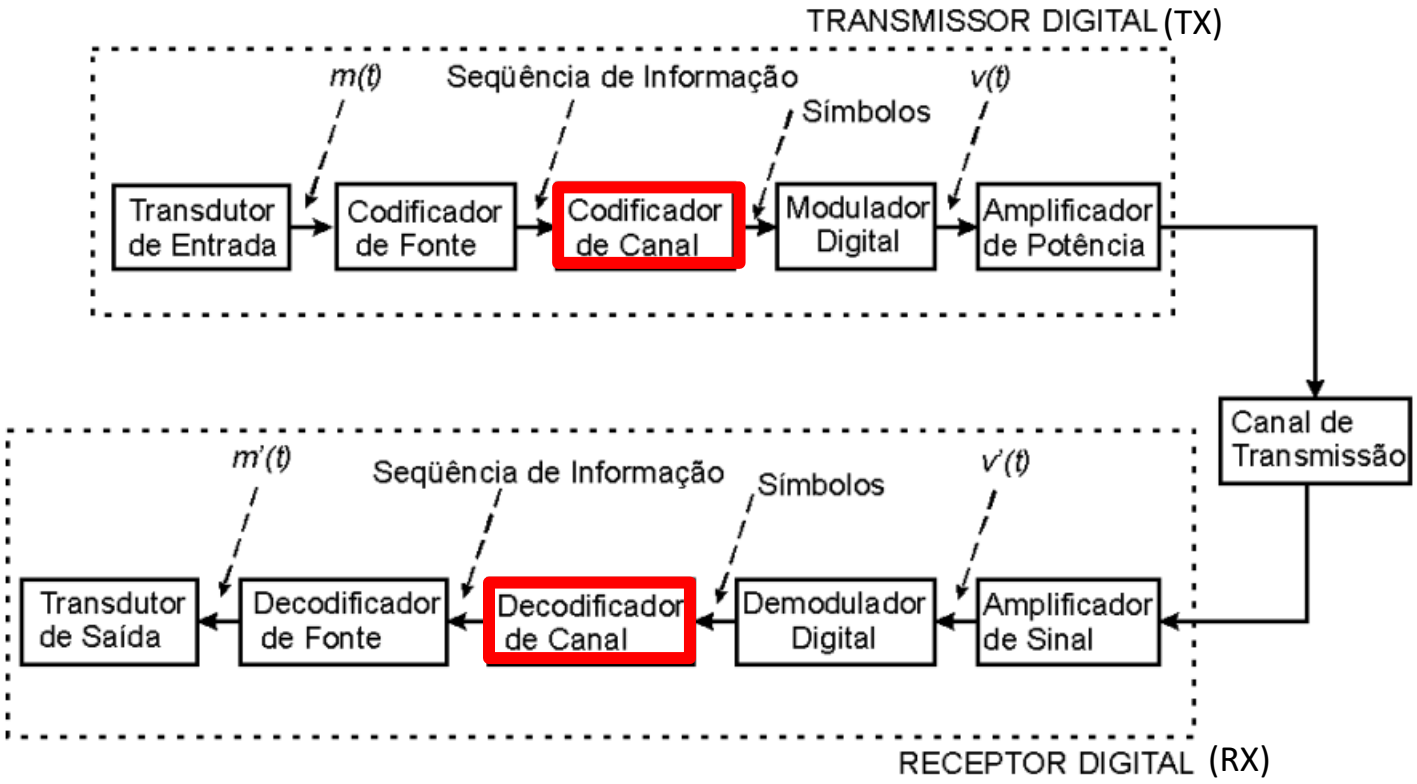
Centro de Tecnologia – Departamento de Eletrônica e Computação

UFSM00261 – SISTEMAS DE COMUNICAÇÃO DIGITAL I

Prof. Fernando DeCastro

Codificação de Canal

Conforme já brevemente discutido nos slides 3, 8 e 9 do Cap I , o Codificador de Canal (*channel encoder*) marca c/ bits de paridade as palavras binárias recebidas do Codificador de Fonte de modo que o Decodificador de Canal (*channel decoder*) corrija os erros em bits causados pelo ruído branco aditivo originado no canal e no *front end* de RF do RX.



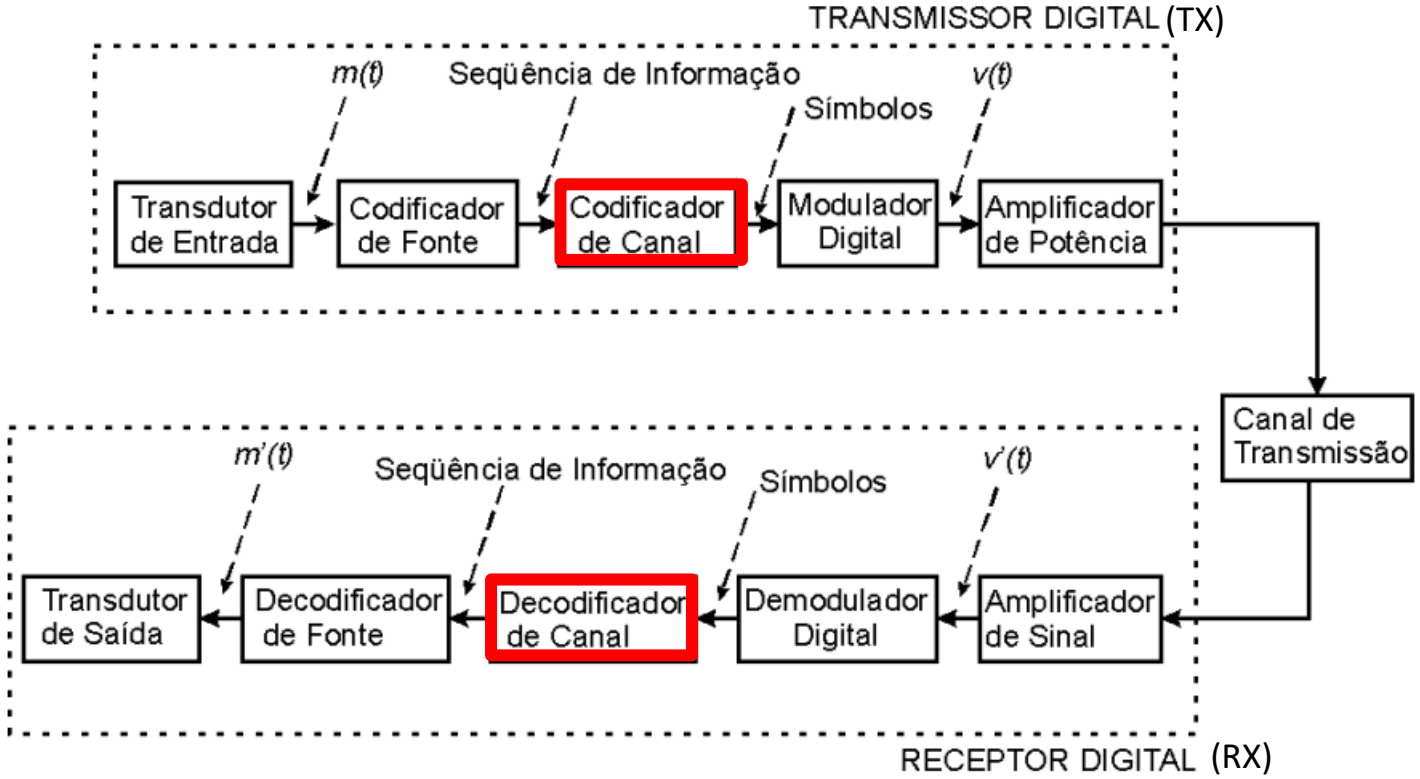
Portanto, a Codificação de Canal é o processo responsável em um sistema digital por manter a BER (*bit error rate*) dentro de um limite máximo considerado aceitável no âmbito das especificações de desempenho de um determinado sistema digital.

Codificação de Canal

Quando informação digital é enviada pelo TX através do canal de transmissão, ruído e interferência inerentes a qualquer canal prático degradam o sinal de forma que as palavras binárias recebidas no RX contêm erros.

O usuário do sistema digital geralmente estabelece uma taxa de erro máxima aceitável – por exemplo, um bit errado em 1×10^6 bits recebidos (i.e., $BER = 1 \times 10^{-6}$) – acima da qual os dados recebidos não são considerados utilizáveis pelo usuário. Esta taxa de erro máxima aceitável depende da informação que transita pelo canal.

A título de comparação, a BER máxima permitida para transmissão de voz através de telefonia celular é muito maior do que a BER exigida para transmissão de dados. Isto ocorre porque, na pior das hipóteses, mesmo sob uma alta BER e consequente distorção, o sistema auditivo humano é capaz de compreender o significado das frases pelo contexto da conversa, o que já não acontece quando dois computadores trocam dados binários.



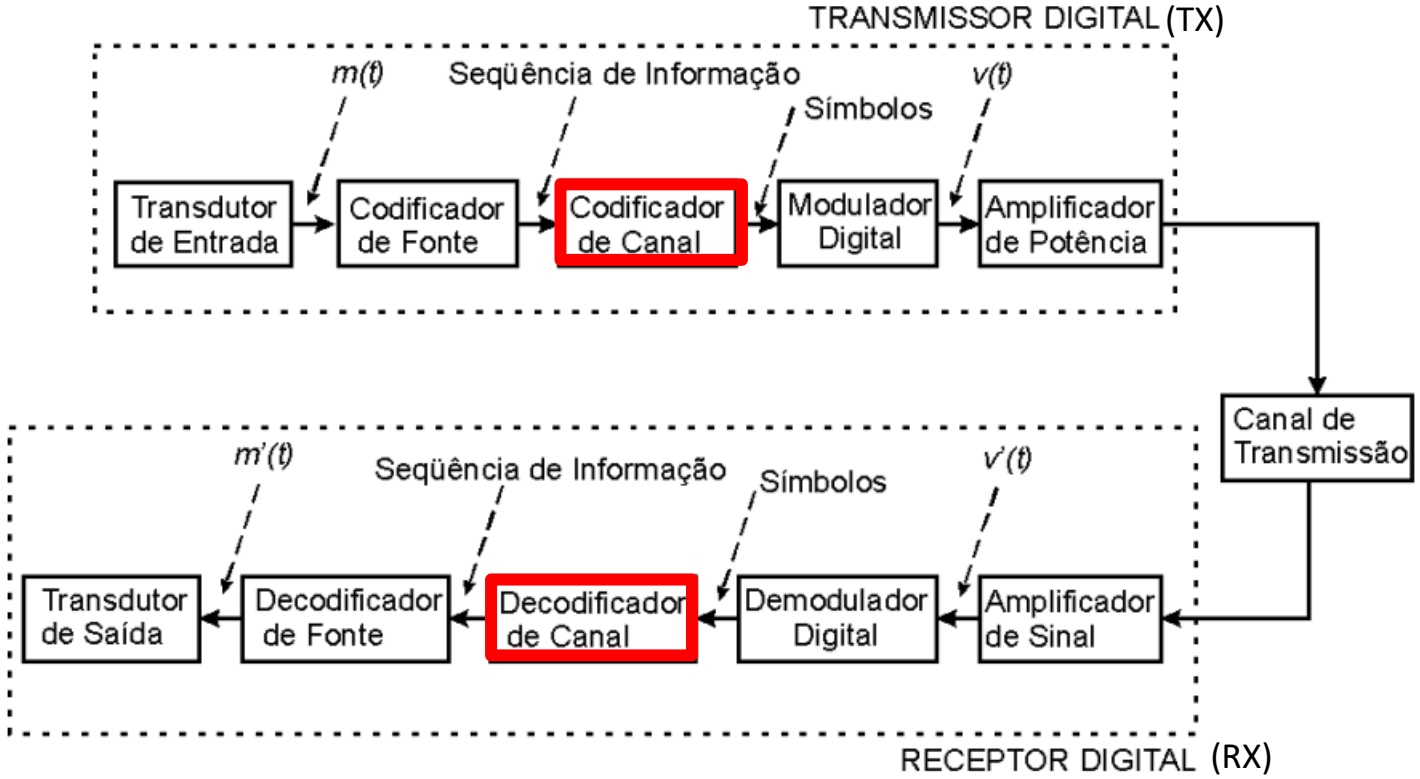
Codificação de Canal

O Codificador de Canal é, portanto, o responsável em um sistema digital por manter a BER dentro de um limite máximo aceitável.

A possibilidade do uso de codificação para controlar com eficiência a BER de um sistema de comunicação digital foi demonstrada por Shannon em 1948, através do Teorema Fundamental de Shannon, já discutido nos slides 34 a 46 do Cap I das notas de aula:

Teorema Fundamental de Shannon:

Se a taxa (= velocidade) de transmissão R [bits/s] da informação a ser enviada pelo canal é menor que um valor C [bits/s] denominada de Capacidade do Canal, então a comunicação através do canal pode ser estabelecida com probabilidade de erro tão baixa quanto se deseje, através do uso de um código adequado para correção de erro.



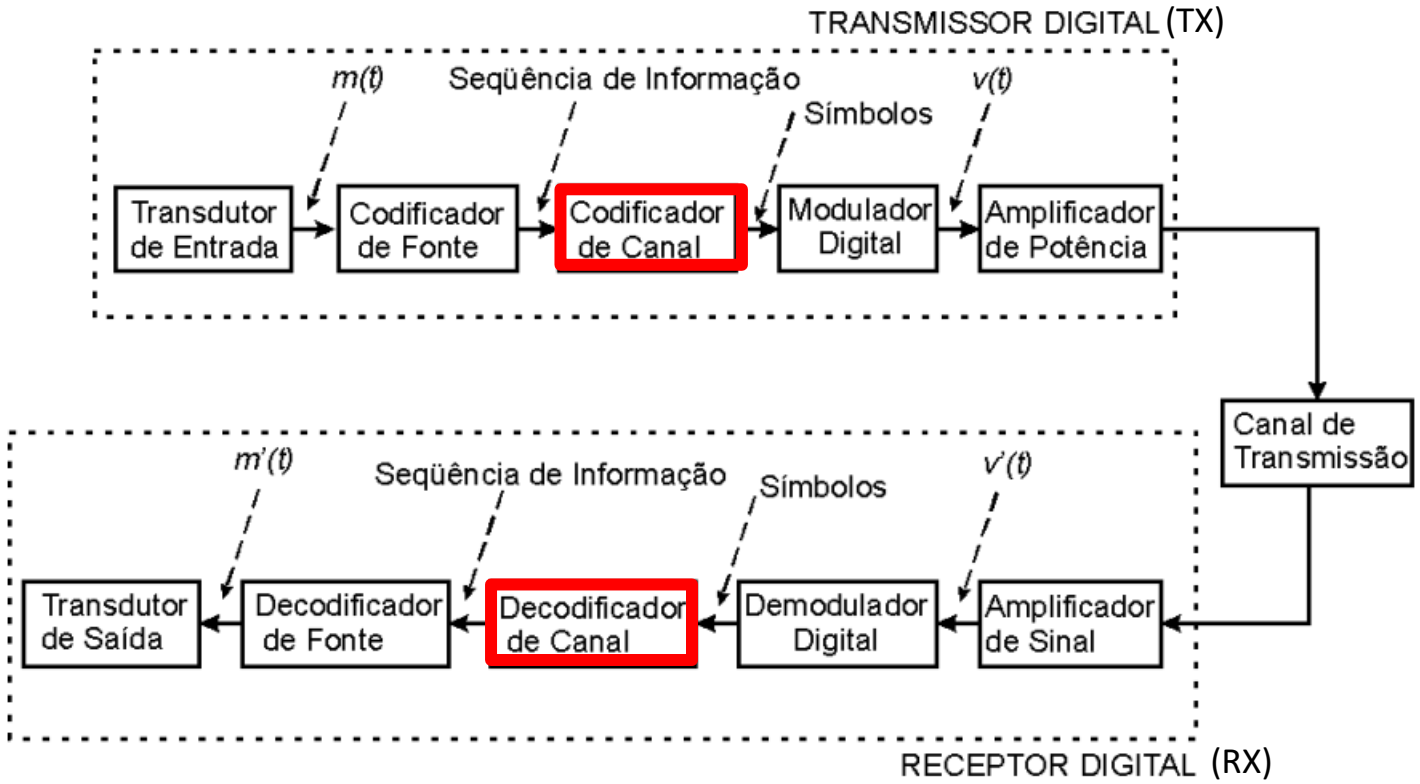
Códigos corretores de erro

Desta maneira, o Teorema Fundamental de Shannon estabelece a existência de um código corretor de erro tal que a informação pode ser enviada pelo TX e transmitida através do canal de transmissão com uma BER arbitrariamente baixa resultante no RX, desde que a taxa de transmissão R [bits/s] seja menor ou igual à capacidade do canal C [bits/s].

Neste estudo daremos enfoque às técnicas de correção de erro mais importantes no âmbito de duas grandes classes de códigos para correção de erro:

(1) códigos de bloco (ver https://en.wikipedia.org/wiki/Block_code).

(2) códigos convolucionais (ver https://en.wikipedia.org/wiki/Convolutional_code)



Códigos de Bloco

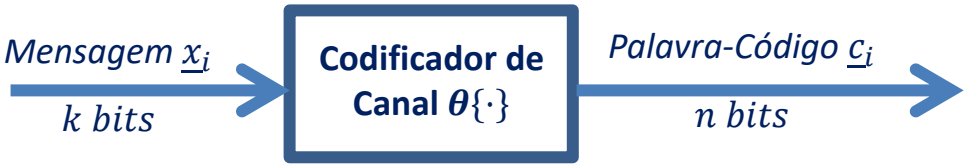
De maneira similar aos códigos para compressão por entropia discutidos no Cap II, um código de bloco pode ser considerado como um operador $\boldsymbol{\theta}\{\cdot\}$, tal que $\boldsymbol{C} = \boldsymbol{\theta}\{\boldsymbol{X}\}$, onde:

$\boldsymbol{X} = \{\underline{x}_i\} = \{\underline{x}_0, \underline{x}_1, \dots, \underline{x}_{M-1}\}$ é o conjunto de M possíveis **mensagens** $\underline{x}_i$ a serem codificadas e

$\boldsymbol{C} = = \{\underline{c}_i\} = \{\underline{c}_0, \underline{c}_1, \dots, \underline{c}_{M-1}\}$ é o conjunto de M possíveis **palavras-código** $\underline{c}_i$ resultantes da codificação, com $i = 0,1, \dots, M-1$.

O operador $\boldsymbol{\theta}\{\cdot\}$ é definido por um ***codebook*** que efetua um mapeamento unívoco entre cada mensagem $\underline{x}_i$ e a respectiva palavra-código $\underline{c}_i$.

O **conjunto de caracteres do código** ou **alfabeto do código** é o conjunto $\mathbf{A} = \{a_0, a_1, \dots, a_{D-1}\}$ composto por D elementos, de cuja composição são formadas cada mensagem e sua respectiva palavra-código (para códigos binários $\mathbf{A} = \{0,1\}$).



Cada mensagem $\underline{x}_i \in \boldsymbol{X}$ é considereda como sendo um vetor $\underline{x}_i = [x_{i(k-1)} x_{i(k-2)} \dots x_{i1} x_{i0}]$ de k componentes, $x_{ij} \in \boldsymbol{A}$, sendo $j = k-1, k-2, \dots 1,0$.

Visto que os k componentes da i-ésima mensagem $\underline{x}_i$ pertencem ao alfabeto $\boldsymbol{A}$, é válida a relação de pertinência $\underline{x}_i \in \boldsymbol{A}^k$.

Da mesma forma, cada palavra-código $\underline{c}_i \in \boldsymbol{C}$ é um vetor $\underline{c}_i = [c_{i(n-1)} c_{i(n-2)} \dots c_{i1} c_{i0}]$ de n componentes $c_{ij} \in \boldsymbol{A}$, sendo $j = n-1, n-2, \dots 1,0$.

Visto que os n componentes da i-ésima palavra-código $\underline{c}_i$ pertencem ao alfabeto $\boldsymbol{A}$, vale a relação de pertinência $\underline{c}_i \in \boldsymbol{A}^n$.

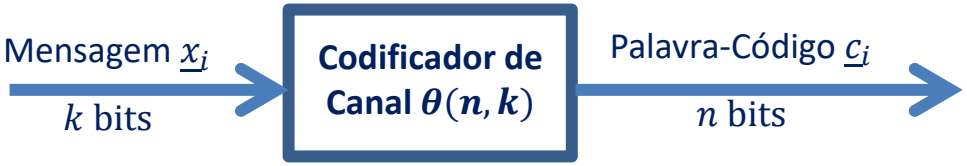
Por exemplo: a i-ésima palavra-código binária 0101, de $n = 4$ bits, é representada pelo vetor $\underline{c}_i$= [0 1 0 1], onde $\underline{c}_i \in \boldsymbol{A}^4$ sendo o alfabeto definido por $\mathbf{A} = \{0,1\}$.

c_{i3} c_{i0}

Códigos de Bloco binários

Um código de bloco binário $\boldsymbol{\theta}\{\cdot\}$ mapeia um conjunto $\boldsymbol{X} = \{\underline{x}_i\} = \{\underline{x}_0, \underline{x}_1, \ldots, \underline{x}_{M-1}\}$ de $M = 2^k$ mensagens binárias, cada uma delas com k bits, em um conjunto $\boldsymbol{C} = \{\underline{c}_i\} = \{\underline{c}_0, \underline{c}_1, \ldots, \underline{c}_{M-1}\}$ palavras-código binárias, cada uma delas com n bits, onde $n > k$. O mapeamento é definido por um *codebook* que implementa a operação $\underline{c}_i = \boldsymbol{\theta}\{\underline{x}_i\}$.

Um *codebook* que implementa o mapeamento $\underline{c}_i = \boldsymbol{\theta}\{\underline{x}_i\}$ de um código binário, cujas mensagens $\underline{x}_i$ de k bits são codificadas em palavras-código $\underline{c}_i$ de n bits, é alternativamente representado pelo operador $\boldsymbol{\theta}(n,k)\{\cdot\}$ ou simplesmente $\boldsymbol{\theta}(n,k)$.



Um código $\boldsymbol{\theta}(n,k)$ é **sistemático** quando cada palavra-código de n bits é formada pelos k bits da respectiva mensagem associada, acrescidos (por justaposição) de $r = n - k$ bits adicionais destinados ao controle e correção de erros, denominados de **bits de paridade**.

Portanto, em um código sistemático cada mensagem contendo k bits de informação é expandida em uma palavra-código de $n = k + r$ bits onde $\boldsymbol{r}$ **é o número de bits representativos da informação redundante adicionada visando o controle e correção de erro.**

Um código $\boldsymbol{\theta}(n,k)$ é **não-sistemático** quando nas palavras-códigos de n bits não aparecem explicitamente representados os k bits de informação da respectiva mensagem associada.

É possível converter um código não-sistemático em um código sistemático. Em função disto, nossa atenção será dada aos códigos sistemáticos.

Tanto o código não sistemático, quanto o código convertido em um código sistemático, possuem a mesma capacidade de correção e de detecção, por isso são ditos códigos equivalentes. **Note, no entanto, que a implementação em hardware de um código sistemático é mais simples do que a de um código não-sistemático.**

Códigos de Bloco binários

Por exemplo, o código $\boldsymbol{\theta}(4,3)$ definido no *codebook* abaixo é **sistemático**, porque cada palavra-código $\underline{c_i}$ de $n = 4$ bits é formada pela justaposição de 1 bit de paridade aos $k = 3$ bits de informação da mensagem $\underline{x_i}$ associada.

Observe que, como $n > k$, no conjunto de todas as 2^npossíveis palavras-códigos de n bits **existem $\boldsymbol{2^n - 2^k}$ elementos que não são associados a qualquer elemento do conjunto $\boldsymbol{X} = \{\underline{x_i}\} = \{\underline{x}_0, \underline{x}_1, \ldots, \underline{x}_{M-1}\}$** de $M = 2^k$ mensagens binárias de k bits.

Por exemplo, para o código binário $\boldsymbol{\theta}(4,3)$ no *codebook* abaixo, existem $2^n - 2^k = 2^4 - 2^3 = 8$ elementos no conjunto de todas as $2^n = 2^4 = 16$ possíveis palavras-códigos de 4 bits sem associação com qualquer mensagem do conjunto $\boldsymbol{X} = \{000, 001, 010, 011, 100, 101, 110, 111\}$.

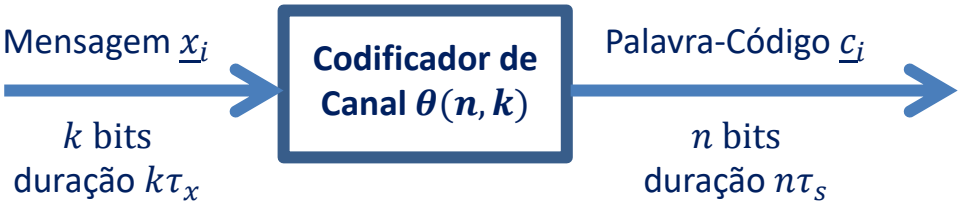
Mensagem $\underline{x_i}$	Palavra-código $\underline{c_i}$
000	0000
001	0011
010	0101
011	0110
100	1001
101	1010
110	1100
111	1111

Razão de codificação

O tempo $n\tau_s$ de duração de cada palavra-código $\underline{c_i}$ deve ser igual ao tempo de duração $k\tau_x$ de cada mensagem $\underline{x_i}$, i.e., $n\tau_s = k\tau_x$ (vide figura abaixo), onde τ_s representa a duração de cada bit em uma palavra-código $\underline{c_i}$ e τ_x representa a duração de cada bit em uma mensagem $\underline{x_i}$.

Se a condição $n\tau_s = k\tau_x$ não é obedecida, as palavras códigos $\underline{c_i}$ resultarão com um tempo de duração $n\tau_s$ diferente do que o tempo de duração $k\tau_x$ das respectivas mensagens $\underline{x_i}$ que a elas deram origem, resultando que o espectro da informação codificada será alterado (conforme brevemente discutido no slide 8 do Cap I). Esta condição implica na inviabilidade de se adotar códigos com uma enorme quantidade $r = n - k$ de bits de paridade, porque, ao aumentar n excessivamente resultará em uma muito pequena duração $\tau_s = \frac{k}{n}\tau_x$ em cada bit das palavras-código $\underline{c_i}$, aumentando a BW final do espectro do sinal transmitido através do canal de transmissão (conforme já discutido no slide 5 do Cap I). Note que aumentar o número $r = n - k$ de bits de paridade em um código aumenta a distância de Hamming entre as palavras-código do *codebook*, o que aumenta a robustez do código – o que é desejável (ver discussão no slide 9 do Cap I). No entanto, a condição $n\tau_s = k\tau_x$ impõe um limite prático para o aumento do número $r = n - k$ de bits de paridade em função do indesejável aumento da BW final do espectro do sinal transmitido através do canal de transmissão

Dado que a condição $n\tau_s = k\tau_x$ precisa ser obedecida para efeito de não alterar o espectro da informação codificada, então a **razão de codificação $\boldsymbol{R_c}$ de um código de bloco** é definida por $\boldsymbol{R_c = k/n = \tau_s/\tau_x}$, **sendo $\boldsymbol{n > k}$**.



Peso de uma Palavra-Código

O **peso** de uma palavra-código $\underline{c_i}$ é definido como o número de dígitos "1" nela presentes.

O conjunto de todos os pesos de um código constitui a **distribuição de pesos** do código.

Por exemplo, o peso da palavra-código $\underline{c_i} = [0 \quad 1 \quad 0 \quad 1]$ é 2.

Quando todas as M palavras-código $\underline{c_i}$ têm pesos iguais, o código é denominado de **código de peso constante**.

Códigos de Bloco – representação polinomial

O processo de codificação/decodificação de um código de bloco baseia-se na propriedade algébrica de que o conjunto de **palavras-código** $\boldsymbol{C} = \{\underline{c_i}\} = \{\underline{c}_0, \underline{c}_1, \dots, \underline{c}_{M-1}\}$ pode ser mapeado em um **conjunto de polinômios** $\{C_i(p)\} = \{C_0(p), C_1(p), \dots C_{M-1}(p)\}$.

Os componentes do **vetor** $\underline{c_i} = [c_{i(n-1)}\ \ c_{i(n-2)}\ \ \dots\ \ c_{i1}\ \ c_{i0}]$ que representa a i-ésima **palavra-código** $\underline{c_i}$ correspondem aos coeficientes do polinômio $C_i(p) = c_{i(n-1)}p^{n-1} + c_{i(n-2)}p^{n-2} + \dots + c_{i1}p + c_{i0}$ associado à palavra-código $\underline{c_i}$.

A mesma propriedade algébrica pode ser aplicada sobre o conjunto de **mensagens** $\boldsymbol{X} = \{\underline{x_i}\} = \{\underline{x}_0, \underline{x}_1, \dots, \underline{x}_{M-1}\}$ de modo que este também pode ser mapeado em um conjunto de polinômios $\{X_i(p)\} = \{X_0(p), X_1(p) \cdots X_{M-1}(p)\}$.

Os componentes do **vetor** $\underline{x_i} = [x_{i(k-1)}x_{i(k-2)} \dots x_{i1}x_{i0}]$ que representa a i-ésima **mensagem** $\underline{x_i}$ correspondem aos coeficientes do polinômio $X_i(p) = x_{i(k-1)}p^{n-1} + x_{i(k-2)}p^{n-2} + \dots + x_{i1}p + x_{i0}$ associado à mensagem $\underline{x_i}$.

Por este motivo os códigos de bloco são também denominados de **<u>códigos polinomiais</u>**.

Por exemplo, a representação polinomial do código sistemático $\boldsymbol{\theta}(4,3)$ definido no *codebook* do slide 8 é mostrada na tabela abaixo:

Mensagem $\underline{x_i}$	Polinômio $X_i(p)$	Palavra-código $\underline{c_i}$	Polinômio $C_i(p)$
000	0	0000	0
001	1	0011	$p+1$
010	p	0101	p^2+1
011	$p+1$	0110	p^2+p
100	p^2	1001	p^3+1
101	p^2+1	1010	p^3+p
110	p^2+p	1100	p^3+p^2
111	p^2+p+1	1111	p^3+p^2+p+1

Códigos de Bloco – representação polinomial

O processo de codificação/decodificação envolve operações aritméticas de adição e multiplicação binárias realizadas sobre o conjunto de polinômios $\{C_i(p)\} = \{C_0(p), C_1(p) \cdots C_{M-1}(p)\}$ que representam as palavras-código, conforme veremos nos próximos slides.

Um código corretor de erro deve ser tal que o conjunto de polinômios $\{C_i(p)\}$ e as operações aritméticas binárias sobre ele definidas obedeçam a determinado regramento algébrico, caso contrário a unicidade e o custo computacional no hardware do processo de codificação/decodificação resultarão prejudicados.

Este conjunto de polinômios e o respectivo regramento algébrico a ele associado é denominado de **campo algébrico**.

Um campo algébrico, conforme veremos nos próximos slides, é uma entidade matemática estudada em Álgebra Linear.

Campo Algébrico

Um campo algébrico **F** é um conjunto de elementos que permite duas operações sobre seus elementos – adição e multiplicação – operações que satisfazem às seguintes propriedades:

Adição

1- O conjunto **F** é fechado sob a adição, i.e., se a,b ∈ **F** então (a + b) ∈ **F** .

2- A adição em **F** é associativa, i.e., se a,b,c ∈ **F** então a + (b + c) = (a + b)+ c.

3- A adição em **F** é comutativa, i.e., se a,b ∈ **F** então a + b = b + a .

4- O conjunto **F** contém um único elemento denominado **zero**, representado por "0", que satisfaz a condição a + 0 = a, ∀ a ∈ **F**.

5- Cada elemento em **F** tem o seu **elemento negativo (simétrico)**. Se b ∈ **F** então seu simétrico é denotado por − b tal que b + (− b) = 0 . Se a ∈ **F** , então a subtração a − b entre os elementos a e b é definida como a + (− b).

Multiplicação

1- O conjunto **F** é fechado sob a multiplicação, i.e., se *a*,*b* ∈ **F** então *ab* ∈ **F**.

2- A multiplicação em **F** é associativa, i.e., se **F** ∈ *c b a* , então $a(bc)$=(ab)c

3- A multiplicação em **F** é comutativa, i.e., se *a*,*b* ∈ **F** então *ab* = *ba*.

4- A multiplicação em **F** é distributiva sobre a adição, i.e., se *a*,*b*,*c* ∈ **F** então *a*(*b* + *c*) = *ab* + *ac*.

5- O conjunto **F** contém um único elemento denominado **identidade**, representado por "1", que satisfaz a condição 1*a* = *a*, ∀ *a* ∈ **F**.

6- Cada elemento de **F**, exceto o elemento 0 , possui um elemento **inverso**. Assim, se *b* ∈ **F** e *b* ≠ 0 então o inverso de *b* é definido como b^{-1} tal que bb^{-1}= 1. Se *a* ∈ **F** , então a divisão *a* / *b* entre os elementos *a* e *b* é definida como ab^{-1}.

Campo de Galois GF(2)

Por exemplo, o conjunto $\mathbb{R}$ dos números reais é um campo algébrico com infinitos elementos, assim como também o é o conjunto dos números complexos $\mathbb{C}$. Estes dois conjuntos obedecem às propriedades dos campos algébricos descritas anteriormente.

Um campo algébrico finito com D elementos é denominado de Campo de Galois (*Galois Field*) e é designado por **GF**(D).

Nem para todos os valores de D é possível formar um campo.

Em geral, quando D é primo (ou uma potência inteira de um número primo) é possível construir o campo finito **GF**(D) consistindo dos elementos $\{0,1 \cdots D-1\}$, desde que as operações de adição e multiplicação sobre **GF**(D) sejam operações **módulo *D*** .

Nota: Uma operação **op é módulo *D*** quando pode ser representada por $(a \ \mathbf{op} \ b) \bmod D$, onde $x \bmod y$ é o operador que resulta no resto da divisão x/y.

Por exemplo, a operação de **soma módulo 5** entre os números **4** e **3**, $(4 \ \mathbf{op} \ 3) \bmod 5$, resulta em **2** visto que o resto da divisão 7/5 é 2, portanto $(4 + 3) \bmod 5 = 2$.

Dado que o hardware do codificador e do decodificador de canal executa operações binárias, o enfoque em nosso estudo será o **Campo de Galois para $D = 2$, denominado GF(2).**

O GF(2) é formado pelo conjunto {0,1} e pelas operações módulo 2 de soma "$\oplus$" e multiplicação "$\cdot$" definidas nas tabelas (I) e (II):

Note nas tabelas (I) e (II) que:

(I) A soma entre dois elementos a e b pertencentes a GF(2) é implementada pela operação lógica $a \oplus b$ (ou a XOR b).

(II) A multiplicação entre dois elementos a e b pertencentes a GF(2) é implementada pela operação lógica $a \cdot b$ (ou a AND b).

Tabela (I)
Soma sobre GF(2)

$\oplus$	0	1
0	0	1
1	1	0

Tabela (II)
Multiplicação sobre GF(2)

.	0	1
0	0	0
1	0	1

Campo de Galois GF(2)

Dada a facilidade de implementação em hardware das operações de soma e multiplicação com portas lógicas AND e XOR, é usual os códigos corretores serem construídos em GF(2).

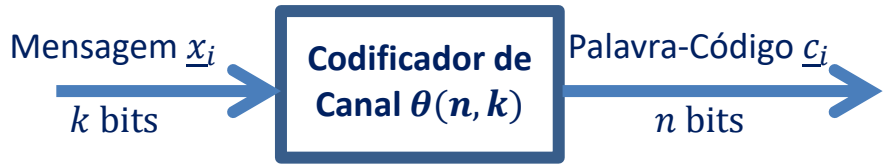
Assim, um código corretor de erro binário com alfabeto $\mathrm{A} = \{0,1\}$ é tal que os coeficientes dos polinômios em $\{C_i(p)\}$ pertencem a GF(2).

No hardware do codificador do TX e do decodificador do RX as operações aritméticas realizadas sobre o conjunto de polinômios$\{C_i(p)\} = \{C_0(p), C_1(p) \cdots C_{M-1}(p)\}$ (ou, equivalentemente, sobre o conjunto de palavras-código $C = \{\underline{c}_i\} = \{\underline{c}_0, \underline{c}_1 \cdots \underline{c}_{M-1}\}$) durante o processo de codificação/decodificação obedecem às tabelas (I) e (II) do slide anterior.

Capacidade de detecção e correção de erro de um código binário

Suponhamos que $\underline{c_i}$ e $\underline{c_j}$ sejam duas palavras-código quaisquer no *codebook* de um código $\boldsymbol{\theta}(n,k)$. Uma medida da dessemelhança entre duas palavras-código $\underline{c_i}$ e $\underline{c_j}$ é o número de bits em posições correspondentes que diferem entre si. Esta medida é denominada de **Distância de Hamming** e é denotada por $d\mathrm{H} = d_{ij}$.

Por exemplo, sejam $\boldsymbol{\underline{c_i}}$ **= [0 1 0 1] e** $\boldsymbol{\underline{c_j}}$ **= [1 0 0 0]**. Então a distância de Hamming é $\boldsymbol{d\mathbf{H} = d_{ij}}$ **= 3.** Observe que d_{ij} sempre satisfaz a condição $0 < d_{ij} \leq n, i \neq j$, para duas palavras-código $\underline{c_i}$ e $\underline{c_j}$, ambas de n bits (por definição, em um código $\boldsymbol{\theta}(n,k)$, $\underline{c_i} \neq \underline{c_j}$ $\forall i$ e $\forall j$ com $i \neq j$). **O menor valor no conjunto** $\boldsymbol{\{d_{ij}\}}$ **é a distância mínima do código, denotada como** $\boldsymbol{d_{min}}$, sendo $i,j = 0,1 \cdots M-1,\ i \neq j$ e $M = 2^k$.



Por exemplo, $d_{min} = 2$ para o código do código sistemático $\boldsymbol{\theta}(n=4,k=3)$ definido no *codebook* do slide 8, cujas $M = 2^k = 8$ palavras código no *codebook* são $\boldsymbol{C} = \{\underline{c_i}\} = \{\underline{c_0}, \underline{c_1} \cdots \underline{c_{M-1}}\}$ = {0000, 0011, 0101, 0110, 1001, 1010, 1100, 1111}.

A Distância de Hamming $\boldsymbol{d_{ij}}$ **é uma medida do grau de separação (dissimilaridade) entre duas palavras-código** $\boldsymbol{\underline{c_i}}$ **e** $\boldsymbol{\underline{c_j}}$ no *codebook* $\boldsymbol{C}$, conforme já discutimos no slide 9 do Cap I. Desta maneira d_{min} define a capacidade do decodificador do código $\boldsymbol{\theta}(n,k)$ no RX em detectar e corrigir palavras-código que são recebidas em erro como consequência do ruído aditivo presente no canal de transmissão. **Note, portanto, que quanto maior** $\boldsymbol{d_{min}}$ **maior a capacidade de um código** $\boldsymbol{\theta}(\boldsymbol{n},\boldsymbol{k})$ **detectar e corrigir bits recebidos em erro.**

Uma técnica de decodificação ótima é a **decodificação por síndrome**, que veremos adiante Uma técnica alternativa mas de maior custo computacional é o ***Maximum-Likelihood Decoding*** **(MLD)**, conforme já discutido no slide 8 do Cap I, e que também veremos adiante. No MLD, para corrigir os bits errados em uma palavra-código $\underline{\widetilde{c}_\mathrm{r}}$ recebida no decodificador do RX, o decodificador MLD determina as M respectivas distâncias de Hamming $d\mathrm{H}_i$ entre $\underline{\widetilde{c}_\mathrm{r}}$ recebido e as *M* palavras código $\underline{c_i}$ do *codebook,* com $i = 0,1 \cdots M-1$, resultando no conjunto $\boldsymbol{d\mathbf{H}}$ ={dH_0, $dH_1 \cdots dH_{M-1}$}. O decodificador MLD identifica então qual a menor $d\mathrm{H}_i$ no conjunto $\boldsymbol{d\mathbf{H}}$ e infere que a palavra-código $\underline{c_i}$ que foi transmitida pelo TX é aquela que corresponde à menor $d\mathrm{H}_i$ no conjunto $\boldsymbol{d\mathbf{H}}$**.**

Capacidade de detecção e correção de erro de um código binário

Note portanto que, se dessemelhança mínima entre as palavras-código do *codebook* dada pelo valor de d_{min} é insuficiente em relação ao número t de bits recebidos em erro, então o decodificador se torna incapaz de corrigir os bits errados, ou se torna incapaz até mesmo de detectar os bits errados.

Para um código corretor binário $\boldsymbol{\theta}(n,k)$ que é capaz de **detectar** até d bits errados em cada palavra-código $\underline{c_i}$ de n bits recebida no decodificador do RX e que é capaz de **corrigir** até t bits errados em cada palavra-código $\underline{c_i}$ *de* n bits recebida no decodificador do RX , é possível demonstrar que d e t são dados por:

$$d = d_{min} - 1 \qquad (1) \qquad\qquad t = \left\lfloor \frac{d_{min} - 1}{2} \right\rfloor \qquad (2)$$

sendo $\lfloor . \rfloor$ o operador que resulta no inteiro mais próximo e menor que o argumento.

Novamente, note de (1) e (2) que quanto maior a d_{min} resultante do conjunto de palavras códigos no *codebook* de um código $\boldsymbol{\theta}(n,k)$ maior será o número de bits recebidos em erro que o código é capaz detectar e corrigir.

Por exemplo, para o código sistemático $\boldsymbol{\theta}(n=4,k=3)$ definido no *codebook* do slide 8 e abaixo reproduzido temos que $d_{min} = 2,$ de modo que de (1) e (2) obtemos:

$$d = d_{min} - 1 = 2 - 1 = 1 \qquad t = \left\lfloor \frac{d_{min}-1}{2} \right\rfloor = \left\lfloor \frac{2-1}{2} \right\rfloor = 0$$

Portanto o código sistemático $\boldsymbol{\theta}(n=4,k=3)$ com $d_{min} = 2$ dado no *codebook* ao lado é capaz de detectar no máximo $d = d_{min} - 1 = 1$ um bit errado por palavra-código recebida no decodificador do RX mas é incapaz de corrigir qualquer bit errado em uma palavra-código recebida visto que $t = \left\lfloor \frac{d_{min}-1}{2} \right\rfloor = 0.$

De fato, o *codebook* ao lado define um simples código *parity-check.* Um código *parity-check* apenas verifica através do bit de paridade se ocorreu algum bit errado, mas não é capaz de corrigir o erro. Especificamente este código sistemático efetua o XOR entre os bits na palavra-código $\underline{c_i}$ correspondentes ao da mensagem $\underline{x_i}$ e compara com o bit de paridade em $\underline{c_i}$.

Mensagem $\underline{x_i}$	Palavra-código $\underline{c_i}$
000	0000
001	0011
010	0101
011	0110
100	1001
101	1010
110	1100
111	1111

A Matriz Geradora de um Código $\theta(n,k)$

Consideremos a i-ésima mensagem do *codebook* de um código binário $\boldsymbol{\theta}(n,k)$ representada pelo vetor $\underline{x_i} = [x_{i0}\ x_{i1}\ \dots x_{i(k-1)}]$ e seja a i-ésima palavra-código do *codebook* representada pelo vetor $\underline{c_i} = [c_{i0}\ c_{i1}\ \dots c_{i(n-1)}]$, onde $i = 0, 1, \dots, M-1, M = 2^k$.

O processo de codificação da mensagem $\underline{x_i} = [x_{i0}\ x_{i1}\ \dots x_{i(k-1)}]$ na respectiva palavra-código $\underline{c_i} = [c_{i0}\ c_{i1}\ \dots c_{i(n-1)}]$ efetuado pelo *codebook* de um código binário $\boldsymbol{\theta}(n,k)$ é representado em forma matricial através de:

$$\underline{c_i} = \underline{x_i}\mathbf{G} \tag{3}$$

onde a matriz $\mathbf{G}_{k\times n}$ é denominada de **matriz geradora** do código $\boldsymbol{\theta}(n,k)$ sendo dada por:

$$\mathbf{G} = \begin{bmatrix} g_{00} & g_{01} & \cdots & g_{0(n-1)} \\ g_{10} & g_{11} & \cdots & g_{1(n-1)} \\ \vdots & \vdots & & \vdots \\ g_{(k-1)0} & g_{(k-1)1} & \cdots & g_{(k-1)(n-1)} \end{bmatrix} \tag{4}$$

A Matriz Geradora de um Código $\theta(n,k)$

Podemos interpretar a matriz $\mathbf{G}$ como um conjunto de k vetores-linha $\underline{g_j}$, $j = 0, 1, \ldots, k-1$, tal que

$$\mathbf{G} = \begin{bmatrix} g_{00} & g_{01} & \cdots & g_{0(n-1)} \\ g_{10} & g_{11} & \cdots & g_{1(n-1)} \\ \vdots & \vdots & & \vdots \\ g_{(k-1)0} & g_{(k-1)1} & \cdots & g_{(k-1)(n-1)} \end{bmatrix} = \begin{bmatrix} \leftarrow & \underline{g_0} & \rightarrow \\ \leftarrow & \underline{g_1} & \rightarrow \\ & \vdots & \\ \leftarrow & \underline{g_{(k-1)}} & \rightarrow \end{bmatrix} \tag{5}$$

Desta maneira, cada palavra-código $\underline{c_i} = [c_{i0}\ c_{i1} \ldots c_{i(n-1)}]$ obtida através da equação (3) $\underline{c_i} = \underline{x_i}\mathbf{G}$ é simplesmente uma **combinação linear dos vetores $\underline{g_j}$** , sendo os coeficientes da combinação linear dados pelos componentes da mensagem associada $\underline{x_i} = [x_{i0}\ x_{i1} \ldots x_{i(k-1)}]$, isto é:

$$\underline{c_i} = \underline{x_i}\mathbf{G} = x_{i0}\,\underline{g_0} + x_{i1}\,\underline{g_1} \cdots \underline{g_{(k-1)}} \tag{6}$$

A Matriz Geradora de um Código $\theta(n,k)$

Exemplo 1: Consideremos o código sistemático $\boldsymbol{\theta}(n=4,k=3)$ definido no *codebook* do slide 8 e reproduzido ao lado. Verifique se a matriz $\mathbf{G}$ abaixo é a matriz geradora deste código $\boldsymbol{\theta}(4,3)$.

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

Mensagem $\underline{x}_i$	Palavra-código $\underline{c}_i$
000	0000
001	0011
010	0101
011	0110
100	1001
101	1010
110	1100
111	1111

Solução: Cada palavra-código $\underline{c}_i=[c_{i0}\ c_{i1}\ \ldots c_{i(n-1)}]$ de $n=4$ bits é gerada através de $\underline{c}_i=\underline{x}_i\mathbf{G}$ a partir da respectiva mensagem $\underline{x}_i=[x_{i0}\ x_{i1}\ \ldots x_{i(k-1)}]$ de $k=3$ bits. No total, existem $2^k=8$ palavras-código em $\boldsymbol{\theta}(4,3)$. Assim, utilizando (6) para determinar $\underline{c}_i=\underline{x}_i\mathbf{G}$, e lembrando que **no GF(2) a multiplicação é efetuada através da operação AND e a soma é efetuada através da operação XOR**, obtemos:

$\underline{x}_i$	$\underline{x}_i\mathbf{G}=\underline{c}_i$		$\underline{x}_i$	$\underline{x}_i\mathbf{G}=\underline{c}_i$
$[0\ 0\ 0]$	$[0\ 0\ 0]\begin{bmatrix}1&0&0&1\\0&1&0&1\\0&0&1&1\end{bmatrix}=[0\ 0\ 0\ 0]$		$[1\ 0\ 0]$	$[1\ 0\ 0]\begin{bmatrix}1&0&0&1\\0&1&0&1\\0&0&1&1\end{bmatrix}=[1\ 0\ 0\ 1]$
$[0\ 0\ 1]$	$[0\ 0\ 1]\begin{bmatrix}1&0&0&1\\0&1&0&1\\0&0&1&1\end{bmatrix}=[0\ 0\ 1\ 1]$		$[1\ 0\ 1]$	$[1\ 0\ 1]\begin{bmatrix}1&0&0&1\\0&1&0&1\\0&0&1&1\end{bmatrix}=[1\ 0\ 1\ 0]$
$[0\ 1\ 0]$	$[0\ 1\ 0]\begin{bmatrix}1&0&0&1\\0&1&0&1\\0&0&1&1\end{bmatrix}=[0\ 1\ 0\ 1]$		$[1\ 1\ 0]$	$[1\ 1\ 0]\begin{bmatrix}1&0&0&1\\0&1&0&1\\0&0&1&1\end{bmatrix}=[1\ 1\ 0\ 0]$
$[0\ 1\ 1]$	$[0\ 1\ 1]\begin{bmatrix}1&0&0&1\\0&1&0&1\\0&0&1&1\end{bmatrix}=[0\ 1\ 1\ 0]$		$[1\ 1\ 1]$	$[1\ 1\ 1]\begin{bmatrix}1&0&0&1\\0&1&0&1\\0&0&1&1\end{bmatrix}=[1\ 1\ 1\ 1]$

Portanto $\mathbf{G}$ é a matriz geradora do código $\boldsymbol{\theta}(4,3)$ especificado no *codebook* acima.

A Matriz Geradora de um Código $\theta(n,k)$

Qualquer matriz geradora **G** de um código $\boldsymbol{\theta}(n,k)$ pode ser reduzida à **forma sistemática** através de **operações elementares em suas linhas e permutações em suas colunas** quando, então, o código gerado é sistemático.

Uma matriz geradora **G** encontra-se na forma sistemática quando é formatada conforme abaixo:

$$\mathbf{G} = \left[\mathbf{I}_k \;\vdots\; \mathbf{P}\right] = \left[\begin{array}{ccccc|ccc} 1 & 0 & 0 & \cdots & 0 & p_{00} & \cdots & p_{0(n-k-1)} \\ 0 & 1 & 0 & \cdots & 0 & p_{10} & \cdots & p_{1(n-k-1)} \\ \vdots & \vdots & \vdots & & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & \cdots & 1 & p_{(k-1)0} & \cdots & p_{(k-1)(n-k-1)} \end{array}\right] \qquad (7)$$

onde $\mathbf{I}_k$ é a matriz identidade $k \times k$ e **P** é uma matriz $k \times (n-k)$ que determina os $n-k$ bits de paridade na palavra-código $\underline{c_i}$ de n bits a partir dos k bits da mensagem $\underline{x_i}$.

Por exemplo, a matriz **G** de Exemplo 1 no slide anterior está na forma sistemática dada por (7) e portanto o *codebook* do código sistemático $\boldsymbol{\theta}(4,3)$ gerado por **G** exibe palavras código $\underline{c_i}$ tais que os k bits mais à esquerda de cada $\underline{c_i}$ são uma cópia da respectiva mensagem $\underline{x_i}$:

$$\mathrm{G} = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} \Rightarrow \underline{c_i} = \underline{x_i}\mathbf{G} = [b_2 \quad b_1 \quad b_0] \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} \Rightarrow$$

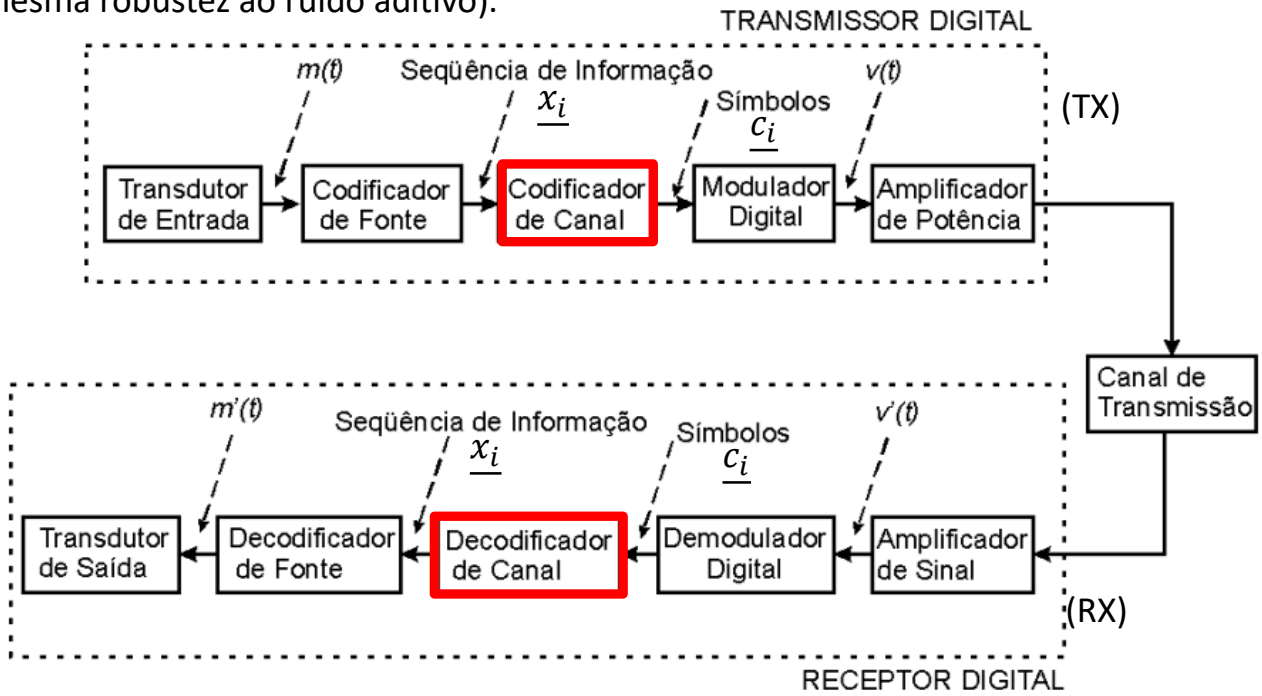
Mensagem $\underline{x_i}$	Palavra-código $\underline{c_i}$
000	0000
001	0011
010	0101
011	0110
100	1001
101	1010
110	1100
111	1111

A Matriz Geradora de um Código $\theta(n,k)$

Dois códigos que diferem somente na ordem de suas palavras-código (símbolos) $\underline{c_i}$ nos seus respectivos *codebooks* no Codificador de Canal no TX resultam no mesmo valor de BER (*bit error rate*) na saída do Decodificador de Canal do RX porque as distâncias de Hamming entre as palavras-código $\underline{c_i}$ de seus respectivos *codebooks* são as mesmas. Tais códigos são denominados **equivalentes**. Lembrando aqui que os erros de bit nas mensagens $\underline{x_i}$ decodificadas na saída do Decodificador de Canal no RX são decorrentes do nível de ruído aditivo no Canal de Transmissão e que excede a capacidade de correção de ambos os códigos, resultando no valor de BER não nula.

Especificamente, o código $\boldsymbol{\theta_e(n,k)}$ **é equivalente** ao código $\boldsymbol{\theta}(n,k)$ se a matriz geradora $\mathbf{G_e}$ de $\boldsymbol{\theta_e}(\boldsymbol{n},\boldsymbol{k})$ puder ser obtida através da **permutação de colunas** da matriz $\mathbf{G}$ geradora de $\boldsymbol{\theta}(n,k)$ ou através de **operações elementares realizadas entre as linhas de $\mathbf{G}$. Uma operação elementar em GF(2) entre duas linhas de uma matriz consiste em permutar as linhas ou em substituir uma linha pela soma dela com outra linha.**

Assim sempre podemos transformar uma matriz $\mathbf{G}$ qualquer para a forma sistemática $\mathbf{G}^*$ através de operações elementares entre linhas ou permutações entre colunas, mantendo a equivalência entre os respectivos códigos gerados (i.e., mantendo a mesma robustez ao ruído aditivo).



Note que a implementação em hardware de um código sistemático é mais simples do que a de um código não-sistemático e, por esta razão, é sempre preferível trabalhar com códigos sistemáticos.

A Matriz Geradora de um Código $\theta(n,k)$

Exemplo 2: Dada a matriz geradora $\mathbf{G} = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$ pede-se: **(a)** Transformar $\mathbf{G}$ para a sua forma sistemática $\mathbf{G}^*$.

(b) Verifique comparativamente se $\mathbf{G}^*$ gera um código equivalente ao código gerado por $\mathbf{G}$.

Solução: **(a)** Visto que a matriz geradora é uma matriz $\mathbf{G}_{3\times 4}$ então o código gerado será um código $\boldsymbol{\theta}(4,3)$. $\mathbf{G}^*$ pode ser obtida pelo seguinte conjunto de operações elementares feito sobre as linhas de $\mathbf{G}$:

Operação Elementar	Matriz $\mathbf{G}$ alterada
$L_0 \leftrightarrow L_2$	$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}$
$L_0 \leftarrow (L_0 + L_1)$	$\begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}$
$L_0 \leftarrow (L_0 + L_2)$	$\mathbf{G}^* = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}$

A Matriz Geradora de um Código $\theta(n, k)$

(b) O código gerado por **G** é:

$\underline{x}_i$	$\underline{x}_i\mathbf{G} = \underline{c}_i$	$\underline{x}_i$	$\underline{x}_i\mathbf{G} = \underline{c}_i$
$[0\ 0\ 0]$	$[0\ 0\ 0]\begin{bmatrix}0&0&1&1\\0&1&0&1\\1&1&1&1\end{bmatrix} = [0\ 0\ 0\ 0]$	$[1\ 0\ 0]$	$[1\ 0\ 0]\begin{bmatrix}0&0&1&1\\0&1&0&1\\1&1&1&1\end{bmatrix} = [0\ 0\ 1\ 1]$
$[0\ 0\ 1]$	$[0\ 0\ 1]\begin{bmatrix}0&0&1&1\\0&1&0&1\\1&1&1&1\end{bmatrix} = [1\ 1\ 1\ 1]$	$[1\ 0\ 1]$	$[1\ 0\ 1]\begin{bmatrix}0&0&1&1\\0&1&0&1\\1&1&1&1\end{bmatrix} = [1\ 1\ 0\ 0]$
$[0\ 1\ 0]$	$[0\ 1\ 0]\begin{bmatrix}0&0&1&1\\0&1&0&1\\1&1&1&1\end{bmatrix} = [0\ 1\ 0\ 1]$	$[1\ 1\ 0]$	$[1\ 1\ 0]\begin{bmatrix}0&0&1&1\\0&1&0&1\\1&1&1&1\end{bmatrix} = [0\ 1\ 1\ 0]$
$[0\ 1\ 1]$	$[0\ 1\ 1]\begin{bmatrix}0&0&1&1\\0&1&0&1\\1&1&1&1\end{bmatrix} = [1\ 0\ 1\ 0]$	$[1\ 1\ 1]$	$[1\ 1\ 1]\begin{bmatrix}0&0&1&1\\0&1&0&1\\1&1&1&1\end{bmatrix} = [1\ 0\ 0\ 1]$

O código gerado por $\mathbf{G}^*$ é o mesmo código especificado no *codebook* do Exemplo 1 no slide 19. Comparativamente, note que as distâncias de Hamming entre as palavras-código $\underline{c_i}$ dos códigos gerados por $\mathbf{G}^*$ e $\mathbf{G}$ são as mesmas. A única diferença que se observa é na ordem de suas palavras-código (símbolos) $\underline{c_i}$ nos seus respectivos *codebooks*. Portanto, ambos os códigos apresentam a mesma robustez ao ruído aditivo.

A Matriz de Paridade de um Código $\boldsymbol{\theta(n,k)}$

Conforme vimos nos slides anteriores, a matriz geradora $\mathbf{G}$ é usada no codificador de canal do TX para gerar o *codebook* de um código $\boldsymbol{\theta}(n,k)$ que, na grande maioria das situações práticas, será um código sistemático. A razão para isto é o fato de que a implementação em hardware de um código sistemático é mais simples do que a de um código não-sistemático.

No decodificador de canal do RX, uma matriz $\mathbf{H}$ denominada de **matriz de paridade** é utilizada para detectar erros de bits nas palavras-código $\underline{c_i}$ recebidas. A matriz de paridade $\mathbf{H}$ do RX é determinada a partir da matriz geradora $\mathbf{G}$ do TX, conforme veremos a seguir. Seja então um código $\boldsymbol{\theta}(n,k)$ com matriz geradora $\mathbf{G}$ dada na forma sistemática (ver slide 20):

$$\mathbf{G} = [\mathbf{I}_k \quad \vdots \quad \mathbf{P}] \tag{7A}$$

Uma observação preliminar: Anteriormente no Cap III estávamos adotando a notação $\underline{c_i} = [c_{i0}\, c_{i1} \ldots c_{i(n-1)}]$ e $\underline{x_i} = [x_{i0}\, x_{i1} \ldots x_{i(k-1)}]$ para atender à representação matricial de $\mathbf{G}$ (ver equação (5) no slide 18). Mas de agora em diante vamos inverter a ordem dos índices na notação de $\underline{c_i}$ e $\underline{x_i}$, de modo que c_{i0} e x_{i0} representem o bit de ordem 0 (i.e., o bit mais à direita em uma palavre binária, conforme é usual em notação de hardware(ver página 3 de https://www.fccdecastro.com.br/pdf/ED_C5.pdf e a página 15 de https://www.fccdecastro.com.br/pdf/ED_C3.pdf).

Seja então a i-ésima palavra-código do *codebook* de um código $\boldsymbol{\theta}(n,k)$ **sistemático** dada por $\underline{c_i} = [c_{i(n-1)}c_{i(n-2)} \ldots c_{i1}c_{i0}]$, a qual resulta da respectiva mensagem $\underline{x_i} = [x_{i(k-1)}x_{i(k-2)} \ldots x_{i1}x_{i0}]$ através da equação (3), i.e., através de $\underline{c_i} = \underline{x_i}\mathbf{G}$.

Dado que $\boldsymbol{\theta}(n,k)$ é sistemático então sua matriz geradora $\mathbf{G}$ é da forma sistemática conforme equação (7A) acima, e portanto a palavra-código $\underline{c_i}$ pode ser decomposta em $\underline{c_i} = [\underline{x_i}\ \underline{a_i}]$ onde $\underline{a_i} = \underline{x_i}\mathbf{P}$ é um **vetor-linha que contém os $\boldsymbol{n-k}$ bits de paridade da palavra-código $\boldsymbol{\underline{c_i}}$.** Visto que $\underline{a_i} = \underline{x_i}\mathbf{P}$, e considerando que a soma em **GF**(2) é uma operação módulo 2 (i.e., soma é XOR), então podemos somar $\underline{x_i}\mathbf{P}$ à esquerda e à direita de $\underline{a_i} = \underline{x_i}\mathbf{P}$, resultando em:

$$\underline{x_i}\mathbf{P} + \underline{a_i} = \underline{x_i}\mathbf{P} + \underline{x_i}\mathbf{P} \Longrightarrow \underline{x_i}\mathbf{P} + \underline{a_i} = \underline{0} \tag{8}$$

que pode ser escrita matricialmente conforme (9) abaixo, definindo algebricamente a matriz de paridade $\mathbf{H}$:

$$\left[\underline{x_i}\ \ \underline{a_i}\right]\begin{bmatrix}\mathbf{P}\\ \mathbf{I}_{n-k}\end{bmatrix} = \underline{0} \tag{9}$$

Matriz de paridade transposta $\mathbf{H}^T$

A Matriz de Paridade de um Código $\theta(n,k)$

Da equação (9) no slide anterior obtemos então a definição para a matriz de paridade $\mathbf{H}$:

$$\mathbf{H} = (\mathbf{H}^T)^T = \begin{bmatrix} \mathbf{P} \\ \mathbf{I}_{n-k} \end{bmatrix}^T = [\mathbf{P}^T \quad \vdots \quad (\mathbf{I}_{n-k})^T] = [\mathbf{P}^T \quad \vdots \quad \mathbf{I}_{n-k}] \tag{10}$$

$$\mathbf{H} = [\mathbf{P}^T \quad \vdots \quad \mathbf{I}_{n-k}] \tag{11}$$

Note em (11) que $\mathbf{H}$ é uma matriz de $(n-k) \times n$ bits. Da equação (9) no slide anterior temos ainda que :

$$\underbrace{[\underline{x_i} \ \underline{a_i}]}_{\underline{c_i}} \underbrace{\begin{bmatrix} \mathbf{P} \\ \mathbf{I}_{n-k} \end{bmatrix}}_{\mathbf{H}^T} = \underline{0} \quad \rightarrow \boxed{\underline{c_i}\mathbf{H}^T = \underline{0}} \tag{12}$$

A equação (12) $\underline{c_i}\mathbf{H}^T = \underline{0}$ estabelece que cada palavra-código $\underline{c_i}$ do código $\boldsymbol{\theta}(n,k)$ é ortogonal a cada linha da matriz $\mathbf{H}$. Isto decorre do fato de que se dois vetores $\underline{u}$ e $\underline{v}$ são ortogonais então o produto escalar $\underline{u} \cdot \underline{v}^T$ resulta nulo, i.e. $u \cdot \underline{v}^T = 0$ (ver https://en.wikipedia.org/wiki/Dot_product).

Deste modo a equação (12) permite detectar se a palavra código $\underline{c_i}$ recebida na entrada do Decodificador de Canal do RX apresenta algum bit recebido em erro em consequência da degradação de sinal imposta pelo ruído aditivo do canal de transmissão. Ou seja, se $\underline{c_i}\mathbf{H}^T = \underline{0}$ então $\underline{c_i}$ foi recebida sem erro em seus bits e se $\underline{c_i}\mathbf{H}^T \neq \underline{0}$ então $\underline{c_i}$ foi recebida com algum bit em erro.

Especificamente, seja $\underline{c_i}$ a palavra-código transmitida e $\underline{y_i}$ a palavra-código recebida:

Se $\underline{y_i}\mathbf{H}^T \neq 0$ então $\underline{y_i} \neq \underline{c_i}$ e, portanto, $\underline{y_i}$ apresenta pelo menos um bit recebido em erro.

Se $\underline{y_i}\mathbf{H}^T = 0$ então $\underline{y_i} = \underline{c_i}$ e, portanto, $\underline{y_i}$ não apresenta bits recebidos em erros.

Por este motivo, $\mathbf{H}_{(n-k)\times n}$ é denominada de matriz de paridade.

A Matriz de Paridade de um Código $\theta(n, k)$

Exemplo 3: Considere a matriz geradora $\mathbf{G} = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$ do Exemplo 2 no slide 22.

Pede-se:

(a) Determine a matriz de paridade **H** do código $\boldsymbol{\theta}(4,3)$ do Exemplo 2.

(b) Verifique se $\mathbf{G}\mathbf{H}^T = \underline{0}$.

(c) Verifique se $\underline{c_i}\mathbf{H}^T = \underline{0}$.

Solução:

(a) A matriz geradora **G** de $\boldsymbol{\theta}(4,3)$ na forma sistemática é (ver solução do item (a) do Exemplo 2 no slide 22):

$$\mathbf{G} = \left[\mathbf{I}_3 \vdots \mathbf{P}\right] = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} \Rightarrow \mathbf{H} = \left[\mathbf{P}^T \vdots \mathbf{I}_{n-k}\right] = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix}$$

Note que **H** é uma matriz de $(n-k) \times n$ bits, com $n = 4$ e $k = 3$.

(b) Verificando se $\mathbf{G}\mathbf{H}^T = \underline{0}$:

$$\mathbf{G}\mathbf{H}^T = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$\surd$ verificado

A Matriz de Paridade de um Código $\theta(n, k)$

(c) Verificando se $\underline{c_i}\mathrm{H}^T = \underline{0}$ (onde o conjunto de palavras-código $\underline{c_i}$ está explicitado na solução do item (b) do Exemplo 2 no slide 23):

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

$\surd$ verificado

Decodificação pela Mínima Distância (MLD - *Maximum-Likelihood Decoding*)

Uma técnica de decodificação conceitualmente simples mas de custo computacional significativo é o ***Maximum-Likelihood Decoding* (MLD)**, conforme já discutido no slide 8 do Cap I.

No RX a palavra-código recebida $\underline{y_i}$, de n bits, é recebida na entrada do decodificador MLD do código $\boldsymbol{\theta}(n,k)$ adotado no TX. O *codebook* do código $\boldsymbol{\theta}(n,k)$ é formado por $M = 2^k$ possíveis palavras-código $\underline{c_j}$, $j = 0, 1, \ldots, M-1$, sendo k o número de bits em cada mensagem $\underline{x_j}$ associada através do *codebook* à respectiva palavra código $\underline{c_j}$ de n bits.

Daí, o decodificador MLD no RX compara $\underline{y_i}$ recebida com as $M = 2^k$ possíveis palavras-código $\underline{c_j}$ *do codebook* do código $\boldsymbol{\theta}(n,k)$ e decide em favor daquela palavra-código $\underline{c_j}$ (portanto, em favor da mensagem $\underline{x_j}$ associada) que é mais próxima da palavra-código $\underline{y_i}$ recebida, proximidade que é medida através da distância de Hamming entre $\underline{y_i}$ e $\underline{c_j}$.

Esta operação de decodificação efetuada pelo decodificador MLD no RX é expressa matematicamente através de:

$$\theta^{-1}\left\{\underline{y_i}\right\} = \arg\min_{\underline{c_j}} \left|\underline{y_i} - \underline{c_j}\right|_H \qquad (13)$$

onde a palavra-código $\underline{c_j}$ pertence ao conjunto $\mathbf{C}$ de palavras-código do *codebook*, i.e., $\underline{c_j} \in \mathbf{C}$, sendo $\mathbf{C} = \left\{\underline{c_i}\right\} = \{\underline{c_0}, \underline{c_1} \cdots \underline{c_{M-1}}\}$ e onde $\left|\underline{y_i} - \underline{c_j}\right|_H$ denota a distância de Hamming entre a palavra-código recebida $\underline{y_i}$ e a palavra-código $\underline{c_j}$ do *codebook* (ver discussão em vermelho no slide 11 do Cap II).

Arranjo padrão e decodificação por síndrome

O problema da técnica de decodificação MLD discutida no slide anterior é a necessidade de calcular todas as 2^k distâncias de Hamming $|\underline{y_i} - \underline{c_j}|_H$ p/ decodificar a palavra-código $\underline{y_i}$ recebida, o que representa algum custo computacional.

Uma maneira mais eficiente de implementar um decodificador MLD, mas aproveitando as propriedades da matriz de paridade $\mathbf{H}_{(n-k)\times n}$ de um código $\boldsymbol{\theta}(n,k)$, é a técnica de decodificação denominada de **Decodificação por Arranjo Padrão** (*Standard Array Decoding* – ver https://en.wikipedia.org/wiki/Standard_array) utilizando o conceito de **síndrome** (https://en.wikipedia.org/wiki/Decoding_methods#Syndrome_decoding).

Conforme veremos a seguir, na decodificação por síndrome o número de distâncias de Hamming $|\underline{y_i} - \underline{c_j}|_H$ calculadas é reduzido de 2^k para 2^{n-k}, lembrando que, na prática, usualmente $n - k < k$.

Seja $\underline{c_i}$ a palavra-código do código $\boldsymbol{\theta}(n,k)$ enviada pelo TX através do canal de transmissão e seja $\underline{y_i}$ a palavra-código recebida na entrada do decodificador do RX.

Devido ao ruído aditivo no canal de transmissão, a palavra-código $\underline{y_i}$ recebida pode conter erros, de modo que $\underline{y_i}$ pode ser expressa por:

$$\underline{y_i} = \underline{c_i} + \underline{e_i} \tag{14}$$

onde $\underline{e_i}$ é o vetor-linha de n bits que representa o **Padrão de Erro** (i.e., os bits errados em $\underline{y_i}$) resultante da degradação do sinal causada pelo ruído aditivo no canal. Por exemplo, seja $\underline{c_i} = [0\ 1\ 0\ 1]$ a palavra-código enviada pelo TX através do canal de transmissão e seja $\underline{y_i} = [0\ 1\ 0\ 0]$ a palavra-código recebida na entrada do decodificador do RX. E daí, de (14), padrão de erro é $\underline{e_i} = [0\ 0\ 0\ 1]$. Note de (14) que se soubermos uma estimativa do valor $\underline{e_i}$ e se somarmos em GF(2) o valor $\underline{e_i}$ a ambos os lados da equação obtemos a estimativa da palavra-código $\underline{c_{dec}}$ transmitida pelo TX (que é a palavra-código decodificada):

$$\underline{y_i} + \underline{e_i} = \underline{c_i} + \underline{e_i} + \underline{e_i} = \underline{c_i} = \underline{c_{dec}} \tag{15}$$

O **Peso do Padrão de Erro** é a distância de Hamming entre e $\underline{y_i}$ *e* $\underline{c_i}$, i.e., é o número de bits em posições correspondentes de $\underline{y_i}$ *e* $\underline{c_i}$ que diferem entre si. Portanto, no exemplo do parágrafo anterior o Peso do Padrão de Erro é 1 porque em $\underline{e_i} = [0\ 0\ 0\ 1]$ há somente um bit de valor 1.

Arranjo padrão e decodificação por síndrome

Pós-multiplicando a equação (14) $\underline{y_i} = \underline{c_i} + \underline{e_i}$ por $\mathbf{H}^T$ obtemos:

$$\underline{y_i}\mathbf{H}^T = \left(\underline{c_i} + \underline{e_i}\right)\mathbf{H}^T = \underbrace{\underline{c_i}\mathbf{H}^T}_{=\underline{0}} + \underline{e_i}\mathbf{H}^T = \underline{e_i}\mathbf{H}^T \qquad (16)$$

$= \underline{0} \Longrightarrow$ ver equação (12) no slide 25

Da equação (16) define-se a **síndrome do padrão de erro**, ou simplesmente **síndrome**, como o vetor $\underline{s_i}$ originado a partir do padrão de erro $\underline{e_i}$ decorrente da palavra-código $\underline{y_i} = \underline{c_i} + \underline{e_i}$ recebida na entrada do decodificador do RX, vetor $\underline{s_i}$ que é dado por:

$$\underline{s_i} = \underline{e_i}\mathbf{H}^T \qquad (17)$$

Note em (17) que a matriz de paridade $\mathbf{H}$ é uma matriz de $(n-k) \times n$ bits, $\underline{e_i}$ é um vetor-linha de n bits e $\underline{s_i}$ é um vetor-linha de $n-k$ bits.

É importante enfatizar que o conjunto de síndromes $\mathbf{S} = \left\{\underline{s_i}\right\}$ é determinado **apenas a partir do conjunto de padrões de erro** $\left\{\underline{e_i}\right\}$, e **não** pelo conjunto $\mathbf{C} = \left\{\underline{c_i}\right\}$ de palavras-código transmitidas, como podemos inferir de $\underline{s_i} = \underline{y_i}\mathbf{H}^T = \underline{e_i}\mathbf{H}^T$.

Observe ainda que:

$\underline{e_i}$ é um vetor-linha de n bits → então existem 2^n possíveis padrões de erro no conjunto $\left\{\underline{e_i}\right\}$.

$\underline{s_i}$ é um vetor-linha de $n-k$ bits → então existem 2^{n-k} possíveis síndromes no conjunto $\mathbf{S}$.

Como existem menos síndromes $\underline{s_i}$ do que padrões de erro $\underline{e_i}$, a consequência é que $\underline{s_i} = \underline{e_i}\mathbf{H}^T$ **mapeia diferentes padrões de erro $\underline{\mathbf{e_i}}$ na mesma síndrome $\underline{\boldsymbol{s_i}}$.**

Arranjo padrão e decodificação por síndrome

O mapeamento do padrão de erro $\underline{e_i}$ em uma síndrome $\underline{s_i}$, através da equação (17) $\underline{s_i} = \underline{e_i}\mathbf{H}^T$ resulta na **Tabela de Síndromes**, com 2^{n-k} síndromes. Por exemplo, abaixo é mostrada a Tabela de Síndromes para um código $\boldsymbol{\theta}(5,2)$.

Síndrome $\underline{s_i}$	Padrão de Erro $\underline{e_i}$
[0 0 0]	[0 0 0 0 0]
[0 0 1]	[0 0 0 0 1]
[0 1 0]	[0 0 0 1 0]
[0 1 1]	[0 1 0 0 0]
[1 0 0]	[0 0 1 0 0]
[1 0 1]	[1 0 0 0 0]
[1 1 0]	[1 1 0 0 0]
[1 1 1]	[1 0 0 1 0]

Arranjo padrão e decodificação por síndrome

O processo de decodificação por síndrome da palavra-código recebida $\underline{y_i}$ é constituído das seguintes etapas:

(I) Determinar a síndrome $\underline{s_i}$ a partir da palavra-código recebida $\underline{y_i}$ através da equação (16) $\underline{s_i} = \underline{y_i}\mathbf{H}^T \quad (= \underline{e_i}\mathbf{H}^T)$.

(II) Identificar o padrão de erro $\underline{e_i}$ associado à síndrome $\underline{s_i}$ obtida em (I) consultando a Tabela de Síndromes para o código $\boldsymbol{\theta}(n, k)$. A Tabela de Síndromes é determinada previamente e uma única vez, sendo única para cada código $\boldsymbol{\theta}(n, k)$.

(III) Determinar a palavra-código decodificada $\underline{c_{dec}}$ através da equação (15) $\underline{c_{dec}} = \underline{y_i} + \underline{e_i}$, onde $\underline{e_i}$ foi obtido em (II).

(IV) Determinar a estimativa da mensagem transmitida $\underline{x_{dec}}$. Para códigos sistemáticos, a mensagem $\underline{x_{dec}}$ corresponde aos primeiros bits da palavra-código $\underline{c_{dec}}$ obtida em (III). Deste modo, para obter $\underline{x_{dec}}$ basta descartar os $n - k$ bits de paridade de $\underline{c_{dec}}$.

Arranjo padrão e decodificação por síndrome

Assim como a Tabela de Síndromes (ver slide 31), o Arranjo Padrão (AP) é também uma tabela que possui 2^{n-k} **linhas**, respectivas às 2^{n-k} possíveis síndromes de um código $\theta(n,k)$, conforme mostrado na tabela abaixo.

O número de **colunas** do AP é 2^k , correspondendo ao número de palavras-código $\underline{c_i}$ no *codebook* do código $\theta(n,k)$.

Para efeito de construção de um AP, a linha superior do AP recebe a designação $\mathrm{L0}$ e a coluna mais à esquerda recebe a designação $\mathrm{C0}$.

O AP é formado de $2^{n-k} \times 2^k = 2^n$ células (i.e. 2^n possíveis padrões de erro – ver slide 29).

Forma geral do Arranjo Padrão (AP) para um código $\theta(n,k)$:

AP p/ $\theta(n,k)$	C0	C1	C2	…	$\mathrm{C}(2^k-1)$
L0	$\underline{e}_0 = \underline{c}_0 = \underline{0}$	$\underline{c}_1$	$\underline{c}_2$	…	$\underline{c}_{(2^k-1)}$
L1	$\underline{e}_1$	$\underline{c}_1 + \underline{e}_1$	$\underline{c}_2 + \underline{e}_1$	…	$\underline{c}_{(2^k-1)} + \underline{e}_1$
L2	$\underline{e}_2$	$\underline{c}_1 + \underline{e}_2$	$\underline{c}_2 + \underline{e}_2$	…	$\underline{c}_{(2^k-1)} + \underline{e}_2$
⋮	⋮	⋮	⋮		⋮
$\mathrm{L}(2^{n-k}-1)$	$\underline{e}_{(2^{n-k}-1)}$	$\underline{c}_1 + \underline{e}_{(2^{n-k}-1)}$	$\underline{c}_2 + \underline{e}_{(2^{n-k}-1)}$	…	$\underline{c}_{(2^k-1)} + \underline{e}_{(2^{n-k}-1)}$

Arranjo padrão e decodificação por síndrome

Na linha L0 do AP são listadas, da esquerda para a direita, as 2^k palavras-código $\underline{c}_i$ do *codebook* de $\boldsymbol{\theta}(n,k)$, cada uma delas representada por um vetor de n bits definido por $\underline{c}_i = [c_{i(n-1)} c_{i(n-2)} \ldots c_{i1} c_{i0}]$.

A palavra-código $\underline{c}_0$ pertencente à célula identificada pela intersecção da coluna C0 com a linha L0 (célula $\mathrm{L0} \times \mathrm{C0}$) obrigatoriamente deve ser aquela representada pelo vetor $\underline{0}$.

Na coluna C0, abaixo da palavra-código $\underline{0}$, são listados, de alto a baixo, os $2^{n-k} - 1$ padrões de erro relativos à palavra-código $\underline{c}_0 = \underline{0}$.

Primeiramente são listados na coluna C0 todos os n padrões de erro de peso 1, isto é, todos os padrões de erro que resultam de uma Distância de Hamming unitária entre a palavra-código $\underline{y}$ recebida e $\underline{c}_0 = \underline{0}$.

Se $2^{n-k} > n$, então lista-se a seguir em C0 todos os possíveis padrões de erro de peso 2. Em seguida lista-se em C0 todos os possíveis padrões de erro de peso 3, e assim sucessivamente até que todas as 2^{n-k} células de C0 estejam preenchidas. Neste contexto, $\underline{e}_0 = \underline{c}_0 = 0$ representa o padrão de erro de peso 0, isto é, representa a não-ocorrência de erro.

AP p/ $\theta(n,k)$	C0	C1	C2	…	$\mathrm{C}(2^k-1)$
L0	$\underline{e}_0 = \underline{c}_0 = \underline{0}$	$\underline{c}_1$	$\underline{c}_2$	…	$\underline{c}_{(2^k-1)}$
L1	$\underline{e}_1$	$\underline{c}_1 + \underline{e}_1$	$\underline{c}_2 + \underline{e}_1$	…	$\underline{c}_{(2^k-1)} + \underline{e}_1$
L2	$\underline{e}_2$	$\underline{c}_1 + \underline{e}_2$	$\underline{c}_2 + \underline{e}_2$	…	$\underline{c}_{(2^k-1)} + \underline{e}_2$
⋮	⋮	⋮	⋮		⋮
$\mathrm{L}(2^{n-k}-1)$	$\underline{e}_{(2^{n-k}-1)}$	$\underline{c}_1 + \underline{e}_{(2^{n-k}-1)}$	$\underline{c}_2 + \underline{e}_{(2^{n-k}-1)}$	…	$\underline{c}_{(2^k-1)} + \underline{e}_{(2^{n-k}-1)}$

Arranjo padrão e decodificação por síndrome

Nota: Visto que cada linha do AP necessita corresponder a uma única síndrome dentre as 2^{n-k} possíveis síndromes, devemos ter o cuidado de, na construção de $C0$, assegurar que distintos padrões de erro de peso maior que 1 em $C0$ correspondam a síndromes que são distintas entre si e que são simultaneamente distintas daquelas síndromes que correspondem a padrões de erro de peso 1 (conforme veremos no Exemplo 4).

AP p/ $\theta(n,k)$	C0	C1	C2	…	$C(2^k-1)$
L0	$\underline{e}_0 = \underline{c}_0 = \underline{0}$	$\underline{c}_1$	$\underline{c}_2$	…	$\underline{c}_{(2^k-1)}$
L1	$\underline{e}_1$	$\underline{c}_1 + \underline{e}_1$	$\underline{c}_2 + \underline{e}_1$	…	$\underline{c}_{(2^k-1)} + \underline{e}_1$
L2	$\underline{e}_2$	$\underline{c}_1 + \underline{e}_2$	$\underline{c}_2 + \underline{e}_2$	…	$\underline{c}_{(2^k-1)} + \underline{e}_2$
⋮	⋮	⋮	⋮		⋮
$L(2^{n-k}-1)$	$\underline{e}_{(2^{n-k}-1)}$	$\underline{c}_1 + \underline{e}_{(2^{n-k}-1)}$	$\underline{c}_2 + \underline{e}_{(2^{n-k}-1)}$	…	$\underline{c}_{(2^k-1)} + \underline{e}_{(2^{n-k}-1)}$

Arranjo padrão e decodificação por síndrome

Dando prosseguimento à construção do AP, soma-se o padrão de erro contido na i-ésima célula de C0 à palavra-código na célula L$i \times$ C1 e coloca-se o resultado da soma na i-ésima célula em C1, com $i = 0{,}1 \cdots 2^{n-k} - 1$.

A seguir, soma-se o padrão de erro contido na i-ésima célula de C0 à palavra-código na célula L$i \times$ C2 e coloca-se o resultado da soma na i-ésima célula em C2, com $i = 0{,}1 \cdots 2^{n-k} - 1$.

E assim faz-se sucessivamente até completar a última coluna C$\left(2^k - 1\right)$, mais à direita no AP.

AP p/ $\theta(n,k)$	C0	C1	C2	⋯	C(2^k-1)
L0	$\underline{e}_0 = \underline{c}_0 = \underline{0}$	$\underline{c}_1$	$\underline{c}_2$	⋯	$\underline{c}_{(2^k-1)}$
L1	$\underline{e}_1$	$\underline{c}_1 + \underline{e}_1$	$\underline{c}_2 + \underline{e}_1$	⋯	$\underline{c}_{(2^k-1)} + \underline{e}_1$
L2	$\underline{e}_2$	$\underline{c}_1 + \underline{e}_2$	$\underline{c}_2 + \underline{e}_2$	⋯	$\underline{c}_{(2^k-1)} + \underline{e}_2$
⋮	⋮	⋮	⋮		⋮
L$(2^{n-k}-1)$	$\underline{e}_{(2^{n-k}-1)}$	$\underline{c}_1 + \underline{e}_{(2^{n-k}-1)}$	$\underline{c}_2 + \underline{e}_{(2^{n-k}-1)}$	⋯	$\underline{c}_{(2^k-1)} + \underline{e}_{(2^{n-k}-1)}$

Arranjo padrão e decodificação por síndrome

Exemplo 4: Seja o Codificador de Canal no TX de um sistema de comunicação digital que utiliza o código de bloco $\boldsymbol{\theta}(n,k)$ gerado pela matriz geradora G abaixo especificada:

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

Pede-se: **(a)** Construa um possível AP para este código $\boldsymbol{\theta}(n,k)$ e a Tabela de Síndromes associada, visando o posterior projeto do decodificador no RX.

(b) Suponha que o TX digital envie a palavra-código $\underline{c}$= [1 0 1 0 1] através do canal de transmissão. O ruído aditivo do canal degrada o sinal de forma que o demodulador no RX entrega para o decodificador a palavra-código $\underline{y}$ = [1 1 1 0 1] (erro no bit b_3 – considerando aqui a representação $[b_4\ b_3\ b_2\ b_1 b_0]$). Verifique a capacidade do decodificador em detectar e corrigir este erro.

(c) Suponha que o ruído/interferência no canal seja intenso de forma que o demodulador no RX entrega para o decodificador a palavra-código $\underline{y}$ = [1 1 1 1 1] (erro nos bits b_1 e b_3). Verifique a capacidade do decodificador em detectar e corrigir este erro duplo.

Solução:

(a) A matriz geradora G não necessita ser transformada por permutação de colunas ou por operações elementares em linhas visto que **já encontra-se na forma sistemática**, isto é:

$$\mathbf{G} = \left[\mathbf{I}_k \vdots \mathbf{P}\right] \Rightarrow \mathbf{G} = \left[\begin{array}{cc:ccc} 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \end{array}\right] = \left[\mathbf{I}_2 \vdots \mathbf{P}\right]$$

Visto que $\mathbf{G}_{k\times n}$ dada no enunciado é uma matriz de dimensões $\mathbf{G}_{2\times 5}$, então o código em questão é $\boldsymbol{\theta}(5,2)$, e, portanto, $n = 5$ e $k = 2$.

As $2^k = 2^2 = 4$ palavras-código $\underline{c_i}$ do código $\boldsymbol{\theta}(5,2)$ gerado por **G** são obtidas da equação (3) $\underline{c_i} = \underline{x_i}\mathbf{G}$ do slide 17, e resultam conforme *codebook* ao lado:

$\underline{x_i}$	$\underline{c_i}$
$\underline{c_0} = [0\ \ 0]\mathbf{G}$	$= [0\ \ 0\ \ 0\ \ 0\ \ 0]$
$\underline{c_1} = [0\ \ 1]\mathbf{G}$	$= [0\ \ 1\ \ 0\ \ 1\ \ 1]$
$\underline{c_2} = [1\ \ 0]\mathbf{G}$	$= [1\ \ 0\ \ 1\ \ 0\ \ 1]$
$\underline{c_3} = [1\ \ 1]\mathbf{G}$	$= [1\ \ 1\ \ 1\ \ 1\ \ 0]$

Arranjo padrão e decodificação por síndrome

Da matriz geradora $\mathbf{G}$ dada no enunciado (que já está na forma sistemática $\mathbf{G} = [\mathbf{I}_k \ \vdots \ \mathbf{P}]$, conforme equação (7A) do slide 24) e da equação (11) do slide 25 obtemos a matriz de paridade $\mathbf{H}$ para este código $\boldsymbol{\theta}(5,2)$:

$$\mathbf{H}_{(n-k)\times n} = \left[\mathbf{P}^T \ \vdots \ \mathbf{I}_{n-k}\right] = \left[\begin{array}{cc:ccc} 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \end{array}\right]$$

Conforme discussão nos slides 32 a 35, para determinar os padrões de erro da coluna C0 do AP precisamos verificar quais as síndromes resultantes dos $n = 5$ padrões de erro de peso 1 para que não ocorra igualdade com as síndromes resultantes dos padrões de erro de peso maior que 1.

Os padrões de erro $\underline{e_i}$ de peso 1 para este código $\boldsymbol{\theta}(5,2)$ são: $[0\ 0\ 0\ 0\ 1]$, $[0\ 0\ 0\ 1\ 0]$, $[0\ 0\ 1\ 0\ 0]$, $[0\ 1\ 0\ 0\ 0]$ e $[1\ 0\ 0\ 0\ 0]$.

O mapeamento destes padrões de erro $\underline{e_i}$ de peso 1 nas respectivas síndromes $\underline{s_i}$ é obtido através da equação (17) $\underline{s_i} = \underline{e_i}\mathbf{H}^T$ no slide 30 e resulta na Tabela de Síndromes ao lado.

As síndromes $\underline{s_i}$ resultantes dos padrões de erro de peso 1 são portanto:

$$[0\ 0\ 1], [0\ 1\ 0], [1\ 0\ 0], [0\ 1\ 1], [1\ 0\ 1]$$

Obviamente a síndrome resultante do padrão de erro de peso 0 (inexistência de erro) é

$$[0\ 0\ 0\ 0\ 0]\ \mathbf{H}^T = [0\ 0\ 0].$$

$\underline{e_i}$	$\underline{e_i}\mathbf{H}^T = \underline{s_i}$
$[0\ 0\ 0\ 0\ 1]$	$[0\ 0\ 0\ 0\ 1]\mathbf{H}^T = [0\ 0\ 1]$
$[0\ 0\ 0\ 1\ 0]$	$[0\ 0\ 0\ 1\ 0]\mathbf{H}^T = [0\ 1\ 0]$
$[0\ 0\ 1\ 0\ 0]$	$[0\ 0\ 1\ 0\ 0]\mathbf{H}^T = [1\ 0\ 0]$
$[0\ 1\ 0\ 0\ 0]$	$[0\ 1\ 0\ 0\ 0]\mathbf{H}^T = [0\ 1\ 1]$
$[1\ 0\ 0\ 0\ 0]$	$[1\ 0\ 0\ 0\ 0]\mathbf{H}^T = [1\ 0\ 1]$

Arranjo padrão e decodificação por síndrome

Conforme slide 33, o AP a ser construído possui $2^{n-k} = 2^{5-2} = 8$ linhas (correspondentes às 2^{n-k} síndromes).

Já determinamos no slide anterior $n+1 = 6$ síndromes (padrões de erro de peso 0 e peso 1).

Ainda faltam determinar $2^{n-k} - (n+1) = 8 - (5+1) = 2$ síndromes.

Estas 2 síndromes faltantes devem **obrigatoriamente** ser distintas entre si e distintas das $n+1 = 6$ síndromes já determinadas (ver nota no slide 35) .

Tendo esta condição em mente, verifica-se na Tabela de Síndromes (ao lado) que as síndromes faltantes são $[1\ 1\ 0]$ e $[1\ 1\ 1]$.

Note que os 2 padrões de erro que resultam respectivamente nestas 2 síndromes $[1\ 1\ 0]$ e $[1\ 1\ 1]$ devem ser padrões de erro de **peso 2,** visto que já esgotamos os possíveis padrões de erro de peso 0 e de peso 1.

Neste contexto, se expressarmos o padrão de erro por $\underline{e_i} = [b_4\ b_3\ b_2\ b_1 b_0]$, onde b_i representa a ordem do bit, e da equação (17) $\underline{s_i} = \underline{e_i}\mathbf{H}^T$ do slide 30 temos então para a síndrome $\underline{s_i} = [1\ 1\ 0]$:

$\underline{e}_i$	$\underline{e}_i\mathbf{H}^T = \underline{s}_i$
$[0\ 0\ 0\ 0\ 1]$	$[0\ 0\ 0\ 0\ 1]\mathbf{H}^T = [0\ 0\ 1]$
$[0\ 0\ 0\ 1\ 0]$	$[0\ 0\ 0\ 1\ 0]\mathbf{H}^T = [0\ 1\ 0]$
$[0\ 0\ 1\ 0\ 0]$	$[0\ 0\ 1\ 0\ 0]\mathbf{H}^T = [1\ 0\ 0]$
$[0\ 1\ 0\ 0\ 0]$	$[0\ 1\ 0\ 0\ 0]\mathbf{H}^T = [0\ 1\ 1]$
$[1\ 0\ 0\ 0\ 0]$	$[1\ 0\ 0\ 0\ 0]\mathbf{H}^T = [1\ 0\ 1]$

$$[1\ \ 1\ \ 0] = [b_4\ \ b_3\ \ b_2\ \ b_1\ \ b_0]\begin{bmatrix}1 & 0 & 1\\ 0 & 1 & 1\\ 1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix} \qquad (18)$$

Arranjo padrão e decodificação por síndrome

$$\begin{bmatrix}1 & 1 & 0\end{bmatrix} = \begin{bmatrix}b_4 & b_3 & b_2 & b_1 & b_0\end{bmatrix} \begin{bmatrix}1 & 0 & 1\\ 0 & 1 & 1\\ 1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix} \quad (18)$$

A equação matricial (18) resulta no seguinte sistema de equações em $\mathrm{GF}(2)$:

$$\begin{aligned} &b_4 + b_2 = 1 \rightarrow b_4 = b_2 + 1 \rightarrow b_4 = \overline{b_2} \\ &b_3 + b_1 = 1 \rightarrow b_3 = b_1 + 1 \rightarrow b_3 = \overline{b_1} \\ &b_4 + b_3 + b_0 = 0 \rightarrow b_2 + 1 + b_1 + 1 + b_0 = 0 \rightarrow b_2 + b_1 + b_0 = 0 \end{aligned} \quad (19)$$

onde $\overline{b}$ representa a negação do valor lógico do bit *b*.

Um possível padrão de erro de peso 2 que obedece às equações (19) acima é $\underline{e_i} = [1\ 1\ 0\ 0\ 0]$. Portanto este será o padrão de erro que associaremos à síndrome $\underline{s_i} = [1\ 1\ 0]$.

Arranjo padrão e decodificação por síndrome

De mesma forma, expressando o padrão de erro por $\underline{e_i} = [b_4\ b_3\ b_2\ b_1 b_0]$, onde b_i representa a ordem do bit, da equação (17) $\underline{s_i} = \underline{e_i}\mathbf{H}^T$ do slide 30 temos para a síndrome $\underline{s_i} = [1\ 1\ 1]$:

$$[1 \quad 1 \quad 1] = [b_4 \quad b_3 \quad b_2 \quad b_1 \quad b_0] \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{20}$$

A equação matricial (20) resulta no seguinte sistema de equações em $\mathrm{GF}(2)$:

$$\begin{aligned} b_4 + b_2 = 1 &\rightarrow b_4 = b_2 + 1 \rightarrow b_4 = \overline{b_2} \\ b_3 + b_1 = 1 &\rightarrow b_3 = b_1 + 1 \rightarrow b_3 = \overline{b_1} \\ b_4 + b_3 + b_0 = 1 &\rightarrow b_2 + 1 + b_1 + 1 + b_0 = 1 \rightarrow b_2 + b_1 + b_0 = 1 \end{aligned} \tag{21}$$

Um possível padrão de erro de peso 2, distinto do anterior, que obedece às equações (21) acima é $\underline{e_i} = [1\ 0\ 0\ 1\ 0]$. Portanto este será o padrão de erro que associaremos à síndrome $\underline{s_i} = [1\ 1\ 1]$.

Arranjo padrão e decodificação por síndrome

De posse destes resultados, e conforme discussão nos slides 32 a 35, o AP é construído como:

Arranjo Padrão:				
	C0	C1	C2	C3
L0	[0 0 0 0 0]	[0 1 0 1 1]	[1 0 1 0 1]	[1 1 1 1 0]
L1	[0 0 0 0 1]	[0 1 0 1 0]	[1 0 1 0 0]	[1 1 1 1 1]
L2	[0 0 0 1 0]	[0 1 0 0 1]	[1 0 1 1 1]	[1 1 1 0 0]
L3	[0 0 1 0 0]	[0 1 1 1 1]	[1 0 0 0 1]	[1 1 0 1 0]
L4	[0 1 0 0 0]	[0 0 0 1 1]	[1 1 1 0 1]	[1 0 1 1 0]
L5	[1 0 0 0 0]	[1 1 0 1 1]	[0 0 1 0 1]	[0 1 1 1 0]
L6	[1 1 0 0 0]	[1 0 0 1 1]	[0 1 1 0 1]	[0 0 1 1 0]
L7	[1 0 0 1 0]	[1 1 0 0 1]	[0 0 1 1 1]	[0 1 1 0 0]

Arranjo padrão e decodificação por síndrome

E a Tabela de Síndromes completa para implementação do decodificador de canal no RX resulta em:

Tabela de Síndromes (implementada em ROM):	
Síndrome $\underline{s}_i$	Padrão de Erro $\underline{e}_i$
[0 0 0]	[0 0 0 0 0]
[0 0 1]	[0 0 0 0 1]
[0 1 0]	[0 0 0 1 0]
[0 1 1]	[0 1 0 0 0]
[1 0 0]	[0 0 1 0 0]
[1 0 1]	[1 0 0 0 0]
[1 1 0]	[1 1 0 0 0]
[1 1 1]	[1 0 0 1 0]

Arranjo padrão e decodificação por síndrome

(b) Da equação (16) $\underline{s_i} = \underline{y_i}\mathbf{H}^T \quad (= \underline{e_i}\mathbf{H}^T)$ do slide (30) e dada a palavra-código recebida $\underline{y_i} = [1\ 1\ 1\ 0\ 1]$, obtemos $\underline{r} = \underline{y_i}\mathbf{H}^T = [0\ 1\ 1]$, onde $\underline{r}$ é a síndrome da palavra-código recebida $\underline{y_i} = [1\ 1\ 1\ 0\ 1]$.

Calculando as $2^{n-k} = 2^{5-2} = 8$ distâncias de Hamming $\left|\underline{r} - \underline{s_i}\right|_H$ na Tabela de Síndromes no slide 43 verifica-se que $\left|\underline{r} - \underline{s_i}\right|_H = 0$ para $\underline{s_i} = [0\ 1\ 1]$ e que o padrão de erro correspondente é $\underline{e_i} = [0\ 1\ 0\ 0\ 0]$.

Da equação (15) do slide 29 obtemos a palavra-código decodificada $\underline{c_{dec}}$ e do *codebook* do slide 37 obtemos a correspondente mensagem $\underline{x_{dec}}$:

$$\underline{c_{dec}} = \underline{y_i} + \underline{e_i} = [1\ 0\ 1\ 0\ 1] \quad \rightarrow \quad \underline{x_{dec}} = [1\ 0]$$

Portanto, para este caso, o decodificador detectou e corrigiu o erro.

Arranjo padrão e decodificação por síndrome

(c) Da equação (16) $\underline{s_i} = \underline{y_i}\mathbf{H}^T \quad (= \underline{e_i}\mathbf{H}^T)$ do slide (30) e dada a palavra-código recebida $\underline{y_i} = [1\ 1\ 1\ 1\ 1]$, obtemos $\underline{r} = \underline{y_i}\mathbf{H}^T = [0\ 0\ 1]$, onde $\underline{r}$ é a síndrome da palavra-código recebida $\underline{y_i} = [1\ 1\ 1\ 1\ 1]$.

Calculando as $2^{n-k} = 2^{5-2} = 8$ distâncias de Hamming $\left|\underline{r} - \underline{s_i}\right|_H$ na Tabela de Síndromes no slide 43 verifica-se que $\left|\underline{r} - \underline{s_i}\right|_H = 0$ para $\underline{s_i} = [0\ 0\ 1]$ e que o padrão de erro correspondente é $\underline{e_i} = [0\ 0\ 0\ 0\ 1]$.

Da equação (15) do slide 29 obtemos a palavra-código decodificada $\underline{c_{dec}}$ e do *codebook* do slide 37 obtemos a correspondente mensagem $\underline{x_{dec}}$:

$$\underline{c_{dec}} = \underline{y_i} + \underline{e_i} = [1\ 1\ 1\ 1\ 0] \quad \rightarrow \quad \underline{x_{dec}} = [1\ 1]$$

Portanto, para este caso, o decodificador detectou o erro mas **não corrigiu** o erro.

Arranjo padrão e decodificação por síndrome

A impossibilidade deste código $\boldsymbol{\theta}(5,2)$ corrigir todos os padrões de erro com peso maior que 1 pode ser também verificada bastando consultar a coluna C0 do AP no slide 42.

Por inspeção da coluna C0 conclui-se que este código corrige todos os 5 padrões de erro de peso 1 possíveis e somente 2 padrões de erro de peso 2, quais sejam, $\underline{e_i}$= [1 1 0 0 0] e $\underline{e_i}$ = [1 0 0 1 0].

Em geral o projetista do código escolhe os padrões de erro de peso w que corrigem w erros com base em alguma peculiaridade do sistema digital.

Neste Exemplo 4 o número total de padrões de erro de peso 2 é dado pela combinação de $n = 5$ bits tomados de $m = 2$ a m, isto é, $Comb(n,m) = Comb(5,2) = 10$, onde $Comb(n,m) = n!/\,[m!\,(n-m)!]$.

No entanto, na construção do AP foi possível utilizar apenas 2 deles: $\underline{e_i} = [1\ 1\ 0\ 0\ 0]$ e $\underline{e_i} = [1\ 0\ 0\ 1\ 0]$.

O projetista do código escolheu então este padrões de erro de peso 2 que corrigem 2 erros simultâneos porque nestas posições de bits está armazenada informação binária que dependem uma da outra e que são cruciais para o desempenho do sistema digital. Por exemplo, o bit de controle do tipo de criptografia (DES, AES) e o bit de controle de quantos bits (128, 256) são usados na criptografia.

Nos links abaixo encontra-se disponível uma videoaula c/ a revisão passo a passo da solução deste exemplo (Exemplo 4 – (slide 37):

Videoaula Codificação de Canal - Decodificação por Arranjo Padrão 1/3 - Profa. Cristina De Castro

Videoaula Codificação de Canal - Decodificação por Arranjo Padrão 2/3 - Profa. Cristina De Castro

Videoaula Codificação de Canal - Decodificação por Arranjo Padrão 3/3 - Profa. Cristina De Castro

Principais Códigos de Blocos Binários - Códigos de Hadamard

Códigos de Hadamard são códigos $\boldsymbol{\theta}(n,k) = \boldsymbol{\theta}(2^k,k)$ caracterizados por $d_{\min} = n/2$.

Em geral, os Códigos de Hadamard apresentam baixa razão de codificação $R_C = k/n = \tau_s/\tau_x = k/2^k$, onde τ_s representa a largura (duração no tempo) dos bits em uma palavra-código e τ_x representa a largura dos bits na respectiva mensagem (ver slide 9).

Portanto, como τ_s/τ_x é pequeno, o uso de um Código de Hadamard implica em um considerável aumento na largura do espectro do sinal transmitido no canal de transmissão, e, por isso, não é muito utilizado, exceto em sistemas *spread spectrum* (que será estudado em Sistemas de Comunicação Digital II).

A matriz geradora de um Código de Hadamard $\boldsymbol{\theta}(n,k) = \boldsymbol{\theta}(2^k,k)$ caracteriza-se pelas suas $n = 2^k$ colunas serem formadas por todos os vetores distintos k dimensionais em $\mathrm{GF}(2)$. Por exemplo, abaixo é mostrado a matriz geradora de um Código de Hadamard para $k = 3$, i.e., $\boldsymbol{\theta}(n,k) = \boldsymbol{\theta}(2^k,k) = \boldsymbol{\theta}(8,3)$:

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

$$\begin{array}{|cccccccc|} \hline 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ \hline \end{array}$$

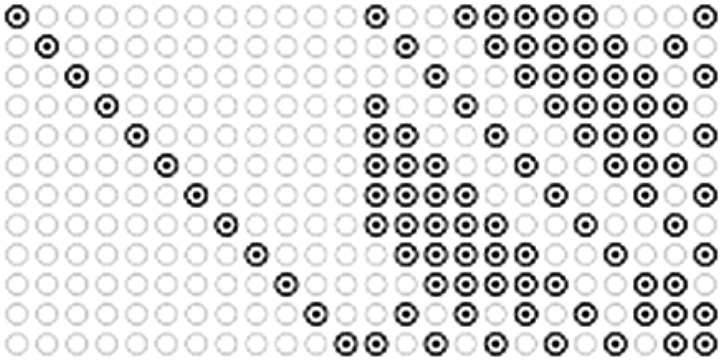
valor decimal do número binário armazenado na respectiva coluna de $\mathbf{G}$

Ver https://en.wikipedia.org/wiki/Hadamard_code .

Principais Códigos de Blocos Binários - Código de Golay

O Código de Golay é um código $\boldsymbol{\theta}(n,k) = \boldsymbol{\theta}(23,12)$ com $d_{\min} = 7$. A matriz geradora $\mathbf{G}_{k\times n} = \mathbf{G}_{12\times 23}$ é conforme abaixo:

$\mathbf{G}_{12\times 23} =$



$\odot = 1$

$\bigcirc = 0$

O Código de Golay apresenta as seguintes características:

- Capacidade de correção de até $t = \left\lfloor \frac{d_{min}-1}{2} \right\rfloor = \left\lfloor \frac{7-1}{2} \right\rfloor = 3$ erros de bit simultâneos
- Capacidade de detecção de até $d = d_{min} - 1 = 7 - 1 = 6$ erros de bit simultâneos.
- O Código de Golay é um código peculiar porque ele é o único código conhecido de 23 bits capaz de corrigir até 3 erros de bit simultâneos.

O Código de Golay é usado nas missões da NASA no *deep space* , como por exemplo, nas missões Voyager 1 e Voyager 2. Ver https://en.wikipedia.org/wiki/Binary_Golay_code .

Principais Códigos de Blocos Binários - Códigos de Hamming

Códigos de Hamming são códigos $\boldsymbol{\theta}(n,k) = \boldsymbol{\theta}(2^m - 1, 2^m - 1 - m)$ largamente utilizados pela facilidade de construção, aliada a uma distância mínima $d_{\min} = 3$ (detecta até 2 erros simultâneos e corrige até 1 erro), sendo $m = n - k$ um inteiro positivo. Por exemplo, se $m = 3$, obtemos um Código de Hamming $\boldsymbol{\theta}(7,4)$.

A construção de um código de bloco $\boldsymbol{\theta}(n,k)$ consiste em definir a sua matriz de paridade $\mathbf{H}_{(n-k)\times n}$ e, a partir da definição de $\mathbf{H}_{(n-k)\times n}$ determinar a sua matriz geradora $\mathbf{G}_{k\times n}$ com base nas equações (7) e (11), i.e., $\mathbf{G}_{k\times n} = [\mathbf{I}_k \ \vdots \ \mathbf{P}]$ e $\mathbf{H}_{(n-k)\times n} = [\mathbf{P}^T \ \vdots \ \mathbf{I}_{n-k}]$.

É uma propriedade de um Código de Hamming $\boldsymbol{\theta}(2^m - 1, 2^m - 1 - m)$ que a sua matriz de paridade $\mathbf{H}_{(n-k)\times n}$ caracteriza-se pelas suas $n = 2^m - 1$ colunas serem formadas por todos os vetores distintos m dimensionais em $\mathrm{GF}(2)$, exceto o vetor $\underline{0}$. Por exemplo, um código $\boldsymbol{\theta}(3,1)$ é um Código de Hamming com $m = 2$ em que a matriz $\mathbf{H}$ é formada pelos $n = 3$ vetores colunas $[0\ 1]^T$, $[1\ 0]^T$, $[1\ 1]^T$, conforme abaixo:

$$\mathbf{H}_{(n-k)\times n} = \mathbf{H}_{2\times 3} = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}$$

Principais Códigos de Blocos Binários - Códigos de Hamming

Exemplo 5: O Codificador de Canal de um TX digital adota um Código de Hamming $\boldsymbol{\theta}(n,k)$ em que as mensagens recebidas do Codificador de Fonte são palavras binárias de $k = 4$ bits. **Pede-se: (a)** Determine a matriz geradora **G** deste código. **(b)** Quantos erros de bit simultâneos este código é capaz de corrigir?

Solução:

(a) Conforme slide anterior, para $k = 4$ temos que $k = 4 = 2^m - 1 - m$, e, resolvendo esta equação para a incógnita m obtemos $m = 3$. E daí $n = 2^m - 1 = 7$, e, portanto, este Código de Hamming é $\boldsymbol{\theta}(n,k) = \boldsymbol{\theta}(7,4)$.

Conforme visto no slide anterior a matriz de paridade $\mathbf{H}_{(n-k)\times n}$ de um Código de Hamming $\boldsymbol{\theta}(2^m - 1, 2^m - 1 - m)$ caracteriza-se pelas suas $n = 2^m - 1$ colunas serem formadas por todos os vetores distintos m dimensionais em $\mathrm{GF}(2)$, exceto o vetor $\underline{0}$. Temos então:

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

1	2	3	4	5	6	7

valor decimal do número binário armazenado na respectiva coluna de **H**

$$\mathbf{H}_{(n-k)\times n} = [\mathbf{P}^T \ \vdots \ \mathbf{I}_{n-k}]$$

(permutação de colunas p/ converter **H** p/ a forma sistemática)

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

$\mathbf{P}^T$ $\quad$ $\mathbf{I}_{n-k}$

A partir de **H** na forma sistemática obtemos:

$$\mathbf{G}_{k\times n} = [\mathbf{I}_k \ \vdots \ \mathbf{P}] \longrightarrow \mathbf{G} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

$\mathbf{I}_k$ $\quad$ $\mathbf{P}$

(b) Todos os Códigos de Hamming têm $d_{\min} = 3$. Assim, da equação 2 no slide 16 temos:

$$t = \left\lfloor \frac{d_{min} - 1}{2} \right\rfloor = \left\lfloor \frac{3-1}{2} \right\rfloor = 1$$

Principais Códigos de Blocos - Códigos Reed-Solomon

Códigos Reed-Solomon constituem uma sub-classe de uma ampla classe de códigos cíclicos denominada de Códigos BCH (Bose – Chaudhuri – Hocquenghem) – ver https://en.wikipedia.org/wiki/Reed%E2%80%93Solomon_error_correction .

Os códigos Reed-Solomon (RS) encontram-se entre os códigos com alta capacidade de correção de erro, sendo largamente utilizados em muitos sistemas digitais como:

–Dispositivos de armazenamento (Fita Magnética,CDs, DVD, códigos de barra, etc.).
–Comunicações Móveis e wireless (Telefonia celular, links de microondas, etc.).
–Comunicações via Satélite.
–Televisão Digital.

Vimos em slides anteriores que um código de bloco binário $\theta(n,k)$ codifica mensagens de k **bits** em palavras-código de n **bits**, podendo corrigir até $t = \left\lfloor \frac{d_{\min}-1}{2} \right\rfloor$ **bits** errados.

Um Código Reed-Solomon $\theta(n,k)$, representado por $\mathbf{RS}(n,k)$, codifica mensagens de k **símbolos** em palavras-código de n **símbolos**, sendo capaz de corrigir até $t = \left\lfloor \frac{n-k}{2} \right\rfloor$ **símbolos** errados.

Cada **símbolo** em uma palavra-código (ou em uma mensagem) de um código $\mathbf{RS}(n,k)$ é um **bloco de m bits**.

Daí, portanto, o poder de correção de erro de um código $\mathbf{RS}(n,k)$: Mesmo que **todos** os m bits de cada um dos t símbolos recebidos estejam errados, o código $\mathbf{RS}(n,k)$ efetua a correção não importando a localização dos símbolos na palavra-código.

Ainda, não importando o número e a posição dos bits errados em cada símbolo, o código $\mathbf{RS}(n,k)$ corrigirá até t símbolos e, caso o número de símbolos errados ultrapassar t, o código $\mathbf{RS}(n,k)$ detectará esta situação.

No contexto do codificador de canal de um sistema de comunicações digitais esta característica é extremamente vantajosa porque permite a correção de um surto de $m \times t$ bits sequenciais recebidos em erro (*error burst correction*).

Se o número de erros ultrapassar t, então o código $\mathbf{RS}(n,k)$ avisa o sistema de que não foi capaz de corrigir todos os erros.

Principais Códigos de Blocos - Códigos Reed-Solomon

É de especial interesse o caso em que $m = 8$, quando cada símbolo representa 1 *byte*.

Por exemplo, consideremos um código $\mathbf{RS}(n,k) = \mathbf{RS}(20,16)$ com $m = 8$. Suponhamos que queiramos codificar a mensagem de $k = 16$ bytes (i.e., $k = 16$ símbolos) abaixo especificada:

255	100	012	098	120	003	233	111	077	163	000	001	088	200	101	007

O código $\mathbf{RS}(20,16)$ adiciona $n - k = 4$ bytes (símbolos) de paridade e codifica a mensagem acima na palavra-código em forma sistemática abaixo:

255	100	012	098	120	003	233	111	077	163	000	001	088	200	101	007	208	107	221	076

O exemplo acima é baseado no CODEC RS descrito em linguagem C no link Reed-Solomon CODEC - 24 KB **(.c)** (CODEC - COder and DECcoder).

Este código $\mathbf{RS}(20,16)$ tem a capacidade de corrigir até $t = \left\lfloor \frac{n-k}{2} \right\rfloor = \left\lfloor \frac{20-16}{2} \right\rfloor = 2$ símbolos $\rightarrow 16$ bits contíguos.

Observe que nenhum símbolo é maior do que 255, valor máximo decimal para 1 byte ($m = 8$ bits).

Observe também que as operações entre polinômios são todas executadas em $\mathrm{GF}(2^m) = GF(2^8) = \mathrm{GF}(256)$.

Foge ao escopo deste texto o estudo da álgebra de polinômios em $\mathrm{GF}(2^m)$ e, portanto, não nos aprofundaremos na teoria dos Códigos Reed-Solomon.

Códigos LDPC – Low-Density Parity-Check

Os códigos *Low-Density Parity-Check* (LDPC) são uma subcategoria dos códigos de bloco lineares e foram, originalmente, introduzidos por Gallager nos anos 1960 (ver https://en.wikipedia.org/wiki/Low-density_parity-check_code). Códigos LDPC são códigos de bloco com matriz de paridade $\boldsymbol{H}$ com muitos 0s e poucos 1s.

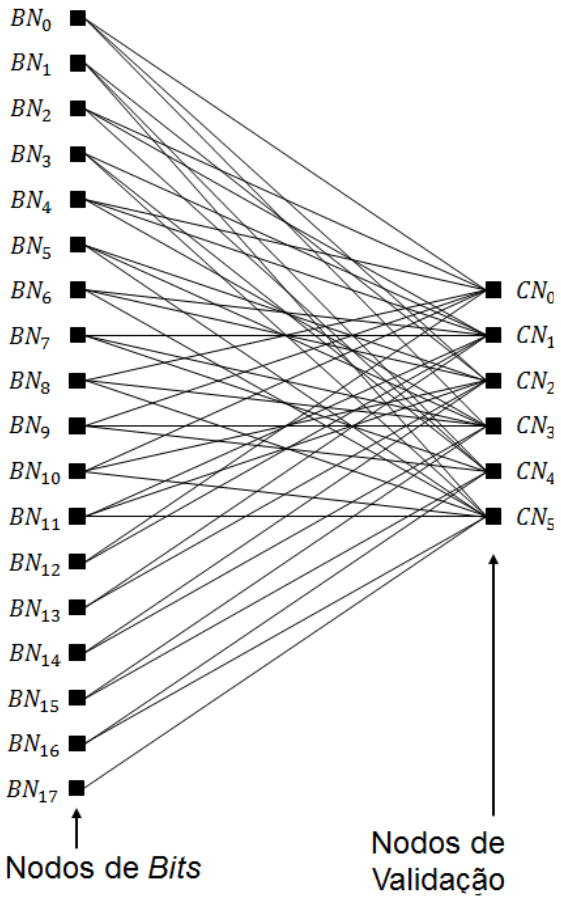
Códigos LDPC longos, quando decodificados com o algoritmo Soma-Produto (SPA – ver https://en.wikipedia.org/wiki/Belief_propagation e ver seção 5 de https://www.fccdecastro.com.br/pdf/LDPCintro.pdf), são capazes de atingir um desempenho muito próximo ao limite de Shannon.

Além do excelente desempenho, o processo de codificação e decodificação adotado pelos códigos LDPC é menos complexo, quando comparado a uma outra classe de códigos cujo desempenho também aproxima-se do Limite de Shannon – os denominados Turbo Códigos (que discutiremos adiante).

A decodificação dos códigos LDPC é realizada através de um **processo iterativo do tipo *soft-decision*** denominado *message passing algorithm* (especificamente a subclasse ***belief propagation algorithm* – BPA**), onde as probabilidades de ocorrência dos bits das palavras-código são passadas entre o conjunto de nodos de validação CN (*check nodes*) e o conjunto de nodos de bits BN (*bit nodes*), representados por um **Grafo de Tanner**, conforme mostrado ao lado (ver APÊNDICE A no slide 92). Em cada rodada de iterações as probabilidades dos bits são passadas dos nodos BN p/ os nodos CN e dos nodos CN de volta aos nodos BN até que as probabilidades dos bits indiquem a convergência do processo iterativo.

O grafo de Tanner ao lado representa a equação (12) do slide 25 ,i.e., $\underline{c_i}\mathbf{H}^T = \underline{0}$, sendo $\underline{c_i}$ correspondente à palavra binária aplicada ao nodo BN, e o resultado da equação (síndrome) correspondente à palavra binária resultante nos nodos CN. A diferença aqui é que as operações não são efetuadas no GF(2) c/ o valor do bits, mas sim em termos de *soft-decisions* do BPA baseadas na probabilidade de ocorrência do valor do bit.

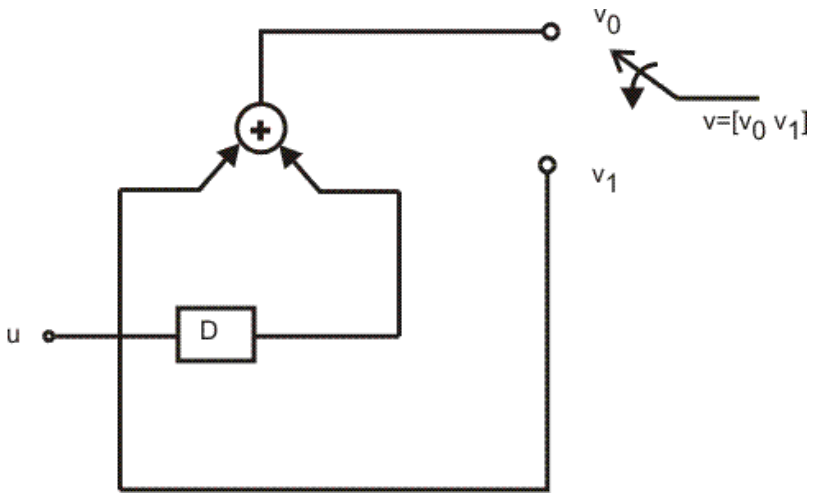
Neste link do GitHub https://github.com/topics/ldpc estão disponibilizados códigos fonte em C, C++, Python e scripts Matlab p/ a implementação prática de códigos LDPC.

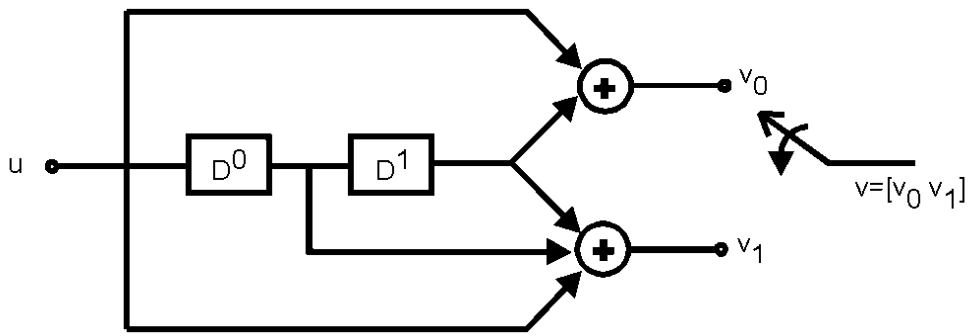


Códigos Convolucionais – Decodificador de Viterbi

- Um código convolucional é gerado pela combinação linear em $\mathbf{GF}(2)$ das saídas de um *shift-register* de *K* estágios.
- A sequência de bits a ser codificada é aplicada na entrada do *shift-register*, e este executa a convolução em $\mathbf{GF}(2)$ entre a sequência de entrada e a resposta ao impulso da **máquina de estado** (*state machine*) representada pelo *shift-register*.
- A saída da máquina de estado constitui, portanto, a sequência codificada.
- Diferem dos códigos de bloco porque o mapeamento entrada/saída do codificador não é dado por uma matriz geradora e, sim, dependente do estado da máquina.

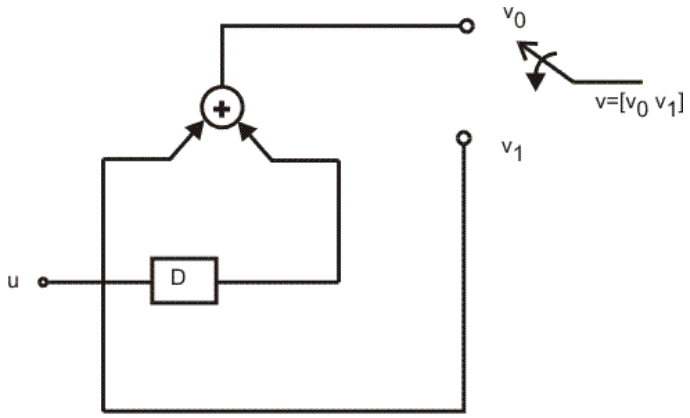
Exemplos de Codificadores Convolucionais



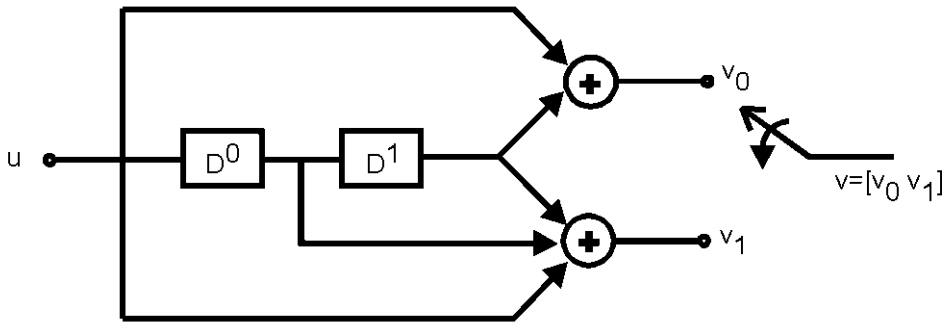


Códigos Convolucionais – Decodificador de Viterbi

- O nº. de estados da máquina de estado de um codificador convolucional é 2^K, sendo *K* o nº. de estágios do *shift-register*.
- No contexto de códigos convolucionais (K + 1) recebe o nome de *constraint length*.
- A razão entre o nº. de entradas e o nº. de saídas da máquina de estados define a razão de codificação R_c do codificador.



Codificador Convolucional com $K = 1$ estágio ($2^K = 2$ estados) e $R_c = 1/2$.



Codificador Convolucional com $K = 2$ estágios ($2^K = 4$ estados) e $R_c = 1/2$.

Códigos Convolucionais – Decodificador de Viterbi

Um codificador Convolucional de 1 estágio, 2 estados e $R_c = 1/2$ poderá apresentar, por exemplo, o seguinte mapeamenteo entrada/saída →

Entrada	Estado	Saída
0	0	O1
1	0	10
0	1	11
1	1	00

Se o mapeamento entrada/saída do codificador não é dado por uma matriz geradora e, sim, dependente do estado da máquina, uma entrada "1", como no exemplo acima, poderá ser mapeada em uma saída "10", mas também poderá ser mapeada em uma saída "00".

Por que este codificador será utilizável?

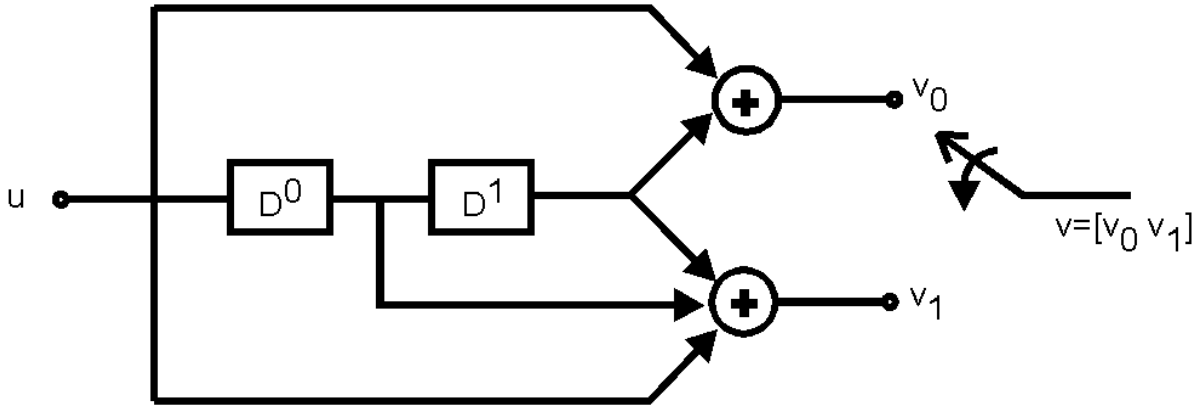
Como uma máquina de estados construída a partir de um shift-register apresenta um conjunto finito de transições permitidas entre estados, quando a sequência a ser codificada é a ela submetida, implicitamente ficarão restritas as transições da sequência codificada em sua saída.

Ou seja, há um conjunto conhecido e finito de possíveis transições de estado.

Se o receptor conhecer a tabela de transições permitidas, então os erros gerados por degradação do sinal no canal de comunicações poderão ser identificados e corrigidos por um Decodificador de Viterbi.

Códigos Convolucionais – Decodificador de Viterbi

A Figura abaixo apresenta um codificador convolucional com 2 estágios e 4 estados e $R_c = 1/2$.



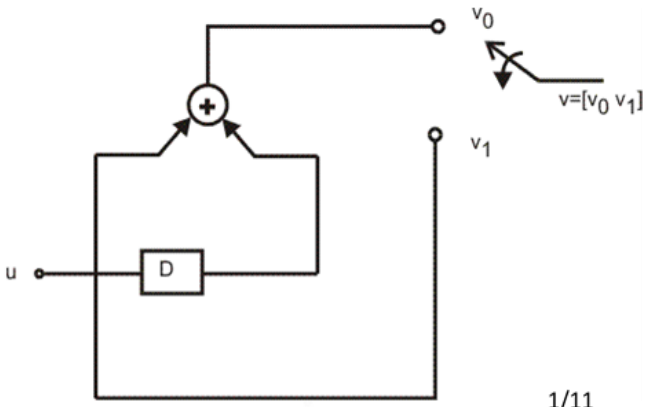
- A sequência de bits a ser codificada é representada por u, e a saída do codificador é a sequência de bits v.
- Visto que $R_c = 1/2$, para cada bit de u são gerados dois bits em v.
- O estágio D^0 transfere o valor lógico em sua entrada para a sua saída **imediatamente após** a ocorrência da borda de descida do pulso de *clock* (não representado na figura).
- De forma idêntica opera o estágio D^1 .
- Representando a saída do estágio D^0 por D^0 e representando a saída do estágio D^1 por D^1, então o par de bits $D^0\ D^1$ identifica um dos estados da máquina de estado.

Códigos Convolucionais – Decodificador de Viterbi

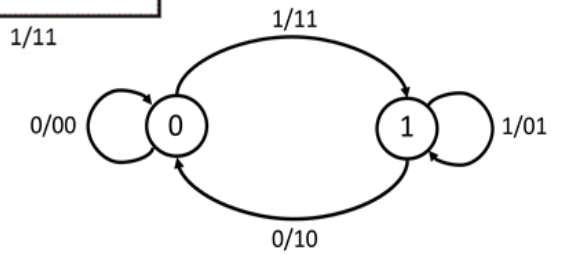
Estudaremos a codificação e a decodificação no contexto de Códigos Convolucionais, por meio de dois exemplos:

Exemplo 1:

A Figura ao lado apresenta o diagrama de transição de estados de um codificador convolucional com $K = 1$ estágio, $2^K = 2$ estados e $R_c = 1/2$.



No diagrama de transição de estados deste codificador, Figura ao lado, cada círculo representa um estado (D) dentre os 2 possíveis estados.



O diagrama é construído a partir dos estados individuais considerando as **transições permitidas** a partir de cada estado como consequência do valor lógico de u.

Supondo que a máquina de estado encontre-se no estado **0** (i.e., $D = 0$ na Figura acima).

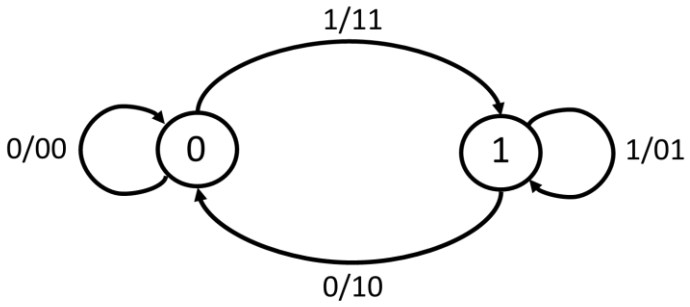
- Se $u = 1$ a saída resultante é $v = 11$ e, após a borda de descida do *clock*, a máquina vai para o estado **1**.
- Se $u = 0$ a saída resultante é $v = 00$ e, após a borda de descida do *clock*, a máquina vai para o estado **0**.

Supondo que a máquina de estado encontre-se no estado **1** (i.e., $D = 1$ na Figura acima).

- Se $u = 1$ a saída resultante é $v = 01$ e, após a borda de descida do *clock*, a máquina vai para o estado **1**.
- Se $u = 0$ a saída resultante é $v = 10$ e, após a borda de descida do *clock*, a máquina vai para o estado **0**.

Códigos Convolucionais – Decodificador de Viterbi

Diagrama de transição de estados



Exemplo de codificação p/ o codificador do slide anterior

u	1	0	1	1	1
D_{atual}	0	1	0	1	1
D_{futuro}	1	0	1	1	1
v	11	10	11	01	01
r	11	10	11	11	01

Dada uma seqüência *u* a ser codificada (1 0 1 1 1), a saída *v* no codificador de um transmissor digital é enviada ao receptor através do canal de transmissão, sendo recebida como uma sequência *r*.

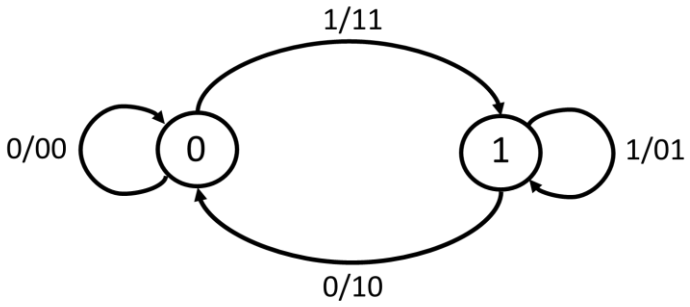
A Tabela acima apresenta uma possível sequência *u* e a resultante sequência *v* para o codificador da Figura.

É mostrada também a trajetória do estado D à medida que *u* é codificada, partindo inicialmente do estado 0.

Se nenhuma degradação de sinal ocorreu no canal de transmissão, $r = v$.

Códigos Convolucionais – Decodificador de Viterbi

Diagrama de transição de estados



Exemplo de codificação p/ o codificador do slide anterior

u	1	0	1	1	1
D_{atual}	0	1	0	1	1
D_{futuro}	1	0	1	1	1
v	11	10	11	01	01
r	11	10	11	11	01

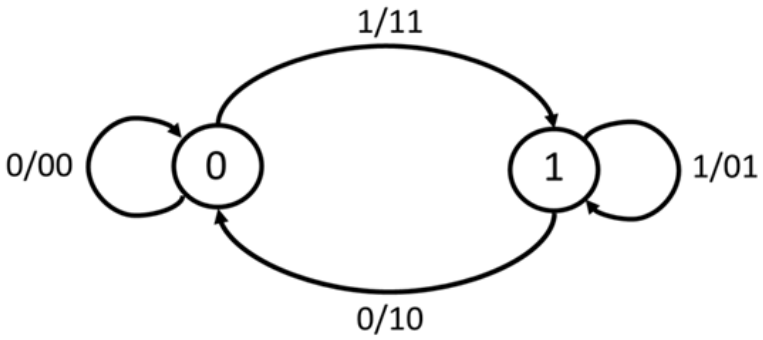
Assumindo que v seja enviado através de um canal de transmissão com ruído/interferência, a Tabela mostra uma possível sequência r recebida com 1 erro.

No receptor digital, o decodificador utiliza um algoritmo de decodificação baseado no princípio de mínima distância (MLSE – *maximum likelihood sequence detector*) denominado Algoritmo de Viterbi.

Vamos decodificar a sequência r da Tabela através do Algoritmo de Viterbi para testar a capacidade de correção de erros do mesmo.

Códigos Convolucionais – Decodificador de Viterbi

O diagrama de treliça mostra todas as trajetórias (caminhos) das transições de estado da máquina de estado do codificador a cada instante *i* de codificação, a partir do estado 00.



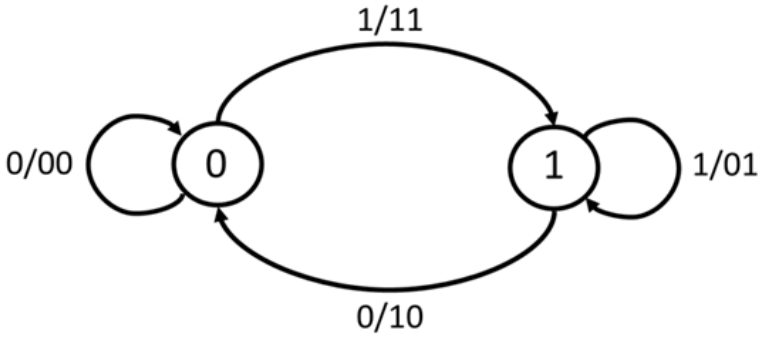
Cada ramo da treliça começa e termina em um estado, representando, assim, uma transição permitida.

Cada ramo é identificado por u/v_0v_1, isto é, a saída v do codificador quando, ao aplicarmos u em sua entrada, a máquina de estado executa a transição representada pelo ramo em questão.

A partir do diagrama de transição de estados, inicia-se a montagem da treliça de Viterbi adequada ao codificador convolucional da Figura, para decodificação.

Códigos Convolucionais – Decodificador de Viterbi

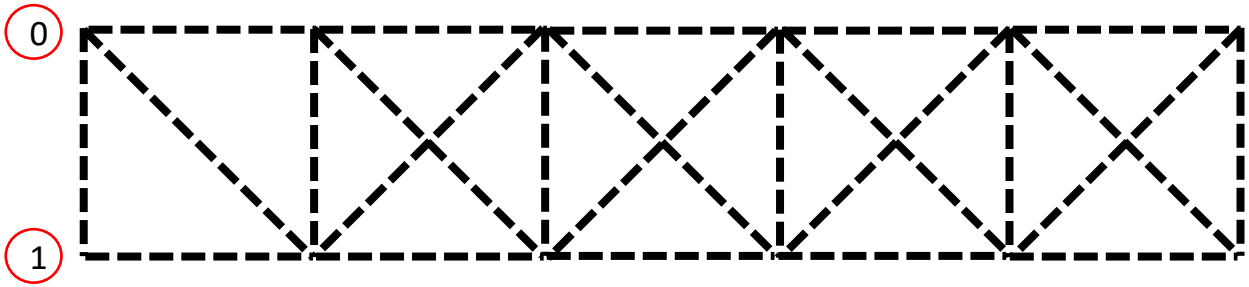
O diagrama de treliça mostra todas as trajetórias (caminhos) das transições de estado da máquina de estado do codificador a cada instante *i* de codificação, a partir do estado 00.



Cada ramo da treliça começa e termina em um estado, representando, assim, uma transição permitida.

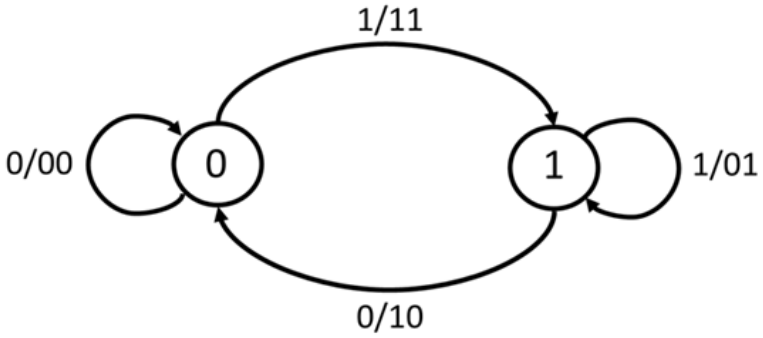
Cada ramo é identificado por u/v_0v_1, isto é, a saída v do codificador quando, ao aplicarmos u em sua entrada, a máquina de estado executa a transição representada pelo ramo em questão.

A partir do diagrama de transição de estados, inicia-se a montagem da treliça de Viterbi adequada ao codificador convolucional da Figura, para decodificação.



Códigos Convolucionais – Decodificador de Viterbi

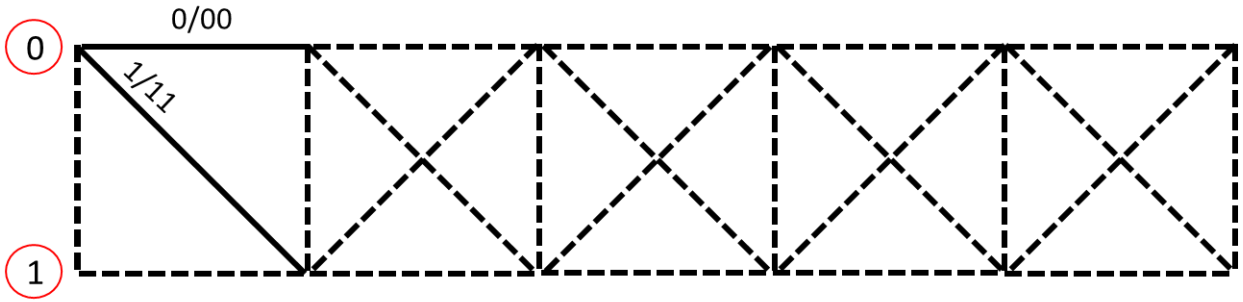
O diagrama de treliça mostra todas as trajetórias (caminhos) das transições de estado da máquina de estado do codificador a cada instante *i* de codificação, a partir do estado 00.



Cada ramo da treliça começa e termina em um estado, representando, assim, uma transição permitida.

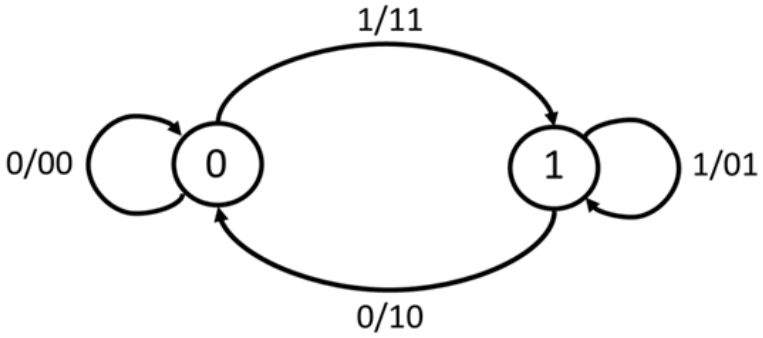
Cada ramo é identificado por u/v_0v_1, isto é, a saída v do codificador quando, ao aplicarmos u em sua entrada, a máquina de estado executa a transição representada pelo ramo em questão.

A partir do diagrama de transição de estados, inicia-se a montagem da treliça de Viterbi adequada ao codificador convolucional da Figura, para decodificação.



Códigos Convolucionais – Decodificador de Viterbi

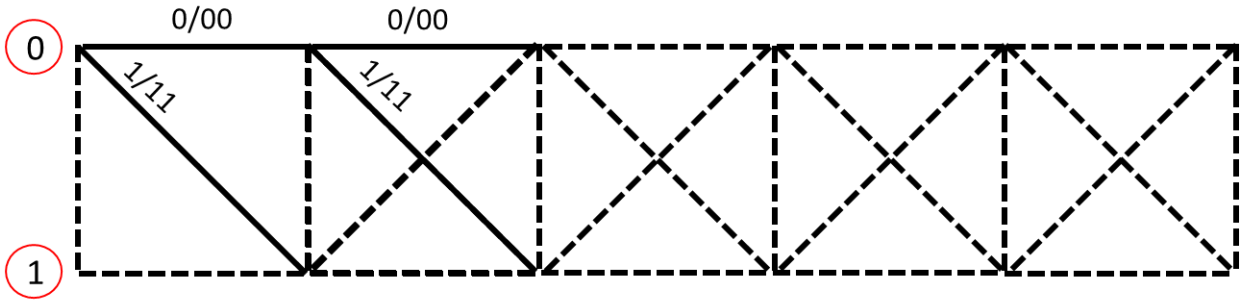
O diagrama de treliça mostra todas as trajetórias (caminhos) das transições de estado da máquina de estado do codificador a cada instante *i* de codificação, a partir do estado 00.



Cada ramo da treliça começa e termina em um estado, representando, assim, uma transição permitida.

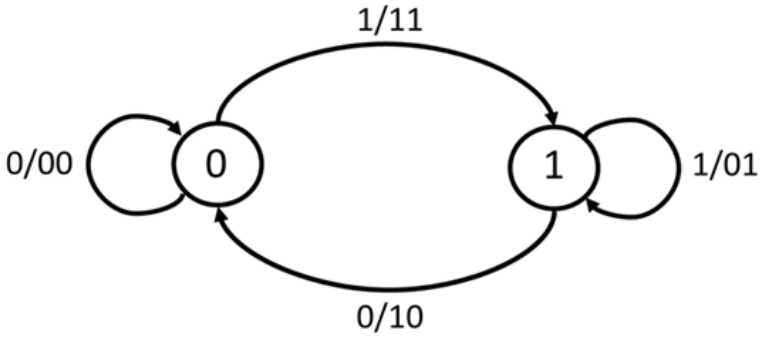
Cada ramo é identificado por u/v_0v_1, isto é, a saída v do codificador quando, ao aplicarmos u em sua entrada, a máquina de estado executa a transição representada pelo ramo em questão.

A partir do diagrama de transição de estados, inicia-se a montagem da treliça de Viterbi adequada ao codificador convolucional da Figura, para decodificação.



Códigos Convolucionais – Decodificador de Viterbi

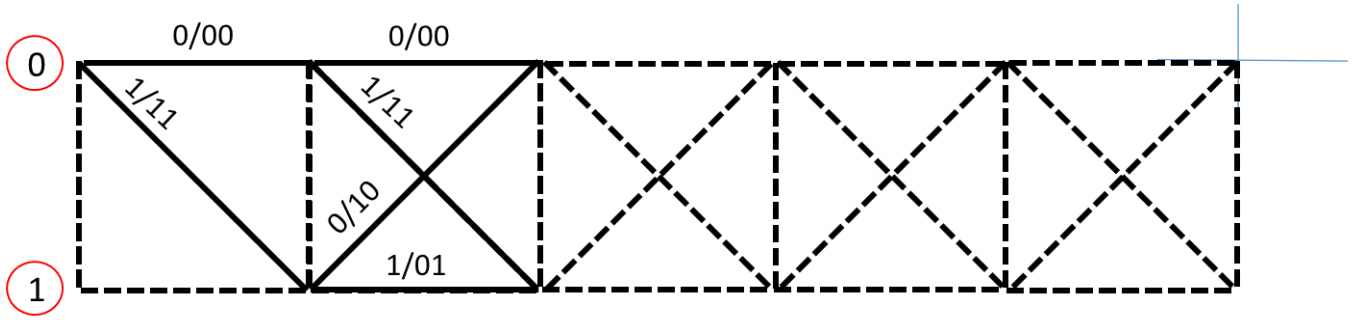
O diagrama de treliça mostra todas as trajetórias (caminhos) das transições de estado da máquina de estado do codificador a cada instante *i* de codificação, a partir do estado 00.



Cada ramo da treliça começa e termina em um estado, representando, assim, uma transição permitida.

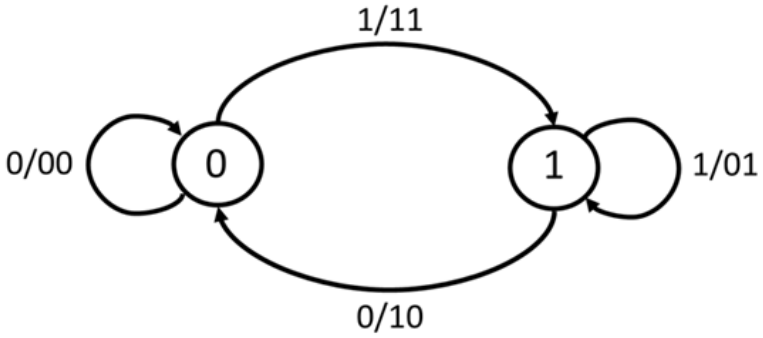
Cada ramo é identificado por u/v_0v_1, isto é, a saída v do codificador quando, ao aplicarmos u em sua entrada, a máquina de estado executa a transição representada pelo ramo em questão.

A partir do diagrama de transição de estados, inicia-se a montagem da treliça de Viterbi adequada ao codificador convolucional da Figura, para decodificação.



Códigos Convolucionais – Decodificador de Viterbi

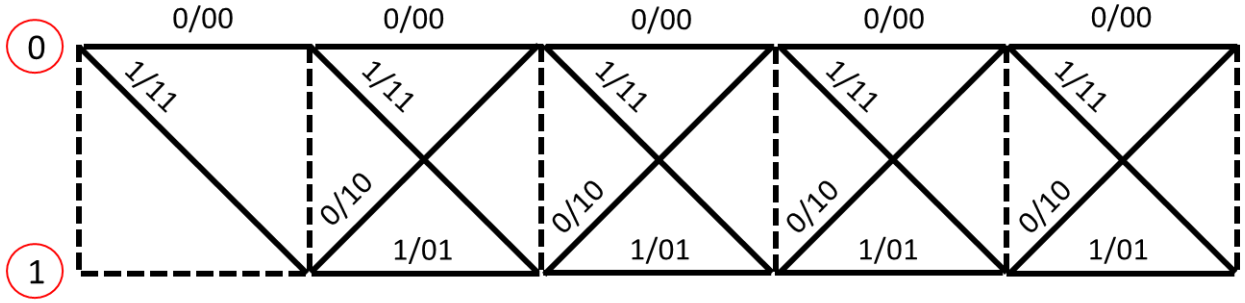
O diagrama de treliça mostra todas as trajetórias (caminhos) das transições de estado da máquina de estado do codificador a cada instante *i* de codificação, a partir do estado 00.



Cada ramo da treliça começa e termina em um estado, representando, assim, uma transição permitida.

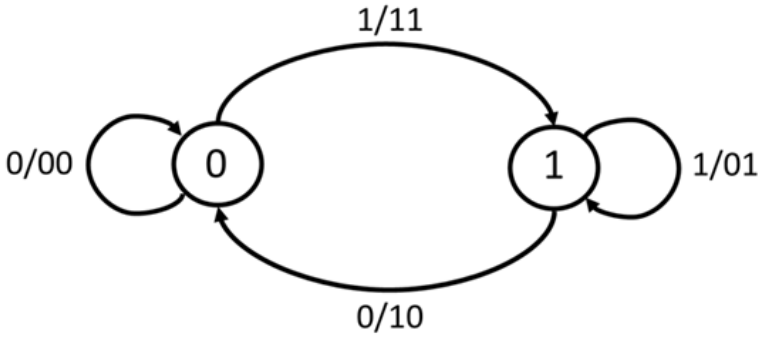
Cada ramo é identificado por u/v_0v_1, isto é, a saída v do codificador quando, ao aplicarmos u em sua entrada, a máquina de estado executa a transição representada pelo ramo em questão.

A partir do diagrama de transição de estados, inicia-se a montagem da treliça de Viterbi adequada ao codificador convolucional da Figura, para decodificação.



Códigos Convolucionais – Decodificador de Viterbi

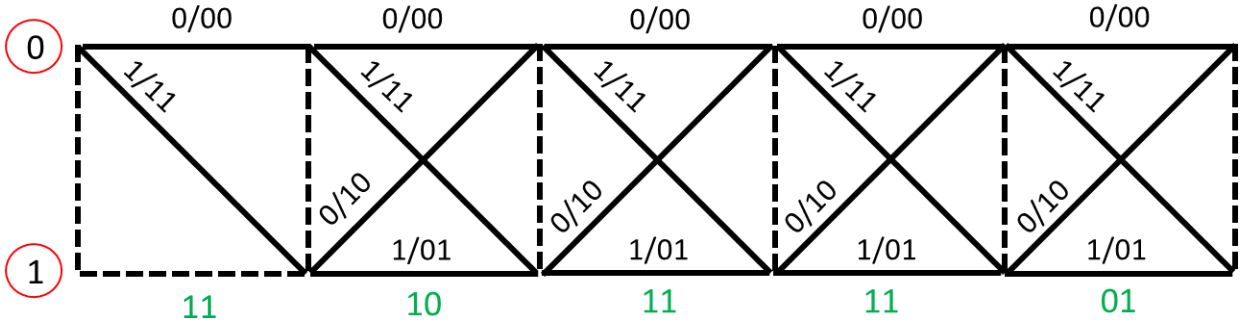
O diagrama de treliça mostra todas as trajetórias (caminhos) das transições de estado da máquina de estado do codificador a cada instante *i* de codificação, a partir do estado 00.



Cada ramo da treliça começa e termina em um estado, representando, assim, uma transição permitida.

Cada ramo é identificado por u/v_0v_1, isto é, a saída v do codificador quando, ao aplicarmos u em sua entrada, a máquina de estado executa a transição representada pelo ramo em questão.

A partir do diagrama de transição de estados, inicia-se a montagem da treliça de Viterbi adequada ao codificador convolucional da Figura, para decodificação.



Códigos Convolucionais – Decodificador de Viterbi

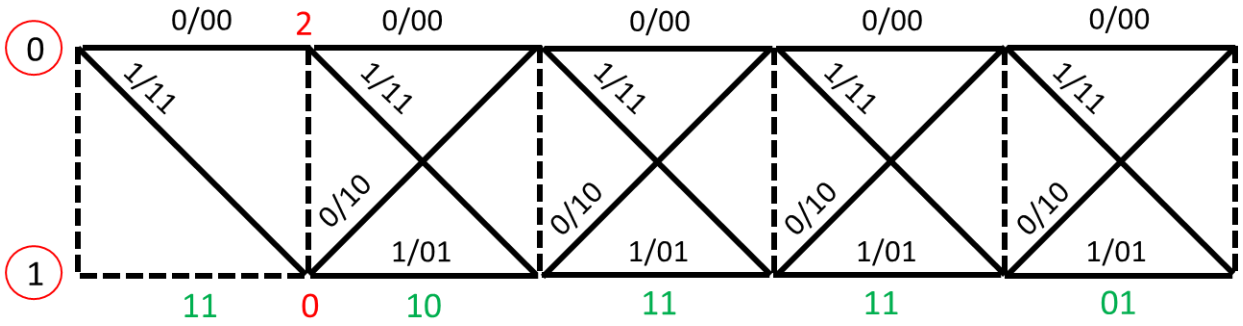
A técnica de decodificação consiste em acumular em cada nó da treliça as Distâncias de Hamming entre a saída v do codificador e a sequência r recebida a cada instante i.

Se mais de um caminho chega a um nó "mata-se" aqueles de maior **métrica** (maior distância acumulada) – caminhos marcados com **x** na Figura a seguir – ficando apenas aquele de menor métrica, denominado de **caminho sobrevivente**.

A métrica acumulada de cada caminho encontra-se em negrito à direita de cada nó, na Figura.

Métricas em vermelho representam ramos que incidem no nó "por cima" e métricas em azul representam ramos que incidem no nó "por baixo", já que, no máximo 2 ramos incidem em um nó para este decodificador.

Métricas sublinhadas representam métricas de caminhos sobreviventes.



Códigos Convolucionais – Decodificador de Viterbi

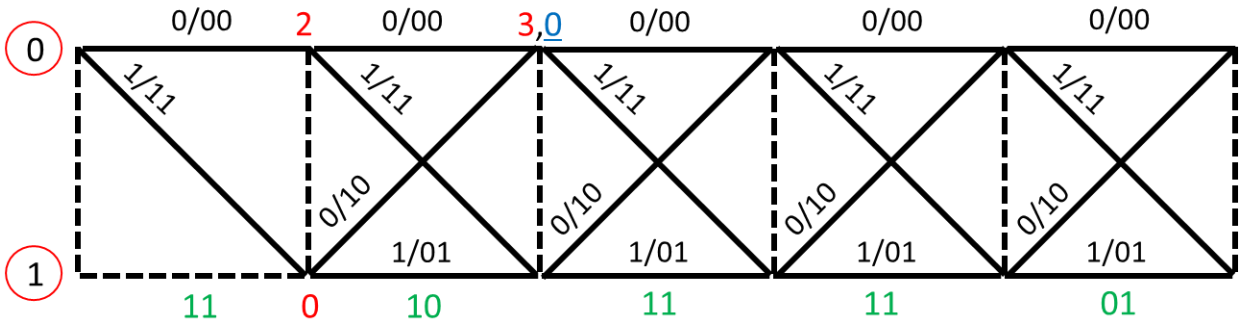
A técnica de decodificação consiste em acumular em cada nó da treliça as Distâncias de Hamming entre a saída v do codificador e a sequência r recebida a cada instante i.

Se mais de um caminho chega a um nó "mata-se" aqueles de maior **métrica** (maior distância acumulada) – caminhos marcados com **x** na Figura a seguir – ficando apenas aquele de menor métrica, denominado de **caminho sobrevivente**.

A métrica acumulada de cada caminho encontra-se em negrito à direita de cada nó, na Figura.

Métricas em vermelho representam ramos que incidem no nó "por cima" e métricas em azul representam ramos que incidem no nó "por baixo", já que, no máximo 2 ramos incidem em um nó para este decodificador.

Métricas sublinhadas representam métricas de caminhos sobreviventes.



Códigos Convolucionais – Decodificador de Viterbi

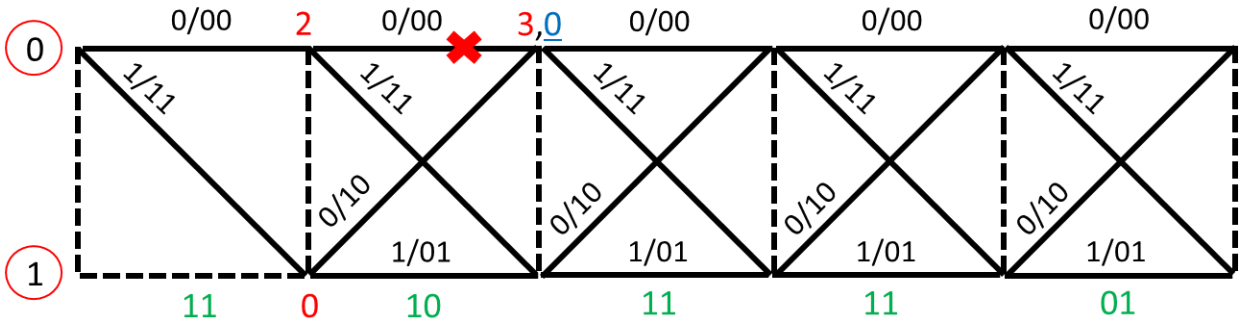
A técnica de decodificação consiste em acumular em cada nó da treliça as Distâncias de Hamming entre a saída v do codificador e a sequência r recebida a cada instante i.

Se mais de um caminho chega a um nó "mata-se" aqueles de maior **métrica** (maior distância acumulada) – caminhos marcados com **x** na Figura a seguir – ficando apenas aquele de menor métrica, denominado de **caminho sobrevivente**.

A métrica acumulada de cada caminho encontra-se em negrito à direita de cada nó, na Figura.

Métricas em vermelho representam ramos que incidem no nó "por cima" e métricas em azul representam ramos que incidem no nó "por baixo", já que, no máximo 2 ramos incidem em um nó para este decodificador.

Métricas sublinhadas representam métricas de caminhos sobreviventes.



Códigos Convolucionais – Decodificador de Viterbi

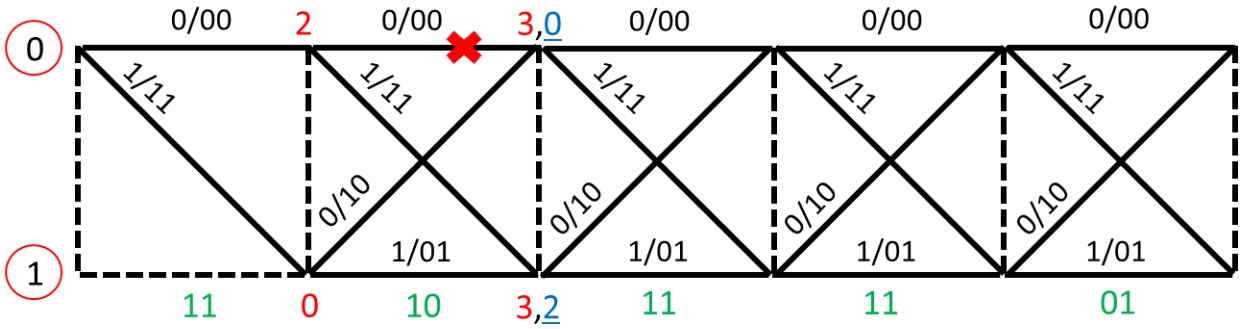
A técnica de decodificação consiste em acumular em cada nó da treliça as Distâncias de Hamming entre a saída v do codificador e a sequência r recebida a cada instante i.

Se mais de um caminho chega a um nó "mata-se" aqueles de maior **métrica** (maior distância acumulada) – caminhos marcados com **x** na Figura a seguir – ficando apenas aquele de menor métrica, denominado de **caminho sobrevivente**.

A métrica acumulada de cada caminho encontra-se em negrito à direita de cada nó, na Figura.

Métricas em vermelho representam ramos que incidem no nó "por cima" e métricas em azul representam ramos que incidem no nó "por baixo", já que, no máximo 2 ramos incidem em um nó para este decodificador.

Métricas sublinhadas representam métricas de caminhos sobreviventes.



Códigos Convolucionais – Decodificador de Viterbi

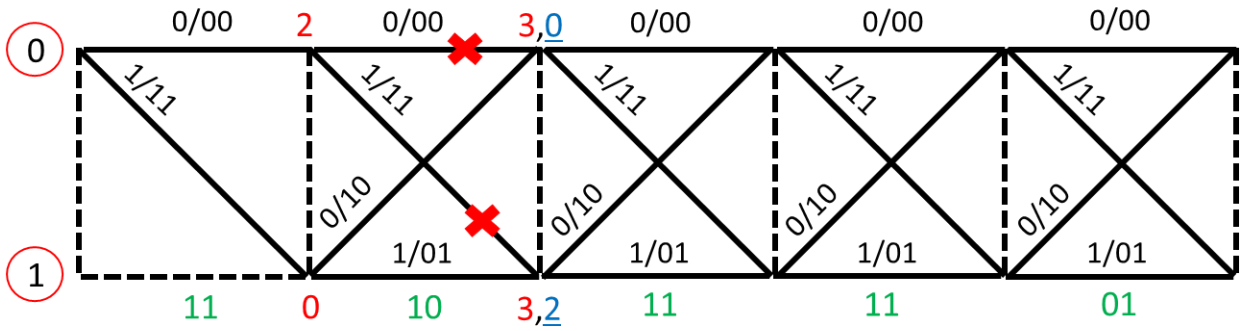
A técnica de decodificação consiste em acumular em cada nó da treliça as Distâncias de Hamming entre a saída v do codificador e a sequência r recebida a cada instante i.

Se mais de um caminho chega a um nó "mata-se" aqueles de maior **métrica** (maior distância acumulada) – caminhos marcados com **x** na Figura a seguir – ficando apenas aquele de menor métrica, denominado de **caminho sobrevivente**.

A métrica acumulada de cada caminho encontra-se em negrito à direita de cada nó, na Figura.

Métricas em vermelho representam ramos que incidem no nó "por cima" e métricas em azul representam ramos que incidem no nó "por baixo", já que, no máximo 2 ramos incidem em um nó para este decodificador.

Métricas sublinhadas representam métricas de caminhos sobreviventes.



Códigos Convolucionais – Decodificador de Viterbi

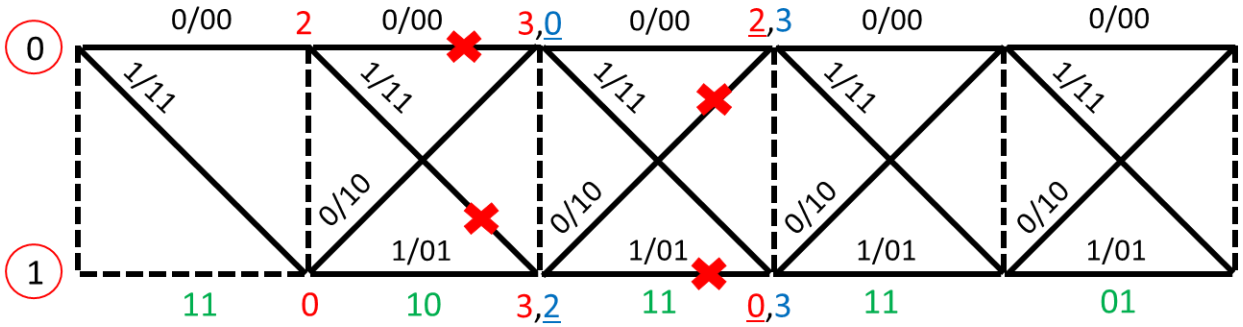
A técnica de decodificação consiste em acumular em cada nó da treliça as Distâncias de Hamming entre a saída v do codificador e a sequência r recebida a cada instante i.

Se mais de um caminho chega a um nó "mata-se" aqueles de maior **métrica** (maior distância acumulada) – caminhos marcados com **x** na Figura a seguir – ficando apenas aquele de menor métrica, denominado de **caminho sobrevivente**.

A métrica acumulada de cada caminho encontra-se em negrito à direita de cada nó, na Figura.

Métricas em vermelho representam ramos que incidem no nó "por cima" e métricas em azul representam ramos que incidem no nó "por baixo", já que, no máximo 2 ramos incidem em um nó para este decodificador.

Métricas sublinhadas representam métricas de caminhos sobreviventes.



Códigos Convolucionais – Decodificador de Viterbi

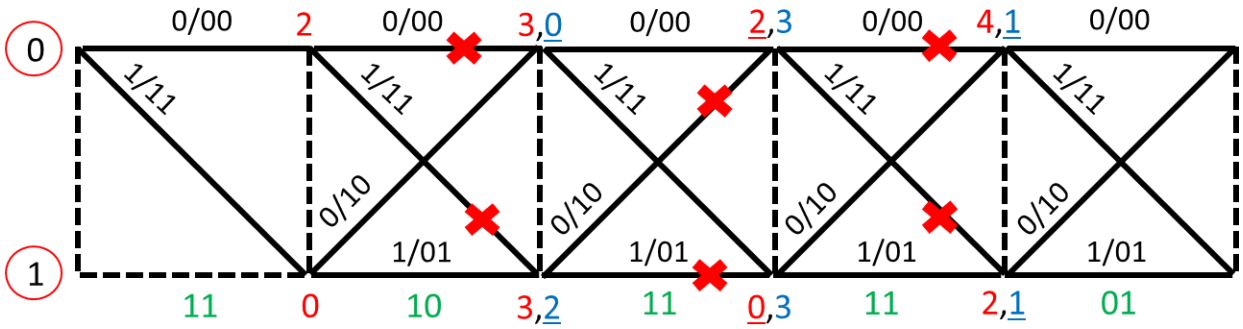
A técnica de decodificação consiste em acumular em cada nó da treliça as Distâncias de Hamming entre a saída v do codificador e a sequência r recebida a cada instante i.

Se mais de um caminho chega a um nó "mata-se" aqueles de maior **métrica** (maior distância acumulada) – caminhos marcados com **x** na Figura a seguir – ficando apenas aquele de menor métrica, denominado de **caminho sobrevivente**.

A métrica acumulada de cada caminho encontra-se em negrito à direita de cada nó, na Figura.

Métricas em vermelho representam ramos que incidem no nó "por cima" e métricas em azul representam ramos que incidem no nó "por baixo", já que, no máximo 2 ramos incidem em um nó para este decodificador.

Métricas sublinhadas representam métricas de caminhos sobreviventes.



Códigos Convolucionais – Decodificador de Viterbi

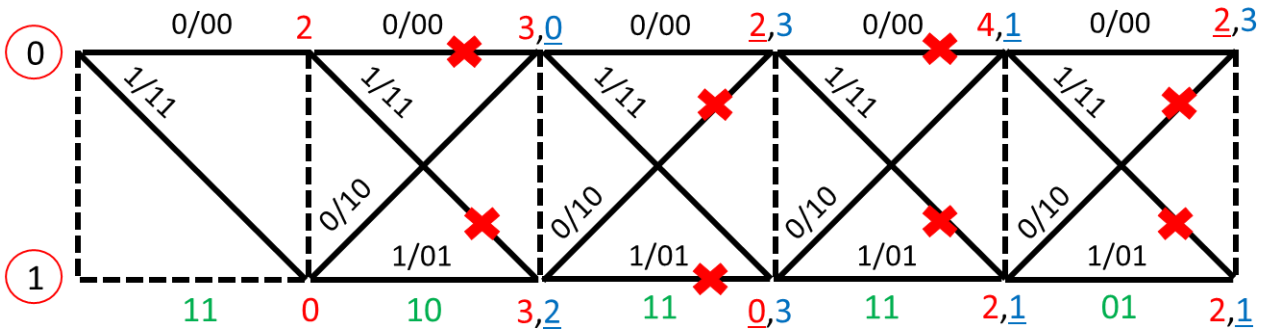
A técnica de decodificação consiste em acumular em cada nó da treliça as Distâncias de Hamming entre a saída v do codificador e a sequência r recebida a cada instante i.

Se mais de um caminho chega a um nó "mata-se" aqueles de maior **métrica** (maior distância acumulada) – caminhos marcados com **x** na Figura a seguir – ficando apenas aquele de menor métrica, denominado de **caminho sobrevivente**.

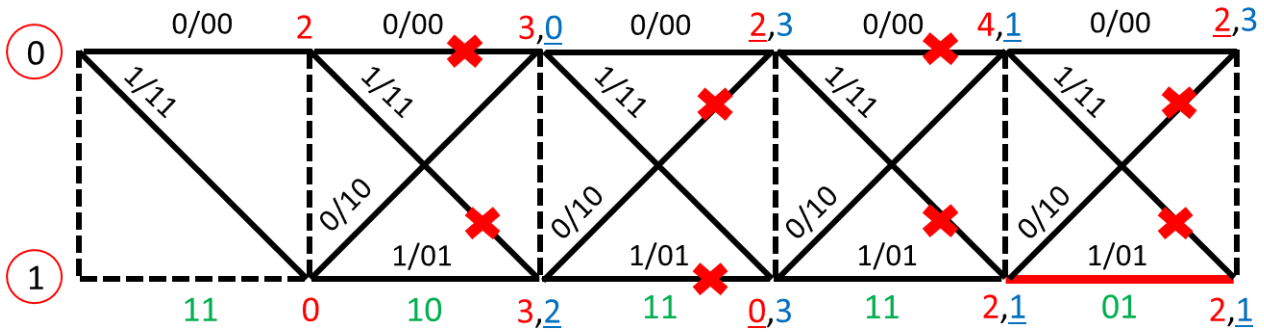
A métrica acumulada de cada caminho encontra-se em negrito à direita de cada nó, na Figura.

Métricas em vermelho representam ramos que incidem no nó "por cima" e métricas em azul representam ramos que incidem no nó "por baixo", já que, no máximo 2 ramos incidem em um nó para este decodificador.

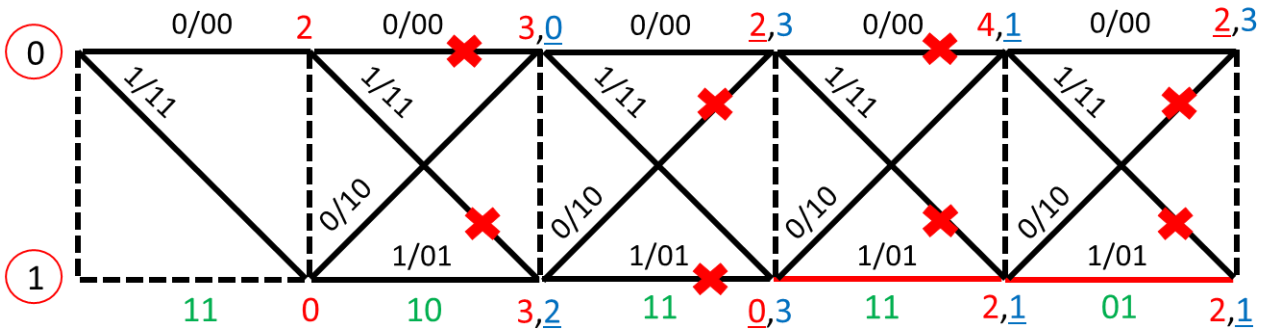
Métricas sublinhadas representam métricas de caminhos sobreviventes.



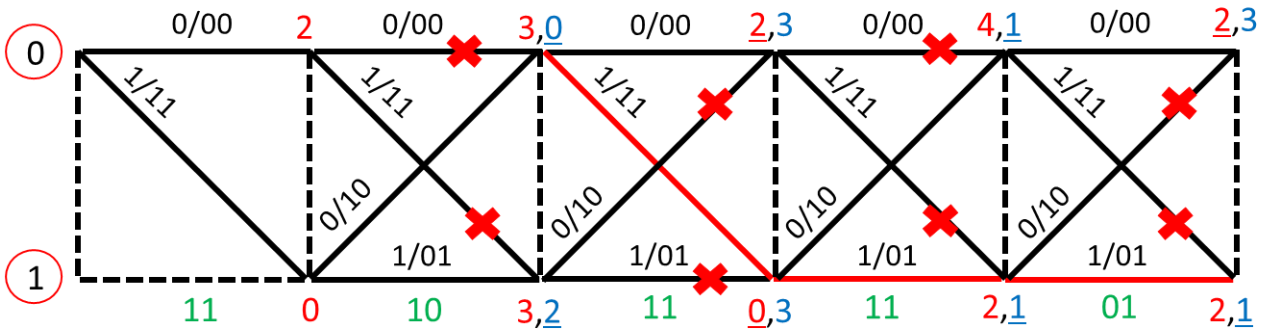
A decodificação final é iniciada a partir do caminho sobrevivente de menor métrica acumulada, identificando cada ramo sobrevivente da direita para a esquerda na treliça.



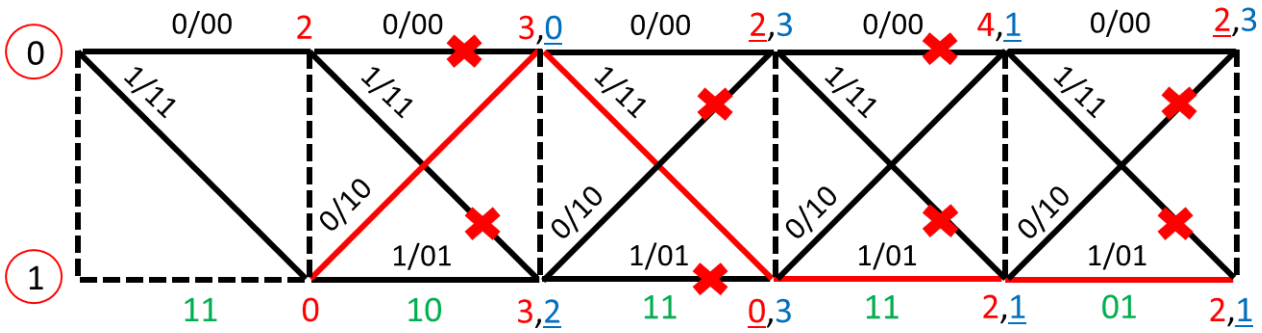
A decodificação final é iniciada a partir do caminho sobrevivente de menor métrica acumulada, identificando cada ramo sobrevivente da direita para a esquerda na treliça.



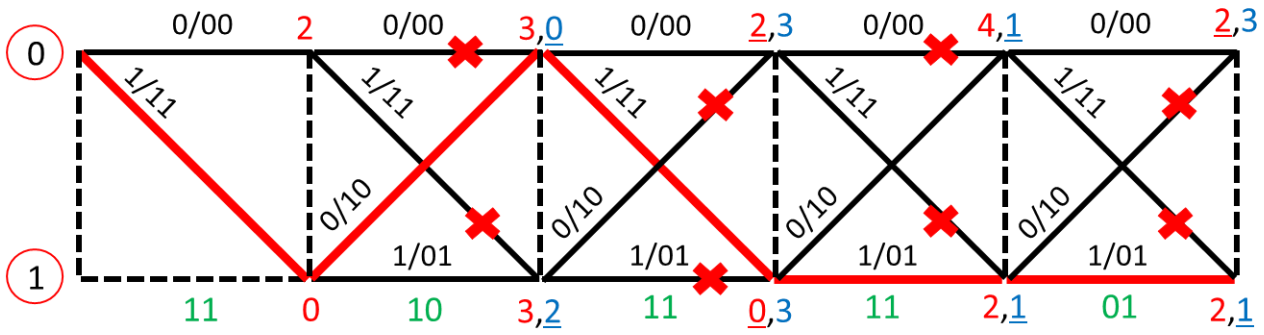
A decodificação final é iniciada a partir do caminho sobrevivente de menor métrica acumulada, identificando cada ramo sobrevivente da direita para a esquerda na treliça.



A decodificação final é iniciada a partir do caminho sobrevivente de menor métrica acumulada, identificando cada ramo sobrevivente da direita para a esquerda na treliça.



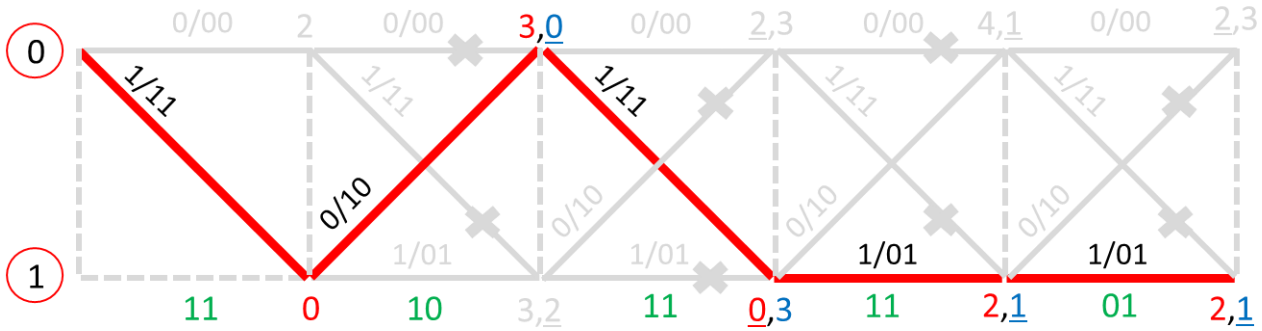
A decodificação final é iniciada a partir do caminho sobrevivente de menor métrica acumulada, identificando cada ramo sobrevivente da direita para a esquerda na treliça.



A decodificação final é iniciada a partir do caminho sobrevivente de menor métrica acumulada, identificando cada ramo sobrevivente da direita para a esquerda na treliça.

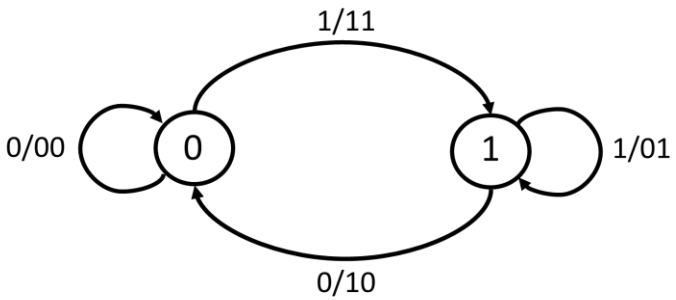
Ao lermos o valor de u nos identificadores u/v_0v_1 de cada ramo sobrevivente na Figura, verificamos que a sequência originalmente transmitida foi u = [10111], o que concorda com u originalmente transmitido.

Portanto, o decodificador identificou e corrigiu o erro ocorrido.



Códigos Convolucionais – Decodificador de Viterbi

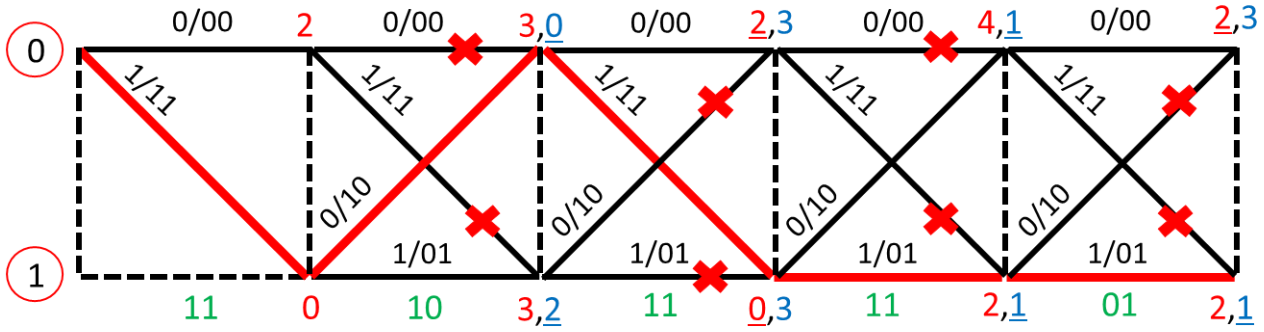
Diagrama de Transição de Estados



Exemplo de codificação para o codificador da Figura

u	1	0	1	1	1
D_{atual}	0	1	0	1	1
D_{futuro}	1	0	1	1	1
v	11	10	11	01	01
r	11	10	11	11	01

Diagrama de Treliça do Decodificador de Viterbi



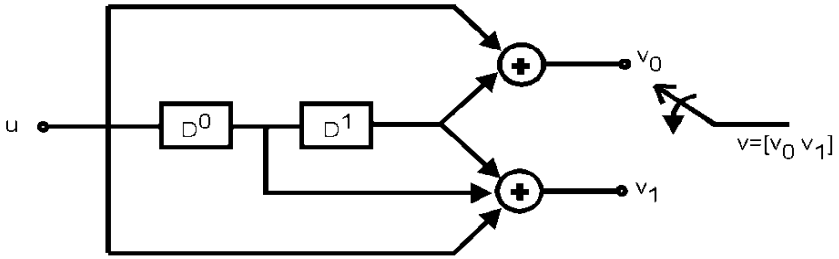
Nos links abaixo encontram-se disponíveis duas videoaulas c/ a revisão passo a passo da solução deste exemplo (Exemplo 1 – slide 58):

Videoaula Codificação de Canal - Codificador Convolucional - Profa. Cristina De Castro

Videoaula Codificação de Canal - Decodificador de Viterbi - Profa. Cristina De Castro

Exemplo 2:

A Figura abaixo apresenta o diagrama de transição de estados do codificador convolucional da Figura ao lado. Na Figura, cada círculo representa um estado D^0D^1 dentre os $2^K = 4$ possíveis estados.

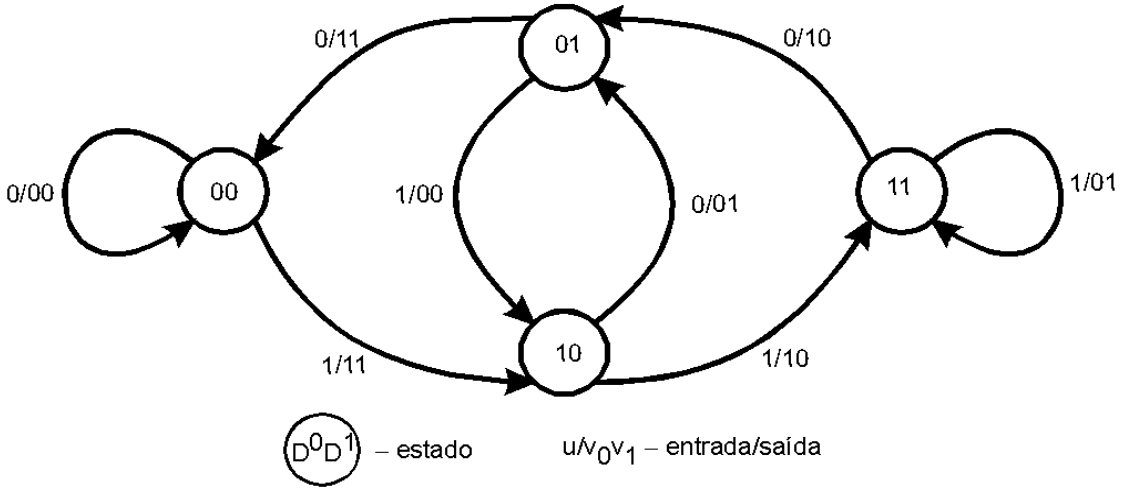


O diagrama é construído a partir dos estados individuais considerando as **<u>transições permitidas</u>** a partir de cada estado como consequência do valor lógico de u.

Por exemplo, suponhamos que a máquina de estado encontre-se no estado **10** (i.e., $D^0 = 1$ e $D^1 = 0$ na Figura acima).

$\Rightarrow$ Se $u = 1$ a saída resultante é $v = 10$ e, após a borda de descida do *clock*, a máquina vai para o estado **11**.

$\Rightarrow$ Se $u = 0$ a saída resultante é $v = 01$ e, após a borda de descida do *clock*, a máquina vai para o estado **01**.



Códigos Convolucionais – Decodificador de Viterbi

Dada uma seqüência u a ser codificada, a saída v no codificador de um transmissor digital é enviada ao receptor através do canal de transmissão, sendo recebida como uma sequência r .

Se nenhuma degradação de sinal ocorreu no canal de transmissão, r = v.

A Tabela a seguir mostra uma possível sequência u e a resultante sequência v para o codificador do Exemplo 2.

É mostrada também a trajetória do estado D^0D^1, à medida que u é codificada, partindo inicialmente do estado 00.

Assumindo que v seja enviado através de um canal de transmissão com ruído/interferência, a Tabela apresenta uma possível sequência r recebida com 2 erros.

Codificação					
u =	0	1	1	0	0
D^0D^1 atual =	00	00	10	11	01
D^0D^1 futuro =	00	10	11	01	00
v =	00	11	10	10	11
r =	0***1***	11	1***1***	10	11

Códigos Convolucionais – Decodificador de Viterbi

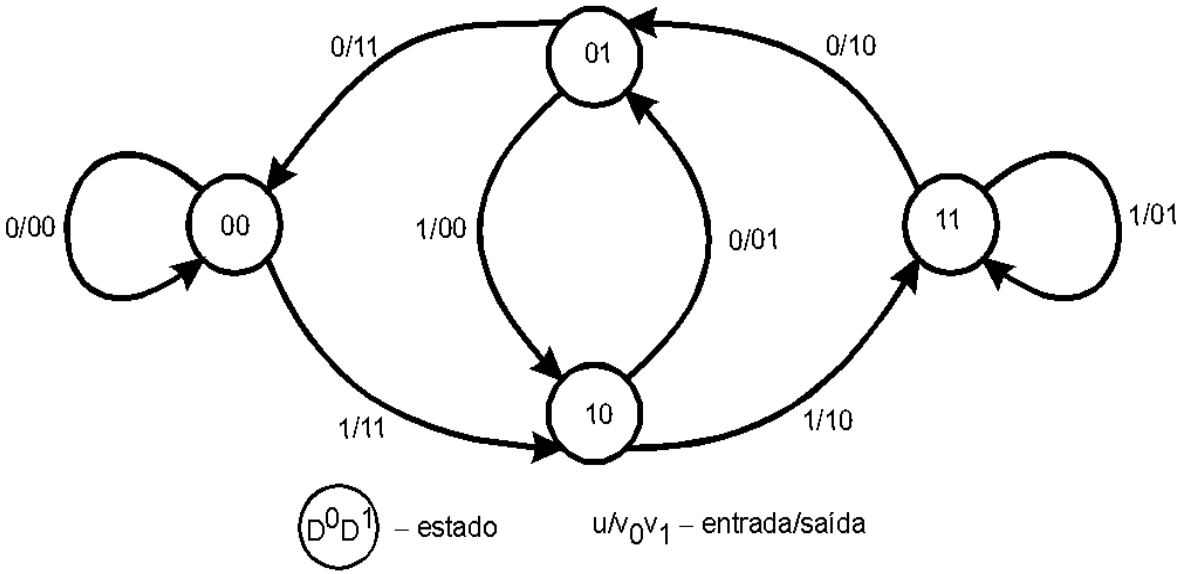


Diagrama de transição de estados do codificador convolucional.

No receptor digital, o decodificador utiliza um algoritmo de decodificação baseado no princípio de mínima distância (MLSE – *maximum likelihood sequence detector*) denominado **Algoritmo de Viterbi**. (Vamos decodificar a sequência r da Tabela através do Algoritmo de Viterbi para testar a capacidade de correção de erros do mesmo.)

A Figura a seguir apresenta o **diagrama de treliça** utilizado pelo Decodificador de Viterbi adequado a este codificador convolucional.

Códigos Convolucionais – Decodificador de Viterbi

⇒O diagrama de treliça mostra todas as trajetórias (caminhos) das transições de estado da máquina de estado do codificador a cada instante *i* de codificação, a partir do estado 00.

⇒Cada ramo da treliça começa e termina em um estado, representando, assim, uma transição permitida.

⇒Cada ramo é identificado por u/v_0v_1, isto é, a saída v do codificador quando, ao aplicarmos u em sua entrada, a máquina de estado executa a transição representada pelo ramo em questão.

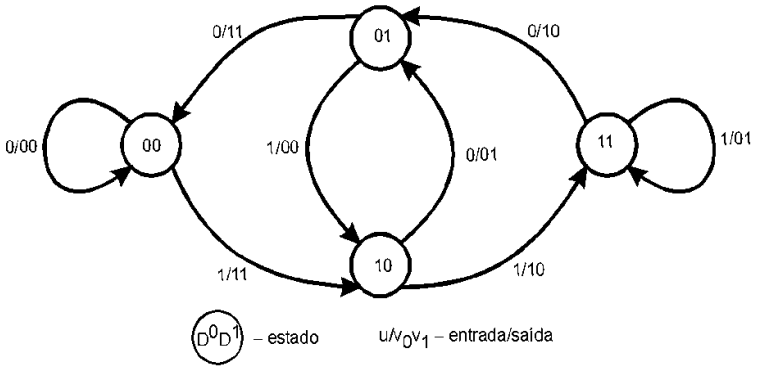


Diagrama de transição de estados do codificador convolucional.

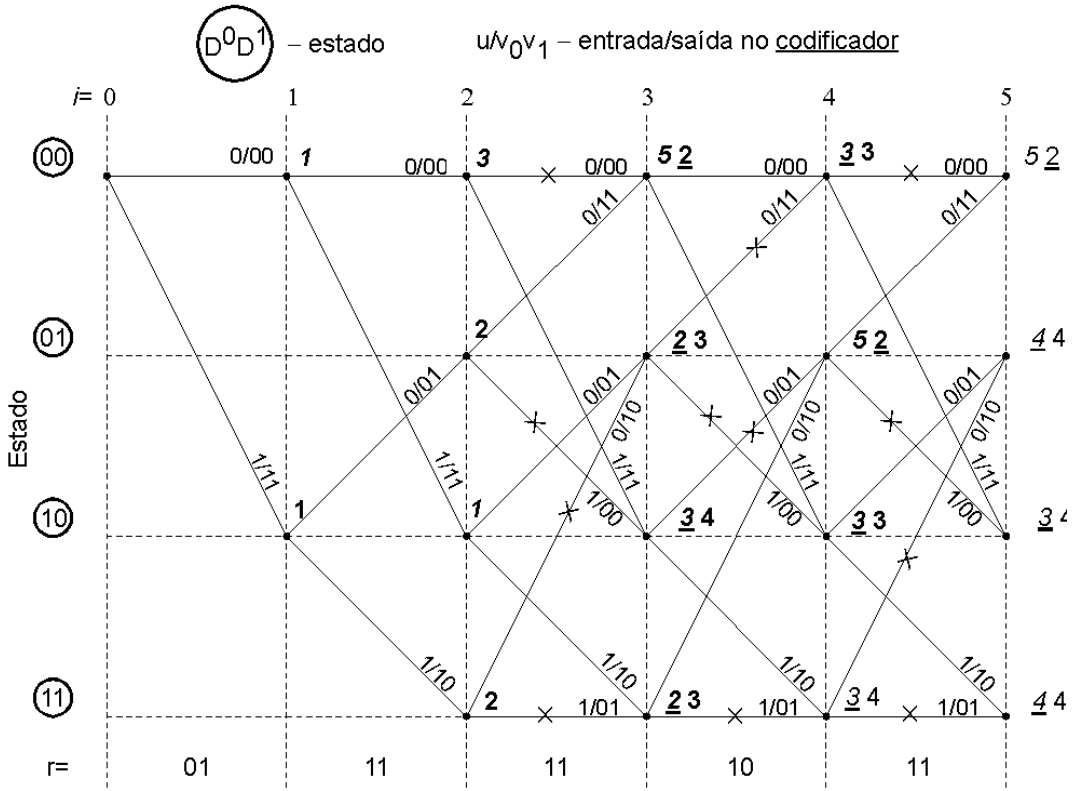


Diagrama de Treliça do Decodificador de Viterbi para o codificador convolucional.

Códigos Convolucionais – Decodificador de Viterbi

A **técnica de decodificação** consiste em acumular em cada nó da treliça as Distâncias de Hamming entre a saída v do codificador e a sequência r recebida a cada instante *i*.

Se mais de um caminho chega a um nó "mata-se" aqueles de maior **métrica** (maior distância acumulada) – caminhos marcados com **x** na Figura – ficando apenas aquele de menor métrica, denominado de **caminho sobrevivente**.

A métrica acumulada de cada caminho encontra-se em negrito à direita de cada nó na Figura.

Métricas sublinhadas representam métricas de caminhos sobreviventes.

Métricas em itálico representam ramos que incidem no nó "por cima" e métricas em não-itálico representam ramos que incidem no nó "por baixo", já que, no máximo 2 ramos incidem em um nó para este decodificador.

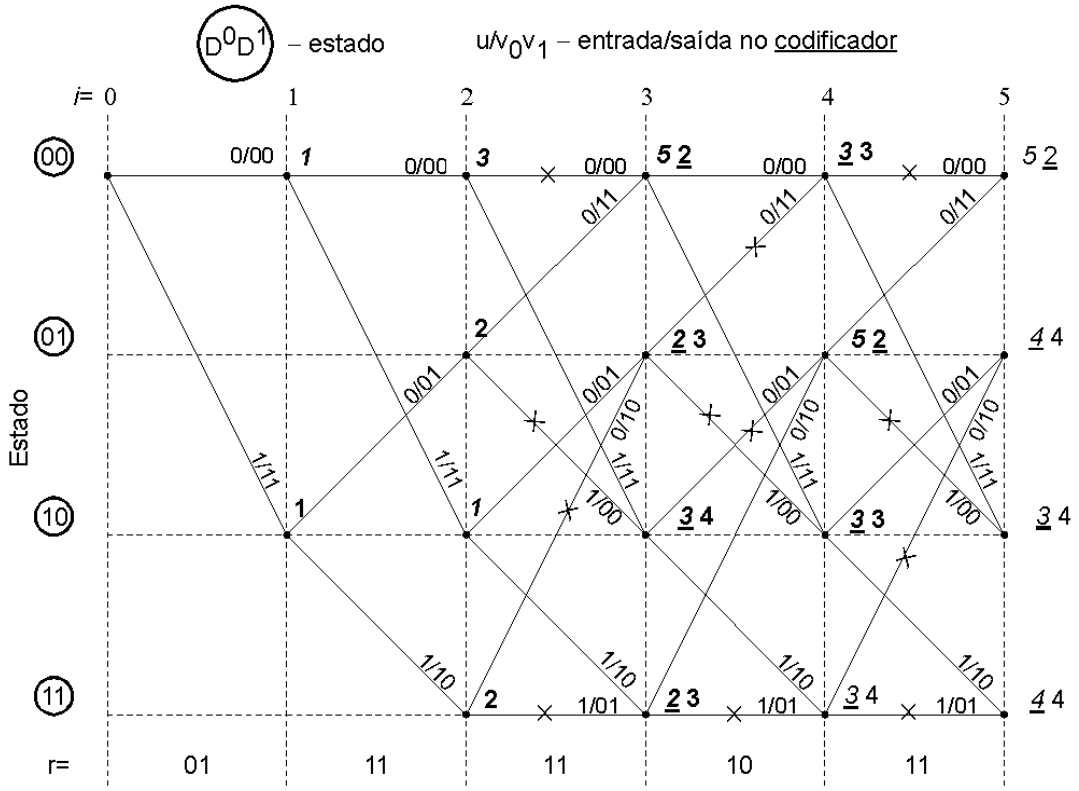


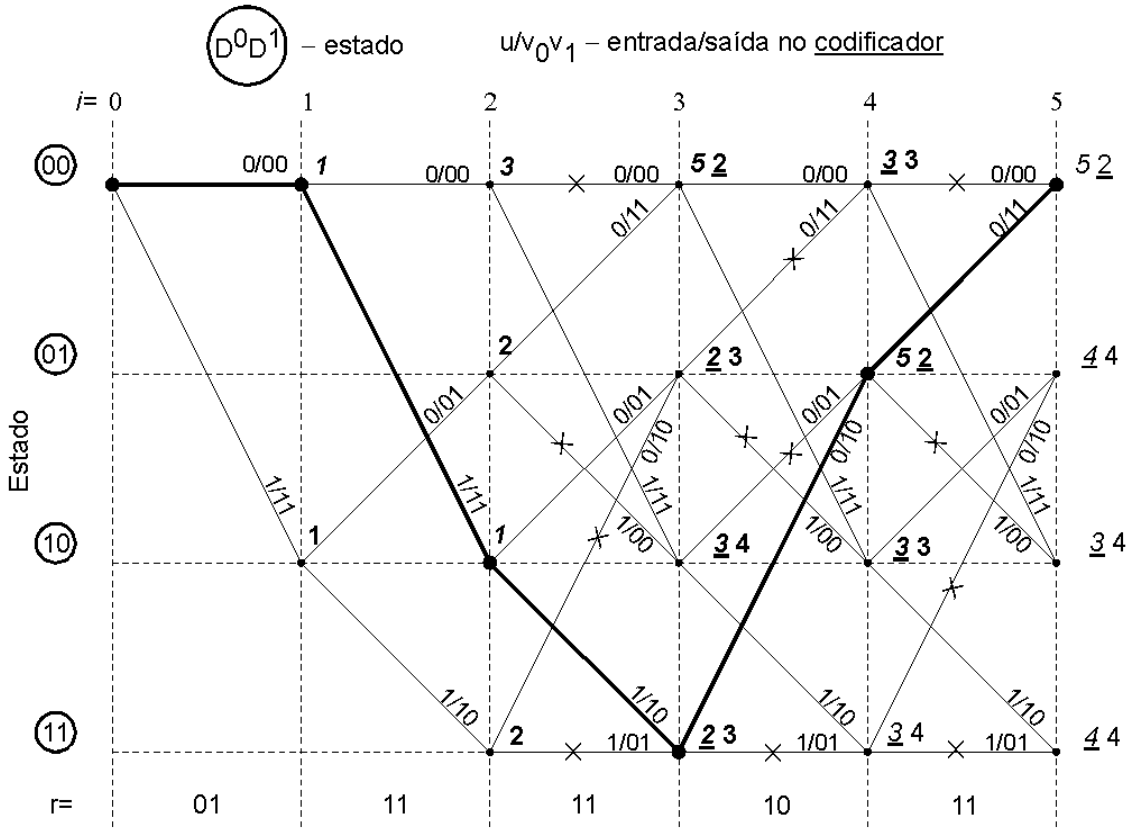
Diagrama de Treliça do Decodificador de Viterbi para o codificador convolucional.

Códigos Convolucionais – Decodificador de Viterbi

A decodificação final é iniciada a partir do caminho sobrevivente de menor métrica acumulada, identificando cada ramo sobrevivente da direita para a esquerda na treliça, conforma mostra a Figura.

Ao lermos o valor de u nos identificadores u/v_0v_1 de cada ramo sobrevivente na Figura, verificamos que a sequência originalmente transmitida foi $u=[01100]$, o que concorda com u mostrado na Tabela (efetivamente transmitido).

Portanto, o decodificador identificou e corrigiu os 2 erros.



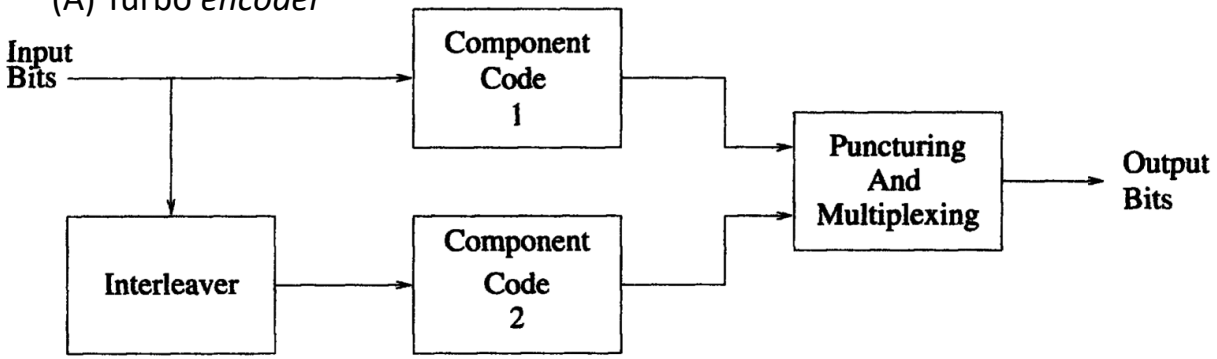
Decodificando a sequência r.

Neste link do GitHub https://github.com/topics/viterbi-algorithm?l=c estão disponibilizados códigos fonte em linguagem C no âmbito da implementação prática de códigos convolucionais e decodificadores de Viterbi. Neste link https://github.com/topics/viterbi-decoder?l=c%2B%2B estão disponibilizados códigos fonte em linguagem C++. E neste link https://github.com/topics/viterbi-algorithm?l=matlab estão disponibilizados scripts Matlab sobre o mesmo tema.

Turbo Códigos

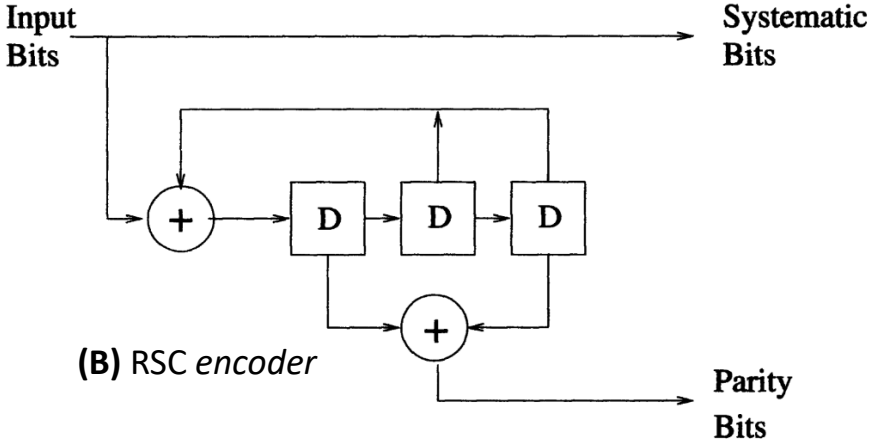
Turbo Códigos (TCs) foram propostos em 1993 por Berrou, Glavieux e Thitimajashima, que relataram excelentes resultados de ganho de codificação, aproximando-se do Limite de Shannon (ver https://en.wikipedia.org/wiki/Turbo_code). A sequência de informação é codificada duas vezes, com um *interleaver* (ver https://en.wikipedia.org/wiki/Burst_error-correcting_code#Interleaved_codes) entre os dois codificadores servindo para tornar as duas sequências de dados codificadas aproximadamente estatisticamente independentes uma da outra, conforme mostrado em (A) abaixo.

(A) Turbo *encoder*



O excelente desempenho de TCs é devido à ação do *interleaver*, que, indiretamente "aumenta a duração da sequência de bits de paridade" aumentando a capacidade de correção (quanto maior o comprimento do *interleaver*, melhor o desempenho do TC).

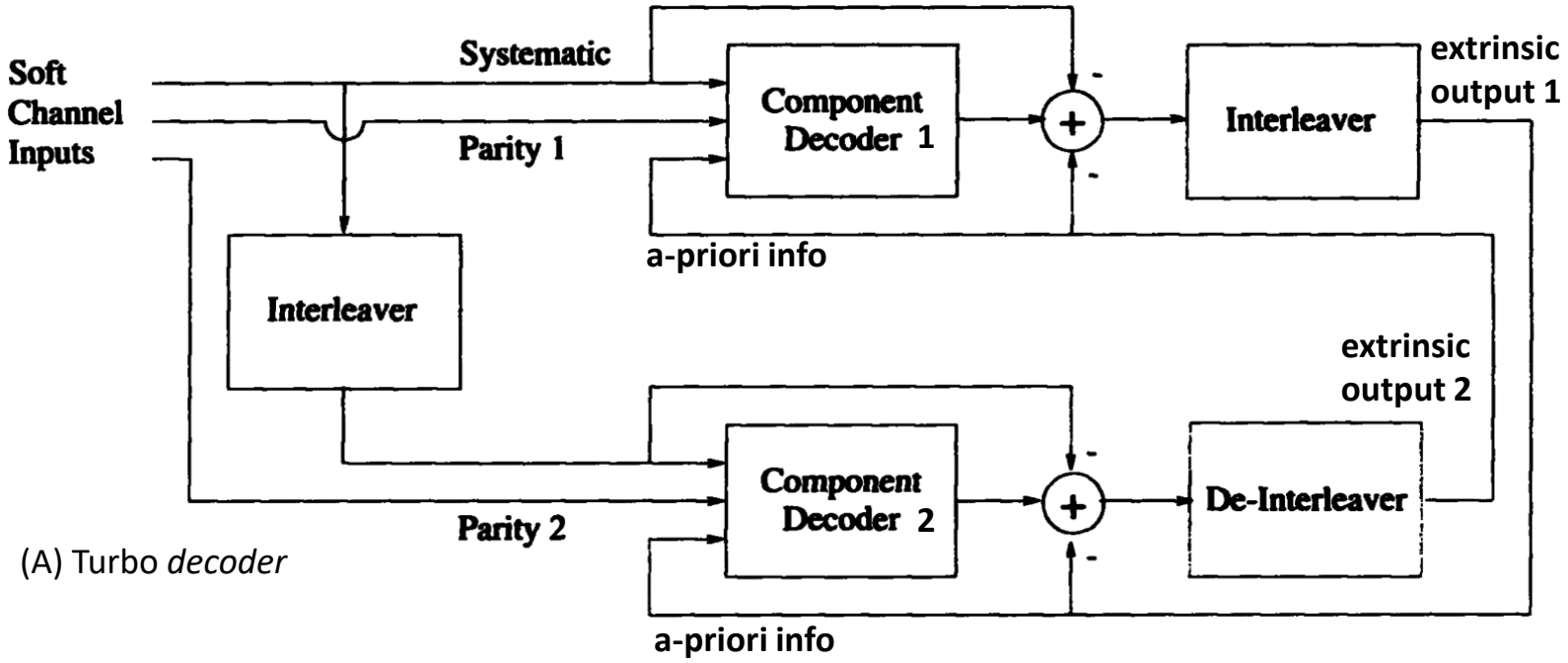
Usualmente o turbo *encoder* (*encoder* = codificador) utiliza codificadores convolucionais sistemáticos recursivos (RSC – *Recursive Systematic Convolutional*) com razão de codificação ½ (ver slide 9), i.e., para cada bit de entrada na entrada do codificador RSC resultam dois bits de saída, conforme mostrado em (B). Cada codificador RSC ("Component Code 1" e "Component Code 2" em (A) acima) produz em sua saída um *stream* de bits sistemáticos que é equivalente ao *stream* de bits de informação em sua entrada e seguidos dos bits de paridade acrescidos pelo codificador. *Puncturing* é então aplicado aos bits de paridade em cada um dos dois *streams* (ver https://en.wikipedia.org/wiki/Punctured_code) de modo a aumentar a razão de codificação de cada codificador. Os dois *streams* são então multiplexados e transmitidos.



(B) RSC *encoder*

Turbo Códigos

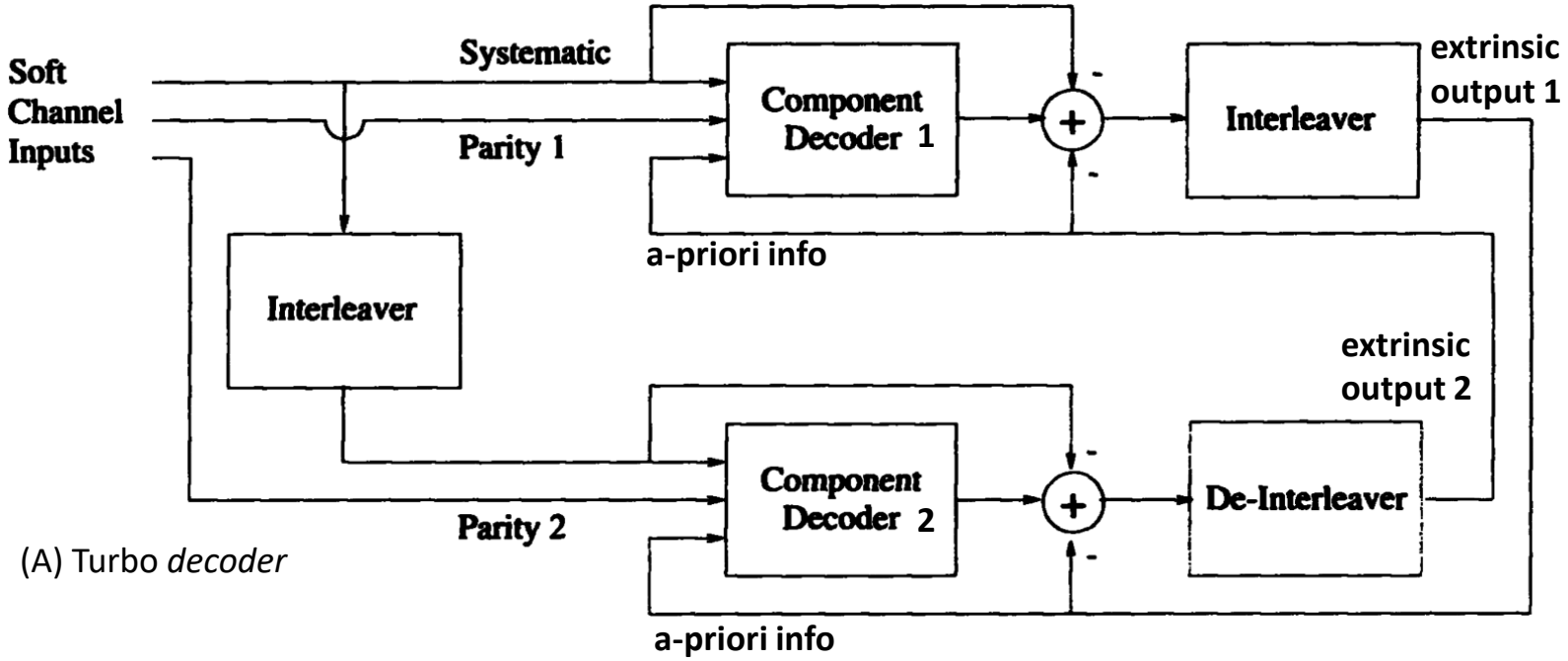
No decodificador são usados dois *decoders* RSC ("Component Decoder" conforme mostrado em (A)). O *stream* de entrada de cada *decoder* não são os bits em si, mas sim a probabilidade de ocorrência do valor do bit, representação denominada de ***soft input***. Mesmo acontece para a saída do decoder, denominada de ***soft output*** . O valor em ponto flutuante de um ***soft input*** (ou de um ***soft output***) fornece não apenas uma indicação se um determinado bit é 0 ou 1, mas também uma indicação que dá a probabilidade (verossimilhança) de que o bit tenha sido decodificado corretamente. O decodificador turbo opera iterativamente. Usualmente cada decodificador é um decodificador de Viterbi adaptado para *soft inputs* e *soft outputs* (ver https://en.wikipedia.org/wiki/Viterbi_algorithm#:~:text=Soft%20output%20Viterbi%20algorithm,-The%20soft%20output&text=SOVA). Na 1ª iteração, o 1º decodificador RSC resulta em uma sequência de s*oft outputs*, fornecendo uma estimativa da sequência de bits originalmente transmitidos baseado unicamente nas *soft inputs* provenientes do canal. Ele também gera a saída extrínseca "extrinsic output 1". A saída extrínseca para um determinado bit não é baseada na entrada do canal para aquele bit, mas nas informações dos bits próximos na sequência e nos condicionantes impostos pelo código que está sendo usado. Esta saída extrínseca do 1º decodificador é usada pelo 2º decodificador RSC como informação a-priori, e esta informação junto com os *soft inputs* do canal são usados pelo 2º decodificador RSC para gerar a sua *soft output* e a "extrinsic output 2".



(A) Turbo *decoder*

Turbo Códigos

Na 2ª iteração, a informação extrínseca do 2º decodificador obtida na 1ª iteração é usada como a informação a-priori para o 1º decodificador, e usando esta informação a-priori, o 1º decodificador decodifica mais bits corretamente do que na 1ª iteração. Este ciclo continua, sendo que em cada iteração ambos os decodificadores RSC produzem um s*oft output* e uma "extrinsic output" baseadas no *soft input* do canal e baseada na informação a priori obtida da "extrinsic output" do decodificador anterior. Após cada iteração a taxa de erro de bits (BER) na sequência decodificada reduz, mas a taxa de redução na BER vai diminuindo a medida que o número de iterações aumenta de modo que, por razões de complexidade, geralmente apenas entre 4 e 12 iterações são usadas. Uma descrição detalhada da operação do decodificador mostrado em (A) é encontrada na seção "5.3 Turbo Decoder" na página 110 de https://www.fccdecastro.com.br/pdf/TCTESTC.pdf .



(A) Turbo *decoder*

APÊNDICE A – Grafo de Tanner, matriz de Paridade H e síndrome $\underline{s_i}$

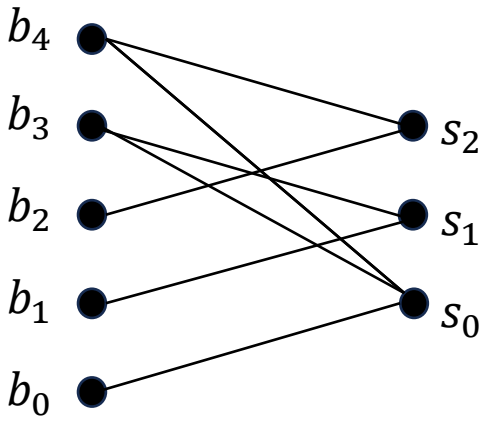
$$\underline{c_i}\mathbf{H}^T = \underline{s_i}$$

$$[b_4 \quad b_3 \quad b_2 \quad b_1 \quad b_0]\begin{bmatrix}1 & 0 & 1\\ 0 & 1 & 1\\ 1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix} = [s_2 \quad s_1 \quad s_0]$$

$$b_4 + b_2 = s_2$$

$$b_3 + b_1 = s_1$$

$$b_4 + b_3 + b_0 = s_0$$



Grafo de Tanner resultante de **H**