UNIVERSIDADE ESTADUAL DE CAMPINAS

FACULDADE DE ENGENHARIA ELÉTRICA

DECOM

Introdução ao Sistema de Gravação de Dados

por Eng. Virgínia Pereira Fernandes

orientador Prof. Dr. Celso de Almeida

Dissertação submetida à Faculdade de Engenharia Elétrica da Universidade Estadual de Campinas, para preenchimento dos pré-requisitos parciais para obtenção do Título de Mestre em Engenharia Elétrica.

Este exemplar corresponde à redação final da tese defendida por Virgínia P. Fernandes e aprovada pela Comissão Julgadora em 31 / 03 / 93

Orientador

junho 1993

"Sempre antes de realizar um sonho, a Alma do Mundo resolve testar tudo aquilo que foi aprendido durante a caminhada. Ela faz isto não porque seja má, mas para que possamos, junto com o nosso sonho, conquistar também as lições que aprendemos seguindo em direção a ele.

É o momento em que a maior parte das pessoas desiste. É o que chamamos, em linguagem do deserto, de morrer de sede quando as tamareiras já apareceram no horizonte".

Paulo Coelho

Com amor e afeição
aos meus pais Virgínio e Divanilda,
e meus irmãos Valeska e Júnior.

Agradecimentos

Agradeço em especial a Deus por ter me dado força para que eu conquistasse mais esta etapa em minha vida.

À minha família que mesmo distante, sempre esteve próxima.

A Débora, Marília e Silvana por esses dois anos e meio de convivência.

Aos amigos, Carla, Darli, Fran, Guilhermo, Léo, Mangili, Sandra e Zake pela energia positiva durante esse trabalho.

Aos amigos, Aleandro, Caixeta, Dalila, Francisco, Jean, Jônio e Ruben por terem me auxiliado nas demais dificuldades da tese.

Ao Prof. Dr. Celso de Almeida pela oportunidade, amizade e orientação.

Agradeço também, por terem aceito o convite para participar da Banca Examinadora os Profs. Drs. Hélio Waldman, Mauro A. O. da Costa e Silva e Jaime Portugheis.

Resumo

O trabalho apresenta uma introdução a sistemas de gravação de dados.

Estudamos com ênfase a importância do emprego de códigos de linha em sistemas de gravação de dados. Tais códigos são responsáveis pela diminuição da interferência inter-simbólica, auxiliam na recuperação do sincronismo, além de adequar o espectro do sinal às exigências do canal, e ainda proporcionam aumento da densidade de armazenamento de dados.

Finalmente, tratamos de sistemas na presença de ruído. Deste modo, mencionamos a necessidade do uso dos códigos corretores de erro, a fim de corrigir os erros provocados pelos ruídos do canal, ou de regiões defeituosas no prório meio de gravação. Assim, podemos obter um armazenamento confiável de dados.

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

O trabalho apresenta uma abordagem inicial ao sistema de escrita e leitura de dados.

Consideramos em quase todo o texto, o sistema na ausência de ruído. Somente no capítulo 5 tratamos da problemática do ruído na decodificação dos dados.

Primeiramente, apresentamos o sistema com seus principais componentes, descrevemos os processos de escrita, leitura de dados e detecção de pico.

Sabemos que o principal objetivo em um sistema de gravação de dados é atingir uma alta densidade no armazenamento confiável de dados, e uma rápida velocidade de leitura e busca. Mas, como obter uma alta densidade de armazenamento? Veremos que através do uso dos códigos de linha, ou mais especificamente com uma classe especial destes, os códigos RLL ou códigos (d, k), conseguiremos um aumento da densidade de armazenamento de dados. Além disso, estes códigos auxiliam na recuperação do sincronismo e diminuem a interferência inter-simbólica.

No capítulo 2 trataremos especialmente dos códigos RLL, e veremos que estes são definidos em termos de dois parâmetros d e k, os quais são responsáveis em restringir a seqüência do canal. Analisaremos a eficiência de alguns códigos RLL, ou seja, o quanto estes códigos podem atingir da capacidade de canal. Para isto, descreveremos dois métodos de obtenção do valor da capacidade de canal sem ruído. O primeiro, baseado na representação em treliça dos códigos (d, k). E o segundo, através de sua representação na matriz

de transição de estado finito. Construímos várias tabelas com alguns códigos (d, k), que apresentam valores de sua capacidade de canal, ganho de densidade de armazenamento e eficiência.

A seguir, trataremos das principais técnicas de codificação e decodificação de códigos *RLL*. Como citamos anteriormente, ainda na ausência de ruído.

Construiremos alguns códigos *RLL*, e faremos comparações em termos de eficiência e grau de implementação destes em relação à técnica utilizada. Pois, devemos sempre analisar a viabilidade do uso de uma determinada técnica, procurando um equilíbrio entre uma fácil implementação e uma boa eficiência.

Finalmente, trataremos do sistema na presença de ruído. Neste caso, não poderíamos deixar de mencionar o emprego dos códigos corretores de erro. Pois, para obtermos um armazenamento confiável devemos corrigir os eventuais erros provocados pelo sistema.

Mostraremos o uso do código corretor de erro e do código *RLL* combinados, isto é, a criação de um código *RLL* que além de satisfazer as restrições do canal, possue capacidade de correção de erro.

E a fim de solucionar o problema da redução da eficiência do código corretor de erro, quando este é usado em cascata com o código *RLL*, apresentaremos um novo código *RLL*: o código *RLL* preservando distância. Este código preserva as propriedades de distância livre de Hamming do código corretor de erro. Também mostraremos, que a decodificação de Viterbi pode ser utilizada em códigos *RLL*.

Concluindo, faremos comentários sobre o assunto apresentado, e citaremos alguns temas de interesse para pesquisa posterior.

O programa utilizado para o auxílio da construção de códigos *RLL* de comprimento variável está no apêndice. As demais classes de códigos foram obtidas, em parte usando programas computacionais, como foi o caso do MATLAB, ou sem qualquer auxílio computacional.

Capítulo 2

Sistema de Gravação e Leitura de Dados

2.1 Introdução

Os sistemas de armazenamento de dados possuem aspectos semelhantes com relação aos sistemas de comunicações. Em ambos os sistemas estamos interessados em obter um meio eficiente e confiável de transportar a informação. Enquanto os sistemas de comunicações transmitem a informação de um ponto para outro, os sistemas de armazenamento de dados, transportam a informação de um dado momento para outro. A diferença mais substancial consiste no fato, de que os bits no sistema de armazenamento estão associados a um intervalo espacial, enquanto em sistemas de comunicações a um intervalo temporal.

Enquanto em sistemas de comunicações estamos procurando maximizar a taxa em $bits/s$, pela qual a informação digital possa ser transmitida e recebida confiavelmente, em sistemas de gravação de dados procuramos maximizar a densidade de área em $bits/mm^2$ de modo a armazenar e recuperar confiavelmente a informação.

A maior parte do aumento da densidade de armazenamento obtido até então, resulta de melhorias na parte do sistema que nós chamamos de canal, incluindo o próprio meio de gravação, e as cabeças de leitura e escrita.

Uma importante diferença entre sistemas de comunicação e armazenamento de dados é o valor da taxa de erro decodificada. Em sistemas de comunicações, a meta é usar uma taxa de erro de 10^{-5} a 10^{-9}. Para aplicações em informática, sistemas de armazena-

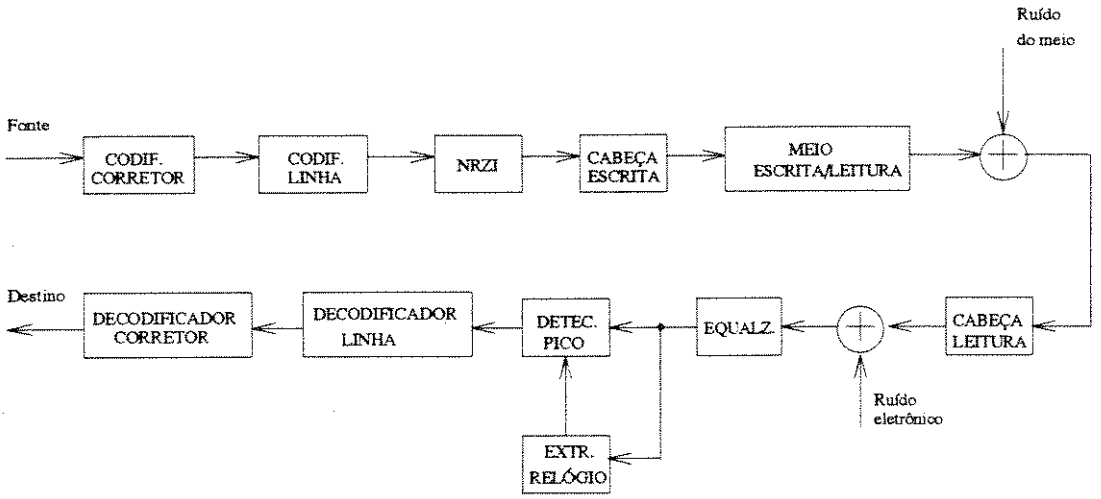

Figura 2.1: Sistema de gravação de dados

mento, entretanto, freqüentemente necessitam taxas de erro de 10^{-12} a 10^{-15} [1].

2.2 Processo de Gravação e Leitura de Dados

2.2.1 Introdução

Um sistema de gravação e leitura de dados pode ser modelado conforme a figura 2.1.

Os dados são primeiro passados por um codificador de correção de erro, tal como o código Reed-Solomon, devido aos eventuais erros aleatórios, ou em surto provocados pelo canal de gravação. A saída do codificador de correção de erro é então codificada usando um código de linha. A finalidade do codificador de linha é converter a seqüência de informação em uma nova seqüência que satisfaça às limitações do canal. Após uma pré-codificação, que tem por objetivo facilitar a detecção de pico, a seqüência de dados está pronta para ser armazenada. No processo de leitura, o sinal de tensão na saída da cabeça é equalizado, de modo a diminuir a interferência inter-simbólica (IIS), e então passa pelo detector de pico, onde os intervalos de bit contendo os picos de tensão são convertidos em $1s$ e os intervalos de bit sem picos são convertidos em $0s$. A seqüência binária correspondente do detector de pico é passada pelo decodificador de linha e então passada pelo decodificador de correção

de erro.

Descreveremos cada bloco a seguir.

Código Corretor de Erro

Como em um sistema de gravação de dados temos a presença de ruído, provocado seja pelos ruídos em geral do canal ou regiões defeituosas no meio de gravação, devemos procurar uma forma para minimizar tais erros, ou seja, tornar o sistema mais confiável. Então, devido às imperfeições do canal, é que utilizamos os códigos corretores de erro.

Códigos de Linha

Como foi dito anteriormente, os códigos de linha são utilizados para adequar a seqüência de dados às limitações do canal, como por exemplo: auxiliar a extração de sincronismo, combater a IIS, conformar o espectro da seqüência em relação às características do canal e prover ganho de densidade de armazenamento.

Muitos códigos de gravação para canais com detecção de pico em uso atualmente, pertencem à classe de códigos limitados por comprimento serial (*Run Length Limited - RLL*). Códigos RLL são caracterizados por dois parâmetros (d, k). que representam, o número mínimo e máximo de $0s$ entre $1s$ consecutivos na seqüência codificada, respectivamente.

Estes códigos são descritos em termos das restrições que produzem no sinal de escrita. O sinal da fonte não possue restrição alguma, desse modo o intervalo mínimo e máximo entre transições vale T_b e infinito, respectivamente, onde T_b é a largura de um bit. Quando um código (d, k) é utilizado, o intervalo de tempo mínimo entre as transições é T_b e o máximo intervalo de tempo entre transições adjacentes é kT_b.

A necessidade do parâmetro k ser finito segue do fato de que o circuito extrator de sincronismo, como veremos no processo de detecção de pico, precisa constantemente ser excitado com transições para que se possa recuperar um relógio que mantenha o sincronismo com os dados.

O parâmetro d permite uma compressão dos bits de canal, através do controle

da distância mínima entre transições. Desse modo, consegue-se diminuir a IIS, e portanto, aumentar a densidade de gravação de dados.

O problema da teoria de codificação de linha é construir um código de mapeamento simples e eficiente entre as seqüências geradas pela fonte e as seqüências do código de restrição (d, k). Um outro parâmetro importante do código é a sua taxa, que é o tamanho do bloco da fonte dividido pelo tamanho do bloco codificado.

O capítulo 3 está voltado basicamente aos aspectos de construção de códigos *RLL*.

Pré-codificação *NRZI*

Em geral, uma seqüência (dk) não é empregada em gravação de dados sem uma simples pré-codificação. O motivo é explicado logo a seguir.

Caso não se usasse a pré-codificação, em ocorrências de longas seqüências de $0s$, o aparecimento de um erro dentro desta seqüência, causaria o surgimento de um surto de erros. Portanto, a pré-codificação evita justamente este tipo de problema, ou seja, o aparecimento de um erro dá origem a apenas um erro após a detecção.

Além disso, a pré-codificação non-return-to-zero-inverse (*NRZI*) faz o seguinte tipo de conversão entre a seqüência de bits fornecidos pelo codificador de linha e a seqüência a ser armazenada:

- O bit 1 corresponde à uma transição no meio do período de bit, quer seja uma transição positiva ou negativa.
- O bit 0 corresponde à ausência de transição.

Por exemplo, a seqüência da figura 2.2 mostra a conversão da seqüência (d, k) 1010010100 em uma forma de onda adequada ao meio magnético.

Podemos verificar que a mínima e a máxima distâncias entre transições consecutivas da seqüência do canal originadas da seqüência (dk) valem $(d+1)$ e $(k+1)$ símbolos, respectivamente.

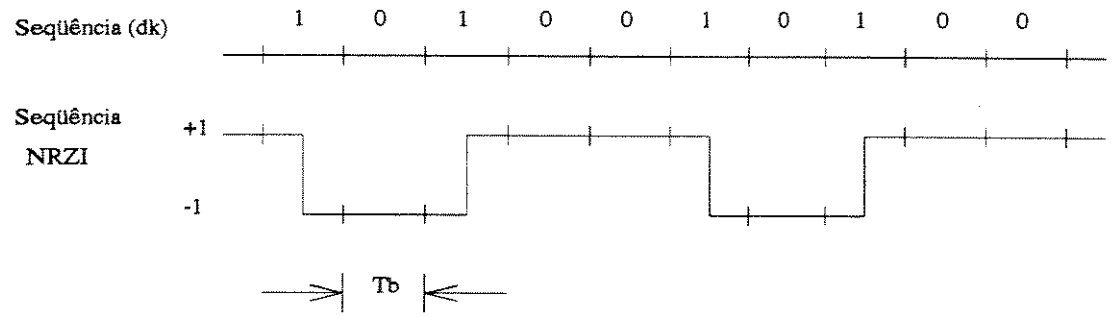

Figura 2.2: Pré-codificação *NRZI*

Existe uma grandeza denominada ganho de densidade de armazenamento de dados que expressa um aumento no número de bits gravados por área, e que depende do espaçamento mínimo entre transições. Esta grandeza é muito importante, e será utilizada para seleção de códigos eficientes no capítulo 3.

2.2.2 Processo de Escrita de Dados

O processo de escrita de dados é realizado através de uma corrente, que percorre uma bobina, que faz parte da cabeça de escrita, conforme ilustra a figura 2.3. O meio magnético ao passar pela cabeça de escrita sofre a indução de um fluxo magnético, cujo sentido de orientação está relacionado à polaridade da corrente de escrita na bobina.

Tipos de Armazenamento de Dados

Podemos dividir a forma de armazenamento de dados em duas classes: bits armazenados circularmente e bits armazenados linearmente.

Como um exemplo de armazenamento circular, consideremos como meio magnético um disco rígido. Este se move em um movimento circular sob a cabeça de escrita. A informação é armazenada no disco em trilhas concêntricas, sendo que a largura da trilha está diretamente relacionada com o tamanho da cabeça de escrita. A densidade de armazenamento é definida então, como sendo o produto do número de trilhas/mm e a densidade linear de informação ao longo da trilha em bits/mm.

Na forma de armazenamento linear, temos os sistemas de fita magnética, que ainda se dividem em dois tipos de processos de gravação [1]. No primeiro tipo, a cabeça permanece estacionária, enquanto a fita é arrastada sobre a mesma. No segundo tipo, chamado de sistema de cabeça rotatória, a cabeça é colocada em um tambor, que gira em relação a fita, que se move linearmente. O objetivo é aumentar a velocidade da fita em relação à cabeça, e portanto, aumentar a taxa de bits a serem armazenados. Este tipo de sistema é usado em gravação de videocassete, digital audio tape (DAT) [2], e em outras aplicações, onde é necessária uma maior largura de banda. A velocidade da cabeça em relação à fita no primeiro tipo, está associada a velocidade de movimentação da fita. Enquanto no segundo tipo, a velocidade da cabeça em relação à fita é uma função da velocidade de rotação do tambor e também da velocidade com que a fita passa pelo mesmo.

Em sistemas de fita com múltiplas cabeças de escrita, a informação geralmente é armazenada simultaneamente em várias trilhas, enquanto em sistemas de armazenamento em disco rígido, apenas uma só cabeça escreve a informação em uma única trilha.

Sistemas de disco rígido geralmente tem uma única cabeça para ambos os processos de escrita e leitura. Em produtos recentes a cabeça de escrita está fisicamente acoplada à cabeça de leitura, enquanto sistemas de fita podem ter separadas, a cabeça de leitura da de escrita, tal que o sistema possa ser lido enquanto está sendo gravado, de modo a se tornar mais confiável.

2.2.3 Processo de Leitura de Dados

Os domínios magnéticos gerados pela corrente de escrita no meio, induzem um fluxo magnético na cabeça de leitura. Deste modo, a cabeça de leitura ao passar pelo meio magnético produzirá uma tensão proporcional à variação temporal deste fluxo magnético, ou seja, uma tensão que está relacionada à variação temporal da corrente de escrita da trilha que está sendo lida. Os processos de escrita e leitura de dados estão ilustrados na figura 2.3.

Tipos de Cabeças de Leitura

Existem dois tipos de cabeças de leitura [1]:

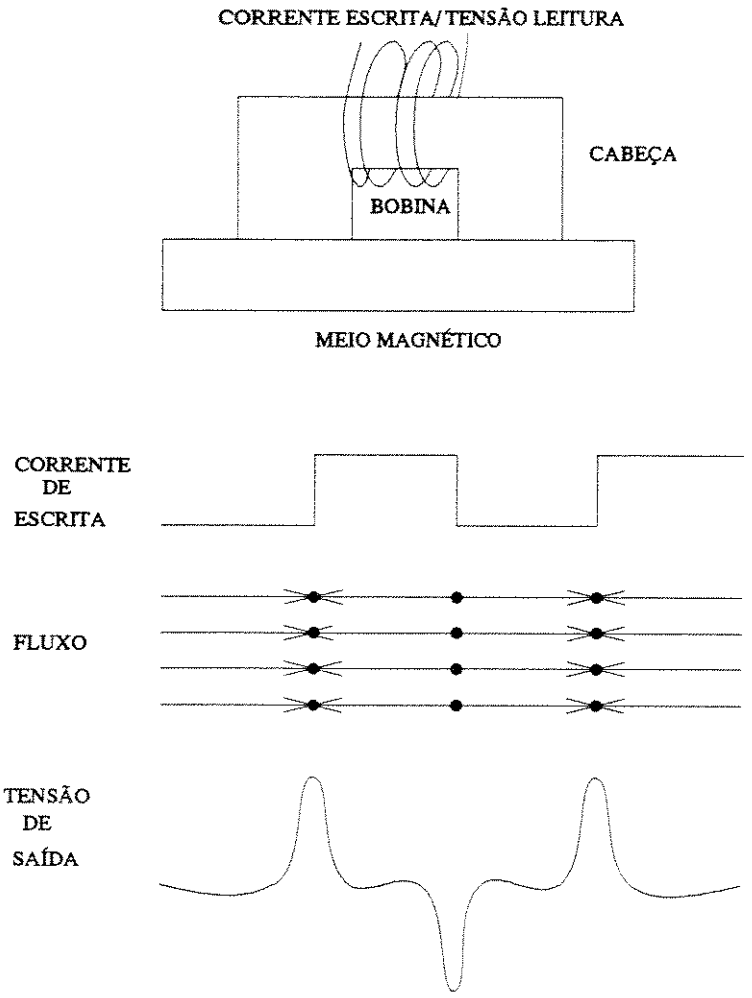

Figura 2.3: Processo de escrita e leitura de dados

- cabeça indutiva, que contém uma bobina e que produz uma tensão proporcional à derivada do fluxo magnético que passa através desta bobina;
- cabeça magneto-resistiva, que produz uma tensão diretamente proporcional ao fluxo sentido pela cabeça. Este tipo de cabeça produz maiores tensões de leitura do que a cabeça indutiva, mas tem uma faixa dinâmica limitada para operação linear.

Apenas as cabeças indutivas tem sido utilizadas para a gravação de dados.

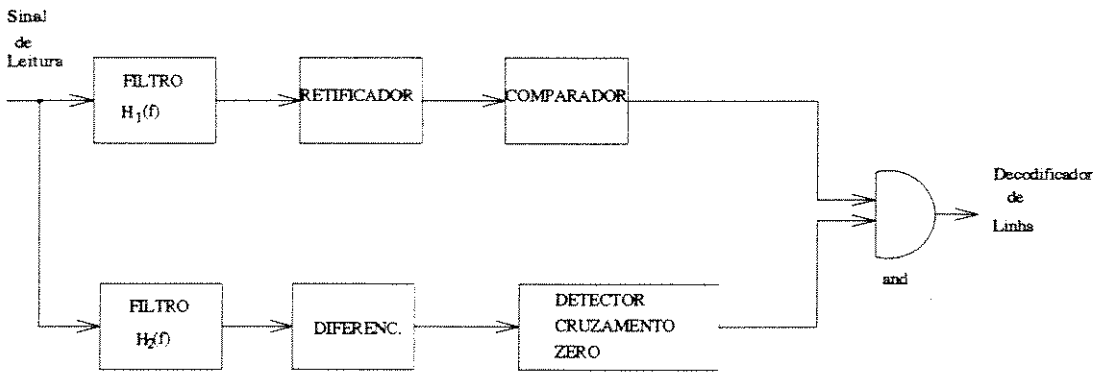

Figura 2.4: Detector de pico

Equalização

A equalização é feita através do uso de um filtro linear na saída da cabeça de leitura. o qual muda a resposta impulsiva global do sistema antes do detector de pico, deixando o pulso mais estreito e diminuindo a IIS. Deste modo, este filtro adequa o sinal de leitura facilitando a detecção de dados.

Processo de Detecção de Pico

A Detecção de Pico é um processo que tem por finalidade detectar as transições da forma de onda do sinal de leitura.

O detector de pico tem a vantagem de ser robusto e extremamente simples de implementar. Entretanto, sua melhor faixa de trabalho é para baixas densidades lineares. Em altas densidades é mais conveniente se usar as técnicas de resposta parcial que não serão estudadas neste trabalho. Um diagrama de bloco de um típico detector de pico é mostrado na figura 2.4 [1].

Existem dois caminhos através do detector. O caminho do topo é usado para verificar se o pico tem amplitude acima de um limiar. Este caminho consiste de um filtro linear $H_1(f)$, um retificador de onda completa e um comparador, que faz o teste de limiar. O outro caminho é usado para localizar a ocorrência do pico no tempo, isto é, pela diferenciação do sinal após o filtro linear $H_2(f)$, e então passando o sinal diferenciado através do detector

de cruzamento zero. O detector de pico apenas aceita um pico se a amplitude deste for grande o suficiente para passar pelo teste de habilitação do sinal, e se a derivada for próxima de zero.

O processo de detecção de pico é ilustrado na figura 2.5 e descrito a seguir.

O sinal na saída do codificador de linha é representado pelos bits 0 e 1. Após este sinal passar pelo pré-codificador *NRZI*, temos a corrente de escrita. Neste sinal, o bit 1 é representado por uma transição e o bit 0 pela ausência de transição.

A tensão de leitura por sua vez é a variação temporal da corrente de escrita. Assim, os picos da tensão de leitura representam as posições das transições da corrente de escrita, isto é, a derivada máxima.

Um circuito extrator de sincronismo é usado para identificar o momento das posições dos picos detectados. O extrator produz um relógio de período T_b segundos, pelo qual identifica os intervalos de bit do canal.

Na seqüência regenerada teremos um bit 1 se no instante da borda de subida do relógio tivermos o módulo da amplitude de pico maior que um certo limiar. Se o módulo da amplitude estiver abaixo do limiar, detectaremos um bit 0. Observe que a seqüência regenerada é a própria seqüência fornecida pelo codificador de linha, ou seja, a detecção de pico já realiza a decodificação *NRZI*.

Os circuitos de extração mais comumente usados são os circuitos tanque (sintonizado), ou os circuitos phase locked loop (*PLL*).

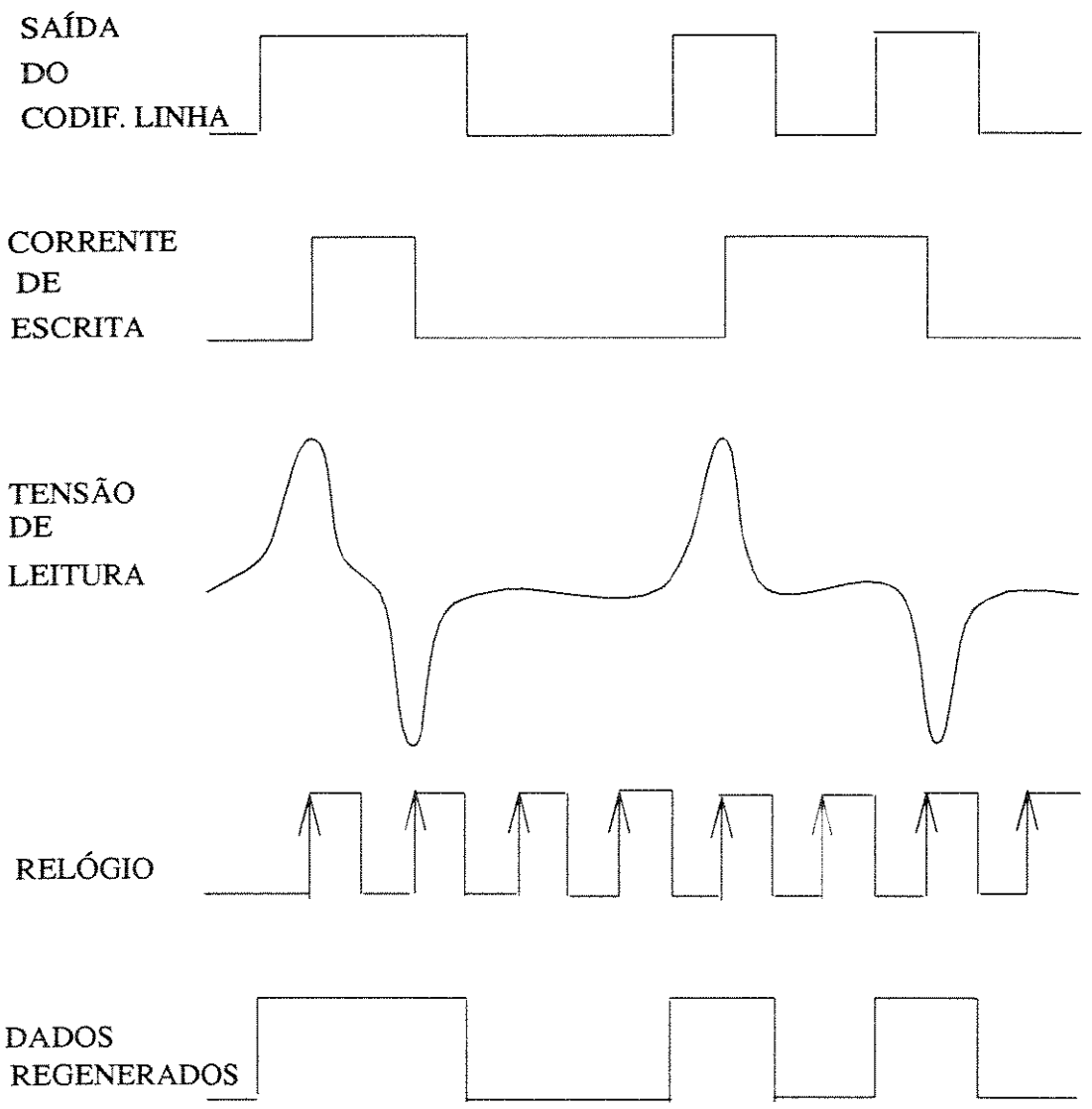

Figura 2.5: Processo de detecção de pico

2.3 Dificuldades Sistêmicas

No processo de leitura, a tensão induzida na cabeça é proporcional à variação temporal do fluxo magnético registrado no meio. Devida a esta diferenciação implícita, o canal não permite a passagem do nível D.C., de tal modo que o sinal a ser gravado deva possuir baixo conteúdo espectral próximo a 0Hz.

O canal de gravação magnética é inerentemente não linear devido aos efeitos da histerese no meio magnético. Tais efeitos dificultam a existência de sinais multiníveis. Entretanto, existe um modo de linearização do canal para uma dada faixa restrita de entradas. Isto é feito, restringindo a corrente que passa pela cabeça de escrita a apenas dois possíveis valores, por exemplo, $+A$ e $-A$, onde a amplitude A é suficientemente grande para saturar o meio magnético em duas direções. Dessa forma, o efeito da histerese pode ser ignorado. Este tipo de gravação é chamada de gravação de saturação, e todos dispositivos de gravação digital práticos utilizam esta técnica.

Após a informação ter sido gravada na trilha, o meio magnético encontra-se alternativamente magnetizado ao longo da mesma. Uma forma de onda da corrente de escrita em dois níveis e sua correspondente magnetização padrão na trilha são mostradas na figura 2.3. Para aumentar a densidade linear de armazenamento, o espaçamento entre as transições deve ser decrescido. Portanto, enquanto para altas densidades de armazenamento, a IIS torna-se intensa e inevitável, em sistemas de comunicações devido ao uso de configurações multiníveis pode-se aumentar a taxa de dados transmitidos, pois devido a ausência do efeito da histerese, a IIS é mantida em níveis moderados com uma simples equalização.

O canal magnético produz ruído de diversas formas, que irão deteriorar a qualidade de leitura do mesmo. Os principais tipos de ruído são: ruído do material magnético, ruído de sobre-escrita, etc. Além disso, existe o ruído eletrônico, que é devido aos estágios iniciais do processo de leitura.

O meio magnético pode apresentar regiões defeituosas, ou material estranho em sua superfície. Em ambas situações, o processo de leitura não conseguirá resgatar os bits gravados. Neste caso, os códigos corretores de erro Reed-Solomon entrelaçados são os mais usados, pois possuem excelentes características para a correção de erros isolados ou em

surtos.

Um outro problema é o da variação temporal do ganho do canal que afeta o sinal de leitura [1]. Isto pode ser devido a muitos fatores, incluindo a variação da distância entre a cabeça de leitura e o meio. Às vezes, isto também ocorre, pela presença de defeitos físicos no próprio meio. Em sistemas de disco, é feito um mapeamento de cada trecho da superfície do mesmo, a fim de identificar possíveis regiões com defeito, que não serão utilizadas.

Capítulo 3

Códigos de Linha

3.1 Introdução

O problema da codificação de dados pode ser dividido em duas principais categorias: codificação de fonte e codificação de canal.

A codificação de fonte é uma técnica utilizada com o objetivo de remover a redundância do sinal. Já a codificação de canal visa proporcionar uma transmissão confiável pelo mesmo, colocando redundância no sinal. Em essência, a teoria de informação mostra que um canal pode ser confiável dado que uma fração fixa de sua taxa de símbolos seja usada para redundância.

Em sistemas de gravação de dados, a codificação de canal é realizada em duas etapas sucessivas: código corretor de erro e código de linha, também denominado de código de gravação. Um diagrama de blocos de um sistema de gravação deste tipo é mostrado na figura 3.1.

Primeiramente, os dados gerados pela fonte na forma de símbolos binários são

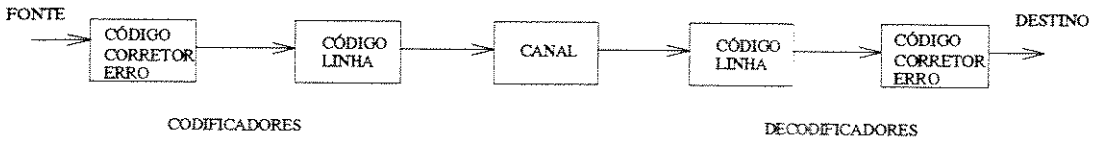

Figura 3.1: Diagrama de blocos de um sistema de gravação de dados

passados por um código corretor de erro seguido de um código de gravação apropriado. A saída gerada pelo código de gravação é armazenada no meio na forma de magnetizações positivas e negativas. No processo inverso, ou seja, durante a leitura, as magnetizações geram um sinal que após passar pelo decodificador de gravação e decodificador de correção de erro, fornecem uma seqüência de bits que é desejável ser a mais próxima possível da seqüência originada pela fonte.

O controle de correção de erro é realizado pela adição de símbolos extras à seqüência da fonte. Estes símbolos extras tornam possível ao receptor detectar e/ou corrigir alguns dos erros que podem ocorrer na seqüência lida. O principal problema é conseguir a proteção necessária contra os inevitáveis erros da transmissão sem pagar um alto preço na adição de símbolos extras. Os códigos corretores de erro mais importantes para aplicações em gravação de dados são aqueles que pertencem à família de códigos Reed-Solomon, e em geral adicionados a um circuito entrelaçador (interleaving). A razão de seu emprego em sistemas de gravação é que estes podem combater tanto erros isolados quanto em surtos.

O código de gravação aceita uma seqüência de bits vinda do código corretor de erro como entrada e converte esta seqüência em uma outra seqüência adequada às exigências do canal. O objetivo destes códigos é produzir características desejáveis na seqüência de dados que geralmente não estão presentes na seqüência da fonte de informação utilizada, como por exemplo, evitar longas seqüências de $0s$ e de $1s$.

As restrições físicas ditadas pelo canal conduzem ao fato de que nem todas as seqüências de entrada podem ser utilizadas. Por exemplo, as cabeças magnéticas não respondem a sinais de baixa freqüência, deste modo, a fim de minimizar a distorção dos dados recuperados, elimina-se as componentes em baixa freqüência nos dados armazenados. Além disso, a probabilidade de erro é reduzida evitando-se estas seqüências que são conhecidas a priori por serem mais vulneráveis às imperfeições do canal. O processo atual de seleção e projeto dos códigos de gravação está fortemente vinculado ao compromisso de satisfazer as exigências do canal. A escolha de um particular código depende de numerosos fatores a serem avaliados: a relação sinal/ruído, extração do sincronismo, não linearidades e a largura de banda, que gera o problema da interferência inter-simbólica. Um código de gravação deve mostrar ainda uma certa robustez contra as mudanças dinâmicas das características do canal.

Um codificador de gravação tem a tarefa de substituir blocos da seqüência de informação vinda do codificador de correção de erro, em uma nova seqüência de blocos que obedeça às restrições do canal. Deste modo, m bits de informação são substituídos por n bits de canal, desde que $n > m$. A taxa do código é dada pelo quociente $R = \frac{m}{n}$. No lado do receptor, o decodificador processa o mapeamento inverso originando a seqüência digital da fonte.

3.2 Seqüências Limitadas por Comprimento Serial (RLL)

Códigos de gravação de dados baseados em seqüências limitadas por comprimento serial (RLL) estão sendo eficazmente aplicados em sistemas de gravação magnética e óptica. Os códigos RLL ou códigos (d, k) são similares aos códigos de linha criados em transmissão de dados binários. Entretanto, estes códigos são usados não só com o propósito de sincronização e conformação espectral. Além disso, conforme veremos mais tarde, proporcionam também ganho de densidade de armazenamento e, através do aumento da distância entre transições, aliviam a interferência inter-simbólica [3].

Definição 1 *Uma seqüência (dk) satisfaz simultaneamente às duas condições [4]:*

- *Tem no mínimo d zeros entre dois uns consecutivos.*
- *Tem no máximo k zeros entre dois uns consecutivos.*

Exemplo 1 *Seja a seqüência (dk): 0010001001000010001. De acordo com a definição acima, vemos que esta tem no mínimo dois e no máximo quatro zeros entre dois uns consecutivos. Logo, esta seqüência possui d=2 e k=4, ou seja, pertence a um código (2,4).*

O parâmetro d permite uma compressão dos bits de canal, através do controle da distância mínima entre transições. Se esta distância for grande, facilita a diminuição da interferência inter-simbólica. Por outro lado, para se aumentar a densidade de gravação de dados, a distância entre transições deve ser a menor possível. Com isso, fica ao nosso critério procurar um ponto de equilíbrio entre a diminuição da interferência inter-simbólica e o aumento do ganho de densidade, ou sacrificar um dos dois. O parâmetro k não permite

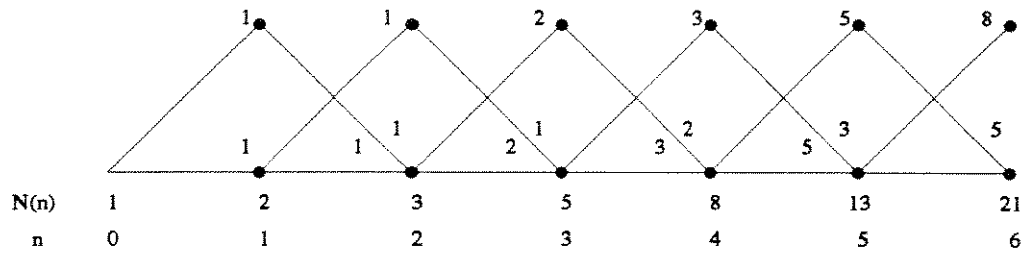

Figura 3.2: Representação em treliça do código (1,∞)

a ocorrência de seqüências muito longas sem transição, facilitando dessa forma a extração do relógio. Os limites dos valores de d e k são escolhidos dependendo de vários fatores, tais como a resposta do canal, a desejada densidade de informação e as características de jitter e ruído [5].

Os três códigos *RLL* mais populares são o código (1,3) ou código de Miller, e os códigos (1,7) e (2,7) que possuem taxas 1/2, 2/3 e 1/2, respectivamente [3].

3.3 Número de Seqüências (dk)

O número de palavras-código de um código (d,k) cresce com o aumento do tamanho do bloco. Um modo de se obter o número de seqüências para uma dada restrição (d,k) é através da representação em treliça do número de palavras-código em função do comprimento do bloco, conforme ilustra o exemplo seguinte.

Exemplo 2 *Seja o código $(d,k) = (1,\infty)$. De acordo com os valores de d e k, sabemos que para $d = 1$, teremos no mínimo um 0 entre dois 1s consecutivos, e para $k = \infty$, poderemos ter uma seqüência infinita de 0s. Portanto, a representação em treliça do código $(1,\infty)$ é a da figura 3.2.*

É fácil constatar que neste caso temos somente 2 estados, dependendo se o último bit recebido for 0 ou 1. A regra adotada na construção da treliça neste trabalho, utiliza uma transição ascendente na treliça quando recebemos o bit 1 e uma transição descendente quando recebemos o bit 0.

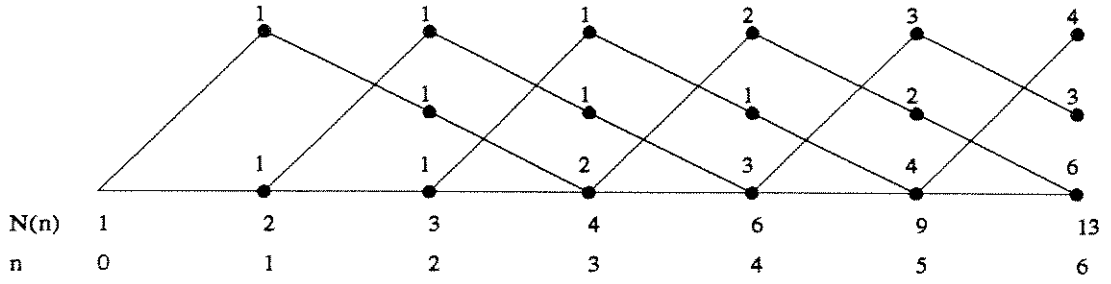

Figura 3.3: Representação em treliça do código (2,∞)

No caso em questão do código (1,∞), não poderemos ter 1*s consecutivos, deste modo, toda vez que recebermos um 1 logo em seguida teremos que receber pelo menos um 0, e se tivermos um 0 poderemos receber tanto um 1 quanto um 0. O número associado aos ramos da treliça indicam todas as seqüências possíveis até aquele nó. Os números abaixo da treliça representam o número total de seqüências para um dado comprimento de bloco. Por exemplo, sendo* $N(n)$ *o número de seqüências* (d,k) *distintas de comprimento* n*, temos que para* $n = 3$*, cinco palavras-código obedecem a restrição* $(1,\infty)$*. São elas: 000, 010, 001, 100 e 101.*

Notamos a formação de uma série sob a treliça, onde cada número é a soma de seus dois anteriores. Estes números são chamados de números de Fibonacci. A série de Fibonacci é bem conhecida, sendo que a razão de dois números sucessivos de Fibonacci $\frac{N(n+1)}{N(n)}$*, quando* n *torna-se muito grande (* $n \to \infty$*), é dita razão áurea* $g = \frac{(1+\sqrt{5})}{2}$*.*

Se repetirmos o procedimento para $d > 1$*, obteremos treliças em que* $N(n)$ *obedece a uma série de Fibonacci modificada. Mostramos a representação em treliça dos códigos* $(2,\infty)$ *e* $(3,\infty)$ *nas figuras 3.3 e 3.4.*

Notamos que o número de estados na treliça é igual a $(d+1)$*, para as seqüências* (d,k) *com* $k = \infty$*, que chamamos também de seqüências* (d)*.*

Observando a formação das séries associadas às seqüências (d), notamos que estas são da forma [4]

$$N(n) = 0, \quad n < 0$$

$$N(0) = 1$$

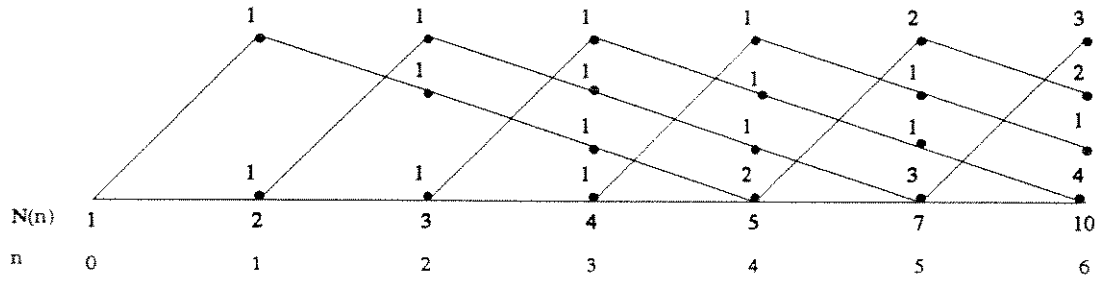

Figura 3.4: Representação em treliça do código (3,∞)

$$N(n) = n+1, \quad 1 \leq n \leq d+1$$

$$N(n) = N(n-1) + N(n-d-1), \quad n > d+1 \tag{3.1}$$

Quando $d = 0$ encontramos $N(n) = 2N(n-1)$, isto é, quando não existe restrição, o número de palavras-código dobra quando um bit é adicionado, resultado bastante conhecido.

A tabela 3.1 lista o número de seqüências (d) distintas em função do comprimento n, de acordo com o valor do parâmetro d.

d	n = 4	n = 5	n = 6	n = 7	n = 8	n = 9	n = 10	n = 11	n = 12	n = 13	n = 14
1	8	13	21	34	55	89	144	233	377	610	987
2	6	9	13	19	28	41	60	88	129	189	277
3	5	7	10	14	19	26	36	50	69	95	131
4	5	6	8	11	15	20	26	34	45	60	80
5	5	6	7	9	12	16	21	27	34	43	55

Tabela 3.1: Número de seqüências (d) distintas

Como vimos, as relações anteriores foram obtidas para a restrição k livre ($k = \infty$). Agora, vamos obter o número de seqüências de comprimento n para um valor específico do par de restrição (d, k).

Exemplo 3 *Seja a representação em treliça do código (1,2) na figura 3.5. De acordo com o par de restrição (1,2), devemos ter no mínimo um e no máximo dois 0s entre 1s consecutivos, respectivamente. Para um comprimento $n = 4$, apenas cinco palavras-código satisfazem as restrições. São elas: 0010, 0100, 0101, 1001 e 1010.*

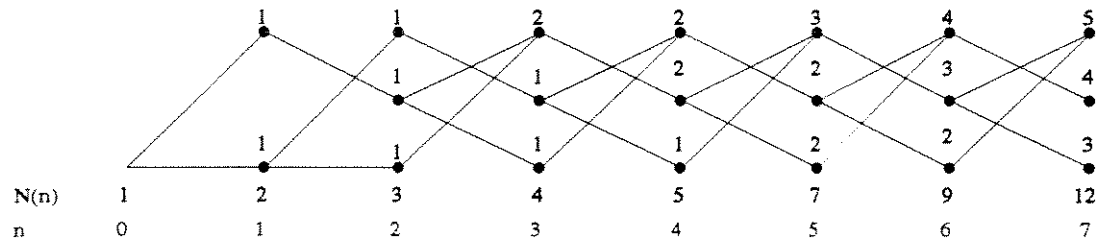

Figura 3.5: Representação em treliça do código (1,2)

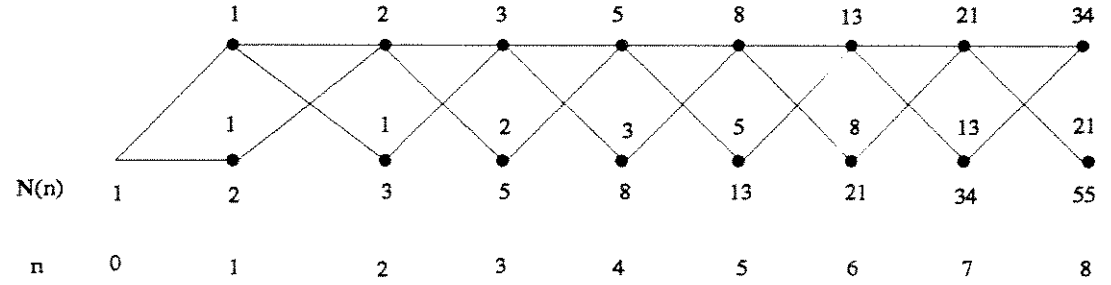

Figura 3.6: Representação em treliça do código (0,1)

As figuras 3.6, 3.7 e 3.8 mostram as treliças dos códigos (0,1), (2,4) e (3,4), respectivamente.

Notamos, que as treliças para as seqüências (d,k) possuem $(k+1)$ estados. De modo similar, é possível obter o número de seqüências (d,k) de comprimento n.

Através da observação das treliças anteriores, temos que o número de seqüências (d,k) de comprimento n é dado por [4]

$$N(n) = 0, \quad n < 0$$

$$N(0) = 1$$

$$N(n) = n+1, \quad 1 \leq n \leq d+1$$

$$N(n) = N(n-1) + N(n-d-1), \quad d+1 \leq n \leq k$$

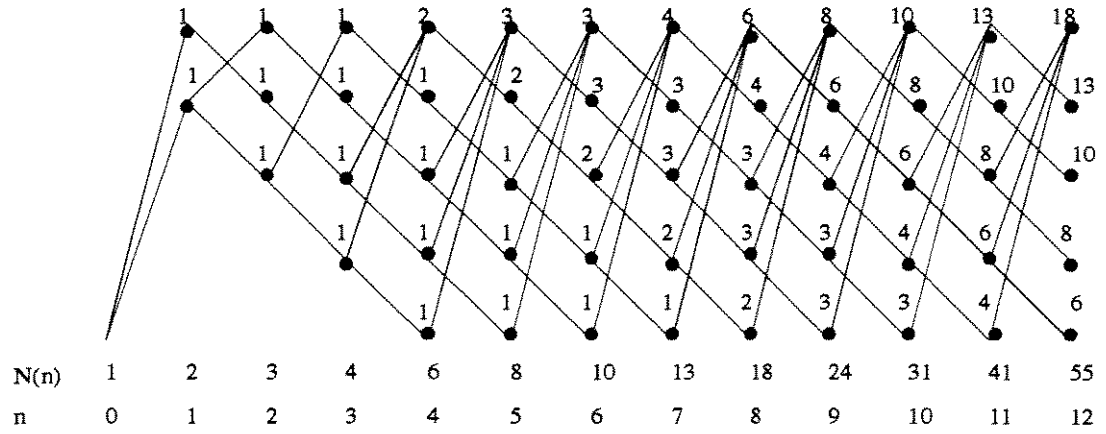

Figura 3.7: Representação em treliça do código (2,4)

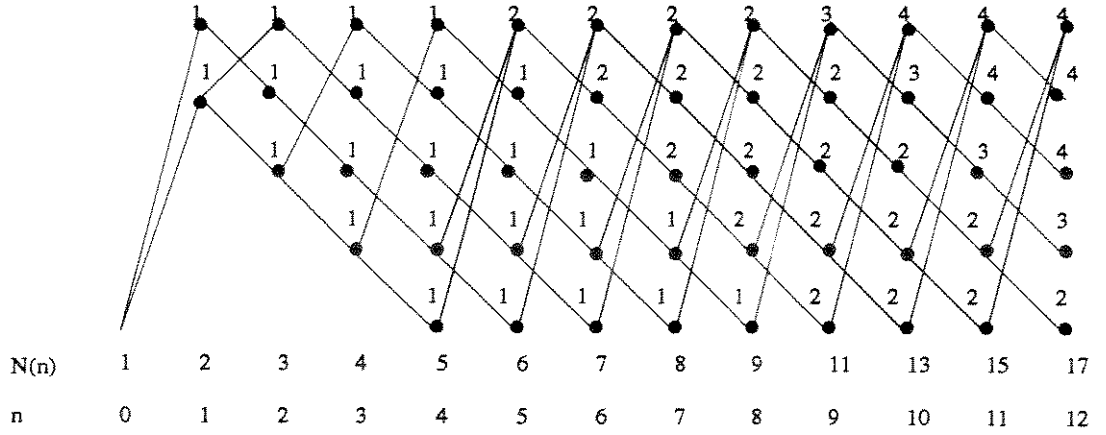

Figura 3.8: Representação em treliça do código (3,4)

$$N(n) = d + k + 1 - n + \sum_{i=d}^{k} N(n-i-1), \quad k < n \leq d + k$$

$$N(n) = \sum_{i=d}^{k} N(n-i-1), \quad n > d + k \tag{3.2}$$

O caso do par de restrição (d, k), com $d = 0$ e k tendo valor limitado, pode ser derivado como um caso especial da definição geral das seqüências (d, k).

As tabelas 3.2, 3.3 e 3.4 listam o número de seqüências (d, k) distintas de comprimento n, para algumas restrições (d, k).

k	n = 4	n = 5	n = 6	n = 7	n = 8	n = 9	n = 10	n = 11	n = 12
1	8	13	21	34	55	89	144	233	377
2	11	20	37	68	125	230	423	778	1431
3	12	23	44	85	164	316	609	1174	2263
4	12	24	47	92	181	356	700	1376	2705
5	12	24	48	95	188	373	740	1468	2912
6	12	24	48	96	191	380	757	1508	3004
7	12	24	48	96	192	383	764	1525	3044
8	12	24	48	96	192	384	767	1532	3061
9	12	24	48	96	192	384	768	1535	3068
10	12	24	48	96	192	384	768	1536	3071

Tabela 3.2: Número de seqüências $(0, k)$ distintas

k	n = 4	n = 5	n = 6	n = 7	n = 8	n = 9	n = 10	n = 11	n = 12
2	5	7	9	12	16	21	28	37	49
3	7	10	15	22	32	47	69	101	148
4	8	12	18	28	43	66	101	155	238
5	8	13	20	31	49	77	121	190	298
6	8	13	21	33	52	83	132	210	334
7	8	13	21	34	54	86	138	221	354
8	8	13	21	34	55	87	141	226	364
9	8	13	21	34	55	87	142	228	368
10	8	13	21	34	55	87	142	229	370

Tabela 3.3: Número de seqüências $(1, k)$ distintas

k	n = 4	n = 5	n = 6	n = 7	n = 8	n = 9	n = 10	n = 11	n = 12
3	5	6	7	9	11	13	16	20	24
4	6	8	10	13	18	24	31	41	55
5	6	9	12	16	22	31	43	59	81
6	6	9	13	18	25	35	50	71	100
7	6	9	13	19	27	37	54	78	111
8	6	9	13	19	28	38	56	82	117
9	6	9	13	19	28	38	57	84	120
10	6	9	13	19	28	38	57	85	122

Tabela 3.4: Número de seqüências $(2, k)$ distintas

3.4 Capacidade do Canal sem Ruído

Como foi dito anteriomente, a codificação de canal é efetuada mapeando-se blocos de m bits em blocos de n bits que satisfaçam às restrições d e k. Desta forma, busca-se sempre códigos com máxima taxa. Qual o valor máximo de $R = \frac{m}{n}$ que pode ser obtido, dadas as restrições d e k? A resposta foi fornecida por Shannon. O máximo valor de R que pode ser conseguido é denominado *Capacidade do canal sem ruído.* A capacidade, denotada por $C(d, k)$ é dada por [4]

$$C(d,k) = \lim_{n\to\infty} \frac{1}{n} \log_2 N(n) \tag{3.3}$$

Como podemos observar, a equação 3.3 necessita de maneira explícita do valor do número de seqüências $N(n)$ em função de seu comprimento n. Um modo de obtermos este valor desejado é através da resolução das equações recursivas referidas às seqüências (d) e (d, k).

De acordo com a equação 3.1, o número de seqüências (d) é

$$N(n) = N(n-1) + N(n-d-1), \qquad n > d+1$$

Como vimos a razão da série para $n \to \infty$ vale

$$\lim_{n\to\infty} \frac{N(n)}{N(n-1)} = \delta$$

e portanto, pode-se obter

$$\lim_{n\to\infty} \frac{N(n)}{N(n-i)} = \delta^i$$

Fazendo $i = n$

$$\lim_{n\to\infty} N(n) = \delta^n \tag{3.4}$$

Portanto, das equações 3.3 e 3.4, temos

$$C(d,\infty) = \lim_{n\to\infty} \frac{1}{n} \log_2 N(n) = \log_2 \delta \tag{3.5}$$

onde δ é a razão da série.

Dividindo a equação 3.1 pelo menor termo, ou seja, $N(n-d-1)$, temos

$$\frac{N(n)}{N(n-d-1)} = \frac{N(n-1)}{N(n-d-1)} + \frac{N(n-d-1)}{N(n-d-1)}$$

Escrevendo em função da razão, obtemos a equação característica

$$\delta^{d+1} - \delta^d - 1 = 0 \tag{3.6}$$

A maior solução de δ da equação 3.6 nos fornecerá a razão da série, que na equação 3.5 nos fornecerá a capacidade de canal.

O valor $C(d,\infty)$ fornece a máxima taxa possível atingida por um código dada as restrições do canal, e depende da razão da série do número de seqüências da treliça.

Exemplo 4 *Seja o código (1,∞), sua equação característica é da forma*

$$\delta^2 - \delta - 1 = 0$$

com soluções

$$\delta_1 = \frac{1}{2}(1+\sqrt{5}) \text{ e } \delta_2 = \frac{1}{2}(1-\sqrt{5})$$

Sendo $\delta = máx_i\{\delta_i\} = \frac{1}{2}(1+\sqrt{5})$, *que é igual à razão áurea obtida da série de Fibonacci. Portanto, o logaritmo na base 2 da razão da série de Fibonacci nos leva à capacidade de canal.*

$$C(1,\infty) = \log_2 \frac{1+\sqrt{5}}{2} = 0,694$$

Isto significa, que é possível construir códigos com taxa $R \leq 0,694$.

De forma similar ao caso das seqüências de restrição d é possível obter a capacidade $C(d,k)$ para k finito.

A equação 3.2 apresenta o número de seqüências (d,k)

$$N(n) = \sum_{i=d}^{k} N(n-i-1), \quad n > d+k$$

$$N(n) = N(n-d-1) + N(n-d-2) + \ldots + N(n-k-1)$$

Dividindo por $N(n-k-1)$

$$\frac{N(n)}{N(n-k-1)} = \frac{N(n-d-1)}{N(n-k-1)} + \frac{N(n-d-2)}{N(n-k-1)} + \ldots + \frac{N(n-k-1)}{N(n-k-1)}$$

Escrevendo a equação em termos da razão δ, temos a equação característica.

$$\delta^{k+1} = \delta^{k-d} + \delta^{k-d-1} + \ldots + 1 \tag{3.7}$$

O lado direito da equação 3.7 é uma progressão geométrica, cuja soma vale

$$\frac{\delta^{k-d+1} - 1}{\delta - 1}$$

Logo, a equação característica é da forma

$$\delta^{k+2} - \delta^{k+1} - \delta^{k-d+1} + 1 = 0 \tag{3.8}$$

A capacidade $C(d,k)$ é o logaritmo na base 2 da maior raiz real da equação característica 3.8.

3.5 Ganho de Densidade de Armazenamento de Dados

O valor do ganho de densidade de armazenamento expressa a mínima distância física entre transições consecutivas de uma seqüência RLL, para uma dada taxa de informação, em relação à mínima distância entre transições de seqüências sem restrição, e é dado por

$$\mathcal{G} = (d+1)R \tag{3.9}$$

onde R é a taxa do código RLL, de tal forma que ao codificarmos estaremos estreitando os bits em R vezes e como teremos sempre pelo menos d $0s$ entre $1s$, isto significa, que duas transições estarão separadas de pelo menos $(d+1)$ bits [6]. Portanto, os melhores códigos RLL são aqueles que maximizam o produto $(d+1)R$. As tabelas 3.5 e 3.6 listam a capacidade $\mathcal{C}(d,k)$ e o ganho máximo de densidade $\mathcal{G}_{\text{máx}}$ para alguns pares de restrição (d,k), respectivamente. E as tabelas 3.7 e 3.8 mostram a capacidade e o ganho máximo de densidade $(R=\mathcal{C})$, para códigos RLL com (d,∞), $(d,10)$ e $(2,k)$.

k	d = 0	d = 1	d = 2	d = 3	d = 4	d = 5	d = 6
2	0,8791	0,4057					
3	0,9468	0,5515	0,2878				
4	0,9752	0,6174	0,4057	0,2232			
5	0,9881	0,6509	0,4650	0,3218	0,1823		
6	0,9942	0,6690	0,4979	0,3746	0,2669	0.1542	
7	0,9971	0,6793	0,5174	0,4057	0,3142	0.2281	0,1335
8	0,9986	0,6853	0,5293	0,4251	0,3432	0.2709	0,1993
9	0,9993	0,6888	0,5369	0,4376	0,3620	0.2979	0,2382
10	0,9996	0,6909	0,5418	0,4460	0,3746	0.3158	0,2633
11	0,9998	0,6922	0,5450	0,4516	0,3833	0.3282	0,2804
12	0,9999	0,6930	0,5471	0,4555	0,3894	0.3369	0,2924
13	0,9999	0,6935	0,5485	0,4583	0,3937	0.3432	0,3011
14	0,9999	0,6938	0,5495	0,4602	0,3968	0.3478	0,3074
15	0,9999	0,6939	0,5501	0,4615	0,3991	0.3513	0,3122
∞	1,0000	0,6942	0,5515	0,4650	0,4057	0.3620	0,3282

Tabela 3.5: Capacidade $\mathcal{C}(d,k)$ para códigos (d,k)

k	d = 0	d = 1	d = 2	d = 3	d = 4	d = 5	d = 6
2	0,8791	0,8114					
3	0,9468	1,1030	0,8634				
4	0,9752	1,2348	1,2171	0,8928			
5	0,9881	1,3018	1,3950	1,2872	0,9115		
6	0,9942	1,3380	1,4937	1,4984	1,3345	0,9252	
7	0,9971	1,3586	1,5516	1,6228	1,5710	1,3686	0,9345
8	0,9986	1,3706	1,5879	1,7004	1,7160	1,6254	1,3951
9	0,9993	1,3776	1,6107	1,7504	1,8100	1,7874	1,6674
10	0,9996	1,3818	1,6254	1,7840	1,8730	1,8948	1,8431
11	0,9998	1,3844	1,6350	1,8064	1,9165	1,9692	1,9628
12	0,9999	1,3860	1,6413	1,8220	1,9470	2,0214	2,0468
13	0,9999	1,3870	1,6455	1,8332	1,9685	2,0592	2,1077
14	0,9999	1,3876	1,6485	1,8408	1,9840	2,0868	2,1518
15	0,9999	1,3878	1,6503	1,8460	1,9955	2,1078	2,1854

Tabela 3.6: Ganho máximo de densidade de armazenamento $\mathcal{G}_{\text{máx}}$ para códigos (d, k)

Observamos que o ganho de densidade atinge um valor máximo para d finito, dado k. E um valor máximo para k infinito, dado d.

Fixando k e aumentando o valor de d, estaremos inicialmente aumentando o ganho de armazenamento, pois o mesmo é proporcional a $d + 1$. Entretanto, à medida que d cresce, a capacidade do canal passa a diminuir, pois a restrição aumenta, de tal modo que o ganho de armazenamento atinge um valor ótimo para d finito.

Fixando d e aumentando o valor de k estaremos relaxando a restrição, ou seja,

d	C	$\mathcal{G}_{\text{máx}}$
1	0,6909	1,3818
2	0,5418	1,6254
3	0,4460	1,7840
4	0,3746	1,8730
5	0,3158	1,8948
6	0,2633	1,8431
7	0,2126	1,70[illegible]3

Tabela 3.7: Capacidade e ganho de densidade máximo para códigos RLL com $(d, 10)$.

k	C	$\mathcal{G}_{\text{máx}}$
3	0,2878	0,8634
4	0,4057	1,2171
5	0,4650	1,3950
6	0,4979	1,4937
7	0,5174	1,5522
8	0,5293	1,5879
9	0,5369	1,6107
10	0,5418	1,6254

Tabela 3.8: Capacidade e ganho de densidade máximo para códigos *RLL* com $(2, k)$.

um maior número de palavras-código satisfarão às condições impostas pelo canal, e como a taxa depende do número de palavras-código, esta aumentará e conseqüentemente o ganho.

3.6 Matriz de Transição de Estado

Uma técnica alternativa para se calcular a capacidade de canal, é baseada no diagrama de transição de estado finito (*FSTD - finite state transition diagram*), o qual representa as seqüências binárias que satisfazem ao par de restrição (d, k).

Apesar desta técnica ser vista como um outro modo de se obter a capacidade de canal, notamos que ambas as técnicas possuem o mesmo princípio de construção. Ou seja, representam os estados do canal e suas transições, quer seja por uma treliça ou diagrama de transição de estado.

O *FSTD* consiste basicamente de um grafo com nós, chamados de estados, e de ramos direcionados entre eles que representam as transições, representados por setas. Os ramos são rotulados por bits de canal e os caminhos através do grafo obedecem às restrições impostas na construção dos códigos *RLL*.

A figura 3.9 ilustra um diagrama de transição de estado geral. Para uma seqüência (d, k), existem $(k + 1)$ estados que são denotados por $\{\sigma_1, \ldots, \sigma_{k+1}\}$. A transmissão de um 0 leva a seqüência do estado σ_i para o estado σ_{i+1}, $i \leq k$. Um 1 pode ser transmitido apenas quando o grafo ocupar os estados $\sigma_{d+1}, \ldots, \sigma_{k+1}$, enquanto que no

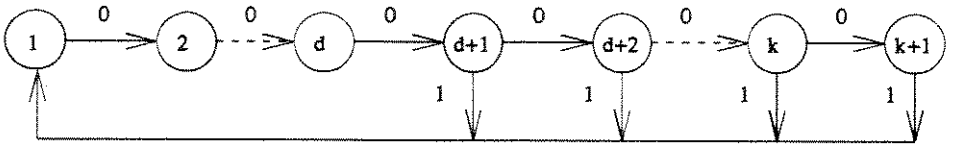

Figura 3.9: Diagrama de transição de estado finito (*FSTD*) para uma seqüência (d, k)

estado σ_{k+1} apenas um 1 pode ser transmitido. O diagrama retorna ao estado σ_1 após a transição de qualquer 1. Qualquer caminho através do *FSTD* define uma seqüência (d, k) permitida [4].

A matriz de transição T, que mostra o número de caminhos direto que vão do estado σ_i para o estado σ_j, sem passar por um estado estacionário, é dada pelo arranjo $(k+1) \times (k+1)$ com entradas t_{ij}, onde

$$t_{ij} = 1, \qquad j = i+1, \quad 1 \leq i \leq k$$

$$t_{ij} = 1, \qquad j = 1, \quad d+1 \leq i \leq k+1$$

$$t_{ij} = 0, \qquad \text{caso contrário}$$

Em forma matricial, temos

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

Exemplo 5 *Seja o código (1,3). Sendo $k = 3$, seu FSTD possui $k+1 = 4$ estados, conforme mostra a figura 3.10. Sua matriz de transição é dada por*

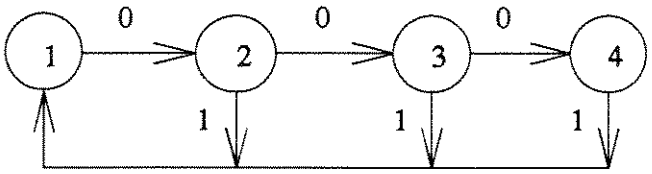

Figura 3.10: Diagrama de transição de estado finito (*FSTD*) para a seqüência (1,3)

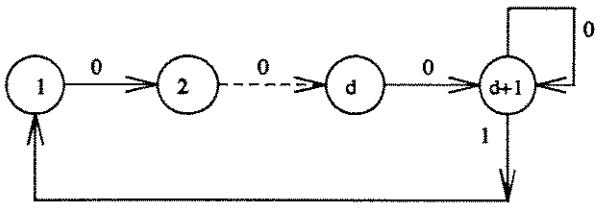

Figura 3.11: Diagrama de transição de estado finito (*FSTD*) para uma seqüência (d)

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

Um canal com restrição d pode ser modelado com $(d+1)$ estados, conforme ilustra a figura 3.11. Apenas no estado σ_{d+1} tanto um 0 como um 1 podem ser transmitidos, nos outros estados apenas um 0.

A matriz de transição T é dada pelo arranjo $(d+1) \times (d+1)$ com entradas t_{ij}, onde

$$t_{ij} = 1, \qquad j = i+1, \quad 1 \leq i \leq d$$

$$t_{ij} = 1, \qquad j = i = d+1$$

$$t_{ij} = 1, \qquad j = 1, \quad i = d+1$$

$$t_{ij} = 0, \qquad \text{caso contrário}$$

Em forma matricial, temos

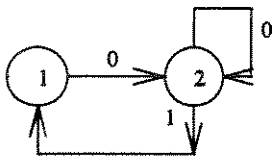

Figura 3.12: Diagrama de transição de estado finito (*FSTD*) para a seqüência (1,∞)

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

Exemplo 6 *Seja o código (1,∞) da figura 3.12, seu FSTD possui* $d+1=2$ *estados e sua matriz de transição é*

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

A estrutura do *FSTD* permite que calculemos a capacidade de um código de restrição (d,k) e também é muito útil para encontrarmos o número de seqüências que partem e terminam em certos estados. Por exemplo, o número de seqüências distintas de comprimento n que partem do estado σ_i e terminam no estado σ_j é dado pelo ij *ésimo* elemento de T^n.

Um modo alternativo para expressar a capacidade de um *FSTD* que emite seqüências de restrição (d,k) é [4]

$$C(d,k) = \lim_{n\to\infty} \frac{1}{n} \log_2 \sum_{i,j} [T]_{ij}^n$$

onde $[T]_{ij}^n$ denota o ij *ésimo* elemento de T^n. Shannon mostrou que [4]

$$C(d,k) = \log_2 \lambda \tag{3.10}$$

onde λ é o maior auto-valor positivo de T. Podemos obter o valor de λ através da equação característica

$$det[T - \lambda I] = 0 \tag{3.11}$$

onde λ é a maior raiz real e I é a matriz identidade.

Vamos agora obter o valor da capacidade de canal sem ruído através da matriz de transição de estado.

Exemplo 7 *Seja o código* $(0,1)$, *sua matriz de transição é*

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

O maior auto-valor positivo de T, *vale* $\lambda = 1,6180$. *Então, pela equação 3.10, temos*

$$C(d,k) = \log_2 1,6180 = 0,6942$$

Este resultado concorda com a capacidade obtida pela série de Fibonacci no exemplo 4.

3.7 Conclusão

Foram apresentados neste capítulo, importantes conceitos como: códigos RLL, capacidade do canal sem ruído e ganho de densidade de armazenamento de dados.

Mostramos também dois métodos para obtenção do valor da capacidade de canal. O primeiro, baseado na construção da treliça do número de palavras-código que satisfazem ao par de restrição (d,k). E o segundo método, expressa a capacidade através da matriz de transição associada a um diagrama de transição de estado finito, que gera seqüências de restrição (d,k).

Podemos observar que ambos os métodos possuem o mesmo princípio, pois estes estão baseados na representação dos estados do canal e suas possíveis transições. Pelas equações 3.5 e 3.10, concluímos que a razão asint'otica (para $n \to \infty$) da série do número de seqüências da treliça é igual ao maior auto-valor positivo da matriz de transição de estado.

A partir daí, obtivemos várias tabelas com os valores do número de seqüências (d,k), da capacidade de canal e ganho máximo de armazenamento para alguns códigos (d,k).

Ambos os métodos são de fácil compreensão e implementação, ficando a critério do projetista o uso determinado de qualquer um destes.

Capítulo 4

Técnicas de Codificação e Decodificação de Códigos RLL

4.1 Introdução

Neste capítulo estudaremos técnicas de codificação e decodificação que produzem seqüências *RLL*. É importante que estas técnicas sejam tão eficientes quanto possível, a fim de que possamos obter taxas próximas à capacidade de canal, e que possuem baixa complexidade.

4.2 Códigos *RLL* de Comprimento Fixo

O método para a construção de códigos *RLL* de estado único, utiliza palavras-código que possam ser livremente cascateadas sem violar as restrições do canal. As palavras-código usadas tem uma correspondência injetora (uma a uma) com as palavras fonte. Baseando-nos neste procedimento, estenderemos este mecanismo para códigos, onde o mapeamento não seja injetor, mas seja, por exemplo, função da última palavra-código transmitida.

Códigos que operam com uma única representação das palavras da fonte são chamados de códigos independentes de estado, ou de estado único, e códigos cuja correspondência não seja injetora são ditos códigos dependentes de estado. Os códigos dependen-

tes de estado freqüentemente possuem a vantagem de atingirem maior eficiência. A fim de tornarmos mais claro este conceito, mostraremos um exemplo de um código $(1, \infty)$ [4].

Exemplo 8 *Seja o seguinte código* $(1, \infty)$. *As palavras-código indicadas na tabela 4.1, ilustram um simples código de bloco que converte blocos fonte de comprimento* $m = 3$ *em palavras-código de comprimento* $n = 5$.

palavras da fonte	*palavras-código*
000	00000
001	00001
010	00010
011	00100
100	00101
101	01000
110	01001
111	01010

Tabela 4.1: Código $(1, \infty)$ de comprimento fixo

A coluna da esquerda mostra as oito palavras da fonte. Notamos que as palavras-código indicadas podem ser livremente cascateadas sem violar a restrição $d = 1$, *pois foi selecionado o 0 para ser o primeiro dígito de cada palavra-código. A tabela 3.1 do capítulo 3, mostra que existem exatamente oito palavras-código de comprimento* $n - 1 = 4$, *que possuem restrição* $d = 1$.

A taxa do código é $m/n = 3/5 < C(1, \infty) = 0.69$. *A eficiência do código, designada por* η, *expressa a relação entre a taxa do código e a capacidade de canal. Neste caso, a eficiência vale*

$$\eta = \frac{R}{C(d,k)} = \frac{0,60}{0,69} = 0,86$$

Este resultado nos mostra que boas eficiências são possíveis de se atingir utilizando construções simples. Melhores eficiências são conseguidas às custas de maiores tamanhos de bloco.

A decodificação na ausência de ruído é realizada de modo muito simples. O decodificador ignora o primeiro bit de cada palavra-código recebida, e usando a tabela de consulta, este faz o mapeamento dos quatros bits restantes. recuperando desta forma a palavra da fonte.

As palavras-código foram associadas às palavras da fonte de modo arbitrário, e evidentemente, outras distribuições podem ser escolhidas. Pode existir um outro mapeamento que simplifique a implementação das tabelas de look-up para o codificador e decodificador.

O procedimento para implementar codificadores (d, ∞) de estado único é direto. Para isto, basta escolhermos palavras-código de comprimento n, em que os primeiros d bits de cada palavra-código sejam 0. O número de palavras-código que satisfaz as restrições é $N_d(n-d)$, que pode ser calculado pela equação 3.1 ou usando a tabela 3.1.

Agora, se quisermos tornar a restrição k finita. que origina um outro código, basta fazermos algumas alterações. Por exemplo, no código $(1, \infty)$ ilustrado na tabela 4.1, o primeiro bit é sempre 0. Se entretanto, o último bit da palavra-código anterior e o segundo bit da palavra-código atual forem 0, então o primeiro bit desta palavra-código pode ser 1 sem que seja violada a restrição $d = 1$ do canal. A regra que conduz a seleção dos primeiros bits, os bits de junção ou bits de confluência, chamada regra de merging, pode ser implementada facilmente com algum hardware extra. Notamos que com a regra de merging acima indicada, o código $(1, \infty)$ apresentado torna-se um código $(1, 6)$.

A eficiência do código $(1, 6)$ acima vale

$$\eta = \frac{3/5}{C(1,6)} = \frac{0,60}{0,669} = 0,897$$

A melhoria da eficiência em relação ao código $(1, \infty)$ não é um indicativo que para obtermos grandes eficiências devamos diminuir a capacidade de canal. A escolha de k está ligada ao circuito de extração de sincronismo, ou seja, seu valor limita o número máximo de 0s consecutivos na seqüência do canal, sem que se produza um tremor (jitter) apreciável do relógio extraído.

O processo de decodificação é o mesmo realizado para o código $(1, \infty)$, pois o primeiro bit, o bit de merging, é omitido de qualquer modo. visto que neste caso, este bit

também não carrega informação. O código $(1,6)$ ilustra o princípio da codificação dependente de estado, quer dizer, a atual palavra-código transmitida depende de palavras-código anteriores. Também ilustra o princípio da decodificação independente de estado, visto que a palavra da fonte recuperada não depende das palavras-código anteriores.

Com o procedimento descrito anteriormente para a construção de seqüências de comprimento fixo, obtemos alguns códigos ilustrados nas tabelas 4.2, 4.3, 4.4 e 4.5.

palavras da fonte	*palavras-código*
00	00000
01	00001
10	00010
11	00100

Tabela 4.2: Código $(2,\infty)$ de comprimento fixo

palavras da fonte	*palavras-código*
00	000000
01	000001
10	000010
11	000100

Tabela 4.3: Código $(3,\infty)$ de comprimento fixo

palavras da fonte	*palavras-código*
00	0000000
01	0000001
10	0000010
11	0000100

Tabela 4.4: Código $(4,\infty)$ de comprimento fixo

palavras da fonte	*palavras-código*
00	00000000
01	00000001
10	00000010
11	00000100

Tabela 4.5: Código $(5, \infty)$ de comprimento fixo

A tabela 4.6 mostra a taxa, a capacidade e a eficiência para os códigos $(2, \infty)$, $(3, \infty)$, $(4, \infty)$ e $(5, \infty)$.

(d, k)	R	C	η
$(2, \infty)$	2/5	0,5515	0,73
$(3, \infty)$	2/6	0,4650	0,72
$(4, \infty)$	2/7	0,4057	0,70
$(5, \infty)$	2/8	0,3620	0,69

Tabela 4.6: Taxa, capacidade e eficiência de alguns códigos de comprimento fixo com $k = \infty$

Tornando a restrição k finita, obtemos os códigos mostrados na tabela 4.7.

(d, k)	R	C	η
$(2, 7)$	2/5	0,5172	0,77
$(3, 9)$	2/6	0,4376	0,76
$(4, 11)$	2/7	0,3833	0,75
$(5, 13)$	2/8	0,3432	0,73

Tabela 4.7: Taxa, capacidade e eficiência de alguns códigos de comprimento fixo com k finito

Como vimos, esta técnica busca construir de forma simples um código de bloco de taxa m/n, que satisfaça as restrições do canal, não se preocupando em obter ótimas eficiências, mas sim uma fácil implementação.

4.2.1 Algoritmo para Obtenção de Códigos de Comprimento Fixo

A principal dificuldade na construção de códigos de comprimento fixo, é encontrar um subconjunto dos estados do canal, conhecidos como estados principais. Destes estados partem um número suficiente de seqüências, pelo menos 2^m, de comprimento n que terminam em outros estados principais. A existência de um conjunto de estados principais é uma condição suficiente para a criação de um código com a taxa e comprimento de palavra-código especificados.

Como foi dito na secção anterior, códigos independentes de estado são aqueles que possuem um único estado principal, e os dependentes de estado possuem mais de um estado principal.

Franaszek desenvolveu um algoritmo de eliminação sucessiva para determinação deste conjunto de estados principais que satisfazem as restrições do canal, através de operações na matriz de transição [4].

Seja m o comprimento da palavra da fonte e n o comprimento da palavra-código. Como já sabemos, as restrições d e k definem o número de estados do canal. O número de seqüências (dk) de comprimento n que partem do estado σ_i e terminam no estado σ_j, e formam um conjunto de estados principais, devem satisfazer a seguinte equação

$$\psi(\sigma_i) \geq 2^m \tag{4.1}$$

Deste modo, $\psi(\sigma_i)$ é um conjunto de estados principais. Pois, pode haver outros subconjuntos de estados proncipais se continuarmos com as eliminações.

Introduzimos a seguir a inequação do auto-vetor aproximado para orientar a construção. Seja um vetor binário $\vec{v} = (v_1, \ldots, v_{k+1})^T$, $v_i \in \{0, 1\}$. Um código de comprimento fixo com parâmetros especificados pode ser determinado se existir um vetor binário $\vec{v}$ que satisfaça [4]

$$T^n \vec{v} \geq 2^m \vec{v} \tag{4.2}$$

Não é difícil ver que o conjunto $\{\sigma_1, \ldots, \sigma_l\}$ para o qual $v_1 = \ldots = v_l = 1$ é um conjunto dos estados principais.

O exemplo a seguir ilustra os pontos mencionados anteriormente.

Exemplo 9 *Modulação de Freqüência Modificada (MFM) ou Código* $(1,3)$ *com taxa 1/2 [5].*

Este é um código muito popular, pois possui simplicidade e fácil implementação, e se tornou um padrão na indústria de tecnologia de discos rígidos e flexíveis.

O código MFM possui alta eficiência $\eta = 0,5/0,5515 \simeq 91\%$ *e ganho de armazenamento* $\mathcal{G} = 0,5(1+1) = 1$. *Da figura 4.1, podemos tirar a matriz de transição* T.

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

Além disso, dado que $n = 2$, *precisamos calcular* T^2.

$$T^2 = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

O diagrama de transição de estado finito de segunda extensão é mostrado na figura 4.2. Como o diagrama é de segunda extensão, partimos de cada estado e chegamos a qualquer outro estado, utilizando dois ramos consecutivos, no caso palavras de informação de 2 bits. Por exemplo, do estado 1 podemos chegar ao estado 3 com a palavra 00, do estado 4 ao estado 2 com a palavra 10, e assim sucessivamente. É claro, que nem todos os estados possuem ligação entre si com apenas dois ramos, como é o caso dos estados 1 e 4, conforme ilustra a figura 4.2.

Utilizando o algoritmo de eliminação sucessiva, notamos que o estado 4, ver figura 4.2, tem que ser eliminado, pois este não satisfaz a equação 4.1, ou seja, temos apenas um ramo partindo deste, quando deveríamos ter 2. Ou pela análise da matriz T^2, *vemos que a soma dos elementos da quarta linha é apenas um. Deste modo, eliminamos a quarta linha e a quarta coluna de* T^2. *A submatriz* T^* *então obtida tem a soma dos elementos em cada linha igual a 2, logo satisfaz a equação 4.2.*

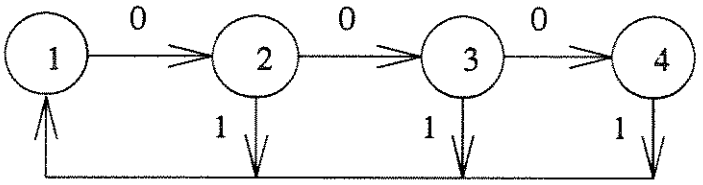

Figura 4.1: Diagrama de transição de estado finito G para a restrição $(1,3)$

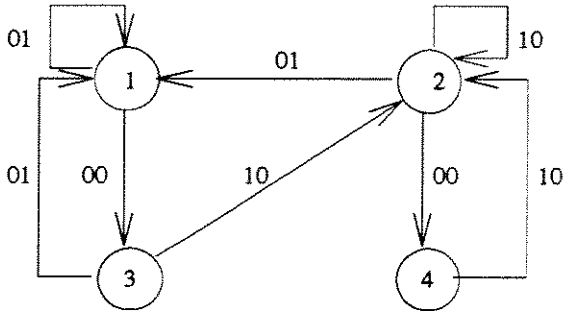

Figura 4.2: Diagrama de transição de estado finito de segunda extensão G^2 para a restrição $(1,3)$

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

Portanto, σ_1, σ_2 e σ_3 formam um conjunto de estados principais. O grafo resultante G^*, visto na figura 4.3, mostra as palavras-código associadas aos estados principais. As palavras-código saindo de cada estado estão descritas abaixo.

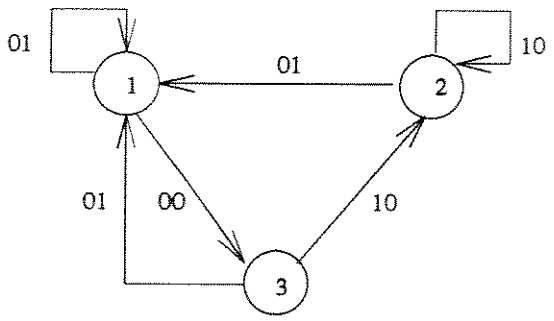

Figura 4.3: Diagrama de transição de estado finito modificado G^* da seqüência $(1,3)$

Dados	Estado A	Estado B
0	10/A	00/A
1	01/B	01/B

Código	Dados
X0	0
01	1

Tabela 4.8: Codificador e decodificador do código MFM

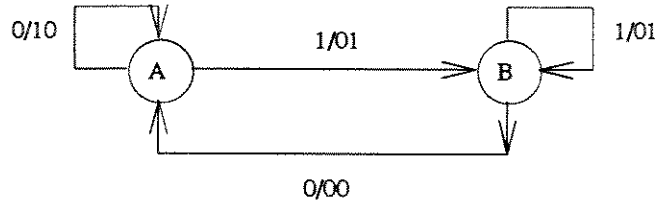

Figura 4.4: Máquina de estado finito do código MFM

$$W(\sigma_1) = \begin{cases} 01 \\ 00 \end{cases} \qquad W(\sigma_2) = \begin{cases} 01 \\ 10 \end{cases} \qquad W(\sigma_3) = \begin{cases} 01 \\ 10 \end{cases}$$

Como se vê, os estados σ_2 e σ_3 tem mesma regra de codificação, e portanto podem ser unidos em um único estado σ_2' Portanto, podemos construir um código com apenas 2 estados, que pode ser visto na figura 4.4.

O codificador é caracterizado por duas regras de codificação. As colunas A e B descrevem os códigos de bloco para as duas regras. A regra A é usada para codificar um bit de informação se o bit anterior for 0, e a regra B se o bit anterior for 1.

Os nós, ou estados, no grafo correspondem às colunas na tabela do codificador, e os ramos são rotulados segundo o par x/y onde x é a palavra de informação e y é a sua correspondente palavra-código. A rotulação dos ramos que partem de um estado define a regra de codificação, e o estado no qual um ramo termina indica a próxima regra de codificação. Em cada ciclo do codificador, temos a codificação de um único bit de informação em dois bits de código.

A decodificação na ausência de ruído mostrada na tabela 4.8 é do tipo independente de estado, e é realizada simplesmente descartando o primeiro bit redundante em cada bloco de 2 bits recebidos. O símbolo X na tabela 4.8 representa uma posição, cujo valor não afeta a decodificação. Por exemplo, tanto 10 como 00, quando decodificados correspondem a 0.

De modo a fortalecer o entendimento do assunto, vamos analisar um outro exemplo.

Exemplo 10 *Seja um código (d,k) com as seguintes restrições* $d = 2$ *e* $k = \infty$. *Sua matriz de transição é dada por*

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

Como já foi visto, a capacidade de canal é o logaritmo na base 2 do maior auto-valor positivo de T. *Portanto, a capacidade vale*

$$\mathcal{C}(2, \infty) = \log_2 1,465 = 0,551$$

Após várias eliminações sucessivas, a procura de uma matriz T^n *que gere um codificador, onde a condição* $m/n = 1/2$ *seja obedecida, chegamos ao comprimento mínimo de palavra-código* $n = 14$. *Seja a matriz de transição*

$$T^{14} = \begin{bmatrix} 41 & 28 & 60 \\ 60 & 41 & 88 \\ 88 & 60 & 129 \end{bmatrix}$$

Visto que todos os três estados satisfazem a equação 4.1,

$$\sum_{j=1}^{3} [T]_{1j}^{14} = 129$$

$$\sum_{j=1}^{3} [T]_{2j}^{14} = 189$$

$$\sum_{j=1}^{3} [T]_{3j}^{14} = 277$$

ou seja, mais do que $2^m = 2^7 = 128$ seqüências partem de cada estado, estes são os estados principais. A fim de construirmos um código de taxa 7/14, podemos escolher 128 das 129 possíveis seqüências e associar palavras-código para as correspondentes palavras da fonte. Observando a matriz T^{14}, vemos que o elemento inferior da direita é maior do que 128. Portanto, apenas o estado σ_3 é suficiente para codificação. Sendo assim, este código em particular pode ser codificado e decodificado por um único estado principal, visto que todas as seqüências partem e terminam no estado σ_3. Neste caso, o conjunto mínimo de estados principais é um subconjunto de Franesk.

Uma outra maneira seria consultando a tabela 3.1. Podemos ver que para $d = 2$, existem exatamente 129 seqüências de comprimento 12. A adição de dois bits de merging, representados por 0, completa o projeto do código $(2, \infty)$ com taxa 7/(12+2) = 7/14. O que mostra que a técnica de codificação de estado único é um caso particular da codificação de múltiplos estados.

Códigos de comprimento fixo foram calculados por Franaszek através do processo de eliminação sucessiva, os resultados são mostrados na tabela 4.9 [4].

d	k	m	n	$\mathcal{G} = R(d+1)$	$\eta = R/\mathcal{C}(d,k)$
0	1	3	5	0,60	0,864
0	2	4	5	0,80	0,910
0	3	9	10	0,90	0,951
1	3	1	2	1,00	0,907
1	7	22	33	1,33	0,981
2	5	4	10	1,20	0,860
2	7	17	34	1,50	0,962
2	10	8	16	1,50	0,951
3	7	46	115	1,60	0,923
3	11	8	20	1,60	0,986
4	9	9	27	1,67	0,921
4	14	12	33	1,82	0,916
5	12	9	30	1,80	0,890
5	17	15	45	2,00	0,937

Tabela 4.9: Códigos de comprimento fixo

Este método em alguns casos se torna impraticável devido ao grande valor do comprimento de bloco que obtemos, na procura de uma matriz T^n que gere um codificador com uma taxa $R < \mathcal{C}$ pré-estabelecida.

4.3 Codificação Enumerativa de Seqüências (d)

Nos exemplos anteriores, assumimos que as tabelas de codificação e decodificação utilizadas continham todas as possíveis palavras-código. Contudo, quando as palavras-código se tornam muito longas e em número excessivo, o método de procura direta na tabela pode se tornar impraticável.

Entretanto, podemos gerar palavras-código por um processo algébrico, chamado codificação enumerativa, que significa que não precisamos ter necessariamente todas as palavras-código armazenadas na tabela. O objetivo é desenvolver uma técnica geral de codificação e decodificação de seqüências (d,k). Aqui mostraremos esta técnica apenas para seqüências (d), o procedimento pode ser generalizado para seqüências (d,k).

Com este propósito, vamos estabelecer um mapeamento do conjunto $D(d,n)$ de seqüências d de comprimento n no conjunto de inteiros $0,1,\ldots,|D(d,n)|-1$, onde $|D(d,n)| = N_d(n)$ representa a cardinalidade de $D(d,n)$. O conjunto D pode ser ordenado como segue: se $\vec{x} = (x_{n-1},\ldots,x_0) \in D$ e $\vec{y} = (y_{n-1},\ldots,y_0) \in D$, então $\vec{y}$ é dito ser menor do que $\vec{x}$, ou seja, $\vec{y} < \vec{x}$, se existir um i entre os limites $0 < i < n$, tal que $y_i < x_i$ e $x_j = y_j$, $i < j < n$. Por exemplo, $00101 < 01010$.

A posição de $\vec{x}$ na ordenação lexicográfica das seqüências é definida como sendo o rank de $\vec{x}$, chamado $r(\vec{x})$, isto significa que $r(\vec{x})$ é o maior número de todos $\vec{y}$ em D, com $\vec{y} < \vec{x}$.

Teorema 1 *O rank $r(\vec{x})$ das seqüências (d), onde $\vec{x} \in D$, pode ser calculado de acordo com a seguinte equação [4]*

$$r(\vec{x}) = \sum_{j=0}^{n-1} N_d(j)x_j \tag{4.3}$$

Daremos um exemplo para melhor compreensão deste conceito.

Exemplo 11 *Considere o conjunto $D(d,n) = D(1,4)$, ou seja, seqüências de restrição $d = 1$ de comprimento $n = 4$. Da equação 3.1, temos*

$$N_d(n) = n+1, \quad 1 \le n \le d+1$$

$$N_d(n) = N(n-1) + N(n-d-1), \quad n > d+1$$

Logo: $N_1(0) = 1$, $N_1(1) = 2$, $N_1(2) = 3$, $N_1(3) = 5$ e $N_1(4) = 8$. Portanto, para $n = 4$ teremos oito palavras-código, das $2^n = 2^4 = 16$ combinações possíveis, que obedecem esta restrição. Usando a equação 4.3, obtemos o rank destas palavras-código.

Por exemplo, para $\vec{x} = 0010$, temos

$$r(0010) = \sum_{j=0}^{3} N_1(j)x_j$$

$$r(0010) = N_1(0)x_0 + N_1(1)x_1 + N_1(2)x_2 + N_1(3)x_3$$

$$= 1.x_0 + 2.x_1 + 3.x_2 + 5.x_3$$

$$= 1.0 + 2.1 + 3.0 + 5.0$$

$$r(0010) = 2$$

E assim, calculamos para cada palavra-código, conforme ilustra a tabela 4.10.

$\vec{x}$	$r(\vec{x})$	$\vec{y}$
0000	0	000
0001	1	001
0010	2	010
0100	3	011
0101	4	100
1000	5	101
1001	6	110
1010	7	111

Tabela 4.10: Tabela de decodificação enumerativa do conjunto $D(1,4)$

Desta maneira, conseguimos criar um decodificador, pois dado que recebemos $\vec{x} = (0010)$, calculamos o seu rank $r(\vec{x}) = 2$. O vetor cuja representação decimal vale 2 é $\vec{y} = (010)$, que é o vetor decodificado.

A função inversa, isto é, a conversão de um dado inteiro l em uma seqüência (d,k) com rank l pode ser feita como segue.

Algoritmo 1 *Dado o conjunto $D(d,n)$ de seqüências e um inteiro l, $l = 0, 1, \ldots, |D(d,n)| - 1$. O algoritmo encontra $\vec{x}$, tal que $r(\vec{x}) = l$ [4].*

Seja $\hat{l} = l$. Para j de $n-1$ a 0, faça

- *se $\hat{l} \geq N_d(j)$ então $x_j = 1$, e $\hat{l} = \hat{l} - N_d(j)$*
- *caso contrário, $x_j = 0$*

Exemplo 12 *Seja o caso anterior,* $D(d,n) = D(1,4)$. *Sabemos que:* $N_1(0) = 1$, $N_1(1) = 2$, $N_1(2) = 3$, $N_1(3) = 5$ *e* $N_1(4) = 8$. *Encontrar* $\vec{x}$, *tal que* $r(\vec{x}) = 4$.

Seja $\hat{l} = 4$. *Para* j *de 3 a 0, temos*

$4 < N_1(3) = 5 \Rightarrow x_3 = 0$

$4 > N_1(2) = 3 \Rightarrow x_2 = 1$ *e* $\hat{l} = 4 - N_1(2) = 1$

$1 < N_1(1) = 2 \Rightarrow x_1 = 0$

$1 = N_1(0) = 1 \Rightarrow x_0 = 1$

Logo: $\vec{x} = 0101$, *o que coincide com a tabela 4.10.*

Portanto, temos um algoritmo que nos permite codificar uma dada seqüência da fonte, por exemplo, $\vec{y} = (001)$. A representação decimal de $\vec{y}$ é 1. Temos então, que procurar a palavra-código cujo rank é 1.

A técnica de rank descrita na definição 1, permite a construção de codificadores e decodificadores de canal de complexidade moderada.

4.4 Códigos Síncronos de Comprimento Variável

Como vimos nas secções anteriores, a tentativa de se aumentar a eficiência dos códigos de comprimento fixo dependentes de estado, resulta num aumento do comprimento das palavras-código, e desse modo, numa maior complexidade do codificador e decodificador.

Os códigos de comprimento variável combinam as vantagens do uso de palavras-código de comprimento pequeno e longo, onde as de comprimento pequeno ocorrem mais freqüentemente do que as de comprimento longo. Além disso, permitem uma marcante redução na complexidade do codificador e decodificador em relação aos códigos de comprimento fixo de taxa equivalente e com as mesmas restrições.

Há porém, a exigência da transmissão síncrona, ou seja, a taxa deve ser fixa, visando simplificar a implementação. Além disso, o fato de que cada palavra deve ter um número inteiro de bits de informação, conduz ao fato de que os comprimentos das palavras-código devam ser múltiplos inteiros de uma palavra básica de comprimento n, onde n é o menor inteiro, tal que a razão m/n seja obtida através de inteiros.

Para aplicarmos o método [4], escolhemos uma palavra da fonte de comprimento m e uma palavra-código básica de comprimento n, junto com um comprimento máximo Mn. Portanto, as palavras podem ser de comprimento jn, $j = 1, 2, \ldots, M$. Apresentaremos dois exemplos de códigos de comprimento variável, o primeiro dirigido ao problema da construção do codificador e o segundo analisando a decodificação.

Exemplo 13 *Escolhemos os mesmos parâmetros de restrição do exemplo 10,* $d = 2$ *e* $k = \infty$. *A capacidade do código* $(2, \infty)$ *vale* $C(2, \infty) = 0,5515$, *portanto podemos usar taxa 1/2.*

Para a taxa $R = 1/2$, *teremos duas palavras de informação e três palavras-código, que obedecem a restrição* $(2, \infty)$. *Contudo, cascateando estas palavras vemos que as*

palavras de informação	palavras-código
0	00
1	01
	10

Tabela 4.11: Combinação de palavras do código $(2, \infty)$ para taxa $1/2$

mesmas não satisfazem a restrição $(2, \infty)$.

Para que não tenhamos tantos problemas no cascateamento, é interessante que o comprimento de bloco n seja maior do que d. Por isso, escolheremos $n = 4$, *e teremos taxa 2/4 e seis palavras-código que obedecem a restrição. São elas: 0000, 0001, 0010, 0100, 1000 e 1001. Entretanto, apenas algumas combinações podem ser cascateadas sem violar a restrição, por exemplo, as palavras 0000, 0100 e 1000, conforme tabela 4.12. As demais palavras serão eliminadas.*

palavras de informação	palavras-código
00	0000
01	0100
10	1000
11	

Tabela 4.12: Combinação de palavras do código $(2, \infty)$ para taxa 2/4

Como para taxa 2/4, temos quatro palavras de informação e apenas três palavras-código que obedecem a restrição, e que podem ser cascateadas, agregamos as duas primeiras palavras de informação em apenas uma, obtendo a tabela 4.13.

palavras de informação	*palavras-código*
0	00
10	0100
11	1000

Tabela 4.13: Código $(2, \infty)$ síncrono de comprimento variável

Notamos que se o bit de informação for 0, a palavra 00 será transmitida, ao invés de 0000, pois a taxa 1/2 será mantida. Caso contrário, se o bit de informação for 1, o codificador transmitirá 0100 ou 1000, dependendo se o próximo bit for 0 ou 1, respectivamente.

A decodificação pode ser realizada univocamente sem conhecermos onde os blocos de comprimento variável começam ou terminam, pois este código satisfaz a condição de prefixo. Um código de bloco de comprimento variável é univocamente decodificável se o mesmo for um código de prefixo, isto é, se em seu conjunto de M seqüências binárias $\{c_0, \ldots, c_{M-1}\}$, a palavra-código c_u não é o início de outra palavra-código c_v para qualquer $u \neq v$ e para todo u.

Este exemplo ilustra muito bem as vantagens da codificação de bloco de comprimento variável, e mostra como o código de comprimento fixo com um dicionário de 128 palavras, conforme o do exemplo 10, pode ser substituído por um com apenas três palavras. Além disso, transmitiremos a palavra 0 a metade das vezes, e a outra metade será dividida com as palavras 10 e 11.

Exemplo 14 *Código $(2,7)$ de taxa 1/2.*

A tabela 4.14 ilustra o processo de codificação do código $(2,7)$ de taxa 1/2. A construção do mesmo, obtida de forma heurística, será omitida aqui.

palavras de informação	*palavras-código*
10	1000
11	0100
011	000100
010	001000
000	100100
0011	00100100
0010	00001000

Tabela 4.14: Código $(2,7)$ síncrono de comprimento variável

A codificação dos dados de entrada é realizada dividindo-se a seqüência da fonte em blocos de dois, três ou quatro bits, e então mapeando estes nos blocos correspondentes do canal. É importante lembrar que estas palavras satisfazem a condição de prefixo. A seguir mostraremos como esta tabela é usada.

Seja a seqüência fonte 010111010. Após a segmentação da seqüência fonte, temos 010 11 10 10 .... Usando a tabela 4.14, temos a correspondente seqüência de saída 001000 0100 1000 1000 ....

A tabela 4.15 mostra as mesmas palavras da fonte com uma permutação de determinadas palavras-código, que conforme veremos apresentam vantagens em relação às palavras da tabela 4.14.

palavras de informação	*palavras-código*
10	0100
11	1000
011	001000
010	100100
000	000100
0011	00001000
0010	00100100

Tabela 4.15: Código $(2,7)$ síncrono de comprimento variável

No caso da tabela 4.15, a decodificação da mensagem recebida pode ser realizada através de um registrador de deslocamento de comprimento oito. O conteúdo do registrador é decodificado pela expressão Booleana [4]

$$d_0 = x_4\bar{x}_1 + x_7\bar{x}_5x_3 + x_3x_8 + x_6,$$

onde d_0 é o bit decodificado e x_i, $1 \leq i \leq 8$, representa o conteúdo do registrador de deslocamento. Como pode ser visto pela expressão Booleana acima, a propagação dos erros na seqüência decodificada, devido a um único erro de bit de canal é limitada. Qualquer erro no bit recebido pode transmitir um erro em até dois bits de dados decodificados consecutivos: o bit atual e o bit anterior.

A tabela 4.14 foi apresentada por Franaszek [4], e esta possui uma desvantagem, é necessário um registrador de deslocamento de comprimento doze, que aumenta a extensão do erro em até seis bits decodificados. Este exemplo demonstra que a distribuição das palavras-código em um código de comprimento variável pode ter um efeito decisivo na característica de propagação de erro do código. Inspecionando a tabela 4.9, vemos que o código de bloco de comprimento fixo de menor comprimento que gera o código $(2,7)$ tem palavras-código de comprimento $n = 34$. Evidentemente, o código síncrono de comprimento variável é mais atrativo devido as respectivas exigências de hardware.

4.5 Técnica de Codificação Antecipativa

Um código de bloco é dito ser antecipativa, se a codificação e decodificação de um bloco corrente depende de blocos de palavras à frente no tempo. Esta técnica tem sido

usada para produzir códigos *RLL* práticos e eficientes.

O projeto do código é realizado com o uso da inequação 4.2 do auto-vetor aproximado. Seja um código de taxa m/n. O auto-vetor aproximado $\vec{v}$ é um vetor não negativo de elementos inteiros, que satisfaz a inequação [5]

$$T^n \vec{v} \geq 2^m \vec{v}$$

Se a matriz T^n não possue uma submatriz que obtida pela eliminação de linhas e colunas correspondentes, cuja soma dos elementos de linhas pelo menos igual a 2^m, então algum componente de $\vec{v}$ será maior do que 1 e a antecipaçào será necessário. Isto é, precisamos de blocos de palavras à frente para realizarmos a decodificação. A maior componente de $\vec{v}$ determina a máxima extensão de antecipativa, ou seja, o máximo número de blocos de palavras à frente que poderemos precisar para a decodificação.

Um exemplo de um projeto de um código baseado no método antecipativo é o código $(1,7)$ de taxa $2/3$.

Exemplo 15 *Seja o código* $(1,7)$ *com capacidade* $C(1,7) = 0,6793$, *portanto a taxa 2/3 pode ser utilizada. Esta técnica se inicia de modo similar ao código de bloco de comprimento fixo, onde palavras de dois bits são convertidas em palavras-código de 3 bits. Como a taxa é igual a 2/3, teremos* $2^m = 2^2 = 4$ *palavras de informação e* $2^n = 2^3 = 8$ *palavras, sendo que das oito palavras, apenas quatro obedecem a restrição* $(1,7)$. *São elas: 001, 010, 100 e 101.*

Para entendermos o algoritmo de antecipação, nós começaremos com a tabela 4.16 de codificação básica de um código $(1,7)$.

palavras de informação	*palavras-código*
00	001
01	010
10	100
11	101

Tabela 4.16: Tabela de codificação básica do código $(1,7)$

Cascateando estas quatro palavras-código, vemos que algumas combinações violam a restrição $(1,7)$, o que mostra a tabela 4.17.

00.10	001.100
00.11	001.101
11.10	101.100
11.11	101.101

Tabela 4.17: Tabela de violação da restrição $(1,7)$

Para que isto não ocorra criamos a tabela 4.18 de substituição da violação.

palavras de informação	*palavras-código*
00.10	001.000
00.11	010.000
11.10	101.000
11.11	100.000

Tabela 4.18: Tabela de substituição da violação do código $(1,7)$

Obtivemos a partir das tabelas 4.16 e 4.18, a tabela 4.19 de codificação antecipativa para o código $(1,7)$. Observe a criação de um estado de violação, condição necessária para que a restrição $(1,7)$ seja obedecida, quando as palavras-código forem cascateadas.

Dados	Estado A	Estado B	Estado C	Estado D	Estado V
00	001/A	010/A	100/A	101/A	000/A
01	001/B	010/B	100/B	101/B	000/B
10	001/V	010/C	100/C	101/V	000/C
11	010/V	010/D	100/D	100/V	000/D

Tabela 4.19: Tabela de codificação de antecipativa do código $(1,7)$

Para entendermos melhor, suponha a seguinte seqüência de entrada:

00 01 11 01 10 11 11 00 00 ...

De acordo, com a tabela de codificação 4.19, teremos a saída da forma:

001 010 101 010 100 100 000 001 ...

E é fácil verificar que obteremos a seqüência de entrada, caso decodifiquemos esta seqüência.

A tabela 4.20 mostra a taxa, a capacidade, o ganho de densidade e a eficiência para alguns pares de restrição (d,k), utilizando a técnica de codificação antecipativa.

(d,k)	$R=m/n$	$\mathcal{C}(d,k)$	$\mathcal{G}=R(d+1)$	$\eta=R/\mathcal{C}$
(1,7)	2/3	0,68	1,33	0,98
(2,9)	2/4	0,54	1,50	0,93
(3,∞)	2/5	0,46	1,60	0,87

Tabela 4.20: Taxa R, capacidade $\mathcal{C}$, ganho de densidade $\mathcal{G}$ e eficiência η para alguns pares de restrição (d,k)

4.6 Algoritmo de Blocos Deslizantes

O principal objetivo do algoritmo de blocos deslizantes é modificar a máquina de transição de estado finito pela divisão e/ou junção de alguns estados do canal, de modo a obter uma nova máquina de estado finito que obedeça às restrições do canal.

O algoritmo usa o auto-vetor $\vec{v}$ para orientar a construção de um codificador, ou seja, uma máquina de transição de estado finito. A componente do auto-vetor associada a um estado s no *FSTD* de extensão n (G^n) vale v_s. O algoritmo produz um novo *FSTD*, que nós chamaremos H, pela divisão do estado s em v_s novos estados e pela especificação dos ramos de transição no novo conjunto de estados.

O procedimento de divisão de estados garante que o *FSTD* H resultante gere o mesmo conjunto de seqüências do código de restrição que G^n, mas em adição, cada estado tem pelo menos 2^m ramos de saída. Além disso, o algoritmo de divisão de estados garante que o decodificador para este código tenha uma estrutura de de blocos deslizantes finita, quer dizer, um número finito de blocos de palavras que devemos precisar para decodificação, assegurando uma propagação de erro limitada.

Este algoritmo tem a vantagem de possuir bases matemáticas bem conhecidas. O procedimento para a construção de códigos *RLL* de taxa $m/n \leq C$ tem obtido bom êxito.

Para melhor compreensão ilustraremos a idéia com exemplo seguinte.

Exemplo 16 *Seja o código* $(0,1)$ *de taxa 2/3. Suas matrizes de transição* T *e* T^3 *são dadas por*

$$T = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \quad e \quad T^3 = \begin{bmatrix} 3 & 2 \\ 2 & 1 \end{bmatrix}$$

Os diagramas de transição de estado finito associados a T *e* T^3 *são mostrados nas figuras 4.5 e 4.6, respectivamente.*

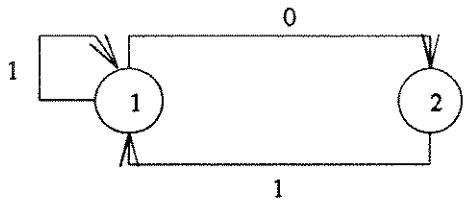

Figura 4.5: Diagrama de transição de estado finito G da seqüência $(0,1)$

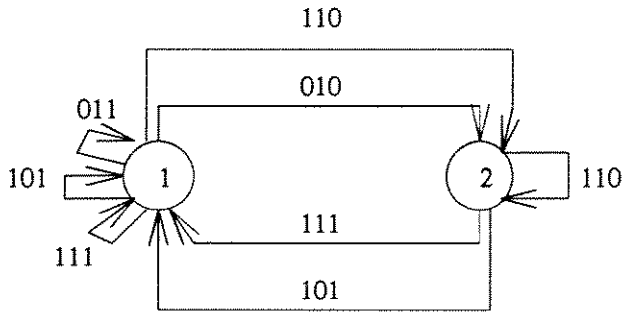

Figura 4.6: Diagrama de transição de estado finito de terceira extensão G^3 da seqüência $(0,1)$

Um auto-vetor que satisfaz a inequação 4.2 é o vetor $\begin{bmatrix} 2 \\ 1 \end{bmatrix}$ *pois,*

$$T^3\vec{v} = \begin{bmatrix} 3 & 2 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 1 \end{bmatrix} \geq 2^2 \begin{bmatrix} 2 \\ 1 \end{bmatrix} = 2^2\vec{v}$$

Este auto-vetor aproximado $\vec{v} = (2,1)$ *indica que o estado 1 será dividido em dois estados, enquanto o estado 2 não poderá ser dividido. Os dois estados em que o estado 1 é dividido são chamados* 1^1 *e* 1^2*. Dessa forma, os ramos que saem e retornam ao estado 2 permanecem inalterados. Os ramos que saíam do estado 2 e íam para o estado 1, agora vão para os estados* 1^1 *e* 1^2*. Os ramos de saída do estado 1 serão particionados em dois grupos que sairão dos dois novos estados resultantes, obedecendo a condição de que cada estado tenha pelo menos* 2^m *ramos de saída.*

Definição 2 *O peso de um estado terminal é definido como sendo o valor da componente correspondente do auto-vetor* $\vec{v}$*.*

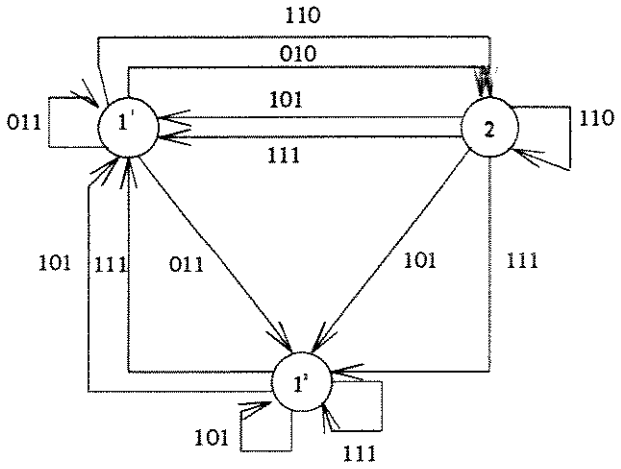

Figura 4.7: Diagrama de transição de estado finito resultante da seqüência $(0, 1)$

Sendo $\vec{v} = (2, 1)$, isto significa que, cada ramo que chega ao estado 1 possue peso 2 e cada ramo que chega ao estado 2 possue peso 1. Assim, pela figura 4.6, vemos que os ramos que saem do estado 1 tem peso total 8.

A regra de desmembramento exige que a soma dos pesos dos estados terminais dos ramos em cada grupo deva ser pelo menos um múltiplo inteiro de 2^2. Separando os ramos que saem do estado 1 em dois grupos [011, 110, 010] e [101, 111], notamos que ambos tem peso total exatamente 4. Isto quer dizer que a regra é obedecida. O primeiro grupo consistirá de ramos que sairão do novo estado 1^1, e o segundo de ramos que sairão do estado 1^2.

Notemos que pode existir mais de uma escolha de partição de ramo que satisfaça a regra de divisão. O FSTD resultante, chamado H, é mostrado na figura 4.7.

O FSTD H obedece às mesmas restrições que G^3, mas tem pelo menos quatro ramos de saída em cada estado. Podemos assim, descartar o enlace do estado 2 por exemplo. Podemos ainda, classificar os ramos para cada estado com palavras de entrada de 2 bits para obter um codificador.

Os estados 1^2 e 2 possuem mesma regra de codificação, então estes podem ser incorporados, simplificando mais o codificador FSM que passa a ter apenas dois estado, conforme a figura 4.8.

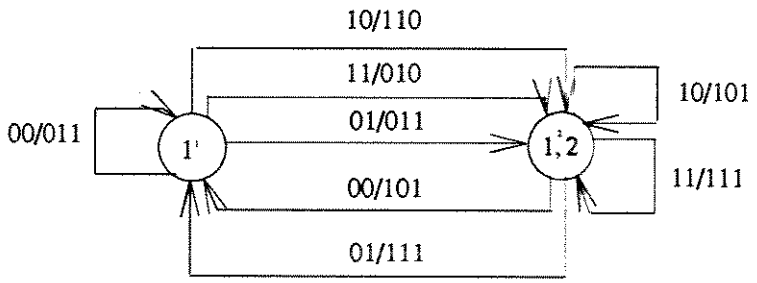

Figura 4.8: Codificador da máquina de estado finito da seqüência $(0,1)$

O algoritmo de divisão de estado que generaliza o método de codificação de estado de comprimento fixo, corresponde ao caso especial de um auto-vetor aproximado inteiro com algum componente maior do que 1.

4.7 Conclusão

As técnicas de comprimento fixo buscam construir de forma simples um código de bloco de taxa m/n, que satisfaça as restrições do canal, não se preocupando em obter ótimas eficiências. Contudo, vimos que o algoritmo para obtenção destes códigos pode se tornar inviável devido ao grande aumento do comprimento de bloco necessário para alcançarmos um codificador eficiente.

A codificação enumerativa nos fornece um método algébrico de codificação e decodificação das palavras-código, com isso, evitamos o problema da procura excessiva em tabelas quando as palavras-código se tornam muito longas, e se apresentam em grandes quantidades.

A diferença entre as técnicas de antecipativa e de blocos deslizantes está apenas no rigor matemático oferecido pela segunda, pois ambas utilizam-se do mesmo princípio. Vimos também, que o algoritmo de blocos deslizantes oferece uma sólida estrutura teórica para a construção de códigos de gravação de dados.

Capítulo 5

Codificação e Decodificação de Dados na Presença de Ruído

5.1 Introdução

Um canal de gravação magnética é projetado de modo a alcançar uma alta densidade de armazenamento com uma alta confiabilidade de dados. Em um sistema de gravação de dados típico, um codificador e decodificador de correção de erro e um codificador e decodificador de gravação são empregados independentemente, como mostra a figura 5.1.

Os códigos limitados por comprimento serial (RLL) realizam importantes propósitos, incluindo a diminuição da interferência inter-simbólica, auxiliam na recuperação do sincronismo, e são usados para satisfazer as restrições do canal de gravação, a fim de obter uma alta densidade de armazenamento de dados. Entretanto, estes códigos não corrigem os eventuais erros provocados pelo canal. Devido a este problema, os códigos corretores de erro (ECC - error correcting codes) são necessários para a correção de tais erros, de modo a

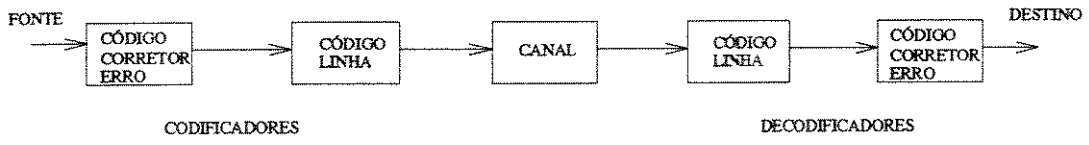

Figura 5.1: Diagrama de blocos de um sistema de gravação de dados

um sistema de armazernamento confiável.

Quando um código *ECC* e um código *RLL* são utilizados separadamente, estes são cascateados, com o *ECC* sendo o código exterior e o *RLL* sendo o código interior. No processo de escrita, primeiramente os dados são passados por um codificador *ECC* e a saída deste é codificada pelo codificador de gravação. Na leitura, as operações são desempenhadas em ordem reversa, isto é, os bits de canal primeiro são passados através de um decodificador de gravação e a saída deste, é então passada pelo decodificador *ECC*.

A fim de simplificar e melhorar o desempenho dos esquemas de codificação e decodificação convencionais, os códigos *ECC/RLL* combinados são utilizados [9]. Códigos *ECC/RLL* combinados são códigos *RLL*, cujas palavras-código não apenas satisfazem as restrições d e k, mas também possuem capacidade de correção de erro. Estes códigos combinados são representados em treliça, e possuem uma fácil implementação tanto para a codificação quanto para o decodificação, visto que apenas um codificador e um decodificador são necessários.

Na próxima secção, dois códigos *ECC/RLL* combinados são descritos. Um é o código $(1,3)$ de dois estados e o outro é o código $(2,6)$ de quatro estados.

5.2 Códigos de Treliça *ECC/RLL* Combinados

A figura 5.2 mostra um código $(1,3)$ de dois estados com distância livre de Hamming igual a 4. A figura 5.3, mostra o código $(2,6)$ de quatro estados e distância livre de Hamming igual a 3 [9].

Definição 3 *A distância livre de Hamming é definida como sendo a menor distância de Hamming entre quaisquer duas seqüências que partem de um mesmo estado e terminam em um mesmo estado posterior.*

Os ramos da treliça são classificados por x/y, onde x indica os dados de entrada e y representa os dados de saída. A taxa para ambos os códigos é igual a $1/3$.

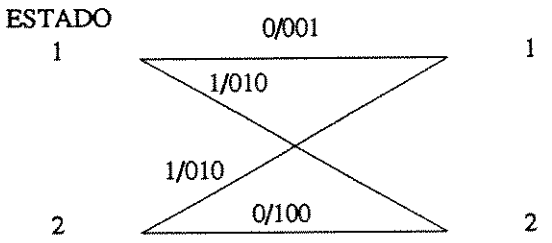

Figura 5.2: Representação em treliça do código $(1,3)$

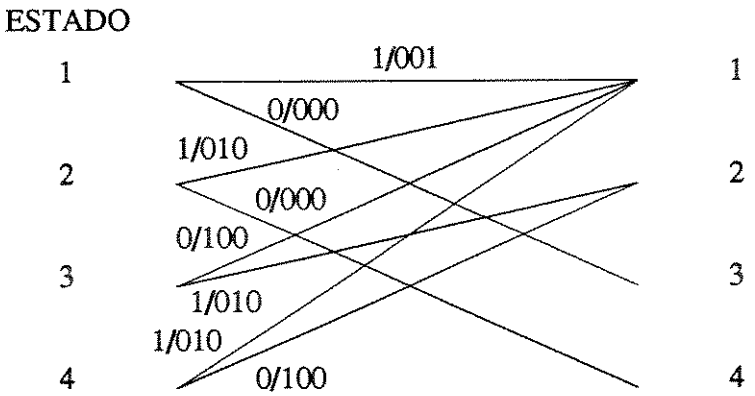

Figura 5.3: Representação em treliça do código $(2,6)$

Definição 4 *Ganho de codificação por decisão suave [8].*

O ganho da relação sinal/ruído de um código de treliça usando decisão suave é dado por

$$\mathcal{G}_{COD} = Rd_{free} \tag{5.1}$$

onde, d_{free} é a distância livre e R é a taxa do código convolucional.

Deste modo, o ganho $\mathcal{G}_{COD}$ para os códigos $(1,3)$ e $(2,6)$ vale $4/3$ e 1, respectivamente.

Estes códigos, que também são de gravação e de taxa $1/3$, tem ganho de densidade de armazenamento $\mathcal{G} = R(d+1)$, igual a $2/3$ e 1, respectivamente. Como podemos notar, não obtivemos nenhum ganho de armazenamento. Entretanto, o código $(1,3)$ apresenta um pequeno ganho de codificação, na ordem de 1,24 dB.

Sabemos que para aumentarmos a densidade de armazenamento de dados, devemos diminuir a distância mínima entre transições. Com isso. corremos o risco de introduzir IIS, e conseqüentemente aumentarmos a probabilidade de erro. Contudo, ao empregarmos a codificação corretora estaremos adicionando bits redundantes à seqüência da fonte, e por sua vez diminuindo a probabilidade de erro. Assim, poderemos para uma mesma probabilidade de erro aumentar a densidade de armazenamento de dados. fazendo uso da codificação.

Portanto, se o canal permitir que este ganho de codificação possa ser transferido para o ganho de armazenamento, então o ganho de armazenamento resultante será o produto dos ganhos de codificação e de armazenamento, ou seja

$$\mathcal{G}_{ARMZ\,TOTAL} = \mathcal{G}_{COD} \cdot \mathcal{G}_{ARMAZ} \tag{5.2}$$

Apesar de levarmos em conta o ganho dos *ECC*. ainda assim estes códigos não são muito bons, mas possuem a vantagem de ter fácil implementação.

5.3 Códigos Limitados por Comprimento Serial Preservando Distância

5.3.1 Introdução

Como vimos, o emprego de códigos corretores de erro e códigos de gravação combinados conduz a um método de codificação e decodificação simples em sistemas de gravação magnética. A idéia básica é criar códigos que ofereçam alguma capacidade de correção de erros e ao mesmo tempo satisfaçam as restrições do canal. Estes tipos de códigos são de grande interesse, pois quando os códigos *ECC* e *RLL* são usados separados, o código *RLL* poderá reduzir a efetividade do código *ECC*.

Com o propósito de solucionar este problema, foi introduzido um subconjunto de códigos *RLL*, que chamamos de códigos *RLL* que preservam distância [10]. Além de satisfazer às restrições (d, k), estes códigos tem a propriedade de que a distância de Hamming entre quaisquer duas seqüências de saída do codificador é pelo menos tão grande quanto a distância de Hamming entre as correspondentes seqüências de entrada do mesmo. Desse modo, quando usado em cascata com um código binário *ECC*, um código *RLL* preservando distância não reduzirá a distância de Hamming total da combinação *ECC/RLL* para algo abaixo da distância de Hamming do código *ECC* sozinho. Mostraremos como códigos *RLL* preservando distância podem ser usados com códigos binários convolucionais, de modo que se preserve as propriedades de distância do código convolucional original.

5.3.2 Códigos Limitados por Comprimento Serial Preservando Distância

Apesar de muitos códigos *RLL* utilizados atualmente não serem projetados para preservar a distância, um destes códigos, o código de Miller, satisfaz aos critérios de preservação da distância. Por essa razão, utilizaremos este código bem conhecido para ilustrar exatamente o que queremos dizer por códigos *RLL* preservando distância.

O código de Miller, também conhecido como MFM, como vimos na capítulo 4, é um código $(d, k) = (1, 3)$ de taxa $1/2$. Considere o codificador para o código de Miller mostrado na tabela 5.1.

Estado 1		
Entrada	Saída	Próximo Estado
0	10	1
1	01	2

Estado 2		
Entrada	Saída	Próximo Estado
0	00	1
1	01	2

Tabela 5.1: Codificador do código de Miller

O codificador consiste de uma máquina de dois estados que converte um único bit de entrada em dois bits de saída. Para verificar que este código preserva distância, usaremos uma nova ferramenta chamada perfil distância.

Definição 5 *O perfil distância de um conjunto de seqüências é uma matriz com elementos positivos [10], que valem*

$$D(x_i, x_j) = \textit{distância de Hamming entre os blocos } x_i \textit{ e } x_j.$$

Exemplo 17 *Como exemplo, o perfil de distância para o código de Miller é mostrado na tabela 5.2.*

	0	1
0	0	1
1	1	0

(a)

	10	01
10	0	2
01	2	0

(b)

	00	01
00	0	1
01	1	0

(c)

Tabela 5.2: Perfil distância para o código de Miller

(a) *Perfil distância de entrada*

(b) *Perfil distância de saída para o estado 1*

(c) *Perfil distância de saída para o estado 2*

Notemos que a matriz é simétrica, pois $D(x_i, x_j) = D(x_j, x_i)$ *para todos os valores de* i *e* j. *Observamos também, que a tabela 5.2(a) é o perfil distância de entrada para o código de Miller, pois as entradas no codificador em qualquer estado são 0 ou 1. Similarmente, o perfil distância de saída para o estado 1 do código de Miller é mostrado na tabela 5.2(b). Esta matriz compara as duas seqüências 10 e 01, que são saídas do codificador no estado 1, para as entradas 0 e 1, respectivamente. Finalmente, o perfil distância de*

saída para o estado 2 do código de Miller é mostrado na tabela 5.2(c). Este perfil distância compara as seqüências 00 e 01 correspondentes a saída do codificador no estado 2.

Tendo definido perfil distância de entrada e saída , podemos agora descrever o que significa um código preservando distância.

Um código preserva distância quando o perfil distância da seqüência de saída é pelo menos tão grande quanto o correspondente perfil distância da seqüência de entrada. Para o código de Miller, isto significa que a distância entre 10 e 01, e 00 e 01 deve ser pelo menos tão grande quanto a distância entre 0 e 1. Estas exigências podem ser verificadas pela inspeção do perfil distância da tabela 5.2.

5.3.3 Geração de Códigos Preservando Distância

Descreveremos alguns códigos simples que podem ser obtidos através da adição de bits de enchimento em seqüências binárias arbitrárias.

Exemplo 18 *Considere o codificador da tabela 5.3. Este código foi gerado, partindo-se de todas as quatro possíveis seqüências binárias de dois bits {00, 01, 10, 11}. Criamos um código* $(d,k) = (0,2)$ *de taxa 2/3 pela inserção do bit 1 após os dois bits das seqüências. Assim, 00 é convertido em 001, 01 em 011, 10 em 101 e 11 em 111. Claramente, a distância entre as seqüências não foi mudada pela inserção de um bit, no caso o bit 1, em uma posição fixa em cada seqüência. Isto é evidente no perfil distância de entrada e saída mostrados na tabela 5.4. Também é fácil verificar que a saída do codificador satisfaz a restrição* $(0,2)$.

Este método pode ser generalizado para construir um código $(0,k)$ *preservando distância de taxa* $k/(k+1)$. *Isto é, nós partimos com todas as possíveis seqüências binárias de k bits, e inserimos o bit 1 no final de cada saída. O ganho de densidade de armazenamento para esta classe vale* $\mathcal{G} = \frac{k}{k+1}$.

Entrada	Saída
00	001
01	011
10	101
11	111

Tabela 5.3: Codificador para o código $(0,2)$ de taxa $2/3$

	00	01	10	11
00	0	1	1	2
01	1	0	2	1
10	1	2	0	1
11	2	1	1	0

(a)

	001	011	101	111
001	0	1	1	2
011	1	0	2	1
101	1	2	0	1
111	2	1	1	0

(b)

Tabela 5.4: Perfil distância para o código $(0,2)$ de taxa $2/3$
(a) *Perfil distância de entrada*
(b) *Perfil distância de saída*

Exemplo 19 *Como um segundo exemplo de um código simples, considere o código* $(1,5)$ *que preserva distância, de taxa 2/4, mostrado na tabela 5.5. As seqüências de saída deste código foram obtidas da seqüência de entrada em duas etapas. Primeiro, inserimos um bit 0 após cada bit de entrada. Isto não muda as propriedades de distância das seqüências, mas nos fornece uma restrição* $(1,\infty)$. *Segundo, substituímos a seqüência 0000 por 0100, assim obtivemos a restrição* $(1,5)$. *Note que não diminuímos a distância entre as seqüências. O perfil distância para este código é mostrado na tabela 5.6.*

Entrada	Saída
00	0100
01	0010
10	1000
11	1010

Tabela 5.5: Codificador para o código $(1,5)$ de taxa $2/4$

	00	01	10	11
00	0	1	1	2
01	1	0	2	1
10	1	2	0	1
11	2	1	1	0

(*a*)

	0100	0010	1000	1010
0100	0	2	2	3
0010	2	0	2	1
1000	2	2	0	1
1010	3	1	1	0

(*b*)

Tabela 5.6: Perfil distância para o código $(1,5)$ de taxa $2/4$

(*a*) *Perfil distância de entrada*

(*b*) *Perfil distância de saída*

O código da tabela 5.5 pode ser generalizado em uma família de códigos $(d, 2d+3)$ que converte seqüências de entrada de 2 bits em seqüências de saída de $(2d+2)$ bits, para $d \geq 1$, adicionando d zeros após cada bit de entrada. A palavra $000\ldots0$ deve ser substituída pela palavra $010\ldots0$. A taxa do código neste caso é $2/(2d+2) = 1/(d+1)$, e o perfil distância é o mesmo para todos os valores de d. O ganho de densidade de armazenamento para esta família vale $\mathcal{G} = 1$.

Através dos códigos anteriores, observamos que um grande número de códigos *RLL* preservando distância podem ser criados, pela inserção de $1s$ e $0s$, ou ambos, entre os bits da seqüência de entrada.

5.3.4 Construção de Códigos *ECC/RLL* Cascateados

Considere um código *RLL* de taxa R_{RLL} fazendo parte de um esquema em cascata com um código corretor de erro.

O código resultante terá uma taxa igual a $R = R_{ECC}.R_{RLL}$, onde R_{RLL} é a taxa do mapeamento *RLL*, e distância livre igual a d_{free}, isto é, mesma distância livre do código convolucional sozinho, desde que o mapeamento *RLL* preserve distância. Como um exemplo, considere a treliça do código convolucional de $d_{free} = 5$ e taxa $1/2$ mostrada na figura 5.4. Cada ramo da treliça tem classificação da forma A/BC, onde A é a entrada do codificador e BC é a saída. Suponha que as seqüências de saída deste codificador convolucional foram cascateadas com o codificador $(1,5)$ que preserva distância da tabela 5.5.

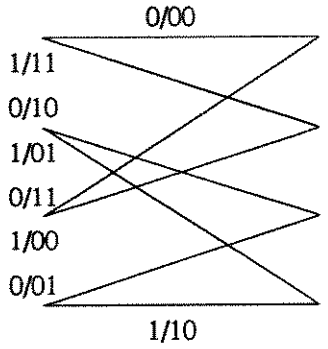

Figura 5.4: Representação em treliça do código convolucional de taxa $1/2$ e $d_{free} = 5$

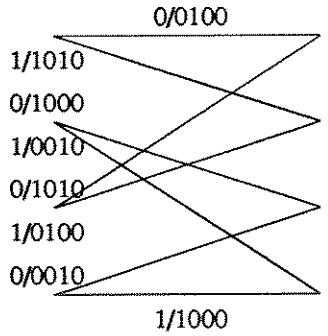

Figura 5.5: Representação em treliça do código de taxa $1/4$ e $d_{free} = 5$

A treliça resultante *ECC/RLL* é mostrada na figura 5.5. Este novo código tem uma taxa total $1/4$, $d_{free} = 5$ e satisfaz a restrição $(1, 5)$. A decodificação para este código pode ser realizada utilizando-se o algoritmo de Viterbi.

Outros códigos cascateados com maior d_{free} podem ser gerados se usarmos um código convolucional com maior memória. Por exemplo, a tabela 5.7 lista a distância livre e o número de estados de uma série de códigos convolucionais de taxa $1/2$, que poderiam ser cascateados com qualquer um dos códigos *RLL* $(d, 2d + 3)$ de taxa $1/(d + 1)$ da tabela 5.2 para criar um novo código com restrição $(d, 2d + 3)$ e taxa $1/(2d + 2)$. Também incluído na tabela 5.7 estão as distâncias livre e o número de estados para uma série de códigos convolucionais de taxa $1/3$, que poderiam ser usadas com um código $(2, 7)$ de taxa $3/8$ para criar um código *ECC/RLL* de taxa $1/8$ com qualquer uma das distâncias desta tabela. Neste caso, o número de estados no codificador será duas vezes o número de estado dado na tabela 5.7, já que o codificador $(2, 7)$ tem dois estados.

R	d_{free}	$n^{\underline{o}}$ de estados	R	d_{free}	$n^{\underline{o}}$ de estados
1/2	5	4	1/3	8	4
1/2	6	8	1/3	10	8
1/2	7	16	1/3	12	16
1/2	8	32	1/3	13	32
1/2	10	64	1/3	15	64

Tabela 5.7: Alguns códigos convolucionais de taxa 1/2 e taxa 1/3

5.4 Decodificação de Viterbi de Códigos *ECC/RLL* Combinados

5.4.1 Decodificação de Máxima Verossimilhança

Um decodificador de máxima verossimilhança escolhe $\vec{v}$ como seqüência decodificada, a partir da seqüência recebida $\vec{r}$, tal que se maximize a função de verossimilhança [8].

Para um canal sem memória, temos

$$P(\vec{r}|\vec{v}) = \prod_{i=0}^{N} P(r_i|v_i) \tag{5.3}$$

onde N é o comprimento da seqüência.

Maximizar $P(\vec{r}|\vec{v})$ é equivalente quando as seqüências v forem equiprováveis, a minimizarmos a probabilidade de erro.

$$P(E) = \sum_{\vec{r}} P(E|\vec{r})P(\vec{r})$$

onde $P(E|\vec{r}) = P(\vec{v} \neq \vec{v}|\vec{r})$.

Tirando o logaritmo de ambos os lados da equação 5.3, temos que a produtória se transforma em somatória e chamando o logaritmo da probabilidade de métrica, temos que

$$M(\vec{r}|\vec{v}) = \sum_{i=0}^{N} M(r_i|v_i)$$

onde $M(r_i|v_i) = \log P(r_i|v_i)$.

Como o algoritmo de Viterbi decodifica o caminho da treliça que tem maior métrica, ele é dito ser de máxima verossimilhança.

Para um canal binário simétrico em que a probabilidade de acertar é $(1-p)$ e a de errar é p

$$P(r_i|v_i) = p, \quad r_i \neq v_i$$

$$P(r_i|v_i) = 1-p, \quad r_i = v_i$$

Dada uma seqüência recebida $\vec{r}$, que descorda com a seqüência transmitida $\vec{v}$ em d posições, então

$$P(\vec{r}|\vec{v}) = p^d(1-p)^{N-d}$$

$$M(\vec{r}|\vec{v}) = d\log(\frac{p}{1-p}) + N\log(1-p)$$

onde, p^d é a probabilidade de erro em d posições, e $(1-p)^{N-d}$ é a probabilidade de acerto em $N-d$ posições.

Como $N\log(1-p)$ é constante e $\log(\frac{p}{1-p}) \leq 0$, para $p < 1/2$, então, neste caso, maximizar a métrica é equivalente a escolhermos na treliça a seqüência que possue menor distância de Hamming em relação a seqüência recebida.

5.4.2 Decodificação de Viterbi de Códigos *ECC/RLL* Combinados

Como um primeiro exemplo, mostraremos que a decodificação de Viterbi pode ser empregada em códigos RLL, principalmente nos esquemas combinados.

Exemplo 20 *Seja a treliça do código de Miller mostrada na figura 5.6. Suponhamos a seqüência de entrada* $\vec{u} = (0,1,0,1,0)$. *Então, a seqüência de saída é* $\vec{v} = (10,01,00,01,00)$. *Este código tem* $d_{free} = 1$, *portanto não é poderoso quanto à capacidade de correção de erro.*

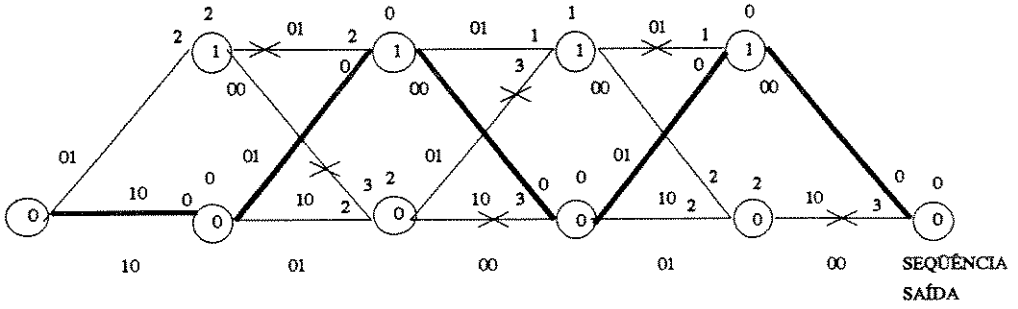

Figura 5.6: Representação em treliça do código de Miller

Os números no final de cada ramo representam o pêso de Hamming da seqüência, isto é, a distância entre a seqüência de saída e a seqüência transmitida. Os ramos em negrito representam o caminho que foi decodificado.

Como podemos notar pela figura 5.6, obtivemos exatamente a seqüência de entrada através da decodificação de Viterbi, já que foi suposto a ocorrência de nenhum erro. Caso tivesse ocorrido erro em determinadas posições a recuperação de informação seria comprometida.

Exemplo 21 *Agora, seja a treliça do código da figura 5.5, mostrada na figura 5.7. Este código tem* $d_{free} = 5$.

Suponhamos que $\vec{u} = (0,0,1,1,1)$. *Então,* $\vec{v} = (0010, 1010, 1010, 0010, 1000)$. *Neste caso, seja a palavra recebida* $\vec{r} = (0010, 1010, 1000, 0010, 1000)$, *isto é, ocorre um erro na terceira palavra de saída, de 1010 para 1000. Observando a treliça, notamos que através da decodificação de Viterbi obtivemos a palavra de entrada, apesar de ter ocorrido 1 erro.*

5.5 Conclusão

Neste capítulo analisamos o sistema de armazenamento de dados na presença de ruído. Sendo assim, a fim de tornarmos o sistema mais confiável empregamos um código corretor de erro.

Apresentamos uma classe de códigos *ECC/RLL* combinados, em que as palavras-código além de satisfazer as restrições (d,k), possuem propriedades de correção de erro.

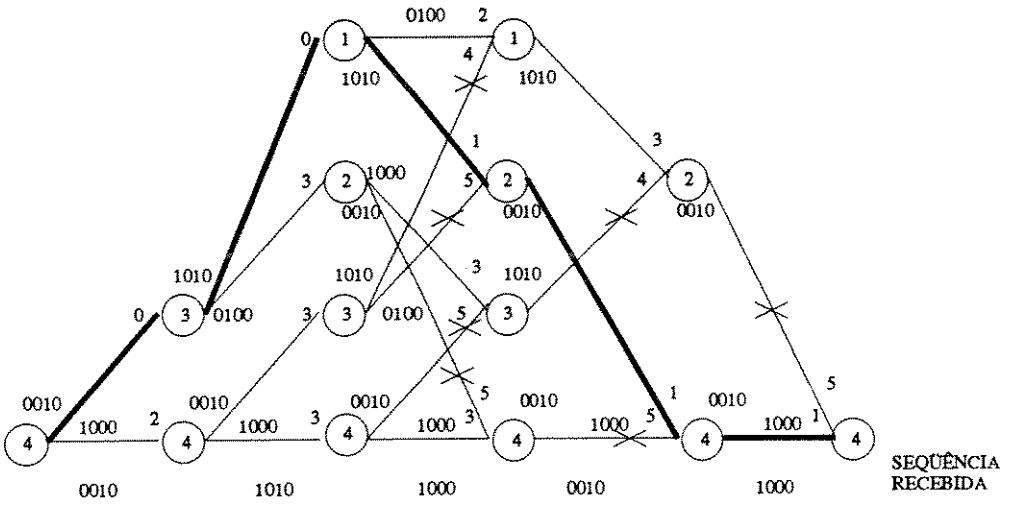

Figura 5.7: Representação em treliça do código de quatro estados e $d_{free} = 5$

Entretanto, os códigos combinados mostrados neste trabalho não possuem bom desempenho.

Mostramos também, uma classe de códigos *RLL* que preserva distância, ou seja, mantém as propriedades de distância de Hamming do código corretor de erro.

Para finalizar a tese, procedemos a decodificação de Viterbi do código de Miller na ausência de ruído, e de um código combinado *ECC/RLL* na presença de ruído.

Capítulo 6

Conclusão

6.1 Comentários Finais e Conclusões

Vimos a importância do uso dos códigos limitados por comprimento serial em sistemas de gravação magnética, pois além de proporcionar um aumento da densidade de armazenamento de dados, minimizam o problema da interferência inter-simbólica, facilitam a recuperação do sincronismo, e produzem uma conformação do espectro do sinal.

Apresentamos definições de capacidade de canal sem ruído e ganho de densidade de armazenamento. Dois pontos de vista para obtenção do valor da capacidade de canal foram mostrados. O primeiro, baseado na construção da treliça do número de palavras-código que satisfazem ao par de restrição (d, k). Verificamos que o logaritmo na base 2 da razão da série associada à treliça nos leva a capacidade de canal. Já no segundo método, a capacidade é calculada através da matriz de transição associada a um diagrama de transição de estado finito, que emite seqüências de restrição (d, k). O valor da capacidade de canal é o logaritmo na base 2 do maior auto-valor positivo desta matriz. Deste modo, obtivemos os valores da capacidade de canal, ganho de densidade de armazenamento e a eficiência de alguns códigos (d, k).

Na apresentação das técnicas de codificação e decodificação de dados, vimos que a técnica de comprimento fixo busca construir de forma simples um código de bloco de taxa m/n, que satisfaça as restrições do canal, não se preocupando em obter eficiências muito boas, mas sim uma implementação simples. Os códigos de comprimento fixo podem

se tornar impraticáveis devido ao grande valor do comprimento de bloco necessário para se obter boas eficiências.

Vimos que a codificação enumerativa nos fornece um modo algébrico de codificação e decodificação das palavras-código, com isso, evitamos o problema da procura excessiva em tabelas quando as palavras-código se tornam muito longas, e se apresentam em grandes quantidades.

A técnica de comprimento variável reduz a complexidade dos codificadores e decodificadores em relação a códigos de tamanho de bloco fixo. Além disso, oferecem a possibilidade de usarmos palavras de comprimento pequeno mais freqüentemente do que as de comprimento longo, dando origem a códigos eficientes.

Observamos que através das técnicas de codificação e decodificação de antecipativa e de blocos deslizantes podemos obter boas eficiências, ou seja, obtemos maior ganho de densidade de armazenamento, pois $\mathcal{G} = (d+1)R$. Contudo, a decodificação de dados torna-se mais complexa se quisermos aumentar a eficiência, pois teremos que olhar algumas janelas à frente.

A diferença entre as técnicas de antecipação e de blocos deslizantes está apenas no rigor matemático oferecido pela segunda, pois ambas possuem o mesmo princípio de construção. Vimos também, que o algoritmo de blocos deslizantes oferece uma sólida estrutura teórica para a construção de códigos de gravação de dados.

Nos primeiros capítulos o sistema de gravação de dados foi considerado ideal, ou seja, na ausência de ruído. Mas, como na verdade não possuímos sistemas assim, apresentamos no capítulo 5, a codificação e decodificação de dados na presença de ruído. Mencionamos a existência de classes de códigos *ECC/RLL* combinados, em que as palavras-código não apenas satisfazem as restrições d e k, mas também possuem capacidade de correção de erro. Estes códigos *ECC/RLL* combinados são representados em treliça, e possuem uma fácil implementação tanto para a codificação quanto para a decodificação.

Os códigos combinados mostrados neste trabalho não apresentam bom desempenho.

Apresentamos uma classe de códigos *RLL*, o código *RLL* que preserva distância, que mantém as propriedades de distância de Hamming do código corretor de erro. Pois,

quando um código corretor de erro é usado em cascata com um código RLL, o segundo pode diminuir a eficiência do primeiro.

6.2 Contribuições

No capítulo 3 o método para obtenção da capacidade de canal, através da razão da série do número de palavras-código em função do tamanho do bloco, foi desenvolvido independentemente por nós. Até então, não conhecíamos que constava das referências bibliográficas. A posteriori, quando do término do desenvolvimento do método, descobriu-se no artigo *Run length-Limited Sequences* publicado por K. A. S. Immink, em novembro de 1990, o mesmo método. Entretanto, este fato reforçou a consistência dos resultados obtidos.

Gostaríamos de enfatizar a construção de alguns códigos não disponíveis na literatura.

- Códigos de comprimento fixo com $k = \infty$: tabelas 4.2, 4.3, 4.4 e 4.5. Obtidos sem auxílio computacional.
- Códigos de comprimento fixo com k finito: tabela 4.7. Obtidos sem auxílio computacional.
- Códigos de comprimento variável (apêndice c). Obtidos com o auxílio do programa que consta no apêndice a.
- Códigos look-ahead (apêndice b). Obtidos sem auxílio computacional.

6.3 Trabalhos Futuros

Como temas de trabalhos futuros propomos as seguintes linhas de pesquisa:

- Estudos de esquemas de códigos corretores de erro e códigos de linha combinados.
- Estudos de novos código de linha que preservem a distância.
- Simulação de sistemas de armazenamento de dados usando modelos de canal de gravação.

- Estudos de novos código de linha que preservem a distância.
- Simulação de sistemas de armazenamento de dados usando modelos de canal de gravação.
- Obtenção de códigos de linha com pouco conteúdo espectral em baixas freqüências.

Apêndice a

Programa Utilizado

a.1 Programa para o cálculo das palavras-código que satisfazem uma certa restrição (d, k)

```
C LEITURA DE DADOS

*
DIMENSION Z(20),Z1(20)
INTEGER M, Z, E, D, K ,L ,N ,P ,CONT1, CONT2, X, L1, R, S, Y,VH,Z1
*
READ(*,*)  M, D, K
*
*
        OPEN(UNIT=16,FILE='fort1.dad',STATUS='unknown')
        OPEN(UNIT=17,FILE='fort2.dad',STATUS='unknown')
*
C CALCULO DAS PALAVRAS-CODIGO
*
CONT1=0
CONT2=0
E=2**M-1
```

```
      DO 20 I=0, E
        Y=I
    DO 30 J=1, M
         Z(J)=MOD(Y,2)
            Y= INT(Y/2)
  30              CONTINUE
*
C ELIMINACAO DAS PALAVRAS-CODIGO QUE NAO SATISFAZEM A RESTRICAO (D,K)
*
*
Y=0
X=0
N=0
P=0
L1=0
    DO 40 L=1, M
             IF (Z(L).EQ.1) THEN
     Y=Y+1
 IF (L.GE.2) THEN
      L1=L-1
              IF (Z(L).EQ.Z(L1)) THEN
          X=0
      END IF
          END IF
                          IF (Y.EQ.2) THEN
              IF (X.GE.D) THEN
          IF (X.LE.K) THEN
              N=1
      Y=Y-1
      X=0
          ELSE
     N=0
     Y=Y-1
```

```
         END IF
      ELSE
         N=0
         Y=Y-1
                              END IF
             IF (N.EQ.0) THEN
          GO TO 20
      END IF
         END IF
     ELSE IF (Y.EQ.1) THEN
        X=X+1
             ELSE
        P=P+1
     END IF
  40 CONTINUE
IF (Y.EQ.1) THEN
    IF (X.LE.K) THEN
IF (P.LE.K) THEN
                              WRITE (16,*) (Z(N),N=1,M)
    CONT1=CONT1+1
END IF
    END IF
END IF
IF (P.EQ.M) THEN
    IF (P.LE.K) THEN
                WRITE (16,*) (Z(N),N=1,M)
                        CONT1=CONT1+1
            END IF
END IF
IF (Y.EQ.1) THEN
    IF (X.GT.0) THEN
IF (N.EQ.1) THEN
                          WRITE (16,*) (Z(N),N=1,M)
```

```
                CONT1=CONT1+1
END IF
    END IF
END IF
  20            CONTINUE
*
C ELIMINAR AS PALAVRAS QUE TERMINAM COM ATE (D-1) ZEROS
CLOSE(16)
OPEN(UNIT=16,FILE='fort1.dad',STATUS='OLD')
        DO 60 VV=1,CONT1
   READ(16,*) (Z1(VH),VH=1,M)
           R=0
   K=M
   DO 50 S=1, D
      IF (Z1(K).EQ.0) THEN
          K=K-1
      ELSE
          R=1
      END IF
   50    CONTINUE
   IF (R.EQ.0) THEN
       WRITE (17,*) (Z1(VH),VH=1,M)
       CONT2=CONT2+1
     END IF
   60           CONTINUE
```

Apêndice b

Códigos Antecipativos

Neste apêndice apresentamos as tabelas de codificação dos códigos antecipativos mostrados na tabela 4.20. Estes códigos foram obtidos sem o auxílio computacional.

Dados	Estado A	Estado B	Estado C	Estado D	Estado V
00	00001/A	00010/A	00100/A	01000/A	00000/A
01	00001/B	00010/B	00100/B	01000/B	00000/B
10	00001/C	00010/C	00100/C	01000/C	00000/C
11	00001/V	00010/D	00100/D	01000/D	00000/D

Tabela b.1: Tabela do código $(2, 9)$ de taxa $R = 2/5$

Dados	Estado A	Estado B	Estado C	Estado D	Estado V
00	01000/A	00001/V	00010/V	00100/A	00000/A
01	01000/B	00001/B	00010/B	00100/B	00000/B
10	01000/C	00001/C	00010/C	00100/C	00000/C
11	01000/D	00000/V	00010/D	00100/D	00000/D

Tabela b.2: Tabela do código $(3, \infty)$ de taxa $R = 2/5$

Apêndice c

Códigos de Comprimento Variável

A seguir mostraremos os códigos de comprimento variável, obtidos com o auxílio do programa que consta no apêndice *a*.

palavras da fonte	*palavras-código*
00	000
0100	000010
0101	001010
0110	101010
0111	010010
1000	100010
1001	000100
1010	010100
1011	100100
1100	001000
1101	101000
1110	010000
1111	100000

Tabela c.1: Tabela do código $(1, \infty)$

palavras da fonte	*palavras-código*
00	00000
0100	1000000000
0101	0100000000
0110	0010000000
0111	0001000000
1000	0000100000
1001	1000100000
1010	0000010000
1011	1000010000
1100	0100010000
1101	0000001000
1110	1000001000
1111	0100001000

Tabela c.2: Tabela do código $(3, \infty)$

palavras da fonte	*palavras-código*
0	000
10	100000
11	010000

Tabela c.3: Tabela do código $(4, \infty)$

palavras da fonte	*palavras-código*
0	000
100	100000000
101	010000000
110	001000000
111	000100000

Tabela c.4: Tabela do código $(5, \infty)$

BIBLIOGRAFIA

[1] **P. H. Siegel and J. K. Wolf:** *Modulation and Coding for Information Storage*, IEEE Communications Magazine, pp.68-86, December 1991.

[2] **E. Tan and B. Vermeulen:** *Digital audio tape for data storage*, IEEE Spectrum, pp.34-38, October1989.

[3] **J. M. Cioffi, W. L. Abbott, H. K. Thapar, C. M. Melas and K. D. Fisher:** *Adaptive Equalization in Magnetic-Disk Storage Channels*, IEEE Communications Magazine, pp.14-27, February 1990.

[4] **K. A. S. Immink:** *Runlength-Limited Sequences*, Proceedings of the IEEE, vol. 78, n^{o}11, pp.1745-1759, November 1990.

[5] **P. H. Siegel:** *Recording Codes for Digital Magnetic Storage*, IEEE Transactions on Magnetics, vol. MAG-21, n^{o}5, pp.1344-1349, September 1985.

[6] **M. Mattavelli:** *Low-Frequency Suppression in RLL Codes for Optical Recording*, Philips Research Laboratories, The Netherlands, pp.109-115.

[7] **S. Haykin:** *Digital Communications*, Editora Wiley, 1988.

[8] **S. Lin and D. J. Costello, Jr.:** *Error Control Coding: Fundamentals and Applications*, cap.11, pp.315-349.

[9] **Y. Lin and J. K. Wolf:** *Combined ECC/RLL Codes*, IEEE Transactions on Magnetics, vol.24, n^{o}6, pp.2527-2529, November 1988.

[10] **C. A. French:** *Distance Preserving Run-Length Limited Codes*, IEEE Transactions on Magnetics, vol. 25, n^{o}5, pp.4093-4095, September 1989.