UNIVERSIDADE ESTADUAL DE CAMPINAS

Faculdade de Engenharia Elétrica e de Computação

Departamento de Comunicações

CÓDIGOS CORRETORES DE ERRO COM BOAS PROPRIEDADES DE CODIFICAÇÃO DE LINHA

Evelio Martín García Fernández

Orientador

Prof. Dr. Renato Baldini Filho

Dissertação apresentada à Faculdade de Engenharia Elétrica e de Computação da Universidade Estadual de Campinas como parte dos requisitos exigidos para obtenção do Título de Mestre em Engenharia Elétrica

Campinas, Maio de 1997

Este exemplar corresponde à redação final da tese defendida por EVELIO MARTÍN GARCÍA FERNÁNDEZ e aprovada pela Comissão Julgadora em 12/05/97.

R. E. Baldini Filho
Orientador

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA - BAE - UNICAMP

G165c

García Fernández, Evelio Martín
Códigos corretores de erro com boas propriedades de codificação de linha / Evelio Martín García Fernández.-- Campinas, SP: [s.n.], 1997.

Orientador: Renato Baldini Filho.
Dissertação (mestrado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação

1. Códigos de controle de erros (Teoria da informação). 2. Sincronização. I. Baldini Filho, Renato. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

Resumo

Esta tese apresenta um método para procurar códigos que combinem características de controle de erros e propriedades de codificação de linha. Tradicionalmente isto tem sido feito através de duas operações de codificação em cascata. Existem algumas desvantagens com este tipo de configuração: a natureza não linear do decodificador de linha pode levar a propagação de erros, e mais ainda, ambas as operações de codificação introduzem redundância na mensagem digital para fazê-la compatível com as características físicas do canal de comunicação. Como a redundância é introduzida duas vezes, a taxa de transmissão de dados é reduzida. No método apresentado neste trabalho, os códigos de blocos com "runlength" limitado são obtidos a partir de uma classe lateral apropriada de um código de bloco linear transparente. O método é baseado na modificação da matriz geradora na forma sistemática do código de bloco transparente e não da matriz de verificação de paridade como usualmente é feito. Vários algoritmos são apresentados para determinar o limitante mínimo para o "runlength" e para encontrar um código específico que satisfaz esse limitante. Os resultados para vários códigos são apresentados em tabelas. Finalmente, o efeito da limitação do "runlength" desses códigos é examinado em termos do espectro de potência dos mesmos. As principais vantagens do método proposto são sua simplicidade e generalidade, ou seja, o método pode ser aplicado em qualquer código de bloco transparente sem restrições de comprimento ou distância mínima de Hamming.

Abstract

The present work presents a method to find codes which combine error control with linecoding features. Traditionally, this has been achieved by cascading these two separate coding operations. There are some disadvantages with this type of coding configuration; in particular, the non linear nature of the line decoder can lead to error propagation. On the other hand, both coding operations involve the introduction of redundancy into a digital message to make it compatible with the physical characteristics of the available channel. As redundancy is introduced twice, at each stage the rate of the transmitted data is reduced. In the method presented here, the combined codes are obtained by taking an appropriate coset of a modified linear transparent error control code. The method is based on the modification of the generator matrix of the transparent linear systematic block code instead of the parity check matrix as usual. Algorithms are presented for determining the minimum runlength bound for a given code and for finding the particular code that satisfies these runlength bounds. The results for a wide range of codes are presented in tabular form. Finally, the effect of limiting the runlength of these codes is examined in terms of its power spectrum. The main advantages of the method presented here are its simplicity and generality, i.e., it can be applied in any transparent block code with no restriction of block length or minimum Hamming distance.

Agradecimentos

Ao Professor Doutor Renato Baldini Filho, pela sua profunda colaboração para a realização deste trabalho, aconselhando e sugerindo-me idéias que foram muito importantes.

Aos professores da Faculdade de Engenharia Elétrica e Computação da UNICAMP, especialmente do Departamento de Comunicações, por sua grande contribuição para a culminação deste trabalho.

Ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), pela bolsa de estudo concedida através do Convênio PEC-PG.

À Profa. Rosa Lydia pela colaboração na redação e na revisão ortográfica deste trabalho.

Aos amigos do Departamento, especialmente Rivelino, Luis Rómulo, Vicente, Christof e Favio, pela companhia durante o Mestrado.

À minha família, pelo carinho e atenção que dispensaram a mim durante toda minha vida.

Glossário

A_a	Evento em que duas amostras de uma seqüência codificada encontram-se em diferentes palavras código.
A_a^c	Complemento do evento A_a .
AMI	Alternate Mark Inversion (código retorno a zero bipolar).
BCH	Códigos Bose Chaudhuri Hocquenguem.
C	Código de Bloco (n,k).
C^i	Palavras código de C.
C_j^i	Primeiros j bits de C^i.
C_j	Conjunto de todos os vetores de C_j^i .
d	Quantidade mínima de zeros entre dois uns lógicos numa seqüência com "runlength" limitado.
DC	Direct Current.
d_{min}	Distância mínima de Hamming de um Código de Bloco.
e	Vetor padrão de erro.
$e(X)$	Polinômio que representa o vetor padrão de erro.

f	Freqüência.
G	Matriz geradora de um Código de Bloco.
G_j	Matriz formada pelas primeiras j colunas da matriz G de C.
G'	Matriz geradora modificada.
$g(X)$	Polinômio gerador de um Código Cíclico.
H	Matriz de verificação de paridade de um Código de Bloco.
$\tilde{H}$	Matriz de verificação de paridade modificada.
$\tilde{h}_{i,j}$	Elemento de $\tilde{H}$.
i_n	Seqüência de entrada a um codificador.
$i(X)$	Polinômio que representa a seqüência de informação num Código Cíclico.
k	Número de dígitos de informação numa palavra código de um Código de Bloco (n,k).
k	Número máximo de zeros consecutivos numa seqüência com "runlength" limitado.
l	Número máximo de zeros consecutivos no início de uma seqüência com "runlength" limitado.
L	Comprimento da janela no peridiograma de Welch.
LD	Linearmente dependente.
LI	Linearmente independente.
m	Comprimento de uma palavra código para a análise espectral.
m	Vetor modificador.

$M(X)$	Polinômio que representa o vetor modificador.
$\text{Min}R_{END}$	Valor mínimo do número de 1/0s consecutivos ao final de uma palavra código.
$\text{Min}RL_{MAX}$	Valor mínimo do número máximo de 1/0s consecutivos numa seqüência codificada.
$\text{Min}R_{MID}$	Valor mínimo do número de 1/0s consecutivos numa palavra código.
$\text{Min}R_{START}$	Valor mínimo do número de 1/0s consecutivos no início de uma palavra código.
n	Comprimento das palavras código num Código de Bloco (n,k).
NRZ	Código não retorno a zero.
$N(S_1)$	Número de elementos contidos em S_1.
$N(S_2)$	Número de elementos contidos em S_2.
PLL	Laço de controle automático de fase.
PT	Código pseudoternário.
r	Número máximo de zeros consecutivos ao final de uma seqüência com "runlength" limitado.
r	Vetor recebido.
$r(X)$	Polinômio que representa o vetor recebido.
$R(a)$	Função de autocorrelação de uma seqüência codificada.
R_{END}	Número de 1/0s consecutivos ao final de uma palavra código.
RLL	Seqüência com "runlength" limitado.

RL_{MAX}	Máximo número de 1/0s consecutivos numa seqüência codificada.
R_{MID}	Número de 1/0s consecutivos numa palavra código.
R_{START}	Número de 1/0s consecutivos ao início de uma palavra código.
$R_Y(a)$	Função de autocorrelação de $Y(a)$.
RZ	Código retorno a zero.
S	Submatriz de $\tilde{H}$.
S_1	Subconjunto de colunas LD no início de G'.
S_2	Subconjunto de colunas LD ao final de G'.
$S(\omega)$	Função densidade espectral de potência de uma seqüência codificada.
t	Capacidade de correção de erro de um Código de Bloco.
T	Tempo de transmissão de um símbolo numa seqüência codificada.
T_m	Tempo de transmissão de uma palavra código.
$t(X)$	Polinômio código de um Código Cíclico.
U^i	Dígitos de informação de C.
V_j	Espaço vetorial de dimensão j.
$\mathbf{v}_1 ... \mathbf{v}_{2^k}$	Vetores códigos de um Código de Bloco.
ω	Freqüência angular.
$X(n)$	Seqüência codificada.
$Y(n)$	Seqüência discreta de fase aleatória.

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

CONCEITOS GERAIS SOBRE CODIFICAÇÃO DE LINHA

1.1 Introdução

As palavras "código" e "codificação" têm uma longa e intrincada história e são usadas comumente por diferentes pessoas com significados diferentes. Do ponto de vista das telecomunicações, a codificação consiste em adaptar a fonte de sinal ao canal de transmissão com o propósito de aumentar a confiabilidade e eficiência da comunicação [Cattermole, 1983]. A confiabilidade expressa-se geralmente em termos da probabilidade de recepção de informação errada, ou seja, informação que difere da originalmente transmitida. O controle de erros tem a ver com as técnicas e métodos de se enviar informação desde uma fonte até um destino (receptor) com a menor quantidade possível de erros e tem as suas origens nos trabalhos de Shannon [Immink, 1990]. A Teoria de

Informação divide o problema da codificação em duas categorias principais: codificação de fonte e codificação de canal. A codificação de fonte consiste na redução da taxa de símbolos da fonte através da eliminação de redundância na informação gerada por essa fonte. Assim, a codificação de canal tem como objetivo, lograr uma alta confiabilidade na transmissão fazendo um uso eficiente da capacidade do canal de comunicações. Em essência, a Teoria de Informação mostra que um canal de comunicação pode ser arbitrariamente confiável desde que uma fração fixa do canal seja utilizada para transmitir redundância que posteriormente será usada na recepção para detectar e/ou corrigir erros que tenham ocorridos devido ao ruído no processo de transmissão. Até que ponto pode-se alcançar na prática o limite teórico imposto é uma pergunta que ainda hoje não tem resposta. O que existe na atualidade é um consenso sobre estruturas de códigos "bons" que garantem um uso bastante eficiente do canal.

Em muitos sistemas de comunicações, a codificação de canal realiza-se em dois passos: a) codificação para controle de erros e b) codificação de linha, como mostra a figura 1.1.

A codificação para controle de erros se realiza mediante a adição sistemática de símbolos redundantes à mensagem a ser transmitida. Esses símbolos extras possibilitam que o receptor possa detectar e/ou corrigir alguns dos erros que tenham acontecido na seqüência recebida. A principal restrição está em alcançar a requerida proteção contra os erros que inevitavelmente vão ocorrer na transmissão sem ter que pagar um preço muito alto pela adição de símbolos extras. Existem muitas famílias diferentes de códigos corretores de erros que são usados indistintamente de acordo com a aplicação. Entre eles

podem-se mencionar os códigos para correção de erros aleatórios e os códigos para correção de erros em surto.

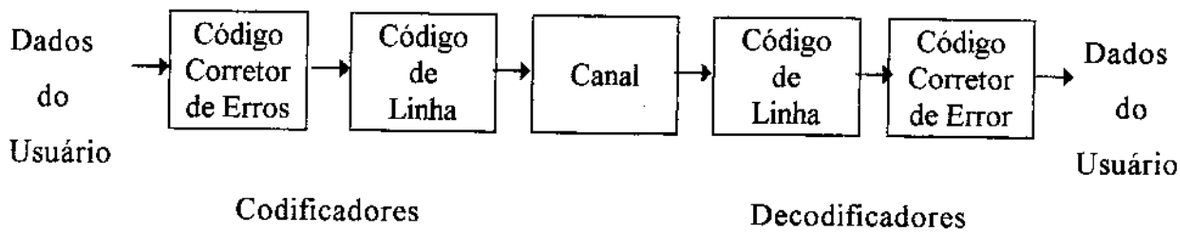

Figura 1.1: Diagrama em blocos de um Sistema de Comunicações.

O codificador de linha aceita na sua entrada as seqüências codificadas pelo código corretor de erros e as converte numa forma de onda que facilita o processo de detecção na presença do ruído e de imperfeições introduzidas pelo canal de transmissão.

1.2 Requerimentos dos Códigos de Linha

Os códigos de linha podem ser projetados para realizar diferentes funções dependendo da aplicação em que vão ser utilizados. Essas aplicações podem ser em sistemas de transmissão em banda básica, sistemas de fibras óticas, sistemas de rádio, sistemas de gravação magnética, etc.

De forma geral, as principais funções dos códigos de linha são as seguintes:

a) Minimizar a vulnerabilidade à interferência intersimbólica.

b) Possibilitar a extração da referência de sinalização na recepção.

c) Adequar o espectro do sinal transmitido às exigências do canal.

d) Aleatorização da seqüência de bits a serem transmitidas.

O principal requerimento para os sistemas de transmissão em banda básica é dado pela operação em tempo real do decodificador (só são permitidos pequenos atrasos na detecção). Nestes sistemas, a codificação de linha é usada principalmente para prevenir, na medida possível, os efeitos das imperfeições do canal. Quando a distorção e interferências constituem as principais fontes de degradação, além do ruído aditivo, essas imperfeições vão estar relacionadas com as propriedades espectrais do sinal transmitido. Na prática, usualmente acontece que a função de transferência do canal é ruim perto dos limites da faixa. Portanto, um bom projeto de código de linha deve concentrar a potência do sinal transmitido no centro da largura de faixa de transmissão. Para cumprir com este objetivo, os códigos devem ser projetados para conformar o espectro do sinal transmitido [Benedetto, 1987].

Além da conformação espectral, outros importantes requerimentos são necessários na hora de projetar estes códigos. De grande importância são a redução do componente de corrente contínua e as propriedades de sincronismo que estes códigos devem apresentar.

Um canal em banda básica real não apresenta uma verdadeira caraterística passa baixas. Sempre vai existir uma freqüência de corte inferior devido a componentes de acoplamento, transformadores de isolamento, etc. Isto produz uma interferência intersimbólica de forma previsível que vai se manter durante um longo intervalo de tempo, provavelmente durante algumas centenas de dígitos [Cattermole 1983]. Isto faz

necessário a redução ao máximo possível do conteúdo espectral nas regiões próximas à freqüência zero.

A técnica mais utilizada para extrair o sincronismo de bit no receptor consiste em utilizar um laço de controle automático de fase (PLL) que utilize os cruzamentos por zero do sinal como referência. Portanto, em muitos casos é necessário a presença de um grande número de transições na seqüência de sinal recebida e que não existam intervalos de tempo longos entre transições. O quanto durará esse intervalo depende do circuito de sincronismo. Circuitos práticos podem tolerar um tempo de não transições de 20 a 30 dígitos, escolhendo-se entretanto entre 4 e 8 dígitos como um valor adequado.

1.3 Códigos de Linha Binários

A forma mais fácil de se adaptar o espectro do sinal transmitido num sistema de sinalização binário consiste na seleção adequada da forma de onda a ser transmitida. Na sua implementação mais simples, esses códigos de linha são conformados somente pelo projeto do sinal a ser transmitido.

Alguns dos formatos binários mais utilizados são mostrados na figura 1.2, os quais são simplesmente esquemas de sinalização binários. O código não retorno a zero (NRZ) designa os dois níveis antipodais do sinal aos dígitos binários. No código retorno a zero (RZ), o símbolo "um" é representado por pulsos que têm duração igual à metade da

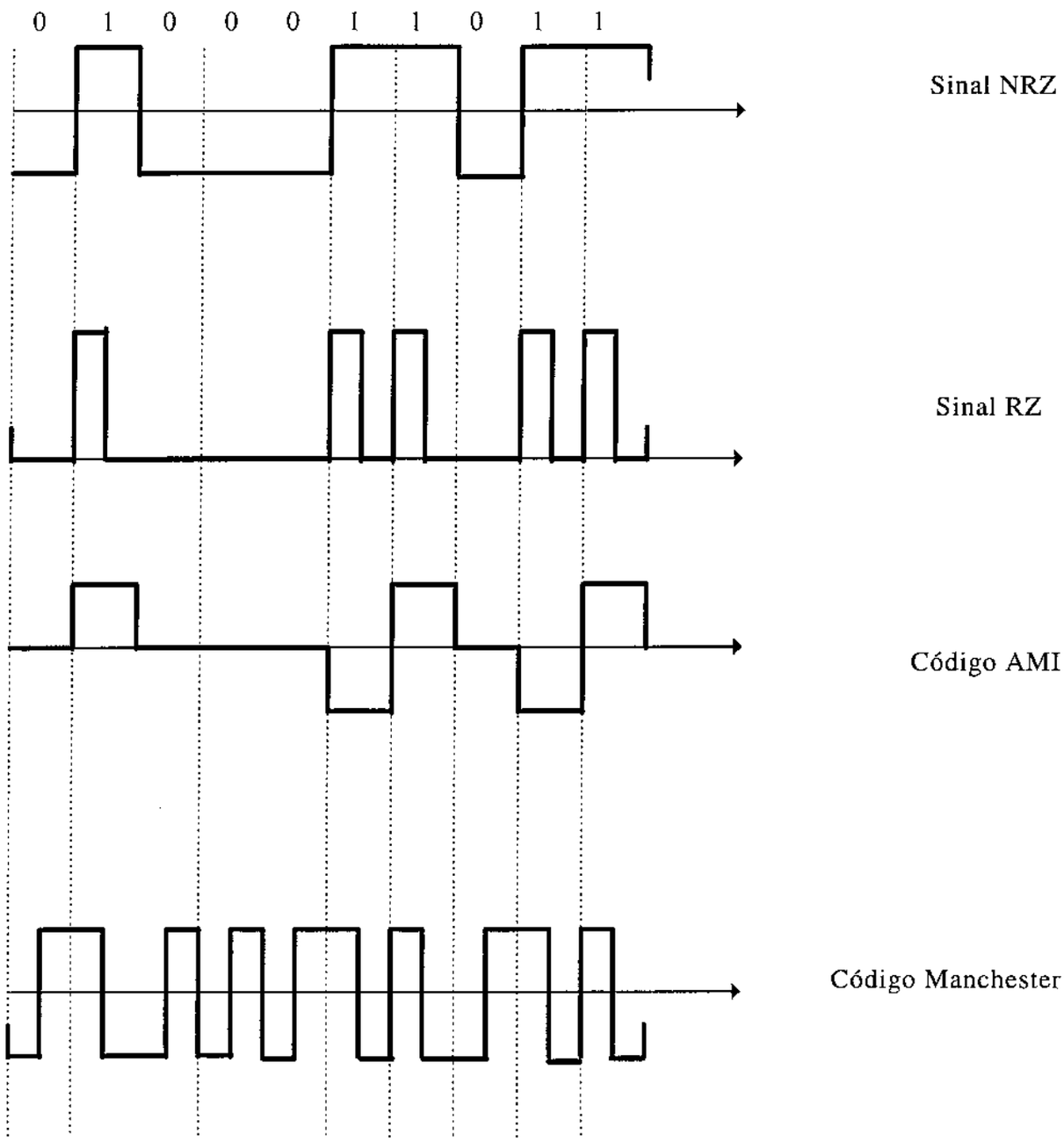

Figura 1.2 Exemplos de formas de onda usadas em códigos de linha binários

duração do símbolo a transmitir, e o símbolo "zero" é representado pela ausência de pulso. Este código tem um alto conteúdo espectral em $f = 0$. O código retorno a zero bipolar, mais conhecido como código AMI, utiliza três níveis de amplitude, sendo que os pulsos positivo e negativo de iguais amplitudes são usados de forma alternada para se transmitir o símbolo "um", e a ausência de pulso é sempre utilizada para a transmissão do símbolo "zero". A maior vantagem deste código é que o espectro de potência do sinal transmitido não tem componente de corrente direita (nível DC) e o conteúdo espectral nas freqüências baixas é pequeno quando a probabilidade de ocorrência dos símbolos "um" e "zero" é a mesma. O código de Manchester (ou bifase) utiliza duas formas de onda simétricas e com conteúdo DC igual a zero no intervalo de duração de um bit. Estas formas de onda são associadas aos bits 0 e 1. Este código apresenta um nível DC igual a zero mas o seu conteúdo espectral estende-se até freqüências aproximadamente duas vezes maiores do que nos casos anteriores [Benedetto, 1987].

1.4 Codificação por Níveis Correlativos

As técnicas de codificação por níveis correlativos introduzem deliberadamente uma quantidade controlada de interferência intersimbólica sobre um conjunto de vários símbolos. Como resultado se produz uma conformação espectral do sinal transmitido ao preço de um incremento no número de níveis a serem transmitidos com respeito à seqüência original. Note-se que em essência se produz uma introdução de redundância na

amplitude do sinal transmitido. Ou seja, existem mais níveis de sinal dos que os estritamente necessários. Neste código, o codificador introduz correlação entre os símbolos transmitidos, possibilitando a conformação do espectro do sinal transmitido.

Entre este códigos podem-se citar o código AMI (alternate-mark-inversion), o código duobinário e os códigos pseudoternários (PT) Entre os que se destacam estão os códigos alfabéticos como 4B3T e MS43 [Benedetto, 1987]. Deve-se assinalar que estes códigos têm a caraterística de propagar erros durante o processo de detecção.

1.5 Seqüências com "Runlength" Limitado

Em sistemas de gravação magnética, uma técnica comunmente utilizada para atenuar os efeitos da interferência intersimbólica e manter o sincronismo de bit, consiste em impor restrições de "runlength" na seqüência de dados [Lee, 1989]. Por "runlength" entende-se a duração, geralmente expressa em número de símbolos binários, de um intervalo sem transições. Assim, os códigos para gravação magnética são baseados em seqüências com "runlength" limitado (RLL) que têm alcançado uma grande aceitação em aplicações de gravação de discos tanto por métodos óticos como magnéticos [Immink, 1990]. As seqüências com "runlength" limitado caraterizam-se por dois parâmetros, d e k, que estipulam o mínimo e o máximo "runlength", respectivamente, que podem ocorrer na seqüência. O parâmetro d controla a mais alta freqüência de transição e tem relação com a interferência intersimbólica quando a seqüência é transmitida por canais limitados em

faixa. Se as transições estão muito perto, a interferência intersimbólica entre símbolos adjacentes pode-se tornar excessiva. Nas seqüências RLL, a ocorrência deste fenômeno é evitada. O parâmetro k garante uma freqüência nas transições adequada para a operação dos circuitos de sincronismo no receptor. Os valores específicos que serão dados a estes dois parâmetros dependem de vários fatores tais como: a resposta do canal, a densidade da informação desejada e as características do ruído.

Em essência uma seqüência RLL satisfaz simultaneamente as seguintes condições:

a) restrição d: dois "uns" lógicos estão separados por ao menos d "zeros" consecutivos,

b) restrição k: qualquer seqüência de "zeros" consecutivos tem comprimento k como máximo.

Existem diferentes classes de códigos baseados em seqüências RLL. Entre eles podem-se citar os códigos (d,k) de comprimento fixo, os códigos de comprimento fixo baseados em seqüências $(dklr)$ e os códigos síncronos de comprimento variável.

Nos códigos de comprimento fixo, o codificador agrupa a seqüência de bits proveniente da fonte em blocos de comprimento m e de acordo com as regras de codificação, esse blocos são convertidos em palavras de n símbolos de canal. Para isto, utilizam-se palavras código que podem ser colocadas em cascata livremente sem alterar as restrições impostas à seqüência. As palavras código utilizadas tem uma correspondência um a um com as palavras da fonte.

Os códigos baseados em seqüências $(dklr)$ estão formados por seqüências (d,k) com duas restrições adicionais:

a) Restrição *l*: o número de "zeros" consecutivos ao início da seqüência é ao máximo *l*.

b) Restrição *r*: o número de "zeros" consecutivos ao final da seqüência é ao máximo *r*.

Nestes códigos, em geral, não podem ser colocadas seqüências de comprimento *n* em cascata sem alterar as restrições (d,k) entre palavras sucessivas. Para preservar estas restrições é necessário colocar bits adicionais entre seqüências adjacentes.

Para aumentar a eficiência dos códigos de comprimento fixo, é necessário incrementar o comprimento das palavras códigos, o que se traduz num aumento da complexidade do codificador e decodificador. Os códigos de comprimento variável oferecem a possibilidade de se utilizar palavras de menor comprimento com maior freqüência que aquelas com comprimentos maiores. Isto permite uma marcante redução na complexidade do codificador e decodificador com relação aos códigos de comprimento fixo com propriedades similares. Devido ao comprimento variável das seqüências, é necessário que a transmissão seja realizada de forma síncrona.

1.6 Conclusão

Na maioria dos sistemas de comunicações, principalmente aqueles que têm a ver com gravação magnética e ótica, a codificação de canal deve assegurar características tanto de controle de erros como de codificação de linha. Tradicionalmente isto tem sido feito realizando em cascata as duas operações de codificação como mostrado na figura 1.1. Isto traz como conseqüência que, devido à natureza não linear do decodificador de

linha, pode haver propagação de erros durante o processo de decodificação [Popplewell, 1993]. Isto significa que o código corretor de erros vai ter que combater erros produzidos não só no canal, mas também, erros introduzidos pelo decodificador de linha. Portanto, uma parte do potencial do código corretor de erros será consumido na detecção e/ou correção destes erros estendidos. Devido ao fato de que, tanto o controle de erros como a codificação de linha estão baseados na introdução de redundância na seqüência a transmitir, seria interessante explorar formas nas quais essas operações possam ser combinadas convenientemente numa única operação.

Capítulo 2

CÓDIGOS CORRETORES DE ERROS COM "RUNLENGTH" LIMITADO

2.1 Introdução

Existem na atualidade uma grande variedade de classes de códigos corretores de erros. Historicamente esses códigos têm sido classificados em códigos de bloco e códigos convolucionais. Os códigos de blocos podem ser lineares ou não lineares.

Um código linear binário tem a propriedade de que duas palavras código podem ser somadas usando-se aritmética módulo-2, para produzir uma outra palavra que também pertence ao código. Os códigos utilizados em aplicações práticas são quase sempre códigos lineares.

Para se gerar um código de bloco (n,k) linear, o codificador aceita blocos de k bits de informação e adiciona n-k bits de redundância algebricamente relacionados com os k

bits de informação, produzindo na sua saída palavras código de n bits onde $n > k$. Os códigos de bloco em que os bits de informação são transmitidos inalteradamente, são chamados de códigos sistemáticos. Em aplicações que requerem tanto detecção como correção de erros, o uso de códigos sistemáticos simplifica a implementação do decodificador [Haykin, 1994].

Devido à propriedade de linearidade, facilita-se muito o trabalho com estes códigos mediante a representação dos mesmos através de matrizes. A matriz geradora (geralmente conhecida como matriz G), é usada na operação de codificação no transmissor. A matriz de verificação de paridade (ou matriz H), é geralmente usada na operação de decodificação no receptor para a correção e/ou detecção de erros.

A capacidade de correção e/ou detecção de erros de um código de bloco linear está relacionada diretamente com a distância de Hamming mínima $d_{\min}$ do código. No caso de códigos de bloco binários, a distância mínima é definida como o menor peso de Hamming encontrado nas palavras código excluindo-se a palavra composta inteiramente por zeros. O peso de Hamming de uma palavra código é definido como o número de elementos diferentes de zero na palavra código. Se um código de bloco é usado para detectar e corrigir todos os padrões de erro cujo peso de Hamming seja menor ou igual a t, a sua distância mínima deve ser igual ou maior que $2t+1$.

Na decodificação dos códigos de bloco utiliza-se muito um esquema conhecido como arranjo padrão [Lin, 1983]. Seja C um código linear (n,k). Sejam $\mathbf{v}_1$, $\mathbf{v}_2$,..., $\mathbf{v}_{2^k}$ os vetores códigos de C. Seja qual for o vetor código transmitido pelo canal ruidoso, o vetor recebido $\mathbf{r}$ pode ser qualquer uma das 2^n n-uplas que formam o espaço vetorial de

dimensão n com símbolos binários. Qualquer esquema de decodificação usado no receptor é uma regra para particionar os 2^n vetores possíveis de serem recebidos em 2^k subconjuntos disjuntos D_1, D_2, ..., D_{2^k} tal que o vetor código $\mathbf{v}_i$ está contido no subconjunto D_i para $1 \leq i \leq 2^k$. Portanto, cada subconjunto D_i tem uma correspondência um a um com um vetor código $\mathbf{v}_i$. Se o vetor recebido $\mathbf{r}$ se encontra no subconjunto D_i, $\mathbf{r}$ é decodificado como $\mathbf{v}_i$. A decodificação estará correta se e somente se o vetor recebido $\mathbf{r}$ estiver no subconjunto D_i correspondente ao vetor código realmente transmitido.

O particionamento é feito baseado na estrutura linear do código. Primeiro, os 2^k vetores códigos de C são colocados numa linha com o vetor $\mathbf{v}_1 = (0, 0, ..., 0)$ como primeiro elemento da esquerda. Das restantes $2^n - 2^k$ n-uplas, é selecionada a n-upla $\mathbf{e}_2$ de menor peso de Hamming e colocada embaixo do vetor zero $\mathbf{v}_1$. Agora é formada a segunda linha, somando $\mathbf{e}_2$ a cada vetor código $\mathbf{v}_i$ da primeira linha e colocando a soma $\mathbf{e}_2 + \mathbf{v}_i$ embaixo de $\mathbf{v}_i$. Tendo completada a segunda linha, é selecionada uma n-upla, $\mathbf{e}_3$ de menor peso de Hamming, que ainda não tenha sido utilizada e colocada embaixo de $\mathbf{e}_2$. Então, a terceira linha é formada somando $\mathbf{e}_3$ a cada vetor código $\mathbf{v}_i$ da primeira linha e colocando $\mathbf{e}_3 + \mathbf{v}_i$ embaixo de $\mathbf{v}_i$. O processo continua até que todas as n-uplas sejam usadas. Assim, obtém-se um arranjo de linhas e colunas como mostrado na figura 2.1. Este arranjo é chamado de arranjo padrão do código.

Como pode-se ver na figura 2.1, no arranjo padrão existem $2^n/2^k = 2^{n-k}$ linhas diferentes e cada uma delas é composta por 2^k elementos diferentes. Essas linhas são chamadas de classes laterais do código C e a primeira n-upla $\mathbf{e}_j$ de cada classe lateral é denominada de líder da classe lateral. Qualquer elemento numa classe lateral pode ser

utilizado como líder da classe lateral. Isso não muda os elementos da classe lateral, simplesmente permuta-os de posição, mas geralmente é utilizado como líder de classe lateral os vetores de menor peso de Hamming.

$v_1 = 0$	v_2	...	v_i	...	v_{2^k}
e_2	$e_2 + v_2$	...	$e_2 + v_i$	...	$e_2 + v_{2^k}$
e_3	$e_3 + v_2$	...	$e_3 + v_i$	...	$e_3 + v_{2^k}$
.					.
.					.
.					.
e_l	$e_l + v_2$	...	$e_l + v_i$	...	$e_l + v_{2^k}$
.					.
.					.
.					.
$e_{2^{n-k}}$	$e_{2^{n-k}} + v_2$	...	$e_{2^{n-k}} + v_i$	...	$e_{2^{n-k}} + v_{2^k}$

Figura 2.1: Arranjo Padrão para um Código de Bloco Linear (n,k).

Os códigos cíclicos constituem uma subclasse de códigos de bloco lineares que têm a vantagem de serem fáceis de se codificar. Mais ainda, os códigos cíclicos possuem uma estrutura matemática bem definida, o que tem possibilitado o desenvolvimento de esquemas de codificação muito eficientes para eles.

Diz-se que um código de bloco binário é cíclico se ele exibe duas propriedades fundamentais:

a) A soma de duas palavras códigos é também uma palavra código,

b) Qualquer deslocamento cíclico de uma palavra código é também uma palavra código.

Esta última propriedade possibilita a descrição das palavras código através de polinômios, sendo que o polinômio de menor grau dentre todos os polinômios código é chamado de polinômio gerador $g(X)$, através do qual pode ser descrito o código de forma equivalente à matriz geradora G.

2.2 Relação entre Códigos de Bloco Lineares e Códigos de Linha

Em geral, os códigos de bloco lineares não possuem boas propriedades de codificação de linha pois: a) eles têm uma disparidade acumulada ilimitada (um desbalanceamento acumulado entre "uns" e "zeros" enviados através do canal) e portanto não produzem a eliminação da componente de corrente contínua (nível DC) no espectro de potência e b) eles têm um "runlength" ilimitado (número de símbolos idênticos consecutivos enviados através do canal), o que pode levar a problemas de sincronismo no receptor.

Todos os códigos de bloco binários lineares e transparentes[1] têm "runlength" ilimitado devido à inclusão, no conjunto das palavras código, da palavra toda "um" e da toda "zero". Assim, para produzir um código de bloco que não tenha o "runlength" ilimitado é necessário encontrar um conjunto de palavras códigos que não contenham as

[1] Um código de bloco transparente é aquele em que o inverso de cada palavra código é também uma palavra código.

palavras formadas por todos os símbolos iguais a zero ou a um, mas que mantenham as propriedades de distância mínima do código original. Isto pode ser feito, formando classes laterais do código original mediante a adição à todas as palavras código, de uma palavra de n bits não pertencente ao código que será chamada de vetor de modificação. A linearidade do código assegura que as propriedades de distância mínima serão preservadas e portanto a capacidade de correção de erros. Isto significa que a seqüência de informação pode ser codificada de acordo com o código original, mas antes da transmissão, um vetor modificador (líder de cosset) é somado a cada palavra código fazendo com que a seqüência de dados enviada pelo canal tenha um "runglength" limitado a um determinado valor. No receptor é feita a operação inversa, portanto, antes da decodificação da palavra recebida é extraído o vetor modificador e em seguida decodificada segundo o código corretor de erro original, como mostrado na figura 2.2.

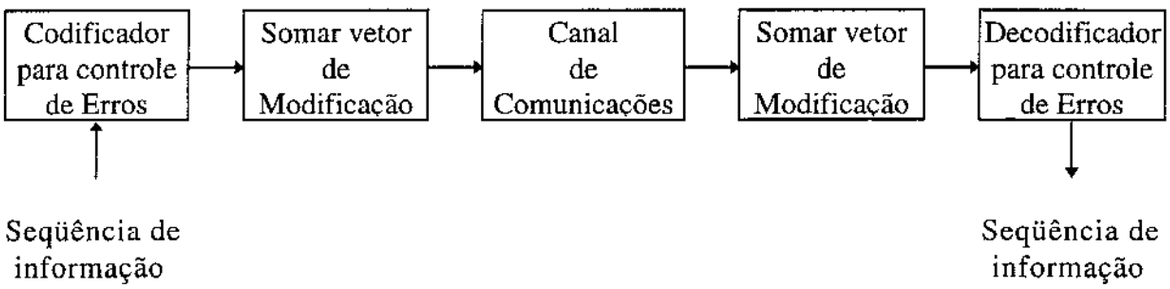

Figura 2.2: Diagrama de Blocos de um Sistema que Utiliza Códigos Corretores de Erro com "Runlength" Limitado.

2.3 Modificação de um Código de Bloco Transparente

Existem 2^{n-k} classes laterais diferentes associadas a um código de linha corretor de erros sendo que uma delas é o conjunto das palavras códigos do código corretor de erros original. Portanto, existem $2^{n-k}-1$ classes laterais que não contém as palavras formadas totalmente por "uns" e por "zeros" e consequentemente produzem um código com "runlength" limitado. O problema é: qual a classe lateral que proporciona o mínimo "runlength"? Isto pode ser resolvido em duas etapas: primeiro, determinar o mínimo "runlength" possível de se alcançar com uma determinada classe lateral e segundo, encontrar um vetor modificador para formar a classe lateral que satisfaz o mínimo "runlength" previamente determinado. A seguir será descrito um método para se alcançar este objetivo [Popplewell, 1992].

2.4 Determinação do Limitante Mínimo Possível para o "Runlength" de um Código de Linha Corretor de Erros [Popplewell, 1992]

Dado um conjunto de palavras código que não contém as palavras formadas totalmente por "uns" e por "zeros", tem-se que o máximo "runlength" (RL_{MAX}) é definido como o máximo número de 1/0s consecutivos em uma palavra código (R_{MID}), ou o

máximo número de 1/0s no início de uma palavra código (R_{START}) mais o máximo número de 1/0s ao final de uma palavra código (R_{END}), tomando-se entre as duas possibilidades o maior valor. Portanto,

$$RL_{MAX} = \text{MAX}\{R_{END} + R_{START} , R_{MID} \} \tag{2.1}$$

Dado um código de bloco corretor de erros (n,k) com matriz geradora $G = [g_{ij}]$, utilizando os três teoremas seguintes pode-se determinar os valores mínimos atingíveis por R_{END} , R_{START} e R_{MID} , que serão chamados de MinR_{END} , MinR_{START} e MinR_{MID} respectivamente, para alguma classe lateral do código corretor de erros, e portanto definir um limitante inferior para o máximo "runlength" (MinRL_{MAX}) do código.

Teorema 1

MinR_{START} = o número de colunas linearmente independentes (LI) consecutivas no início da matriz G.

Teorema 2

Min R_{END} = o número de colunas linearmente independentes consecutivas no final da matriz G.

Teorema 3

MinR_{MID} = o máximo número de colunas linearmente independentes consecutivas na matriz G.

Prova do Teorema 1

Definições:

C: Código de linha corretor de erros (n,k).

G_j : Matriz formada pelas primeiras j colunas da matriz geradora G de C.

C^i e U^i , $0 \le i \le 2^k - 1$: Palavras códigos e dígitos de informação, respectivamente, de C.

C^i_j : Primeiros j bits de C^i .

C_j : Conjunto de todos os vetores de C^i_j , $0 \le i \le 2^k - 1$.

V_j : Espaço vetorial de dimensão j.

Para se encontrar uma classe lateral que tenha um mínimo de j 1/0s nos primeiros j bits de qualquer elemento da classe lateral, é suficiente encontrar um líder de classe lateral ou vetor modificador **m** que satisfaça:

$$m_1, m_2, \ldots m_{j+1} \notin C_{j+1} \tag{2.2}$$

Isto é possível se,

$$C_{j+1} \neq V_{j+1} \tag{2.3}$$

e o mínimo valor de $j = \mathrm{Min} R_{START}$ acontecerá quando

$$C_j = V_j \tag{2.4}$$

e

$$C_{j+1} \neq V_{j+1} \tag{2.5}$$

Assim, a equação 2.4 vai-se cumprir se G_j tiver j linhas LI ou equivalentemente, j colunas LI, toda vez que $C^i_j = U^i G_j$. Da mesma forma, a equação 2.5 acontecerá se G_{j+1} tiver menos que $j+1$ colunas LI.

Portanto, para encontrar o valor mínimo de j deve-se ter,

G_j com j colunas LI e,

G_{j+1} com menos de $j+1$ colunas LI.

Ou seja, o mínimo valor de j = MinR_{START} , é igual ao número de colunas LI consecutivas no começo de G como explicitado no teorema 1. De modo análogo, pode-se provar os teoremas 2 e 3.

Deve-se notar que como o rank de uma matriz (n,k) corresponde ao número de linhas ou colunas linearmente independentes, o rank $\leq$ MIN $\{n,k\}= k$ para todos os códigos onde Min$R_{MID} \leq k$, Min$R_{END} \leq k$, e Min$R_{START} \leq k$.

A conseqüência destes teoremas é que o ordenamento das colunas da matriz geradora G pode afetar significativamente o limitante MinRL_{MAX} e como o reordenamento das colunas de G não afeta as caraterísticas de correção de erro do código, podem ser realizadas permutações de colunas numa matriz G em particular para fazer com que MinR_{START} , MinR_{MID} e MinR_{END} sejam o menor possível e assim produzir um limitante inferior para o "runlength" do código.

Os teoremas possibilitam, sempre que seja possível, encontrar classes laterais que cumpram com os limitantes MinR_{END} , MinR_{START} e MinR_{MID} e, portanto, determinar MinRL_{MAX} , o limitante inferior de RL_{MAX} usando a equação 2.1. Entretanto, não é possível saber diretamente se existe uma classe lateral que possa atingir esses limitantes nem como selecionar um vetor modificador adequado para produzir essa classe lateral.

2.5 Método para encontrar o vetor modificador

Popplewell e O'Reilly [Popplewell, 1992] propuzeram um método para encontrar a classe lateral que melhor se aproxima do limitante MinRL_{MAX}:

a) Dada a matriz G do código de bloco, formar a matriz H do código.

b) Reordenar a matriz H realizando operações elementares sob as linhas para produzir uma matriz $\tilde{H}$ que satisfaz as seguintes condições:

(i) $\tilde{H}$ tem ao menos uma linha tal que,

$$\tilde{h}_{z,i} = 0 \text{ para todo } i = \begin{matrix} \text{Min}R_{START} + 2, \\ \text{Min}R_{START} + 3, \\ \vdots \\ \text{Min}R_{START} + n \end{matrix} \qquad (2.6)$$

(ii) $\tilde{H}$ tem ao menos uma linha tal que,

$$\tilde{h}_{z,i} = 0 \text{ para todo } i = 1, 2, \ldots, p, \begin{matrix} \text{Min}RL_{MAX} + p + 2, \\ \text{Min}RL_{MAX} + p + 3, \\ \vdots \\ \text{Min}RL_{MAX} + p + n \end{matrix} \qquad (2.7)$$

para todo $p = 1, 2, \ldots, n - \text{Min}RL_{MAX} - 2$.

(iii) $\tilde{H}$ tem pelo menos uma linha tal que,

$$\tilde{h}_{z,i} = 0 \text{ para todo } i = 1, 2, \ldots, n - \text{Min}R_{END} - 1 \qquad (2.8)$$

A matriz de verificação de paridade pode ser colocada desta forma usando um método similar à eliminação Gaussiana; o procedimento requer um máximo de

$$(n-k)(n-k-1) \approx (n-k)^2 \text{ operações sob as linhas} \qquad (2.9)$$

Portanto, para um determinado código, a complexidade do procedimento aumenta com o quadrado do número de bits de verificação de paridade.

Note que se $\text{Min}RL_{MAX} > n-2$ então a condição (iii) não se aplica.

c) Se $\tilde{H}$ não satisfaz todas as condições, então não existe uma classe lateral capaz de atingir o limitante RL_{MAX} e portanto, um ou mais de um dos limitantes para $\text{Min}R_{END}$, $\text{Min}R_{START}$ ou $\text{Min}R_{MID}$ devem ser aumentados até que a condição (*b*) seja satisfeita.

d) Quando $\tilde{H}$ satisfaz todas as condições em (*b*) então forma-se uma nova matriz S a qual é construida a partir de todas as linhas de $\tilde{H}$ que satisfazem ao menos uma das condições (i), (ii) e (iii).

e) Forma-se a seguinte equação matricial:

$$\mathbf{m}S^t = 1 \qquad (2.10)$$

f) Qualquer solução desta equação produzirá um vetor modificador **m** que vai formar uma classe lateral do código original que satisfaz os limitantes predeterminados para RL_{MAX}.

Prova do método.

Para encontrar um vetor modificador, **m**, que produz uma classe lateral que cumpra com os limitantes de "runlength" predeterminados, é preciso encontrar um **m** que satisfaça simultaneamente as seguintes condições:

$$\left.\begin{array}{lcl} m_1 \ldots m_{MinR_{START}+1} & \neq & C^i_{1,MinR_{START}+1} \\ m_2 \ldots m_{MinRL_{MAX}+2} & \neq & C^i_{2,MinRL_{MAX}+2} \\ m_3 \ldots m_{MinRL_{MAX}+3} & \neq & C^i_{3,MinRL_{MAX}+3} \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ m_{n-MinRL_{MAX}-2} \ldots m_{n-1} & \neq & C^i_{n-MinRL_{MAX}-2,n-1} \\ m_{n-MinR_{END}-1} \ldots m_n & \neq & C^i_{n-MinR_{END}-1,n} \end{array}\right\}$$

$$\text{para todo } i = 0, 1, \ldots, 2^k - 1 \tag{2.11}$$

onde $C^i_{j,k}$ é o conjunto das palavras formadas por palavras código com bits de j até k.

Como é conhecido, se **r** é um elemento do conjunto das palavras de um código de bloco corretor de erro e $H = \left[h_{i,j}\right]$ é a matriz de verificação de paridade, então $\mathbf{r}H^t = 0$. Pelo contrário, se **r** não pertence ao conjunto das palavras código, então $\mathbf{r}H^t \neq 0$. Portanto, como **m** não é uma palavra código, deve satisfazer ao menos uma das seguintes equações:

$$\sum_{i=1}^{n} m_i h_{1,i} = 1, \ \sum_{i=1}^{n} m_i h_{2,i} = 1, \ \ldots, \ \sum_{i=1}^{n} m_i h_{n-k,i} = 1 \tag{2.12}$$

Para satisfazer a primeira desigualdade na equação 2.11 se requer que

$$\sum_{i=1}^{n} m_i h_{z,i} = 1 \tag{2.13}$$

para algum $z = 1, 2, \ldots, n-k$.

Agora, se $h_{z,i} = 0$ para todo $i = \text{Min}R_{START} + 2, \text{Min}R_{START} + 3, \ldots, n$, então a equação 2.13 pode ser expressa como

$$\sum_{i=1}^{MinR_{START}+1} m_i h_{z,i} = 1 \tag{2.14}$$

Assim, se são selecionados $m_1 \ldots m_{MinR_{START}+1}$ tais que a equação 2.14 seja satisfeita, então pode-se afirmar que a primeira desigualdade na equação 2.11 e portanto o limitante $\text{Min}R_{START}$ é satisfeito independentemente dos valores selecionados para $m_{MinR_{START}+2} \ldots m_n$.

Considerando agora a segunda desigualdade na equação 2.11 , se existe alguma linha z de H tal que $h_{z,i} = 0$ para todo $i = 1, \text{Min}RL_{MAX} + 3, \text{Min}RL_{MAX} + 4, \ldots, n$

isto é,

$$m_1 + \sum_{i=\text{Min}RL_{MAX}+3}^{n} m_i h_{z,i} = 1 \tag{2.15}$$

então a primeira desigualdade na equação 2.11 é satisfeita simultaneamente com a segunda desigualdade na equação 2.11 se são selecionados $m_1 \ldots m_{MinRL_{MAX}+2}$ tais que $m_1 \ldots m_{MinR_{START}+1}$ satisfaça a equação 2.14 e $m_2 \ldots m_{MinRL_{MAX}+2}$ satisfaça a equação 2.15. Isto possibilita que **m** cumpra com o limitante $\text{Min}R_{START}$ e também em $\text{Min}RL_{MAX}$ sob os primeiros $\text{Min}RL_{MAX} + 2$ bits de **m**.

Assim, em geral, para satisfazer simultaneamente o conjunto de desigualdades na equação 2.9 é suficiente ter ao menos uma linha z de H tal que,

$$h_{z,i} = 0 \text{ para todo } i = \text{Min}R_{START} + 2, \text{Min}R_{START} + 3, \ldots, n \tag{2.16}$$

ao menos uma linha tal que,

$$h_{z,i} = 0 \text{ para todo } i = 1, 2, \ldots, p, \text{Min}RL_{MAX} + p + 2, \text{Min}RL_{MAX} + p + 3, \ldots, n \quad (2.17)$$

onde

$$p = 1, 2, \ldots, n - \text{Min}RL_{MAX} - 2$$

e ao menos uma linha tal que,

$$h_{z,i} = 0 \text{ para todo } i = 1, 2, \ldots, n - \text{Min}R_{END} - 1 \quad (2.18)$$

Portanto, para encontrar um vetor modificador que cumpra com o limitante predeterminado para o "runlength" é suficiente realizar operações elementares nas linhas de H e formar a matriz S com as linhas que satisfaçam as condições acima. Logo, resolvendo a equação matricial $\mathbf{m}S^t = 1$, obtêm-se uma solução, ficando provado o método.

Exemplo utilizando o procedimento anterior para obtenção de um código de linha com "runlength" limitado baseado no código BCH (15,5), que tem o seguinte polinômio gerador:

$$g(X) = X^{10} + X^8 + X^5 + X^4 + X^2 + X + 1 \quad (2.19)$$

Formando a matriz geradora, tem-se que:

$$G = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \end{bmatrix} \quad (2.20)$$

Trocando a coluna 4 com a 11 e a coluna 8 com a 12 obtém-se a seguinte matriz geradora modificada:

$$G' = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \end{bmatrix} \quad (2.21)$$

Usando os teoremas 1, 2 e 3 pode-se encontrar,

$\text{Min}R_{END} = 3$, $\text{Min}R_{START} = 3$ e $\text{Min}R_{MID} = 5$

Assim,

$$RL_{MAX} = \text{MAX}\ \{3 + 3, 5\} = 6 \quad (2.22)$$

Para encontrar o vetor modificador que satisfaça esse limitante tem-se que:

a) Formar a matriz de verificação de paridade

$$H = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \quad (2.23)$$

b) As condições para $\tilde{H}$ são:

(i) Uma linha com os últimos 11 elementos iguais a zero.

(ii) Uma linha com o primeiro e os últimos 7 elementos iguais a zero, uma linha com os primeiros 2 e os últimos 6 elementos iguais a zero,

.
.
.

uma linha com os primeiros 7 e o último elemento iguais a zero.

(iii) Uma linha com os primeiros 11 elementos iguais a zero.

Realizando operações elementares sobre as linhas de H, obtém-se:

$$\tilde{H} = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \end{bmatrix} \tag{2.24}$$

c) A linha 1 satisfaz a condição (i), as linhas 2-7 satisfazem a condição (ii) e a linha 8 satisfaz a condição (iii). As linhas 9 e 10 são redundantes.

d) Formar a matriz S,

$$S = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix} \tag{2.25}$$

e) Agora, resolvendo a equação $\mathbf{m}S^t = 1$, pode-se obter como uma possível solução,

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

A tabela 2.1 mostra uma lista das palavras código modificadas junto às palavras do código BCH (15,5) original.

Note que não é possível encontrar mais do que RL_{MAX} = 6 1/0s consecutivos como previsto da equação 2.22. Existem muitas outras soluções possíveis para a equação $\mathbf{m}S^t = 1$, sendo que cada uma delas produzirá uma classe lateral com RL_{MAX} = 6.

Palavras da Fonte	Palavras Código	Palavras Código Modificadas
00000	000000000000000	100001001010001
00001	000011101100101	100010100110100
00010	000011011011010	100010010001011
00011	000000110111111	100001111101110
00100	001110100011100	101111101001101
00101	001101001111001	101100000101**000**
00110	001101111000110	101100110010111
00111	001110010100011	101111011110010
01000	011001110110000	111000111100001
01001	011010011010101	111011010000100
01010	011010101101010	111011100111011
01011	011001000001111	111000001011110
01100	010111010101100	110110011111101
01101	010100111001001	110101110011**000**
01110	010100001110110	110101000100111
01111	010111100010011	110110101000010
10000	111111001000000	011110000010001
10001	111100100100101	011101101110100
10010	111100010011010	011101011001011
10011	111111111111111	011110110101110
10100	110001101011100	010000100001101
10101	110010000111001	010011001101**000**
10110	110010110000110	010011111010111
10111	110001011100011	010000010110010
11000	100110111110000	**000**111110100001
11001	100101010010101	**000**100011000100
11010	100101100101010	**000**100101111011
11011	100110001001111	**000**111000011110
11100	101000011101100	001001010111101
11101	101011110001001	001010111011**000**
11110	101011000110110	001010001100111
11111	101000101010011	001001100000010

Tabela 2.1: Palavras Código do Código BCH (15,5) Original e Modificado.

2.6 Conclusão

Neste capítulo foi apresentado um procedimento conhecido [Popplewell, 1992] para modificar códigos corretores de erro conhecidos e formar códigos com "runlength" limitado que mantêm inalterada a capacidade de correção de erro e a eficiência do código original. Estes códigos podem ser usados de forma isolada no lugar de um código corretor de erros convencional ou como base para a construção de códigos corretores de erros livres de nível DC.

O procedimento é fundamentado na identificação de vetores modificadores que têm correspondência com limitantes ótimos para o "runlength" do código. Essas idéias são aplicáveis de forma geral a todos os códigos lineares transparentes ao mesmo tempo em que o procedimento torna-se complexo na medida que se aumenta o comprimento do código.

O aspecto mas importante a assinalar é que uma vez que o vetor modificador é identificado, a sua implementação praticamente não aumenta a complexidade com respeito a um código corretor de erros convencional.

Alguns dos resultados obtidos são mostrados na tabela 2.2.

Código Corretor de Erros	*Limitante do "Runlength"*
BCH(15,5)	6
BCH(15,7)	7
BCH(15,11)	17
BCH(31,16)	16
BCH(31,21)	32
BCH(31,26)	38
Golay(23,12)	16

Tabela 2.2: Códigos Corretores de Erro com "Runlength" Limitado.

Do exemplo mostrado a partir do código BCH (15,5), deve-se notar que se tomou como ponto de partida a matriz geradora original na forma não sistemática. Isto aumenta a complexidade na hora de se gerar o código com respeito a esquemas que utilizam codificação sistemática.

No próximo capítulo será exposto uma variante deste procedimento utilizando códigos sistemáticos.

Capítulo 3

MODIFICAÇÃO DE CÓDIGOS CÍCLICOS SISTEMÁTICOS

3.1 Introdução

No capítulo anterior foi apresentado um método para modificar códigos cíclicos transparentes a partir da matriz geradora na forma não sistemática. Nesse método, as colunas da matriz geradora são reordenadas de acordo com os teoremas 1, 2 e 3, sendo que a permutação ótima é alcançada mediante a escolha de dois subconjuntos de colunas linearmente dependentes S_1 e S_2 do conjunto S (o conjunto formado por todas as colunas de G) tal que, $N(S_1)+N(S_2)$ e $S_1 \cap S_2$ são minimizados, onde $N(T)$ é o número de elementos no conjunto T. Assim, as primeiras $N(S_1)$ colunas de G são o subconjunto S_1 e

as últimas $N(S_2)$ colunas são S_2 ou vice-versa. As colunas restantes são os vetores pertencentes ao subconjunto $S-(S_1 \cup S_2)$.

Uma vez encontrados esses subconjuntos, é formada a matriz de verificação de paridade e por meio de operações elementares sob as linhas, a matriz H é reordenada de acordo com as condições (i), (ii) e (iii), dadas no capítulo 2.

Neste capítulo será apresentado um método mais simples para modificar códigos cíclicos transparentes a partir da matriz geradora na forma sistemática que atingem o limitante no "runlength" definido pelos teoremas 1, 2 e 3.

3.2 Método para Modificar Códigos Cíclicos Transparentes Sistemáticos

O método para modificar códigos cíclicos transparentes sistemáticos de tal forma que eles se tornem adequados para alcançar o limitante de "runlength" é dado abaixo:

a) A partir do polinômio gerador $g(X)$ do código, formar a matriz geradora G na forma sistemática.

b) Encontrar um subconjunto S_1 de colunas linearmente dependentes, tal que $N(S_1)$ seja mínimo.

Para se encontrar este subconjunto, é selecionada dentre as $n-k$ colunas de verificação de paridade da matriz G, a coluna (ou menor combinação de colunas) com menor quantidade de uns. Sejam:

t = vetor soma das colunas selecionadas,

nu = número de uns de **t**,

nc = número de colunas contidas nessa combinação. Então, para que as colunas contidas em S_1 sejam L.D., tem-se que:

$$N(S_1) = nu + nc \tag{3.1}$$

sendo que as *nu* colunas contidas neste subconjunto são selecionadas dentre as *k* colunas da submatriz de identidade de *G*, especificamente aquelas em que a posição dos uns coincide com a posição dos uns do vetor **t**.

c) Colocar o subconjunto S_1 no início da matriz geradora modificada *G'*.

Desta forma e tendo em conta o teorema 1 tem-se que:

$$MinR_{START} = N(S_1) - 1 = nu + nc - 1 \tag{3.2}$$

d) Procurar nas colunas ainda não utilizadas das $n-k$ colunas de verificação de paridade de *G*, se existe alguma em que a posição dos uns não coincida com a posição dos uns do vetor **t** no passo *b*. A quantidade de uns dessa coluna deve ser a menor possível.

Dependendo da existência ou não de alguma coluna que satisfaça a condição anterior, dois casos são possíveis de ocorrer:

<u>*Caso* 1</u>: Existe uma coluna como descrita no passo *d*. Neste caso $N(S_1) \cap N(S_2) = 0$, e o método continua da seguinte forma:

e) Colocar a coluna selecionada como sendo a última coluna da matriz geradora modificada *G'*. Seja,

mnu = número de uns da coluna anterior

f) Colocar a esquerda da coluna anterior as *mnu* colunas dentre as colunas da submatriz de identidade de G ainda não utilizadas de modo que a posição dos uns coincida com a posição dos uns da última coluna de G'. Assim,

$$N(S_2) = mnu + 1 \tag{3.3}$$

Desta forma e tendo em conta o teorema 2, tem-se que:

$$MinR_{END} = N(S_2) - 1 = mnu \tag{3.4}$$

g) Completar a matriz G' com as restantes colunas de G.

Caso 2: Não existe uma coluna como descrita no passo *d*. Neste caso $N(S_1) \cap N(S_2) \neq 0$ e portanto, os dois subconjuntos S_1 e S_2 vão se sobrepor. O método continua buscando minimizar esta interseção:

e) Colocar na continuação do subconjunto S_1 as colunas ainda não utilizadas das $n-k$ colunas de verificação de paridade de G. Seja **q**, o vetor soma dessas colunas.

Ao fazer esta operação deve-se ter em conta que a ordem das *nu* colunas que estão contidas em S_1 não pode ser qualquer ordem. As últimas colunas à direita de S_1 serão aquelas em que a posição dos uns coincide com posições de uns do vetor **q** e do vetor **t** simultaneamente. Seja r = número de uns do vetor **q** cujas posições coincidem com posições de uns do vetor **t**. Então, existirão r colunas fazendo parte tanto de S_1 como de S_2.

f) Colocar à direita das colunas do passo *e* aquelas colunas da submatriz de identidade de G em que a posição dos uns coincida com a posição dos uns restantes de **q**.

g) Completar à direita com as restantes colunas de *G*.

Desta forma,

$$N(S_2) = n - N(S_1) + r \tag{3.5}$$

Assim, tendo em conta o teorema 3 tem-se que:

$$\text{Min}R_{END} = N(S_2) - 1 = n - N(S_1) + r - 1 = n - \text{Min}R_{START} + r - 2 \tag{3.6}$$

No caso particular dos códigos de Hamming todas as $n-k$ colunas de verificação de paridade têm igual número de uns e a combinação que tem a menor quantidade de uns é a formada pela soma de todas elas. Então, ao executar o passo *b*, deve colocar-se ao início da matriz *G'* qualquer combinação de $n-k-1$ colunas de verificação de paridade e continuar como descrito no método anterior.

Uma vez modificada a matriz geradora e encontrados os limitantes $\text{Min}R_{START}$ e $\text{Min}R_{END}$, pode-se encontrar o limitante inferior do "runlength" do código a partir de:

$$\text{Min}RL_{MAX} = \text{MAX}\{\text{Min}R_{START} + \text{Min}R_{END} \text{ , } \text{Min}R_{MID}\} \tag{3.7}$$

Por último, a partir da matriz *G'*, procura-se o líder de classe lateral (ou vetor **m**) que somado a todas as palavras código geradas por *G'*, produz um código que atinge os limitantes encontrados previamente.

Para maior compreensão do método, a seguir serão vistos exemplos de cada um dos três casos possíveis.

3.3 Exemplo de Modificação de Códigos em que os Subconjuntos S_1 e S_2 não se Sobrepõem

Para tratar o caso em que os subconjuntos S_1 e S_2 não se sobrepõem, se aplicará o caso 1 do método anterior para modificar o código de Golay (23,12) que tem o polinômio gerador $g(X) = X^{11} + X^9 + X^7 + X^6 + X^5 + X + 1$.

a) A matriz geradora na forma sistemática é dada por:

$$G = \begin{bmatrix}
1&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0&1&1&1&0&0&0&1\\
0&1&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&0&0&1&0&0&1\\
0&0&1&0&0&0&0&0&0&0&0&0&1&1&0&1&0&0&1&0&1&0&1\\
0&0&0&1&0&0&0&0&0&0&0&0&1&1&0&0&0&1&1&1&0&1&1\\
0&0&0&0&1&0&0&0&0&0&0&0&1&1&0&0&1&1&0&1&1&0&0\\
0&0&0&0&0&1&0&0&0&0&0&0&1&1&0&0&1&1&0&1&1&1&0\\
0&0&0&0&0&0&1&0&0&0&0&0&0&1&1&0&0&1&1&0&1&1&1\\
0&0&0&0&0&0&0&1&0&0&0&0&1&0&1&1&0&1&1&1&1&0&0\\
0&0&0&0&0&0&0&0&1&0&0&0&0&1&0&1&1&0&1&1&1&1&0\\
0&0&0&0&0&0&0&0&0&1&0&0&0&0&1&0&1&1&0&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&1&0&1&0&1&1&1&0&0&0&1&1&0\\
0&0&0&0&0&0&0&0&0&0&0&1&0&1&0&1&1&1&0&0&0&1&1
\end{bmatrix} \tag{3.8}$$

b) Fazendo todas as combinações lineares das colunas de verificação de paridade da matriz anterior, tem-se que a combinação com menor número de uns é:

$$\mathbf{t} = c_{13} + c_{15} + c_{16} + c_{18} + c_{19} + c_{20} + c_{21} \tag{3.9}$$

Portanto a matriz geradora modificada G' começa a ser montada com estas colunas da seguinte forma:

$$G'=\begin{bmatrix} 1\ 1\ 0\ 1\ 1\ 0\ 0 \\ 1\ 1\ 1\ 0\ 0\ 1\ 0 \\ 1\ 0\ 1\ 0\ 1\ 0\ 1 \\ 1\ 0\ 0\ 1\ 1\ 1\ 0 \\ 1\ 0\ 0\ 1\ 0\ 1\ 1 \\ 0\ 1\ 0\ 1\ 1\ 0\ 1 \\ 0\ 1\ 1\ 0\ 1\ 1\ 0 \\ 1\ 1\ 1\ 1\ 1\ 1\ 1 \\ 0\ 0\ 1\ 0\ 1\ 1\ 1 \\ 0\ 1\ 0\ 1\ 0\ 1\ 1 \\ 1\ 1\ 1\ 0\ 0\ 0\ 1 \\ 0\ 0\ 1\ 1\ 0\ 0\ 0 \end{bmatrix} \tag{3.10}$$

onde,

$$\mathbf{t}=\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = c_8 \tag{3.11}$$

$nu = 1$

$nc = 7$

Portanto, da equação 3.1 tem-se que:

$$N(S_1) = nu + nc = 8 \tag{3.12}$$

c) Assim, as oito primeiras colunas de G' serão:

$$G' = \begin{bmatrix} 1\ 1\ 0\ 1\ 1\ 0\ 0\ 0 \\ 1\ 1\ 1\ 0\ 0\ 1\ 0\ 0 \\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0 \\ 1\ 0\ 0\ 1\ 1\ 1\ 0\ 0 \\ 1\ 0\ 0\ 1\ 0\ 1\ 1\ 0 \\ 0\ 1\ 0\ 1\ 1\ 0\ 1\ 0 \\ 0\ 1\ 1\ 0\ 1\ 1\ 0\ 0 \\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1 \\ 0\ 0\ 1\ 0\ 1\ 1\ 1\ 0 \\ 0\ 1\ 0\ 1\ 0\ 1\ 1\ 0 \\ 1\ 1\ 1\ 0\ 0\ 0\ 1\ 0 \\ 0\ 0\ 1\ 1\ 0\ 0\ 0\ 0 \end{bmatrix} \tag{3.13}$$

Da equação 3.2 tem-se que:

$$\text{Min}R_{START} = N(S_1) - 1 = 7 \tag{3.14}$$

d) A coluna c_{23} não contém o dígito binário "1" na oitava posição (posição do um do vetor **t**), e não existe nenhuma outra coluna, ainda não utilizada, com menor quantidade de uns, que satisfaça esta condição, portanto este é um exemplo do caso 1, onde $N(S_1) \cap N(S_2) = 0$.

e) A coluna c_{23} é colocada como sendo a última coluna da matriz G':

$$G' = \begin{bmatrix} 1\ 1\ 0\ 1\ 1\ 0\ 0\ 0 & 1 \\ 1\ 1\ 1\ 0\ 0\ 1\ 0\ 0 & 1 \\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0 & 1 \\ 1\ 0\ 0\ 1\ 1\ 1\ 0\ 0 & 1 \\ 1\ 0\ 0\ 1\ 0\ 1\ 1\ 0 & 0 \\ 0\ 1\ 0\ 1\ 1\ 0\ 1\ 0 & 0 \\ 0\ 1\ 1\ 0\ 1\ 1\ 0\ 0 & 1 \\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1 & 0 \\ 0\ 0\ 1\ 0\ 1\ 1\ 1\ 0 & 0 \\ 0\ 1\ 0\ 1\ 0\ 1\ 1\ 0 & 1 \\ 1\ 1\ 1\ 0\ 0\ 0\ 1\ 0 & 0 \\ 0\ 0\ 1\ 1\ 0\ 0\ 0\ 0 & 1 \end{bmatrix} \qquad (3.15)$$

Note que:

$mnu = 7$

f) Agora serão colocadas à esquerda da última coluna de G', $mnu = 7$ colunas da submatriz de identidade de G, levando-se em conta que a posição dos uns dessas colunas devem coincidir com a posição dos uns da última coluna de G'. Esta matriz fica da seguinte forma:

$$G' = \begin{bmatrix} 1\,1\,0\,1\,1\,0\,0\,0 & 0\,0\,0\,0\,0\,0\,1\,1 \\ 1\,1\,1\,0\,0\,1\,0\,0 & 0\,0\,0\,0\,0\,1\,0\,1 \\ 1\,0\,1\,0\,1\,0\,1\,0 & 0\,0\,0\,0\,1\,0\,0\,1 \\ 1\,0\,0\,1\,1\,1\,0\,0 & 0\,0\,0\,1\,0\,0\,0\,1 \\ 1\,0\,0\,1\,0\,1\,1\,0 & 0\,0\,0\,0\,0\,0\,0\,0 \\ 0\,1\,0\,1\,1\,0\,1\,0 & 0\,0\,0\,0\,0\,0\,0\,0 \\ 0\,1\,1\,0\,1\,1\,0\,0 & 0\,0\,1\,0\,0\,0\,0\,1 \\ 1\,1\,1\,1\,1\,1\,1\,1 & 0\,0\,0\,0\,0\,0\,0\,0 \\ 0\,0\,1\,0\,1\,1\,1\,0 & 0\,0\,0\,0\,0\,0\,0\,0 \\ 0\,1\,0\,1\,0\,1\,1\,0 & 0\,1\,0\,0\,0\,0\,0\,1 \\ 1\,1\,1\,0\,0\,0\,1\,0 & 0\,0\,0\,0\,0\,0\,0\,0 \\ 0\,0\,1\,1\,0\,0\,0\,0 & 1\,0\,0\,0\,0\,0\,0\,1 \end{bmatrix} \tag{3.16}$$

Da matriz anterior e tendo em conta a equação 3.3, tem-se que:

$$N(S_2) = mnu + 1 = 8 \tag{3.17}$$

e da equação 3.4,

$$\text{Min}R_{END} = N(S_2) - 1 = 7 \tag{3.18}$$

g) Na continuação, a matriz *G'* é completada com as colunas restantes não utilizadas de *G*:

$$G' = \begin{bmatrix}
1&1&0&1&1&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&1&1\\
1&1&1&0&0&1&0&0&0&0&0&0&1&1&0&0&0&0&0&0&1&0&1\\
1&0&1&0&1&0&1&0&0&0&0&0&1&0&0&0&0&0&0&1&0&0&1\\
1&0&0&1&1&1&0&0&0&0&0&0&1&0&1&0&0&0&1&0&0&0&1\\
1&0&0&1&0&1&1&0&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0\\
0&1&0&1&1&0&1&0&0&1&0&0&1&0&1&0&0&0&0&0&0&0&0\\
0&1&1&0&1&1&0&0&0&0&0&0&0&0&1&0&0&1&0&0&0&0&1\\
1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&1&0&1&1&1&0&0&0&1&0&1&1&1&0&0&0&0&0&0&0&0\\
0&1&0&1&0&1&1&0&0&0&0&0&0&1&1&0&1&0&0&0&0&0&1\\
1&1&1&0&0&0&1&0&0&0&0&1&0&1&1&0&0&0&0&0&0&0&0\\
0&0&1&1&0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0&0&0&1
\end{bmatrix} \qquad (3.19)$$

h) Por último, da equação 3.7 tem-se que:

$$\mathrm{Min}RL_{MAX} = \mathrm{MAX}\{\mathrm{Min}R_{START} + \mathrm{Min}R_{END}\ ,\ \mathrm{Min}R_{MID}\} = \mathrm{MAX}\{7+7\ ,\ 12\} = 14 \qquad (3.20)$$

Através de busca computacional é simples achar um líder de classe lateral **m** que produz um código com os limitantes anteriores, sendo um deles:

$$\mathbf{m} = [0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0] \qquad (3.21)$$

3.4 Exemplo de Modificação de Códigos em que os Subconjuntos S_1 e S_2 se Sobrepõem

Este é um exemplo do caso 2 e pode ser visto mediante a modificação do código cíclico (15,9,4) que tem o polinômio gerador $g(X) = X^6 + X^4 + X^3 + X^2 + 1$.

a) A partir de $g(X)$, formar a matriz G na forma sistemática:

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \end{bmatrix} \tag{3.22}$$

b) Fazendo as combinações lineares das colunas de verificação de paridade de G, obtém-se que a combinação com menor número de uns é:

$$\mathbf{t} = c_{10} + c_{11} + c_{14} + c_{15} \tag{3.23}$$

Portanto a matriz geradora modificada G' começa com estas colunas:

$$G' = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} \tag{3.24}$$

onde,

$$t = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = c_2 + c_8 \tag{3.25}$$

$nu = 2$ e

$nc = 4$.

Portanto, da equação 3.1 tem-se que:

$$N(S_1) = min\_nu + min\_nc = 6 \tag{3.26}$$

c) Desta forma, as seis primeiras colunas de G' serão:

$$G' = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \end{bmatrix} \tag{3.27}$$

Da equação 3.2 tem-se que:

$$\text{Min}R_{START} = N(S_1) - 1 = 5 \tag{3.28}$$

d) Nenhuma das duas colunas ainda não utilizadas (c_{12} e c_{13}) da submatriz de verificação de paridade de G tem zeros nas duas posições dos uns do vetor **t**, portanto, $N(S_1) \cap N(S_2) \neq 0$.

e) Neste caso serão colocadas na continuação de S_1 , essas duas colunas:

$$G' = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \end{bmatrix} \tag{3.29}$$

O vetor soma dessas duas colunas será:

$$\mathbf{q} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \tag{3.30}$$

e o número de uns do vetor **q** cujas posições coincidem com posições de uns do vetor **t** é $r = 2$.

Note que o vetor **q** contém uns nas mesmas $r = 2$ posições que o vetor **t**, portanto, não importa a ordem das últimas duas colunas de S_1. Essas colunas então vão ser comuns tanto a S_1 como a S_2.

f) Agora serão colocadas à direita dessas duas colunas, as colunas da submatriz de identidade de G em que a posição dos uns coincide com a posição dos uns restantes de **q**:

$$G' = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix} \tag{3.31}$$

g) Na continuação, a matriz G' é completada com as restantes colunas de G:

$$G' = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.32}$$

Da matriz anterior, e tendo em conta a equação 3.5, tem-se que:

$$N(S_2) = n - N(S_1) + r = 15 - 6 + 2 = 11 \tag{3.33}$$

e da equação 3.6,

$$\text{Min}R_{END} = N(S_2) - 1 = 10 \tag{3.34}$$

Uma vez encontrados esses dois limitantes tem-se, da equação 3.7, que:

$$\text{Min}RL_{MAX} = \text{MAX}\{\text{Min}R_{START} + \text{Min}R_{END}\ ,\ \text{Min}R_{MID}\} = \text{MAX}\{15\ ,\ 9\} = 15. \tag{3.35}$$

Um líder de classe lateral que produz um código com esse limitante é:

$$\mathbf{m} = [0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0] \tag{3.36}$$

3.5 Exemplo de Modificação de Códigos de Hamming

Como exemplo deste caso pode-se ver a modificação do código BCH (15,11) que tem o polinômio gerador $g(X) = X^4 + X + 1$.

a) A matriz G na forma sistemática é:

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \end{bmatrix} \tag{3.37}$$

Da matriz anterior pode-se ver que as quatro colunas de verificação de paridade tem sete uns cada, e que a combinação linear com menor quantidade de uns é a formada

pela soma de todas elas, contendo um total de quatro uns. Neste caso, ao executar o passo *b* do método, devem-se colocar ao início de *G'*, uma combinação de três dessas colunas (qualquer combinação de três colunas vai ter um total de cinco uns). Assim, a matriz *G'* pode começar da seguinte forma:

$$G' = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \qquad (3.38)$$

A partir daí o método continua da forma descrita anteriormente sendo que este é um exemplo do caso 2. Portanto a matriz *G'* resultante fica:

$$G' = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad (3.39)$$

onde,

$\text{Min}R_{START} = 7$,

$\text{Min}R_{END} = 10$ e

$\text{Min}RL_{MAX} = 17$.

Um vetor modificador que produz um código com esses limitantes é:

$$\mathbf{m} = [0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0] \tag{3.40}$$

3.6 Capacidade de Correção e/ou Detecção de Erros dos Códigos de Bloco com "Runlength" Limitado.

Uma vez encontrado o líder de classe lateral adequado para produzir o código com "runlength" limitado, a codificação da seqüência de informação pode ser feita da maneira convencional, mas antes da transmissão das palavras código, adiciona-se o vetor **m** a cada uma delas, garantindo-se que a seqüência de dados enviada pelo canal esteja limitada no "runlength". No receptor é feita a operação inversa e portanto, antes da correção de erros, modifica-se a palavra recebida para ser entregue ao decodificador.

Sejam

$t(X)$: polinômio que representa a palavra código,

$i(X)$: polinômio de menor grau do que k, que representa a seqüência de informação,

$M(X)$: polinômio que representa o vetor modificador,

então, as palavras código geradas pelos códigos de bloco com "runlength" limitado podem ser representadas como:

$$t(X) = i(X)g(X) + M(X) \tag{3.41}$$

As propriedades de correção e/ou detecção de erros destes códigos vão ser preservadas. Se $t(X)$ é transmitido e $r(X) = t(X) + e(X)$ é recebido, então , no receptor o primeiro passo feito é a subtração de $M(X)$ do polinômio recebido $r(X)$. Como resultado é obtido o polinômio $i(X)g(X) + e(X)$ e a correção de erros pode ser feita com o decodificador do código original.

3.7 Resultados

Na tabela 3.1 são mostrados os resultados obtidos na modificação de alguns códigos cíclicos transparentes conhecidos.

Os limitantes inferiores de $\text{Min}R_{START}$, $\text{Min}R_{END}$ e $\text{Min}RL_{MAX}$ são os valores mínimos possíveis de serem encontrados para esses códigos.

A posição dos uns do vetor modificador representa a posição que ocupam os dígitos binários "1" na palavra de n bits que representa o vetor modificador **m**, onde $\mathbf{m} = m_1 m_2 ... m_n$. Por exemplo, no caso particular do código BCH (15,5) as posições representadas por (12,4,10) significam que o vetor **m** tem uns nas posições m_{12}, m_4 e m_{10}, ou seja, $\mathbf{m} = [0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 0\ 0]$.

Código	$MinR_{START}$	$MinR_{END}$	$MinRL_{MAX}$	*Posições dos uns do Vetor Modificador*
BCH (15,5)	3	3	6	(12,4,10)
BCH (15,7)	3	3	7	(15,9,1)
BCH (15,11)	7	10	17	(5)
BCH (31,16)	7	7	16	(24,8,22)
BCH (31,21)	11	20	31	(13,10,22)
BCH (31,26)	15	22	37	(10)
GOLAY (23,12)	7	7	14	(16,8)
(15,7,3)	3	3	7	(13,7,2)
(15,9,3)	5	9	14	(6)
(15,9,4)	5	10	15	(5)
(21,12,5)	7	7	14	(14,6)

Tabela 3.1: Códigos Cíclicos Transparentes Modificados.

3.8 Conclusão

Neste capítulo foi apresentado um método para a obtenção de códigos cíclicos com "runlength" limitado a partir de códigos cíclicos transparentes conhecidos. O procedimento é mais simples do que o apresentado no capítulo 2 pois é baseado no reordenamento da matriz geradora do código na forma sistemática sem ter que utilizar a matriz de verificação de paridade. Por isso, a sua implementação é fácil de ser feita

através de programas computacionais. Os programas através dos quais foram desenvolvidos os exemplos mostrados neste capítulo estão no apéndice A.

Capítulo 4

CARATERÍSTICAS ESPECTRAIS DOS CÓDIGOS DE LINHA CORRETORES DE ERROS

4.1 Introdução

Um dos objetivos fundamentais do trabalho com códigos corretores de erro com "runlength" limitado é a diminuição do conteúdo espectral do sinal transmitido na região das baixas freqüências. Para se calcular a densidade espectral de potência do sinal binário enviado pelo canal de comunicações é preciso ter em conta que, em geral, uma seqüência de palavras código pode ser considerada como um processo estocástico cuja estatística vai depender da estatística dos símbolos de informação e das características do código.

Sem perda de generalidade, pode-se supor que a seqüência de entrada do codificador, $i_n\ (-\infty < n < +\infty)$ é um processo estacionário onde os símbolos i_n são estatisticamente independentes. No caso dos códigos de bloco a seqüência de símbolos codificados na saída do codificador não é mais, de maneira geral, um processo estacionário, pois a estacionaridade da seqüência de símbolos de entrada só implica estacionaridade na seqüência de palavras código de saída e não necessariamente na seqüência de símbolos dessas palavras código.

Portanto, o sinal codificado é um processo cicloestacionário com período igual ao tempo de transmissão de uma palavra código, $T_m = mT$, onde T é o tempo de transmissão de um símbolo e m é o comprimento das palavras código [Cariolaro, 1974].

Define-se um processo cicloestacionário como sendo um processo estocástico cujas propriedades estatísticas se repetem num determinado período de tempo.

4.2 Densidade Espectral de Potência de um Código Transparente

Existem diferentes procedimentos para calcular a função densidade espectral de potência, $S(\omega)$, de seqüências codificadas. Todos eles envolvem o cálculo da seguinte soma infinita de cosenos [Popplewell, 1992]:

$$S(\omega) = R(0) + 2\sum_{a=1}^{\infty} R(a)\cos(\omega a) \tag{4.1}$$

onde $R(a)$ é a função de autocorrelação da seqüência codificada $X(n)$.

Quando a seqüência codificada é obtida a partir de um código transparente, pode-se fazer simplificações para reduzir a equação 4.1 a uma série finita de m termos, onde m é o comprimento das palavras código. Isto pode ser feito da seguinte forma:

Seja $Y(n)$ a seqüência discreta de fase aleatória que corresponde à seqüência codificada cicloestacionária $X(n)$ de palavras código de comprimento m.

Então,

$$R_Y(a) = E\{Y(n)Y(n+a)\} \tag{4.2}$$

$$R\ (a) = \frac{1}{m}\sum_{=1} E\{X(n)X(n+a)\}, \quad a \in \{...-1,0,1,...\} \tag{4.3}$$

devido à fase aleatória.

Se é definido A_a como o evento: $X(n)$ e $X(n + a)$ estão em diferentes palavras código e A_a^c como o complemento do evento anterior, então a equação 4.3 pode ser escrita como

$$R\ (a) = \frac{1}{m}\sum_{=1}\left\{P(A_a)E\left\{X(n)X(n+a)\middle|A_a\right\} + P(A_a)E\left\{X(n)X(n+a)\middle|A_a\right\}\right\} \tag{4.4}$$

Sejam,

$$Z_1(a) = P(A_a)E\left\{X(n)X(n+a)\middle|A_a\right\} \tag{4.5}$$

e

$$Z_2(a) = P(A_a^c)E\left\{X(n)X(n+a)\middle|A_a^c\right\} \tag{4.6}$$

Considerando $Z_1(a)$:

$$Z_1(a) = P(A_a)E\left\{X(n)X(n+a)\middle|A_a\right\}$$

$$= P(A_a)\sum_{i=-1,1}\sum_{j=-1,1} P(X(n)=i)iP\left(X(n+a)=j\middle|X(n)=i,A_a\right)j \tag{4.7}$$

Devido ao condicionamento sob o evento A_a , as variáveis aleatórias $X(n)$ e $X(n+a)$ são independentes, portanto:

$$Z_1(a) = P(A_a)\left\{\sum_{i=-1,1} P(X(n)=i)i \sum_{j=-1,1} P(X(n+a)=j|A_a)j\right\} \quad (4.8)$$

Supondo que todas as palavras códigos são equiprováveis e que o código é transparente, tem-se que:

$$P(X(n)=i) = P(X(n)=j) = \frac{1}{2} \text{ , para todo } i,j \in \{-1,1\} \quad (4.9)$$

portanto:

$$Z_1(a) = P(A_a)\left\{\frac{1}{2}\sum_i i \sum_j j\right\} \quad (4.10)$$

e

$$\sum_i i = \sum_j j = 0 \text{ desde que } i,j \in \{-1,1\}$$

assim:

$$Z_1(a) = 0 \quad (4.10)$$

Considerando agora $Z_2(a)$:

$$Z_2(a) = P(A_a^c)E\{X(n)X(n+a)|A_a^c\} \quad a = \ldots -1,0,1,\ldots \quad (4.11)$$

Como $X(n)$ e $X(n+a)$ estão na mesma palavra código, a equação 4.11 pode ser escrita como:

$$Z_2(a) = P(A_a^c)E\{X(n)X(n+a)|A_a^c\} \quad a = -m+1,\ldots -1,0,1,\ldots,m-1 \quad (4.12)$$

Assim, substituindo as equações 4.10 e 4.12 na equação 4.3, tem-se que:

$$R_Y(a)=\begin{cases}\dfrac{1}{m}\displaystyle\sum_{n=1}^{m}P\left(A_a^c\right)E\left\{X(n)X(n+a)\middle|A_a^c\right\}, & a=-m+1,\ldots,-1,0,1,\ldots,m-1\\ 0 & ,\ |a|\geq m\end{cases} \tag{4.13}$$

ou

$$R_Y=\begin{cases}\dfrac{1}{m}\displaystyle\sum_{n=1}^{m-a}E\{X(n)X(n+a)\} & ,\ a=-m+1,\ldots,-1,0,1,\ldots,m-1\\ 0 & ,\ |a|\geq m\end{cases} \tag{4.14}$$

Substituindo a equação 4.14 na equação 4.1, tem-se que:

$$S(\omega)=\frac{1}{m}\sum_{n=1}^{m}E\{X(n)X(n)\}+2\sum_{a=1}^{m-1}\frac{1}{m}\sum_{n=1}^{m-a}E\{X(n)X(n+a)\cos(\omega a)\} \tag{4.15}$$

4.3 Resultados

Tendo em conta o resultado obtido na equação 4.15, uma estimativa da densidade espectral de potência de seqüências codificadas segundo o método do capítulo 3, pode ser feita usando-se o método das médias de peridiogramas de Welch [Oppenheim, 1989].

Neste método, a seqüência de símbolos codificados $X(n)$ é dividida em segmentos de L amostras com uma janela de comprimento L aplicada a cada um desses segmentos. A seguir é calculada a estimativa da densidade espectral de potência de cada segmento e logo após a média sob todos os segmentos.

No caso particular das seqüências X(n) serem obtidas a partir de um código de linha corretor de erro, o valor de L deve ser igual ao valor de m, o comprimento das palavras código, dado que da equação 4.14, pode-se notar que $R_y = 0$ para $|a| \geq m$.

Como exemplo, a figura 4.1 mostra a região das baixas freqüências do espectro de potência do código BCH (15,5) original e do mesmo código logo após ser modificado com um vetor modificador adequado para se alcançar o limitante mínimo do "runlength".

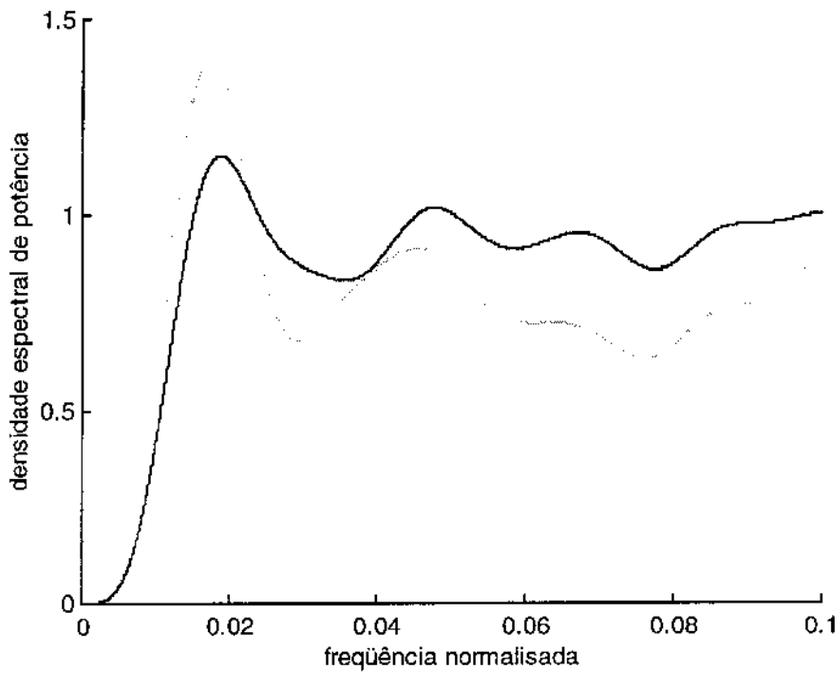

Fig. 4.1 Espectro de Potência de um Código de Linha Corretor de Erro.

..... **m** = 000100000101000

——— **m** = 000000000000000

Na figura anterior, a curva mais escura representa a densidade espectral de potência do código de linha corretor de erro formado a partir do código cíclico BCH (15,5) onde o vetor $\mathbf{m} = 000100000101000 = (12,4,10)$ foi somado a todas as palavras códigos. Com este vetor modificador é alcançado o limitante inferior do "runlength" para este código (ver tabela 3.1).

A curva menos escura representa a densidade espectral de potência do mesmo código sem modificar. Note-se que o código modificado apresenta uma redução considerável no conteúdo espectral nas freqüências mais baixas.

4.4 Conclusão

Neste capítulo foram vistas as caraterísticas espectrais dos códigos de linha corretores de erro e o efeito do uso de vetores modificadores nesses códigos. Dos resultados obtidos aqui pode-se concluir que a utilização de vetores modificadores que fazem com que o código específico atinja o limitante inferior do "runlength" calculado segundo o método do capítulo 3, vai levar a mudanças na forma do espectro de freqüências do código, especificamente vai acontecer uma redução do conteúdo espectral na região das baixas freqüências. Isto é de grande importância para aplicações em que o canal de comunicações não tenha uma boa resposta nessa região.

Portanto, o uso de vetores modificadores ótimos além de reduzir o "runlength" do código, vai melhorar as caraterísticas espectrais do mesmo.

Capítulo 5

CONCLUSÕES FINAIS

5.1 Comentários e Comparações sobre Códigos de Linha Corretores de Erro

O método para encontrar códigos com "runlength" limitado apresentado neste trabalho (capítulo 3), mostra ser superior em alguns aspectos ao método de Popplewell [1992], apresentado no capítulo 2.

O método de Popplewell começa com o reordenamento da matriz geradora na forma não sistemática de acordo com os teoremas 1, 2 e 3, enunciados no capítulo 2 e daí é feito um procedimento de uma certa complexidade para reordenar a matriz de verificação de paridade segundo o método descrito nesse capítulo. Uma vez encontrada a matriz de verificação de paridade que satisfaz as condições impostas pelo método, mediante a solução de uma equação matricial (equação 2.8) é encontrado o vetor

modificador que, somado a todas as palavras do código original, vai levar à obtenção do código com "runlength" limitado. Se a matriz de verificação de paridade não pode ser reordenada segundo as condições impostas no capítulo 2, então não é possível alcançar os limitantes garantidos pelos teoremas 1, 2 e 3.

No método proposto no capítulo 3, os teoremas 1, 2 e 3 são aplicados à matriz geradora do código na forma sistemática. Uma vez encontrados os valores para se alcançar o limitante inferior para o "runlength" do código, mediante um algoritmo simples, encontra-se o vetor modificador para obter o código desejado através do líder de coset adequado. Desta forma não é preciso ter que utilizar a matriz de verificação de paridade no processo e os limitantes garantidos pelos teoremas são alcançados de forma exata.

Na tabela 5.1 são mostrados, de forma comparativa, os resultados obtidos a partir dos dois métodos anteriores.

Código Corretor de Erro	$MinRL_{MAX}$ *(método de Popplewell)*	$MinRL_{MAX}$ *(método do capítulo 3)*
BCH (15,5)	6	6
BCH (15,7)	7	7
BCH (15,11)	17	17
BCH (3116)	16	16
BCH (31,21)	32	32
BCH (31,26)	38	37
GOLAY (23,12)	16	14

Tabela 5.1: Comparação entre os Limitantes do "Runlength" Obtidos nos Capítulos 2 e 3.

Da tabela anterior, pode-se notar que os resultados obtidos com ambos os métodos são similares. Mais ainda, no caso específico dos códigos BCH (31,26) e GOLAY (23,12), (últimas duas linhas da tabela 5.1), os valores de MinRL_{MAX} obtidos pelo método do capítulo 3 são menores do que os obtidos pelo método de Popplewell.

O algoritmo desenvolvido para encontrar o vetor modificador que leva ao código com "runlength" limitado é ótimo no sentido que a quantidade de uns que vai ter esse vetor será a menor possível a diferença do método de Popplewell. Por exemplo, o vetor **m** obtido no exemplo do capítulo 2 para modificar o código BCH (15,5) segundo o método de Popplewell contem 5 uns enquanto que a quantidade de uns desse vetor segundo o método do capítulo 3 é igual a 3 (ver tabela 3.1). Isto leva a uma menor complexidade no codificador pois serão dois bits a menos a serem somados as palavras do código.

No capítulo 4, foi comprovado que os códigos com "runlength" limitado apresentam uma redução do conteúdo espectral na região das baixas freqüências. Portanto, o fato de limitar inferiormente a quantidade de símbolos iguais consecutivos na seqüência de dados enviada pelo canal, além de facilitar a operação de sincronização no receptor, vai melhorar as caraterísticas espectrais do código, facilitando sua utilização em canais que não tenham uma boa resposta nessa região. A figura 5.1 mostra o densidade espectral de potência do código BCH (15,5) modificado pelos dois métodos anteriormente mencionados. Note-se que praticamente não existe nenhuma diferença entre as duas curvas.

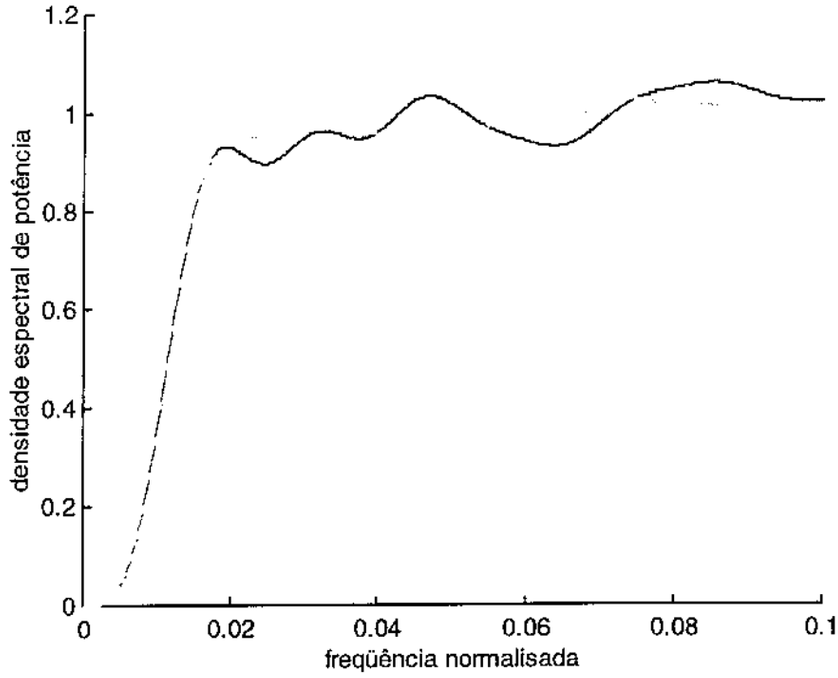

Figura 5.1: Espectro de Potência do Código BCH (15,5) Modificado.

..... Segundo o método de Popplewell

——— Segundo o método do capítulo 3

5.2 Futuros Trabalhos

Como recomendação para futuros trabalhos seria interessante explorar a possibilidade de encontrar um limitante inferior para a quantidade mínima de símbolos iguais enviados pelo canal com a utilização de códigos de linha com "runlength" limitado. Desta forma, a seqüência de dados enviados pelo canal além de garantir boas

condições para o sincronismo no receptor, vai ser mais imune à interferência intersimbólica.

Uma outra proposta para futuros trabalhos pode ser explorar a possibilidade de projetar códigos de linha corretores de erro que possam serem modificados pelo procedimento descrito aqui sem ter que partir de códigos de bloco conhecidos.

5.3 Conclusão Final

Neste trabalho é apresentado um procedimento simples para obter códigos corretores de erro com boas propriedades de codificação de linha. Os aspectos mais importantes a serem sinalizados sobre o método desenvolvido são:

- O ponto de partida para a obtenção dos códigos modificados é a matriz geradora do código na forma sistemática. Isto facilita o processo de codificação.

- Não é preciso trabalhar com a matriz de verificação de paridade.

- A quantidade de uns dos vetores modificadores obtidos é mínima. Isto simplifica o codificador.

- Os algoritmos para a modificação dos códigos são simples de serem implementados mediante programas computacionais.

- Os resultados obtidos em alguns dos códigos testados são melhores do que em trabalhos publicados anteriormente [Popplewell, 1992].

- Uma vez encontrado o vetor apropriado para modificar um código, a complexidade de implementação do codificador para o código modificado praticamente não aumenta com respeito ao código original.

- O vetor modificador encontrado para todos os códigos testados garante de forma exata os limitantes estabelecidos segundo os teoremas do capítulo 2.

Finalmente deve ser destacado que as principais vantagens do método apresentado neste trabalho são sua simplicidade e generalidade pois ele pode ser aplicado a qualquer código de bloco transparente sem que sejam produzidas mudanças na estrutura algébrica dos mesmos nem nas propriedades de correção e/ou detecção de erros desses códigos.

Apéndice A

Programas de Computação Utilizados na Obtenção de Códigos de Linha com "Runlength" Limitado

A.1 Programa para Modificar o Código do Exemplo 3.3

```
#include <stdio.h>
#include <math.h>
static int g[65][65];

main()
{

        static int i,j,k,n,nu,min_nu,l,s,nc,min_nc,w,mnu,z,c,rstart,rend,nze,nzd,te;
        static int i1,i2,i3,i4,i5,i6,i7,i8,i9,i10,i11,i12;
        static int x[58],u[58],b[200][58],d[58],p[58][58],t[58];
        static int r=0,m,nzm,rll;
        int e=0;
        FILE *o,*fopen();
```

```
/*entrada do polinômio gerador*/
printf("k=");
scanf("%d", &k);
printf("n=");
scanf("%d", &n);

printf("entre com o polinomio gerador g(X)\n");
for(i=k;i<=n;++i){
        printf("X**%d = ", n-i);
        scanf("%d", &g[k][i]);
}

/*formação da matriz geradora na forma sistemática*/

for(i=k-1;i>=1;--i){
        g[i][n]=g[i+1][1];
        for(j=1;j<=n-1;++j)
                g[i][j]=g[i+1][j+1];
        if(g[i][k]!=0){
                for(l=1;l<=n;++l)
                        g[i][l]=(g[i][l]+g[k][l])%2;
        }
}
/*modificação da matriz geradora segundo os teoremas 1, 2 e 3*/
min_nu=k;
min_nc=n-k;
nc=0;

for(i11=0;i11<=1;++i11){
for(i10=0;i10<=1;++i10){
for(i9=0;i9<=1;++i9){
for(i8=0;i8<=1;++i8){
for(i7=0;i7<=1;++i7){
for(i6=0;i6<=1;++i6){
for(i5=0;i5<=1;++i5){
for(i4=0;i4<=1;++i4){
for(i3=0;i3<=1;++i3){
for(i2=0;i2<=1;++i2){
for(i1=0;i1<=1;++i1){
u[k+11]=i11;
u[k+10]=i10;
u[k+9]=i9;
```

```
u[k+8]=i8;
u[k+7]=i7;
u[k+6]=i6;
u[k+5]=i5;
u[k+4]=i4;
u[k+3]=i3;
u[k+2]=i2;
u[k+1]=i1;
nu=0;
for(i=1;i<=k;++i){
        x[i]=0;
        for(j=k+1;j<=n;++j)
                x[i]=(x[i]+u[j]*g[i][j])%2;
        if(x[i]==1)
                ++nu;
}
if(nu!=0){
        if(nu<=min_nu){
                if(nu<min_nu){
                        r=1;
                for(j=k+1;j<=n;++j)
                                b[r][j]=u[j];
                        min_nu=nu;
                }
                else{
                        ++r;
                        for(j=k+1;j<=n;++j)
                                b[r][j]=u[j];
                }
        }
}

}
}
}
}
}
}
}
}
}
}
}
```

```
for(i=1;i<=r;++i){
        for(j=k+1;j<=n;++j){
                if(b[i][j]!=0)
                        ++nc;
        }
        if(nc<=min_nc){
                min_nc=nc;
                for(j=k+1;j<=n;++j)
                        d[j]=b[i][j];
        }
}
l=0;
s=min_nc+min_nu;
w=n+1;
r=0;
mnu=k;
for(j=k+1;j<=n;++j){
        if(d[j]!=0){
                ++l;
                for(i=1;i<=k;++i){
                        p[i][l]=g[i][j];
                        t[i]=(t[i]+p[i][l])%2;
                }
        }
}
for(j=k+1;j<=n;++j){
        r=0;
        if(d[j]==0){
                for(i=1;i<=k;++i){
                        if(g[i][j]==1)
                                ++r;
                }
                if(r<=mnu){
                        mnu=r;
                        c=0;
                        for(i=1;i<=k;++i){
                                if(t[i]==0)
                                        z=j;
                                else{
                                        if(g[i][j]==0)
                                                z=j;
                                        else
                                                ++c;
```

```
                        }
                    }
                    if(c==0){
                        for(i=1;i<=k;++i)
                            p[i][n]=g[i][j];
                    }
                }
            }
        }
        --w;
        for(i=1;i<=k;++i){
            if(t[i]==1){
                ++l;
                p[i][l]=1;
            }
            else{
                if(p[i][n]==1){
                    --w;
                    p[i][w]=1;
                }
                else{
                    ++s;
                    p[i][s]=1;
                }
            }
        }
        for(j=k+1;j<=n;++j){
            if(d[j]==0){
                if(j!=z){
                    ++s;
                    for(i=1;i<=k;++i)
                        p[i][s]=g[i][j];
                }
            }
        }
        o=fopen("Ngolay.txt","w");
        fprintf(o,"CODIGO GOLAY (23_12)\n");
        fprintf(o,"\n matriz geradora:\n");
        for(l=1;l<=k;++l){
            for(i=1;i<=n;++i)
                fprintf(o," %d", g[l][i]);
            fprintf(o,"\n");
        }
```

```
fprintf(o,"\n matriz modificada:\n");
for(i=1;i<=k;++i){
        for(j=1;j<=n;++j)
                fprintf(o," %d", p[i][j]);
        fprintf(o,"\n");
}
rstart=min_nc+min_nu-1;
fprintf(o,"R_start=%d",rstart);
fprintf(o,"\n");
rend=n-w;
fprintf(o,"R_end=%d", rend);
fprintf(o,"\n");
if((rstart+rend)>k){
        rll=rstart+rend;
        fprintf(o,"RLL= %d", rll);
}
else{
        rll=k;
        fprintf(o,"RLL= %d", rll);
}
fprintf(o,"\n");

/*busca do vetor modificador*/
nzd=0;
nze=0;
nzm=0;

for(m=n-rend;m<=n;++m){
for(j=rstart+1;j>=1;--j){
for(i12=0;i12<=1;++i12){
for(i11=0;i11<=1;++i11){
for(i10=0;i10<=1;++i10){
for(i9=0;i9<=1;++i9){
for(i8=0;i8<=1;++i8){
for(i7=0;i7<=1;++i7){
for(i6=0;i6<=1;++i6){
for(i5=0;i5<=1;++i5){
for(i4=0;i4<=1;++i4){
for(i3=0;i3<=1;++i3){
for(i2=0;i2<=1;++i2){
for(i1=0;i1<=1;++i1){
```

```
                for(i=1;i<=n;++i)
                        x[i]=(i1*p[1][i]+i2*p[2][i]+i3*p[3][i]+i4*p[4][i]+i5*p[5
                        ][i]+i6*p[6][i]+i7*p[7][i]+i8*p[8][i]+i9*p[9][i]+i10*
                      p[10][i]+i11*p[11][i]+i12*p[12][i])%2;
                x[m]=(x[m]+1)%2;
                x[j]=(x[j]+1)%2;
                w=0;
                for(i=n;i>=1;--i){
                        if(x[i]!=0)
                                goto a;
                        ++w;
                }
a:              if(w>nzd)
                        nzd=w;
                te=0;
                for(i=1;i<=n;++i){
                        if(x[i]!=0)
                                goto y;
                        ++te;
                }
y:              if(te>nze)
                        nze=te;
                e=0;
                for(i=1;i<=n;++i){
                        if(x[i]==0){
                                ++e;
                                if(e>nzm)
                                        nzm=e;
                        }
                        else
                                e=0;
                }

        }
        }
        }
        }
        }
        }
        }
        }
        }
```

```
        }
        }


        if(nze==rstart&&nzd==rend&&nzm<=rll)
                goto f;

        nzd=0;
        nze=0;
        nzm=0;

        }
        }
 f:     fprintf(o,"\n posicao dos uns= (%d,%d)", m,j);
        fprintf(o,"\n");
        fclose(o);
}
```

A.2 Programa para Modificar o Código do Exemplo 3.4

```
#include <stdio.h>
#include <math.h>
static int g[65][65];
main()
{
        static int i,j,k,n,nu,min_nu,l,s,nc,min_nc,v,w,mnu,rstart,rend,nze,nzd,te;
        static int i1,i2,i3,i4,i5,i6,i7,i8,i9;
        static int x[58],u[58],b[200][58],d[58],p[58][58],t[58],q[58];
        int r=0,m,nzm,e,rll=0;
        FILE *o,*fopen();

/*entrada do polinômio gerador*/

        printf("k=");
        scanf("%d", &k);
        printf("n=");
        scanf("%d", &n);
```

```
printf("entre com o polinomio gerador g(X)\n");
for(i=k;i<=n;++i){
        printf("X**%d = ", n-i);
        scanf("%d", &g[k][i]);
}

/*formação da matriz geradora na forma sistemática*/

for(i=k-1;i>=1;--i){
        g[i][n]=g[i+1][1];
        for(j=1;j<=n-1;++j)
                g[i][j]=g[i+1][j+1];
        if(g[i][k]!=0){
                for(l=1;l<=n;++l)
                        g[i][l]=(g[i][l]+g[k][l])%2;
        }
}

/*modificação da matriz geradora segundo os teoremas 1, 2 e 3*/
min_nu=k;
min_nc=n-k;
nc=0;
for(i6=0;i6<=1;++i6){
for(i5=0;i5<=1;++i5){
for(i4=0;i4<=1;++i4){
for(i3=0;i3<=1;++i3){
for(i2=0;i2<=1;++i2){
for(i1=0;i1<=1;++i1){
u[k+6]=i6;
u[k+5]=i5;
u[k+4]=i4;
u[k+3]=i3;
u[k+2]=i2;
u[k+1]=i1;
nu=0;
for(i=1;i<=k;++i){
        x[i]=0;
        for(j=k+1;j<=n;++j)
                x[i]=(x[i]+u[j]*g[i][j])%2;
        if(x[i]==1)
                ++nu;
}
```

```
if(nu!=0){
        if(nu<=min_nu){
                if(nu<min_nu){
                        r=1;
                        for(j=k+1;j<=n;++j)
                                b[r][j]=u[j];
                        min_nu=nu;
                }
                else{
                        ++r;
                        for(j=k+1;j<=n;++j)
                                b[r][j]=u[j];
                }
        }
}

}
}
}
}
}
}
for(i=1;i<=r;++i){
        for(j=k+1;j<=n;++j){
                if(b[i][j]!=0)
                        ++nc;
        }
        if(nc<=min_nc){
                min_nc=nc;
                for(j=k+1;j<=n;++j)
                        d[j]=b[i][j];
        }
}
l=0;
s=min_nc+min_nu;
w=n+1;
r=0;
mnu=k;
for(j=k+1;j<=n;++j){
        if(d[j]!=0){
                ++l;
                for(i=1;i<=k;++i)
                        p[i][l]=g[i][j];
```

```
            }
            else{
                    ++s;
                    for(i=1;i<=k;++i)
                            p[i][s]=g[i][j];
            }
    }
    for(i=1;i<=k;++i){
            for(j=min_nc+min_nu+1;j<=n-k+min_nu;++j)
                    q[i]=(q[i]+p[i][j])%2;
    }
    r=min_nc+min_nu+1;
    s=min_nc;
    v=n-k+min_nu;
    w=n+1;
    for(i=1;i<=k;++i){
            for(j=1;j<=min_nc;++j)
                    t[i]=(t[i]+p[i][j])%2;
            if(t[i]==1){
                    if(q[i]==1){
                            --r;
                            p[i][r]=1;
                    }
                    else{
                            ++s;
                            p[i][s]=1;
                    }
            }
            else{
                    if(q[i]==1){
                            ++v;
                            p[i][v]=1;
                    }
                    else{
                            --w;
                            p[i][w]=1;
                    }
            }
    }
    o=fopen("Nc15_9_4.txt","w");
    fprintf(o,"CODIGO (15_9_4)\n");
    fprintf(o,"\n matriz geradora:\n");
    for(l=1;l<=k;++l){
```

```
        for(i=1;i<=n;++i)
                fprintf(o," %d", g[l][i]);
        fprintf(o,"\n");
}

fprintf(o,"\n matriz modificada:\n");
for(i=1;i<=k;++i){
        for(j=1;j<=n;++j)
                fprintf(o," %d", p[i][j]);
        fprintf(o,"\n");
}
rstart=min_nc+min_nu-1;
fprintf(o,"R_start=%d",rstart);
fprintf(o,"\n");
rend=n-r;
fprintf(o,"R_end=%d",rend);
fprintf(o,"\n");
if((rstart+rend)>k){
        rll=rstart+rend;
        fprintf(o,"RLL= %d", rll);
}
else{
        rll=k;
        fprintf(o,"RLL= %d", rll);
}
fprintf(o,"\n");

/*busca do vetor modificador*/

nzd=0;
nze=0;
nzm=0;
s=0;
for(m=n;m>=1;--m){
for(i9=0;i9<=1;++i9){
for(i8=0;i8<=1;++i8){
for(i7=0;i7<=1;++i7){
for(i6=0;i6<=1;++i6){
for(i5=0;i5<=1;++i5){
for(i4=0;i4<=1;++i4){
for(i3=0;i3<=1;++i3){
for(i2=0;i2<=1;++i2){
for(i1=0;i1<=1;++i1){
```

```
                for(i=1;i<=n;++i)
                        x[i]=(i1*p[1][i]+i2*p[2][i]+i3*p[3][i]+i4*p[4][i]+i5*p[5
                        ][i]+i6*p[6][i]+i7*p[7][i]+i8*p[8][i]+i9*p[9][i])%2;
                x[m]=(x[m]+1)%2;
                w=1;
                for(i=n;i>=2;--i){
                        if(x[i]!=x[i-1])
                                goto a;
                        ++w;
                }
a:              if(w>nzd)
                        nzd=w;
                te=1;
                for(i=1;i<=n;++i){
                        if(x[i]!=x[i+1])
                                goto y;
                        ++te;
                }
y:              if(te>nze)
                        nze=te;
                e=1;
                for(i=1;i<=n;++i){
                        if(x[i]==x[i+1]){
                                ++e;
                                if(e>nzm)
                                        nzm=e;
                        }
                        else
                                e=1;
                }
        }
        }
        }
        }
        }
        }
        }
        }

        if(nze==rstart&&nzd==rend&&nzm<=rll)
                goto f;
```

```
        nzd=0;
        nze=0;
        nzm=0;
        s=0;
        }
f:      fprintf(o,"\n posicao dos uns= %d", m);
        fprintf(o,"\n");
        fclose(o);
}
```

A.3 Programa para Modificar o Código do Exemplo 3.5

```
#include <stdio.h>
#include <math.h>
static int g[65][65];
extern unsigned_stklen=10000;
main(){
        static int i,j,k,n,min_nu,l,s,v,w,rstart,rend,nze,nzd,m;
        static int i1,i2,i3,i4,i5,i6,i7,i8,i9,i10,i11;
        static int x[58],d[58],p[58][58];
        static int r=0,rll,nzm,te;
        static int e=0;
        FILE *o,*fopen();

        /*entrada do polinômio gerador*/
        printf("k=");
        scanf("%d", &k);
        printf("n=");
        scanf("%d", &n);

        printf("entre com o polinomio gerador g(X)\n");
        for(i=k;i<=n;++i){
                printf("X**%d = ", n-i);
                scanf("%d", &g[k][i]);
        }

        /*formação da matriz geradora na forma sistemática*/
```

```
for(i=k-1;i>=1;--i){
        g[i][n]=g[i+1][1];
        for(j=1;j<=n-1;++j)
                g[i][j]=g[i+1][j+1];
        if(g[i][k]!=0){
                for(l=1;l<=n;++l)
                        g[i][l]=(g[i][l]+g[k][l])%2;
        }
}

/*modificação da matriz geradora segundo os teoremas 1, 2 e 3*/

min_nu=0;
for(j=k+1;j<=n-1;++j){
        for(i=1;i<=k;++i)
                p[i][j-k]=g[i][j];
}
for(i=1;i<=k;++i){
        for(j=1;j<=n-k-1;++j)
                d[i]=(d[i]+p[i][j])%2;
        if(d[i]==1)
                ++min_nu;
}
s=n-k+min_nu;
for(i=1;i<=k;++i)
        p[i][s]=g[i][n];

l=n-k-1;
r=v=s;
w=n+1;
for(i=1;i<=k;++i){
        if(d[i]==1){
                if(p[i][s]==1){
                        --r;
                        p[i][r]=1;
                }
                else{
                        ++l;
                        p[i][l]=1;
                }
        }
        else{
```

```
                if(p[i][s]==1){
                        ++v;
                        p[i][v]=1;
                }
                else{
                        --w;
                        p[i][w]=1;
                }
        }
}

o=fopen("bch15_11.txt","w");
fprintf(o,"CODIGO BCH (15_11)\n");
fprintf(o,"\n matriz geradora:\n");
for(l=1;l<=k;++l){
        for(i=1;i<=n;++i)
                fprintf(o," %d", g[l][i]);
        fprintf(o,"\n");
}
fprintf(o,"\n matriz modificada:\n");
for(i=1;i<=k;++i){
        for(j=1;j<=n;++j)
                fprintf(o," %d", p[i][j]);
        fprintf(o,"\n");
}
rstart=n-k+min_nu-2;
fprintf(o,"R_start=%d",rstart);
fprintf(o,"\n");
rend=n-r;
fprintf(o,"R_end=%d",rend);
fprintf(o,"\n");

if((rstart+rend)>k){
        rll=rstart+rend;
        fprintf(o,"RLL= %d", rll);
}
else{
        rll=k;
        fprintf(o,"RLL= %d", rll);
}
fprintf(o,"\n");

/*busca do vetor modificador*/
```

```
        nzd=0;
        nze=0;
        nzm=0;
        for(m=4;m<=8;++m){

        for(i11=0;i11<=1;++i11){
        for(i10=0;i10<=1;++i10){
        for(i9=0;i9<=1;++i9){
        for(i8=0;i8<=1;++i8){
        for(i7=0;i7<=1;++i7){
        for(i6=0;i6<=1;++i6){
        for(i5=0;i5<=1;++i5){
        for(i4=0;i4<=1;++i4){
        for(i3=0;i3<=1;++i3){
        for(i2=0;i2<=1;++i2){
        for(i1=0;i1<=1;++i1){
                for(i=1;i<=n;++i)
                        x[i]=(i1*p[1][i]+i2*p[2][i]+i3*p[3][i]+i4*p[4][i]+i5*p[5
                        ][i]+i6*p[6][i]+i7*p[7][i]+i8*p[8][i]+i9*p[9][i]+i10*
                      p[10][i]+i11*p[11][i])%2;
                x[m]=(x[m]+1)%2;
                w=0;
                for(i=n;i>=1;--i){
                        if(x[i]!=0)
                                goto a;
                        ++w;
                }
a:              if(w>nzd)
                        nzd=w;
                te=0;
                for(i=1;i<=n;++i){
                        if(x[i]!=0)
                                goto y;
                        ++te;
                }
y:              if(te>nze)
                        nze=te;
                e=0;
                for(i=1;i<=n;++i){
                        if(x[i]==0){
                                ++e;
                                if(e>nzm)
```

```
                                nzm=e;
                        }
                        else
                                e=0;
                }

        }
        }
        }
        }
        }
        }
        }
        }
        }
        }
        }

        if(nze==rstart&&nzd==rend&&nzm<=rll)
                goto f;

        nzd=0;
        nze=0;
        nzm=0;
        }
f:      fprintf(o,"\n posicao dos uns= %d", m);
        fprintf(o,"\n");
        fclose(o);
}
```

Referências

[Abdel, 1991] K. A. S. Abdel-Ghaffar and J. H. Weber, "Bounds and Constructions for Runlenth-Limited Error-Control Block Codes", *IEEE Transactions on Information Theory*, VOL. 37, nº. 3, May 1991.

[Abramson, 1963] N. Abramson, *Information Theory and Coding*, Mc Graw-Hill, 1963] .

[Ash, 1965] R. B. Ash, *Information Theory*, Wiley, 1965.

[Beenker, 1983] G. F. M. Beenker and K. A. S. Immink, "A Generalized Method for Encoding and Decoding Runlength-Limited Binary Sequences", *IEEE Transactions on Information Theory*, VOL. IT-29, September 1983.

[Benedetto, 1987] S. Benedetto, E. Biglieri and V. Castellani, *Digital Transmission Theory*, Prentice Hall, 1987.

[Berlekamp, 1968] E. R. Berlekamp, *Algebraic Coding Theory*, New York, Mc Graw-Hill,1968.

[Blaum, 1991] Mario Blaum, "Combining ECC with Modulation: Perfomance Comparisons", *IEEE Transactions on Information Theory*, VOL. 37, nº. 3, May 1991.

[Brooks, 1983] R. M. Brooks, "Linecoding for Optical Fibre Systems", *International Journal of Electronics*, VOL. 55, nº. 1, 1983.

[Bylanski, 1988] Ingram Pand Bylanski, *Digital Transmission Systems*, 2nd. ed., Peter Peregrinus 1988.

[Cariolaro, 1974] G. L. Cariolaro and G. P. Tronca, "Spectra of Block Coded Digital Signals", *IEEE Transactions on Communications*, VOL. COM-22, nº. 10, October 1974.

[Cariolaro, 1983] G. L. Cariolaro, G.L. Pierobon and G. P. Tronca, "Analysis of Codes and Spectra Calculations", *International Journal of Electronics*, VOL. 55, nº. 1, 1983.

[Cattermole, 1983] K. W. Cattermole, "Principles of Digital Line Coding", *International Journal of Electronics*, VOL. 55, nº. 1, 1983.

[Ferreira, 1984] H. Ferreira, "Lower Bounds on the Minimum Hamming Distance Achievable with Runlength Constrained or DC-Free Block Codes and the Synthesis of a (16,8) D_{min} = 4 DC-Free Block Code", *IEEE Transactions on Magnetism*, VOL. MAG-20, 1984.

[Forney, 1984] G. D. Forney, G. R. Lang, F. M. Longstaff and S. U. Qureshi, "Efficient Modulation for Band Limited Channels", *IEEE Journal on Selected Areas in Communications*, VOL. Sac-2, nº. 5, September 1984.

[Franklin, 1972] J. N. Franklin and J. R. Pierce, "Spectra and Efficiency of Binary Codes withouth DC", *IEEE Transactions on Communications*, VOL. COM-22, 1972.

[Furukawa, 1984] T. Furukawa, M. Ozaki and K. Tanaka, "On a DC Free Block Modulation Code", *IEEE Transaction on Magnetism*, VOL. MAG-20, nº. 5, September 1984.

[Gallager, 1968] R. G. Gallager, *Information Theory and Reliable Communication*, Wiley, 1968.

[Gallopoulos, 1989] A. Gallopoulos, C. Heegard and P. H. Siegel, "The Power Spectrum of Run-Length-Limited Codes", *IEEE Transactions on Communications*, VOL. COM-37, September 1989.

[Haykin, 1988] Simon Haykin, *Digital Communications*, Wiley, 1988.

[Haykin, 1994] Simon Haykin, *Communication Systems*, 3rd. ed., Wiley, 1994.

[Imai, 1977] H. Imai, "A New Multilevel Coding Method Using Error-Correcting Codes", *IEEE Transactions on Information Theory*, VOL. IT-23, nº. 3, 1977.

[Immink, 1990] K. A. S. Immink, "Runlength-Limited Sequences", *Proceedings of the IEEE*, VOL. 78, nº. 11, November 1990.

[Kernighan, 1978] Brian W. Kernighan and Dennis M. Ritchie, *The C Programming Language*, Bell Laboratories, 1978.

[Kokkos, 1992A. Kokkos, J. J. O'Reilly, A. Popplewell and S. Williams, "Evaluation of a Class of Error Control Line Codes: an Error Perfomance Perspective", *IEE Proceedings-I*, VOL. 139, nº. 2, April 1992.

[Lee, 1989] Patrick Lee and J. K. Wolf, "A General Error-Correcting Code Construction for Run-Length Limited Binary Channels", *IEEE Transactions on Information Theory*, VOL. 35, nº. 6, November 1989.

[Lin, 1983] Shu Lin and Daniel J. Costello Jr., *Error Control Coding: Fundamentals and Applications*, Prentice Hall, Inc. Englewood Cliffs, New Jersey, 1983.

[Mee, 1987] C. D. Mee and E. D. Daniel, *Magnetic Recording*, Mc Graww-Hill, 1987.

[Mac, Williams 1977] F. J. Mac Williams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*, North Holland, 1977.

[Oppenheim, 1989] A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing*, Englewood Cliffs, N. J.: Prentice-Hall, 1989.

[O'Reilly, 1990-1] J. J. O'Reilly and A. Popplewell, "Class of Disparity Reducing Transmission Codes with Embeded Error Protection", *IEE Proceedings*, VOL. 137, Pt. 1, nº. 2, April 1990.

[O'Reilly, 1990-2] J. J. O'Reilly and A. Popplewell, "A Further Note on DC-Free Coset Codes", *IEEE Transactions on Information Theory*, VOL. 36, nº. 3, May 1990.

[Peek, 1985] J. B. H. Peek, "Communications Aspects of the Compact Disk Digital Audio System", *IEEE Communication Magazine*, VOL. 23, February 1985.

[Peterson, 1972] W. Wesley Peterson and E. J. Weldon Jr., *Error Correcting Codes*, Second Edition, MIT Press, 1972.

[Popplewell, 1988] A. Popplewell and J. J. O'Reilly, "Spectral Characteristics of a Class of DC Free Error-Correcting Transmission Codes", *Electronics Letters*, VOL. 24, nº. 15, July 1988.

[Popplewell, 1990] A. Popplewell and J. J. O'Reilly, "Spectral Characterisation and Perfomance Evaluation for a New Class of Error Control Line Codes", *IEE Proceedings*, VOL. 137, Pt. 1, nº. 4, August 1990.

[Popplewell, 1992] A. Popplewell and J. J. O'Reilly, "Runlength Limited Binary Error Control Codes", *IEE Proceedings-1*, VOL. 139, nº. 3, June 1992.

[Popplewell, 1993] A. Popplewell and J. J. O'Reilly, "New Class of Runlength-Limited Error-Control Codes with Minimum Distance 4", *IEE Proceedings-I*, VOL. 140, nº. 2, April 1993.

[Popplewell, 1994-1] A. Popplewell and J. J. O'Reilly, "Algorithms Suitable for Low Complexity Implementation of a Class of Runlength-Limited Error Control Codes", *IEE Proc.-Commun.*, VOL. 141, nº. 3, June 1994.

[Popplewell, 1994-2] A. Popplewell and J. J. O'Reilly, "Soft-Decision Decoding Techniques for a Class of Runlength-Limited Error Control Codes", *IEE Proc.-Commun.*, VOL. 141, nº. 3, June 1994.

[Popplewell, 1995] A. Popplewell and J. J. O'Reilly, "A Simple Strategy for Constructing a Class of DC-Free Error-Correcting Codes with Minimum Distance 4", *IEEE Transactions on Information Theory*, VOL. 41, nº. 4, July 1995.

[Proakis, 1983] J. G. Proakis, *Digital Communications*, Mc Graw-Hill, 1983.

[Sayegh, 1986] S. I. Sayegh, "A Class of Optimun Block Codes in Signal Space", *IEEE Transactions on Communications*, VOL. Com-34, nº. 10, October 1986.

[Siegel, 1985] P. H. Siegel, "Recording Codes for Digital Magnetic Storage", *IEEE Transactions on Magnetism*, VOL. MAG-21, nº. 5, September 1985.

[Slepian, 1968] D. Slepian, "Group Codes for the Gaussian Channel", *The Bell System Technical Journal*, April 1968.

[Ungerboeck, 1982] G. Ungerboeck, "Channel Coding with Multilevel/Phase Signals", *IEEE Transactions on Information Theory*, VOL. IT-28, nº.1, January 1982.

[Wood, 1986] R. W. Wood, "Magnetic Recording Systems", *Proceedings of the IEEE*, VOL. 74, November 1986.

[Ytrehus, 1991] Øyvind Ytrehus, "Runlength-Limited Codes for Mixed-Error Channels", *IEEE Transactions on Information Theory*, VOL. 37, nº. 6, November 1991.

[Zehavi, 1988] Efhraim Zehavi and J. K. Wolf, "On Runlength Codes", *IEEE Transactions on Information Theory*, VOL. 34, nº. 1, January 1988.