2. Códigos de blocos

2.1. O que é um código de blocos?

Vamos supor que temos uma mensagem que queremos guardar em disco ou em outro meio, para uso posterior, ou que queremos transmitir à distância através de um canal de comunicações ruidoso.

Já vimos que nesses casos é conveniente introduzir redundância na nossa mensagem se queremos que chegue ao destino ou que seja reproduzida com fidelidade suficiente, ou seja, com uma probabilidade de erro pequena. Já sabemos que existem diversas maneiras de introduzir essa redundância de um modo eficiente: ou se usam códigos de blocos, ou cíclicos, ou convolucionais. Vamos começar pelos códigos de blocos.

Suponhamos então que queremos codificar uma mensagem constituída por uma sequência de símbolos extraídos de um alfabeto com q elementos (se o código for binário $q = 2$). O que o codificador faz, em primeiro lugar, é "partir", ou seccionar, a mensagem em "blocos" com um certo número, k, de símbolos e a cada um desses blocos vai acrescentar alguns símbolos extra, redundantes, chamados *símbolos de verificação de paridade*, ou simplesmente *símbolos de paridade*, de modo que os blocos ficam mais longos, com n símbolos. A estes blocos chamaremos *palavras de código*. A Fig. 2.1 mostra o que acontece.

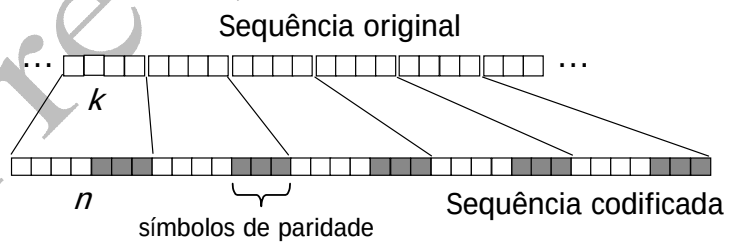


Fig. 2.1 Acrescentando símbolos de paridade à sequência de informação.

A "arte" da codificação está na escolha adequada dos símbolos de paridade: não podem ser uns símbolos quaisquer. Na verdade, o desempenho do código perante a existência de erros e a necessidade de os detectar ou corrigir depende da qualidade da redundância que for introduzida na mensagem.

O número de símbolos de informação, k, é a *dimensão* do código e o número de símbolos da palavra, n, é o seu *comprimento*. Diz-se então que temos um código (n,k) com uma *taxa de código* $R_c = k/n$.

Como o número de símbolos de informação é k, então o código tem q^k palavras diferentes; tratando-se de um código binário o número de palavras de código é 2^k.

Vamos necessitar de usar vectores e matrizes quando tratamos de códigos de blocos. Assim, para começar, associemos o bloco de k símbolos de informação a um vector **X** de k símbolos e, de igual modo, associemos cada palavra de código a um vector **Y** de n símbolos:

$$\boldsymbol{X} = \begin{bmatrix} x_1 & x_2 & \dots & x_k \end{bmatrix}$$

$$\boldsymbol{Y} = \begin{bmatrix} y_1 & y_2 & \dots & y_n \end{bmatrix}$$

Suponhamos que os símbolos de paridade, c_i, ficam todos juntos, no início ou no fim de cada palavra de código, isto é,

$$\boldsymbol{Y} = \begin{bmatrix} x_1 & x_2 & \dots & x_k & c_1 & \dots & c_{n-k} \end{bmatrix} = \begin{bmatrix} X & | & C \end{bmatrix}$$

ou

$$\boldsymbol{Y} = \begin{bmatrix} c_1 & \dots & c_{n-k} & x_1 & x_2 & \dots & x_k \end{bmatrix} = \begin{bmatrix} C & | & X \end{bmatrix}.$$

A um código com esta localização de símbolos de paridade chama-se *código sistemático*.

Vamos considerar exclusivamente códigos sistemáticos. De momento vamos limitar-nos também aos códigos binários, e com uma condição adicional: eles terão de ser *lineares*, querendo com isto dizer-se que devem satisfazer as seguintes duas condições:

1. *A palavra de código nula pertence ao conjunto de palavras de código.*

2. *A soma de duas palavras de código é também uma palavra de código.*

Esta soma é feita em aritmética binária, bit a bit e sem transporte, sendo as regras de adição as seguintes:

+	0	1
0	0	1
1	1	0

Trata-se da adição em módulo 2, ou operação OU exclusivo. Assim, se tivermos dois vectores **Z** e **Y**, o vector-soma é calculado somando elemento a elemento,

$$\boldsymbol{Z}+\boldsymbol{Y}=\begin{bmatrix} z_1+y_1 & z_2+y_2 & \cdots & z_i+y_i & \cdots & z_n+y_n \end{bmatrix} \qquad y_i, z_i = 0,1.$$

Como se vê, tanto faz somar como subtrair (por exemplo, $1-1=1+1=0$).

Exemplo 2.1

Um código sistemático linear (5,3) tem 8 palavras de código. Vamos determiná-las partindo da seguinte tabela incompleta:

Mensagens	**Palavras de código**
000	00000
100	10011
010	01010
001	00101
110	?
101	?
011	?
111	?

Como a soma de duas palavras de código é também uma palavra de código então a 2ª e a 3ª palavras vão originar a 5ª:

$$\begin{array}{rr} & 10011 \\ + & 01010 \\ \hline & 11001 \end{array}$$

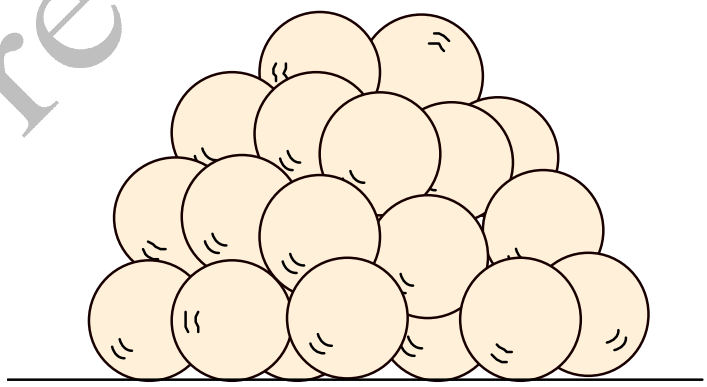
As três palavras restantes são determinadas da mesma maneira:

2ª + 4ª palavras	3ª + 4ª palavras	2ª + 3ª + 4ª palavras
10011	01010	10011
+ 00101	+ 00101	01010
10110	01111	+ 00101
		11100

Repare-se que no exemplo anterior calculámos as palavras de código à custa das $k = 3$ palavras cuja componente de mensagem apenas tem um bit "1" ([10011], [01010] e [00101]). Estas palavras são linearmente independentes, significando que nenhuma delas pode ser obtida por combinação linear das outras duas. Poderemos fazer sempre assim em qualquer código linear: as restantes $2^k - (k+1)$ palavras de código (excluindo portanto a palavra nula) são obtidas por combinação linear de k palavras linearmente independentes.

2.2. *Pesos, distâncias e esferas*

Esferas?! Parece estranho este título. O que é que terão pesos e distâncias, e esferas, a ver com códigos? Ora bem, nem estes pesos se medem em newtons nem as distâncias se medem em metros. Na verdade, aqui os significados são outros, mais simples. Mas esferas são esferas…



2.2.1. Pesos e distâncias

Os nossos pesos e distâncias têm as seguintes definições:

- *Peso do vector X, w(X): é o número de elementos não nulos do vector.*

- *Distância de Hamming, d(X,Y), entre dois vectores: é o número de elementos diferentes.*

Por exemplo, o peso de $X = [1011101]$ é $w(X) = 5$ e a distância de Hamming entre **X** e o vector $\boldsymbol{Y} = [1100101]$ é $d(X,Y) = 3$.

Repare-se que o vector-soma de **X** e **Y**, $\boldsymbol{Z} = \boldsymbol{X} + \boldsymbol{Y} = [0111000]$, tem um peso igual à distância 3, isto é,

$$w(Z) = w(X + Y) = 3 = d(X,Y)$$

De facto,

a distância de Hamming entre dois vectores é igual ao peso do seu vector-soma.

Um código binário *(n, k)* tem 2^k palavras de *n* bits. Estas não estão todas à mesma distância de Hamming umas das outras. A menor dessas distâncias, chamada *distância mínima* do código, tem muita importância pois ajuda a estabelecer as capacidades de correcção e detecção de erros do código, como veremos. Temos, portanto:

- *Distância mínima de um código, d_{min}: é a menor distância de Hamming entre dois vectores válidos do código.*

A distância de Hamming entre dois vectores X e Y de código é igual ao peso do vector-soma, que pertence ao código por este ser linear. Logo, o conjunto de distâncias de Hamming entre cada dois vectores de código é igual ao conjunto dos pesos de todos eles. De todos os vectores *não-nulos* um apresenta o menor peso. Esse peso é, portanto, a distância mínima, d_{min}, do código. Então, alternativamente, podemos dizer que

a distância mínima de um código é igual ao menor peso não nulo de todas as suas palavras de código.

Exemplo 2.2

Um código (4,2) tem 4 palavras de código,

{0000, 1010, 0111, 1101}

Os pesos de cada palavra são, respectivamente, 0, 2, 3 e 3. O menor peso não nulo é 2 e por isso a distância mínima do código é $d_{\min} = 2$: cada palavra difere das outras em pelo menos dois bits.

Representemos cada um dos 2^n vectores de n bits por um ponto[1]. Ao conjunto de todos esses pontos chama-se "espaço de sinal". Desse conjunto apenas 2^k são palavras de um código *(n,k)*, isto é, a maioria dos pontos do espaço de sinal não representa vectores de código. A situação é ilustrada na Fig. 2.2, onde estão representados cinco vectores de código (os pontos "maiores"), cada um envolto numa "nuvem" de pontos que lhe estão mais próximos, nuvem a que iremos chamar *região de decisão*.

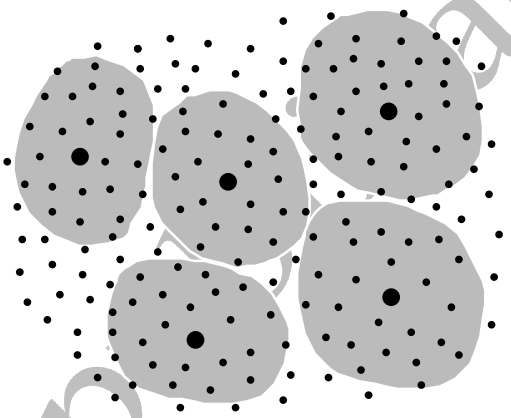


Fig. 2.2 Constelação de pontos representando vectores de *n* bits.

Admitamos, por exemplo, que *n=7* e *k=4*. Então existem 16 pontos de código num universo de 128. Sejam os seguintes:

0000000	1000110
0001111	1001001
0010011	1010101
0011100	1011010
0100101	1100011
0101010	1101100
0110110	1110000
0111001	1111111

[1]Esta representação deveria ser feita no espaço de *n* dimensões. Infelizmente não é possível visualizarmos esse espaço quando *n>3*.

Este código, em particular, pertence à família dos código de Hamming, que iremos encontrar mais tarde. Não há nenhuma palavra de código, ou ponto, com peso inferior a 3, logo a separação mínima entre palavras válidas é $d_{min}=3$. Mas existem palavras não válidas que estão a distâncias de Hamming inferiores a 3. Pergunta-se: quais são os pontos que estão à distância de Hamming unitária de um qualquer ponto? A resposta é simples: serão aqueles cujos vectores somados ao vector do ponto resultam num vector de peso unitário. Então sendo $n=7$ existem 7 pontos nessas condições. Do mesmo modo existirão $\binom{7}{2}=21$ à distância de 2 unidades, e assim por diante.

Se se tratar de um código de repetição – no qual $k=1$ e os bits de paridade são iguais ao bit de mensagem – só teremos duas palavras de código. Por exemplo, no código de repetição (6,1) as palavras de código são [000000] e [111111]; existirão então $\binom{6}{1}=6$ pontos à distância de 1 unidade de cada vector de código, e $\binom{6}{2}=15$ pontos à distância de 2. À distância de 6 só existirá $\binom{6}{6}=1$ ponto, que é o outro ponto do código.

A Fig. 2.3 ilustra o que se passa com dois pontos, A e B, do código de Hamming (7,4) e com os dois pontos do código de repetição (6,1), estando os pontos não válidos situados na superfície de esferas concêntricas à volta dos primeiros[2].

Tomemos o código (7,4) e imaginemos que a palavra **Y**=[1011010] corresponde ao ponto A e foi enviada através de um canal ruidoso; admitamos que por causa do ruído não foi **Y** que foi recebida no descodificador mas sim a palavra **Z**=[1011000]. Portanto, o penúltimo bit a contar da esquerda está errado. Então é como se ao vector original **Y** tivesse sido adicionado um vector **E**=[0000010] de modo que $\boldsymbol{Z}=\boldsymbol{Y}+\boldsymbol{E}$. A um vector como **E**, onde os bits "1" indicam os bits errados, vamos chamar *padrão de erro*, ou *vector de erro*. Com um padrão de 1 erro, como este **E**, a palavra original não é transformada noutra palavra do código; basta verificar que **Z**, que pode ser o ponto α (por só diferir de **Y** em um bit), não consta da lista das dezasseis palavras de código apresentadas atrás. Com padrões de dois erros também não há transformação em palavra válida

(por exemplo, se $\boldsymbol{E}$*=[0101000] então* $\boldsymbol{Z}$*=[1110010] (ponto* β*)),*

[2]Na figura não é possível representar esferas. Optou-se por circunferências como aproximação.

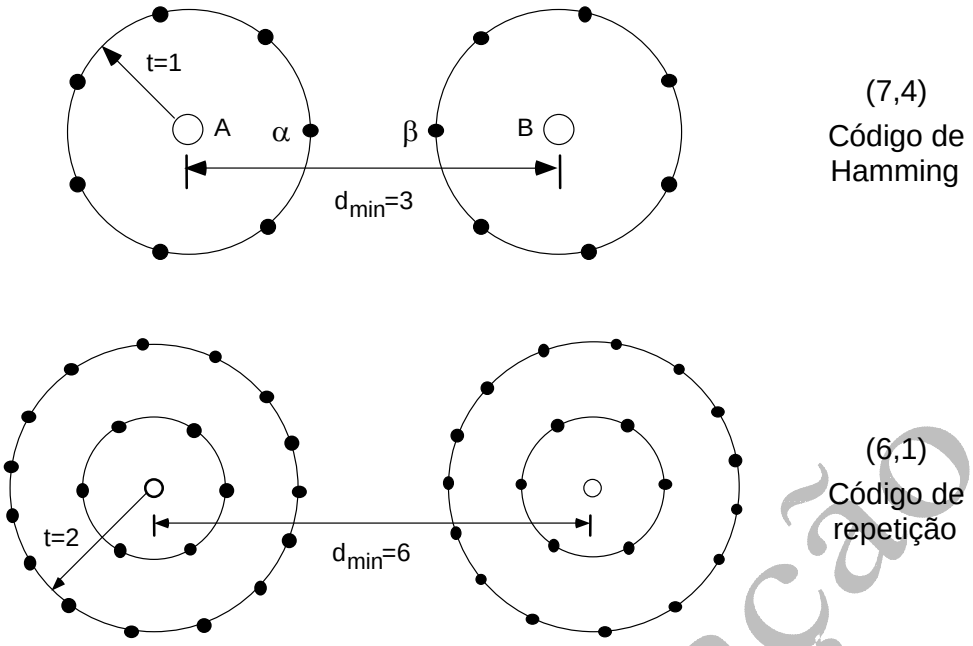


Fig. 2.3 Distâncias entre palavras de código.

mas se houver três erros a palavra pode ser convertida numa outra palavra válida do código, como B, porque $d_{min} = 3$

(por exemplo, se $\boldsymbol{E}$*=[0101010] então* $\boldsymbol{Z}$*=[1110000] (ponto B)).*

Neste caso o descodificador não tem meio de saber que a palavra que está a receber está errada, visto ser válida, de facto, e o erro passa despercebido: cometeu-se um *erro de detecção*. Portanto:

> *Um erro não detectado ocorre quando uma palavra válida do código é transformada pelo canal noutra palavra igualmente válida.*

Concluindo: o descodificador de um código (7,4) detecta erros numa palavra recebida se um ou dois dos 7 bits estiverem errados, isto é, são detectados até $l = 2$ erros por palavra. Diremos que este código é um *código detector de erros duplos*.

Podemos fazer melhor que apenas detectar erros: podemos mesmo corrigir alguns. Por exemplo, se tivermos enviado o ponto A e a palavra recebida for α (o que quer dizer que um dos sete bits está errado), o descodificador pode estimar que a palavra enviada terá sido A porque é a palavra válida mais próxima de α. Mas se tiver havido dois erros a correcção é incorrecta pois, como agora o ponto recebido (seja β) está mais próximo do ponto válido da direita, B, que não foi enviado, o descodificador escolhe este — assim, cometeu-se um *erro de descodificação*. Dizemos então que é

corrigido um erro por palavra ($t = 1$), ou que se trata de um *código corrector de erros simples*.

Voltando à Fig. 2.2, digamos que um ponto inválido é correctamente corrigido se pertencer à região de decisão do ponto que lhe terá dado origem. Se pertencer a outra região de decisão comete-se um erro de descodificação.

O segundo exemplo da Fig. 2.3 refere-se ao código de repetição (6,1). Como antes, o descodificador está preparado para uma de três situações: 1) só detecta erros (e não os corrige); 2) só corrige erros; 3) corrige e detecta erros. No primeiro caso, o descodificador detecta palavras erradas se tiverem 5 ou menos erros; no segundo caso corrige-as correctamente se tiverem 1 ou 2 erros; no terceiro caso, se houver até dois erros corrige-os mas se houver 3 erros detecta-os simplesmente, sem os corrigir (porque já não consegue fazê-lo sem errar).

Exemplo 2.3

Quais são as capacidades de detecção e correcção de erros de um código de repetição (4,1)?

Existem 16 palavras de 4 bits e só duas delas são palavras do código: [0000] e [1111]. Algumas das restantes 14 estão mais próximas de [0000] do que de [1111], e vice-versa. Um critério de detecção e correcção pode então passar pela consideração de duas regiões de decisão, cada uma associada a uma palavra válida. A região de decisão associada a [0000] é formada pelo conjunto de palavras de 4 bits de peso 0 ou 1

$$\Phi = \{0000, 1000, 0100, 0001\}$$

enquanto que a região de decisão associada a [1111] é formada pelo conjunto de palavras de peso 3 ou 4

$$\Psi = \{1111, 1110, 1101, 1011\}$$

Se a palavra recebida contiver apenas um bit errado então pertence a um destes conjuntos. Se pertencer ao primeiro o descodificador escolhe [0000] como a palavra válida original. Se pertencer ao segundo escolhe [1111]. E se não pertencer a nenhum dos conjuntos? Ora bem:

- se o descodificador apenas tentar detectar erros, sem os corrigir, consegue detectar todas as palavras com 1, 2 ou 3 erros (por exemplo, enviou-se [0000] e recebeu-se [1101]).

  Sendo assim, o código detecta erros triplos, isto é, $l = 3$.

- se o descodificador corrigir sempre os erros (bem ou mal), a correcção só será bem feita se não houver mais de 1 erro (isto é, se a palavra recebida pertencer a uma das tais regiões de decisão); se houver dois erros a correcção pode estar errada:

  (se se receber 0011 qual foi a palavra enviada?)

  se houver três, a correcção está mesmo errada!

  (se se tiver enviado 0000 e se tiver recebido 0111
  o descodificador escolhe 1111)

  Sendo assim, o código corrige erros simples, isto é, *t=1*.

- se o descodificador corrigir e detectar erros então: se houver 1 erro corrige-o, mas se houver 2 erros não os corrige (pois talvez os corrigisse mal) mas detecta-os.

  1) Enviou-se 0000 e recebeu-se 0001 (há um erro). Como 0001 faz parte do conjunto Φ o erro é corrigido.

  2) Enviou-se 0000 e recebeu-se 0011 (há dois erros). Como 0011 não faz parte do conjunto Φ os erros não são corrigidos mas foram detectados.

  Sendo assim, o código corrige erros simples e também detecta erros duplos, isto é, $t = 1$ e $l = 2$.

Podemos então resumir as considerações anteriores nas seguintes relações:

$$d_{\min} \geq l + 1 \qquad \text{(só detecção)}$$

$$d_{\min} \geq 2t + 1 \qquad \text{(só correcção)}$$

$$d_{\min} \geq t + l + 1 \qquad l > t \qquad \text{(detecção e correcção)}$$

2.2.2. Esferas…

O espaço de sinal pode ser visualizado a três dimensões se o comprimento do código for 3. Se, por exemplo, ao bit "0" corresponder um impulso de amplitude 0 e ao bit "1" corresponder um impulso de amplitude A, as duas palavras [000] e [111] de um código (3,1) correspondem respectivamente aos vértices P_1 e P_2 do cubo da Fig. 2.4, onde os eixos y_1, y_2 e y_3 estão associados ao primeiro, segundo e terceiro bit, respectivamente.

Ao ponto P_1 corresponde uma energia normalizada nula e ao ponto P_2 corresponde uma energia normalizada $E_2 = \sqrt{A^2 + A^2 + A^2} = \sqrt{3}A$ igual à distância do ponto à origem.

Surge-nos um conflito: por um lado gostaríamos que a distância entre os pontos válidos P_1 e P_2 fosse grande

(para que sejam bem distintos)

portanto, quanto maior A, melhor…; mas por outro, muitas vezes essa distância não pode ser muito grande

(porque as amplitudes A não podem ser muito elevadas devido a normas ou a limitações práticas de potência utilizável impostas pela autonomia de baterias e pilhas, etc.)

Tem de se chegar a um compromisso: nem a distância deve ser muito pequena nem pode ser muito grande.

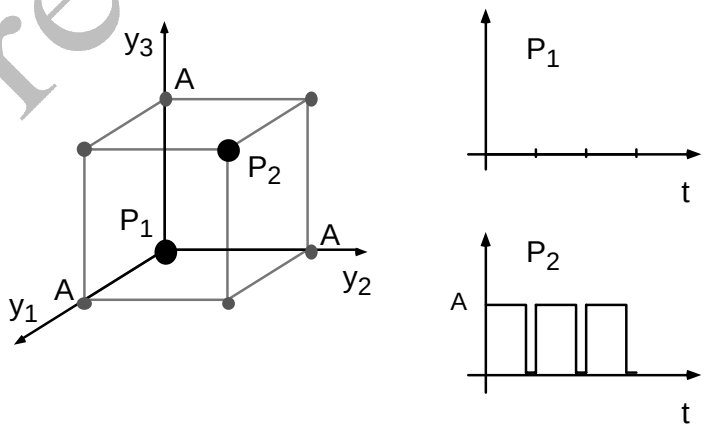


Fig. 2.4 O espaço de sinal de um código (3,1) e impulsos associados.

E é aqui que entram as esferas. Na verdade, sabendo nós que os dois pontos do código estão no centro das respectivas regiões de decisão, podemos imaginar que estas são esféricas, naturalmente.

Os pontos de cada região de decisão esférica encontram-se no interior ou à superfície da esfera. Como a distância entre os centros de esferas adjacentes é, no mínimo, $d_{\min} = 2t + 1$, o raio das esferas é proporcional à distância mínima e vale

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

onde $\lfloor x \rfloor$ representa o *chão* de *x*, ou seja, o maior inteiro não superior a *x*.

Assim, tendo esferas centradas nos pontos de código e com o raio adequado para obtermos uma certa capacidade de correcção de erros, o problema está em empacotá-las de um modo compacto, por forma a ocuparem o menor volume possível (mínima energia). É uma situação idêntica à de, num certo volume de armazém de supermercado, colocar aí o maior número possível de laranjas.

No caso geral o espaço de sinal é composto por 2^k esferas imaginárias. Quantos pontos é que estão dentro e à superfície de cada uma? Ora existem $\binom{n}{1}$ pontos à distância 1 do centro, $\binom{n}{2}$ pontos à distância 2, $\binom{n}{3}$ pontos à distância 3, etc., e à medida que nos afastamos do centro mais pontos vamos encontrar, é claro, até à superfície da esfera, onde vamos ter $\binom{n}{t}$ pontos. Portanto, ao todo e já contando com o ponto válido situado no centro, em cada esfera temos um número de pontos igual a

$$1 + \binom{n}{1} + \binom{n}{2} + \ldots + \binom{n}{t} = \sum_{i=0}^{t} \binom{n}{i}$$

Resumindo:

1) existem 2^n pontos no espaço de sinal.
2) existem 2^k esferas de decisão.
3) cada esfera engloba $\sum_{i=0}^{t} \binom{n}{i}$ pontos.
4) há pontos que podem não pertencer a nenhuma esfera.

Ora como as esferas não se podem interpenetrar (é como se fossem sólidas e indeformáveis) então

$$N^o \text{ de esferas } \times N^o \text{ de pontos /esfera} \leq N^o \text{ total de pontos,}$$

ou

$$2^k \sum_{i=0}^{t} \binom{n}{i} \leq 2^n \quad \Rightarrow \quad \sum_{i=0}^{t} \binom{n}{i} \leq 2^{n-k}$$

A esta desigualdade dá-se o nome de *limite de Hamming*[3]. Ele estabelece uma relação entre o número de símbolos de paridade de um código de blocos, n-k, o comprimento n e a respectiva capacidade de correcção de erros, t.

A questão do empacotamento de esferas num determinado volume não é trivial e já foi objecto de muitos estudos para se chegar a bons resultados.

2.3. Codificação e descodificação

Nas secções seguintes vamos ver como é que se faz a codificação de uma mensagem e respectiva descodificação.

2.3.1. Codificação com matrizes

As palavras de um código (n,k) binário, sistemático e linear obtêm-se efectuando o produto matricial

$$\boldsymbol{Y} = \boldsymbol{X}\,\boldsymbol{G},$$

em que **G** — a matriz geradora — é uma matriz de elementos binários com k linhas e n colunas com a estrutura

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & \cdots & 0 & p_{11} & \cdots & p_{1,n-k} \\ 0 & 1 & \cdots & 0 & p_{21} & \cdots & p_{2,n-k} \\ 0 & 0 & \cdots & 0 & p_{31} & \cdots & p_{3,n-k} \\ \vdots & \vdots & & \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & 1 & p_{k1} & \cdots & p_{k,n-k} \end{bmatrix} = \left[\mathbf{I}_k \middle| \mathbf{P}\right]$$

$$\underbrace{\qquad}_{k\times k} \quad \underbrace{\qquad\qquad}_{k\times(n-k)}$$

[3] Há quem também lhe chame *limite de empacotamento de esferas de Hamming*.

se os bits de paridade forem colocados no fim de cada palavra ($\mathbf{Y} = [\mathbf{X} \mid \mathbf{C}]$) e sendo $\mathbf{I}_k$ a matriz identidade $k \times k$. Se os bits de paridade forem colocados no início **G** tomaria a forma $\mathbf{G} = [\mathbf{P} \mid \mathbf{I}_k]$.

As *k* linhas da matriz geradora **G** são linearmente independentes, ou seja, nenhuma delas consegue ser obtida por adição de outras linhas.

É fácil de verificar que cada linha da matriz **G** é uma palavra de código (experimente com $\mathbf{X} = [10\ldots0]$: **Y** será igual à primeira linha de **G**).

Sendo então $\mathbf{Y} = [\mathbf{X} \mid \mathbf{C}] = \mathbf{XG} = \mathbf{X}[\mathbf{I}_k \mid \mathbf{P}]$, podemos dispensar a matriz identidade no cálculo dos símbolos de paridade, pois fica

$$\mathbf{C} = \mathbf{XP},$$

o que é mais fácil e rápido de calcular visto a submatriz **P** ter menores dimensões que a matriz geradora **G**.

Sendo $\mathbf{X} = [x_1 x_2 \ldots x_k]$ e $\mathbf{C} = [c_1 c_2 \ldots c_{n-k}]$, a equação anterior pode reescrever-se na forma

$$c_j = \sum_{i=1}^{k} x_i p_{ij} \qquad j = 1,2,\ldots,n-k$$

Esta expressão indica que o elemento p_{ij}, situado na linha i e coluna j de **P**, é o coeficiente do símbolo de informação de ordem i (x_i) na equação vectorial que determina o símbolo de paridade c_j. Então por inspecção visual das colunas de **P** podemos imediatamente calcular os símbolos de paridade. Uma outra alternativa para calcular o vector **C** é atender à seguinte regra prática:

> *A multiplicação em módulo 2 de um vector por uma matriz é igual à soma das linhas da matriz cuja numeração corresponda às posições dos bits "1" do vector.*

Por exemplo, se **X** = [0110] basta somar a segunda e a terceira linhas de **P** para calcular **C** = **XP**.

Evidentemente que também podemos calcular as palavras de código somando outras palavras já existentes, como se fez na Secção 2.1.

A codificação pode ser feita com o circuito combinatório da Fig. 2.5.

Um outro esquema de codificação usa uma memória ROM para armazenar a submatriz **P** (Fig. 2.6). Neste esquema os k bits de informação são, por um lado, colocados na saída pelo comutador, como no esquema combinatório, e, por outro, servem para escolher os endereços de memória onde estão colocadas as linhas de **P** que vão ser somadas para se obter o vector de paridade. No fim de cada palavra o comutador coloca na saída os $n-k$ bits de paridade provenientes do *registo de paridade*.

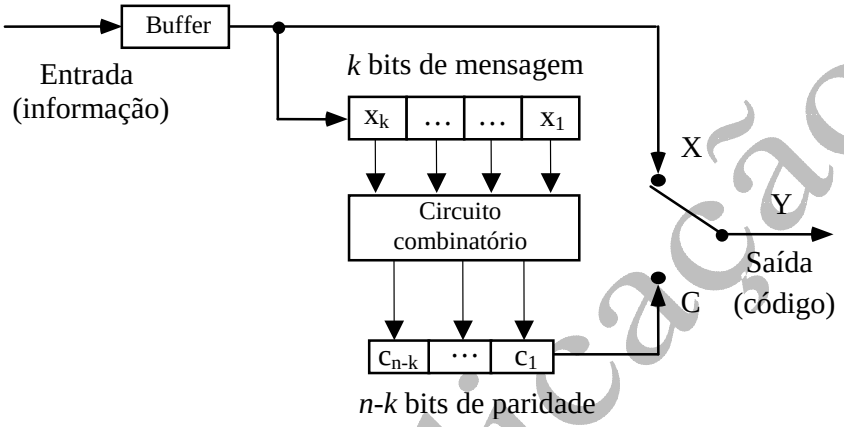


Fig. 2.5 Codificador (n,k) com circuito combinatório.

O esquema com memória é mais conveniente que o esquema com circuitos lógicos para valores elevados de n e $n-k$.

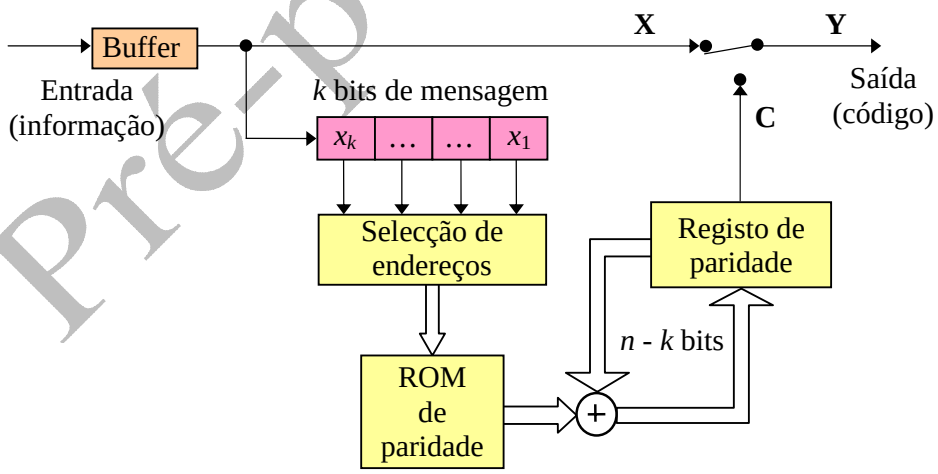


Fig. 2.6 Codificador (n,k) com memória.

Exemplo 2.4

A matriz geradora de um determinado código binário linear é a seguinte:

$$\mathbf{G} = \left[\mathbf{I}_k \middle| \mathbf{P}\right] = \left[\mathbf{I}_4 \middle| \mathbf{P}\right] = \left[\begin{array}{cccc|ccc} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{array}\right]$$

As suas dimensões indicam-nos que se trata de um código (7,4). Se o vector de informação genérico for representado por $\mathbf{X} = \left[x_1 x_2 x_3 x_4\right]$ e se $\boldsymbol{C} = \left[c_1 c_2 c_3\right]$ representar os três bits de verificação de paridade, estes são calculados através de $\boldsymbol{C} = \boldsymbol{X}\,\boldsymbol{P}$, originando o sistema de 3 equações

$$\begin{cases} c_1 = x_1 + x_2 + x_3 + 0 \\ c_2 = 0 + x_2 + x_3 + x_4 \\ c_3 = x_1 + x_2 + 0 + x_4 \end{cases}$$

Poderíamos ter chegado a esta mesma conclusão por inspecção visual de **G**. E como? Observando as colunas da submatriz **P**, pois cada uma vai servir-nos para calcular um símbolo de paridade. Assim, o último elemento da primeira coluna é zero, logo

$$c_1 = x_1 + x_2 + x_3$$

Na segunda coluna de **P** o primeiro elemento, na primeira linha, é zero, logo

$$c_2 = x_2 + x_3 + x_4$$

Finalmente, na terceira coluna existe um elemento nulo na terceira linha, logo

$$c_3 = x_1 + x_2 + x_4$$

Qual é a palavra de código correspondente à mensagem de informação $\mathbf{X} = [1011]$? Basta calcular $\boldsymbol{C} = \left[c_1 c_2 c_3\right]$ substituindo os valores de **X**:

$$\begin{cases} c_1 = 1 + 0 + 1 = 0 \\ c_2 = 0 + 1 + 1 = 0 \\ c_3 = 1 + 0 + 1 = 0 \end{cases}$$

A palavra de código correspondente é $\mathbf{Y} = [1011000]$. Também poderia ter sido obtida somando a 1ª, 3ª e 4ª linhas de **G**:

(1)		1000 101
(3)		0010 110
(4)	+	0001 011
		1011 000

O esquema combinatório da Fig. 2.5 pode ser usado para efectuar a codificação (ver Fig. 2.7).

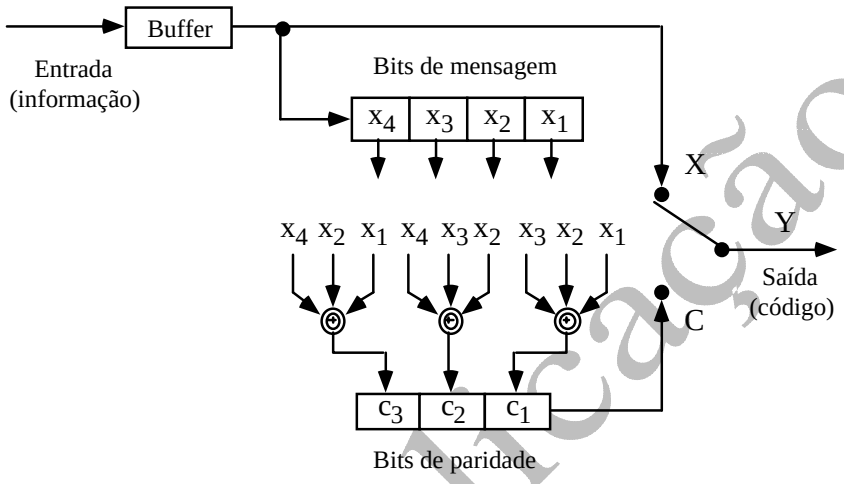


Fig. 2.7 Codificador (7,4).

A operação de codificação também pode ser realizada armazenando a submatriz **P** numa memória ROM com $k \times (n-k) = 4 \times 3$ elementos binários, isto é, com $k = 4$ posições de $n-k = 3$ bits cada:

Posição	Conteúdo
1	101
2	111
3	110
4	011

Recordemos que nos códigos lineares os $n-k$ bits de verificação de paridade podem ser obtidos somando linhas da submatriz **P**, de acordo com a posição dos bits não nulos nos k bits de informação. Então se, como antes, quiséssemos codificar a mensagem $\mathbf{X} = [1011]$, teríamos de ler e somar os conteúdos das posições de memória 1, 3 e 4 (correspondentes às linhas 1, 3 e 4 da submatriz **P**). Os bits de paridade a acrescentar a **X** seriam, como antes,

(1)		101
(3)		110
(4)	+	011
		000

2.3.2. Gráficos bipartidos

As equações de paridade obtidas de **C** = **XP** vão ajudar-nos a desenhar os chamados *gráficos de Tanner* ou *gráficos bipartidos*. Vamos servir-nos de um exemplo. Seja um código de Hamming (7,4) com estas equações de paridade:

$$\begin{cases} c_1 = x_1 + x_2 + x_3 \\ c_2 = x_2 + x_3 + x_4 \\ c_3 = x_1 + x_2 + x_4 \end{cases}$$

Sendo a palavra de código igual a $\mathbf{Y} = [x_1 x_2 x_3 x_4 c_1 c_2 c_3] = [y_1 y_2 y_3 y_4 y_5 y_6 y_7]$ podemos reescrever o sistema de equações anterior desta maneira:

$$\begin{cases} y_1 + y_2 + y_3 + y_5 = 0 \\ y_2 + y_3 + y_4 + y_6 = 0 \\ y_1 + y_2 + y_4 + y_7 = 0 \end{cases}$$

Um gráfico bipartido é constituído pelos chamados *nós de variáveis* e *nós de verificação* (ou *nós de controlo*). Vamos representar os primeiros por *n* círculos e os segundos por $n - k$ quadrados, em grupos separados (ver Fig. 2.8) ligados por segmentos de recta tendo em conta o sistema homogéneo de equações anterior. Assim, neste exemplo vamos ter sete nós de variáveis, cada um correspondente a uma variável y_i, e três nós de verificação, cada um correspondente a uma equação. Por exemplo, ao primeiro nó de verificação vão estar ligadas as variáveis y_1, y_2, y_3 e y_5, cuja soma, de acordo com a primeira equação atrás, é nula. O mesmo se passaria com as variáveis das restantes equações. Ou seja, a soma das variáveis que confluem num dado nó de verificação é nula se a palavra de *n* bits pertencer ao código; se não pertencer, pelo menos uma das somas calculadas nos nós de verificação não será nula. Esta é, assim, uma maneira de detectar a ocorrência de erros nas palavras. Veremos mais adiante que os valores das somas, designados na Fig. 2.8 por s_1, s_2 e s_3, são os elementos de um vector de $n - k$ bits usado na descodificação chamado *síndrome*.

Repare-se que os nós de um dado grupo não estão ligados entre si.

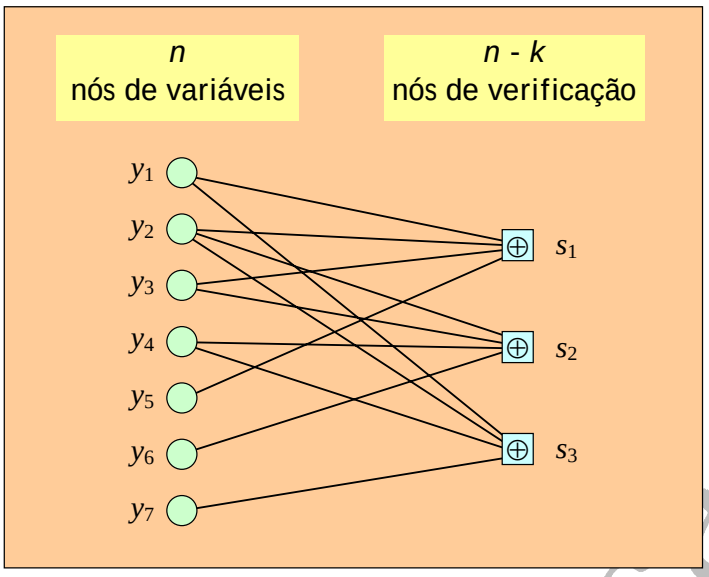


Fig. 2.8 O gráfico bipartido, ou de Tanner, do código (7,4) do exemplo.

2.3.3. A matriz de verificação de paridade

À matriz geradora podemos associar uma outra matriz, chamada *matriz de verificação de paridade*. Designada por **H**, possui *n-k* linhas e *n* colunas com uma estrutura

$$\mathbf{H} = \left[\mathbf{P}^T \middle| \mathbf{I}_{n-k}\right].$$

Tal como em **G**, também as linhas de **H** são linearmente independentes.

A matriz transposta

$$\mathbf{H}^T = \left[\frac{\mathbf{P}}{\mathbf{I_{n-k}}}\right] = \begin{bmatrix} p_{11} & \cdots & p_{1,n-k} \\ p_{21} & \cdots & p_{2,n-k} \\ \vdots & & \\ p_{k1} & \cdots & p_{k,n-k} \\ 1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1 \end{bmatrix}$$

vai ser usada na descodificação.

É a submatriz **P**, de dimensões $k \times (n-k)$ e presente tanto em **G** como em $\mathbf{H}^T$, que vai determinar as propriedades do código, para cada conjunto de valores (n,k). E

se antes as colunas de **G** (e de **P**) serviam para determinar os símbolos de paridade, estes também podem ser obtidos através das colunas de $\mathbf{H}^T$.

É de notar que o produto da matriz geradora **G** pela transposta de **H** é uma matriz $k \times (n-k)$ nula:

$$\mathbf{G}\mathbf{H}^T = \left[\mathbf{I}_k \middle| \mathbf{P}\right] \begin{bmatrix} \mathbf{P} \\ \hline \mathbf{I}_{n-k} \end{bmatrix} = \mathbf{P} + \mathbf{P} = \mathbf{0}$$

Este facto tem muita importância porque é a base das técnicas de descodificação. Imaginemos, por exemplo, que temos uma qualquer palavra de código **Y**. Se a multiplicarmos pela transposta de **H** o que é que obtemos? Vejamos…

$$\boldsymbol{Y}\boldsymbol{H}^T = \boldsymbol{X}\underbrace{\boldsymbol{G}\boldsymbol{H}^T}_{\boldsymbol{0}} = \boldsymbol{0}$$

Dá zero! Então se em vez de termos multiplicado $\mathbf{H}^T$ pela palavra de código **Y** tivéssemos feito a multiplicação por um outro vector de *n* bits, não pertencente ao código, o resultado já não seria nulo. Logo, temos aqui uma maneira de saber se um determinado vector, **Z**, de *n* bits é ou não uma palavra válida do código: multiplicamo-lo por $\mathbf{H}^T$. Se der zero concluímos que… sim, **Z** é uma palavra válida de código; se não der zero concluímos que…não, **Z** não é uma palavra de código.

2.3.4. Determinação de d_{min} através da matriz H

A matriz **H** tem outra utilidade além de servir para descodificar: com ela podemos determinar a distância mínima do código. Vejamos como. Consideremos a palavra 1011000 pertencente ao código (7,4) do Exemplo 2.4. Se multiplicarmos esta palavra pela transposta da matriz **H** o resultado é nulo, já sabemos; sabemos também que ao multiplicar um vector binário por uma matriz binária o resultado é igual à soma das linhas da matriz com a mesma posição dos bits não nulos do vector, ou seja, o produto, nulo, da palavra 1011000 por $\mathbf{H}^T$ é igual à soma das linhas de ordem 1, 3 e 4 de $\mathbf{H}^T$, isto é, é igual à soma de tantas linhas quanto o peso da palavra. Quer dizer, se uma palavra de código tem peso 3 (como neste exemplo), então há três linhas de $\mathbf{H}^T$ (ou três colunas de **H**) que somadas dão zero; se o peso for 4 então há quatro linhas de $\mathbf{H}^T$ que somadas dão zero, e assim por diante. Continuando o raciocínio, se não houver nenhuma palavra de código com peso 5, por exemplo, então também não há nenhum conjunto de cinco linhas em $\mathbf{H}^T$ de soma nula; caso contrário isso implicaria a existência de uma palavra de código de peso 5, contrariando a hipótese. Ora bem, existe pelo menos uma palavra de código cujo peso é igual à distância mínima (pois, recorda-se, a distância mínima de um código é o menor peso de todas

as palavras de código). Sendo assim podemos dizer que existe um conjunto de d_{min} linhas de $\mathbf{H}^T$ que somadas dão zero. Ou seja:

> *A distância mínima de um código é igual ao menor número de linhas de $\mathbf{H}^T$ cuja soma é nula.*

Dito por outras palavras mais formais:

> *A distância mínima é igual ao menor número de linhas de $\mathbf{H}^T$ linearmente dependentes.*

Concluindo: se, por exemplo, $Y = [010110]$ pertencer a um determinado código, a soma das linhas 2, 4 e 5 de $\mathbf{H}^T$ é nula, de certeza.

Exemplo 2.5

Determine a distância mínima do código (8,4) cuja matriz de verificação de paridade é

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

R.: O que atrás se disse para as linhas de $\mathbf{H}^T$ aplica-se, naturalmente, às colunas de **H**. Nesta matriz: 1) nenhuma coluna é igual ao vector nulo, logo, $d_{min} > 1$; 2) todas as colunas são diferentes de zero, logo, não há nenhum par de colunas cuja soma seja **0**.

Já podemos concluir que $d_{\min} \geq 3$.

Mas como qualquer coluna de **H** tem peso ímpar também não há três colunas cuja soma seja **0** (pois o número de uns em qualquer grupo de três colunas é necessariamente ímpar, o que impede que a soma em módulo 2 seja nula pois para isso é necessário que o número de uns seja par. Não?!).

Logo, $d_{\min} \geq 4$.

Se agora somarmos as quatro colunas 1, 2, 6 e 7, por exemplo,

$$\begin{bmatrix}1\\1\\0\\1\end{bmatrix}+\begin{bmatrix}1\\0\\1\\1\end{bmatrix}+\begin{bmatrix}0\\1\\0\\0\end{bmatrix}+\begin{bmatrix}0\\0\\1\\0\end{bmatrix}=\begin{bmatrix}0\\0\\0\\0\end{bmatrix},$$

concluímos finalmente que a distância mínima é 4.

É fácil de verificar que, noutro exemplo, o vector [10100011] de peso 4 pertence ao código. Então de certeza que a soma das colunas 1, 3, 7 e 8 de **H** é zero. Confirme!

2.3.5. Códigos duais

Cada código tem o seu dual. O que é isso?

Vimos que $\mathbf{G}\mathbf{H}^T=\mathbf{0}$. Então também $\mathbf{H}\mathbf{G}^T=\mathbf{0}$, isto é, **H** pode ser a matriz geradora de um novo código e **G** a sua matriz de verificação de paridade.

O código cuja matriz geradora é a matriz de verificação de paridade de um código (n,k) é o *código dual* deste. O código dual de um código (n,k) é um código $(n,n-k)$. Assim, este último tem um número de bits de informação (ou dimensão) igual ao número de bits de paridade do código original.

A matriz geradora de um código dual é designada por $\boldsymbol{G}^{\perp}$.

Exemplo 2.6

Qual é a matriz geradora do código dual do código (7,4) definido pela matriz geradora

$$\mathbf{G}=\begin{bmatrix}1&0&0&0&1&0&1\\0&1&0&0&1&1&1\\0&0&1&0&1&1&0\\0&0&0&1&0&1&1\end{bmatrix}=\left[I_4 \mid P\right]?$$

R.: A matriz de verificação de paridade é

$$\mathbf{H}=\left[P^T \mid I_3\right]=\begin{bmatrix}1&1&1&0&1&0&0\\0&1&1&1&0&1&0\\1&1&0&1&0&0&1\end{bmatrix}$$

O código dual deste código (7, 4) é o código (7, 3) cuja matriz geradora é a matriz de verificação de paridade **H**:

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

2.3.6. A síndrome

Vamos tentar beneficiar do facto de o produto de uma palavra de código por $\mathbf{H}^T$ ser nulo. Assim, e para começar, ao vector-produto de um vector qualquer de *n* bits, **Z**, por $\mathbf{H}^T$ vamos dar o nome de síndrome, **S**:

$$\boldsymbol{S} = \boldsymbol{Z}\boldsymbol{H}^T$$

Este termo, *síndrome*, provém da medicina: segundo o Dicionário da Porto Editora, representa o "conjunto bem determinado de sintomas que não caracterizam necessariamente uma só afecção patológica, uma só doença, mas podem traduzir certa modalidade patogénica".

No nosso contexto vamos calcular síndromes para determinarmos se uma dada palavra recebida apresenta sintomas de alguma "doença", isto é, se tem erros, e posteriormente proceder à descodificação.

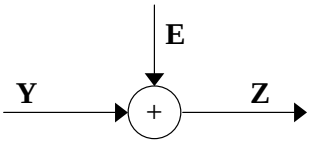
Retomando o que se disse antes, sabemos que:

- Uma síndrome não nula indica que o vector **Z** não pertence ao código, isto é, indica a presença de erros.
- Uma síndrome nula significa que:
  - ou não houve erros introduzidos na transmissão;
  - ou houve erros na transmissão mas foram tais que transformaram a palavra de código enviada numa outra palavra de código válida.

A síndrome é um vector de $n-k$ elementos que, sendo binários, originam 2^{n-k} síndromes possíveis. Ora, de acordo com a definição, **S** calcula-se multiplicando um vector de *n* elementos pela matriz transposta de **H**. Como existem 2^n vectores de *n*

bits e estes originam um número muito menor de síndromes concluímos que tem de haver vários vectores a produzir a mesma síndrome. Mas se é assim então também teremos de concluir que, apesar da definição, $\mathbf{S} = \mathbf{Z}\mathbf{H}^T$, a síndrome não depende exclusivamente da palavra **Z**. Vamos agora ver que, afinal, a síndrome do que depende é dos erros contidos em **Z**:

A palavra **Z** que pretendemos descodificar[4] é igual à soma da palavra enviada **Y** com um *vector de erro* **E**, que pode ou não ser nulo: $\boldsymbol{Z} = \boldsymbol{Y} + \boldsymbol{E}$.

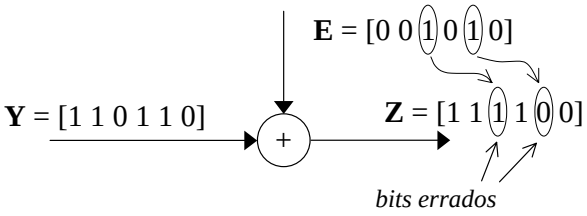


A síndrome respectiva vale

$$\mathbf{S} = \mathbf{Z}\mathbf{H}^T = (\mathbf{Y} + \mathbf{E})\mathbf{H}^T = \underbrace{\mathbf{Y}\mathbf{H}^T}_{0} + \mathbf{E}\mathbf{H}^T = \mathbf{E}\mathbf{H}^T$$

Está provado: realmente a síndrome só depende do vector de erro. No entanto, como existem mais vectores de erro (2^n, exactamente) do que síndromes, estas não determinam univocamente aqueles. Mesmo assim podemos usar **S** para detectar e corrigir erros; a pista a seguir é a dos vectores de erro mais prováveis.

O que é um vector, ou padrão, de erros **E**? É simplesmente um vector de *n* bits em que os bits "1" indicam a posição dos erros em **Z**. Assim, o vector de erro [001010] indica que a palavra **Z** da figura seguinte tem o terceiro e o quinto bits errados:



[4] **Z** corresponde a uma palavra recebida num receptor de telecomunicações ou a uma palavra guardada num dispositivo de armazenamento (memória, CD, etc.). Em qualquer das situações **Z** é a palavra a descodificar.

Exemplo 2.7

A matriz de verificação de paridade de um código linear binário (4,2) é

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

a) Quais são as palavras de código?

b) Quais são as síndromes correspondentes às palavras [0011] e [1000]?

R.: a) Inspeccionando a matriz **H** vemos que os dois primeiros elementos da primeira e segunda linhas (isto é, $\mathbf{P}^T$) são 11 e 10, respectivamente. Então os dois bits de paridade, c_1 e c_2, vêm dados em função dos dois bits de informação, x_1 e x_2, por

$$c_1 = x_1 + x_2$$
$$c_2 = x_1$$

Daqui se tira imediatamente que as quatro palavras de código são {0000, 0110, 1011, 1101}.

b) Cálculo das síndromes

- Síndrome correspondente a [0011]:

$$\mathbf{S}_1 = [0011] \times \begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} = [11]$$

Nem precisaríamos de fazer a multiplicação: como se trata de operações binárias e os bits não nulos de [0011] estão na terceira e quarta posições então basta somar a terceira e a quarta linhas de $\mathbf{H}^T$.

- Síndrome correspondente a [1000]: é a primeira linha de $\mathbf{H}^T$, $S_2 = [11]$.

Como se vê, $\mathbf{S}_1 = \mathbf{S}_2$. Aqui está um exemplo em que dois vectores dão origem à mesma síndrome. O primeiro vector, [0011], pode ser encarado como um padrão de erro que transformou o vector de código [0000] em si próprio. Então podemos dizer que o segundo vector do exemplo, [1000], pode ter sido obtido somando

esse mesmo padrão de erro [0011] a um dos outros vectores de código não nulos, pois as duas síndromes são iguais. De facto, [1000] pode ter sido obtido somando [0011] ao vector de código [1011], como se mostra a seguir:

Palavra de código		Padrão de erro		Palavra recebida
0000	+	0011	=	0011
1011	+	0011	=	1000

2.3.7. Padrões de erro mais prováveis

Dentre os 2^n padrões de erro possíveis quais são os mais prováveis? Para responder a esta pergunta temos de calcular as probabilidades de haver um, dois ou mais erros.

Se a probabilidade de um bit estar errado for $p < 1/2$ (devido ao ruído no canal, por exemplo), então numa palavra de *n* bits a probabilidade de haver 1 bit errado é

$$P(1,n) = \binom{n}{1} p(1-p)^{n-1}$$

visto haver $\binom{n}{1}$ padrões possíveis com um erro. A probabilidade de haver 2 bits errados é

$$P(2,n) = \binom{n}{2} p^2 (1-p)^{n-2}.$$

Generalizando: a probabilidade de haver i bits errados em *n* bits é

$$P(i,n) = \binom{n}{i} p^i (1-p)^{n-i}.$$

O que é que é maior, $P(1,n)$ ou $P(2,n)$? À primeira vista diríamos que é $P(1,n)$. E temos razão… Dividindo o primeiro pelo segundo obtemos

$$\frac{P(1,n)}{P(2,n)} = \frac{\binom{n}{1} p(1-p)^{n-1}}{\binom{n}{2} p^2 (1-p)^{n-2}} = \frac{2}{n-1}\frac{1-p}{p}.$$

Esta fracção é superior a 1 se $p < \frac{2}{1+n}$, o que sempre se verifica na prática. Portanto é mais provável uma palavra conter apenas um erro do que dois; do mesmo modo se poderia mostrar que é mais provável haver dois erros do que três ou mais, e assim por diante.

A conclusão que se tira é que os padrões de erro mais prováveis são os que correspondem a menos erros.

Exemplo 2.8

A probabilidade de erro num determinado canal binário simétrico é $p = 10^{-3}$. Confirme que a probabilidade de haver um bit errado numa palavra de 10 bits é maior que a probabilidade de haver dois erros.

R.: Temos de calcular $P(1,10)$ e $P(2,10)$:

$$P(1,10) = 10 \times 10^{-3} \times \left(1-10^{-3}\right)^{9} = 0,0099$$

$$P(2,10) = 45 \times \left(10^{-3}\right)^{2} \times \left(1-10^{-3}\right)^{8} = 4,46 \times 10^{-5}$$

Está confirmado!

2.3.8. Descodificação com síndromes

A descodificação de códigos de blocos é feita consultando uma tabela de síndromes e padrões de erro.

O número de padrões de erro corrigíveis é de 2^{n-k}, tantos quantas as síndromes possíveis; logo, havendo 2^{n} vectores de erro e só podendo usar 2^{n-k}, o melhor é usar na tabela os vectores de erro mais prováveis — aqueles que têm menor peso — dado que as probabilidade de ocorrência não são iguais. É claro que não podemos ter a certeza de "aqueles" terem sido os vectores de erro mas, pelo menos, são os mais prováveis…

A esta técnica de estimação de erros dá-se o nome de *descodificação de máxima verosimilhança*.

A estimativa da palavra enviada é obtida somando o vector de erro estimado, $\hat{\mathbf{E}}$, à palavra recebida. De facto, como a palavra recebida **Z** é igual a $\boldsymbol{Z} = \boldsymbol{Y} + \boldsymbol{E}$, se estimarmos correctamente **E** podemos recuperar **Y** somando essa estimativa a **Z**,

$$\boldsymbol{Z} + \hat{\boldsymbol{E}} = (\boldsymbol{Y} + \boldsymbol{E}) + \hat{\boldsymbol{E}} = \boldsymbol{Y} \qquad (\text{se } \hat{\boldsymbol{E}} = \boldsymbol{E}),$$

como se mostra no diagrama de descodificação da Fig. 2.9.

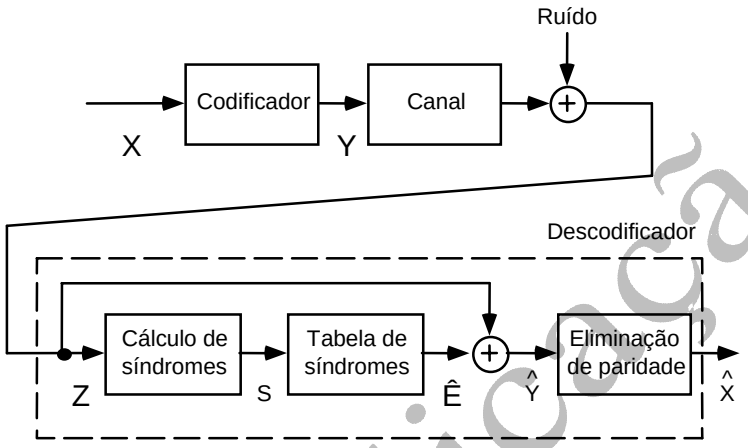


Fig. 2.9 Diagrama de codificação e descodificação de blocos.

As equações de paridade para determinação dos bits de paridade na codificação foram obtidas observando as colunas da submatriz **P**. Na descodificação o vector síndrome é obtido de maneira análoga observando as linhas da matriz **H** (ou as colunas de $\mathbf{H}^T$) e notando onde aparecem os bits "1". Na verdade, se $Z = \left[z_1 z_2 \ldots z_n\right]$ for o vector recebido e $\mathbf{H}^T$ for

$$\mathbf{H}^T = \begin{bmatrix} p_{11} & \cdots & p_{1,n-k} \\ p_{21} & \cdots & p_{2,n-k} \\ \vdots & & \\ p_{k1} & \cdots & p_{k,n-k} \\ 1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1 \end{bmatrix}$$

então

$$\mathbf{S} = [s_1 s_2 \ldots s_{n-k}] = [z_1 z_2 \ldots z_n]\mathbf{H}^T =$$
$$= \begin{bmatrix} (z_1 p_{11} + z_2 p_{21} + \ldots z_k p_{k,1}) + z_{k+1} \\ (z_1 p_{12} + z_2 p_{22} + \ldots z_k p_{k,2}) + z_{k+2} \\ \vdots \\ (z_1 p_{1,n-k} + z_2 p_{2,n-k} + \ldots z_k p_{k,n-k}) + z_n \end{bmatrix}^T$$

Vê-se aqui que cada elemento de **S** é a soma de um bit de paridade (z_{k+1}, z_{k+2}, etc.) com uma combinação de bits de informação que depende dos valores de **P** (como já ocorrera na codificação):

$$s_j = \sum_{i=1}^{k} z_i p_{ij} + z_{k+j} \qquad j = 1,2,\ldots,n-k$$

Podemos interpretar este cálculo como uma comparação entre cada bit de paridade recebido (z_{k+j}) e aquele que teria sido enviado com base nos *k* bits de informação originais ($\sum_{i=1}^{k} z_i p_{ij}$). Na verdade, o descodificador recalcula os *n-k* bits de paridade usando os primeiros *k* bits da palavra recebida e compara-os com os correspondentes bits de paridade recebidos. Se em alguma das *n-k* comparações os bits forem diferentes é porque a palavra recebida está errada.

Para o cálculo da síndrome basta assim conhecer o sistema de equações dos bits de paridade usado na codificação. Por exemplo, se num código (5,3) sistemático com palavras $\mathbf{Y} = [y_1 y_2 y_3 y_4 y_5]$ os bits de paridade forem calculados através do sistema

$$\begin{cases} y_4 = y_1 + y_2 \\ y_5 = y_2 + y_3 \end{cases}$$

a síndrome de uma palavra $\mathbf{Z} = [z_1 z_2 z_3 z_4 z_5]$ é imediatamente calculada na descodificação através de

$$\begin{cases} s_1 = (z_1 + z_2) + z_4 \\ s_2 = (z_2 + z_3) + z_5 \end{cases}$$

O circuito combinatório usado na codificação (ver Exemplo 2.4) pode ser aproveitado para calcular as síndromes na descodificação, já que a maneira de obter os vectores **C** e **S** é basicamente a mesma. O circuito de cálculo, análogo ao da Fig. 2.5, é o seguinte:

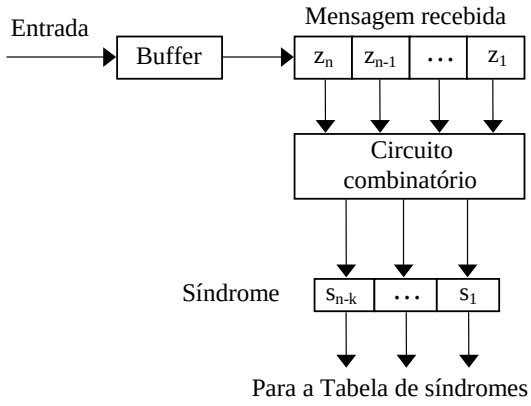


Fig. 2.10 Circuito combinatório de cálculo de síndromes.

Exemplo 2.9

A matriz de verificação de paridade do código (7,4) do Exemplo 2.4 é

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

Os três elementos da síndrome de um vector genérico $\mathbf{Z} = [z_1 z_2 \ldots z_7]$ são dados pelo sistema de equações

$$\begin{cases} s_1 = z_1 + z_2 + z_3 + z_5 \\ s_2 = z_2 + z_3 + z_4 + z_6 \\ s_3 = z_1 + z_2 + z_4 + z_7 \end{cases}$$

resultante da multiplicação matricial $\mathbf{S} = [s_1 s_2 s_3] = \mathbf{Z}\mathbf{H}^T$. No entanto, esta operação não precisa de ser realizada: basta olhar para as três linhas de **H** e ver em cada uma onde aparecem os bits "1".

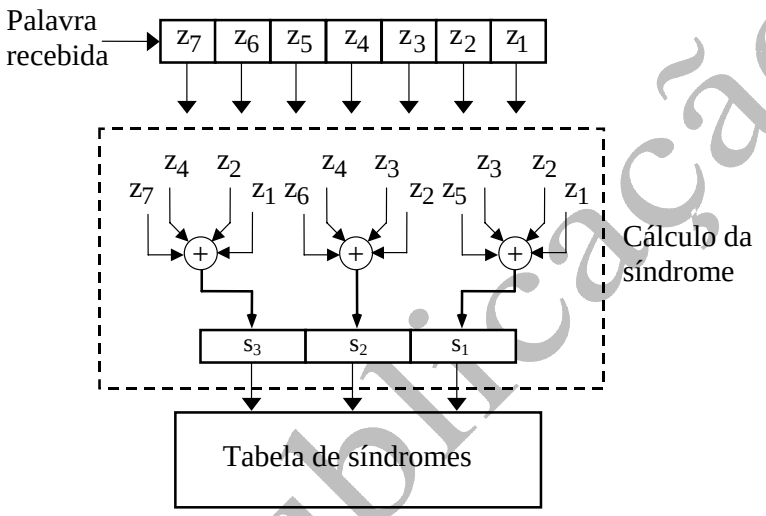
Poderíamos chegar às mesmas conclusões observando as equações de paridade deste código (em que $\mathbf{Y} = [y_1 y_2 \ldots y_7]$ é a palavra codificada):

$$\begin{cases} y_5 = y_1 + y_2 + y_3 \\ y_6 = y_2 + y_3 + y_4 \\ y_7 = y_1 + y_2 + y_4 \end{cases} \Rightarrow \begin{cases} s_1 = (z_1 + z_2 + z_3) + z_5 \\ s_2 = (z_2 + z_3 + z_4) + z_6 \\ s_3 = (z_1 + z_2 + z_4) + z_7 \end{cases}$$

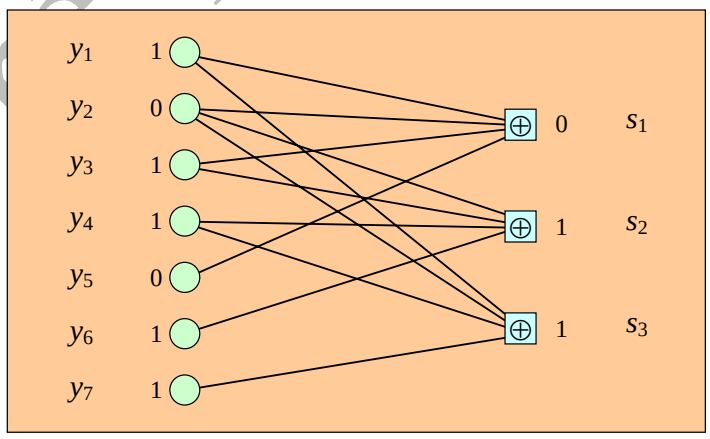
As equações da síndrome podem escrever-se exclusivamente em função dos bits errados. Na verdade, sendo **Y** a palavra de código enviada e $\mathbf{E} = [e_1 e_2 \ldots e_7]$ um padrão de erro genérico, como $\mathbf{S} = \mathbf{Z}\mathbf{H}^T = (\mathbf{Y} + \mathbf{E})\mathbf{H}^T = \mathbf{E}\mathbf{H}^T$ então

$$\begin{cases} s_1 = e_1 + e_2 + e_3 + e_5 \\ s_2 = e_2 + e_3 + e_4 + e_6 \\ s_3 = e_1 + e_2 + e_4 + e_7 \end{cases}$$

O circuito combinatório de cálculo da síndrome é o seguinte:



Qual é a síndrome correspondente à palavra de sete bits **Z** = [1011011]? É fácil de concluir que é **S** = [011]. Poderíamos chegar ao mesmo valor usando um gráfico de Tanner:



Através de um exemplo vamos calcular a tabela de síndromes de um código linear usando a regra da máxima verosimilhança.

Exemplo 2.10

Determine a tabela de síndromes do código sistemático (7,3) caracterizado pela submatriz

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

A palavra recebida **Z**=[1011010] é uma palavra de código? Se não for, qual terá sido a palavra original e a mensagem que lhe deu origem?

R.: Existem 2^4=16 síndromes. Devemos encontrar os 16 padrões de erro mais prováveis, aqueles que têm menor peso; existem $\binom{7}{1} = 7$ padrões com peso 1 e $\binom{7}{2} = 21$ com peso 2. Portanto, na tabela pretendida vamos colocar o padrão nulo, os sete padrões de um erro e 16-(7+1) = 8 dos 21 padrões de dois erros. Uma escolha possível é a tabela seguinte.

Ê	S	Ê	S
0000000	0000	0001000	1000
0000001	0001	0100000	1001
0000010	0010	0001010	1010
0000011	0011	0100010	1011
0000100	0100	1000001	1100
0000101	0101	1000000	1101
0010000	0110	0011000	1110
0010001	0111	0110000	1111

Na Tabela poderíamos ter escolhido o vector de dois erros [1100000]? A sua síndrome é [1100000] $\mathbf{H}^T$ = [0100]; mas este valor foi obtido com o padrão [0000100], que tem peso 1 e por isso é mais provável que [1100000]. Portanto, a

resposta à pergunta é: não, não poderíamos! Isto ensina-nos que a escolha dos padrões de dois erros não é arbitrária.

E a palavra recebida, **Z**=[1011010], é ou não uma palavra de código? As síndromes são obtidas de **H** através do sistema de equações

$$\begin{cases} s_1 = z_1 + z_2 + z_4 \\ s_2 = z_1 + z_3 + z_5 \\ s_3 = z_3 + z_6 \\ s_4 = z_1 + z_2 + z_7 \end{cases}$$

No caso concreto da palavra **Z** a síndrome respectiva é

$$\mathbf{S} = \begin{bmatrix} s_1 & s_2 & s_3 & s_4 \end{bmatrix} = \mathbf{Z}\mathbf{H}^T = [0001]$$

Como $\mathbf{S} \neq \mathbf{0}$ concluimos que a palavra **Z** não é uma palavra de código. Consultando a tabela de síndromes estimamos que o último bit está errado e que terá sido enviada a palavra

$$\hat{\boldsymbol{Y}} = \boldsymbol{Z} + \hat{\boldsymbol{E}} = [1011010] + [0000001] = [1011011]$$

A mensagem de 3 bits original terá sido $\hat{\boldsymbol{X}} = [101]$, dado que o código é sistemático e os bits de paridade foram acrescentados no fim da palavra.

2.3.9. A matriz padrão

A *matriz padrão* de um código (n,k) vai ser-nos útil para determinarmos a capacidade de correcção e detecção de erros do código. É um conjunto ordenado dos 2^n vectores de n bits, dispostos em linhas e colunas de acordo com as seguintes regras:

1) *Na 1ª linha colocamos todos os 2^k vectores do código (n,k), começando pelo vector nulo $\mathbf{Y}_1 = [00\ldots0]$.*

2) *Escolhemos um dos restantes $2^n - 2^k$ vectores (seja $\boldsymbol{E}_2$) e colocamo-lo por baixo de $\mathbf{Y}_1$. A 2ª linha é formada somando E_2 a cada vector Y_i e colocando a soma $E_2 + Y_i$ por baixo de Y_i.*

3) *Nas restantes linhas procede-se de maneira idêntica.*

A matriz resultante tem 2^{n-k} linhas e 2^k colunas; as linhas são designadas por *cosets* e o seu primeiro elemento é o *coset leader*. Genericamente temos uma estrutura assim:

$$\left[\begin{array}{c|cccc} Y_1 = 0 & Y_2 & Y_3 & \cdots & Y_{2^k} \\ \hline E_2 & Y_2 + E_2 & Y_3 + E_2 & \cdots & Y_{2^k} + E_2 \\ E_3 & Y_2 + E_3 & Y_3 + E_3 & \cdots & Y_{2^k} + E_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ E_{2^{n-k}} & Y_2 + E_{2^{n-k}} & Y_3 + E_{2^{n-k}} & \cdots & Y_{2^k} + E_{2^{n-k}} \end{array}\right].$$

A matriz seguinte é uma possível matriz padrão de um código (4,2):

$$\begin{bmatrix} 0000 & 0110 & 1011 & 1101 \\ 0001 & 0111 & 1010 & 1100 \\ 1111 & 1001 & 0100 & 0010 \\ 0011 & 0101 & 1000 & 1110 \end{bmatrix}$$

Qualquer elemento de um "coset" pode ser usado como o seu "coset leader"; aliás é fácil de verificar que os elementos do "coset" não são alterados com a escolha, apenas são permutados.

Seja D_j a coluna de ordem *j* da matriz padrão,

$$\mathbf{D}_j = \begin{bmatrix} \mathbf{Y}_j \\ \mathbf{Y}_j + \mathbf{E}_2 \\ \mathbf{Y}_j + \mathbf{E}_3 \\ \vdots \\ \mathbf{Y}_j + \mathbf{E}_{2^{n-k}} \end{bmatrix}$$

em que os vectores $\mathbf{E}_2$, $\mathbf{E}_3$,..., $\mathbf{E}_{2^{n-k}}$ representam os "coset leaders". Cada coluna D_1, D_2, ... , D_{2^k} pode ser usada para descodificar o código. Como?

Suponhamos que o vector de código $\mathbf{Y}_j$ é transmitido através de um canal ruidoso e se recebe a palavra **Z**. Da expressão de D_j vemos que **Z** está nessa coluna se o vector de erro causado pelo canal for um "coset leader" (por exemplo se $\mathbf{Z} = \mathbf{Y}_j + \mathbf{E}_3$). Nesta situação poderemos, a partir de **Z**, indicar o topo da mesma coluna, $\mathbf{Y}_j$, como tendo sido o vector transmitido, e essa indicação é, de facto, uma *descodificação correcta*.

E se o vector de erro (chamemos-lhe **W**) não for um "coset leader"? Nesse caso a descodificação será incorrecta. Vejamos porquê (Fig. 2.11): se não é um "coset

leader" **W** está localizado no interior da matriz padrão (por exemplo, no "coset" de ordem l e por baixo do vector de código $\mathbf{Y}_i \neq \mathbf{0}$, isto é, $\mathbf{W} = \mathbf{Y}_i + \mathbf{E}_l$). Mas se foi enviado $\mathbf{Y}_j$ e **W** é o padrão de erro, então o vector recebido é

$$\mathbf{Z} = \mathbf{Y}_j + \mathbf{W} = \mathbf{Y}_j + (\mathbf{Y}_i + \mathbf{E}_l) = (\underbrace{\mathbf{Y}_j + \mathbf{Y}_i}_{\mathbf{Y}_s}) + \mathbf{E}_l = \mathbf{Y}_s + E_l$$

O que é que isto quer dizer? Quer dizer que o vector recebido, **Z**, está na coluna D_s encimada pela palavra $\mathbf{Y}_s$ e como tal será descodificado como $\mathbf{Y}_s$, que não é a palavra transmitida.

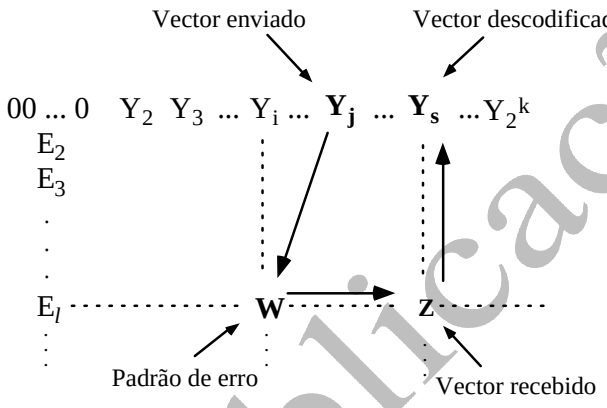


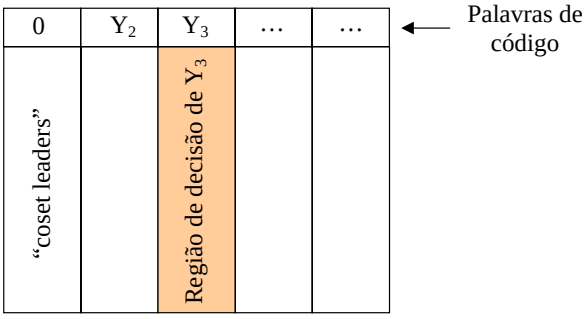
Fig. 2.11 Erro de descodificação na matriz padrão.

Concluímos que a descodificação estará correcta *se e só se* o vector de erro causado pelo canal for um "coset leader". Como há 2^{n-k} "coset leaders" concluímos ainda que

> *Qualquer código de blocos linear (n,k) é capaz de corrigir 2^{n-k} padrões de erro.*

Já que em tempos falámos em regiões de decisão (e esferas…) por que não interpretar as palavras de *n* bits que estão numa dada coluna da matriz padrão como fazendo parte da região de decisão da palavra de código do topo da coluna? É o que se faz na figura seguinte. Deste modo qualquer palavra de *n* bits (um ponto no espaço de sinal) pertence a uma das regiões de decisão assim definidas.

Matriz padrão



O descodificador baseado na matriz padrão e nas regiões de decisão colunares decide-se sempre, bem ou mal, por uma palavra de código: a decisão de corrigir nunca fica por tomar. A um descodificador com esta característica dá-se o nome de *descodificador completo*.

Todos os elementos da mesma linha da matriz padrão têm a mesma síndrome. Para o provar consideremos, por exemplo, o "coset" de ordem *m*, cujos elementos são

$$\left\{\mathbf{E}_m \quad \mathbf{Y}_2+\mathbf{E}_m \quad \cdots \quad \mathbf{Y}_i+\mathbf{E}_m \quad \cdots \quad \mathbf{Y}_{2^k}+\mathbf{E}_m\right\}$$

A síndrome correspondente ao elemento genérico $\mathbf{Y}_i+\mathbf{E}_m$ é

$$\mathbf{S}=(\mathbf{Y}_i+\mathbf{E}_m)\mathbf{H}^T=\underbrace{\mathbf{Y}_i\mathbf{H}^T}_{0}+\mathbf{E}_m\mathbf{H}^T=\mathbf{E}_m\mathbf{H}^T$$

isto é, qualquer que seja o vector $\mathbf{Y}_i+\mathbf{E}_m$ de um "coset" a síndrome é sempre a que corresponde ao seu "coset leader" $\mathbf{E}_m$.

Ora bem, estamos a ver que se estimarmos o "coset leader" podemos estimar o padrão de erro. Como queremos minimizar a probabilidade de um erro de descodificação o melhor é escolher como "coset leaders" os padrões de erro mais prováveis. Já o sabíamos…

Em conclusão: numa matriz padrão

- Qualquer elemento de um "coset" pode ser usado como o seu "coset leader";
- Para minimizar a probabilidade de erro da descodificação devem ser escolhidos como "coset leaders" os padrões de erro mais prováveis, ou seja, os que têm menos peso;
- Todos os elementos de um "coset" têm a mesma síndrome.

Curiosidade histórica

A matriz padrão deve-se a David Slepian (1956).

Exemplo 2.11

Vamos determinar a matriz padrão do código (6,3) cujas palavras são

000000	110100	011010	101110
101001	011101	110011	000111

R.: A matriz padrão tem $2^k = 8$ colunas e $2^{n-k} = 8$ linhas. Na 1ª linha colocamos as oito palavras do código e na 1ª coluna colocamos os padrões de 6 bits com menor peso. O resultado pode ser o seguinte:

Matriz padrão do código (6,3)

000000	110100	011010	101110	101001	011101	110011	000111
000001	110101	011011	101111	101000	011100	110010	000110
000010	110110	011000	101100	101011	011111	110001	000101
000100	110000	011110	101010	101101	011001	110111	000011
001000	111100	010010	100110	100001	010101	111011	001111
010000	100100	001010	111110	111001	001101	100011	010111
100000	010100	111010	001110	001001	111101	010011	100111
010001	100101	001011	111111	111000	001100	100010	010110

Na 1ª coluna da matriz estão todos os padrões com um erro e apenas um padrão com dois erros (010001). São estes os vectores de erro corrigíveis. Assim, $t = 1$. Este valor condiz com o facto da distância mínima do código ser 3 (veja-se o conjunto das palavras de código), pois sabemos que $d_{\min} \geq 2t + 1$.

Um aparte: O padrão 010001 é um "coset leader" com peso 2. Mas nem todos os padrões com peso 2 podem ser "coset leaders". Os padrões 001100 e 100010 poderiam sê-lo, porque estão no mesmo "coset" de 010001. Já o mesmo não podemos dizer de 101000 e 000110, situados num "coset" cujo líder é 000001 (com peso 1), nem de 011000 e 000101 (líder: 000010), 110000 e 000011 (líder: 000100), 010010 e 100001 (líder: 001000), 100100 e 001010 (líder: 010000) e 010100 e 001001 (líder: 100000).

Imaginemos que a palavra enviada foi **Y**=[101001] e que a palavra recebida foi **Z**=[010101]. Observando a matriz padrão estimaríamos $\hat{Y} = [011101]$ (primeiro elemento da coluna onde se encontra **Z**), que é diferente de **Y**. Porque é que nos enganámos? Enganámo-nos porque o vector de erro **E**=**Z**-**Y**=[111100] não é um "coset leader".

Exemplo 2.12

Um código binário linear é gerado pela matriz

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

O código tem uma distância mínima de 4 e corrige todos os erros simples. Corrige ainda erros duplos e, quem sabe, erros triplos também, num total de 15 padrões de erro não nulos corrigíveis.

Suponhamos que as probabilidades de ocorrência de erros em cada palavra de 6 bits estavam de acordo com a seguinte tabela:

Ocorrência	Probabilidade
Não há erros	Mais provável
1 erro	↓
2 erros consecutivos	↓
Outros erros duplos	↓
3 erros	Menos provável

Uma possível matriz padrão para este código e com estas probabilidades de erro seria a seguinte:

Matriz padrão do código (6,2)

000000	101110	011011	110101
000001	101111	011010	110100
000010	101100	011001	110111
000100	101010	011111	110001
001000	100110	010011	111101
010000	111110	001011	100101
100000	001110	111011	010101
000011	101101	011000	110110
000110	101000	011101	110011
001100	100010	010111	111001
110000	011110	101011	000101
001001	100111	010010	111100
010001	111111	001010	100100
100001	001111	111010	010100
111000	010110	100011	001101
110010	011100	101001	000111

Ora bem, há algumas observações a fazer quanto aos "coset leaders":

1) dos cinco padrões de dois erros consecutivos só podemos usar quatro porque um dos "cosets" inclui dois desses cinco padrões (assinalados acima);

2) dos restantes dez padrões de dois erros não consecutivos só podemos usar três porque os outros sete ou já tinham sido colocados nas linhas anteriores ou foram surgindo nos próprios "cosets" dos "leaders";

3) os dois restantes "coset leaders" são padrões de três erros que ainda não estavam na matriz.

2.4. Limites, ou a prova da existência ou inexistência de códigos

Os códigos de blocos gozam de certas propriedades que derivam da definição e da maneira como são construídos. É natural, pois, que a dimensão k e o comprimento

n utilizados condicionem quer a capacidade de correcção de erros quer as probabilidades de estes não serem detectados ou corrigidos.

Vamos em seguida apresentar algumas relações interessantes que ajudam a melhor compreender a escolha dos parâmetros n e k e a estimar o desempenho relativo dos códigos de blocos. Ficaremos então na posse de uma *prova de existência* ou *inexistência* de códigos para certas combinações de n, k e $d_{\min}$.

2.4.1. O limite de Hamming e os códigos perfeitos

Dizer que um código corrige até t erros por palavra significa que todos os padrões com t ou menos erros são corrigíveis; eventualmente também são corrigíveis alguns (mas não todos) padrões com mais de t erros. Ora nós sabemos que existem $\binom{n}{i}$ padrões de i erros, logo o código consegue corrigir $\binom{n}{1}+\binom{n}{2}+\ldots+\binom{n}{t}$ padrões não nulos, pelo menos. Recordando-nos da matriz padrão, vimos que existem $2^{n-k}-1$ "coset leaders" não nulos e que são estes os vectores de erro corrigíveis, de preferência os mais prováveis. Então terá de ser

$$2^{n-k} \geq 1+\binom{n}{1}+\binom{n}{2}+\ldots+\binom{n}{t}=\sum_{i=0}^{t}\binom{n}{i},$$

ou ainda

$$n-k \geq \log\left[1+\binom{n}{1}+\binom{n}{2}+\ldots+\binom{n}{t}\right] \qquad (2.1)$$

Estas relações já eram nossas conhecidas: traduzem o limite de Hamming, que encontrámos quando se falou de regiões de decisão e esferas. Tínhamos então observado, como agora, que este limite é uma relação entre o número de bits de paridade de um código e a sua capacidade de corrigir até t erros por palavras.

A igualdade na equação anterior só se obtém com os chamados *códigos perfeitos*:

> *Diz-se que um código é perfeito se não corrigir nenhuma palavra recebida com mais de t erros. Se corrigir algumas, mas só algumas, palavras com $t+1$ erros ou mais, então o código não é perfeito.*

Os códigos perfeitos são muito raros: os únicos códigos binários perfeitos são os seguintes, dos quais se falará mais adiante:

- códigos de repetição de comprimento ímpar,
- códigos de Hamming,
- código de Golay (23,12).

Apesar do nome, os códigos perfeitos não são necessariamente bons códigos — tendo o mesmo valor de t um código não perfeito corrige mais erros que um código perfeito.

2.4.2. O limite de Singleton

É de prever que a distância mínima de um código dependa da sua dimensão e do seu comprimento. De facto é assim: a distância mínima de um código de blocos (n,k) é limitada superiormente pelo *limite de Singleton*

$$d_{\min} \leq n-k+1 \qquad (2.2)$$

É fácil deduzi-lo: a palavra de código não nula com menor peso tem peso d_{min}, por definição. Ora existem k palavras de código sistemático em que apenas um dos k símbolos de informação é não-nulo (a que corresponde um peso unitário).

> *Se $k=3$, por exemplo, temos três símbolos de informação com peso 1: 001, 010 e 100.*

Com $n-k$ símbolos de paridade tais palavras de código não podem ter peso maior que $1+(n-k)$. Logo, o peso mínimo do código — isto é, d_{min} — não pode ser superior a $1+(n-k)$.

Eis um pequeno exemplo elucidativo:

$$\underbrace{00\ldots001}_{\text{k}}\ \underbrace{??\ldots??}_{\text{n-k}} \quad \Rightarrow \quad peso \leq 1+n-k$$

informação paridade

No máximo o peso seria $1+(n-k)$ se todos os bits de paridade fossem “uns”.

Dos códigos binários só os de repetição é que atingem o maior valor possível, $d_{\min} = n-k+1$; todos os outros códigos binários têm uma distância mínima inferior. Esse valor máximo é também atingido com códigos não binários, como o código de Reed-Solomon (que encontraremos mais tarde).

2.4.3. O limite de Plotkin

O limite de Singleton, $d_{\min} \leq n-k+1$, não é tão apertado ou restritivo como gostaríamos, como se pode ver pelo seguinte exemplo simples: o código binário BCH (15,5), que tem dez bits de paridade, possui uma distância mínima de 7 e no entanto o limite de Singleton só nos informa que, com aqueles valores de *n* e *k*, $d_{\min} \leq 11$.

Em 1960 Plotkin obteve um limite mais apertado que o anterior. Vejamos como.

Se dispusermos todas as 2^k palavras de um código binário como linhas de uma matriz verificamos que em cada uma das *n* colunas metade dos bits (isto é, 2^{k-1}) são zeros e metade são uns. Tomemos como exemplo o código (6,3) apresentado atrás. As suas oito palavras de código dispostas em linha são as seguintes:

000000
110100
011010
101110
101001
011101
110011
000111

Vemos neste exemplo que em cada coluna metade dos bits (quatro) são uns.

Então, e voltando ao caso geral, no conjunto de todas as colunas, isto é, no conjunto de todas as palavras de código, existem $n.2^{k-1}$ uns. Dito de outra maneira, a soma dos pesos de todas as palavras de código é $n.2^{k-1}$. Como há 2^k-1 palavras com peso não nulo, o peso médio é

$$\text{peso médio} = \frac{n.2^{k-1}}{2^k-1}.$$

Ora sendo todos os pesos positivos o peso mínimo (ou seja, $d_{\min}$) não pode ser superior ao peso médio e como tal podemos escrever que

$$d_{\min} \leq \frac{n.2^{k-1}}{2^k-1} \tag{2.3}$$

ou ainda, normalizando em relação ao comprimento,

$$\frac{d_{\min}}{n} \leq \frac{2^{k-1}}{2^k - 1} \tag{2.4}$$

É este o *limite de Plotkin*. Trata-se de um novo limite superior para a distância mínima de qualquer código (n,k) e é mais apertado quando $n >> k$.

Voltando ao código binário BCH(15,5) do início da secção: a aplicação de (2.3) conduz a

$$d_{\min} \leq \frac{15 \times 2^{5-1}}{2^5 - 1} = 7{,}7\,,$$

valor que está bastante próximo do verdadeiro valor da distância mínima deste código, 7.

Com códigos não binários com alfabeto de tamanho q a relação (2.4) escreve-se

$$\frac{d_{\min}}{n} \leq \frac{q^{k-1}(q-1)}{q^k - 1}\,. \tag{2.5}$$

Plotkin mostrou ainda que esta relação origina uma outra entre o número de símbolos de paridade e a distância mínima,

$$n - k \geq \frac{q d_{\min} - 1}{q-1} - 1 - \log_q d_{\min}\,, \qquad \text{se } n \geq \frac{q d_{\min} - 1}{q-1}, \tag{2.6}$$

que para códigos binários se reduz a

$$n - k \geq 2 d_{\min} - 2 - \log_2 d_{\min}\,, \qquad \text{se } n \geq 2 d_{\min} - 1. \tag{2.7}$$

Em face destas relações não se pode esperar que a distância mínima de um qualquer código (6,3), incluindo o do exemplo, seja maior que a que resulta de

$$d_{\min} \leq \frac{n.2^{k-1}}{2^k - 1} = \frac{6 \times 2^2}{2^3 - 1} = 3{,}43$$

No exemplo é $d_{\min} = 3$. Para este valor a relação de Plotkin (2.7) impõe que o número de bits de paridade tenha de ser, no mínimo, 3 (como é, de facto):

$$\begin{aligned} n - k &\geq 2 d_{\min} - 2 - \log_2 d_{\min} = \\ &= 2 \times 3 - 2 - 1{,}59 = 2{,}41 \end{aligned}$$

As relações de Plotkin (2.6) e (2.7) servem, portanto, para calcular o número mínimo de símbolos de paridade de um código de canal (n,k) se desejarmos que

possua uma determinada distância mínima, característica que está intimamente relacionada com a capacidade de correcção de erros *t* através de $d_{\min} \geq 2t+1$.

Na Tabela 2.1 foi usada a expressão (2.7) para calcular o número mínimo de bits de paridade de qualquer código binário, para valores da distância mínima entre 3 e 20.

Tabela 2.1

Número mínimo de bits de paridade em função da distância mínima

d_{min}	*n-k* mínimo	d_{min}	*n-k* mínimo
3	3	12	19
4	4	13	21
5	6	14	23
6	8	15	25
7	10	16	26
8	11	17	28
9	13	18	30
10	15	19	32
11	17	20	34

2.4.4. Outros limites

Na nossa bagagem de limites já levamos os de Singleton, Hamming e Plotkin. Qualquer um deles traça uma fronteira entre o possível e o impossível, entre o que pode existir e o que não existe de certeza. Outros há. Sem cuidarmos das respectivas demonstrações, que em alguns casos são complexas, vamos em seguida apresentar dois limites superiores (como os anteriores) e um limite inferior.

2.4.4.1. O limite de Griesmer

Imaginemos que queremos projectar um código (n,k) para o qual impomos a dimensão *k* e a distância mínima. Quanto pode valer *n*? O *limite de Griesmer* estabelece o seu valor mínimo:

$$n \geq \sum_{i=0}^{k-1} \left\lceil \frac{d_{\min}}{2^i} \right\rceil \tag{2.8}$$

Se, por exemplo, $k=12$ e $d_{\min}=7$, a aplicação da relação anterior conduz a

$$n \geq \sum_{i=0}^{11} \left\lceil \frac{7}{2^i} \right\rceil = 7 + \left\lceil \frac{7}{2} \right\rceil + \left\lceil \frac{7}{4} \right\rceil + \left\lceil \frac{7}{8} \right\rceil + \cdots =$$
$$= 7 + 4 + 2 + 9 \times 1 = 22$$

Poderia ser, por exemplo, $n=23$, originando o código (23,12) .

Em códigos não binários o limite de Griesmer exprime-se como

$$n \geq \sum_{i=0}^{k-1} \left\lceil \frac{d_{\min}}{q^i} \right\rceil \tag{2.9}$$

2.4.4.2. O limite de Elias

Elias obteve um outro limite superior da distância mínima. Na sua forma assimptótica (quando n é muito grande, $n \to \infty$) é expresso como [1]

$$\frac{d_{\min}}{n} \leq 2A(1-A), \tag{2.10}$$

para qualquer A que, pertencendo ao intervalo $0 \leq A \leq \frac{1}{2}$, satisfaça $1 - \Omega(A) > \frac{k}{n}$. A função Ω representa a conhecida entropia de uma fonte binária cujos símbolos têm probabilidades de ocorrência p e $1-p$,

$$\Omega(p) = -p\log_2 p - (1-p)\log_2(1-p),$$

representada graficamente na Fig. 2.12.

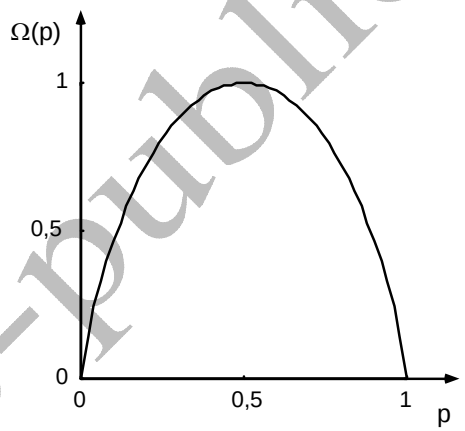


Fig. 2.12 A entropia de uma fonte binária.

A representação gráfica do limite de Elias é apresentada na Fig. 2.13.

2.4.4.3. O limite de Varshamov-Gilbert

Através do *limite de Varshamov-Gilbert* é possível prever a existência de códigos lineares com uma distância mínima superior ou igual a um certo valor, para cada par de valores de n e k. Os códigos nessas condições podem até nem ser conhecidos mas ficamos a saber que, pelo menos, é possível existirem, haja engenho e arte. Assim, sabe-se que existe um código linear (n,k) com uma distância mínima $d_{\min}$ igual ao maior d que satisfaz a desigualdade

$$\sum_{i=0}^{d-2}\binom{n-1}{i} < 2^{n-k} \tag{2.11}$$

O limite de Varshamov-Gilbert é um limite inferior e uma *prova de existência*: com ele ficámos a saber que existe um código *(n,k)* com este valor de $d_{\min}$, no mínimo.

A expressão anterior não é a única designada por limite de Varshamov-Gilbert (abreviemos em VG). Outras relações com a mesma designação surgem na literatura e são apresentadas no Anexo 2.1.

As expressões assimptóticas (*n* elevado) deste limite e do limite de Elias foram usadas na Fig. 2.13 para obter a distância normalizada $d_{\min}/n$ em função da taxa k/n. As cruzes referem-se a códigos de Hamming e ao código de Golay (23,12), todos eles com valores de *n* pequenos.

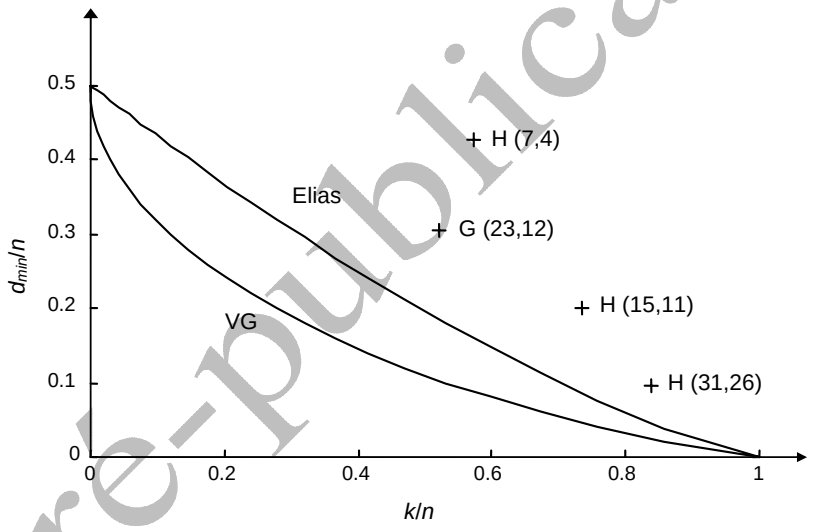


Fig. 2.13 Os limites de Elias e de Varshamov-Gilbert.

Exemplo 2.13

a) Qual é o valor máximo da distância mínima de um código binário (17,9)?

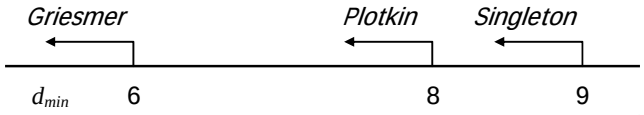
b) Poderá existir um código (31,11) de distância mínima 11?

R.: a) O limite de Singleton indica $d_{\min} \leq 9$ e o de Plotkin $d_{\min} \leq 8$. E o limite de Griesmer? É preciso satisfazer

$$n = 17 \geq \sum_{i=0}^{k-1} \left\lceil \frac{d_{\min}}{2^i} \right\rceil = \sum_{i=0}^{8} \left\lceil \frac{d_{\min}}{2^i} \right\rceil .$$

Fazendo as contas chega-se a $d_{\min} \leq 6$.

A "zona" dos códigos (17,9) impossíveis foi sendo cada vez mais bem delimitada, de Singleton para Plotkin e deste para Griesmer:



b) Segundo Singleton, é possível que o código exista: $d_{\min} \leq 21$; segundo Plotkin também:

$$d_{\min} \leq \left\lfloor \frac{n.2^{k-1}}{2^k - 1} \right\rfloor = 15 .$$

O limite de Hamming também é satisfeito para $t = \frac{d_{\min} - 1}{2} = 5$:

$$2^{31-11} = 1048576 \geq \sum_{i=0}^{5} \binom{31}{i} = 206368$$

O limite de Griesmer também é obedecido pois impõe que o comprimento *n* seja superior ou igual a 29. Quanto ao limite de Varshamov-Gilbert o maior valor de *d* que o satisfaz é 8:

$$\sum_{i=0}^{8-2} \binom{30}{i} = 768212 < 2^{20} = 1048576 .$$

Concluímos, então, que *existe* um código (31,11) de distância mínima 8, pelo menos, a qual não pode ultrapassar 15. Na realidade é conhecido o código BCH (31,11), com $t = 5$ e $d_{\min} = 11$.

Para concluir o assunto diremos que o limite VG é uma *prova de existência* (indica que um código com determinadas características existe) ao passo que os outros limites são uma *prova de inexistência* que identifica ou assinala os códigos não realizáveis.

2.5. Probabilidades de erro

Embora um código de blocos garanta que detecta todos os padrões de erro com $d_{\min} - 1$ ou menos erros, também consegue detectar uma grande fracção de padrões com $d_{\min}$ ou mais erros. Na verdade,

Com um código (n,k) é possível detectar $2^n - 2^k$ *padrões de erro.*

Porquê? Pelo seguinte: só os 2^k padrões iguais às palavras de código é que são indetectáveis (por converterem uma palavra válida noutra igualmente válida); todos os outros $2^n - 2^k$ vectores de erro são detectáveis.

Se o comprimento do código, n, for elevado 2^k é muito menor que o tamanho total da matriz padrão, 2^n, o que mostra que apenas uma pequena fracção dos padrões de erro não é detectada – a fracção constituída por aqueles vectores de erro que são iguais à primeira linha da matriz padrão.

O contrário se passa relativamente aos erros de descodificação. Aqui só são corrigíveis os 2^{n-k} vectores de erro que sejam iguais à primeira coluna da matriz padrão, isto é, que sejam iguais aos "coset leaders". Todos os restantes elementos da matriz são incorrigíveis, como se ilustra na Fig. 2.14.

Resumindo:

- *Um código (n,k) detecta* $2^n - 2^k$ *padrões de erro.*
- *Um código (n,k) só corrige* 2^{n-k} *padrões de erro.*

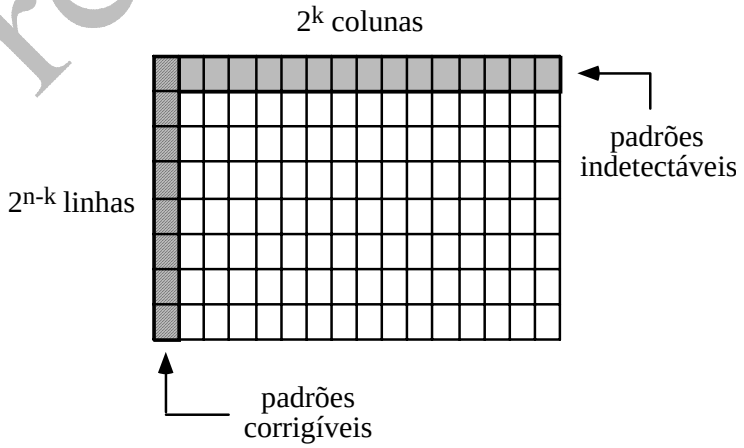


Fig. 2.14 Padrões de erro corrigíveis e indetectáveis.

Estas considerações conduzem-nos à determinação das probabilidades de erro não detectado, P_{end}, e não corrigido, P_{enc}.

2.5.1. Probabilidade de erro não detectado

Seja A_i o número de vectores de código com peso i. Ao conjunto dos números A_0, A_1, …, A_n chama-se *distribuição de pesos* do código, a qual naturalmente verifica a relação

$$\sum_{i=0}^{n} A_i = 2^k .$$

Existindo sempre a palavra nula será sempre $A_0 = 1$ e como não há palavras com peso inferior à distância mínima os valores de A_1 a $A_{d_{\min}-1}$ são nulos.

Ao polinómio

$$A(z) = \sum_{i=0}^{n} A_i z^i = 1 + A_1 z + A_2 z^2 + \ldots + A_n z^n$$

dá-se o nome de *polinómio enumerador de pesos*.

Este polinómio *A(z)* está relacionado com o seu homólogo *B(z)* do código dual (*n*,*n*-*k*) através da chamada *identidade de MacWilliams*:

$$A(z) = 2^{-(n-k)} (1+z)^n B\left(\frac{1-z}{1+z}\right) \tag{2.12}$$

em que

$$B(z) = \sum_{i=0}^{n} B_i z^i$$

e B_i representa o número de palavras do código dual com peso *i*. Assim, basta conhecer um polinómio para conhecer o outro. Por exemplo, se conhecermos *A(z)* o enumerador do dual vale

$$B\left(\frac{1-z}{1+z}\right) = 2^{n-k} (1+z)^{-n} A(z)$$

ou, fazendo $w = \frac{1-z}{1+z}$,

$$B(w) = 2^{-k}(1+w)^n A\left(\frac{1-w}{1+w}\right) \tag{2.13}$$

B(w) pode também ser expresso à custa da própria distribuição de pesos do código base:

$$B(w) = 2^{-k}\sum_{i=0}^{n} A_i (1-w)^i (1+w)^{n-i}$$

A identidade de MacWilliams é particularmente útil nos casos em que o dual tem um enumerador mais fácil de calcular que o do código base.

A probabilidade do descodificador não detectar a presença de erros pode ser calculada a partir da distribuição de pesos. Vejamos como:

Um erro não detectado ocorre apenas quando o vector de erro for idêntico a um dos $2^k - 1$ vectores de código não nulos. Então, havendo A_i palavras de código com peso i há A_i vectores de erro com i erros. Note-se de passagem que existem $\binom{n}{i}$ palavras de n bits com peso i mas deste total apenas A_i são palavras de código e só estas é que vão originar os erros não detectados.

A probabilidade de haver i erros, se a probabilidade de bit errado for p, é $p^i(1-p)^{n-i}$. Como há A_i possibilidades a probabilidade associada é $A_i p^i (1-p)^{n-i}$. A probabilidade de ocorrerem todos os padrões de erro que não são detectados, P_{end}, é, portanto,

$$P_{end} = \sum_{i=1}^{n} A_i p^i (1-p)^{n-i} \tag{2.14}$$

Como $A_1 = A_2 = \ldots = A_{d_{\min}-1} = 0$, podemos reescrever a expressão anterior com outros limites no somatório:

$$P_{end} = \sum_{i=d_{\min}}^{n} A_i p^i (1-p)^{n-i}$$

Os coeficientes A_i levam-nos ao polinómio enumerador de pesos *A(z)*: em primeiro lugar podemos escrever

$$P_{end} = \sum_{i=1}^{n} A_i p^i (1-p)^{n-i} = \sum_{i=1}^{n} A_i p^i \frac{(1-p)^n}{(1-p)^i} = (1-p)^n \sum_{i=1}^{n} \left(\frac{p}{1-p}\right)^i A_i \ .$$

Depois, fazendo a substituição $z = \frac{p}{1-p}$ no polinómio enumerador de pesos escrevemos

$$A(z) = A\left(\frac{p}{1-p}\right) = A_0 + A_1\left(\frac{p}{1-p}\right) + A_2\left(\frac{p}{1-p}\right)^2 + \ldots = 1 + \sum_{i=1}^{n}\left(\frac{p}{1-p}\right)^i A_i$$

e logo a seguir

$$P_{end} = (1-p)^n\left[A\left(\frac{p}{1-p}\right) - 1\right], \tag{2.15}$$

como queríamos.

Esta probabilidade de erro também pode ser calculada a partir do enumerador de pesos do código dual (recorrendo à identidade de MacWilliams) se este, *B(z)*, for mais fácil de determinar que *A(z)*. Conjugando (2.15) com a identidade de MacWilliams chegamos à expressão

$$P_{end} = 2^{-(n-k)} B(1-2p) - (1-p)^n, \tag{2.16}$$

em que $B(1-2p)$ é o valor do enumerador do código dual quando $z = 1-2p$:

$$B(1-2p) = \sum_{i=0}^{n} B_i (1-2p)^i$$

Em resumo, temos duas expressões para calcular a probabilidade de se falhar a detecção de erros com um código (*n*,*k*), uma usando *A(z)* e outra usando *B(z)*, e sendo *p* a probabilidade de erro do canal:

$$P_{end} = (1-p)^n\left[A\left(\frac{p}{1-p}\right) - 1\right]$$

$$P_{end} = 2^{-(n-k)} B(1-2p) - (1-p)^n$$

A última é muito mais fácil de calcular quando $n-k < k$, isto é, quando o número de bits de paridade é inferior ao número de bits de informação. Um exemplo vai mostrá-lo.

Exemplo 2.14

Um dado código (15,11) tem o seguinte polinómio enumerador de pesos:

$$A(z) = \frac{1}{16}\left[(1+z)^{15} + 15(1+z)^{7}(1-z)^{8}\right]$$

e é usado para detecção de erros num canal binário simétrico com probabilidade de erro $p = 10^{-4}$. Calcule a probabilidade dos erros não serem detectados.

R.: Não vamos usar A(z) para calcular a probabilidade pretendida uma vez que $n-k = 4 < k = 11$. Em vez desse polinómio usaremos o enumerador de pesos do código dual de (15,11), que é o código (15,4). Este tem apenas 16 palavras de código, ao passo que o primeiro tem 2048. A identidade de MacWilliams conduzir-nos-ia a

$$B(w) = 1 + 15w^{8}$$

Quer dizer que no código dual (15,4) há uma palavra com peso nulo e as restantes quinze têm peso 8.

A probabilidade de erro não detectado vale, portanto,

$$P_{end} = \frac{1}{16}\left[1+15(1-2p)^{8}\right] - (1-p)^{15} = 3{,}5.10^{-11}$$

2.5.2. Probabilidade de erro não corrigido

Um código *(n,k)* corrige todas as palavra recebidas com t erros e eventualmente algumas, mas só algumas, palavras com $t+1$ erros ou mais. Assim, a probabilidade de erro não corrigido é majorada por

$$P_{enc} \le \sum_{i=t+1}^{n} P(i,n) = \sum_{i=t+1}^{n} \binom{n}{i} p^{i}(1-p)^{n-i} \quad .$$

O valor máximo de P_{enc} atinge-se apenas com códigos perfeitos.

Podemos determinar com mais rigor a verdadeira probabilidade de erro P_{enc} pois um erro de descodificação ocorre **se e só se** o vector de erro não for um "coset leader". Retomando o raciocínio que nos conduziu à probabilidade de erro não detectado, se α_i for o número de "coset leaders" de peso i, a probabilidade do padrão de erro ser um "coset leader" é

$$\sum_{i=0}^{n} \alpha_i p^{i}(1-p)^{n-i}$$

Então a probabilidade que procuramos vale

$$P_{enc} = 1 - \sum_{i=0}^{n} \alpha_i p^i (1-p)^{n-i} \tag{2.17}$$

A probabilidade de erro não corrigido é normalmente muito maior que a probabilidade de erro não detectado: $P_{enc} >> P_{end}$.

Exemplo 2.15

Quais são as probabilidades de erro não detectado e não corrigido se usarmos o código (6,3) do Exemplo 2.11 num canal binário simétrico com probabilidade de erro $p = 10^{-3}$?

R.: *a) Determinação da probabilidade de erro não detectado*

Precisamos da distribuição de pesos do código. Consultando a lista de palavras de código obtemos:

$$\begin{aligned} A_o &= 1 \\ A_1 = A_2 = A_5 &= A_6 = 0 \\ A_3 &= 4 \\ A_4 &= 3 \end{aligned}$$

A probabilidade pretendida vale

$$P_{end} = \sum_{i=1}^{6} A_i p^i (1-p)^{6-i} = 4p^3(1-p)^3 + 3p^4(1-p)^2 \approx 4.10^{-9}$$

Em 1000 milhões de palavras de seis bits transmitidas há em média cerca de 4 palavras erradas que não são detectadas. Quatro em mil milhões!

b) Determinação da probabilidade de erro não corrigido

Agora precisamos dos pesos dos "coset leaders". Consultando a matriz padrão obtemos

$$\begin{aligned} \alpha_0 &= 1 \\ \alpha_1 &= 6 \\ \alpha_2 &= 1 \\ \alpha_3 = \alpha_4 = \alpha_5 &= \alpha_6 = 0 \end{aligned}$$

logo, a probabilidade de erro de descodificação vale

$$P_{enc} = 1 - (1-p)^6 - 6p(1-p)^5 - p^2(1-p)^4 = 1,4.10^{-5}$$

Com este código e este canal cometem-se erros de correcção cerca de catorze vezes em um milhão. E confirma-se que $P_{enc} >> P_{end}$.

2.6. Exemplos de códigos de blocos

2.6.1. Códigos de repetição

Uma forma muito simples de acrescentar redundância a uma sequência de informação é substituir cada bit por um conjunto de bits iguais. Um código destes chama-se código de repetição e já nos tinha aparecido antes, quando o assunto era pesos e medidas com esferas. Trata-se de um código *(n,*1*)* com apenas duas palavras de código, uma só com zeros e a outra só com uns, e taxa $1/n$. A distância mínima e o número de erros corrigíveis valem, respectivamente,:

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

Os códigos de repetição têm uma grande desvantagem: a sua taxa é muito baixa, especialmente se *n* for elevado.

A submatriz **P** tem a forma de um vector linha de $n-1$ elementos iguais a 1. Assim, a matriz geradora é igual ao vector linha

$$\mathbf{G} = \left[1 \mid 1 \quad 1 \quad \cdots \quad 1\right]$$

A matriz de verificação de paridade é igual a

$$\mathbf{H} = \left[\begin{array}{c|ccc} 1 & 1 & 0 & \cdots & 0 \\ 1 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & 0 & 0 & \cdots & 1 \end{array}\right]$$

Supondo que o comprimento *n* é ímpar as decisões de descodificação podem ser tomadas de acordo com uma *regra de maioria* (ou calculando a síndrome, claro!): se a palavra recebida tiver mais zeros que uns escolhe-se o bit “0”, caso contrário, escolhe-se o bit “1”.

Os códigos binários de repetição com n ímpar gozam de uma propriedade rara: são códigos perfeitos. Na verdade, o número de síndromes, 2^{n-1}, é igual ao número de padrões com erros desde 0 até $\frac{n-1}{2}=t$,

$$2^{n-1}=\sum_{i=0}^{t}\binom{n}{i},$$

e nenhum padrão com mais de t erros é corrigível.

O limite de Singleton é atingido com códigos de repetição: $d_{\min}=n-k+1=n$. Um código assim é designado de código MDS (de "Maximum-Distance-Separable").

Exemplo 2.16

Confirmemos que um código de repetição (7,1) é perfeito:

- Capacidade de correcção: $t=\frac{n-1}{2}=3$ (é um código corrector de erros triplos)
- Número de síndromes: $2^6=64$
- Número total de padrões corrigíveis (incluindo o padrão nulo): $1+7+\binom{7}{2}+\binom{7}{3}=1+7+21+35=64$

O número total de padrões corrigíveis é igual ao número de síndromes. Como não sobra mais "espaço" para padrões com mais de t erros concluímos que o código é perfeito.

2.6.2. Códigos de paridade

Nos códigos de paridade acrescenta-se um único bit de paridade ao bloco de k bits de informação de tal modo que o peso da palavra de código de $n=k+1$ bits seja par — código de paridade par — ou ímpar — código de paridade ímpar. Trata-se portanto de um código $(n,n-1)$, ou $(k+1,k)$, com taxa $\frac{k}{k+1}$. A distância mínima é sempre de dois, $d_{\min}=2$.

A maioria dos códigos de paridade em uso tem paridade par. Por omissão será esse o tipo de paridade que consideraremos.

Os códigos de paridade são códigos duais dos códigos de repetição. Por exemplo, o código dual de (6,1) é um código (6,5).

A submatriz **P** de um código de paridade par é uma matriz coluna de *k* linhas constituída só por bits "1":

$$\mathbf{P} = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \\ 1 \end{bmatrix}$$

Portanto, as matrizes **G** e **H** têm a forma genérica

$$\mathbf{G} = \left[\begin{array}{cccc|c} 1 & 0 & \cdots & 0 & 1 \\ 0 & 1 & \cdots & 0 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 1 \end{array}\right] \qquad \text{e} \qquad \mathbf{H} = \left[\begin{array}{cccc|c} 1 & 1 & \cdots & 1 & 1 \end{array}\right]$$

Um número par de erros não é detectado pois a paridade par ou ímpar não é alterada. Pelo contrário, um número ímpar de erros faz com que o vector de *n* bits resultante passe a ter paridade diferente da original, significando que a palavra está errada. Por exemplo, num código de paridade par um erro (que transforma "0" em "1" ou "1" em "0") origina um vector com um número ímpar de bits "1", vector que, por isso mesmo, não é uma palavra do código; logo, o erro é detectado. O erro não é, contudo, corrigido porque não é possível localizá-lo.

A detecção de erros num código de paridade par é muitíssimo simples: os bits z_i da palavra recebida $\mathbf{Z} = \left[z_1 z_2 \ldots z_i \ldots z_n\right]$ são somados em módulo 2. Se o resultado for nulo quer dizer que não houve erros (ou se os houve não foram detectados); se for 1 quer dizer que um ou mais bits estão errados:

$$S = z_1 + z_2 + \ldots + z_n = \begin{cases} 0 & \Rightarrow \quad \text{palavra sem erros} \\ 1 & \Rightarrow \quad \text{palavra com erros} \end{cases}$$

Um código de paridade não consegue corrigir erros e por isso é usado apenas para detecção.

Neste tipo de códigos temos, portanto, as três situações seguintes, consoante o número de erros:

a) *0 erros:* *palavra sem erros*
b) *Número par de erros:* *palavra com erros não detectados*
c) *Número ímpar de erros:* *palavra com erros detectados*

As probabilidades correspondentes são, respectivamente:

a) $P(\text{nenhum erro}) = P(0,n) = (1-\text{p})^n$

b) $P(\text{erro não detectado}) = P_{end} = \sum_{\substack{i=2 \\ i \text{ par}}}^{n} P(i,n) = \sum_{\substack{i=2 \\ i \text{ par}}}^{n} \binom{n}{i} p^i (1-p)^{n-i}$

c) $P(\text{erro detectado}) = \sum_{\substack{i=1 \\ i \text{ ímpar}}}^{n} P(i,n) = \sum_{\substack{i=1 \\ i \text{ ímpar}}}^{n} \binom{n}{i} p^i (1-p)^{n-i}$

A soma destas probabilidades é 1, logo a probabilidade de erro não detectado vale

$$P_{end} = 1 - \left[P(0,n) + \sum_{i \text{ ímpar}} P(i,n) \right]$$

Dado que, sendo p pequeno, a probabilidade de ocorrer um só erro é sempre muito maior que a probabilidade de ocorrerem 3, 5 ou mais erros, isto é, sendo

$$\sum_{i \text{ ímpar}} P(i,n) = P(1,n) + P(3,n) + P(5,n) + \ldots \approx P(1,n),$$

então podemos aproximar a expressão de P_{end} por

$$\begin{aligned} P_{end} &\approx 1 - [P(0,n) + P(1,n)] \qquad (\text{se } p << 1) \\ &= 1 - \left[(1-p)^n + np(1-p)^{n-1} \right] . \end{aligned}$$

Exemplo 2.17

Um código de paridade par (4,3) tem as seguintes palavras de código:

Bits de informação	Palavras de código
000	0000
001	0011
010	0101
011	0110
100	1001
101	1010
110	1100
111	1111

Imaginemos que não conhecíamos estas palavras. Facilmente se constata que um vector recebido 1011 não pertence ao código porque o seu peso é 3: para ser palavra do código o seu peso teria de ser par. Não conseguimos saber quantos erros foram cometidos (sabemos apenas que foi um número ímpar, 1 ou 3) nem onde ocorreram.

A matriz geradora deste código é

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

e a matriz de verificação de paridade é $\mathbf{H} = [1111]$.

Suponhamos que a probabilidade de bit errado no canal é $p = 10^{-3}$. As probabilidades de se cometerem ou não erros de detecção têm os seguintes valores:

$$P(\text{nenhum erro}) = \left(1 - 10^{-3}\right)^4 = 0{,}996006$$

$$P(\text{erro não detectado}) = P_{end} = \binom{4}{2}\left(10^{-3}\right)^2\left(1 - 10^{-3}\right)^2 + \binom{4}{4}\left(10^{-3}\right)^4 =$$

$$= 6 \times 10^{-6}$$

$$P(\text{erro detectado}) = \binom{4}{1} \times 10^{-3}\left(1 - 10^{-3}\right)^3 + \binom{4}{3}\left(10^{-3}\right)^3\left(1 - 10^{-3}\right) =$$

$$= 0{,}004$$

O valor (muito) aproximado de P_{end} poderia ter sido calculado através de

$$P_{end} \approx 1 - [P(0,4) + P(1,4)] =$$

$$= 1 - [0{,}996006 + 0{,}00399] = 6.10^{-6}$$

Em comparação com as probabilidades dos acontecimentos desejáveis (descodificação correcta e detecção de erros) a probabilidade dos erros não serem detectados é bastante baixa: apenas $P_{end} = 6.10^{-6}$.

2.6.3. Códigos de Hamming

Em 1950 R. W. Hamming propôs uma família de códigos que ficou conhecida pelo seu nome.

Os códigos de Hamming são códigos correctores de erros simples ($t = 1$) ou detectores de erros duplos ($l = 2$). Obedecem às seguintes condições:

- Número de bits de paridade: $n - k \geq 3$
- Comprimento: $n = 2^{n-k} - 1$

São, portanto, códigos $(2^{n-k} - 1, k)$. Exemplos incluem as combinações (7,4), (15,11) e (31,26).

A distância mínima de qualquer código de Hamming é sempre 3, independentemente da sua dimensão e comprimento.

Como é que devem ser as matrizes **G** e **H**? Bem, se os códigos corrigem todos os erros simples quer dizer que corrigem todos os padrões com um erro; estes, que ao todo são $\binom{n}{1} = n$, deverão originar *n* síndromes não nulas e distintas. Como existem 2^{n-k} síndromes, uma das quais é um vector nulo, verificamos que, efectivamente, $2^{n-k} = n + 1$. Imaginemos então que o vector de erro é $\mathbf{E} = [e_1 e_2 \ldots e_j \ldots e_n]$ e que $\left[\mathbf{h}_j\right]$ representa a linha de ordem *j* da matriz $\mathbf{H}^T$:

$$\mathbf{H}^T = \begin{bmatrix} \mathbf{h}_1 \\ \vdots \\ \mathbf{h}_j \\ \vdots \\ \mathbf{h}_n \end{bmatrix}$$

Se só houver um erro na posição *j* então $e_{i \neq j} = 0$ e $e_j = 1$ pelo que o padrão correspondente é $\mathbf{E} = [0 \ldots 010 \ldots 0]$. A sua síndrome é igual a linha $\mathbf{h}_j$ de $\mathbf{H}^T$:

$$\mathbf{E}\mathbf{H}^T = [0...010...0]\mathbf{H}^T = [\mathbf{h}_j]$$

O que é que tinha sido dito antes? Era que as síndromes deveriam ser diferentes umas das outras. Ora concluímos agora que as síndromes de erros simples (os que são corrigíveis) são, afinal, iguais às linhas de $\mathbf{H}^T$! Conclusão: nos códigos de Hamming as linhas de $\mathbf{H}^T$ (ou as colunas de $\mathbf{H}$) têm de ser não nulas e todas diferentes umas das outras. É claro que havendo $n = 2^{n-k} - 1$ linhas (ou colunas) não nulas de $n-k$ bits cada uma, basta calcular todas as combinações possíveis de $n-k$ bits (excluindo a combinação nula).

Num código corrector de erros simples concluímos portanto que:

> *Se a síndrome for igual à linha de ordem j da matriz $\mathbf{H}^T$ então houve um erro no j-ésimo bit da palavra de código recebida.*

Uma última observação: as síndromes esgotam-se com todos os padrões de um único erro, não sobrando mais nenhuma para padrões com mais erros. Logo, os códigos binários de Hamming são códigos perfeitos.

2.6.3.1. Distribuição de pesos

A distribuição A_i de pesos das palavras dos códigos de Hamming é conhecida. É dada pelos coeficientes do respectivo polinómio enumerador de pesos:

$$A(z) = \sum_{i=0}^{n} A_i z^i = \frac{1}{n+1}\left[(1+z)^n + n(1+z)^{(n-1)/2}(1-z)^{(n+1)/2}\right]$$

O desenvolvimento do polinómio conduz aos diversos valores de A_i. Sweeney [2] apresenta a seguinte expressão:

$$A_i = \begin{cases} \dfrac{\binom{n}{i} + n(-1)^{i/2}\binom{(n-1)/2}{i/2}}{n+1} & i \text{ par} \\ \dfrac{\binom{n}{i} + n(-1)^{(i+1)/2}\binom{(n-1)/2}{(i-1)/2}}{n+1} & i \text{ ímpar} \end{cases}$$

A partir desta expressão foi construída a Tabela 2.2, onde é apresentada a distribuição dos códigos de Hamming (7,4), (15,11) e (31,26).

Tabela 2.2

Distribuição de pesos de códigos de Hamming

i	$A_i = A_{n-i}$		
	(7,4)	(15,11)	(31,26)
0	1	1	1
1	0	0	0
2	0	0	0
3	7	35	155
4	7	105	1085
5	0	168	5208
6	0	280	22568
7	1	435	82615
8		435	247845
9		280	628680
10		168	1383096
11		105	2648919
12		35	4414865
13		0	6440560
14		0	8280720
15		1	9398115

Embora na Tabela 2.2 não estejam indicados os pesos de todas as palavras do código (31,26) todos são calculáveis pois repetem-se de acordo com $A_i = A_{n-i}$. Assim, podemos dizer que existem 628680 palavras de peso 9 e outras tantas de peso 22.

Não é praticável escrever a expressão concreta do polinómio enumerador de pesos dos códigos, excepto se o seu comprimento for pequeno. O caso mais simples é, evidentemente, o do código (7,4):

$$(7,4): \quad A(z) = 1 + 7z^3 + 7z^4 + z^7$$

2.6.3.2. Duais dos códigos de Hamming

Os códigos duais dos códigos de Hamming têm dimensão $n-k$ e o mesmo comprimento, $2^{n-k}-1$:

$$\text{Hamming } (2^{n-k}-1, k) \quad \rightarrow \quad \text{Dual de Hamming } (2^{n-k}-1, n-k)$$

Veremos no Capítulo 3 que estes códigos duais se chamam *códigos de comprimento máximo* (também há quem lhes chame códigos *simplex*). A sua distribuição de pesos é simplesmente esta: existe uma palavra com peso nulo (como sempre!) e todas as restantes $n = 2^{n-k}-1$ têm peso 2^{n-k-1} igual à distância mínima. Sendo assim, o polinómio enumerador de pesos é igual a

$$B(z) = 1 + \left(2^{n-k}-1\right) z^{2^{n-k-1}} = 1 + nz^{2^{n-k-1}},$$

como se poderia comprovar através da igualdade de MacWilliams.

Na Tabela 2.3 estão indicadas as distâncias mínimas de alguns códigos de Hamming e seus duais.

Tabela 2.3

Alguns códigos de Hamming e seus duais

Código de Hamming	$d_{\min}$	Código dual	$d_{\min}$
(7,4)	3	(7,3)	4
(15,11)	3	(15,4)	8
(31,26)	3	(31,5)	16
(63,57)	3	(63,6)	32
(127,120)	3	(127,7)	64

2.6.3.3. Probabilidade de erro não detectado

A probabilidade de os erros não serem detectados com um código (n,k) qualquer é igual às equações (2.15) e (2.16), como se viu. Com códigos de Hamming é normalmente mais fácil calcular essa probabilidade usando o enumerador de pesos do código dual. De uma maneira ou de outra a expressão final é

$$P_{end} = 2^{-(n-k)}\left[1+n(1-2p)^{2^{n-k-1}}\right]-(1-p)^n$$

Exemplo 2.18

A transposta da matriz de verificação de paridade de um código de Hamming (7,4) é

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

Cada linha é uma combinação diferente de 3 bits. Poderíamos ter colocado essas combinações por outra ordem. A que foi escolhida obedeceu simplesmente à restrição de desejarmos um código sistemático com os bits de paridade no fim das

palavras e é por isso que o conjunto das últimas três linhas de $\mathbf{H}^T$ representa uma matriz identidade.

a) Qual terá sido a palavra enviada se se receber o vector **Z** = [1011100]? Conseguem-se corrigir erros duplos como, por exemplo, **E** = [1000010]?

b) Se for utilizado um canal binário simétrico com uma probabilidade de erro de 10^{-5} qual é a probabilidade dos erros não serem detectados pelo código?

R.: a) Se **Z** = [1011100] então $\mathbf{S} = \mathbf{Z}\mathbf{H}^T = [100]$, que é a 5ª linha de $\mathbf{H}^T$. Concluímos que **Z** contém um erro no quinto bit, isto é, **E** = [0000100]. A palavra enviada foi **Y** = **Z**+**E** = [1011000].

Quanto à segunda questão, a síndrome de **E** = [1000010] é **S**=[111], que corresponde à 2ª linha de $\mathbf{H}^T$. Seríamos levados a pensar que $\hat{\mathbf{E}}$ = [0100000], quando nós sabemos que o vector de erro não é esse mas sim [1000010]. Note-se, no entanto, que a síndrome é igual à soma das linhas 1 e 6 de $\mathbf{H}^T$ (pois os erros tinham ocorrido nas posições 1 e 6). Enfim, estamos a confirmar que o código de Hamming não consegue corrigir erros duplos.

b) A probabilidade de erro desejada é muito fácil de calcular:

$$P_{end} = 2^{-(n-k)}\left[1+n(1-2p)^{2^{n-k-1}}\right] - (1-p)^n =$$
$$= \frac{1}{8}\left[1+7\left(1-2\times 10^{-5}\right)^4\right] - \left(1-10^{-5}\right)^7 = 6{,}7.10^{-15}$$

Com este canal e este código a probabilidade de ocorrerem erros que não sejam detectados é extremamente diminuta.

2.6.3.4. Representação do código de Hamming (7,4) através de diagramas de Venn

Um código de Hamming (7,4) pode ser caracterizado através de um diagrama de Venn, o que nos permite codificá lo e descodificá lo muito facilmente. Como fazer?

- Desenham se três circunferências que se intersectem, colocando duas em cima e uma por baixo. Assim serão criadas sete áreas, numeradas de 1 a 7 na figura abaixo*.
- Colocam-se os quatro bits da mensagem nas áreas de intersecção 1, 2, 3 e 4.
- Os três bits de paridade são colocados nas áreas 5, 6 e 7 de modo que em cada um dos círculos exista um número par de "uns" (ou seja, a soma mod 2 dos bits dentro de cada círculo é 0).

*Esta numeração corresponde a um exemplo; ver explicação a seguir.

O diagrama de Venn mostrado na Fig. 2.15 corresponde à mensagem 1010 aplicada a um codificador (7,4) com as equações de paridade

$$\begin{cases} c_1 = x_1 \oplus x_2 \oplus x_3 \\ c_2 = x_2 \oplus x_3 \oplus x_4 \\ c_3 = x_1 \oplus x_2 \oplus x_4 \end{cases}$$

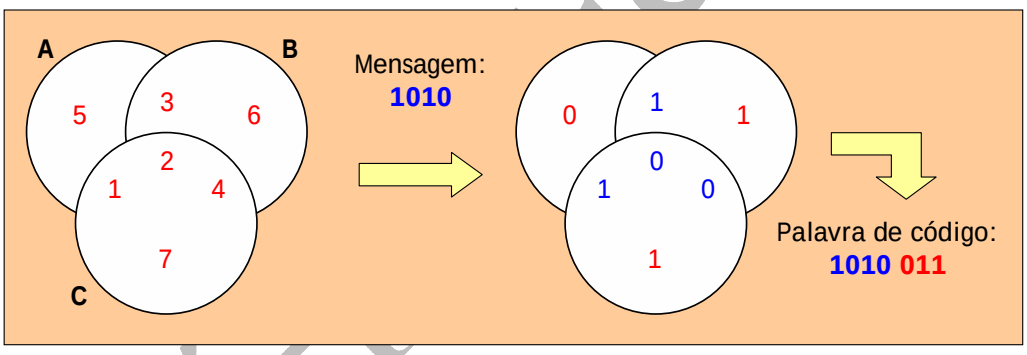


Fig. 2.15 Código de Hamming (7,4): codificação com diagrama de Venn.

A numeração dos círculos A, B e C no diagrama de Venn depende destas equações de paridade, e vice-versa. Por exemplo, como $c_1 = x_1 \oplus x_2 \oplus x_3$ e c_1 é o quinto bit da palavra de código, a área número 5 está no mesmo círculo (A) das áreas com os números 1, 2 e 3. Repare-se que o bit x_2 é comum às três equações de paridade pelo que o número 2 deve ser atribuído à área intersectada pelos três círculos.

Este diagrama serve também para fazer a correcção de erros, como se mostra na Fig. 2.16 com um exemplo no qual o primeiro bit da palavra recebida está errado: como a soma (em mod 2) dos bits dos círculos A e C é 1 concluímos que o bit errado não está em B e pertence à restante intersecção dos círculos A e C, isto é, o erro está em $A \cap C \cap \overline{B}$.

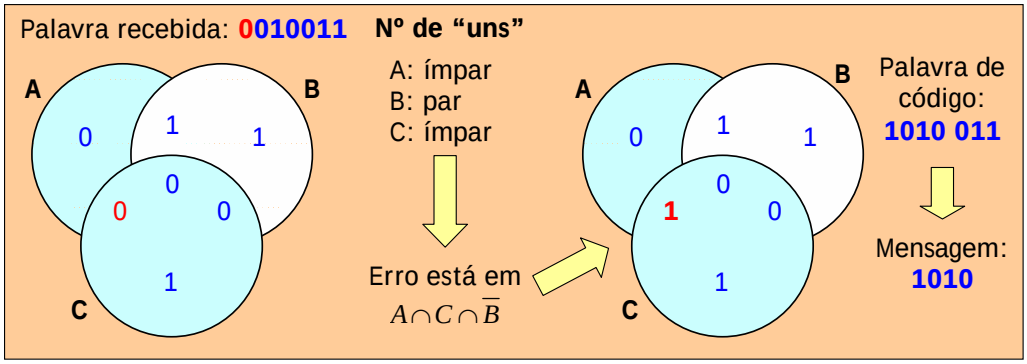


Fig. 2.16 Código de Hamming (7,4): descodificação com diagrama de Venn.

Se a palavra recebida contiver dois erros o código de Hamming não os corrige, como sabemos. O diagrama de Venn da Fig. 2.17 mostra-o claramente para um exemplo em que foi gerada a palavra de código 1010011 e a palavra 0010010 foi recebida. Como se vê, passámos de dois bits errados para três.

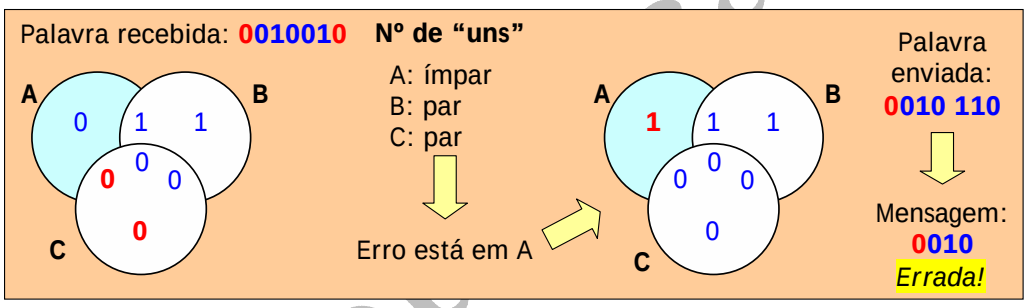


Fig. 2.17 Código de Hamming (7,4) e dois erros: descodificação incorrecta.

O diagrama de Venn pode ser também aplicado ao código de Hamming aumentado[5] (8,4) considerando uma oitava área adicional (tudo o que ficar de fora dos três círculos) para o quarto bit de paridade.

Curiosidade histórica

A aplicação dos diagramas de Venn aos códigos de Hamming (7,4) deve-se ao americano Robert J. McEliece, um proeminente e influente membro da comunidade académica da Codificação e Teoria da Informação. É professor no California Institute of Technology (CalTech), E. U. A.

[5] Os códigos aumentados serão tratados mais adiante.

2.6.4. Código de Golay

O código binário de Golay foi proposto pelo suíço Marcel J. Golay em 1949. É um código (23,12), de dimensão 12, comprimento 23 e distância mínima 7. Como atinge o limite de Hamming,

$$2^{11} = \sum_{i=0}^{3} \binom{23}{i},$$

é um código perfeito corrector de erros triplos ($t = 3$). A distribuição de pesos consta da tabela seguinte.

Peso i	0	7	8	11	12	15	16	23
A_i	1	253	506	1288	1288	506	253	1

2.7. Modificações nos códigos

Os códigos de blocos são por vezes modificados alterando a sua dimensão e comprimento. Há várias razões para o fazer: umas vezes isso faz-se porque os valores de n e k não são os mais convenientes (não servem para preencher uma trama de dados de tamanho normalizado, por exemplo); noutras deseja-se simplesmente melhorar o desempenho. E assim se obtêm os códigos aumentados e encurtados.

2.7.1. Códigos aumentados

Os códigos aumentados obtêm-se de um código (n,k) acrescentando um bit de paridade aos $n-k$ bits de paridade já existentes:

$$(n,k) \xrightarrow{n-k \uparrow} (n+1,k)$$

O bit de paridade a acrescentar tem um valor que garanta paridade par na nova palavra de código. Dessa maneira:

- se as palavras de código antigas tiverem peso ímpar o peso é incrementado.
- se as palavras de código antigas tiverem peso par o peso mantém-se.

Quer isto dizer que se o código original (n,k) tiver distância mínima par, a distância mínima do código aumentado $(n+1,k)$ não se altera mas se o original tiver distância mínima ímpar, esta é incrementada de 1.

A matriz de verificação de paridade, $\mathbf{H}_a$, de um código aumentado obtém-se da matriz **H** do código (n,k) acrescentando-lhe primeiro uma coluna só com zeros e depois uma linha só com uns:

$$\mathbf{H}_a = \left[\begin{array}{ccc|c} & & & 0 \\ & \mathbf{H} & & 0 \\ & & & \vdots \\ & & & 0 \\ \hline 1 & 1 & \cdots\ 1 & 1 \end{array}\right]$$

Exemplo 2.19

P.: Obtenha as palavras de um código aumentado (6,3) a partir do código (5,3) que tem as seguintes palavras:

00000 10011 01010 00101 11001 10110 01111 11100

R.: Basta acrescentar um bit de paridade de modo que o peso de cada nova palavra seja par:

(5,3): 00000 10011 01010 00101 11001 10110 01111 11100

(6,3): 000000 100111 010100 001010 110011 101101 011110 111001

De (5,3) para (6,3) a distância mínima não aumentou porque era par ($d_{\min} = 2$).

2.7.1.1. Códigos de Hamming aumentados

A partir de qualquer código de Hamming podemos obter o respectivo código aumentado acrescentando um bit de paridade de maneira a que o peso da nova palavra seja par. Assim, a distância mínima, que era ímpar, passa de 3 para 4. Teremos então:

- Número de bits de paridade: $n - k \geq 3$
- Comprimento: $n = 2^{n-k-1}$

Eis dois exemplos de códigos de Hamming aumentados $(2^{n-k-1}, k)$:

$$(7,4),\ d_{\min} = 3 \ \rightarrow\ (8,4),\ d_{\min} = 4$$

$$(15,11),\ d_{\min} = 3 \ \rightarrow\ (16,11),\ d_{\min} = 4$$

A distribuição de pesos dos códigos de Hamming aumentados é dada, segundo [3], pelos coeficientes do seu polinómio enumerador de pesos

$$A(z) = \frac{1}{2n}\left[(1+z)^n + (1-z)^n + 2(n-1)\left(1-z^2\right)^{n/2} \right]$$

Exemplo 2.20

A distribuição de pesos do código de Hamming (7,4) foi calculada antes. No código (8,4) o bit extra provoca paridade par e origina uma nova distribuição. As duas formam a tabela seguinte:

Peso i	0	1	2	3	4	5	6	7	8
A_i (7,4)	1	0	0	7	7	0	0	1	—
A_i (8,4)	1	0	0	0	14	0	0	0	1

A distância mínima aumentou, naturalmente, de 3 para 4. Como já conhecíamos a distribuição de pesos do código (7,4) não precisámos de usar o polinómio enumerador de pesos do código aumentado (8,4).

Já que aqui temos as distribuições de pesos vamos calcular as probabilidades de erro não detectado quando o canal através do qual se transmitem os dois códigos apresenta uma probabilidade de bit errado de $p = 10^{-3}$:

- Código de Hamming (7,4):

$$P_{end} = \sum_{i=1}^{7} A_i p^i (1-p)^{7-i} = 7p^3(1-p)^4 + 7p^4(1-p)^3 + p^7 \approx 7.10^{-9}$$

- Código de Hamming aumentado (8,4):

$$P_{end} = \sum_{i=1}^{8} A_i p^i (1-p)^{8-i} = 14p^4(1-p)^3 + p^8 \approx 1{,}4.10^{-11}$$

Melhoria notável!

2.7.1.2. Código de Golay aumentado

O código de Golay (23,12) também é usado na sua versão aumentada:

$$(23,12), d_{\min} = 7 \quad \rightarrow \quad (24,12), d_{\min} = 8$$

Uma das razões para que seja usado é o aumento da distância mínima, como de costume — $d_{\min}$ aumenta de 7 para 8; outra é o facto da taxa do código passar de 12/23 para o atraente valor de 12/24 = 0,5, o que pode facilitar a realização de relógios de sincronismo.

A distribuição de pesos do código de Golay (24,12) obtém-se da distribuição primitiva sabendo que todas as novas palavras têm de ter paridade par. O resultado está na tabela seguinte:

Peso i	0	7	8	11	12	15	16	23	24
A_i (23,12)	1	253	506	1288	1288	506	253	1	—
A_i (24,12)	1	0	759	0	2576	0	759	0	1

O código (23,12) é um código perfeito e corrige todos os erros triplos. O código de Golay aumentado, (24,12), não é perfeito mas corrige todos os erros triplos e alguns (mas não todos) erros quádruplos.

2.7.2. Códigos encurtados

Os códigos encurtados obtêm-se dos originais removendo um ou mais bits de informação. O comprimento do código fica mais pequeno mas o número de bits de paridade mantém-se.

Pegando nas 2^k palavras do código original o que se faz é forçar a zero alguns (l) bits de informação e depois removê-los (não se transmitem). As palavras alteradas (aquelas em que os bits mudaram) desaparecem e ficamos com a estrutura

$$(n,k) \xrightarrow{k\downarrow} (n-l,k-l)$$

No novo código passa a haver 2^{k-l} palavras de código em vez das 2^k originais (se $l=1$, passa a haver metade das palavras) e a distância mínima é a mesma ou superior, dependendo dos bits removidos.

> *A distância mínima não poderia ser menor porque as palavras que restam são um subconjunto das que já existiam removendo bits nulos e por isso têm o mesmo peso.*

A remoção dos bits de ordem i (com $i=1,2,\ldots,k$) é equivalente a eliminar as linhas e as colunas de ordem i da matriz geradora e a eliminar as colunas de ordem i da matriz de verificação de paridade; esta permanece com o mesmo número de linhas

porque o número de bits de paridade não foi alterado. Por este motivo o número de padrões de erro corrigíveis nos códigos encurtados é o mesmo que nos códigos donde derivam, e o número *t* de erros corrigíveis também, pois o número de "coset leaders" da matriz padrão permanece constante.

Não existe regra geral que indique quais são os bits de informação que devem ser removidos. O que normalmente se faz, até porque é simples e conveniente, é remover um ou mais bits de informação consecutivos.

Exemplo 2.21

A matriz geradora de um código de Hamming (7,4) é a seguinte:

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

Determine o código de Hamming encurtado (6, 3) removendo o terceiro bit de informação.

R.: A nova matriz geradora obtém-se da primeira eliminando a 3ª linha e a 3ª coluna:

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

A nova matriz de verificação de paridade obtém-se da original removendo-lhe a terceira coluna:

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

A tabela seguinte mostra como encontrar as palavras do código encurtado a partir do código base.

Palavras do código (7,4)			**Palavras do código (6,3)**
Original	3º bit nulo		Remoção do 3º bit
0 0 0 0 0 0 0	0 0 **0** 0 0 0 0		0 0 0 0 0 0
0 0 0 1 1 1 1	0 0 **0** 1 1 1 1		0 0 1 1 1 1
0 0 1 0 0 1 1	0 0 **0** 0 0 1 1	Alterada	—
0 0 1 1 1 0 0	0 0 **0** 1 1 0 0	Alterada	—
0 1 0 0 1 0 1	0 1 **0** 0 1 0 1		0 1 0 1 0 1
0 1 0 1 0 1 0	0 1 **0** 1 0 1 0		0 1 1 0 1 0
0 1 1 0 1 1 0	0 1 **0** 0 1 1 0	Alterada	—
0 1 1 1 0 0 1	0 1 **0** 1 0 0 1	Alterada	—
1 0 0 0 1 1 0	1 0 **0** 0 1 1 0		1 0 0 1 1 0
1 0 0 1 0 0 1	1 0 **0** 1 0 0 1		1 0 1 0 0 1
1 0 1 0 1 0 1	1 0 **0** 0 1 0 1	Alterada	—
1 0 1 1 0 1 0	1 0 **0** 1 0 1 0	Alterada	—
1 1 0 0 0 1 1	1 1 **0** 0 0 1 1		1 1 0 0 1 1
1 1 0 1 1 0 0	1 1 **0** 1 1 0 0		1 1 1 1 0 0
1 1 1 0 0 0 0	1 1 **0** 0 0 0 0	Alterada	—
1 1 1 1 1 1 1	1 1 **0** 1 1 1 1	Alterada	—
$d_{min} = 3$			$d_{min} = 3$

Repare-se que a distância mínima não se alterou de um código para o outro.

Vamos ver com outro exemplo que a distância mínima pode aumentar com um código encurtado.

Exemplo 2.22

A matriz geradora de um código de Hamming (15,11) com distância mínima 3 é a seguinte:

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

Se se eliminarem as linhas de **G** que em **P** têm peso par (1, 2, 3, 5, 6, 7 e 11) e também as colunas com o mesmo ordinal obtemos a matriz geradora de um código encurtado (8,4) com menos sete bits de informação que o anterior:

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

A transposta da matriz de verificação de paridade deste código encurtado é

$$\mathbf{H}'^{T} = \begin{bmatrix} 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

a) Verifique que o encurtamento produziu um código com uma distância mínima maior que a original.

b) Mostre que o código encurtado (8,4) corrige erros simples e ao mesmo tempo detecta erros duplos.

R.: a) H' é a matriz de verificação de paridade do código (8,4) que aparece no Exemplo 2.5. Aí se viu que a distância mínima de tal código é 4, confirmando que, de facto, a distância mínima aumentou com este encurtamento.

(Vamos recordar: H'^{T} não tem linhas nulas nem iguais e todas têm peso ímpar, logo $d_{\min} \geq 4$; basta encontrar quatro linhas cuja soma seja zero.)

b) Uma palavra recebida $\boldsymbol{Z} = [z_1 z_2 z_3 z_4 z_5 z_6 z_7 z_8]$ origina a síndrome $\boldsymbol{S} = \boldsymbol{Z}\boldsymbol{H}'^{T}$ calculada através das quatro equações

$$\begin{aligned} s_1 &= z_1 + z_2 + z_3 + z_5 \\ s_2 &= z_1 + z_3 + z_4 + z_6 \\ s_3 &= z_2 + z_3 + z_4 + z_7 \\ s_4 &= z_1 + z_2 + z_4 + z_8 \end{aligned}$$

Se $\boldsymbol{E} = [e_1 e_2 e_3 e_4 e_5 e_6 e_7 e_8]$ for um padrão de erros genérico também podemos escrever

$$s_1 = e_1 + e_2 + e_3 + e_5$$
$$s_2 = e_1 + e_3 + e_4 + e_6$$
$$s_3 = e_2 + e_3 + e_4 + e_7$$
$$s_4 = e_1 + e_2 + e_4 + e_8$$

dado que $\boldsymbol{S} = \boldsymbol{Z}\boldsymbol{H}'^{T} = \boldsymbol{E}\boldsymbol{H}'^{T}$. Constata-se neste sistema de equações que, se forem diferentes de zero, cada um dos bits e_1, e_2, e_3 e e_4 (correspondentes a erros nos bits de informação) aparece no cálculo de três dos quatro elementos de **S** e que os bits e_5, e_6, e_7 e e_8 (correspondentes a erros nos bits de paridade) aparecem apenas uma vez cada. Daqui podemos concluir que:

- Se tiver havido um erro o peso de **S** é ímpar (1 ou 3)
- Se tiver havido dois erros o peso de **S** é par (2 ou 4)

Por exemplo, se o padrão de erro for $\boldsymbol{E}_1 = [00100000]$ (um só erro) então a síndrome é $\boldsymbol{S}_1 = [1110]$ (peso ímpar) e se for $\boldsymbol{E}_2 = [00001001]$ (dois erros) então $\boldsymbol{S}_2 = [1001]$ (peso par). Logo, podemos instituir neste código a seguinte regra de descodificação:

1) Se a síndrome for nula assume-se que não houve erro na palavra recebida;
2) Sendo não nula, se a síndrome tiver peso ímpar assume-se que houve um erro, corrigível;
3) Sendo não nula, se a síndrome tiver peso par assume-se que o padrão de erro é incorrigível mas detectável.

Em conclusão, o código corrige erros simples ($t = 1$) e ao mesmo tempo detecta erros duplos ($l = 2$). Estes valores confirmam que quando um código consegue fazer correcção e detecção simultâneas, como este, então $d_{\min} \geq t + l + 1$, com $l > t$.

Anexo 2.1
Os limites de Varshamov-Gilbert

O limite de Varshamov-Gilbert indica que existe um código linear (n,k) com uma distância mínima $d_{\min}$ igual ao maior d que satisfaz a desigualdade

$$\sum_{i=0}^{d-2}\binom{n-1}{i} < 2^{n-k}$$

Na literatura podem ser encontradas outras expressões semelhantes referidas como limite de Varshamov-Gilbert, como, por exemplo, em [2] [4] [5]. No que se segue tomou-se como referência o livro de Peter Sweeney [2].

Assim, sabe-se que existe um código (n,k) que satisfaz

$$\sum_{i=0}^{d_{\min}-2}\binom{n}{i}(q-1)^i < q^{n-k} \le \sum_{i=0}^{d_{\min}-1}\binom{n}{i}(q-1)^i \qquad \text{(A2.1)}$$

onde q é o tamanho do alfabeto. Tratando-se de códigos binários estas expressões simplificam-se em

$$\sum_{i=0}^{d_{\min}-2}\binom{n}{i} < 2^{n-k} \le \sum_{i=0}^{d_{\min}-1}\binom{n}{i} \qquad \text{(A2.2)}$$

A primeira desigualdade indica-nos o valor máximo de $d_{\min}$ para um dado par (n,k); a segunda diz-nos que existe um código (n,k) cuja distância mínima é igual ou superior ao limite inferior calculado.

Deduz-se do limite de Varshamov-Gilbert que, quando $n \to \infty$, podem existir códigos cuja distância mínima respeita

$$\frac{d_{\min}}{n} \ge \alpha, \qquad \text{(A2.3)}$$

em que α é $0 \le \alpha \le \frac{1}{2}$ e satisfaz $\frac{k}{n} = 1-\Omega(\alpha)$. Se $\frac{d_{\min}}{n} \ge \alpha$ e $\frac{d_{\min}}{n} < 1/2$ então $\Omega(d_{\min}/n) \ge \Omega(\alpha)$ (pois a função $\Omega(x)$ é crescente para $x < 0{,}5$) e portanto pode escrever-se que a taxa do código satisfaz

$$\frac{k}{n} \ge 1-\Omega\left(\frac{d_{\min}}{n}\right), \qquad \text{com } d_{\min} < \frac{n}{2}$$

Comparando o limite de Elias da expressão (2.10) com o limite de Varshamov-Gilbert da expressão (A2.3), quando n é muito grande, constatamos que A e α são, afinal, a mesma coisa, pelo que podemos escrever

$$\alpha \le \frac{d_{\min}}{n} \le 2\alpha(1-\alpha), \qquad \text{com } \frac{k}{n} = 1-\Omega(\alpha).$$

Na figura seguinte estão representadas as curvas dos limites assimptóticos de Elias e Varshamov-Gilbert juntamente com os valores concretos obtidos com alguns códigos de Hamming e com o código de Golay (23,12).

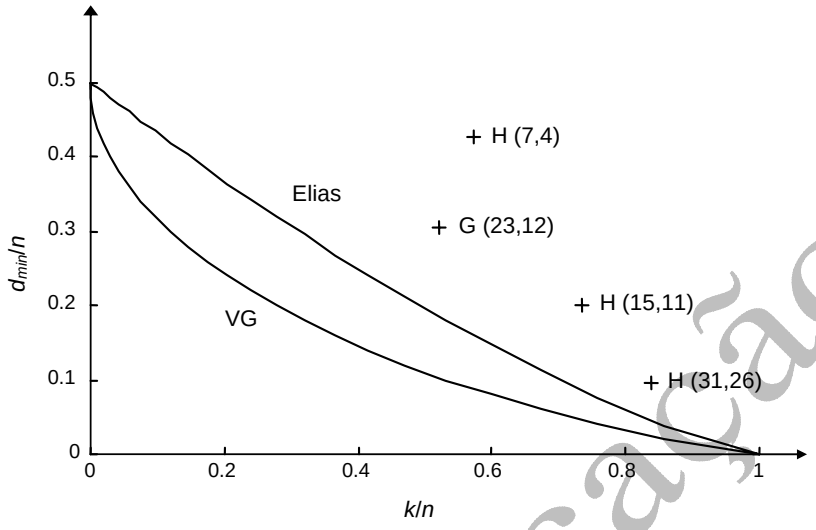


Fig. A2.1. Os limites de Elias e de Varshamov-Gilbert.

2.8. Bibliografia

[1] R. E. Blahut, *Theory and Practice of Error Control Codes*, Addison-Wesley Publishing Company, 1983.

[2] P. Sweeney, *Error Control Coding: An Introduction*, Prentice-Hall, 1991.

[3] A. M. Michelson e A. H. Levesque, *Error-Control Techniques for Digital Communications*, John Wiley & Sons, 1985.

[4] R. D. Gitlin, J. F. Hayes e S. B. Weinstein, *Data Communications Principles*, Plenum Press, 1992.

[5] S. Lin e D. J. Costello, Jr., *Error Control Coding: Fundamentals and Applications*, Prentice-Hall, 1983.

Outras referências

- M. Purser, *Introduction to Error-Correcting Codes*, Artech House, 1995.
- J. G. Proakis, *Digital Communications*, 2ª edição, McGraw-Hill, 1989.

- B. Sklar, *Digital Communications (Fundamentals and Applications)*, Prentice-Hall, 2ª edição, 2000.
- S. Haykin, *Digital Communications*, 4ª edição, J. Wiley, 2000.

Exercícios do Capítulo 2