UNIVERSIDADE ESTADUAL DE CAMPINAS

FACULDADE DE ENGENHARIA ELÉTRICA

DEPARTAMENTO DE COMUNICAÇÕES

SOFTWARE PARA A GERAÇÃO DE CÓDIGOS RLL

EMPREGANDO O ALGORITMO DOS BLOCOS DESLIZANTES

Rossini Trindade Costa

Orientador
Prof. Dr. Celso de Almeida

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

Campinas, junho de 1994

AGRADECIMENTOS

- Ao Prof. Celso de Almeida pela orientação e por ter acreditado em nosso trabalho.
- Ao Prof. Hélio Waldman pelo incentivo, pelo auxílio e pelo interesse nessa pesquisa.
- Ao amigo Carlos Pingarilho pelas discussões e pelo interesse em utilizar nosso programa.
- À minha mãe e ao meu irmão pelo suporte e pelo incentivo ao longo de toda a minha vida.
- À Cristina pela inestimável ajuda e pela companhia.
- Aos amigos Andrade, Dalila, Mauro, Carmelo, Aleandro, Fançony e Leila pela troca de idéias e pelo companheirismo durante o mestrado.
- Aos demais colegas os quais tive o prazer de conhecer nesses últimos anos.
- À CAPES, pelo apoio financeiro.

Em memória de meu pai

Conteúdo

SUMÁRIO

Tendo sido desenvolvido num ramo da matemática conhecido como Dinâmica Simbólica, o algoritmo dos Blocos Deslizantes, constitui-se num procedimento sistemático e eficiente na busca por códigos com as características desejáveis.

Será apresentada neste trabalho, uma discussão completa e com exemplos do algoritmo, sem entrar nos detalhes matemáticos que lhe são inerentes. Serão enfatizadas as técnicas componentes deste (divisão e fusão de estados, além do algoritmo do autovetor aproximado) e sua implementação via um software desenvolvido pelo autor.

Como forma de validação da ferramenta, o programa foi adaptado para gerar códigos RLL, que são códigos cujo objetivo principal é aumentar a densidade de armazenamento de dados em meios magnéticos e óticos.

Diversos códigos obtidos serão apresentados. A análise destes, considerando parâmetros como complexidade do codificador e decodificador, mostrou que os resultados encontram-se próximos dos limitantes permitidos comprovando assim a funcionalidade do software.

Um detalhe importante é que apesar do programa ter sido implementado para códigos RLL, ele pode ser facilmente adaptado para incorporar outros tipos de códigos restritos a sistemas de estados finitos bastando, para isso, pequenas alterações em algumas rotinas do programa.

ABSTRACT

Sliding Block algorithm was developed in a branch of mathematics known as Symbolic Dynamics and it consists of a systematic and efficient procedure to find codes with desirable characteristics.

This work will present a complete discussion of this algorithm, without emphasis on the rigorous mathematical details that are inherent in it.

The techniques that constitute this algorithm will also be emphasized (state merging and splitting) and its implementation through a software developed by the author.

The software was adapted to generate RLL codes, which are codes with the objective to increase data storage density in magnetic and optical media.

Several codes will be presented. Analysis of them, considering parameters like encoder and decoder complexity, and the number of states show that the results were found near to the available bounds showing the efficiency of the software.

It's important to remark that, though the software had been implemented to find RLL codes, it can be easily adapted to obtain other kinds of codes, with small changes in some routines of the software.

TABELA DE SÍMBOLOS

- p: Tamanho do bloco de bits de informação
- q: Tamanho do bloco de bits de codificação
- d: Número mínimo de 0s entre 1s consecutivos numa sequência dk
- k: Número máximo de 0s entre 1s consecutivos numa sequência dk
- $N(n)$: Número de sequências dk de um certo comprimento n
- A: Matriz de transição inicial
- S: Sistema Restrito
- C: Capacidade de uma sequência dk
- λ: Maior autovalor real da matriz de Transições A
- n: Comprimento da sequência dk
- GD: Ganho em densidade de armazenamento
- R: Taxa do código (p/q)
- η: Eficiência de codificação
- a: Antecipação do codificador
- m: Memória do codificador
- $\vec{v}$: Autovetor aproximado
- L: Constante a ser empregada como valor inicial do autovetor
- v_i, y_i: Peso do estado i
- ϵ_k: Ramo k, proveniente de um estado qualquer
- $w(\epsilon_i)$: Peso do ramo ϵ, ou peso do estado terminal de ϵ
- E_i: Conjunto de ramos que saem do estado i
- E_i^j: Conjunto dos ramos que saem do estado j provenientes da divisão do estado i (bifurcação do estado i)
- M: Número de ramos que partem de um estado qualquer
- σ: Número de estados

- SS: Número de divisões de estado (*state splitting*)
- s/t: Formato do rótulo atribuído a um ramo qualquer, sendo s o rótulo do bloco de entrada e t o rótulo do bloco de saída
- J: Tamanho da janela de decodificação
- $F(i)$: Conjunto das sequências de ramos geradas a partir do estado i
- C_{cod}: Complexidade de codificação
- C_{dec}: Complexidade de decodificação
- C_T: Complexidade Total

Capítulo 1

INTRODUÇÃO

Códigos corretores de erros são usados em sistemas de comunicação, quando deseja-se transmitir informação por um dado canal ruidoso visando um aumento na confiabilidade dos dados recebidos.

Um outro tipo de codificação muito empregada nos dias de hoje é conhecida como codificação de linha. Este tipo de codificação, nos sistemas de comunicação, tem por característica principal a adequação do espectro do sinal ao do canal. Possui também um largo emprego nos sistemas de armazenamento de dados, como Compact Disks (CDs), winchesters e floppy-discs. As finalidades, neste caso, são diversas, entre outras pode-se citar: aumentar a densidade de armazenamento de dados em meios magnéticos ou ópticos, reduzir a ocorrência de interferência intersimbólica (ISI) e eliminar as componentes de baixa frequência, de modo a permitir o envio de sinais de controle nesta faixa do espectro.

A figura 1.1 mostra o modelo de um sistema de gravação. Uma fonte gera os dados na forma de símbolos binários. Sobre esses símbolos é introduzida redundância, ou codificação corretora, transformando um bloco de m bits numa sequência de p bits. Esses p bits são então novamente codificados por um codificador de gravação (linha), o qual converterá o bloco de p bits em uma nova sequência de q bits, onde o quociente p/q recebe o nome de taxa do código de gravação. Os dados estão agora prontos para serem escritos no meio magnético, caracterizado pelo canal na figura. O processo de leitura consiste no inverso, os dados são decodificados pelos decodificadores de gravação e corretor de erros, estando, logo após, disponíveis para o usuário.

Dentre os códigos de gravação mais empregados, destacamos aqueles denominados RLL (*Runlength Limited Codes*). Os códigos RLL, também chamados de (dk), são caracterizados por dois parâmetros d e k, os quais delimitam os números mínimo e máximo de zeros entre uns consecutivos numa sequência de informação qualquer. O parâmetro d controla o nível de interferência intersimbólica a ser produzida pelo meio de gravação. Este parâmetro também relaciona-se diretamente com a densi-

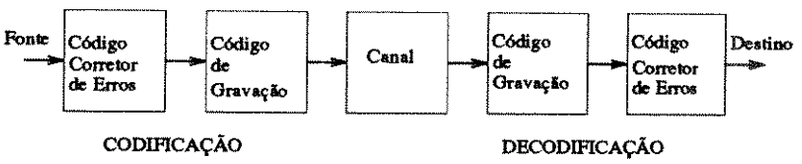


Figura 1.1: Modelo do Sistema de Gravação Magnética

dade de gravação, conforme será visto nos próximos Capítulos. O parametro k tem importância na escolha do circuito de recuperação de relógio na decodificação.

Destacam-se entre os códigos RLL mais conhecidos, o MFM com taxa $1/2$, $d = 1$ e $k = 3$, largamente empregado na tecnologia de Winchesters [1]. Mais recentemente destacamos o código com taxa $1/2$ e $(d, k) = (2, 7)$, utilizado no disco rígido do IBM 3370/80 e o EFM de taxa $8/17$, (2,10) empregado em CDs [2][3].

Existem diversos procedimentos empregados na obtenção desses códigos. Algoritmos como o *look-ahead*, desenvolvido por diversos autores, além de técnicas para obtenção de códigos de comprimento variável, cuja base foi formulada por Franaszek, merecem ser citados [1][4][5][6].

Nos últimos anos uma nova técnica, bastante eficiente, foi proposta e destacou-se, principalmente, pela sua forte e rigorosa base matemática: o Algoritmo de Blocos Deslizantes ou *Sliding-Block* . Seu nome reside no fato de que suas palavras-código resultantes podem ser decodificadas usando uma janela de comprimento finito. O comprimento dessa janela varia de acordo com as opções feitas durante o processo de geração do código. Em geral, visando-se uma menor complexidade do decodificador e uma menor propagação de erros de decodificação, as janelas devem ter o menor comprimento possível. Tendo sido desenvolvida num campo da matemática abstrata, também conhecida como Dinâmica Simbólica, por Adler, Coppersmith e Hassner, esta técnica foi rapidamente absorvida e empregada como um procedimento sistemático na busca por códigos RLL [1][7][8].

Este trabalho dedica-se a descrever o procedimento de construção de códigos RLL a partir do Algoritmo de Blocos Deslizantes. O procedimento foi implementado via software, tornando automatizada, a busca por esses códigos. No final deste trabalho serão apresentados alguns códigos obtidos através do software desenvolvido pelo autor.

O Capítulo 2 deste trabalho tem por finalidade situar o leitor no tema, apresentando uma visão geral dos processos envolvendo gravação magnética. Serão abordados os processos de gravação e leitura de dados com ênfase no detector de

pico. Serão introduzidos também os aspectos básicos dos códigos RLL, suas características principais. Parâmetros como capacidade, eficiência e ganho em densidade serão brevemente discutidos.

No Capítulo 3, um aprofundamento no estudo dos códigos RLL enfatizando a construção destes via Algoritmo dos Blocos Deslizantes será realizado. Técnicas como divisão e fusão de estados serão descritas, sendo também introduzido o conceito de autovetor aproximado. Essas técnicas serão utilizadas em conjunto para a obtenção dos códigos RLL. No final do Capítulo serão apresentados alguns exemplos que ajudarão a compreender melhor o funcionamento deste algoritmo.

No Capítulo 4 será descrito, detalhadamente, o software implementado para a geração dos códigos RLL. Este software aglutina os conceitos e métodos descritos nos Capítulos anteriores para formar uma eficiente ferramenta para a obtenção de códigos, dentre os quais os RLL. Será descrito, inicialmente, o funcionamento do software, a interação homem-máquina, e a seguir o programa principal juntamente com cada subrotina dele componente.

O Capítulo 5 dedicar-se-á à validação da ferramenta desenvolvida. Neste Capítulo serão apresentados e analisados diversos códigos RLL, com diversas restrições e taxas, produzidos pelo software. A análise levará em conta parâmetros como janela de decodificação, complexidade do codificador e densidade de armazenamento.

Por fim, no Capítulo 6 estão contidos as conclusões e as considerações finais acerca do que foi feito, enfatizando também as propostas futuras visando uma continuidade deste trabalho.

Capítulo 2

GRAVAÇÃO MAGNÉTICA: ASPECTOS GERAIS

2.1 EVOLUÇÃO E TENDÊNCIAS EM GRAVAÇÃO MAGNÉTICA

A indústria de gravação magnética, nos últimos anos experimentou avanços consideráveis, movimentando hoje um mercado acima dos US$ 50 bilhões, estimando-se em US$ 100 bilhões para 1995.

Este fato exigiu que consideráveis avanços fossem obtidos nos sistemas de gravação de dados, permitindo o desenvolvimento de dispositivos que armazenassem mais informação em espaços cada vez menores com uma confiabilidade sempre crescente, e que possuíssem um preço reduzido.

Dentro de um sistema de gravação, as maiores contribuições a esses avanços foram dadas no desenvolvimento de novas cabeças de gravação, em mecânica fina (servomecanismos), em melhorias no canal (meio magnético), além de melhorias no processamento digital de sinais.

Tratando-se do processamento digital de sinais, merece destaque a área de codificação, com o desenvolvimento de novos e cada vez mais eficientes códigos de gravação.

No gráfico da Figura 2.1, é mostrada a evolução em densidade superficial do disco rígido da IBM com o passar dos anos. Através dele, pode-se verificar que o avanço em densidade superficial, nos últimos 25 anos vem crescendo a um fator de 10 vêzes por década [9].

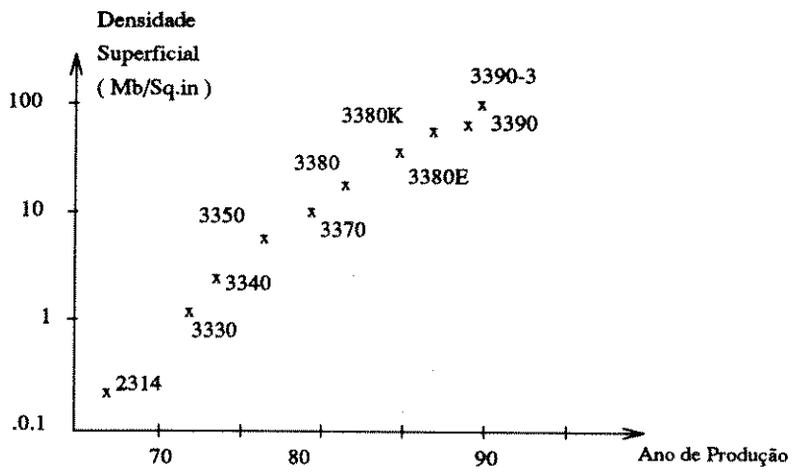


Figura 2.1: Evolução nos sistemas de gravação da IBM com o passar dos anos

2.2 OS PROCESSOS DE LEITURA E GRAVAÇÃO DE DADOS

A maioria dos dispositivos de armazenamento em uso nos dias de hoje empregam gravação por saturação para o processo de escrita, e detecção de pico, para o processo de leitura.

2.2.1 O Processo de Gravação

A Figura 1.1 ilustra o diagrama em blocos de um sistema de gravação de dados abrangendo escrita e leitura. Através dele, pode-se observar que, durante o processo de escrita, os dados do usuário passam inicialmente por um codificador para a correção de erros (ECC), de modo a proteger o sinal contra a ocorrência de erros que possam ocorrer na mensagem recebida. Por tratar-se de um sistema susceptível à ocorrência de erros em surto (*burst*), escolhe-se normalmente códigos do tipo Reed-Solomon, que possuem alta capacidade de correção para este tipo de erro [10]. A saída do ECC é conectada a um codificador para gravação. Este segundo codificador, tem como característica principal, a capacidade de adaptar os dados de entrada numa forma adequada às condições do canal (codificação de linha). O arranjo em cascata empregando codificadores, é conhecido como esquema concatenado [2][10].

Os sistemas de gravação de dados possuem muitos problemas que motivam o emprego dos códigos de modulação. Entre os mais críticos, pode-se destacar:

- O canal de gravação não responde bem a sinais de baixas frequências;
- Problemas de recuperação do relógio na decodificação podem ocorrer se o sinal de leitura possuir longas sequências de 0s e 1s;

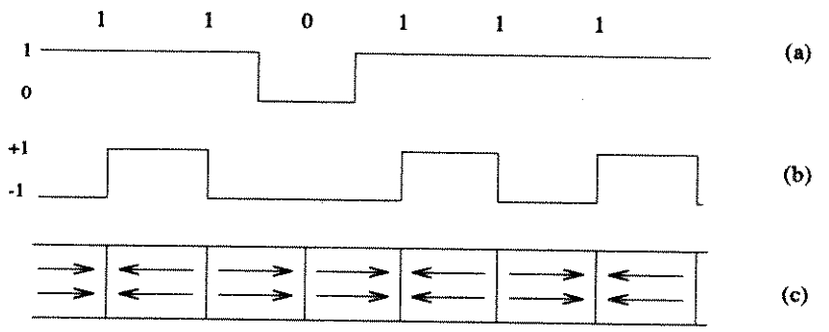


Figura 2.2: O processo de escrita de dados: a) sequência de entrada, b) níveis da corrente de escrita (NRZI), c) orientação dos domínios no meio magnético

- Ocorrência de interferência intersimbólica (ISI), alterando a amplitude e forma dos pulsos;
- Não linearidades no meio magnético (canal) provocadas pelo efeito de histerese podem corromper o sinal de leitura, alterando a forma do sinal.

Uma das famílias de códigos que vem sendo empregadas com bastante sucesso são os códigos denominados (d, k) ou RLL (*Runlength Limited Codes*), sobre os quais ter-se-á muito o que falar nos próximos capítulos.

Após a informação sair do codificador de modulação ela estará praticamente pronta para o processo de escrita propriamente dito, devendo ser antes convertida para o formato NRZI, formando o código RLL.

Na codificação NRZI, a ocorrência do bit 1 é convertida para uma transição entre dois níveis (1 para -1, ou o inverso), enquanto que a ocorrência do bit 0 aparece como ausência de transição, conforme ilustram as partes a) e b) da Figura 2.2.

A razão para o emprego da pré-codificação NRZI é justificada por ser importante para prevenir que a ocorrência de um único erro na escrita, gere um surto de erros no processo de leitura.

Durante a fase de escrita no meio magnético, a informação a ser armazenada, composta de 1s e -1s, é transformada em corrente que ao passar pela cabeça de gravação, induz um fluxo magnético no disco, orientando os magnetos (domínios magnéticos) numa direção determinada pela polaridade da corrente de escrita, conforme as partes b) e c) da Figura 2.2.

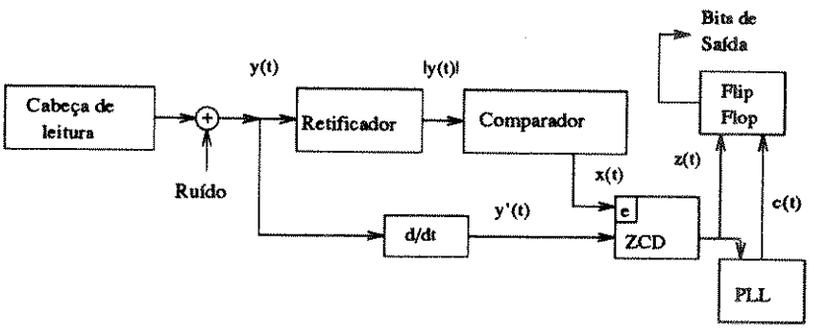


Figura 2.3: Diagrama em blocos de um detector de pico

2.2.2 O Processo de Leitura

O processo de leitura consiste no inverso. À medida que o meio magnético passa sob a cabeça de leitura, é induzida uma tensão na cabeça, proporcional à derivada do fluxo magnético dos magnetos, e portanto proporcional à derivada da corrente de escrita.

Os dados passam por um detector de pico, onde um pico e vale detectado é convertido em bit 1 ou 0, respectivamente. A informação passa então pelos decodificadores de gravação e corretor de erros, estando, logo após, disponíveis para o usuário.

O Detector de Pico

Trata-se de um dispositivo robusto, de fácil implementação e com ótimo desempenho em médias e baixas densidades. As Figuras 2.3 e 2.4 mostram o diagrama em blocos de um detector de pico juntamente com o seu funcionamento.

O sinal lido pela cabeça de gravação atravessa um circuito retificador produzindo o sinal $|y(t)|$ que passa por um comparador, produzindo o sinal x(t). Esta comparação com um limiar, determina o pico que será usado para habilitar o circuito detector de cruzamentos de zero (ZCD).

No caminho inferior, o sinal de leitura passa por um circuito diferenciador, obtendo-se o sinal y'(t), que será usado como amostrador do sinal x(t). Como resultado tem-se o sinal z(t).

Quando a saída x(t) do ramo superior está no nível alto, e simultaneamente, ocorre um cruzamento pelo nível zero, o ZCD detecta esta ocorrência e responde com um nível alto na saída. Não ocorrendo essa situação, a saída será zero.

Um PLL (*Phase-Locked Loop*) é usado para obtenção da frequência original

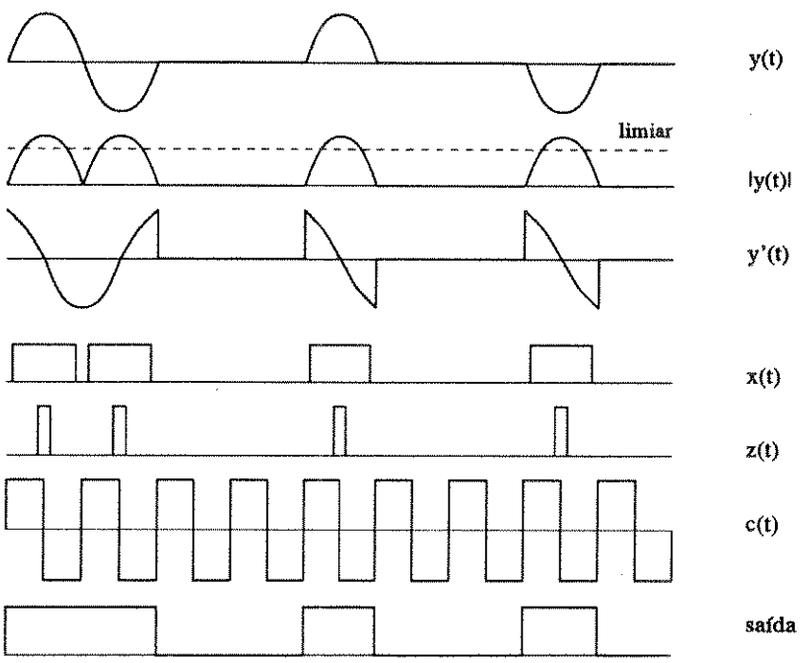


Figura 2.4: Funcionamento de um circuito detector de pico

(relógio) do sinal de entrada. Utilizando como referência o sinal z(t), obtém-se na saída do PLL, o relógio c(t).

Por fim, se durante a descida do relógio c(t) existir sinal na entrada do flip-flop este responderá com nível alto na saída e o nível lógico 1 será detectado, do contrário a saída será 0.

2.2.3 Tendências em Gravação Magnética

Nos últimos anos um novo sistema vem sendo alvo de grande interesse: o detector por amostragem (*sampling detector*). Este dispositivo foi desenvolvido com uma nova tecnologia para processamento de sinais e codificação com aplicações em sistemas que exijam alta densidade de armazenamento. Esta tecnologia utiliza elementos como: equalização, códigos de treliça, decodificadores de Viterbi e filtragem adaptativa [8][9].

Entre os códigos para gravação mais empregados, destacam-se os códigos *dc-free* e *Nyquist-free*. Estes códigos caracterizam-se pela existência de nulos espectrais na frequência dc e em metade da frequência de amostragem, respectivamente.

A técnica de detecção por amostragem não será abordada neste trabalho, contudo os algoritmos aqui empregados para obtenção de códigos aplicados ao detector de pico (RLL) podem ser, neste outro caso, igualmente aplicados.

2.3 ASPECTOS BÁSICOS DOS CÓDIGOS RLL

2.3.1 Representação

Os códigos RLL, criados por Franaszek, caracterizam-se por dois parâmetros (d e k), os quais controlam o número mínimo e máximo de 0s entre 1s consecutivos numa sequência de informação qualquer [1][6].

O parâmetro d controla o nível de interferência inter-simbólica produzida no meio de gravação. Este parâmetro também relaciona-se diretamente com a densidade de gravação, conforme será visto mais adiante. O parâmetro k tem importância na escolha do circuito de recuperação de relógio na decodificação, uma vez que ele inibe a ocorrência de longas sequências de zeros.

O número de sequências dk de um certo comprimento n, chamado $N(n)$ é dado por [6]:

$$N(n) = n + 1 \quad se \quad 1 < n \leq d + 1 \tag{2.1}$$

$$N(n) = N(n-1) + N(n-d-1) \quad se \quad d+1 \leq n \leq k \tag{2.2}$$

$$N(n) = d + k + 1 - n + \sum_{i=d}^{k} N(n-i-1) \quad se \quad k < n \leq d + k \tag{2.3}$$

$$N(n) = \sum_{i=d}^{k} N(n-i-1) \quad se \quad n > d + k \tag{2.4}$$

As sequências (dk) podem ser representadas, por meio de um diagrama de estados (FSTD - *Finite State Transition Diagram*) conforme ilustrado na Figura 2.5. Qualquer caminho através de um FSTD define uma dessas sequências. Seja G o FSTD representando uma sequência (dk) qualquer. O conjunto de todas as sequências finitas obtidas através do FSTD G, define o que é chamado de Sistema Restrito S.

A correspondente forma matricial A, denominada matriz de transições é a seguinte:

$$a_{ij} = \begin{cases} a_{i1} = 1 & \text{se } i \geq d+1 \\ a_{ij} = 1 & \text{se } 1 \leq i \leq k-1 \text{ e } j = i+1 \\ a_{ij} = 0 & \text{demais casos} \end{cases} \tag{2.5}$$

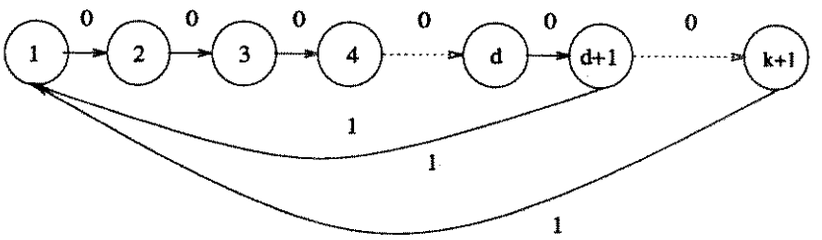


Figura 2.5: Diagrama de estados (FSTD) de uma sequência (dk)

onde os elementos a_{ij} indicam o número de ramos que vão do estado i ao estado j em um FSTD qualquer.

Da Teoria da Informação sabe-se que a capacidade de canal de um canal sem ruído C é definida como sendo a maior taxa com a qual se pode realizar um codificador. A capacidade de canal C de um sistema que emite sequências (dk) é calculada tomando-se o logaritmo na base 2 do maior autovalor (λ) real da matriz de transições A [11]. Ou seja:

$$C(d,k) = \log_2 \lambda \tag{2.6}$$

A capacidade C também pode ser obtida a partir da expressão abaixo:

$$C(d,k) = \lim_{n\to\infty} \frac{\log_2 N(n)}{n} \tag{2.7}$$

De acordo com a equação (2.4), fazendo-se a aproximação $N(n) = z^n$, válida quando $n \to \infty$, obtém-se a expressão abaixo:

$$z^{k+2} - z^{k+1} - z^{k-d+1} + 1 = 0 \tag{2.8}$$

A expressão acima é denominada equação característica. A maior raíz real desta equação constitui o parâmetro λ que será aplicado à (2.6) tendo-se por fim determinar a capacidade para o código dk.

A Tabela 2.1 ilustra uma relação das capacidades para diversas restrições dk. Esta Tabela foi conseguida através de uma rotina componente de um software aqui desenvolvido, a ser descrito com mais detalhes, adiante.

Adicionalmente define-se um parâmetro que especifica o ganho em densidade de armazenamento, que é expresso da seguinte forma [6]:

$$GD = (d+1)R \tag{2.9}$$

onde R é a taxa do código e d, como definido acima, especifica o número mínimo de zeros entre uns consecutivos numa dada sequência (dk).

O ganho em densidade, quando comparado com um sistema não codificado vale então:

$$GD(\%) = (GD - 1) \times 100 \tag{2.10}$$

Um outro parâmetro importante é conhecido como eficiência do código, η. Este parâmetro ilustra quão próximo pode-se estar da capacidade de canal e é representado pela seguinte expressão:

$$\eta = \frac{R}{C(d,k)} \tag{2.11}$$

Tabela 2.1

Tabela de Capacidades para os Códigos dk

k \ d	0	1	2	3	4	5
1	0,6942	0,0	-	-	-	-
2	0,8752	0,4057	0,0	-	-	-
3	0,9468	0,5515	0,2878	0,0	-	-
4	0,9752	0,6174	0,4057	0,2232	0.0	-
5	0,9881	0,6509	0,4650	0,3218	0,1823	0,0
6	0,9941	0,6690	0,4979	0,3746	0,2669	0,1539
7	0,9971	0,6793	0,5174	0,4057	0,3142	0,2281
8	0,9986	0,6853	0,5293	0,4251	0,3432	0,2708
9	0,9993	0,6888	0,5369	0,4376	0,3620	0,2978
10	0,9997	0,6909	0,5418	0,4460	0,3746	0,3158
11	0,9998	0,6922	0,5450	0,4517	0,3832	0,3282
12	0,9999	0,6930	0,5471	0,4556	0,3894	0,3369
13	0,9999	0,6935	0,5485	0,4583	0,3937	0,3432
14	0,9999	0,6938	0,5495	0,4605	0,3968	0,3478
∞	1,0	0,6942	0,5515	0,4650	0,4057	0,3620

2.3.2 Espectro

A Figura 2.6 mostra o espectro de algumas sequências dk quando $k \rightarrow \infty$. A partir desta figura pode-se observar que esses códigos possuem uma má característica de resposta em baixas frequências. Esta característica é particularmente interessante já que nessa faixa usualmente são transmitidas informações de controle, e portanto não é desejável que os códigos RLL tenham espectro de potência apreciável nesta faixa.

Uma vez estabelecidos os parâmetros, as características principais das sequências dk, é necessário agora definir regras para o projeto de codificadores eficientes que possam ser construídos a partir de simples circuitos lógicos.

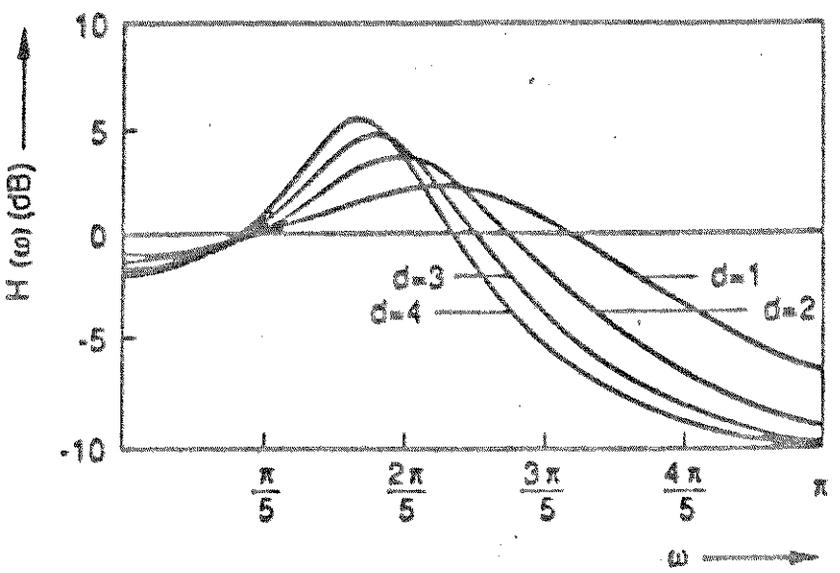


Figura 2.6: Espectro de algumas sequências dk

2.4 TÉCNICAS EMPREGADAS NA OBTENÇÃO DOS CÓDIGOS RLL

Existem diversas técnicas que tem sido empregadas com sucesso na obtenção dos códigos RLL. As mais conhecidas são: comprimento variável, decodificação *look-ahead* e blocos deslizantes. Será descrita a seguir, de maneira breve, cada uma delas.

2.4.1 Comprimento Variável

Esta técnica será discutida utilizando-se de um exemplo. Uma discussão mais profunda acerca dos procedimentos foge aos objetivos deste trabalho. O leitor é convidado a consultar as referências [5][6] para maiores detalhes.

A Tabela 2.2 ilustra um código de taxa 1/2 com restrições ($d = 2, k = 7$) [9]. Este código foi desenvolvido por Franaszek, cujos trabalhos foram pioneiros nesse assunto. Uma variante do código (2,7) é utilizada pela IBM nos seus discos rígidos 3370/3380, conforme a Tabela 2.3 [6].

Tabela 2.2

Tabela de um Codificador (2,7), Taxa 1/2

Dados	Pal. Código
10	0100
11	1000
000	000100
010	100100
011	001000
0010	00100100
0011	00001000

Tabela 2.3

Tabela de um Codificador (2,7), Taxa 1/2

Dados	Pal. Código
10	1000
11	0100
000	100100
010	001000
011	000100
0010	00001000
0011	00100100

Pode-se observar também que nenhuma palavra-código é prefixo de outra sendo, portanto este código classificado como código de prefixo [13]. Esta característica assegura que qualquer concatenação de palavras-código produzirá uma decodificação única.

Entretanto, o procedimento para obtenção de Tabelas como as acima citadas não é simples e uma cuidadosa escolha das palavras-código e atribuição dessas às sequências de informação deve ser feita.

2.4.2 Técnica de Decodificação *Look-Ahead*

Trata-se de uma técnica desenvolvida por vários autores [1][6]. Como na secção anterior, também será utilizado um exemplo para descrição do método.

Seja o caso de um código (1,7), que pela Tabela 2.1, permite que seja escolhido um código taxa 2/3. Um código com essas características é ilustrado na Tabela 2.4.

Tabela 2.4

Tabela de um Codificador (1,7), Taxa 2/3

Dados	Pal. Código
00	101
01	100
10	001
11	010

Através dessa Tabela, pode-se observar que a concatenação de algumas palavras-código viola a restrição (1,7), mais especificamente, as sequências: 00.00, 00.01, 10.00, 10.01.

Visando superar a violação da restrição, apresentou-se como solução uma Tabela de substituição para algumas sequências, conforme mostra a Tabela 2.5. A codificação funcionará da seguinte maneira: ao receber o primeiro bloco de 2 bits, o codificador irá esperar pelo próximo bloco (*look-ahead*). Se a concatenação do bloco seguinte com o atual violar a restrição, o codificador fará uso da Tabela 2.5. Do contrário, se não ocorrer violação, a Tabela 2.4 será utilizada normalmente.

Tabela 2.5

Tabela de Substituição para o Codificador (1,7), Taxa 2/3

Dados	Pal. Código
00.00	101.000
00.01	100.000
10.00	001.000
10.01	010.000

O procedimento a ser seguido no processo de decodificação é o mesmo utilizado na codificação, ou seja, o decodificador irá esperar pelos próximos blocos para decidir como o bloco atual será decodificado. Este é o fundamento da técnica *look-ahead*.

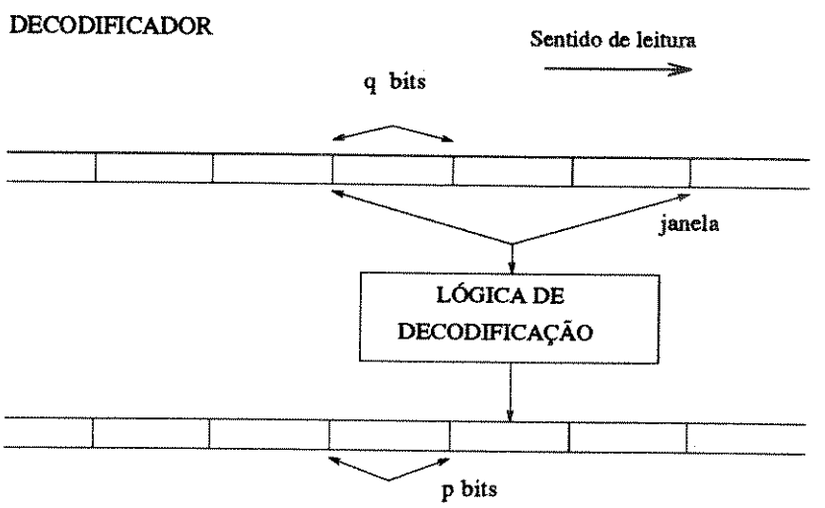


Figura 2.7: Decodificador de blocos deslizantes

2.4.3 Algoritmo dos Blocos Deslizantes (*Sliding Block*)

Esta técnica, também chamada de algoritmo dos blocos deslizantes, vem sendo desenvolvida desde a última década e está sendo empregada com bastante sucesso como um procedimento sistemático na busca de códigos RLL.

Desenvolvido por Adler, Coppersmith e Hassner, o algoritmo usou de técnicas derivadas de um ramo da matemática conhecido como Dinâmica Simbólica [7][8].

O algoritmo faz uso de um conjunto de técnicas denominadas, divisão e fusão de estados, para a obtenção de codificadores com número de estados reduzido [14]. Esses conceitos ficarão mais claros no próximo capítulo, quando serão discutidos detalhadamente.

O termo blocos deslizantes refere-se ao decodificador. Mais explicitamente, diz-se que um decodificador é de blocos deslizantes quando para decodificar uma palavra de comprimento q, é necessário olhar a blocos à frente e m blocos atrás, onde a é dita a antecipação e m a memória do codificador. A propagação de erros neste esquema é finita e fica restrita a uma janela de comprimento $m + a + 1$ blocos. Na decodificação da palavra-código seguinte, o circuito incrementa de uma unidade e toma o bloco seguinte, conforme mostra a Figura 2.7.

A técnica brevemente discutida acima, constitui-se na base desta dissertação e será abordada, com todos os detalhes necessários, nos próximos capítulos.

Antes da passagem para o próximo capítulo é interessante estabelecer aqui algumas definições, envolvendo termos que serão empregados com frequência no decorrer deste trabalho.

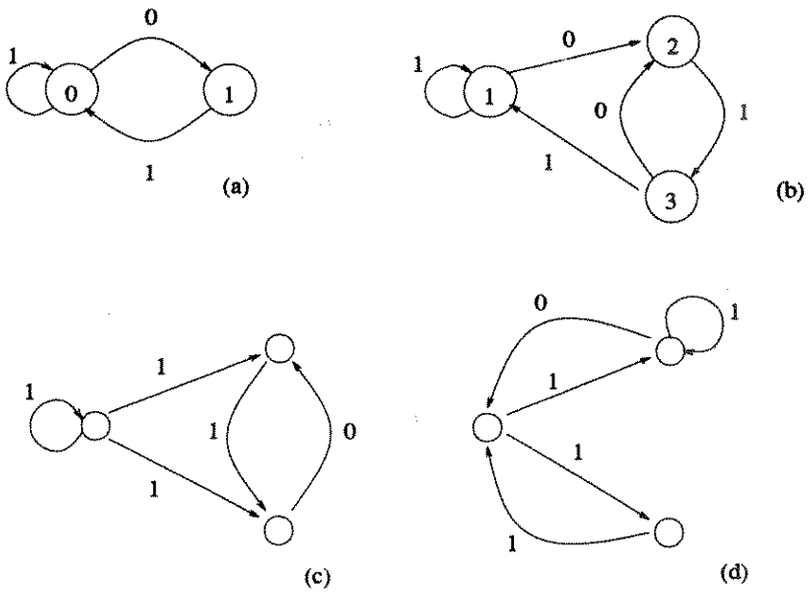


Figura 2.8: Alguns exemplos de FSTDs. (a) e (b) FSTDs determinísticos. (c) e (d) não determinísticos

2.5 CONCEITOS ENVOLVENDO SEQUÊNCIAS *dk*

Um diagrama transição de estados finitos (FSTD) é um diagrama composto de um número finito de estados e de ramos de transição entre estados. Cada ramo de transição está associado a um rótulo como em (2.5). A Figura 2.8 ilustra alguns exemplos.

Diz-se que um codificador de taxa p/q é de estados finitos quando ele puder ser representado via um FSTD, cujos rótulos são da forma s/t, onde s indica a palavra de informação de p bits e s a palavra codificada de q bits. Este codificador aceita sequências de informação de comprimento p bits e as transforma em sequências codificadas de comprimento q bits. Cada sequência codificada depende tanto da sequência de informação quanto do estado atual do codificador.

O conjunto de todas as sequências finitas geradas por um FSTD chama-se sistema restrito S.

Um dado FSTD é determinístico se os ramos de saída de cada estado são associados a sequências de informação distintas. Na Figura 2.8 os FSTDs a e b são determinísticos, enquanto que c e d não.

Define-se um FSTD G como irredutível quando for possível acessar o estado

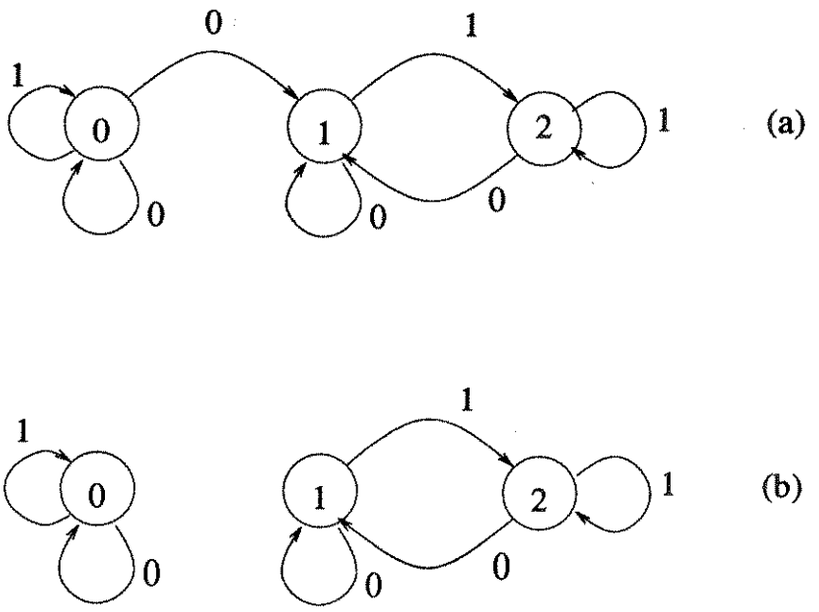


Figura 2.9: Alguns exemplos de FSTDs. (a) FSTD redutível. (b) FSTDs irredutíveis

i a partir de outro qualquer j. Quando esta situação não ocorrer se dirá que ele é redutível. Um FSTD redutível pode ser quebrado em FSTDs irredutíveis. Na Figura 2.9a, o FSTD é redutível enquanto que em 2.9b ele foi quebrado em dois FSTDs irredutíveis [15].

Capítulo 3

O ALGORITMO DOS BLOCOS DESLIZANTES

3.1 A CONSTRUÇÃO DOS CÓDIGOS RLL

Será descrito, no transcorrer deste capítulo, o procedimento completo para a obtenção dos códigos RLL, utilizando o Algoritmo dos Blocos Deslizantes [7][8][16].

Visando o projeto de um codificador RLL, deve-se, inicialmente escolher as restrições d e k com as quais deseja-se trabalhar. Seja G o FSTD associado a esse sistema. A partir destas restrições será obtida a matriz de transições A e daí, conforme descrito no capítulo anterior, será calculada a capacidade de canal C para este sistema.

A escolha da taxa $R = p/q$ com a qual o codificador trabalhará, deverá obedecer à condição de que $R \leq C$. Os blocos codificados terão comprimento q e serão gerados percorrendo-se caminhos de tamanho q no FSTD G; tais blocos representarão os rótulos dos ramos (labels) no novo FSTD, denominado G^q. O FSTD G^q, representado por A^q, obviamente, conterá o mesmo número de estados do FSTD G, representado por A.

O número de sequências distintas de comprimento q, que emanam de um estado i e terminam num estado j, é dado pelo elemento (i, j) de A^q. Uma propriedade interessante é a de que a capacidade de canal para um sistema A^q, vale $q \times C$. Como exemplo, a matriz A^{10} para o código (1,3) é dada por:

$$A^{10} = \begin{bmatrix} 17 & 11 & 8 & 5 \\ 24 & 17 & 11 & 8 \\ 19 & 13 & 9 & 6 \\ 11 & 8 & 5 & 4 \end{bmatrix}$$

Pode ser visto, por exemplo, que há exatamente 13 sequências de comprimento

10 que emanam do estado 3 e terminam no estado 2.

Contudo, para que este diagrama represente, de fato, um codificador, é necessário que se tenha, no mínimo, 2^p ramos emanando de cada estado em G^q. Isto pode ser verificado através de sua matriz de transições, tomando-se a soma de todos os elementos de cada linha da matriz A^q, que nunca poderá ser inferior a 2^p. Esta condição, chamada grau de saída, pode ser sintetizada através da seguinte desigualdade:

$$A^q \vec{v} \geq 2^p \vec{v} \tag{3.1}$$

com $\vec{v} = [11...1]$.

Quando (3.1) é satisfeita para algum vetor coluna inteiro e não negativo então este é denominado *autovetor aproximado*. A existência de autovetores aproximados é garantida pela teoria de Perron-Frobenius para matrizes não negativas [8][15].

Quando $\vec{v}$ puder ser obtido com componentes apenas 0s e 1s, então existe um subconjunto de G^q cujo FSTD satisfaz a condição de grau de saída, e a obtenção do codificador será direta. Nos casos onde esta condição não for satisfeita e tivermos alguma componente em $\vec{v}$ com valor maior que 1, deve-se encontrar meios de modificar o FSTD para se chegar a essa condição e com isso obter o codificador. Um procedimento que tem sido empregado com eficácia garantida e eficiência satisfatória é o algoritmo para divisão de estados (*state splitting*).

Franaszek introduziu um algoritmo eficiente para encontrar autovetores aproximados [8] [12]. Sua descrição é a seguinte:

Algoritmo 3.1 (Franaszek) *:*

1 - Inicie com l=0:

2 - Faça $v^{(0)} = (L, L, ..., L)$*, como sendo um vetor de* $k+1$ *elementos (o valor a ser escolhido para L será- discutido mais adiante) :*

3 - Para cada coordenada i, defina

$$v_i^{(l+1)} = min\lfloor v_i^{(l)}, (1/2^p)(\sum_{j=0}^{k+1} t_{ij} v_j^{(l)})\rfloor \tag{3.2}$$

4 - Se $\vec{v}^{(l+1)} \neq \vec{v}^{(l)}$*, incremente l e volte para 3. Do contrário* $\vec{v} = \vec{v}^{(l)}$*, ou seja,* $\vec{v}$ *é o autovetor aproximado desejado.*

onde t_{ij} é o ij-ésimo elemento da matriz de transições de G^q.

Devem ser feitas algumas considerações a respeito da constante L. Sugere-se que L deva ser escolhido como um inteiro não negativo qualquer ($L = 1$, por exemplo). Se o autovetor aproximado encontrado for um vetor nulo quer dizer que o valor escolhido para L foi muito pequeno. Um novo L, mais alto que o anterior, deverá ser buscado e o processo reiniciado, este procedimento será repetido até que um autovetor aproximado seja encontrado.

Sugere-se que se inicie com $L = 1, 2, 4, ...$, ou seja, crescendo-se de potências de 2 até que um autovetor aproximado seja encontrado, num $L = 2^i$ qualquer. Em seguida, pesquisa-se dentro do intervalo $[2^{i-1}; 2^i]$ onde um autovetor menor deverá ser obtido. Vale salientar que quanto menor o valor de L, mais simples poderá ser o codificador.

Visando uma simplificação maior no número de iterações do algoritmo dos blocos deslizantes, pode-se tentar reduzir as componentes do autovetor aproximado encontrado via algoritmo de Franaszek em componentes ainda menores. Esta simplificação consiste na redução, componente a componente, do autovetor aproximado, testando-se a cada redução a desigualdade 3.1, até que nenhuma redução seja mais possível.

O passo seguinte consiste na simplificação do FSTD original. De posse do vetor $\vec{v}$ deve-se suprimir os i elementos de $\vec{v}$ cujas componentes sejam nulas. Isto corresponderá à eliminação dos i estados no FSTD de G^q, que por sua vez corresponderá à eliminação de todas as linhas e colunas em A^q que envolvam tais estados.

Será verificado então se o FSTD de G^q obedece à condição de irredutibilidade. Se for irredutível pode-se continuar o processo de construção; do contrário, se G^q for redutível, deve-se procurar, em G^q, a componente irredutível H que seja *sink*, ou seja, qualquer ramo que originar-se de qualquer estado em H deverá continuar em H [15]. Toma-se esta componente H e continua-se o processo. Mais adiante, no Capítulo 4, será descrito um algoritmo para busca de componentes irredutíveis.

A partir da expressão 3.1 será definido o pêso de um estado j, como sendo v_j, o valor da j-ésima componente de $\vec{v}$, e o pêso de um ramo ϵ qualquer $w(\epsilon)$, como sendo o pêso do estado terminal de ϵ, da seguinte forma:

$$w(\epsilon) = v_j \tag{3.3}$$

A expressão 3.1, poderá agora ser reescrita como um conjunto de desigualdades escalares, uma para cada estado i, da seguinte forma:

$$\sum_{\epsilon \in E_i} w(\epsilon) \geq 2^p v_i \tag{3.4}$$

onde E_i representa o conjunto de ramos que saem do estado i num FSTD qualquer. Desta forma, a soma dos pêsos dos ramos que emanam do estado i é, no mínimo, igual a 2^p vezes o próprio pêso de i.

Será introduzido agora o conceito de Divisão de Estados.

3.1.1 Divisão de Estados (*State Splitting*)

Seja H um FSTD, e seja $\vec{v}$ o autovetor aproximado associado com componente máxima $v_i > 1$. Isto indica que o estado i deverá ser dividido em novos estados. Seja E_i o conjunto de ramos que saem do estado i em H. Define-se

$$E_i = E_i^1 \cup E_i^2 \tag{3.5}$$

como uma divisão básica de E_i em dois conjuntos disjuntos. Esta divisão definirá um novo FSTD H', o qual consistirá de todos os estados $j \neq i$ adicionados de 2, denominados i_1 e i_2, os quais serão chamados estados descendentes do seu antecessor i. Deve-se atentar então para um rearranjo nos ramos que emanam e chegam a i. Será dado, a seguir, o procedimento da distribuição de ramos dentro dos diversos casos possíveis:

Regra 3.1 (Distribuição de Ramos) *:*

- *CASO I: Ramos em H que iniciam de um estado $j \neq i$ e terminam no estado i (denominados ramos sucessores de j). Para cada ramo deste tipo, dois novos ramos serão criados, e_1 e e_2, um para cada novo estado i_1 e i_2, respectivamente.*
- *CASO II: Ramos em H que iniciam em i e terminam num estado j qualquer (ramos sucessores de i). Suponha que o ramo e pertença ao conjunto E_i^1 (E_i^2), uma partição de E_i. Em H' haverá um ramo correspondente de i_1 (i_2) para j.*
- *CASO III: Ramos em H que iniciam num estado $j \neq i$ e terminam num outro estado qualquer diferente de i. Esses ramos serão reproduzidos em H'.*
- *CASO IV: Ramos em H que iniciam e terminam num estado i qualquer(autoenlace). Suponha que o ramo e pertença ao conjunto E_i^1 (E_i^2). Em H' haverá um autoenlace em E_i^1 (E_i^2) e um ramo saindo de E_i^1 (E_i^2) para E_i^2 (E_i^1).*

A Figura 3.1 ilustra bem cada caso acima. Nela pode-se verificar que i foi particionado em dois novos estados i_1 e i_2 com ramos correspondentes : $E_i^1 = \{a, b\}$ e $E_i^2 = \{c\}$.

Algebricamente, pode-se expressar o problema da divisão de estados da seguinte maneira:

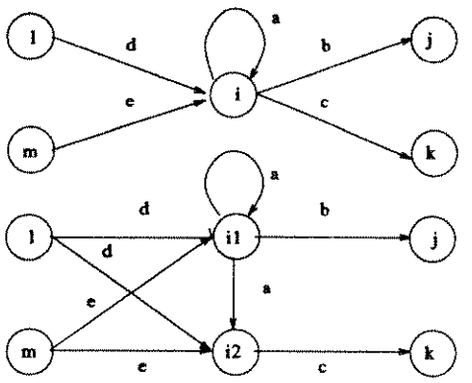


Figura 3.1: Exemplo de divisão de estados

$$\sum_{\epsilon \in E_i^1} w(\epsilon) \geq 2^p y_1 \tag{3.6}$$

$$\sum_{\epsilon \in E_i^2} w(\epsilon) \geq 2^p y_2 \tag{3.7}$$

com y_1 e y_2 satisfazendo

$$y_1 \geq 1 \tag{3.8}$$

$$y_2 \geq 1 \tag{3.9}$$

$$y_1 + y_2 = v_i \tag{3.10}$$

A nova matriz de transições A' representando o novo FSTD H', possuirá $\vec{v'}$ como autovetor aproximado, o qual terá como componentes:

$$v'_j = \begin{cases} v_j & \text{se } j \neq i \\ y_1 & \text{se } j = i_1 \\ y_2 & \text{se } j = i_2 \end{cases}$$

Proposição 1 *Seja A a matriz de transições de um FSTD irredutível H e assuma que não exista um autovetor aproximado de componentes todas iguais a 1. Seja $\vec{v}$ um autovetor aproximado com pelo menos uma componente maior que 1. Então existe pelo menos uma divisão básica de estados em H [8].*

Corolário 1 *É sempre possível encontrar um estado bifurcável entre aqueles estados que possuam péso máximo [8].*

A prova da Proposição 1 não será dada aqui, mas poderá ser encontrada em [8]. A partir da demonstração pode-se extrair a seguinte regra de divisão:

Algoritmo 3.2 (Divisão de Estados) *:*

Tome uma linha i em A^q, representando um estado E_i qualquer no FSTD H^q, com $v_i = v_{max}$ e seja $E_i = \{e_1, e_2, ..., e_M\}$ o conjunto dos ramos que partem do estado i, onde

$$M = \sum_{j=1}^{k+1} a_{ij} \tag{3.11}$$

com $k+1$ sendo o número de colunas da matriz A^q, ou de estados no FSTD H^q e a_{ij} é o ij-ésimo elemento da matriz de transições A^q..

Suponha, sem perda de generalidade, que $w(e_1) < v_{max}$ e seja $W_m = \sum_{k=1}^{m} w(e_k)$ para $m = 1, ...M$. Os seus resíduos R_m módulo n, $n = 2^p$, serão $R_m = W_m(mod(n))$ para $m = 1, ...M$, onde p é o número de bits de informação.

Os n resíduos irão satisfazer a pelo menos uma das seguintes situações:

1) $R_m = 0$ (mod n), para algum m, em que $1 \leq m \leq n$;
2) $R_{m_1} = R_{m_2}$, para m_1 e m_2 quaisquer, em que $1 \leq m_1 < m_2 \leq n$.

Sob uma dessas condições, será empregada, respectivamente, uma das seguintes regras para divisão do estado i:

- *CASO 1:*

$$E_i^1 = \{e_k | k = 1, ..., m\} \tag{3.12}$$

e

$$E_i^2 = E_i - E_i^1 \tag{3.13}$$

- *CASO 2:*

$$E_i^1 = \{e_k | k = m_1 + 1, ..., m_2\} \tag{3.14}$$

e

$$E_i^2 = E_i - E_i^1 \tag{3.15}$$

Como um exemplo deste processo, seja o FSTD H^q ilustrado na Figura 3.2 e seja $\vec{v} = (2, 1)$ o autovetor aproximado a ele associado, onde $p/q = 2/3$ é a taxa de codificação. A matriz A^q que representa o FSTD vale:

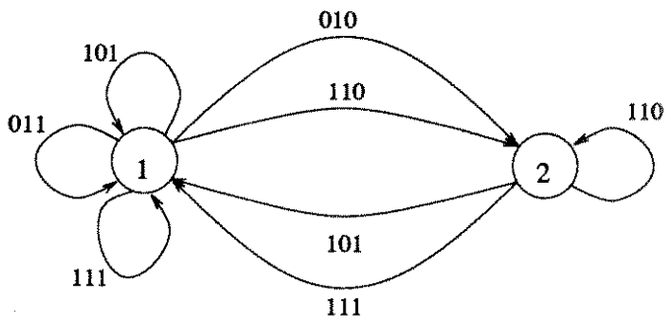


Figura 3.2: Representação do FSTD H

$$A^3 = \begin{bmatrix} 3 & 2 \\ 2 & 1 \end{bmatrix}$$

A distribuição de ramos e rótulos é ilustrada na Tabela 3.1. De acordo com o Corolário 1, o estado bifurcável é o estado 1. Tomando-se a linha 1 da matriz A^q, o número de ramos M que saem do estado 1 é igual a 5.

Deve-se considerar que $w(\epsilon_1) < v_{max} = 2$, assim, tomando-se para ϵ_1 o ramo da quarta coluna da Tabela 3.1.

Tabela 3.1

Distribuição de ramos que saem do estado 1

Estado Inicial	Ramos				
	ϵ_2	ϵ_3	ϵ_5	ϵ_1	ϵ_4
$1^{1,2}$	$1^{1,2,3}$	$1^{1,2,5}$	$1^{1,2,7}$	$2^{1,6}$	$2^{1,2}$

Sendo $n = 2^p = 4$ e $m = 1, \ldots, 5$. A notação empregada para os estados iniciais (do lado esquerdo da tabela) é da forma $i^{\epsilon,f}$, indicando que i será aberto em até f estados descendentes, e denotados $i^{\epsilon}, i^{\epsilon+1}, \ldots, i^{f}$. Para os ramos, a notação é $j^{\epsilon,f,m}$, onde ϵ e f já foram descritos anteriormente, enquanto que m é o rótulo, em decimal, do ramo que vai de i para j.

A Tabela 3.2 ilustra a bifurcação do estado 1 em dois novos estados.

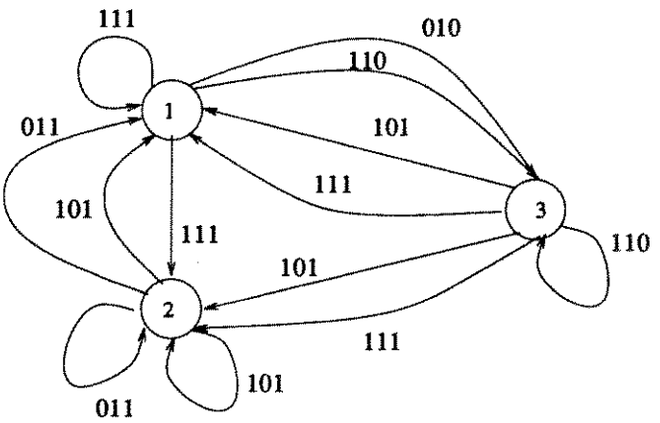


Figura 3.3: Representação do FSTD H após a bifurcação

Tabela 3.2

Exemplo de Bifurcação de Estados

m	W_m	R_m
1	$w(\epsilon_1) = 1$	1
2	$w(\epsilon_2) + W_1 = 3$	3
3	$w(\epsilon_3) + W_2 = 5$	1
4	$w(\epsilon_4) + W_3 = 6$	2
5	$w(\epsilon_5) + W_4 = 8$	0

Desta forma, conclui-se que o CASO 2 é satisfeito, com $m_1 = 1$ e $m_2 = 3$. Assim, a regra de divisão será $E_1^1 = \{\epsilon_2, \epsilon_3\}$ e $E_1^2 = \{\epsilon_1, \epsilon_4, \epsilon_5\}$. Após os remanejamentos de ramos, necessários de acordo com a regra para distribuição de ramos, obtém-se o FSTD da Figura 3.3.

Tendo-se definidas essas regras para divisão de estados, é possível afirmar que o procedimento consiste basicamente, no caso de não encontrar-se um autovetor aproximado com componentes todos 1s, em achar uma sequência de divisões sucessivas de estados (seguindo-se por exemplo a regra dada acima), com o objetivo de decompor o autovetor inicial em componentes inteiras menores ou iguais a 1. Este processo continuará até que, ao final, obtenha-se um novo FSTD H^*, com nova matriz de transições e cujo autovetor aproximado deverá conter componentes todas iguais a 1, indicando assim que o grau de saída no novo FSTD H^* será ao menos 2^p.

Pode-se deduzir facilmente que o número de divisões necessárias (SS) para reduzir o autovetor aproximado $\vec{v}$, será limitado por:

$$1 \leq SS \leq \sum_{i=1}^{k+1}(v_i - 1) \tag{3.16}$$

Desta forma, o FSTD resultante H^* terá um número de estados (σ) limitado a [14]:

$$max(v_i) \leq \sigma \leq \sum_{i=1}^{k+1}(v_i) \tag{3.17}$$

onde k é uma das restrições da sequência (dk) e $max(v_i)$ é a maior componente do autovetor aproximado.

Com este novo FSTD H^* em mãos poder-se-á dar início à construção do codificador. Tendo garantido que de cada estado partem, no mínimo, 2^p ramos, pode-se descartar os excessos, ficando-se com um número de ramos exatamente igual a 2^p. Vale salientar que uma escolha adequada dos ramos a serem descartados poderá propiciar uma simplificação maior na implementação do codificador.

Por fim, o codificador será obtido atribuindo-se rótulos da forma s/t, onde s é o bloco de entrada de comprimento p, enquanto que t é o bloco de comprimento q, produzido pelo codificador. A atribuição do bloco de entrada não deverá ser arbitrária, pois um procedimento cuidadoso de atribuição de rótulos poderá ajudar na simplificação do decodificador, por meio da redução no tamanho da janela de decodificação [8][22]. Na Secção 3.4, serão apresentados exemplos que ajudarão a entender melhor este procedimento.

A Figura 3.4 ilustra a situação em que uma escolha inadequada de rótulos provocou um aumento no tamanho da janela de decodificação de 1 (a) para 2 (b) blocos.

O tamanho da janela de decodificação J é função da memória (m) e da antecipação (a) do FSTD de H^q bem como do número necessário de divisões sucessivas (SS), a fim de ter-se um autovetor aproximado final com componentes todas iguais a 1, ou seja:

$$J \leq m + a + SS + 1 \tag{3.18}$$

Será apresentada agora uma técnica que ajudará a simplificar o projeto do codificador.

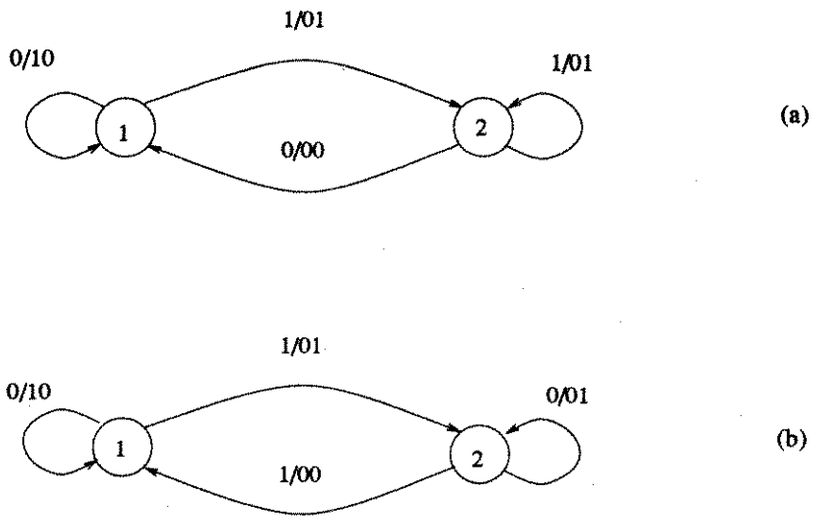


Figura 3.4: Exemplos de atribuição de rótulos. Janela de decodificação de 1 bloco (a) e de 2 blocos (b)

3.1.2 Fusão de Estados (*State Merging*)

A fusão de estados tem por objetivo básico, fundir estados que reunam certas características comuns em um só estado gerando, com isso, um codificador de menor complexidade.

Sejam $E_i = \{e_1, ...e_l\}$ e $E_j = \{f_1, ..., f_l\}$ o conjunto dos ramos emanando de i e j, respectivamente. Se existir, para cada $s = 1, ..., l$, um par de ramos e_s e f_s com o mesmo rótulo e que terminem no mesmo estado, poder-se-á fundir i e j em um só estado, produzindo um novo FSTD com um estado a menos, porém preservando o mesmo grau de saída 2^p.

Suponha que os estados i e j sejam equivalentes, isto é, possuam as propriedades citadas acima. A regra para fusão do estado i com o estado j é realizada da seguinte forma:

Regra 3.2 (Fusão de Estados) :

1 - Eliminam-se todos os ramos que emanam de j;

2 - Redireciona-se para o estado i todos os ramos remanescentes que terminam no estado j;

3 - Elimina-se o estado j.

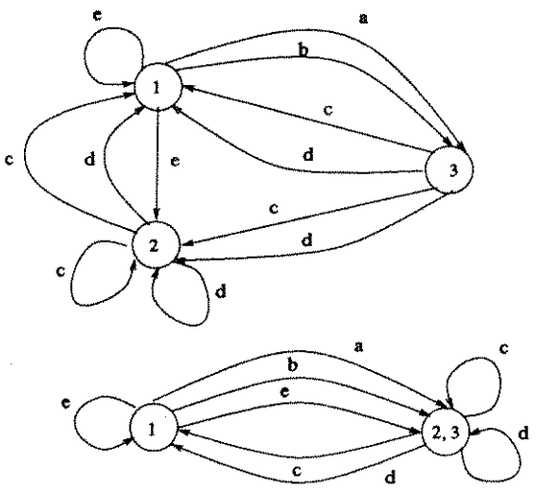


Figura 3.5: Exemplo de fusão dos estados 2 e 3

A Figura 3.5, ilustra o processo de fusão, onde os estados 2 e 3 são equivalentes, ou seja, de 3 partem os ramos {c,d} para 1 e {c,d} para 2; analogamente, partindo de 2 tem-se {c,d} para 1 e {c,d} para 2.

Uma extensão do conceito de fusão é baseada na existência de um ordenamento parcial de estados representado pelo símbolo $\prec$ do FSTD. Este ordenamento baseia-se no conceito de ordenamento de conjuntos. Explicando de uma outra forma, para um estado i, define-se o conjunto seguidor $F(i)$ como o conjunto de todas as palavras de qualquer comprimento partindo de i. Dados dois estados i e j, diz-se que $i \prec j$, se $F(i) \subseteq F(j)$, ou seja, o conjunto de sequências geradas a partir de i está incluído no conjunto de sequências geradas por j. Tendo-se observada esta condição, fundem-se os estados i e j, seguindo-se os passos 1, 2 e 3 citados acima, conforme ilustrado na fig 3.6.

Considere agora a seguinte proposição:

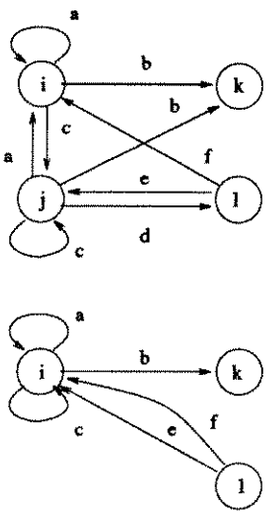
Proposição 2 *Seja G um FSTD com autovetor aproximado $\vec{v}$, e sejam i e j estados em G satisfazendo às seguintes condições [8]:*

a) $i \prec j$, isto é, $F(i) \subseteq F(j)$;
b) $v_i = v_j$.

Seja H o novo FSTD gerado a partir da fusão de i com j e seja $\vec{w}$ o autovetor aproximado a ele associado. Então:

1 - O conjunto de sequências geradas por H é um subconjunto das sequências geradas por G;

2 - O vetor $\vec{w}$, com $w_l = v_l$, para todos os estados l de H é um autovetor aproximado.



Antes da Fusão

Após a Fusão

Figura 3.6: Exemplo de fusão entre os estados i e j

A Proposição 2 acima é a formalização do que foi dito antes e permite que a extensão no conceito de fusão de estados seja aplicada na redução do número de estados do codificador.

3.2 COMPLEXIDADE DO SISTEMA

Serão descritos agora dois parâmetros importantes que ajudarão na escolha de circuitos de codificação/decodificação com mínimo hardware, as complexidades de codificação e de decodificação. Suponha que sejam empregadas, para o codificador e decodificador memórias tipo ROM (Read Only Memory).

3.2.1 Complexidade de Codificação

Para o codificador a ser projetado a memória deverá aceitar como entrada, um certo número de bits associados à palavra de informação (p bits), bem como o estado associado a esta palavra. O número de bits que representará este estado deverá ser igual a $\lceil \log_2 \sigma \rceil$, com $\sigma > 1$, onde σ é o número de estados do codificador. O número de bits necessários como entrada para este codificador deverá, portanto, ser igual a $p + \lceil \log_2 \sigma \rceil$. Com este sistema pode-se ter disponíveis até $2^{p+\lceil \log_2 \sigma \rceil}$ palavras como entrada para o codificador.

Da mesma forma que na entrada, na saída além dos q bits codificados ter-se-á $\lceil \log_2 \sigma \rceil$ bits correspondentes ao próximo estado temporal, de acordo com a Figura 3.7. Desta forma o número total de bits envolvidos no processo de codificação será dado por:

$$C_{cod} = 2^{p+\lceil \log_2 \sigma \rceil}(q + \lceil \log_2 \sigma \rceil) \ \ bits \tag{3.19}$$

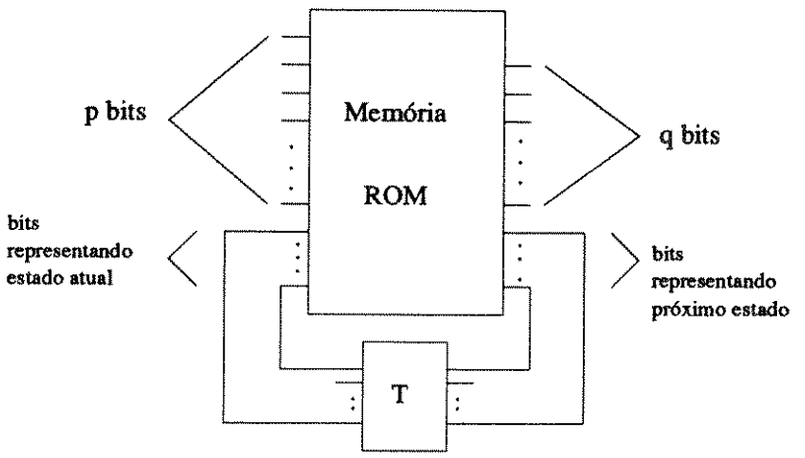


Figura 3.7: Estrutura em blocos de um circuito de codificação empregando memórias ROM

3.2.2 Complexidade de Decodificação

Com um raciocínio análogo à secção anterior, suponha uma janela de decodificação de J blocos de q bits e suponha que o bloco de qJ bits possa vir acompanhado de erros. Nessas condições qualquer palavra com qJ bits é possível de ser recebida. Suponha também que o decodificador saiba como decodificar essas palavras inválidas. O sistema deverá receber até 2^{qJ} palavras distintas e decodificá-las em sequências de p bits. Assim, o número total de bits envolvidos na decodificação será igual a $2^{qJ}p$. Este parâmetro será chamado de complexidade de decodificação.

$$C_{dec} = 2^{qJ}p \quad bits \tag{3.20}$$

3.2.3 Complexidade Total

A complexidade total do sistema será dada pela soma das expressões 3.19 e 3.20, ou seja:

$$C_T = C_{cod} + C_{dec} \quad bits \tag{3.21}$$

3.3 ALGORITMO DOS BLOCOS DESLIZANTES

O algoritmo dos blocos deslizantes encontra-se descrito abaixo como uma sequência de passos. No próximo capítulo, será descrita a transformação deste algoritmo num software para a geração de códigos RLL.

Algoritmo 3.3 (Algoritmo dos Blocos Deslizantes) *:*

1 - Escolha a restrição (d,k) com a qual deseja trabalhar. Obtenha sua matriz de transições A e tome seu FSTD G;

2 - Calcule a Capacidade de Canal (C);

3 - Escolha uma taxa $R = p/q$ para o código em que $R \leq C$;

4 - Determine a q-ésima potência de A;

5 - Utilizando o Algoritmo de Franaszek, obtenha um autovetor aproximado $\vec{v}$;

6 - Descarte os estados com $v_i = 0$ (peso nulo), e busque uma componente irredutível de G^q do tipo sink, se existirem;

7 - Aplique a regra da fusão de estados (facultativo);

8 - Se o autovetor $\vec{v}$ não possuir todas as componentes iguais a 1, aplique a regra de divisão de estados sucessivas vezes até obter-se ao final um autovetor aproximado $\vec{v} = (1111...1)$, implicando num codificador com grau de saída igual a 2^p;

9 - Elimine os ramos em excesso;

10 - Aplique novamente a regra da fusão de estados;

11 - Atribua rótulos ao codificador.

Serão fornecidos agora alguns exemplos que ajudarão numa melhor compreensão do algoritmo.

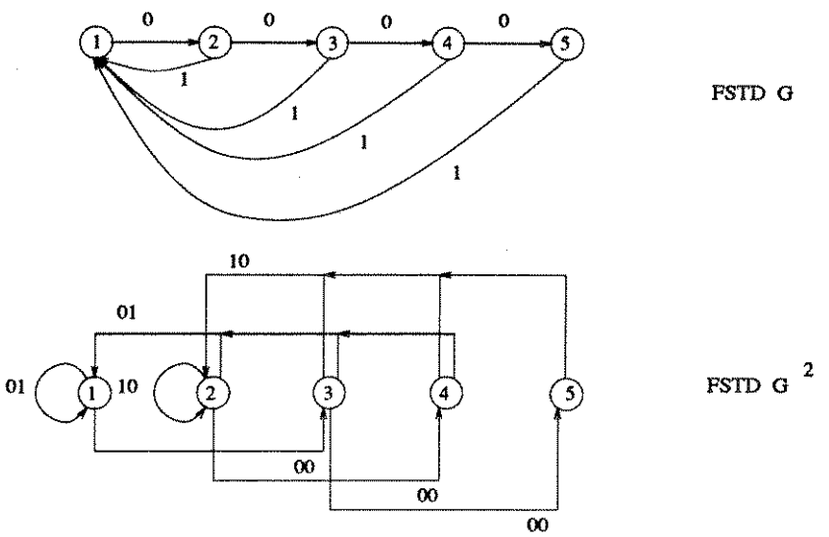


Figura 3.8: Diagrama de estados para G e G^2 com restrição (1,4)

3.4 EXEMPLOS

Exemplo 1: Esta seção inicia derivando um código (d,k)=(1,4). O FSTD G para uma sequência (1,4) é mostrado na Figura 3.8. Sua matriz de transições A é a seguinte:

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

A capacidade de canal obtida com base na Tabela 2.1 vale $C = 0,6174$. Observando-se o fato de que $R \leq C$, será escolhida a taxa $R = p/q = 1/2$. O grau de saída para cada estado deverá ser então $2^p = 2$.

Deve-se então tomar a segunda potência de A, cuja matriz de transições vale:

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

Verifica-se que o autovetor aproximado $\vec{v}$ obtido para A^2, calculado pelo algoritmo de Franaszek, quando $L = 1$, é dado por $\vec{v}^T = [1\ 1\ 1\ 1\ 0]$. Observa-se que o

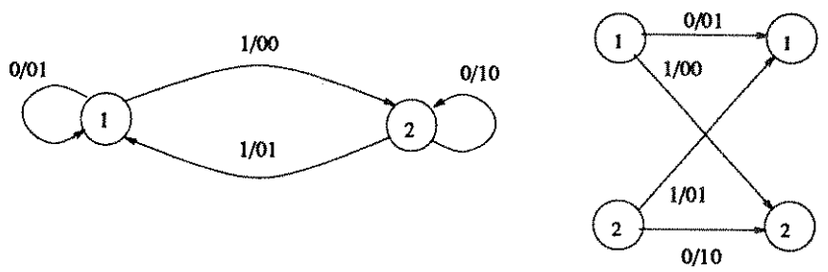


Figura 3.9: Representação do codificador (1,4) em diagrama de estados e treliça

autovetor obtido possui apenas componentes menores ou iguais a 1. Eliminando-se o 5o estado não será necessária a aplicação da regra de divisão de estados.

O autovetor aproximado $\vec{v}$, mostra que o número máximo de estados, antes das fusões de estado, é 4. A Tabela 3.3 mostra os estados com os correspondentes ramos sucessores. A notação empregada para os estados iniciais (do lado esquerdo da tabela) é da forma $i^{e,f}$, indicando que i será aberto em até $f - e + 1$ estados descendentes, e denotados $i^e, i^{e+1}, ..., i^f$, sendo f o pêso do estado i. Para os ramos sucessores, a notação é $j^{e,f,m}$, onde e e f já foram descritos anteriormente, enquanto que m é o rótulo, em decimal, do ramo que vai de i para j. Exemplificando, observando-se a Tabela 3.3, o estado 2^1, indica que o estado 2 tem como sucessores $1^{1,1}$, $2^{1,2}$ e $4^{1,0}$, ou seja, um ramo com rótulo 1 para o estado 1, um ramo com rótulo 2 para o estado 2 e um ramo com rótulo 0 para o estado 4.

Através da matriz acima, pode-se observar que, após a eliminação do estado 5 (que possui peso nulo) a condição de grau de saída já foi satisfeita, sendo assim pode-se passar diretamente para o passo 10 do algoritmo. Deletando-se convenientemente os ramos em excesso a partir do estado 2 (no caso o ramo $4^{1,0}$) obtém-se a Tabela 3.4.

Observa-se agora que os estados 2^1, 3^1 e 4^1 são equivalentes, de acordo com a regra de fusão de estados. Fundindo-se estes estados num único estado, denominado 2, obtém-se a Tabela 3.5.

O codificador (1,4), com taxa 1/2, será obtido atribuindo-se arbitrariamente rótulos de entrada conforme ilustra a Tabela 3.6. É importante observar que com o emprego da técnica de fusão foi possível reduzir o número de estados de 4 para apenas 2.

O diagrama de estados do codificador junto com sua representação em treliça é mostrado na Figura 3.9.

Tabela 3.3

Tabela Inicial de Estados e Ramos

Estado	Sucessores
1^1	$1^{1,1}\ 3^{1,0}$
2^1	$1^{1,1}\ 2^{1,2}\ 4^{1,0}$
3^1	$1^{1,1}\ 2^{1,2}\ 5^{0,0}$
4^1	$1^{1,1}\ 2^{1,2}$
5^0	$2^{1,2}$

Tabela 3.4

Tabela de transições reduzida após eliminação de ramos em excesso

Estado	Sucessores
1^1	$1^{1,1}\ 3^{1,0}$
2^1	$1^{1,1}\ 2^{1,2}$
3^1	$1^{1,1}\ 2^{1,2}$
4^1	$1^{1,1}\ 2^{1,2}$

Tabela 3.5

Após a fusão dos estados equivalentes $2^1, 3^1 e 4^1$

Estado	Sucessores
1	$1^1\ 2^0$
2	$1^1\ 2^2$

Tabela 3.6

Tabela do Codificador

	Entradas	
	0	1
1	1/01	2/00
2	2/10	1/01

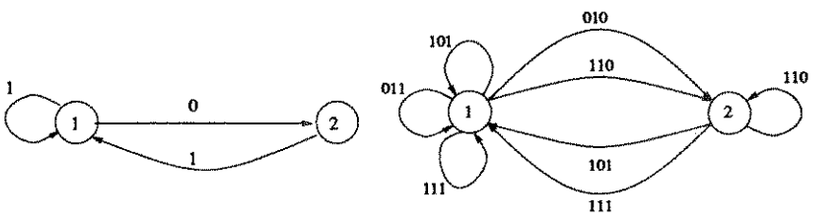


Figura 3.10: Diagrama de estados para G e G^3 com restrição (0,1)

Exemplo 2: O próximo código a ser obtido terá (d,k)=(0,1). O FSTD G para uma sequência (0,1) é mostrado na Figura 3.10. Sua matriz de transições A é a seguinte:

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

Da Tabela 2.1, $C = 0,6942$. Será escolhida a taxa $R = p/q = 2/3$. O grau de saída para cada estado deverá ser então $2^p = 4$.

Deve-se então tomar a terceira potência de A, cuja matriz de transições vale:

$$A^3 = \begin{bmatrix} 3 & 2 \\ 2 & 1 \end{bmatrix}$$

Verifica-se que o autovetor aproximado obtido $\vec{v}$ para A^3, calculado pelo algoritmo de Franaszek, quando $L = 2$, é representado por $\vec{v}^T = [2\ 1]$

A Tabela 3.7 mostra os estados com os correspondentes ramos sucessores. De acordo com o Corolário 1, o único estado que pode ser aberto é o estado 1, pois possui peso 2. A divisão dos mesmos pode ser vista na Tabela 3.8, onde o estado 1 foi dividido em 1^1, e 1^2, com peso 1 cada que terão como sucessores $\{1^{1,2,5}, 1^{1,2,7}\}$ e $\{1^{1,2,3}, 2^{1,6}, 2^{1,2}\}$, respectivamente.

Similarmente, poder-se-ia ter escolhido, para o estado 1, a seguinte divisão:
1^1 $1^{1,2,3}$ $1^{1,2,7}$
1^2 $1^{1,2,5}$ $2^{1,6}$ $2^{1,2}$

É importante ressaltar que uma boa escolha neste arranjo de ramos possibilitará, no final do processo de divisões e fusões de estados, realizar um codificador bem menos complexo.

Na Tabela 3.8, observe que o grau de saída para cada estado é de no mínimo 4. Descartando-se os ramos em excesso de modo conveniente (partindo de 2) e tomando-se estados equivalentes como é o caso dos estados 2^1 e 1^2, pode-se fundi-los num só

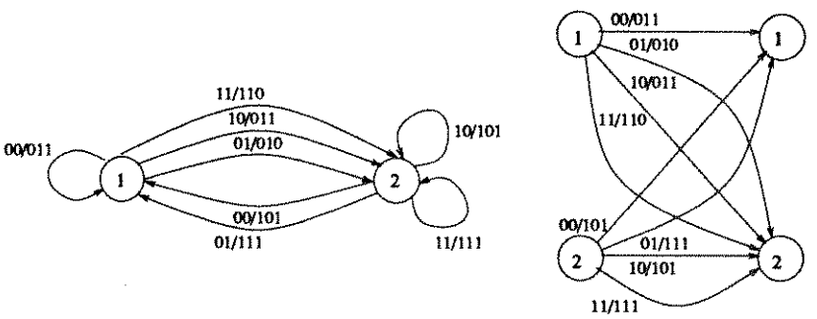


Figura 3.11: Representação do codificador (0,1) em diagrama de estados e treliça

estado denominado 2, conforme mostra a Tabela 3.10.

Para fins de simplificação, sempre que dois ou mais estados tiverem os mesmos sucessores, estes serão escritos juntos, na mesma linha, indicando que estes estados são equivalentes.

O codificador (0,1), com taxa 2/3, será obtido atribuindo-se arbitrariamente rótulos conforme a Tabela 3.11.

O diagrama de estados do codificador junto com sua representação em treliça é mostrado na Figura 3.11.

Tabela 3.7

Tabela Inicial de Estados e Ramos

Estado	Sucessores
$1^{1,2}$	$1^{1,2,3}\ 1^{1,2,5}\ 1^{1,2,7}\ 2^{1,6}\ 2^{1,2}$
2^{1}	$1^{1,2,5}\ 1^{1,2,7}\ 2^{1,6}$

Tabela 3.8

Tabela Após a Divisão do Estado $1^{1,2}$

Estado	Sucessores	Resultado
1^{1}	$1^{1,2,5}\ 1^{1,2,7}$	$1^{1,5}\ 1^{2,5}\ 1^{1,7}\ 1^{2,7}$
1^{2}	$1^{1,2,3}\ 2^{1,2}\ 2^{1,6}$	$1^{1,3}\ 1^{2,3}\ 2^{1,6}\ 2^{1,2}$
2^{1}	$1^{1,2,5}\ 1^{1,2,7}\ 2^{1,6}$	$1^{1,5}\ 1^{2,5}\ 1^{1,7}\ 1^{2,7}\ 2^{1,6}$

Tabela 3.9

Após Descartes Simplificada

Estado	Sucessores
$1^1\ 2^1$	$1^{1,5}\ 1^{2,5}\ 1^{1,7}\ 1^{2,7}$
1^2	$1^{1,3}\ 1^{2,3}\ 2^{1,6}\ 2^{1,2}$

Tabela 3.10

Após a Fusão e Renomeada

Estado	Sucessores
2	$2^5\ 1^5\ 2^7\ 1^7$
1	$2^3\ 1^3\ 2^6\ 2^2$

Tabela 3.11

Tabela do Codificador

	Entradas			
	00	01	10	11
1	1/011	2/010	2/011	2/110
2	1/101	1/111	2/101	2/111

Exemplo 3: Será derivado agora um código (d,k)=(1,7). O FSTD G para uma sequência (1,7) é mostrado na Figura 3.12. Sua matriz de transições A é a seguinte:

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

A capacidade de canal obtida com base na Tabela 2.1 vale, $C = 0,6793$. Observando-se o fato de que $R \leq C$, será escolhida a taxa $R = p/q = 2/3$. O grau de saída para cada estado deverá ser também $2^p = 4$.

Deve-se então tomar a terceira potência de A, cuja matriz de transições vale:

$$A^3 = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 2 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 2 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 2 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 2 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 2 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Verifica-se que o autovetor aproximado obtido $\vec{v}$ para A^3, calculado pelo algoritmo de Franaszek, quando $L = 3$, é dado por:

$$\vec{v}^T = [2\ 3\ 3\ 3\ 2\ 2\ 2\ 1]$$

Aplicando-se o conceito de fusão de estados, observa-se, da Tabela 3.12, que $w(\epsilon_6) = w(\epsilon_5)$ e que os ramos do estado 6 formam um subconjunto com relação aos ramos que saem do estado 5, isto é $6 \prec 5$. Pode-se , assim, fundir os estados 5 e 6, obtendo-se então a Tabela 3.13.

O mesmo, ocorre entre os estados 5 e 7, isto é, $7 \prec 5$ e $w(\epsilon_7) = w(\epsilon_5)$. Observa-se também que os estados 2 e 3 são equivalentes. Fundindo-se os estados (2,3) e (5,7),

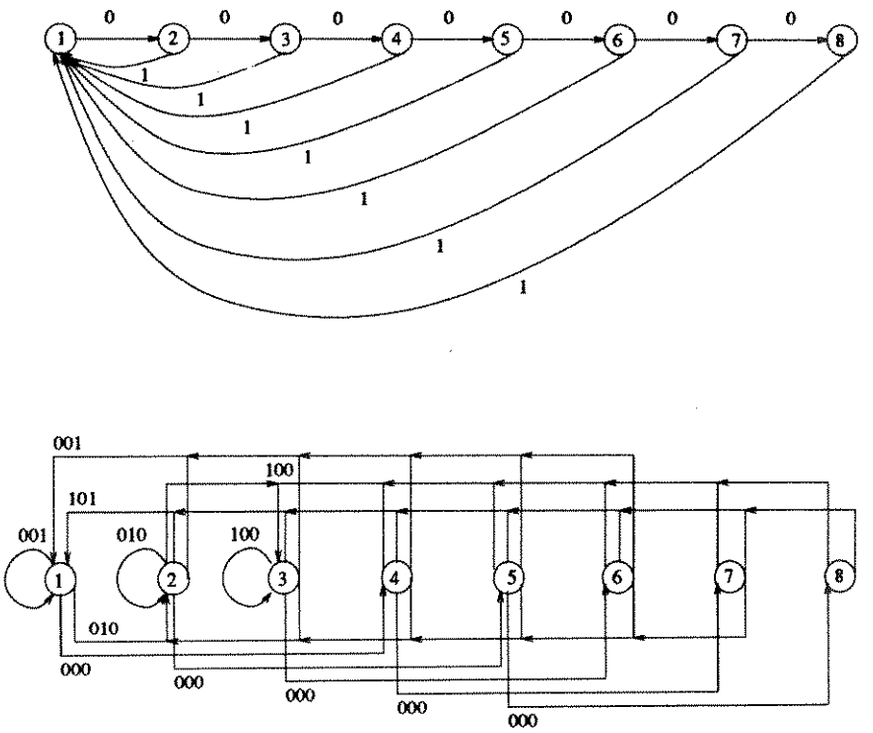


Figura 3.12: Diagrama de estados para G e G^3 com restrição (1,7)

obtém-se a Tabela 3.14.

O processo de fusão termina com a fusão dos estados 2 e 4, já que estes são equivalentes. Tem-se assim a Tabela 3.15.

Como o estado 8 não é alcançado por nenhum ramo, este poderá ser eliminado do processo. Fica-se então com um FSTD irredutível contendo apenas 3 estados, segundo a Tabela 3.16 e cujo autovetor aproximado $\vec{v}$ é igual a [2 3 2].

De acordo com a regra de divisão de estados, o único estado que pode ser aberto, num primeiro passo, é o estado 2 . A divisão do mesmo pode ser vista na Tabela 3.17, onde o estado 2 será dividido em $2^{1,2}$, e 2^3, com pesos 2 e 1, tendo como sucessores $\{1^{1,2,5}, 2^{1,3,2}, 2^{1,3,4}\}$ e $\{1^{1,2,1}, 3^{1,2,0}\}$, respectivamente. O autovetor aproximado $\vec{v}$, neste caso, é igual a [2 2 1 2].

Repetindo-se o procedimento, por mais uma vez, os estados $1^{1,2}$, $2^{1,2}$ e $3^{1,2}$ serão bifurcados de uma só vez, obtendo-se assim a Tabela 3.18. Tendo-se o autovetor aproximado $\vec{v}$ alterado finalmente para [1 1 1 1 1 1 1].

Na Tabela 3.18, observe que o grau de saída para cada estado é, no mínimo, 4. Tomando-se os estados equivalentes, pode-se fundir, num primeiro passo, os estados

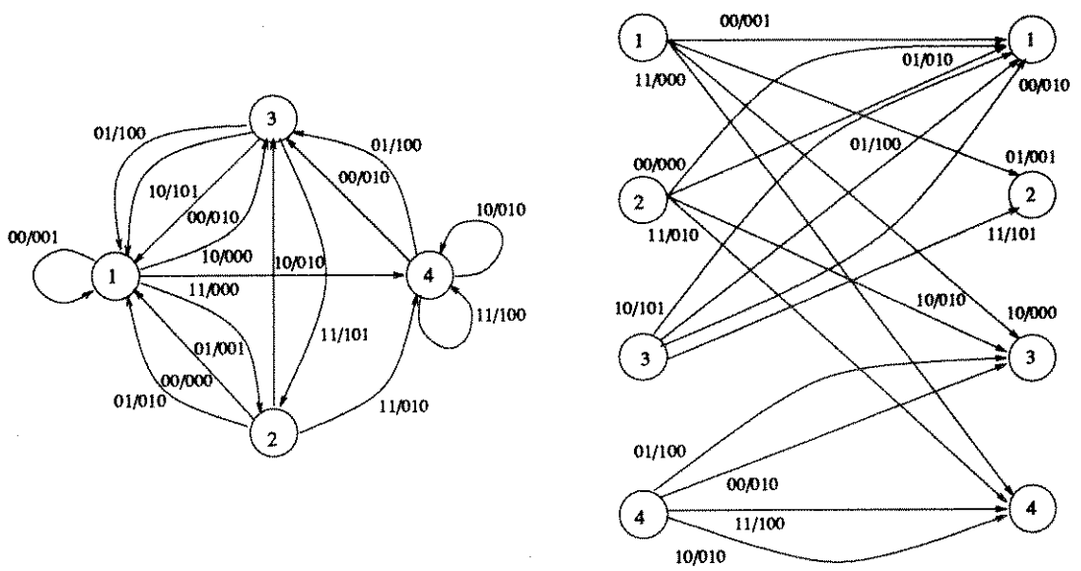


Figura 3.13: Representação do codificador (1,7) em diagrama de estados e treliça

2^1 e 3^2 num só estado denominado 2^1, da mesma forma será feita a fusão entre 2^2 e 3^1 num outro estado, denominado 2^2, conforme a Tabela 3.19. Dando-se continuidade ao processo, será possível realizar ainda mais uma fusão, entre os estados 1^1 e 2^3, obter-se-á assim a Tabela 3.20.

O codificador (1,7) , com taxa 2/3, será obtido atribuindo-se arbitrariamente rótulos da forma s/t, onde s é a entrada de 2 bits, enquanto que t é o bloco de saída de 3 bits, produzido pelo codificador, conforme mostra a Tabela 3.22. Neste caso, o emprego das técnicas de divisão e fusão possibilitou reduzir o número de estados de 8 para apenas 4.

O diagrama de estados do codificador junto com sua representação em treliça é mostrado na Figura 3.13.

Tabela 3.12

Tabela Inicial de Estados e Ramos

Estado	Sucessores
$1^{1,2}$	$1^{1,2,1}\ 2^{1,3,2}\ 4^{1,3,0}$
$2^{1,3}$	$1^{1,2,1}\ 1^{1,2,5}\ 2^{1,3,2}\ 3^{1,3,4}\ 5^{1,2,0}$
$3^{1,3}$	$1^{1,2,1}\ 1^{1,2,5}\ 2^{1,3,2}\ 3^{1,3,4}\ 6^{1,2,0}$
$4^{1,3}$	$1^{1,2,1}\ 1^{1,2,5}\ 2^{1,3,2}\ 3^{1,3,4}\ 7^{1,2,0}$
$5^{1,2}$	$1^{1,2,1}\ 1^{1,2,5}\ 2^{1,3,2}\ 3^{1,3,4}\ 8^{1,2,0}$
$6^{1,2}$	$1^{1,2,1}\ 1^{1,2,5}\ 2^{1,3,2}\ 3^{1,3,4}$
$7^{1,2}$	$1^{1,2,5}\ 2^{1,3,2}\ 3^{1,3,4}$
8^{1}	$1^{1,2,5}\ 3^{1,3,4}$

Tabela 3.13

Fusão dos Estados $5^{1,2}$ e $6^{1,2}$

Estado	Sucessores
$1^{1,2}$	$1^{1,2,1}\ 2^{1,3,2}\ 4^{1,3,0}$
$2^{1,3}$	$1^{1,2,1}\ 1^{1,2,5}\ 2^{1,3,2}\ 3^{1,3,4}\ 5^{1,2,0}$
$3^{1,3}$	$1^{1,2,1}\ 1^{1,2,5}\ 2^{1,3,2}\ 3^{1,3,4}\ 5^{1,2,0}$
$4^{1,3}$	$1^{1,2,1}\ 1^{1,2,5}\ 2^{1,3,2}\ 3^{1,3,4}\ 7^{1,2,0}$
$6^{1,2}\ 5^{1,2}$	$1^{1,2,1}\ 1^{1,2,5}\ 2^{1,3,2}\ 3^{1,3,4}\ 8^{1,2,0}$
$7^{1,2}$	$1^{1,2,5}\ 2^{1,3,2}\ 3^{1,3,4}$
8^{1}	$1^{1,2,5}\ 3^{1,3,4}$

Tabela 3.14

Fusão dos estados $2^{1,3}$ com $3^{1,3}$ e $5^{1,2}$ com $7^{1,2}$

Estado	Sucessores
$1^{1,2}$	$1^{1,2,1}\ 2^{1,3,2}\ 4^{1,3,0}$
$3^{1,3}\ 2^{1,3}$	$1^{1,2,1}\ 1^{1,2,5}\ 2^{1,3,2}\ 2^{1,3,4}\ 7^{1,2,0}$
$4^{1,3}$	$1^{1,2,1}\ 1^{1,2,5}\ 2^{1,3,2}\ 2^{1,3,4}\ 7^{1,2,0}$
$6^{1,2}\ 5^{1,2}\ 7^{1,2}$	$1^{1,2,5}\ 2^{1,3,2}\ 2^{1,3,4}$
8^{1}	$1^{1,2,5}\ 2^{1,3,4}$

Tabela 3.15

Fusão dos Estados $2^{1,3}$ com $4^{1,3}$

Estado	Sucessores
$1^{1,2}$	$1^{1,2,1}\ 2^{1,3,2}\ 2^{1,3,0}$
$4^{1,3}\ 3^{1,3}\ 2^{1,3}$	$1^{1,2,1}\ 1^{1,2,5}\ 2^{1,3,2}\ 2^{1,3,4}\ 7^{1,2,0}$
$6^{1,2}\ 5^{1,2}\ 7^{1,2}$	$1^{1,2,5}\ 2^{1,3,2}\ 2^{1,3,4}$
8^{1}	$1^{1,2,5}\ 2^{1,3,4}$

Tabela 3.16

Tabela Inicial de Estados e Ramos Renomeada e Simplificada

Estado	Sucessores
$1^{1,2}$	$1^{1,2,1}\ 2^{1,3,2}\ 2^{1,3,0}$
$2^{1,3}$	$1^{1,2,1}\ 1^{1,2,5}\ 2^{1,3,2}\ 2^{1,3,4}\ 3^{1,2,0}$
$3^{1,2}$	$1^{1,2,5}\ 2^{1,3,2}\ 2^{1,3,4}$

Tabela 3.17

Divisão do Estado 2

Estado	Sucessores
$1^{1,2}$	$1^{1,2,1}\ 2^{1,2,0}\ 2^{1,2,2}\ 2^{3,0}\ 2^{3,2}$
$2^{1,2}$	$1^{1,2,5}\ 2^{1,2,2}\ 2^{1,2,4}\ 2^{3,2}\ 2^{3,4}$
2^{3}	$1^{1,2,1}\ 3^{1,2,0}$
$3^{1,2}$	$1^{1,2,5}\ 2^{1,2,2}\ 2^{1,2,4}\ 2^{3,2}\ 2^{3,4}$

Tabela 3.18

Divisão dos Estados $1^{1,2}$, $2^{1,2}$ e $3^{1,2}$

Estado	Sucessores
1^{1}	$1^{1,1}\ 1^{2,1}\ 2^{1,0}\ 2^{2,0}$
1^{2}	$2^{1,2}\ 2^{2,2}\ 2^{3,0}\ 2^{3,2}$
2^{1}	$1^{1,5}\ 1^{2,5}\ 2^{3,2}\ 2^{3,4}$
2^{2}	$2^{1,4}\ 2^{1,2}\ 2^{2,2}\ 2^{2,4}$
2^{3}	$1^{1,1}\ 1^{2,1}\ 3^{1,0}\ 3^{2,0}$
3^{1}	$2^{1,2}\ 2^{1,4}\ 2^{2,2}\ 2^{2,4}$
3^{2}	$1^{1,5}\ 1^{2,5}\ 2^{3,2}\ 2^{3,4}$

Tabela 3.19

Fusão dos Estados 2^1 com 3^2 e 2^2 com 3^1

Estado	Sucessores
1^1	$1^{1,1}\ 1^{2,1}\ 2^{1,0}\ 2^{2,0}$
1^2	$2^{1,2}\ 2^{2,2}\ 2^{3,0}\ 2^{3,2}$
$3^2\ 2^1$	$1^{1,5}\ 1^{2,5}\ 2^{3,2}\ 2^{3,4}$
$3^1\ 2^2$	$2^{1,4}\ 2^{1,2}\ 2^{2,2}\ 2^{2,4}$
2^3	$1^{1,1}\ 1^{2,1}\ 2^{1,0}\ 2^{2,0}$

Tabela 3.20

Fusão dos Estados 1^1 com 2^3

Estado	Sucessores
$2^3\ 1^1$	$1^{1,1}\ 1^{2,1}\ 2^{1,0}\ 2^{2,0}$
1^2	$2^{1,2}\ 2^{2,2}\ 1^{1,0}\ 1^{1,2}$
2^1	$1^{1,5}\ 1^{2,5}\ 1^{1,2}\ 1^{1,4}$
2^2	$2^{1,4}\ 2^{1,2}\ 2^{2,2}\ 2^{2,4}$

Tabela 3.21

Tabela de Transições Final Reduzida

Estado	Sucessores
1^1	$1^{1,1}\ 1^{2,1}\ 2^{1,0}\ 2^{2,0}$
1^2	$2^{1,2}\ 2^{2,2}\ 1^{1,0}\ 1^{1,2}$
2^1	$1^{1,5}\ 1^{2,5}\ 1^{1,2}\ 1^{1,4}$
2^2	$2^{1,4}\ 2^{1,2}\ 2^{2,2}\ 2^{2,4}$

Tabela 3.22

Tabela do Codificador

	Entradas			
	00	01	10	11
1	1/001	2/001	3/000	4/000
2	1/000	1/010	3/010	4/010
3	1/010	1/100	1/101	2/101
4	3/010	3/100	4/010	4/100

Capítulo 4

SOFTWARE PARA A GERAÇÃO DE CÓDIGOS RLL

4.1 Descrição

O programa constitui-se de um programa principal (*main*) e de 17 subrotinas, que agindo de acordo com o algoritmo dos blocos deslizantes permite que se obtenham códigos RLL extremamente simplificados e próximos ao limitante inferior no número de estados [14].

O programa foi desenvolvido em linguagem Fortran-77, rodando em ambiente UNIX para estações de trabalho tipo SUN e possui aproximadamente 200 Kbytes de arquivo executável. Sua descrição detalhada será realizada neste capítulo [19].

4.2 Interação com o Usuário

Dentro do software SBLOCK a comunicação homem-máquina é realizada de forma bem simples, necessitando de uns poucos parâmetros, considerados básicos, para que se tenha, no final, o codificador desejado.

Para iniciar a execução do programa, o usuário deverá digitar o seguinte comando:

sblock < *enter* >

Aparecerá na tela a seguinte mensagem:

deseja entrar com numero aleatorio?(0-S,1-N)

Nesta situação, o usuário deverá digitar 0 se a resposta for sim, do contrário deverá digitar 1.

A opção 0 é útil para o controle da aleatoriedade do processo, isto é interessante nos casos onde deseja-se analisar, com mais detalhes, situações já ocorridas, quando já se tem um conhecimento prévio do número aleatório inicial. Caso essa seja a opção escolhida, aparecerá na tela a seguinte mensagem:

digite um numero aleatorio

Deve-se digitar, neste caso, um número inteiro positivo, de preferência com até sete dígitos, e em seguida **ENTER**.

Nos casos em que a opção for pela 1, um número aleatório será obtido a partir de um arquivo externo chamado RANDM.D, que possui armazenado apenas um valor, o qual a cada execução do programa, guarda um valor diferente.

Será mostrado na tela um número aleatório e em seguida, para ambas as opções aparecerá a mensagem:

digite parametros d,k (exceto k infinito)

Neste caso o usuário digitará valores coerentes para d e k. Por razões de memória e tempo de execução, sugere-se que os valores para d e k não devam exceder 10 e 11, respectivamente, com $d < k$.

Após esta fase surgirá na tela a matriz de transições A, correspondente à sequência dk digitada, bem como a capacidade do código.

A seguir, será solicitado:

digite número de bits das palavras de informação e de codificação

Deverão ser digitados dois valores inteiros, cujo quociente resulta na taxa do

código. O valor máximo para os bits codificados é 8. Valores maiores implicam num tempo de processamento elevado.

Aparecerá na tela a matriz de transições correspondente ao FSTD de A^q, onde q representa o número de bits codificados. Junto com esta matriz será mostrado o vetor de rótulos, correspondendo aos rótulos dos ramos do FSTD, de tamanho q. A forma dos rótulos exibidos é a seguinte:

$i = est_0 \quad j = est_1 \quad k = ramo$
$rotulo = rot$
onde i corresponde ao estado de partida representado por est_0, j o estado de chegada representado por est_1, k o número do ramo que vai de i para j e $rotulo$ é o rótulo do ramo em decimal.

A próxima mensagem a surgir será:

digite vetor inicial

Quando deverá ser digitado um valor inteiro correspondendo ao valor inicial para o cálculo do autovetor aproximado. Este valor inicial deverá seguir as considerações feitas para a constante L, descrita no algoritmo de Franaszek.

O autovetor obtido pelo algoritmo será mostrado na tela. Em seguida:

digite o número de iteracoes (split) e descartes

Deverão ser introduzidos dois valores, o primeiro correspondendo ao número de diferentes divisões de estado que fornecerão diferentes codificadores. O segundo valor indica o número de descartes possíveis, após o fim da fase de divisão de estados. Vale salientar que serão realizados, para cada iteração (divisão), o número de descartes digitado. Sendo assim, o número de códigos analisados será igual ao produto desses dois parâmetros: número de divisões e número de descartes.

Por fim, o programa fornecerá na tela o código dk mais simples encontrado, no formato acima descrito e terminará sua execução.

EXEMPLO: Para a obtenção de um código $dk = (1, 7)$, será mostrado na tela o seguinte:

sblock $< enter >$

deseja entrar com numero aleatorio?(0-S,1-N)

0

digite o numero aleatorio
234245

digite parametros d,k (exceto k infinito)
1 7

matriz de transicoes A

$$\begin{matrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{matrix}$$

número de estados = 8
capacidade = 0.679286

digite taxa nbits inf e nbits codif
2 3

Matriz de transicoes A^q

$$\begin{matrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 2 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 2 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 2 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 2 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 2 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{matrix}$$

número de estados=8

rotulos do FSTD

$i = 1\ j = 1\ k = 1$
$rotulo = 1$
$i = 1\ j = 2\ k = 1$
$rotulo = 2$
$i = 1\ j = 4\ k = 1$
$rotulo = 0$
$i = 2\ j = 1\ k = 1$
$rotulo = 5$
$i = 2\ j = 1\ k = 2$
$rotulo = 1$
.
.
.
$i = 7\ j = 3\ k = 1$
$rotulo = 4$
$i = 8\ j = 1\ k = 1$
$rotulo = 5$
$i = 8\ j = 3\ k = 1$
$rotulo = 4$

digite vetor inicial
3

autovetor aproximado

2 3 3 3 2 2 2 1

Verificando fusoes iniciais...

Novo autovetor

2 3 2

matriz

1 2 0
2 2 1
1 2 0

número de estados=3

rotulos do FSTD

i = 1 *j* = 1 *k* = 1
rotulo = 1
i = 1 *j* = 2 *k* = 1
rotulo = 2
i = 1 *j* = 2 *k* = 2
rotulo = 0
.
.
.
i = 2 *j* = 3 *k* = 1
rotulo = 0
i = 3 *j* = 1 *k* = 1
rotulo = 5
i = 3 *j* = 2 *k* = 1
rotulo = 2
i = 3 *j* = 2 *k* = 2
rotulo = 4

digite o numero de iteracoes (split) e descartes

1 3

realizando divisoes e fusoes de estado aguarde ...

Codificador obtido

matriz

1 1 1 1
2 0 1 1
3 1 0 0
0 0 2 2

número de estados=4

rotulos do FSTD

$i = 1\ j = 1\ k = 1$
$rotulo = 1$
$i = 1\ j = 2\ k = 1$
$rotulo = 1$
$i = 1j = 3\ k = 1$
$rotulo = 0$
.
.
.
$i = 4\ j = 3\ k = 1$
$rotulo = 4$
$i = 4\ j = 3\ k = 2$
$rotulo = 2$
$i = 4\ j = 4\ k = 1$
$rotulo = 4$
$i = 4\ j = 4\ k = 2$
$rotulo = 2$

4.3 O Programa Principal

A descrição do programa SBLOCK, será iniciada pela análise do programa principal e a seguir pelas subrotinas deste.

O fluxograma do programa principal está ilustrado nas figuras 4.1 a 4.7. O programa começa com a inicialização e declaração das variáveis a serem empregadas no transcorrer de sua implementação.

Aparece, em seguida, a opção para entrada ou não, via usuário, de um número aleatório, o qual será empregado na função de possibilitar a obtenção de diversos codificadores diferentes. Caso o usuário não deseje entrar com o número aleatório, o programa acessará um outro arquivo, denominado RANDM.D, que fornecerá um número aleatório.

Serão introduzidos, a seguir, os parâmetros relevantes ao projeto dos codificadores desejados, as restrições d e k, a taxa do código e o autovetor inicial. São calculados

a capacidade de canal para o código e o autovetor aproximado. É obtido também o vetor de rótulos para os ramos do FSTD em questão e de posse dessas informações é aplicado o princípio de conjuntos seguidores, descrito na secção 3.2, com o objetivo de simplificar o projeto do codificador.

O próximo passo no fluxograma é solicitar ao usuário o número de tentativas de bifurcações de estado, denominado *nsplit*, e o número de descartes diferentes a serem realizados após cada bifurcação, denominado *ntent*. Estes descartes são necessários para que se obtenha, no final, um codificador com grau de saída, no mínimo 2^{ninf}, onde $ninf$ é o número de bits de informação.

O procedimento de divisões ou bifurcações de estado segue a regra de construção descrita na secção 3.1, onde são verificados sob que condições deverão ser realizadas essas divisões.

Quando um estado é bifurcável é marcada a linha correspondente na matriz A^{ninf} correspondente, verificando-se, a seguir, a regra de divisão a ser empregada (caso 1 ou 2). É realizada então a bifurcação dos estados. Esse processo é realizado em duas fases: uma para alocação de ramos no primeiro sucessor (nparte=1), outra para alocação de ramos no segundo sucessor (nparte=2), segundo as figuras 4.3 e 4.4.

É realizada, posteriormente, a duplicação das colunas marcadas. Este último processo corresponde à duplicação dos ramos que chegam nos estados bifurcados.

A partir dessa fase, o processo de divisão de estados é repetido até que um autovetor com componentes todas iguais a um seja encontrado.

De posse da matriz de transições representando o FSTD após as divisões de estado. É verificado se existem ramos em excesso a serem descartados. Se existirem tais ramos esses serão descartados e a escolha para descarte seguirá um processo aleatório. Ter-se-á obtido, então um FSTD, cuja matriz de transições possuirá grau de saída exatamente 2^{ninf}.

Após essa fase será verificada a condição de equivalência entre estados. Caso existam dois ou mais estados equivalentes, estes serão fundidos num só estado. O algoritmo de fusão verifica todas as condições possíveis para fusão e retorna ao final com um novo FSTD, mais simplificado. Este processo se repete até que o número de tentativas de descartes diferentes, *ntent* seja alcançado.

Terminado este processo, o algoritmo reinicia, a partir do nó (24) do fluxograma, tornando a repetir-se *nsplit* vezes e armazenando em cada fase, o codificador mais simples até então obtido. No final será impresso na tela o codificador mais simples encontrado.

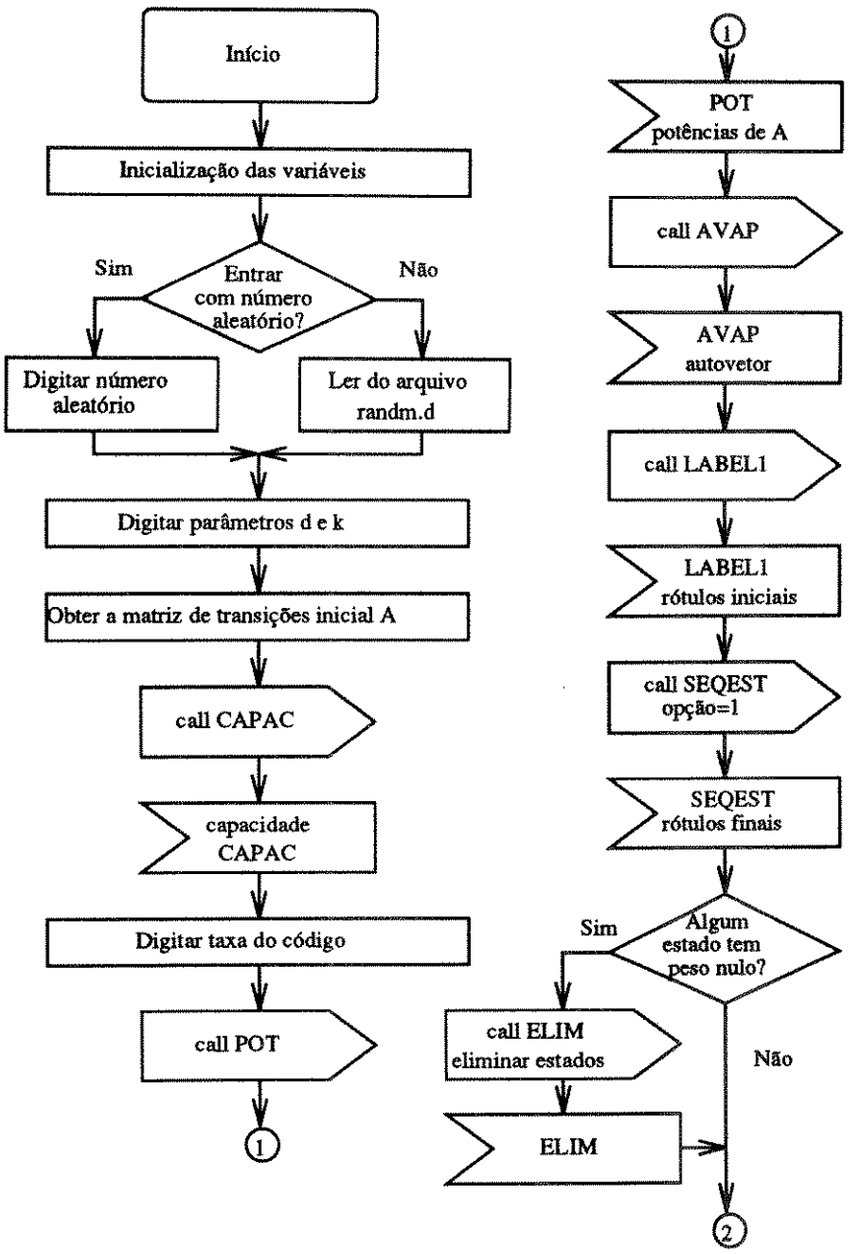


Figura 4.1: Estrutura do Programa Principal

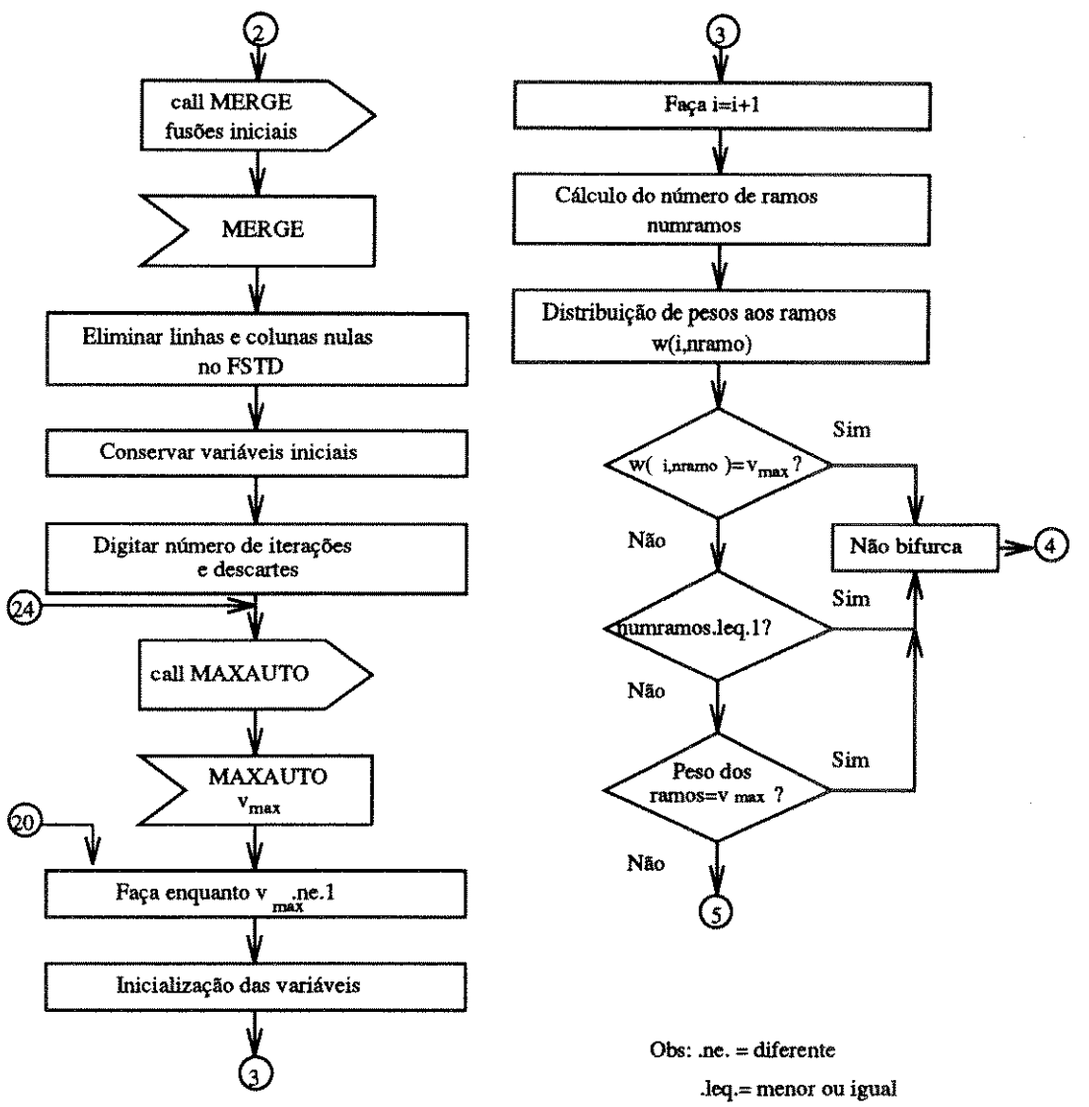


Figura 4.2: Estrutura do Programa Principal (continuação)

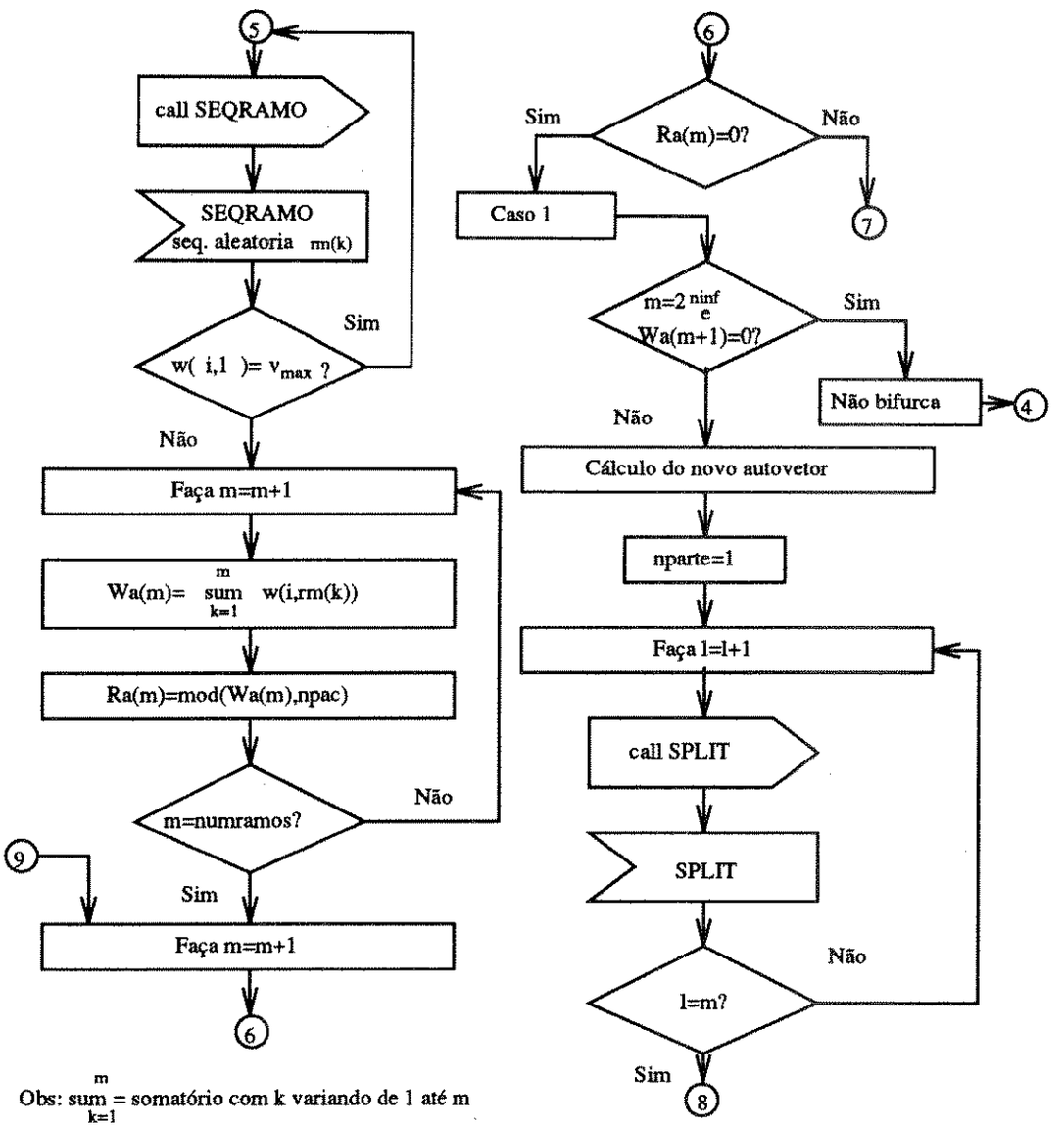


Figura 4.3: Estrutura do Programa Principal (continuação)

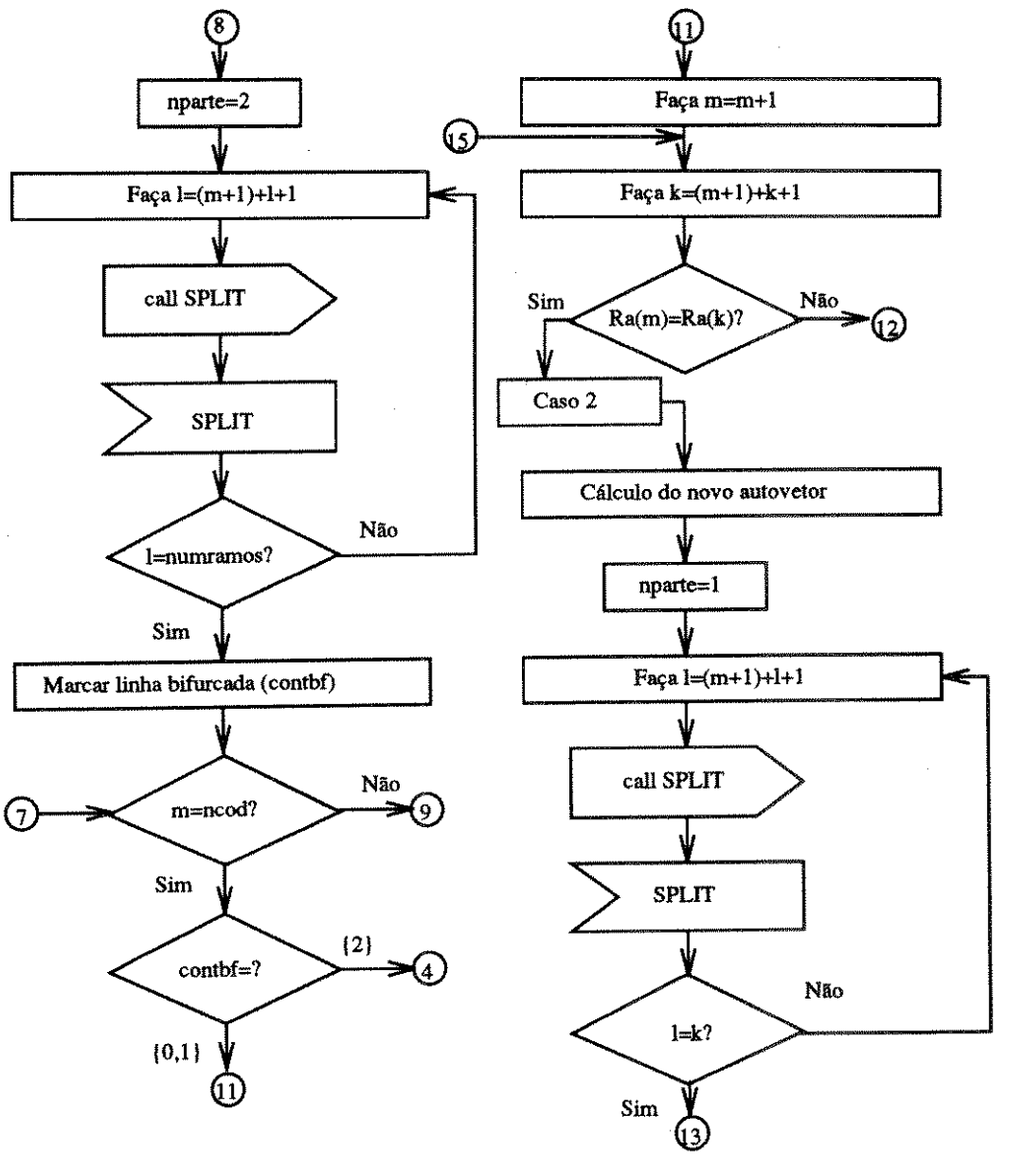


Figura 4.4: Estrutura do Programa Principal (continuação)

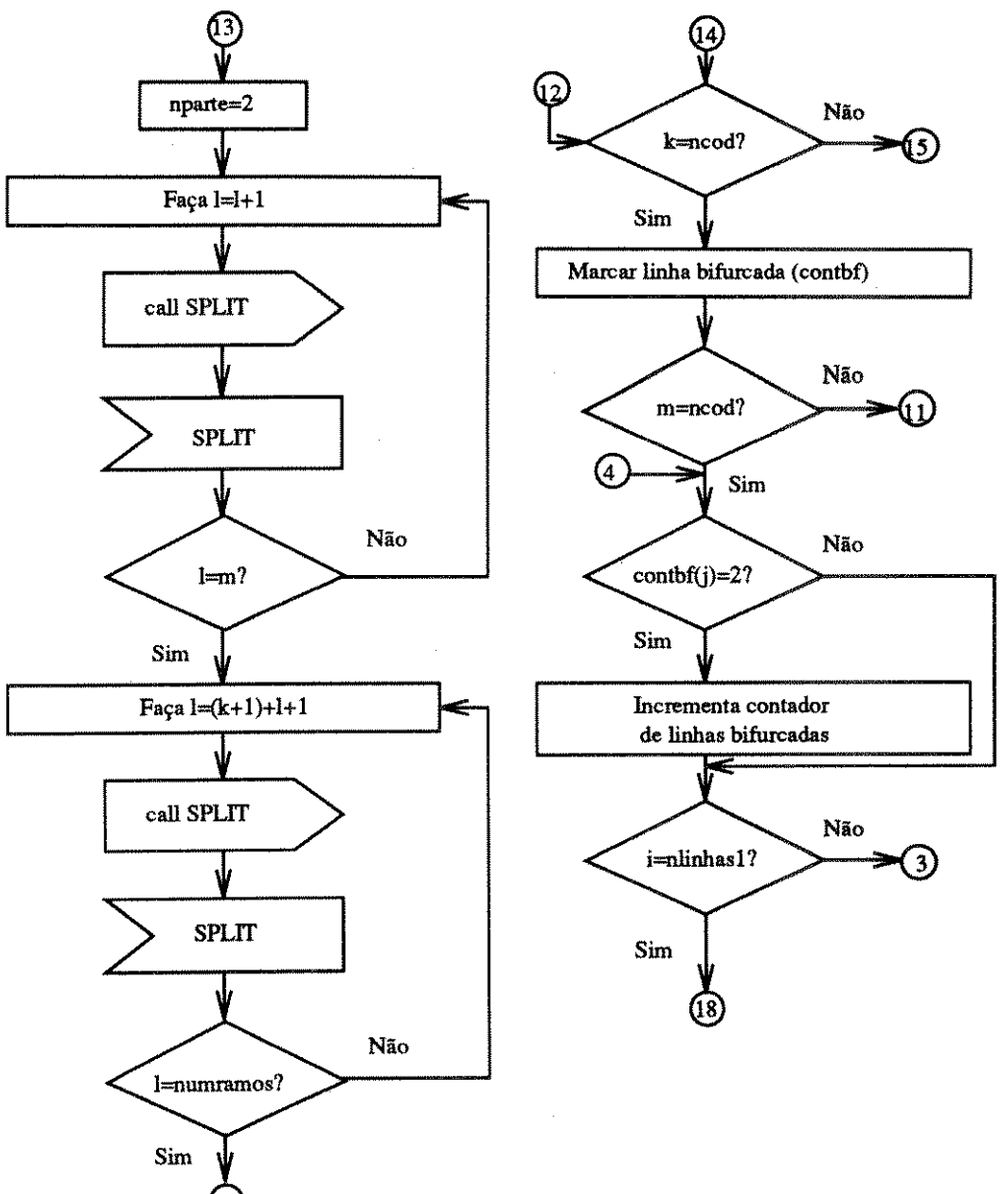


Figura 4.5: Estrutura do Programa Principal (continuação)

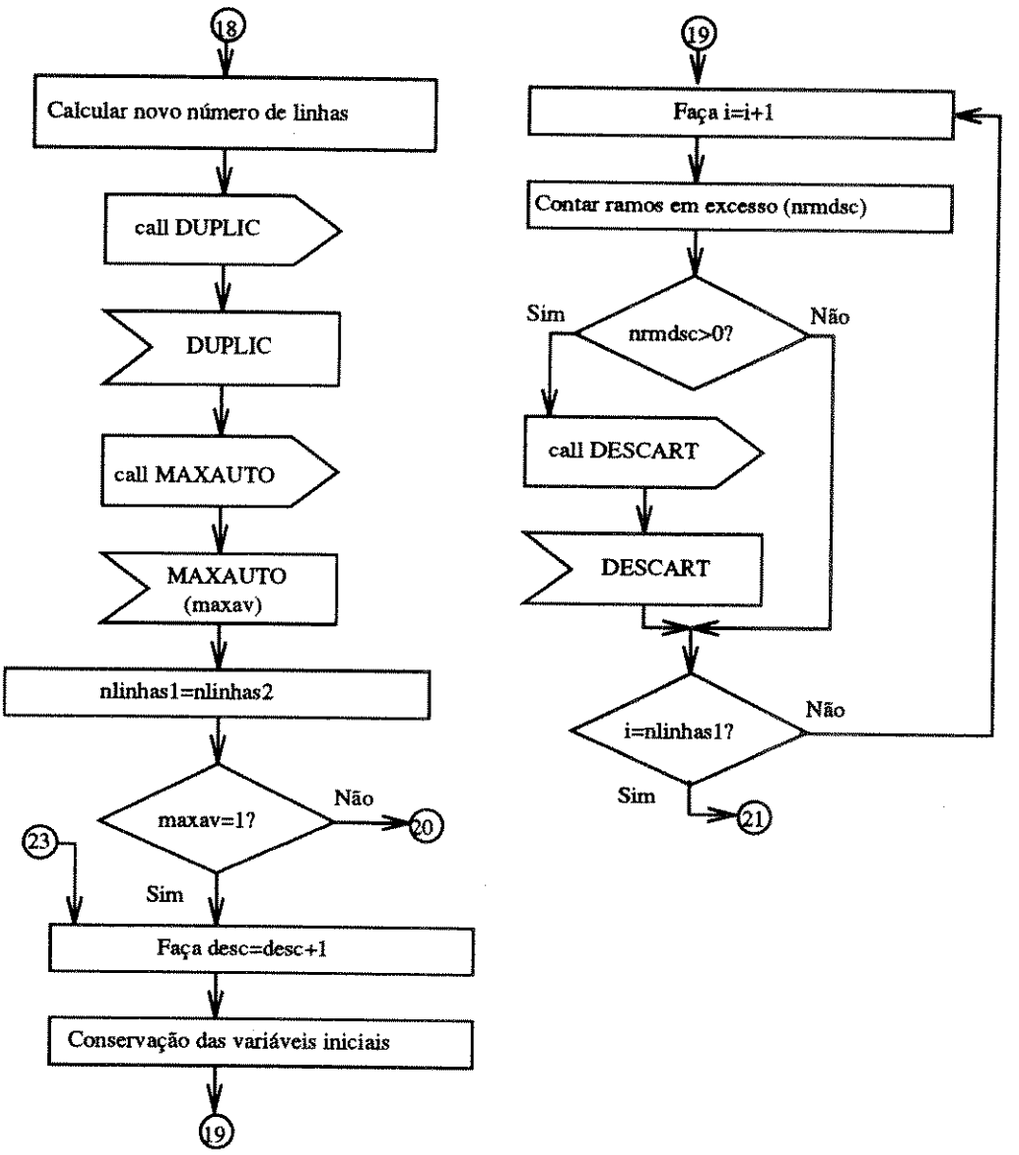


Figura 4.6: Estrutura do Programa Principal (continuação)

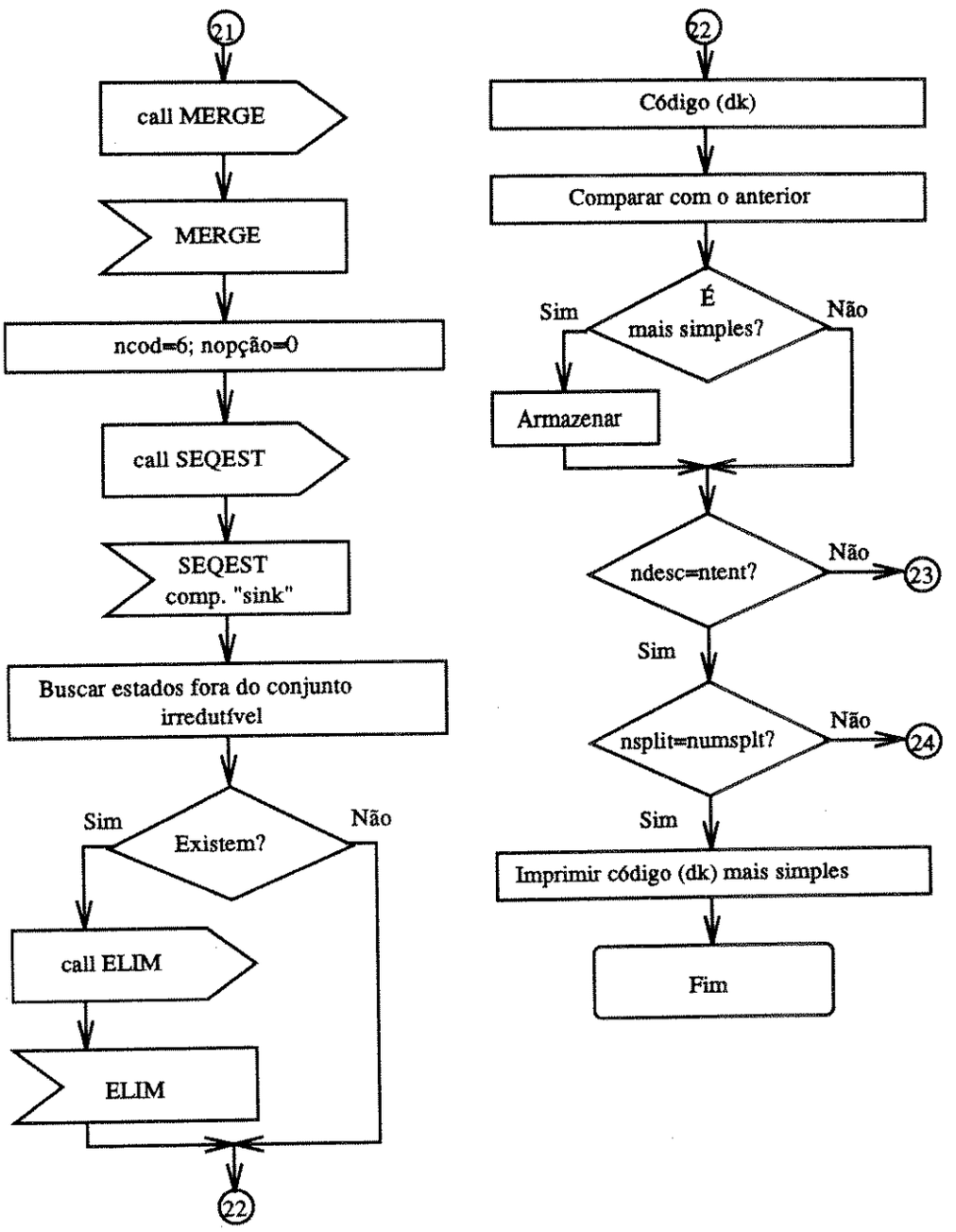


Figura 4.7: Estrutura do Programa Principal (continuação)

4.4 As Subrotinas

Por razões de organização, as rotinas serão descritas a seguir e seus diagramas no final do capítulo.

4.4.1 Subrotina CAPAC

Descrição: Esta rotina calcula a capacidade de canal de um código qualquer, a partir de sua matriz de transições inicial. O processo é iterativo e baseia-se num algoritmo para busca de autovalores [18].

Formato de Chamada:

CALL CAPAC(*resd,resk,A,itol,cap*)

Descrição dos Parâmetros:

- *resd:* Parâmetro de entrada. É a própria restrição d inicial acrescida de uma unidade;
- *resk:* Parâmetro de entrada. É a própria restrição k inicial acrescida de uma unidade. Este parâmetro é o número de linhas da matriz de transições A;
- *A:* Parâmetro de entrada. Representa a matriz de transições do FSTD inicial;
- *itol:* Parâmetro de entrada. Constante utilizada como tolerância para um cálculo da Capacidade de forma precisa.
- *cap:* Parâmetro de saída. Retorna com o valor da Capacidade já calculada.

Comentários: O algoritmo funciona da seguinte maneira:

Algoritmo 4.1 (Capacidade de Canal) *:*

1 - Escolha um vetor inicial $vet_0(i)$*, faça* $itol = 10^{-6}$ *e* $k = 1$*;*

2 - $vet_k(j) = A(i,j)vet_{k-1}(j)$*;*

3 - $\lambda_k = max(vet_k)$*;*

4 - Faça $tol = |(\lambda_k - \lambda_{k-1})/\lambda_{k-1}|$*;*

5 - Se $tol \leq itol$ *então tome* $Cap = log_2\lambda_k$*, senão faça* $vet_k(j) = vet_k(j)/\lambda_k$*;*

6 - Faça $k = k + 1$ *e volte para 2.*

4.4.2 Subrotina POT

Descrição: Esta rotina calcula as potências de uma matriz qualquer.

Formato de Chamada:

CALL POT(*resk*,*A*,*ncod*,*C*)

Descrição dos Parâmetros:

- *resk:* Parâmetro de entrada. É a própria restrição k inicial acrescida de uma unidade. Este parâmetro é o número de linhas da matriz de transições A;
- *A:* Parâmetro de entrada. Representa a matriz de transições do FSTD inicial;
- *ncod:* Parâmetro de entrada. Representa o número de bits da palavra-código. Neste caso é utilizada como a potência na qual vai ser elevada a matriz A.
- *C:* Parâmetro de saída. Retorna com a potência da matriz A, já calculada.

4.4.3 Subrotina AVAP

Descrição: Esta rotina calcula o autovetor aproximado a partir do algoritmo de Franaszek [12].

Formato de Chamada:

CALL AVAP(*numpac*,*resk*,*vet0*,*C*,*AVA*)

Descrição dos Parâmetros:

- *numpac:* Parâmetro de entrada. Indica o número de palavras-código;
- *resk:* Parâmetro de entrada. É a própria restrição k inicial acrescida de uma unidade. Este parâmetro é igual ao número de linhas da matriz de transições inicial A;
- *vet0:* Parâmetro de entrada. Valor inicial para o cálculo do autovetor aproximado;
- *C:* Parâmetro de entrada. Representa a potência da matriz A, já descrito anteriormente;
- *AVA:* Parâmetro de saída. Retorna com o autovetor aproximado já calculado.

Comentários: Conforme descrito no capítulo 3 a rotina inicia por um valor inicial qualquer e segue realizando iterações até que um autovetor aproximado seja encontrado.

4.4.4 Subrotina LABEL1

Descrição: Esta rotina realiza a atribuição de rótulos de comprimento 1, ao FSTD inicial de uma sequência dk, segundo a expressão 2.5.

Formato de Chamada:

CALL LABEL1(*resk*,*A*,*label*)

Descrição dos Parâmetros:

- *resk* Parâmetro de entrada. É a própria restrição k inicial acrescida de uma unidade. Este parâmetro é igual ao número de linhas da matriz de transições A;
- *A:* Parâmetro de entrada. Representa a matriz de transições do FSTD inicial;
- *label(i,j):* Parâmetro de saída. É um vetor de dimensão 2, onde i representa o estado de partida e j o estado de chegada. Retorna com o rótulo dos ramos de saída que ligam o estado i ao estado j no FSTD inicial.

Comentários: A rotina LABEL1 serve apenas para a obtenção de códigos dk. Caso deseje-se utilizar este programa para a obtenção de um outro tipo de código deve-se alterá-la para compor o diagrama de estados do novo código.

4.4.5 Subrotina SEQEST

Descrição: Esta rotina possui duas funções básicas: calcula os rótulos do FSTD de A^{ninf} e auxilia na determinação das componentes irredutíveis de um FSTD qualquer.

Formato de Chamada:

CALL SEQEST(*resk,ncod,label,rotul,opcao,CIRR,nestirr*)

Descrição dos Parâmetros:

- *resk:* Parâmetro de entrada. É a própria restrição k inicial acrescida de uma unidade;
- *ncod:* Parâmetro de entrada. Representa o número de bits da palavra-código;
- *label(i,j):* Parâmetro de entrada. É o vetor de rótulos do FSTD inicial, calculado na rotina LABEL1;
- *rotul(i,j,k)* Parâmetro de saída. É um vetor de dimensão 3, onde i representa o estado de partida, j o estado final e k o número do ramo que vai de i para j. Retorna com os rótulos do FSTD de A^{ncod};
- *opcao:* Parâmetro de entrada. Seleciona a tarefa a ser executada pela subrotina. Com valor 1 determina os rótulos e valor 0 determina as componentes irredutíveis de um FSTD. A opção 0 é dispensável para a determinação de sequências dk;
- *CIRR:* Parâmetro de saída. É um vetor de dimensão 3 contendo os estados da componente irredutível mais simples de um FSTD;
- *nestirr:* Parâmetro de saída. Retorna com o número de estados da componente irredutível mais simples de um FSTD.

Comentários: O procedimento de geração de rótulos consiste em gerar todas as sequências de estado de comprimento igual a $ncod + 1$ obtidas a partir do FSTD inicial e armazenadas num vetor $SEQ(i0, pos)$, onde a componente $i0$ indica o estado de partida e pos indica a posição do estado armazenado, possuindo como valor máximo o comprimento da palavra-código $ncod$ acrescido de uma unidade. Estas sequências podem ser vistas como sendo caminhos de comprimento $ncod$, percorridos neste FSTD.

Em seguida, tomando-se essas sequências, é verificada a existência de conexões entre os estados. Isto é realizado a partir de um vetor denominado $label(m, n)$, obtido

na rotina LABEL1. Caso exista conexão entre dois estados l_1 e l_2 quaisquer (ou seja, $label(l_1, l_2) = 0, 1$), o rótulo armazenado em *label* é transferido para um outro vetor de rótulos, denominado *rotul(i,j,k)*. Toma-se então a próxima conexão, entre l_2 e um outro estado qualquer l_3, e repete-se o procedimento até que se tenha percorrido toda a sequência de estados, quando será obtida uma componente do vetor de rótulos ligando os estados i a j. Será tomada uma nova sequência para análise.

Caso não exista conexão entre dois estados quaisquer (i.e. $label(m, n) = 2$) o processo será abortado e uma nova sequência de estados será tomada. O processo termina com a obtenção de todos os rótulos dos ramos do FSTD de $A^n cod$. Todo esse procedimento de análise de conexões e atribuição de rótulos é realizado numa outra rotina, auxiliar de SEQEST, denominada LABEL2.

Seja por exemplo a obtenção de um rótulo ligando os estados 2 e 1 numa sequência $dk = (1, 7)$, segundo o exemplo 3 do capítulo 3. Uma sequência de estados possível dentro do FSTD de A^3 é a seguinte: 2 3 1.

O vetor $SEQ(i_0, npos)$ terá como valores armazenados:

$SEQ(i_0, 1) = SEQ(2, 1) = 2$
$SEQ(i_0, 2) = SEQ(2, 2) = 3$
$SEQ(i_0, 3) = SEQ(2, 3) = 4$
$SEQ(i_0, 4) = SEQ(2, 4) = 1$

A partir da rotina LABEL2 será verificada a conexão entre cada um desses estados e, em caso positivo, será atribuído o rótulo correspondente. A formação exata do rótulo é mostrada na Tabela 4.1. Verifica-se que, ao final, o rótulo correspondente será $rotul(2, 1, 1) = 1$, em decimal.

Tabela 4.1

Exemplo de Formação de Rótulos

Conexão entre estados	Rótulo	Valor Armazenado (em decimal)
$SEQ(2,1) - SEQ(2,2) = 2 - 3$	$label(2,3) = 0$	$rotul(2,1,1) = 0$
$SEQ(2,2) - SEQ(2,3) = 3 - 4$	$label(3,4) = 0$	$rotul(2,1,1) = 0$
$SEQ(2,3) - SEQ(2,4) = 4 - 1$	$label(4,1) = 1$	$rotul(2,1,1) = 1$

Na busca por componentes irredutíveis (opção = 0), emprega-se esta rotina inicialmente para a geração de sequências de estados, seguindo-se o mesmo proce-

dimento descrito anteriormente, porém sem a preocupação de atribuição de rótulos.

Será chamada então uma outra rotina, denominada AUXIRR a qual tomará as sequências de estado geradas e fornecerá as componentes irredutíveis (se existirem) do FSTD atual. Uma descrição mais completa deste processo será realizado mais adiante.

Dentre as componentes irredutíveis encontradas, será escolhida a componente mais simples, a qual será tomada como o novo FSTD para divisões de estados e posteriores simplificações do FSTD.

4.4.6 Subrotina MERGE

Descrição: A rotina MERGE realiza a fusão entre dois estados quaisquer num dado FSTD. Esta fusão pode ocorrer devido a dois fatores: os estados são equivalentes ou os estados satisfazem as regras de inclusão de estados mencionadas na secção 3.2.

Formato de Chamada:

CALL MERGE(*nlinhas,AVA,B,rotul*)

Descrição dos Parâmetros:

- *nlinhas:* Parâmetro de entrada/saída. Representa o número de estados do FSTD onde se deseja aplicar a fusão de estados. Retorna com o número de estados do FSTD simplificado após as fusões;
- *AVA:* Parâmetro de entrada/saída. É um arranjo contendo o autovetor aproximado do FSTD, onde se deseja aplicar a fusão de estados. Retorna com um autovetor simplificado após as fusões;
- *B:* Parâmetro de entrada/saída. Representa a matriz de transições do FSTD antes das fusões. Retorna com uma matriz representando o novo FSTD;
- *rotul(i,j,k):* Parâmetro de entrada/saída. É o vetor de rótulos, de dimensão 3, onde i representa o estado de partida, j o estado final e k o número do ramo que vai de i para j. Retorna com os rótulos do FSTD após a realização das fusões.

Comentários: A rotina faz uso da noção de estados equivalentes e da regra de inclusão de estados. Esta rotina consiste inicialmente em tomar-se um estado fixo

($i = 1$) e um segundo estado ($j = 2$) como prováveis a serem fundidos.

Se os estados i e j satisfizerem a alguma das condições citadas acima, estes serão fundidos em um só, eliminando-se o estado j e redirecionando-se para i todos os ramos que antes chegavam em j, segundo o algoritmo de fusões descrito no capítulo 3. Toma-se, a seguir, o próximo estado para análise, incrementando-se j. Caso contrário, se as condições não forem satisfeitas, incrementa-se j e aplica-se novamente a regra. O procedimento continua até que todos os estados tenham sido verificados quando então um novo estado $i + 1$ será fixado e as análises reiniciadas.

Ao finalizar esta parte ter-se-á realizado o primeiro passo da fusão. O algoritmo reiniciará indo para o segundo passo testando $i = 1$ e $j = 2$ novamente e repetindo-se todo o procedimento descrito acima até que não hajam mais fusões a serem realizadas. Ter-se-á, por fim, o novo FSTD reduzido após sucessivas fusões de estado.

4.4.7 Subrotina NUMRAMOS

Descrição: Esta rotina calcula o número de ramos de saída de um estado qualquer no FSTD sob análise.

Formato de Chamada:

CALL NUMRAMOS(*nlinhas,numest,B,nramos*)

Descrição dos Parâmetros:

- *nlinhas:* Parâmetro de entrada. Representa o número de estados do FSTD, ou o número de linhas da matriz B;
- *numest:* Parâmetro de entrada. Indica o número do estado onde se deseja calcular o número de ramos de saída;
- *B:* Parâmetro de entrada. Representa a matriz de transições do FSTD atual;
- *nramos:* Parâmetro de saída. Retorna com o número de ramos partindo do estado *numest*.

4.4.8 Subrotina ESCR

Descrição: Esta rotina é utilizada para a formatação dos dados de saída.

Formato de Chamada:

CALL ESCR(*nlinhas,B,rotul*)

Descrição dos Parâmetros:

- *nlinhas:* Parâmetro de entrada. Representa o número de estados do FSTD, ou o número de linhas da matriz B;
- *B:* Parâmetro de entrada. Representa a matriz de transições do FSTD atual;
- *rotul(i,j,k):* Parâmetro de entrada. É o vetor de rótulos, de dimensão 3, onde i representa o estado de partida, j o estado final e k o número do ramo que vai de i para j.

4.4.9 Subrotina MAXAUTO

Descrição: Esta é uma pequena rotina utilizada para calcular o maior valor entre vários numa sequência qualquer. Para esta aplicação ela é empregada para o cálculo da maior componente do autovetor aproximado.

Formato de Chamada:

CALL MAXAUTO(*nlinhas,seq,vmax*)

Descrição dos Parâmetros:

- *nlinhas:* Parâmetro de entrada. Representa o número de elementos na sequência seq;
- *seq:* Parâmetro de entrada. É a sequência de dados a partir da qual deseja-se calcular o maior valor que contém armazenado o autovetor aproximado;
- *vmax:* Parâmetro de saída. Retorna com a maior componente entre aqueles do vetor *seq*.

4.4.10 Subrotina SEQRAMO

Descrição: A rotina SEQRAMO é empregada para gerar sequências aleatórias de números. Neste caso, os números representam os ramos que partem de um estado qualquer.

Formato de Chamada:

CALL SEQRAMO(*nelm,naleat,seqalet*)

Descrição dos Parâmetros:

- *nelm:* Parâmetro de entrada. Representa o número de ramos que partem do estado sob análise;
- *naleat:* Parâmetro de entrada. É um número aleatório qualquer;
- *seqalet(i):* Parâmetro de saída. Retorna com um vetor contendo a sequência aleatória.

Comentários: Esta rotina é importante para garantir a aleatoriedade do processo de divisões de estado, conseguindo com isso, a obtenção de diversos codificadores diferentes.

4.4.11 Subrotina SELRAMO

Descrição: A rotina SELRAMO é empregada na escolha aleatória de um elemento qualquer dentro de um conjunto. É utilizada como auxiliar da rotina SEQRAMO, a qual fornece o conjunto de onde vai ser retirado o elemento aleatório.

Formato de Chamada:

CALL SELRAMO(*nelm,s,nrmesc*)

Descrição dos Parâmetros:

- *nelm:* Parâmetro de entrada. Representa o número de elementos da sequência, de onde vai ser retirado o elemento aleatório;
- *s:* Parâmetro de entrada. É um número aleatório qualquer;
- *nrmesc:* Parâmetro de saída. Fornece o número do ramo aleatoriamente escolhido.

4.4.12 Subrotina LABEL2

Descrição: Esta rotina serve como auxiliar da rotina SEQEST na determinação dos rótulos finais no FSTD de A^{ncod}.

Formato de Chamada:

CALL LABEL2(*SEQ,est0,ncod,label,rotul*)

Descrição dos Parâmetros:

- *SEQ(est0,pos):* Parâmetro de entrada. É um vetor, de dimensão 2 que armazena as sequências de estado já descrito anteriormente (ver subrotina SEQEST). A componente $est0$ indica o estado de partida na sequência e *pos* indica a posição do estado armazenado, possuindo como valor máximo o comprimento da palavra-código *ncod* acrescido de uma unidade;
- *est0:* Parâmetro de entrada. Indica o estado de partida na sequência de estados;
- *ncod:* Parâmetro de entrada. Representa o número de bits da palavra-código;
- *label(i,j):* Parâmetro de saída. É um vetor de dimensão 2, onde i representa o estado de partida e j o estado de chegada. Retorna com os rótulos dos ramos que ligam o estado i ao estado j no FSTD inicial;
- *rotul(i,j,k):* Parâmetro de saída. É um vetor de dimensão 3, onde i representa o estado de partida, j o estado final e k o número do ramo que vai de i para j. Retorna com os rótulos do FSTD de A^{ncod};

Comentários: A tarefa da rotina LABEL2 é a verificação da conexão entre estados, a partir da sequência de estados obtida na rotina SEQEST.

Caso exista conexão entre estados, ou seja $label(i,j) = 0,1$, o rótulo correspondente armazenado em $label(i,j)$ é convertido para o formato decimal e, a seguir armazenado no vetor $rotul(i,j,k)$, que conterá, no final, o rótulo correspondente.

4.4.13 Subrotina AUXIRR

Descrição: A rotina AUXIRR obtém as componentes irredutíveis, se existirem, de um FSTD qualquer.

Formato de Chamada:

CALL AUXIRR(*ir0,nlinhas,comp,rotul,A,nlinhas2,cirr*)

Descrição dos Parâmetros:

- *ir0:* Parâmetro de entrada. Indica o estado a ser analisado como componente ou não do conjunto irredutível;
- *nlinhas:* Parâmetro de entrada. Representa o número de estados do FSTD, ou o número de linhas da matriz B;
- *comp:* Parâmetro de entrada. Representa o comprimento da sequência de estados acrescida de uma unidade;
- *rotul(i,j,k):* Parâmetro de entrada. É um vetor de dimensão 3 que armazena o rótulo do ramo que sai do estado i e vai para o estado j, sendo que k representa o número do ramo que tem essa característica;
- *A:* Parâmetro de entrada. Representa o vetor sequência de estados, já descrito anteriormente;
- *nlinhas2:* Parâmetro de saída. Indica o número de estados da componente irredutível de A^{ncod};
- *cirr:* Parâmetro de saída. Armazena os estados componentes da classe irredutível.

Comentários: O esquema para busca de componentes irredutíveis faz uso de um algoritmo para a busca de classes essenciais sugerido por Seneta [15].

A rotina em si, toma um estado de partida $ir0$, a ser analisado. Tomando-se a primeira sequência de estado válida, gerada na rotina SEQEST que tem $ir0$ como estado inicial, armazenam-se os estados, desta sequência componentes, num vetor auxiliar e em seguida é tomada uma outra sequência, também gerada a partir de $ir0$. É verificado então se os estados dessa nova sequência estão contidos, ou não no vetor auxiliar. Se algum estado não estiver presente no conjunto, este será acrescentado ao vetor. O processo se repete até que todas as sequências de estado válidas tenham sido verificadas, quando será tomado um novo valor, correspondendo a um outro estado

inicial, para $ir0$. O conceito de sequência de estado foi discutido na secção 4.4.5, quando da abordagem da subrotina SEQEST.

Caso o vetor auxiliar final contenha todos os estados do FSTD de origem, o estado $ir0$ que o gerou não fará parte de uma possível componente irredutível do FSTD, do contrário, se não contiver todos os estados, este será armazenado no vetor $cirr(i)$.

O procedimento se repete até que todas as sequências de estado possíveis tenham sido verificadas. No final será obtido um conjunto particular de estados armazenados em $cirr(i)$, componentes de um FSTD irredutível.

Se o conjunto particular $cirr(i)$ contiver todos os estados iniciais então o FSTD sob análise não será redutível.

Tomando-se a matriz de transições do FSTD da figura 4.8 *a*) e observando-se as sequências de estado possíveis, geradas a partir de cada estado, pode-se concluir que os estados 2.3 formam uma componente irredutível do FSTD inicial, cuja matriz de transições vale:

$$\begin{bmatrix} 2 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

As sequências de estado válidas estão mostradas na Tabela 4.2.

Tabela 4.2

Sequências de Estado Possíveis

	Estado de Partida		
	1	2	3
Sequências de estado válidas	111 112 122 123	222 223 232 233	322 323 332 333
Conjunto formado	{1.2.3}	{2.3}	{2.3}

Eliminando-se o estado 1 do processo obtém-se o FSTD da figura 4.8 *b*).

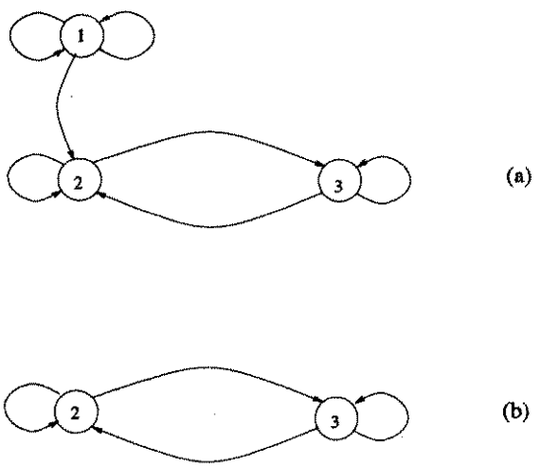


Figura 4.8: Redução de um FSTD em Componentes Irredutíveis

4.4.14 Subrotina ELIM

Descrição: Esta rotina realiza a eliminação de um estado qualquer no FSTD sob análise.

Formato de Chamada:

CALL ELIM(*nelim*,*AVA*,*B*,*nlinhas2*,*rotul*)

Descrição dos Parâmetros:

- *nelim:* Parâmetro de entrada. Indica o número do estado a ser eliminado;
- *AVA:* Parâmetro de entrada/saída. É o autovetor aproximado. Retorna com um novo autovetor, obtido após as eliminações;
- *B:* Parâmetro de entrada/saída. Representa a matriz de transições do FSTD sob análise. Retorna com a nova matriz, após as eliminações;
- *nlinhas2:* Parâmetro de entrada/saída. Representa o número de linhas da matriz de transições. Retorna com um novo número de linhas, após as eliminações;
- *rotul:* Parâmetro de entrada/saída. É um vetor de dimensão 3 que armazena o rótulo do ramo que sai do estado i e vai para o estado j, sendo que k representa o número do ramo que tem essa característica. Realizadas as eliminações retorna com um novo vetor de rótulos.

4.4.15 Subrotina SPLIT

Descrição: Esta rotina realiza as bifurcações de estado, de acordo com o algoritmo para bifurcações de estado descrito na secção 3.1.1.

Formato de Chamada:

CALL SPLIT(*ajst,nlinhas,nestbf,nraleat,nparte,rotul1,B1,B2,rotul2,nlin*)

Descrição dos Parâmetros:

- *ajst:* Parâmetro de entrada. É um parâmetro auto-ajustável utilizado para distribuição de ramos entre os estados descendentes;
- *nlinhas2:* Parâmetro de entrada. Representa o número de linhas da matriz de transições $B1$;
- *nestbf:* Parâmetro de entrada. Representa o número do estado bifurcável;
- *nraleat:* Parâmetro de entrada. Indica o número do ramo que vai ser alocado para uma bifurcação de E_i;
- *nparte:* Parâmetro de entrada. Representa a fase da bifurcação. Pode tomar dois valores: (1) indica fase de alocação de ramos em E_i^1, (2) indica fase de alocação de ramos em E_i^2;
- *rotul1(i,j,k):* Parâmetro de entrada. É um vetor de dimensão 3 que armazena o rótulo do ramo que sai do estado i e vai para o estado j, sendo que k representa o número do ramo que tem essa característica.
- *B1(i,j):* Parâmetro de entrada. Representa a matriz de transições do FSTD bifurcável;
- *B2(i,j):* Parâmetro de saída. Retorna com nova matriz de transições bifurcada;
- *rotul2(i,j,k):* Parâmetro de saída . Representa o vetor de rótulos, com estrutura similar ao vetor $rotul1(i,j,k)$, do FSTD bifurcado;
- *nlin:* Parâmetro de entrada. É, na verdade, um novo valor para a linha corrente, após a bifurcação.

Comentários: A rotina funciona em conjunto com o programa principal que seleciona a linha a ser bifurcável. Ela realiza, basicamente, a distribuição de ramos entre os estados sucessores do estado de origem E_i.

Dentro do programa principal é selecionado um caso entre dois possíveis, segundo a regra de divisão de estados mencionada na secção 3.1.1. De acordo com o caso

obtido será adotado um critério distinto para a distribuição de ramos. Concluída essa fase, a rotina SPLIT será chamada.

Na rotina a distribuição é feita em duas etapas: uma para distribuição dos ramos que partirão do primeiro sucessor (nparte=1), outra para a distribuição dos ramos que partirão do segundo sucessor (nparte=2).

No final do processo serão criadas duas novas linhas na matriz de transições representando dois novos estados no FSTD.

4.4.16 Subrotina RDM

Descrição: Esta rotina é empregada para a geração de números aleatórios.

Formato de Chamada:

CALL RDM(*iy,yfl*)

Descrição dos Parâmetros:

- *iy:* Parâmetro de entrada. É um número qualquer utilizado como valor inicial para o cálculo do número aleatório;
- *yfl:* Parâmetro de saída. Representa o número aleatório gerado.

Comentários: Os números aleatórios gerados obedecem a uma função de distribuição de probabilidades uniforme no intervalo [0,1] [20][21].

4.4.17 Subrotina DUPLIC

Descrição: Esta rotina é utilizada como auxiliar do algoritmo de divisão de estados. Seu objetivo é a duplicação dos ramos que seguem para os estados sucessores, segundo a regra de divisões de estado mencionada na secção 3.1.1.

Formato de Chamada:

CALL DUPLIC(*nlinhas1,nlinhas2,contbf,B1,rotul1,B2,rotul2*)

Descrição dos Parâmetros:

- *nlinhas1:* Parâmetro de entrada. Indica o número de estados no novo FSTD bifurcado;
- *nlinhas2:* Parâmetro de entrada. Indica o número de estados no FSTD antes das bifurcações;
- *contbf(i):* Parâmetro de entrada. Vetor utilizado para identificar se o estado foi bifurcado ($contbf(i) = 2$), ou não ($contbf(i) = 0, 1$);
- *B1(i,j):* Parâmetro de entrada. Representa a matriz de transições do FSTD antes das duplicações.
- *rotul1(i,j,k):* Parâmetro de entrada. É um vetor de dimensão 3 que armazena o rótulo do ramo que sai do estado i e vai para o estado j, sendo que k representa o número do ramo que tem essa característica. Armazena os rótulos antes das duplicacações;
- *B2:* Parâmetro de saída. Representa a matriz de transições do FSTD após as duplicações.
- *rotul2(i,j,k):* Parâmetro de saída. É o vetor de rótulos obtido após as duplicações de colunas na matriz, ou ramos que chegam nos estados sucessores.

4.4.18 Subrotina DESCART

Descrição: A rotina DESCART é empregada para o descarte de ramos em excesso que partem de um estado qualquer. Este procedimento é necessário para que se tenha garantido que de cada estado partam exatamente 2^{ninf} ramos, onde $ninf$ é o número de bits de informação a serem codificados.

Formato de Chamada:

CALL DESCART(*nlinhas,nlin,nrmdsc,B,rotul,naleat*)

Descrição dos Parâmetros:

- *nlinhas:* Parâmetro de entrada. Representa o número de linhas da matriz de transições B;
- *nlin:* Parâmetro de entrada. Indica a linha, onde deseja-se realizar o descarte de ramos em excesso;
- *nrmdsc:* Parâmetro de entrada. Indica o número de ramos que partem do estado *nlin* e que devem ser descartados;
- *B:* Parâmetro de entrada/saída. Representa a matriz de transições do FSTD sob análise. Retorna com a nova matriz, após as eliminações;
- *rotul:* Parâmetro de entrada/saída. É um vetor de dimensão 3 que armazena o rótulo do ramo que sai do estado i e vai para o estado j, sendo que k representa o número do ramo que tem essa característica. Realizadas as eliminações de ramos em excesso, retorna com um novo vetor de rótulos.
- *naleat:* Parâmetro de entrada/saída. Armazena um número aleatório empregado para a escolha dos ramos a serem descartados.

Comentários: O funcionamento da rotina DESCART é simples. O programa principal toma a linha corrente e verifica a necessidade de descarte observando a condição do grau de saída (2^{ninf}). Em caso positivo, é acionada a rotina que tomará a linha onde os ramos deverão ser descartados através da variável *nlin*.

A rotina DESCART aciona uma outra subrotina denominada SELRAMO (já descrita anteriormente) que fará uma escolha aleatória dos ramos a serem descartados. Esses ramos são então buscados e eliminados da matriz de transições $B(i,j)$ e do vetor de rótulos $rotul(i,j,k)$. A rotina volta ao programa principal fornecendo um novo FSTD com exatamente 2^{ninf} ramos partindo de cada estado. O programa principal tomará a seguir um novo estado para análise e o processo será reiniciado.

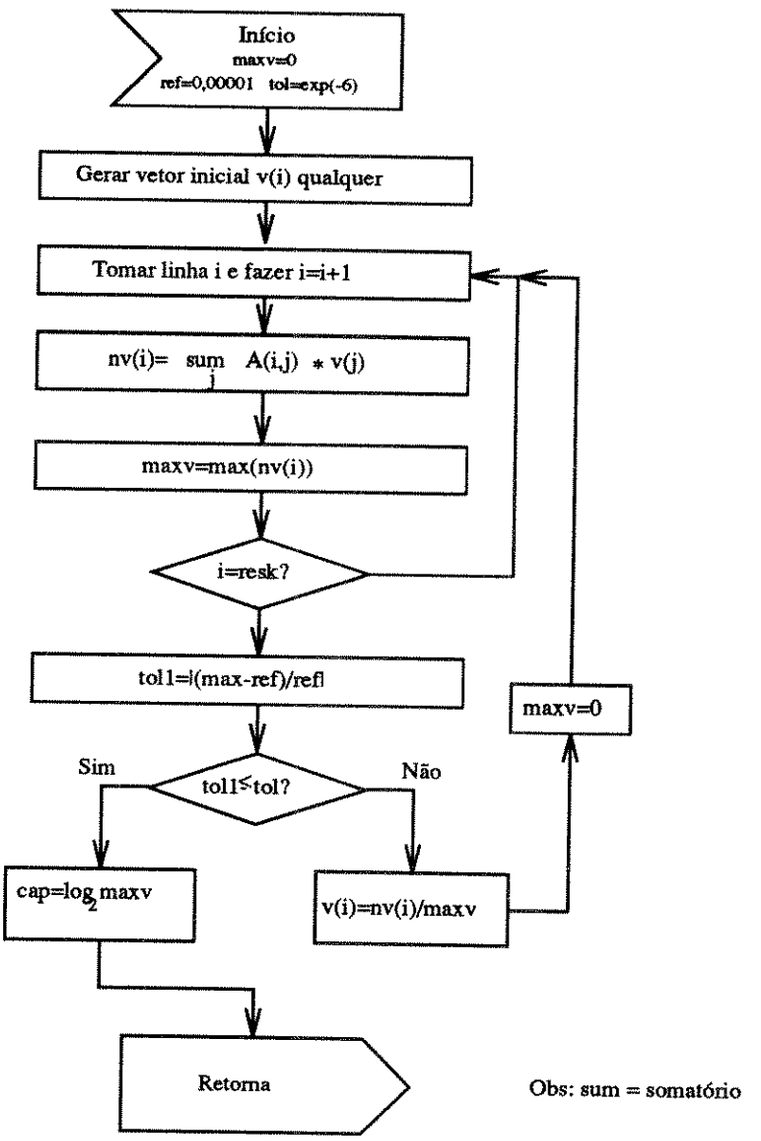


Figura 4.9: Estrutura da Subrotina CAPAC

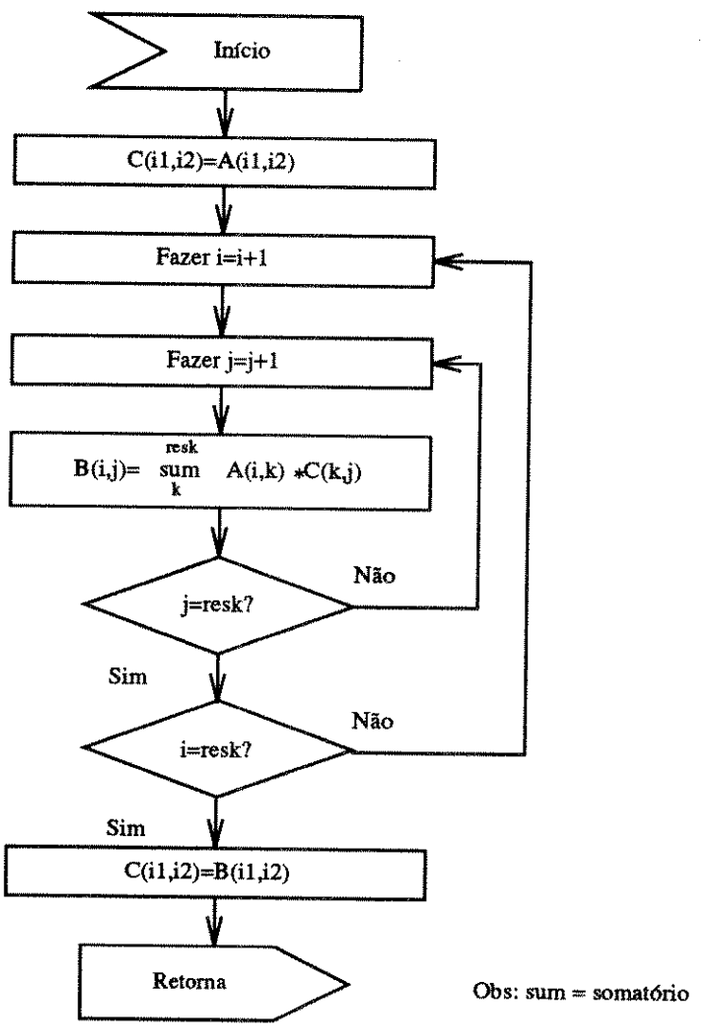


Figura 4.10: Estrutura da Subrotina POT

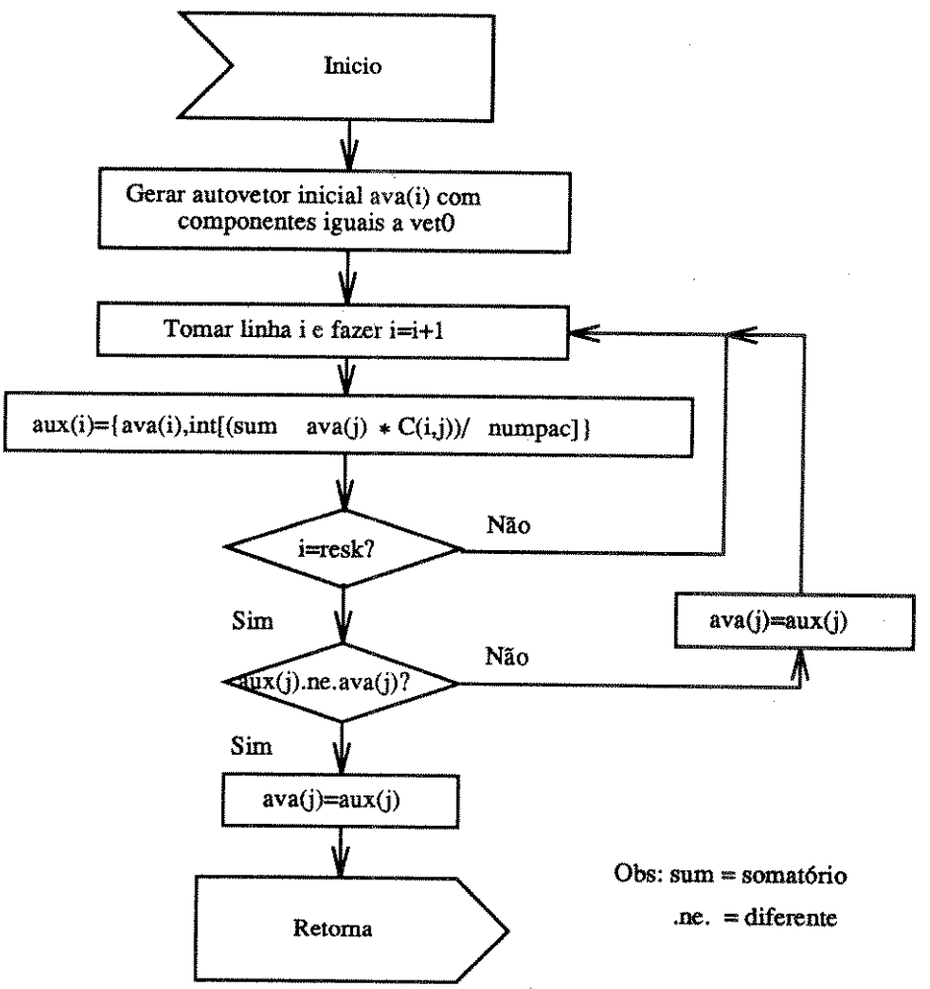


Figura 4.11: Estrutura da Subrotina AVAP

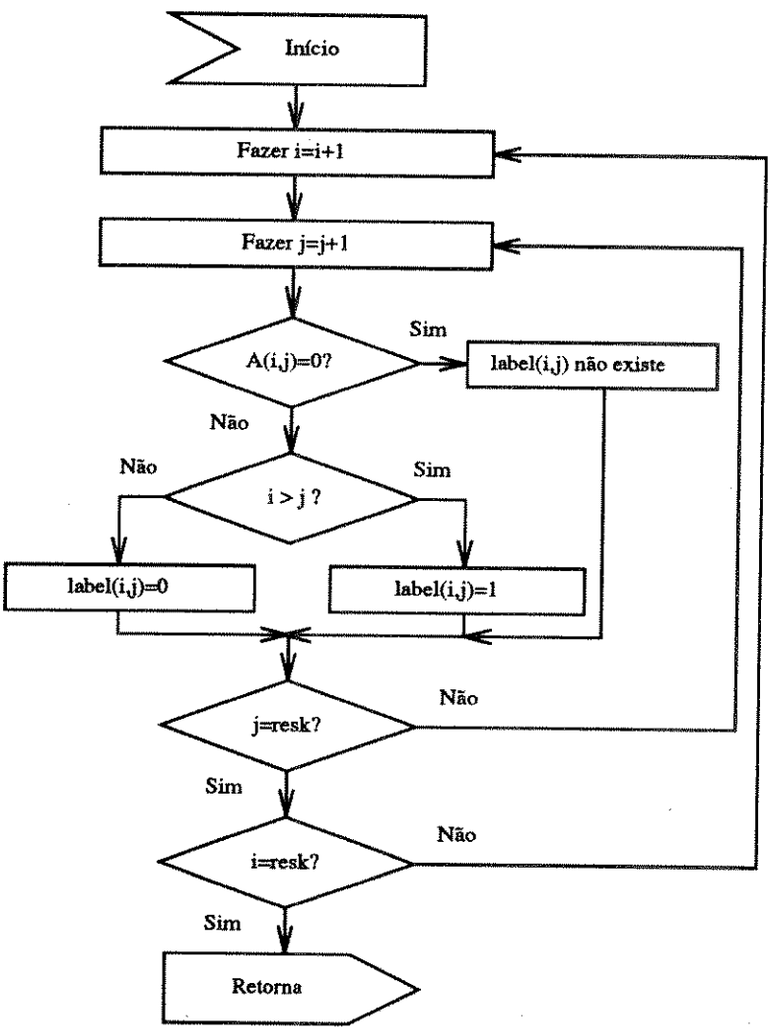


Figura 4.12: Estrutura da Subrotina LABEL1

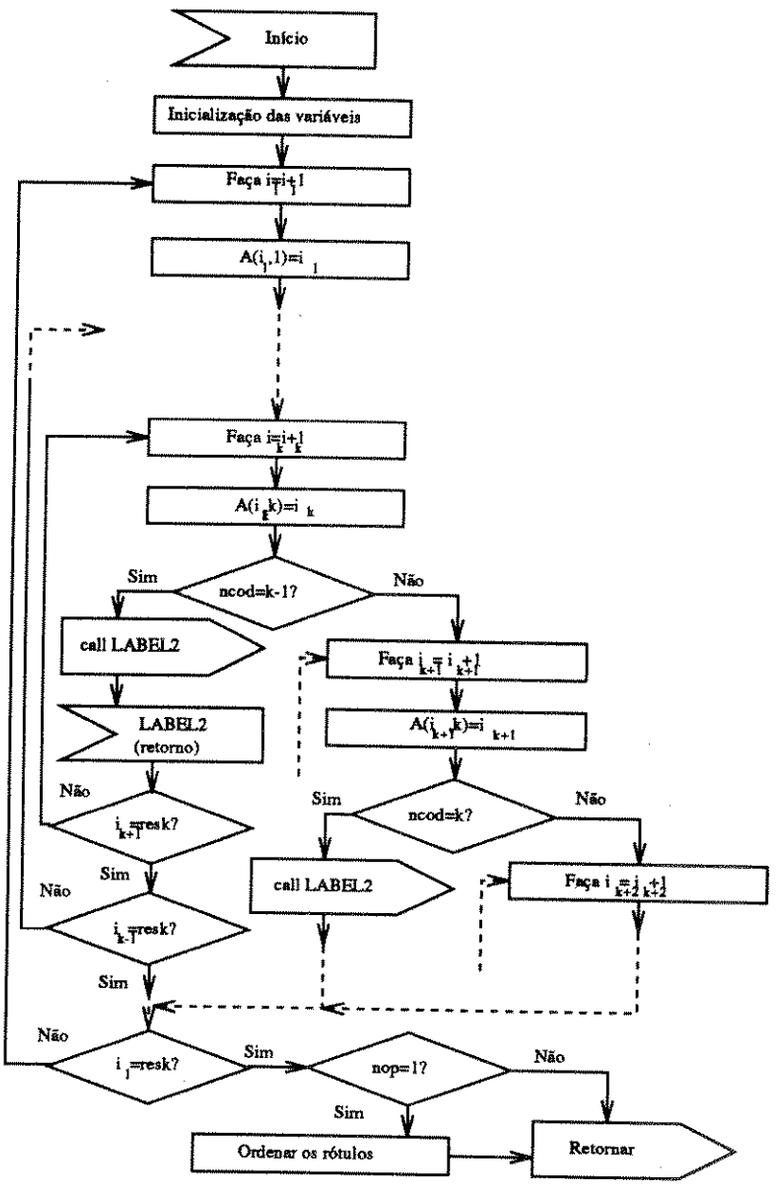


Figura 4.13: Estrutura da Subrotina SEQEST

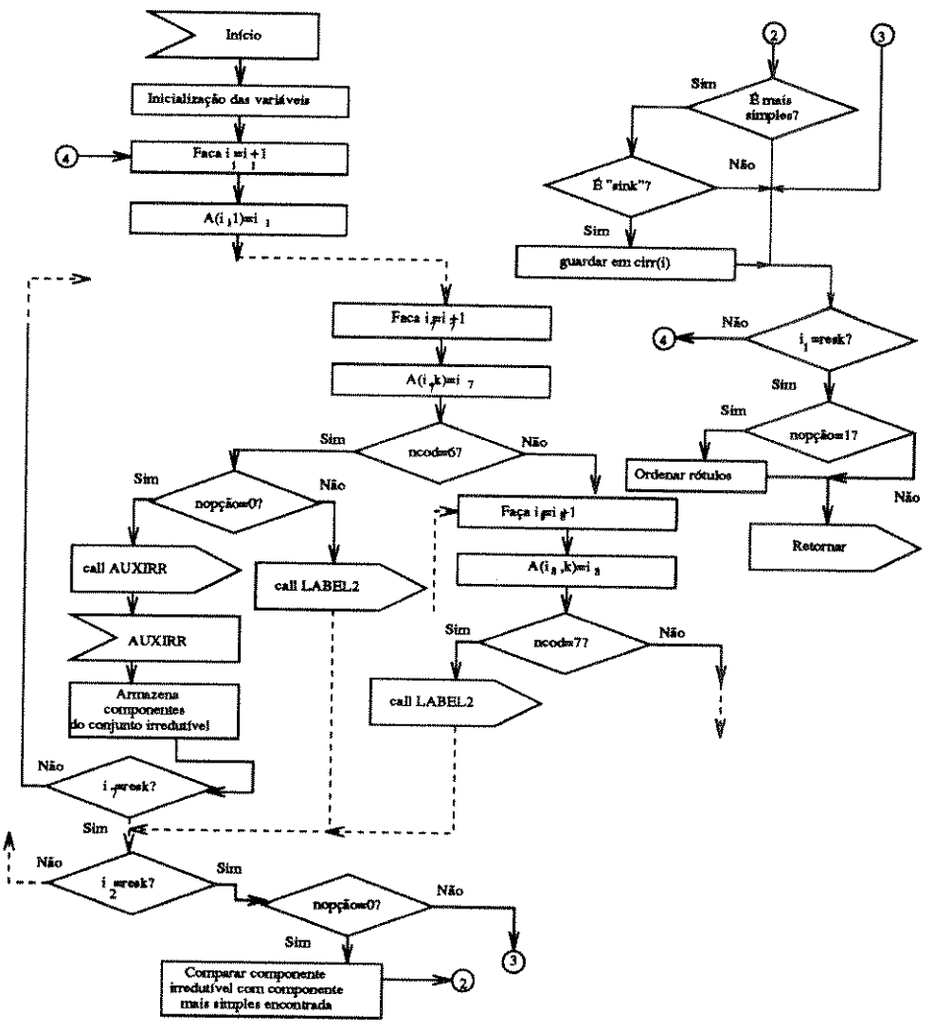


Figura 4.14: Estrutura da Subrotina SEQEST com Ênfase na Busca por Componentes Irredutíveis

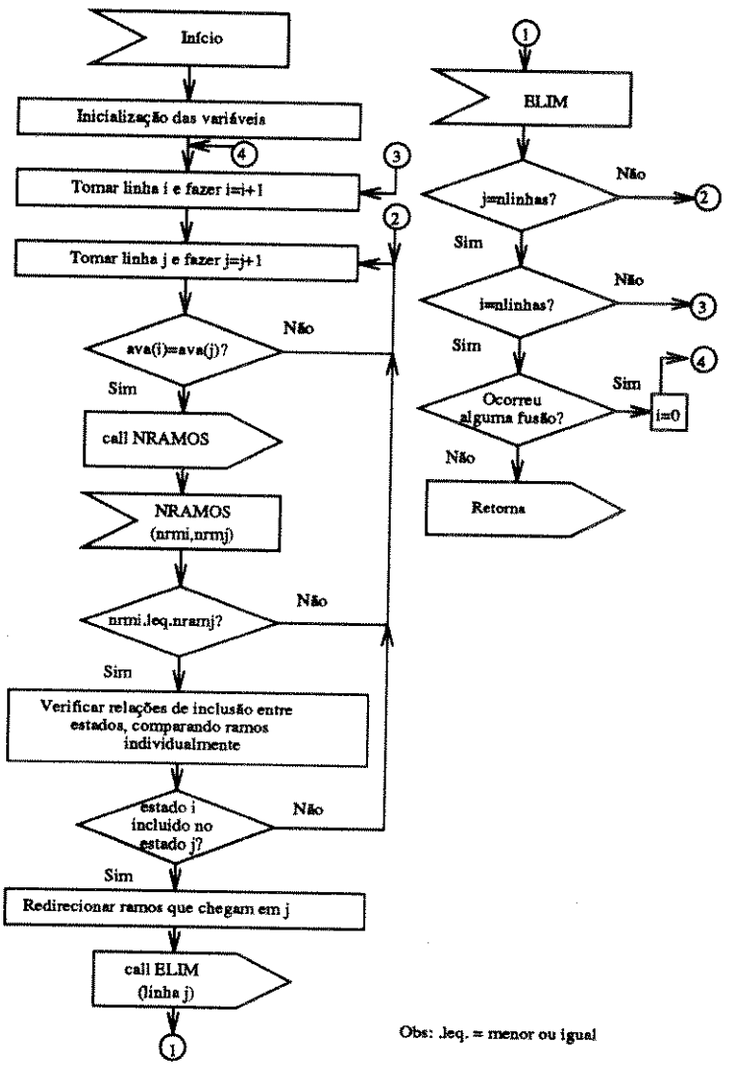


Figura 4.15: Estrutura da Subrotina MERGE

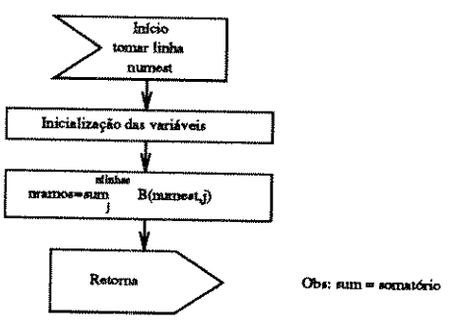


Figura 4.16: Estrutura da Subrotina NUMRAMOS

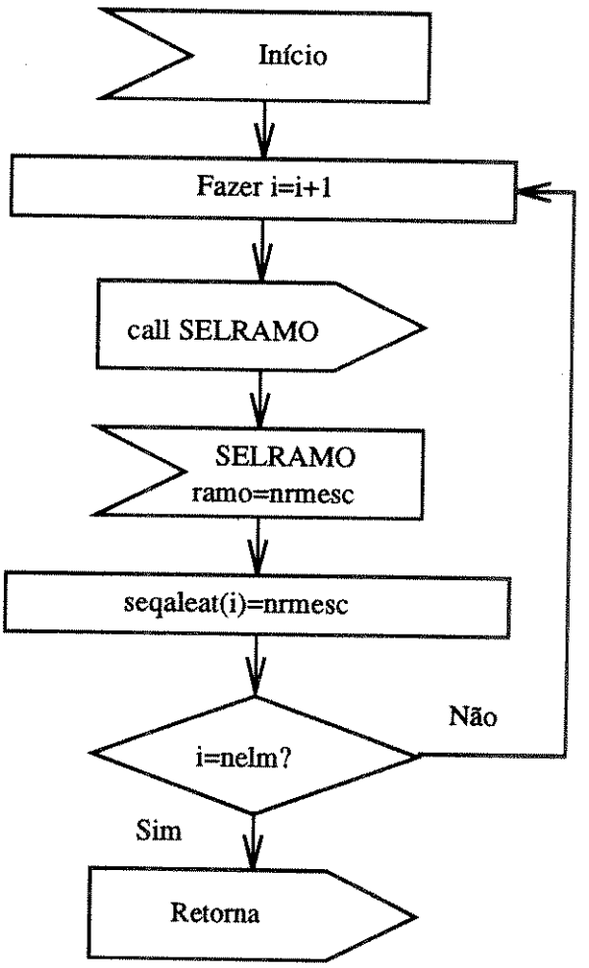


Figura 4.17: Estrutura da Subrotina SEQRAMO

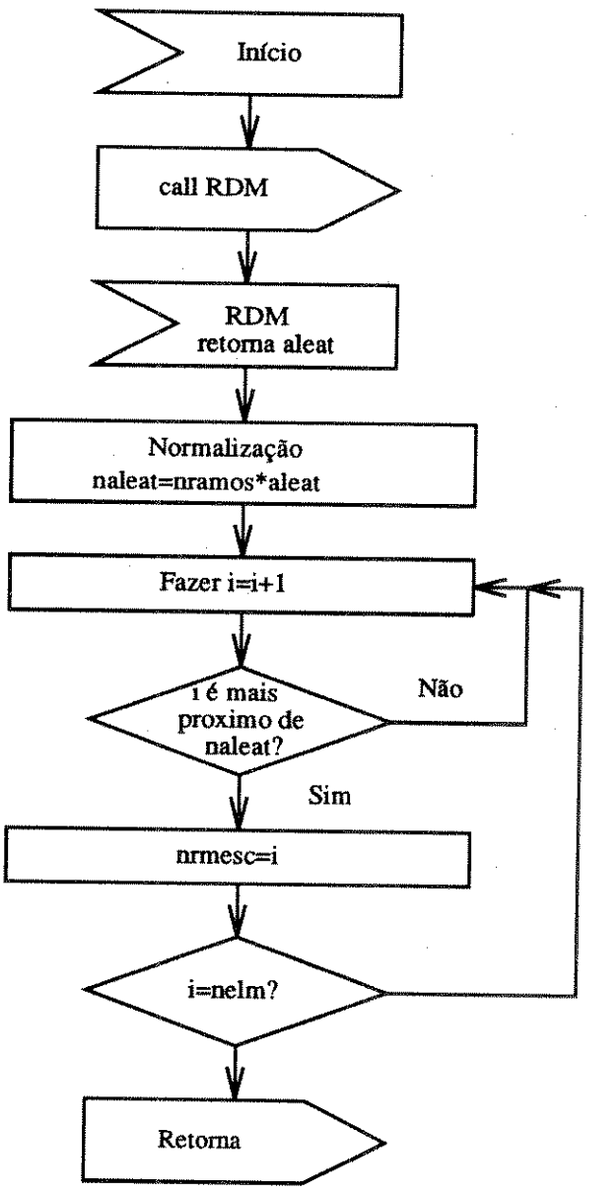


Figura 4.18: Estrutura da Subrotina SELRAMO

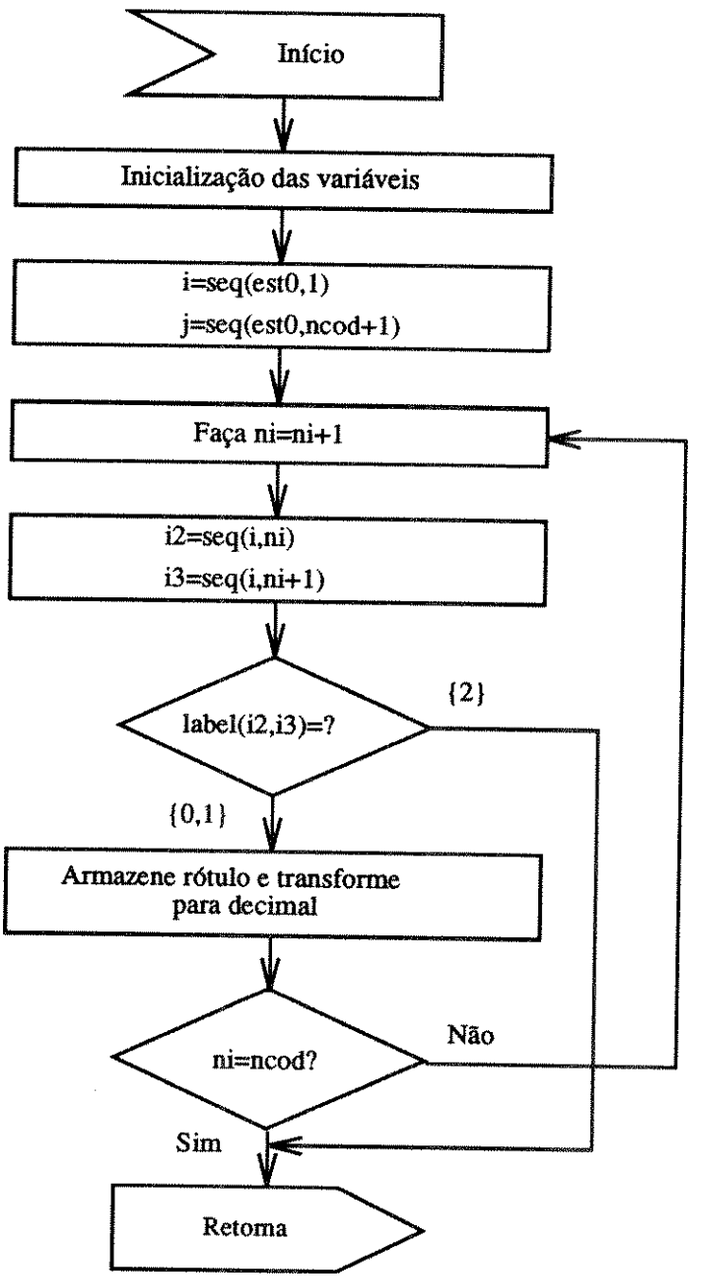


Figura 4.19: Estrutura da Subrotina LABEL2

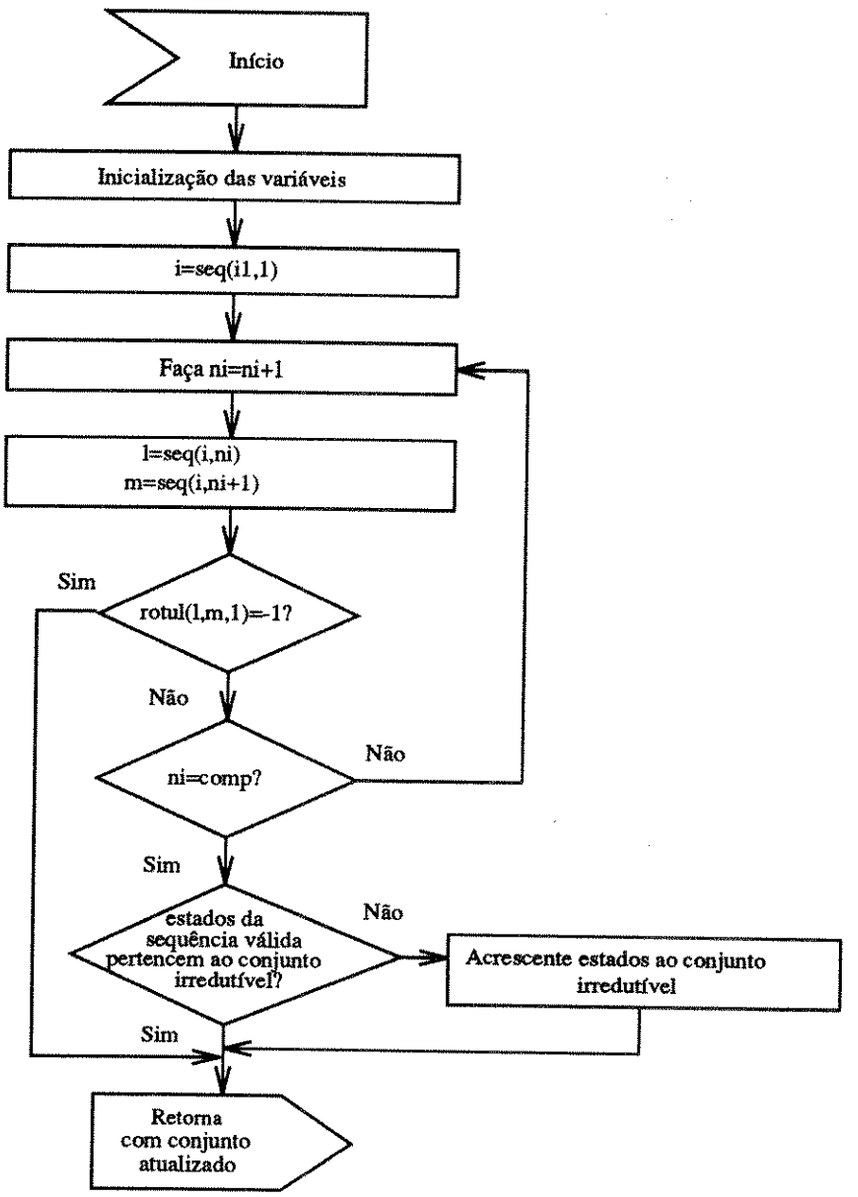


Figura 4.20: Estrutura da Subrotina AUXIRR

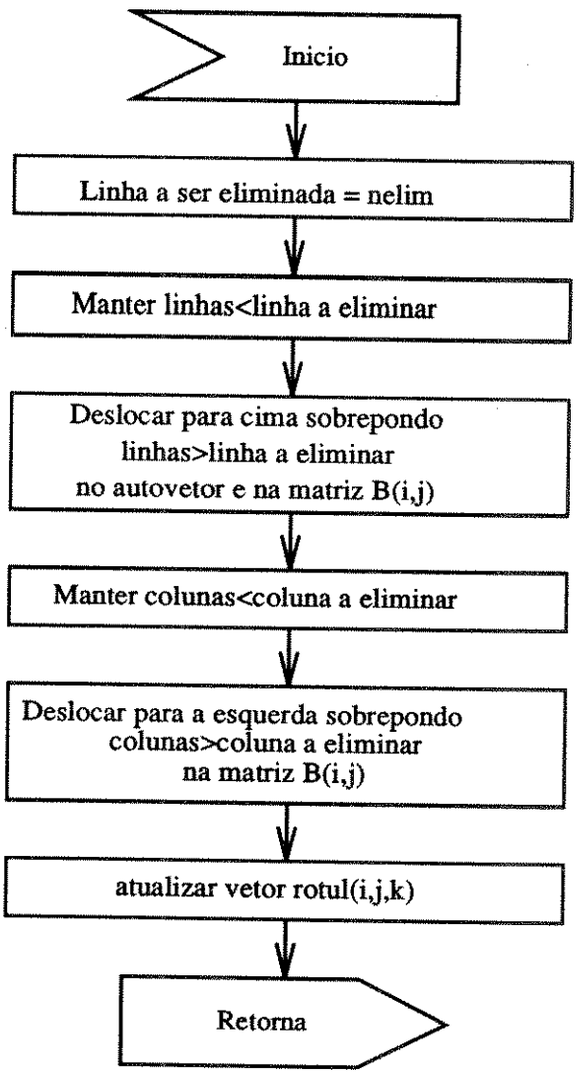


Figura 4.21: Estrutura da Subrotina ELIM

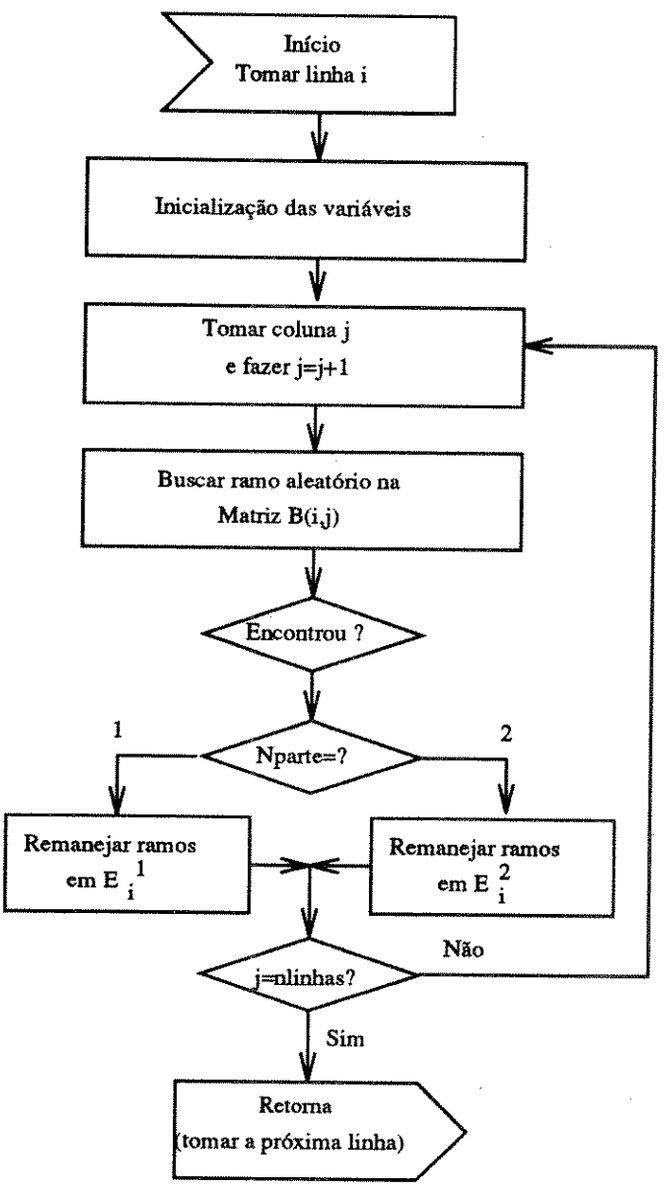


Figura 4.22: Estrutura da Subrotina SPLIT

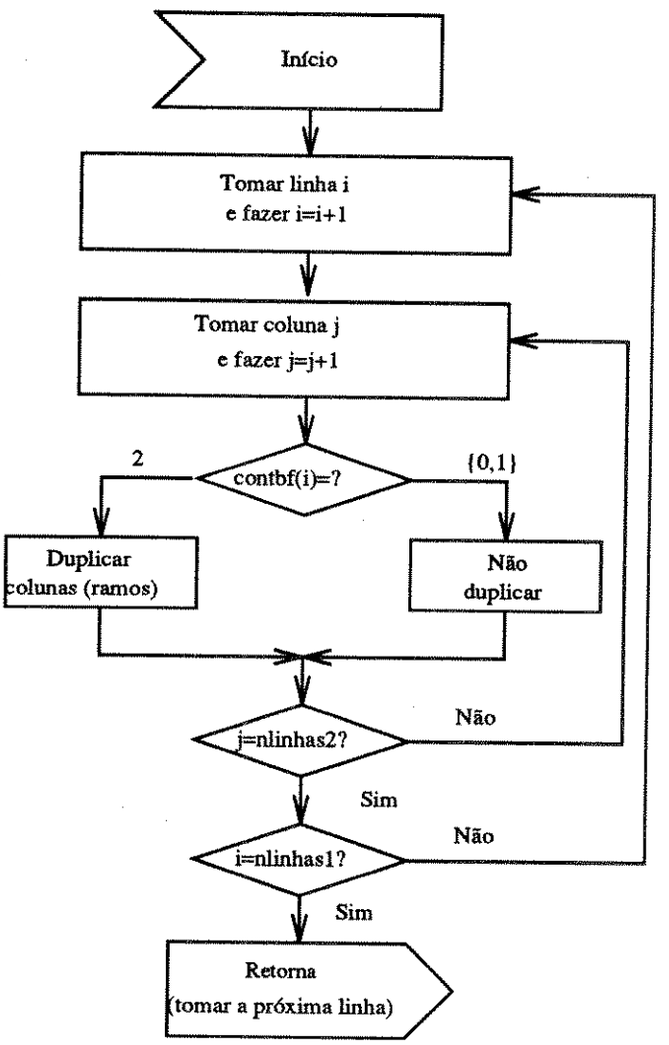


Figura 4.23: Estrutura da Subrotina DUPLIC

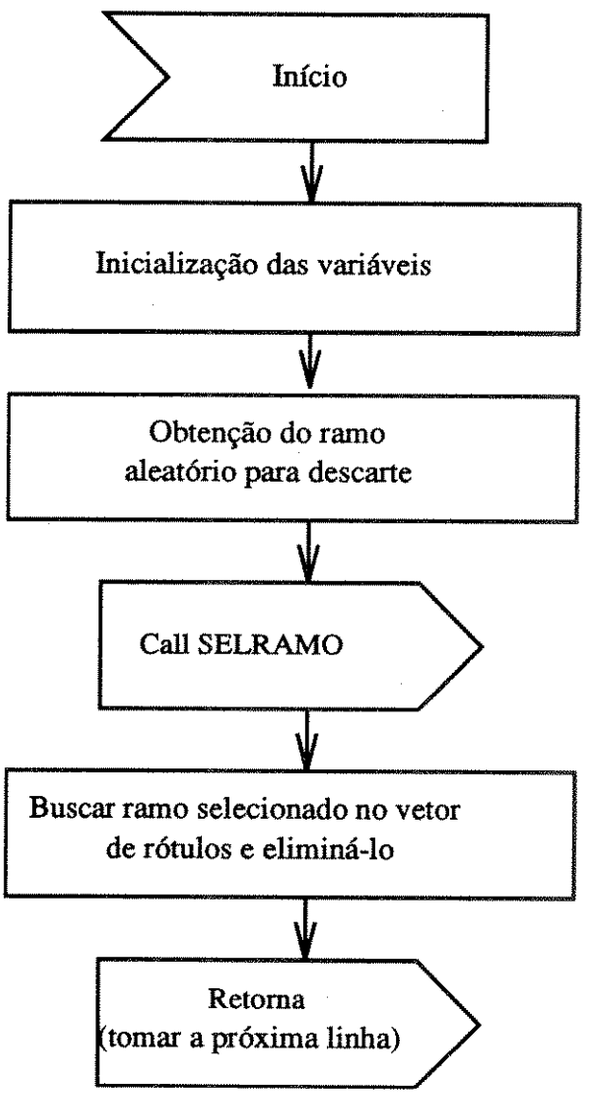


Figura 4.24: Estrutura da Subrotina DESCART

Capítulo 5

RESULTADOS OBTIDOS

5.1 Parâmetros importantes para análise

Visando a validação da ferramenta desenvolvida procedeu-se a uma exaustiva busca por códigos dk dentro das diversas restrições possíveis.

Para este fim foram testadas restrições d de 1 a 5 e k de 6 a 11 para diversas taxas. Dentre os resultados obtidos, os melhores até então conseguidos serão agora analisados. Os códigos são agrupados em ordem crescente de densidade de armazenamento. A busca tinha por fim encontrar códigos dk com menor complexidade possível. A análise abrangeu os seguintes parâmetros:

- **Densidade de armazenamento** - Conforme já foi dito, o parâmetro d influencia diretamente no grau de compactação de informação a ser armazenada no meio magnético, de acordo com a equação 2.9;
- **Eficiência de codificação** - Uma relação taxa/capacidade próxima de 1 indica um codificador eficiente. Em geral, quanto mais próximo de um estiver este parâmetro maior será o número de estados do codificador;
- **Autovetor aproximado** - Tomando-se a componente máxima do autovetor é possível estabelecer um limitante inferior no número de estados do codificador. A teoria estabelece que o número de estados num codificador é limitado superiormente pela soma das componentes do autovetor aproximado e inferiormente pelo valor da componente máxima deste mesmo autovetor;
- **Número de Estados** - Um número pequeno de estados no codificador implica num reduzido número de circuitos para a sua implementação e, portanto, um custo reduzido de produção;
- **Tamanho da janela de decodificação** - De acordo com o que foi definido no Capítulo 3, o tamanho máximo da janela de decodificação é limitado superiormente pela equação 3.16. Um outro fator que tem influência direta no tamanho

da janela de decodificação é a atribuição de rótulos da forma s/t, onde s e t representam os rótulos dos blocos na entrada/saída do codificador.

- **Complexidade de Codificação** - Este parâmetro pode ser medido pelo tamanho da memória necessária à implementação, relacionando-se com o número de estados do codificador bem como com a taxa de codificação, de acordo com a equação 3.19;
- **Complexidade de Decodificação** - Também de acordo com a equação 3.20, a complexidade de decodificação relaciona-se diretamente com o tamanho da janela de decodificação bem como à taxa de codificação;
- **Complexidade Total** - A soma dos dois parâmetros anteriores.

5.2 Alguns códigos obtidos

5.2.1 Código $(d, k) = (1, 6)$, com $R = 3/5$

Esta secção tem início com a obtenção de um código (1,6). De acordo com a Tabela 2.1, a capacidade para este código vale 0,669. A taxa escolhida para este caso foi $R = 3/5$.

Trata-se de um código com ganho em densidade de 1,2 ou 20%, quando comparado a um sistema não codificado. Sua eficiência vale 0,9. Para a obtenção deste código utilizou-se do seguinte autovetor aproximado: (1 1 1 1 1 1 0).

Observa-se assim que a componente máxima do autovetor vale 1. Com a eliminação do 7o estado nenhuma divisão de estados vai ser necessária, e os limitantes no número de estados do codificador ficam entre 1 e 6. Após a aplicação do software, seguido de uma criteriosa atribuição de etiquetas aos rótulos, obteve-se o codificador com número de estados igual a 2, bastante próximo, assim, do limitante inferior. Este codificador encontra-se ilustrado na Tabela 5.1. A atribuição de etiquetas tornou possível a obtenção de um decodificador com apenas 1 janela de decodificação. As complexidades de codificação e decodificação para este código valem, respectivamente 96 e 96.

Tabela 5.1

Tabela do Codificador (1,6), taxa 3/5, com 2 estados

	Entradas							
	000	001	010	011	100	101	110	111
1	2/01010	2/01000	1/01001	1/00001	1/00101	2/00010	2/00100	2/00000
2	2/01010	2/01000	1/01001	1/10001	1/10101	2/10100	2/10010	2/10000

5.2.2 Código $(d, k) = (1, 7)$, com $R = 3/5$

O próximo código a ser analisado é o código (1,7). De acordo com a Tabela 2.1, a capacidade para este código vale 0,6793. A taxa escolhida para este caso foi $R = 3/5$.

Trata-se de um código com ganho em densidade de 1.2 ou 20%, quando comparado a um sistema não codificado. Sua eficiência vale 0,88. Para a obtenção deste código utilizou-se do seguinte autovetor aproximado: (1 1 1 1 1 1 1 0).

Observa-se assim que a componente máxima do autovetor vale 1. Com a eliminação do 8o estado nenhuma divisão de estados vai ser necessária, e os limitantes no número de estados do codificador ficam entre 1 e 7. Após a aplicação do software, seguido de uma criteriosa atribuição de etiquetas aos rótulos, obteve-se o codificador com número de estados igual a 2, bastante próximo, assim, do limitante inferior. Este codificador encontra-se ilustrado na Tabela 5.2. A atribuição de etiquetas tornou possível a obtenção de um decodificador com apenas 1 janela de decodificação. Por fim, as complexidades de codificação e decodificação encontradas para este código valem, respectivamente 96 e 96.

Tabela 5.2

Tabela do Codificador (1,7), taxa 3/5, com 2 estados

	Entradas							
	000	001	010	011	100	101	110	111
1	1/00001	2/00000	1/00101	1/01001	2/01010	2/00100	2/00010	2/01000
2	2/10000	2/10010	1/00101	1/01001	2/01010	2/00100	1/10001	1/10101

5.2.3 Código $(d, k) = (1, 8)$, com $R = 3/5$

O próximo código a ser analisado é o código (1,8). De acordo com a Tabela 2.1, a capacidade para este código vale 0,6853. A taxa escolhida para este caso foi $R = 3/5$.

Trata-se de um código com ganho em densidade de 1,2 ou 20%, quando comparado a um sistema não codificado. Sua eficiência vale, aproximadamente, 0,88. Para a obtenção deste código utilizou-se do seguinte autovetor aproximado: (1 1 1 1 1 1 1 1 0).

Observa-se assim que a componente máxima do autovetor vale 1.Com a eliminação do 9o estado nenhuma divisão de estados vai ser necessária, e os limitantes no número de estados do codificador ficarão entre 1 e 8. Após a aplicação do software, seguido de uma criteriosa atribuição de etiquetas aos rótulos, obteve-se o codificador com número de estados igual a 2, bastante próximo, assim, do limitante inferior. Este codificador encontra-se ilustrado na Tabela 5.3. A atribuição de etiquetas tornou possível a obtenção de um decodificador com apenas 1 janela de decodificação. Por fim, as complexidades de codificação e decodificação encontradas para este código valem, respectivamente 96 e 96.

Tabela 5.3

Tabela do Codificador (1,8), taxa 3/5, com 2 estados

	Entradas							
	000	001	010	011	100	101	110	111
1	1/01001	2/00010	2/00100	2/01000	2/01010	1/00101	1/00001	2/00000
2	1/01001	2/00010	2/00100	2/01000	2/01010	1/10101	1/10001	2/10010

5.2.4 Código $(d, k) = (1, 9)$, com $R = 3/5$

O próximo código a ser analisado é o código (1,9). De acordo com a Tabela 2.1, a capacidade para este código vale 0,6888. A taxa escolhida para este caso foi $R = 3/5$.

Trata-se de um código com ganho em densidade de 1,2 ou 20%, quando comparado a um sistema não codificado. Sua eficiência vale, aproximadamente, 0,87. Para a obtenção deste código utilizou-se do seguinte autovetor aproximado: (1 1 1 1 1 1 1 1 1 0). Este código, na verdade, foi obtido a partir de uma busca por um código (1,10), com as mesmas características deste. Mais adiante será comentada essa ocorrência.

Observa-se assim que a componente máxima do autovetor vale 1. Com a eliminação do 10o estado nenhuma divisão de estados vai ser necessária, e os limitantes no

número de estados do codificador ficarão entre 1 e 9. Após a aplicação do software, seguido de uma criteriosa atribuição de etiquetas aos rótulos, obteve-se o codificador com número de estados igual a 2, bastante próximo, assim, do limitante inferior. A atribuição de etiquetas tornou possível a obtenção de um decodificador com apenas 1 janela de decodificação. Um dos codificadores obtidos é ilustrado na Tabela 5.4. Por fim, as complexidades de codificação e decodificação encontradas para este código valem, respectivamente 96 e 96.

Tabela 5.4

Tabela do Codificador (1,9), taxa 3/5, com 2 estados

	Entradas							
	000	001	010	011	100	101	110	111
1	1/00001	1/01001	2/01000	2/01010	1/00101	2/00100	2/00010	2/00000
2	1/00001	1/01001	2/01000	2/01010	1/10001	2/10000	2/10010	1/10101

5.2.5 Código $(d, k) = (2, 6)$, com $R = 2/5$

O próximo código obtido foi o código (2,6). De acordo com a Tabela 2.1, a capacidade para este código vale 0,4979. A taxa escolhida para este caso foi $R = 2/5$.

Como no código (1,6), o ganho em densidade também é 1,2 ou 20%. Sua eficiência de codificação vale, aproximadamente 0,8. Para a obtenção deste código utilizou-se do seguinte autovetor aproximado: (1 1 1 1 1 1 0).

Observa-se assim que a componente máxima do autovetor vale 1. Com a eliminação do 7o estado nenhum passo de divisão de estados será necessário e os limitantes no número de estados do codificador ficarão entre 1 e 6. Após a aplicação do software seguido de uma criteriosa atribuição de etiquetas aos rótulos, obteve-se o codificador com número de estados igual a 3 e decodificadores com apenas 1 janela de decodificação. Para este caso diversos códigos foram obtidos. Um deles encontra-se ilustrado na Tabela 5.5. Por fim, as complexidades de codificação e decodificação encontradas para este código valem, respectivamente 224 e 32.

Tabela 5.5

Tabela de um Codificador (2,6), taxa 2/5 com 3 Estados

	Entradas			
	00	01	10	11
1	3/00000	1/00001	2/00010	3/00100
2	1/01001	1/00001	2/00010	3/01000
3	1/01001	1/10001	3/10000	2/10010

5.2.6 Código $(d, k) = (2, 7)$, com $R = 2/5$

O próximo código a ser analisado é o código (2,7). De acordo com a Tabela 2.1, a capacidade para este código vale 0,5174. A taxa escolhida para este caso foi $R = 2/5$.

Trata-se de um código com ganho em densidade de 1,2 ou 20%, quando comparado a um sistema não codificado. Sua eficiência vale, aproximadamente, 0,77. Para a obtenção deste código utilizou-se do seguinte autovetor aproximado: (1 1 1 1 1 1 1 0). Este código foi obtido a partir de um código (2,8).

Observa-se assim que a componente máxima do autovetor vale 1. Com a eliminação do 8o estado nenhuma divisão de estados vai ser necessária, e os limitantes no número de estados do codificador ficarão entre 1 e 7. Após a aplicação do software, seguido de uma criteriosa atribuição de etiquetas aos rótulos, obteve-se o codificador com número de estados igual a 2, bastante próximo, assim, do limitante inferior. A atribuição de etiquetas tornou possível a obtenção de um decodificador com apenas 1 janela de decodificação. Um dos codificadores obtidos é ilustrado na Tabela 5.6. Por fim, as complexidades de codificação e decodificação encontradas para este código valem, respectivamente 56 e 32.

Tabela 5.6

Tabela do Codificador (2,7), taxa 2/5 com 2 Estados

	Entradas			
	00	01	10	11
1	2/00000	1/00001	1/00010	2/00100
2	2/10000	1/10001	1/01001	1/10010

5.2.7 Código $(d,k) = (2,8)$, com $R = 2/5$

O próximo código a ser analisado é o código (2,8). De acordo com a Tabela 2.1, a capacidade para este código vale 0,5293. A taxa escolhida para este caso foi $R = 2/5$.

Trata-se de um código com ganho em densidade de 1,2 ou 20%, quando comparado a um sistema não codificado. Sua eficiência vale, aproximadamente, 0,76. Para este código o seguinte autovetor aproximado foi possível: (1 1 1 1 1 1 1 1 0).

Observa-se assim que a componente máxima do autovetor vale 1.Com a eliminação do 9o estado nenhuma divisão de estados vai ser necessária, e os limitantes no número de estados do codificador ficarão entre 1 e 9. Após a aplicação do software, seguido de uma criteriosa atribuição de etiquetas aos rótulos, obteve-se o codificador com número de estados igual a 3. A atribuição de etiquetas tornou possível a obtenção de um decodificador com apenas 1 janela de decodificação. O codificador obtido é ilustrado na Tabela 5.7. Por fim, as complexidades de codificação e decodificação encontradas para este código valem, respectivamente 224 e 32.

Tabela 5.7

Tabela do Codificador (2,8), taxa 2/5 com 3 Estados

	Entradas			
	00	01	10	11
1	1/00001	2/00010	2/00100	3/00000
2	1/00001	2/00010	2/00100	1/01001
3	2/10000	2/00010	2/00100	2/10010

5.2.8 Código $(d,k) = (2,9)$, com $R = 2/5$

O próximo código a ser analisado é o código (2,9). De acordo com a Tabela 2.1, a capacidade para este código vale 0,5369. A taxa escolhida para este caso foi $R = 2/5$.

Trata-se de um código com ganho em densidade de 1,2 ou 20%, quando comparado a um sistema não codificado. Sua eficiência vale, aproximadamente, 0,75. Para a obtenção deste código o seguinte autovetor aproximado foi possível: (1 1 1 1 1 1 1 1 1 0). Este código foi obtido a partir de um código (2,10), com as mesmas características deste.

Observa-se assim que a componente máxima do autovetor vale 1. Com a eliminação do 10o estado nenhuma divisão de estados vai ser necessária, e os limitantes no

número de estados do codificador ficarão entre 1 e 9. Após a aplicação do software, seguido de uma criteriosa atribuição de etiquetas aos rótulos, obteve-se o codificador com número de estados igual a 2, bastante próximo, assim, do limitante inferior. A atribuição de etiquetas tornou possível a obtenção de um decodificador com apenas 1 janela de decodificação. O codificador obtido é ilustrado na Tabela 5.8. Por fim, as complexidades de codificação e decodificação encontradas para este código valem, respectivamente 56 e 32.

Tabela 5.8

Tabela do Codificador (2,9), taxa 2/5 com 2 Estados

	Entradas			
	00	01	10	11
1	1/00001	2/00010	2/00100	2/00000
2	1/00001	2/00010	2/00100	1/01001

5.2.9 Código $(d, k) = (1, 7)$, com $R = 2/3$

O próximo código obtido foi o código (1,7). De acordo com a Tabela 2.1, a capacidade para este código vale 0,6793. A taxa escolhida para este caso foi $R = 2/3$.

Trata-se de um código com ganho em densidade de 1,33 ou 33,33%, maior portanto, que o anterior. Sua eficiência de codificação também é maior e vale, aproximadamente, 0,98. Para a obtenção deste código utilizou-se do seguinte autovetor aproximado: (2 3 3 3 2 2 2 1).

Observa-se assim que a componente máxima do autovetor vale 3. Esta condição indica que o processo de divisão de estados é necessário e que os limitantes no número de estados do codificador são 3 e 18. Após a aplicação do software, durante a fase inicial de fusões, este autovetor reduziu o número máximo para 7 e, no final da busca, o número de estados obtidos foi 4. Segundo a teoria, apesar do limitante inferior não ter sido alcançado, este é o número mínimo de estados possível [14]. A atribuição de etiquetas tornou possível a obtenção de decodificadores com 3 janelas de decodificação. Por fim, as complexidades de codificação e decodificação encontradas para este código valem, respectivamente 80 e 1024.

Para este caso, dois códigos diferentes, foram obtidos, um deles é apresentado na Tabela 5.9.

Tabela 5.9

Tabela do Codificador (1,7), taxa 2/3 com 4 Estados

	Entradas			
	00	01	10	11
1	1/001	4/000	3/000	2/001
2	3/010	1/000	4/010	1/010
3	2/101	1/100	1/101	1/010
4	3/010	3/100	4/010	4/100

5.2.10 Código $(d, k) = (1, 9)$, com $R = 2/3$

O próximo código obtido foi o código (1,9). De acordo com a Tabela 2.1, a capacidade para este código vale 0,6888. A taxa escolhida para este caso também foi $R = 2/3$.

Trata-se de um código com ganho em densidade análogo ao (1,7), ou seja, de 1,33 ou 33,33%. Sua eficiência de codificação é um pouco inferior e vale 0,97. Para a obtenção deste código utilizou-se do seguinte autovetor aproximado: (2 3 3 3 3 3 2 2 2 1).

Observa-se assim que a componente máxima do autovetor vale 3. Esta condição indica que o processo de divisão de estados é necessário e que os limitantes no número de estados do codificador são 3 e 24. Após a aplicação do software, durante a fase inicial de fusões este autovetor reduziu o número máximo de estados para 10 e, no final da busca, o número de estados obtidos foi, como no (1,7), 4. Sem dúvida, um valor bastante próximo do limite mínimo. A atribuição de etiquetas tornou possível a obtenção de decodificadores com 3 janelas de decodificação. Por fim, as complexidades de codificação e decodificação encontradas para este código valem, respectivamente 80 e 1024.

A importante observação, neste caso, foi a drástica redução no número de estados do codificador de 24 no início, para apenas 4 no final. Este fato permitiu uma avaliação do método como sendo de alto caráter de simplificação. O código (1,9) obtido é apresentado na Tabela 5.10.

Tabela 5.10

Tabela do Codificador (1,9), taxa 2/3 com 4 Estados

	Entradas			
	00	01	10	11
1	2/001	1/001	3/000	4/000
2	2/010	1/010	3/010	1/000
3	1/100	2/100	1/101	2/101
4	2/010	1/010	3/010	3/100

5.2.11 Código $(d,k) = (3,6)$, com $R = 1/3$ e $R = 2/6$

O próximo código a ser analisado é o código (3,6). De acordo com a Tabela 2.1, a capacidade para este código vale 0.3746. Dois códigos foram obtidos para as taxas escolhidas: $R = 1/3$ e $R = 2/6$. Será analisado inicialmente para $R = 1/3$, e logo a seguir, $R = 2/6$.

Trata-se de um código com ganho em densidade de 1,33 ou 33,3%, quando comparado a um sistema nào codificado. Sua eficiência vale, aproximadamente, 0,89. Para a obtenção deste código o seguinte autovetor aproximado foi possível: (2 3 4 4 4 3 2). Este código foi obtido com o auxílio da técnica de redução do autovetor aproximado. Neste caso, o autovetor tornou-se igual a: (2 3 4 4 3 2).

Observa-se assim que a componente máxima do autovetor vale 4. Esta condição indica que o algoritmo para divisão de estados é necessário, e que os limitantes no número de estados do codificador ficam entre 4 e 18. Após a aplicação do software, seguido de uma criteriosa atribuição de etiquetas aos rótulos, obteve-se o codificador com número de estados igual a 6. A atribuição de etiquetas tornou possível a obtenção de um decodificador com 4 janelas de decodificação. O codificador obtido é ilustrado na Tabela 5.11. As, complexidades de codificação e decodificação encontradas para este código valem, respectivamente 96 e 4096.

Para o segundo caso, o autovetor aproximado obtido foi: : (1 1 2 2 2 1 0). Observa-se assim que a componente máxima do autovetor vale 2. Esta condição indica que o algoritmo para divisão de estados é necessário, e que os limitantes no número de estados do codificador ficam entre 2 e 9. Após a aplicação do software, seguido de uma criteriosa atribuição de etiquetas aos rótulos, obteve-se o codificador com número de estados igual a 4. A atribuição de etiquetas tornou possível a obtenção de um decodificador com 2 janelas de decodificação. O codificador obtido é ilustrado na Tabela 5.12. Por fim, as complexidades de codificação e decodificação encontradas

para este código valem, respectivamente 128 e 8192. Note, neste caso, que a diminuição no número de estados e na janela de decodificação obtidas, implicaram numa elevação considerável na complexidade do esquema de codificação.

Tabela 5.11

Tabela do Codificador (2,7), taxa 1/3 com 6 Estados

	Entradas	
	0	1
1	4/000	5/000
2	3/000	6/000
3	2/001	1/000
4	2/010	1/010
5	3/010	3/100
6	4/100	1/100

Tabela 5.12

Tabela do Codificador (3,6), taxa 2/6 com 4 Estados

	Entradas			
	00	01	10	11
1	3/000100	2/000100	1/000010	1/000001
2	2/010000	4/010000	2/001000	4/001000
3	3/000100	2/000100	1/000010	1/010001
4	1/100001	4/100000	1/100010	1/010001

5.2.12 Código $(d, k) = (3, 7)$, com $R = 1/3$

O próximo código a ser analisado é o código (3,7). De acordo com a Tabela 2.1, a capacidade para este código vale 0,4057. A taxa escolhida para este caso foi $R = 1/3$.

Trata-se de um código com ganho em densidade de 1,33 ou 33,3%, quando comparado a um sistema não codificado. Sua eficiência vale, aproximadamente, 0,82. Para a obtenção deste código o seguinte autovetor aproximado foi possível: (1 1 2 2 2 2 2 1 1). Este código foi obtido a partir de um (3,8).

Observa-se assim que a componente máxima do autovetor vale 2. Esta condição indica que o algoritmo para divisão de estados é necessário, e que os limitantes no

número de estados do codificador ficam entre 2 e 14. Após a aplicação do software, seguido de uma criteriosa atribuição de etiquetas aos rótulos, obteve-se o codificador com número de estados igual a 3. A atribuição de etiquetas tornou possível a obtenção de um decodificador com 2 janelas de decodificação. O codificador obtido é ilustrado na Tabela 5.13. Por fim, as complexidades de codificação e decodificação encontradas para este código valem, respectivamente 40 e 64.

Tabela 5.13

Tabela do Codificador (3,7), taxa 1/3 com 3 Estados

	Entradas	
	0	1
1	2/000	3/000
2	1/010	1/001
3	2/100	1/100

5.2.13 Código $(d, k) = (3, 8)$, com $R = 2/6$

O próximo código a ser analisado é o código (2,9). De acordo com a Tabela 2.1, a capacidade para este código vale 0,4057. A taxa escolhida para este caso foi $R = 2/6$.

Trata-se de um código com ganho em densidade de 1,33 ou 33,3%, quando comparado a um sistema não codificado. Sua eficiência vale, aproximadamente, 0,82. Para a obtenção deste código o seguinte autovetor aproximado foi possível: (1 1 1 1 1 1 1 1 0).

Observa-se assim que a componente máxima do autovetor vale 1. Com a eliminação do 9o estado nenhuma divisão de estados vai ser necessária, e os limitantes no número de estados do codificador ficarão entre 1 e 9. Após a aplicação do software, seguido de uma criteriosa atribuição de etiquetas aos rótulos, obteve-se o codificador com número de estados igual a 3. A atribuição de etiquetas tornou possível a obtenção de um decodificador com apenas 1 janela de decodificação. Um dos codificadores obtidos é ilustrado na Tabela 5.14. Por fim, as complexidades de codificação e decodificação encontradas para este código valem, respectivamente 128 e 128.

Tabela 5.14

Tabela do Codificador (3,8), taxa 2/6 com 3 Estados

	Entradas			
	00	01	10	11
1	1/000010	2/000100	1/000001	3/000000
2	1/000010	2/000100	1/010001	2/001000
3	1/100010	1/100001	1/010001	3/100000

5.2.14 Código $(d, k) = (2, 7)$, com $R = 1/2$

O próximo código obtido foi o código (2,7). De acordo com a Tabela 2.1, a capacidade para este código vale 0,5174. A taxa escolhida, para este caso, foi $R = 1/2$.

Trata-se de um código com ganho em densidade igual a 1,5 ou 50%. Possui uma boa eficiência de codificação, sendo igual a 0,97. Para a obtenção deste código utilizou-se do seguinte autovetor aproximado: (2 3 4 4 3 3 2 1).

Observa-se assim que a componente máxima do autovetor vale 4. Esta condição indica que o processo de divisão de estados é necessário e que os limitantes no número de estados do codificador ficarão entre 4 e 22. Após a aplicação do software, o número final de estados obtidos foi 7. A busca por bons códigos com essas características sempre exigiu um tempo longo e significativamente superior aos anteriormente analisados, já que não era possível uma simplificação inicial no número de estados via o processo de fusão, entretanto, com a introdução da técnica de simplificação do autovetor aproximado, foi possível chegar-se a estes codificadores rapidamente. O codificador (2,7) pode ser visto na Tabela 5.15. A atribuição de etiquetas tornou possível a obtenção de decodificadores com 4 janelas de decodificação. Por fim, as complexidades de codificação e decodificação encontradas para este código valem, respectivamente 80 e 256.

Tabela 5.15

Tabela do Codificador (2,7), taxa 1/2 com 7 Estados

	Entradas	
	0	1
1	3/00	4/00
2	6/00	5/00
3	1/01	2/01
4	6/00	7/00
5	3/00	3/10
6	1/10	2/10
7	6/00	3/10

5.2.15 Código $(d, k) = (2, 8)$, com $R = 1/2$

O código a ser analisado agora será o código (2,8). De acordo com a Tabela 2.1, a capacidade para este código vale 0,5293. A taxa escolhida para este caso também foi $R = 1/2$.

Trata-se de um código com ganho em densidade análogo ao (2,7), ou seja, de 1,5 ou 50%. Sua eficiência de codificação é um pouco inferior e vale 0,94. Para a obtenção deste código utilizou-se do seguinte autovetor aproximado: (2 3 4 4 4 3 3 2 1).

Observa-se assim que a componente máxima do autovetor vale 4. Esta condição indica que o processo de divisão de estados é necessário e que os limitantes no número de estados do codificador serão 4 e 26. Após a aplicação do software obteve-se um codificador com número de estados igual a 5, sem dúvida, um valor bastante próximo do limite mínimo. Como no código (2,7), a obtenção deste código exigiu um tempo significativamente superior aos anteriormente analisados. A atribuição de etiquetas tornou possível a obtenção de decodificadores com 4 janelas de decodificação.

A grande observação, neste caso foi, além do ótimo ganho em densidade, a drástica redução no número de estados do codificador de 26 estados no início, para apenas 5 no final. O código (2,8) obtido é apresentado na Tabela 5.16. Por fim, as complexidades de codificação e decodificação encontradas para este código valem, respectivamente 80 e 256.

Tabela 5.16

Tabela do Codificador (2,8), taxa 1/2 com 5 Estados

	Entradas	
	0	1
1	2/00	5/00
2	1/01	5/01
3	1/10	5/10
4	2/00	2/10
5	4/00	3/00

5.2.16 Código $(d, k) = (2, 9)$, com $R = 2/4$

O próximo código obtido foi o código (2,9). De acordo com a Tabela 2.1, a capacidade para este código vale 0,5369. A taxa escolhida para este caso também foi $R = 2/4$.

Trata-se de um código com ganho em densidade análogo ao (2,7), ou seja, de 1,5 ou 50%. Sua eficiência de codificação é inferior aos dois anteriores e vale 0,93. Para a obtenção deste código utilizou-se do seguinte autovetor aproximado: (1 1 2 2 2 1 1 1 1 0).

Observa-se assim que a componente máxima do autovetor vale 2. Esta condição indica que o processo de divisão de estados é necessário e que os limitantes no número de estados do codificador ficarão entre 2 e 12. Após a aplicação do software, durante a fase inicial de fusões este autovetor reduziu o número máximo para 4 e no final da busca o número de estados obtidos permaneceu em 4. O codificador (2,9) é mostrado na Tabela 5.17. A atribuição de etiquetas tornou possível a obtenção de decodificadores com 2 janelas de decodificação. Por fim, as complexidades de codificação e decodificação encontradas para este código valem, respectivamente 96 e 512.

Tabela 5.17

Tabela de um Codificador (2,9), taxa 2/4 com 4 Estados

	Entradas			
	00	01	10	11
1	1/0001	4/0000	3/0000	2/0010
2	1/0001	3/0100	4/0100	2/0010
3	1/0001	4/0000	1/1001	2/0010
4	3/1000	3/0100	4/0100	4/1000

5.2.17 Código $(d,k) = (2,10)$, com $R = 1/2$ e $R = 2/4$

O próximo código obtido foi o código (2,10). De acordo com a Tabela 2.1, a capacidade para este código vale 0,5418. Para este caso foram tentadas duas taxas distintas: $R = 1/2$ e $R = 2/4$. Tratam-se de códigos com ganho em densidade de 1,5 ou 50%. Sua eficiência de codificação é um pouco inferior e vale 0,927. Será tomado para análise, inicialmente, o código de taxa 1/2.

Para a obtenção deste código utilizou-se do seguinte autovetor aproximado: (2 3 4 4 4 4 4 3 3 2 1).

Observa-se assim que a componente máxima do autovetor vale 4. Esta condição indica que o processo de divisão de estados é necessário e que os limitantes no número de estados do codificador ficarão entre 4 e 34. Após a aplicação do software, o número final de estados obtido foi 7. Este valor, apesar de não tão próximo do limitante inferior, pode ser considerado baixo tendo-se em vista o número de simplificações realizadas para se reduzir o codificador de 34 para apenas 7 estados. A atribuição de etiquetas tornou possível a obtenção de decodificadores com 4 janelas de decodificação. O código (2,10) obtido, de taxa 1/2 é apresentado na Tabela 5.18. As complexidades de codificação e decodificação encontradas para este código valem, respectivamente 80 e 256.

Tabela 5.18

Tabela do Codificador (2,10), taxa 1/2 com 7 Estados

	Entradas	
	0	1
1	5/00	3/00
2	7/00	4/00
3	2/01	1/01
4	7/00	3/00
5	1/10	2/10
6	3/10	3/00
7	5/00	6/00

O outro caso, o código (2,10) de taxa 2/4 foi obtido a partir do seguinte autovetor aproximado: (1 1 2 2 2 2 1 1 1 1 0).

Observa-se assim que a componente máxima do autovetor vale 2. Esta condição indica que o processo de divisão de estados é necessário e que os limitantes no número de estados do codificador ficarão entre 2 e 14. Durante a fase inicial de fusões o novo

autovetor obtido permitiu que se reduzisse o número máximo para 4, e no final da busca, o número de estados obtidos caiu para 3. A atribuição de etiquetas tornou possível a obtenção de decodificadores com 2 janelas de decodificação. O código (2,10) obtido, de taxa 2/4 é apresentado na Tabela 5.19. Por fim, as complexidades de codificação e decodificação encontradas para este código valem, respectivamente 96 e 512.

Essas duas situações permitem que se conclua que às custas de um aumento no tamanho do bloco, conseguiu-se diminuir significantemente o número de estados do codificador. Entretanto, um resultado inverso, em termos de complexidade, foi obtido.

Tabela 5.19
Tabela do Codificador (2,10), taxa 2/4 com 3 Estados

	Entradas			
	00	01	10	11
1	1/0001	2/0000	3/0000	1/0010
2	2/0100	3/0100	2/1000	3/1000
3	1/0001	2/0000	1/1001	1/0010

5.2.18 Código $(d, k) = (2, 11)$, com $R = 2/4$

O próximo código obtido foi o código (2,11). De acordo com a Tabela 2.1, a capacidade para este código vale 0,5450. A taxa empregada, para este, caso foi $R = 2/4$. Como nos casos anteriores, onde o parâmetro $d = 2$, trata-se de um código com ganho em densidade de 1,5 ou 50%. Sua eficiência de codificação vale aproximadamente 0,92.

Para a obtenção deste código utilizou-se do seguinte autovetor aproximado: (1 1 2 2 2 2 2 1 1 1 1 0).

Observa-se assim que a componente máxima do autovetor vale 2. Esta condição indica que o processo de divisão de estados é necessário e que os limitantes no número de estados do codificador ficarão entre 2 e 16. Durante a fase inicial de fusões, o novo autovetor obtido permitiu que se reduzisse o número máximo para 6, e no final da busca, o número de estados obtidos caiu para 3, um valor bem próximo do limitante inferior. A atribuição de etiquetas tornou possível a obtenção de decodificadores com 2 janelas de decodificação. O código (2,11) obtido é apresentado na Tabela 5.20. Por fim, as complexidades de codificação e decodificação encontradas para este código valem, respectivamente 96 e 512.

Tabela 5.20

Tabela do Codificador (2,11), taxa 2/4 com 3 Estados

	Entradas			
	00	01	10	11
1	1/0001	2/0000	3/0000	1/0010
2	2/0100	1/0100	2/1000	3/1000
3	1/0001	2/0000	1/1001	1/0010

5.2.19 Código $(d, k) = (5, 10)$, com $R = 1/4$

O próximo código a ser analisado é o código (5,10). De acordo com a Tabela 2.1, a capacidade para este código vale 0,3158. A taxa escolhida para este caso foi $R = 1/3$.

Trata-se de um código com ganho em densidade de 1.33 ou 33,3%, quando comparado a um sistema não codificado. Sua eficiência vale, aproximadamente, 0,79. Para a obtenção deste código o seguinte autovetor aproximado foi possível: (1 1 1 2 2 2 2 2 1 1).

Observa-se assim que a componente máxima do autovetor vale 2. Esta condição indica que o algoritmo para divisão de estados é necessário, e que os limitantes no número de estados do codificador ficam entre 2 e 17. Após a aplicação do software, seguido de uma criteriosa atribuição de etiquetas aos rótulos, obteve-se o codificador com número de estados igual a 5. A atribuição de etiquetas tornou possível a obtenção de um decodificador com 2 janelas de decodificação. Um dos codificadores obtidos é ilustrado na Tabela 5.21. . Por fim, as complexidades de codificação e decodificação encontradas para este código valem, respectivamente 112 e 256.

Tabela 5.21

Tabela do Codificador (5,10), taxa 1/4 com 5 Estados

	Entradas	
	0	1
1	4/0000	3/0000
2	4/0000	5/0000
3	4/0000	1/0001
4	2/0010	3/0100
5	2/1000	1/0001

Código $(d,k) = (3.9)$, com $R = 2/5$

agora à análise de codificadores com $d = 3$. Seja o código (3,9) ilustrado na . De acordo com a Tabela 2.1, a capacidade para este código vale 0,4376. ›regada, para este, caso foi $R = 2/5$. Trata-se de um código com ganho de de 1,6 ou 60%. Sua eficiência de codificação vale, aproximadamente,

a obtenção deste código utilizou-se do seguinte autovetor aproximado: 1 1 1 0).

va-se assim que a componente máxima do autovetor vale 2. Este parâmetro processo de divisão de estados é necessário e que os limitantes no número lo codificador ficarão entre 2 e 12. Durante a fase inicial de fusões, o novo btido permitiu que se reduzisse o número máximo para 5 e, no final da nero obtido caiu para 4 estados. A atribuição de etiquetas tornou possível de decodificadores com 2 janelas de decodificação. As complexidades de e decodificação encontradas para este código valem, respectivamente 112

Tabela 5.22
Tabela do Codificador (3,9), taxa 2/5 com 4 Estados

	Entradas			
	00	01	10	11
1	1/00001	3/00000	2/00010	4/00000
2	1/00001	3/00000	2/00010	2/00100
3	3/01000	4/01000	3/10000	4/10000
4	1/00001	1/10001	2/00010	2/00100

Código $(d,k) = (3.10)$, com $R = 2/5$

código (3,10) ilustrado na Tabela 5.23. De acordo com a Tabela 2.1, a ›ara este código vale 0,446. A taxa empregada, para este, caso também . Trata-se de um código com ganho em densidade de 1,6 ou 60%. Sua codificação vale, aproximadamente, 0,9.

obtenção deste código utilizou-se do seguinte autovetor aproximado: 2 1 1 0).

a-se assim que a componente máxima do autovetor vale 2. Este parâmetro processo de divisão de estados é necessário e que os limitantes no número

de estados do codificador ficarão entre 2 e 16. Durante a fase inicial de fusões, o novo autovetor obtido permitiu que se reduzisse o número máximo para 5 e, no final da busca, o número de estados obtidos caiu para 4. A atribuição de etiquetas tornou possível a obtenção de decodificadores com 2 janelas de decodificação. As complexidades de codificação e decodificação encontradas para este código valem, como no caso anterior, respectivamente 112 e 2048.

Tabela 5.23

Tabela do Codificador (3,10), taxa 2/5 com 3 Estados

	Entradas			
	00	01	10	11
1	2/00000	3/00000	1/00010	1/00001
2	1/00100	2/00100	3/01000	2/01000
3	3/10000	2/10000	1/00010	1/10001

5.2.22 Código $(d, k) = (3, 11)$, com $R = 2/5$

Seja agora o código (3,11) ilustrado na Tabela 5.24. De acordo com a Tabela 2.1, a capacidade para este código vale 0,4517. A taxa empregada, para este, caso também foi $R = 2/5$. Trata-se de um código com ganho em densidade de 1,6 ou 60%. Sua eficiência de codificação vale, aproximadamente 0,89.

Para a obtenção deste código utilizou-se do seguinte autovetor aproximado: (1 1 2 2 2 2 2 2 2 1 1 0).

Observa-se assim que a componente máxima do autovetor vale 2. Este parâmetro indica que o processo de divisão de estados é necessário e que os limitantes no número de estados do codificador ficarão entre 2 e 18. Para este código, em particular, o algoritmo de simplificação do autovetor deu bons resultados, após sua aplicação o autovetor foi reduzido para: (1 1 2 2 2 2 0 2 0 0 0 0).

Este resultado apontou agora numa redução, em termos do limitante máximo no número de estados, de 18 para 12 estados. Durante a fase inicial de fusões, o novo autovetor obtido permitiu que se reduzisse o número máximo para 4 e, no final da busca, o número de estados obtidos permaneceu em 4. A atribuição de etiquetas tornou possível a obtenção de decodificadores com 2 janelas de decodificação. As complexidades de codificação e decodificação encontradas para este código valem, como no caso anterior, respectivamente 112 e 2048.

Tabela 5.24

Tabela do Codificador (3,11), taxa 2/5 com 4 Estados

	Entradas			
	00	01	10	11
1	2/00010	1/00001	3/00000	4/00000
2	2/00010	1/00001	1/00100	3/00100
3	3/01000	4/01000	1/00100	3/00100
4	2/00010	1/00001	4/10000	3/10000

Uma situação interessante que surgiu durante várias tentativas de obtenção dos códigos dk, era que às vezes um código com restrição $k = k_1$ era obtido a partir de outro com restrição $k = k_2$, sendo $k_1 < k_2$. Esta situação ocorria sempre quando tinha-se uma eficiência de codificação tal, que permitisse que um outro código de capacidade inferior a este, porém mais próximo da taxa R pudesse ser empregado. Em outras palavras, sempre quando fosse possível aumentar a eficiência de codificação ao máximo por meio de uma redução na capacidade, respeitando-se a condição $R \leq C$. Um exemplo disso é o código (2,8) ilustrado na Tabela 5.12, este código também foi obtido a partir de um (2,10), neste caso a eficiência de codificação subiu de 0,92 para 0,94.

A possível causa para este fato surge, quando das opções para descarte de ramos em excesso, na busca pelo grau de saída desejado. É sabido que esses descartes produzem um decréscimo na capacidade do código (dentro, é claro, dos limites para $R \leq C$), este decréscimo poderá então produzir uma nova restrição k, correspondendo então a um novo código. Trata-se de uma situação de ocorrência comum quando trabalha-se com o algoritmo de blocos deslizantes [22].

Vale citar também uma outra situação peculiar com o emprego deste algoritmo. Muitos outros códigos com as restrições antes mencionadas foram obtidos e não foram neste trabalho incorporados por se tratarem de rotações destes, ou códigos equivalentes. Tratavam-se na verdade dos mesmos códificadores, o que variava entre eles era apenas a posição dos estados na tabela de codificação.

Finalizando este capítulo, é ilustrado na Tabela 5.25 um resumo das principais características dos códigos aqui obtidos. Através dela pode-se observar que, na faixa dos 20% de ganho, os menos complexos são os códigos (2,7) e (2,9); na faixa de 33,3% de ganho, o menos complexo é o código (3,7); na faixa dos 50%, os códigos (2,7), (2,8) e (2,10) com taxa 1/2, contudo, pelo menor número de estados obtido, considera-se o (2,8) como o mais simples; por fim, na faixa dos 50%, pelo mesmo motivo antes comentado, o (3,10) é o menos complexo.

Como um comentário adicional, incluiu-se ainda um código do tipo (2,9) com taxa 2/4 além do já apresentado na Tabela 5.17. Na Tabela 5.25 pode-se verificar que este código tem uma característica inversa do outro, ou seja possui um estado a menos, entretanto, sua janela de decodificação aumentou de uma unidade. Este fato foi o suficiente para elevar em 13 vezes a complexidade total do sistema. A explicação para este fato é simples, enquanto que a complexidade de codificação cresce por um fator linear, relacionado com número de estados (veja equação 3.19), a complexidade de decodificação cresce por um fator exponencial J, conforme a equação 3.20.

Tabela 5.25

Características dos códigos obtidos

(d,k)	Capacidade (C)	R (p/q)	Ganho (d+1)R	Estados	Janelas de Decodificação	C_{cod}	C_{dec}	C_T
(1,6)	0,6690	3/5	1,20	2	1	96	96	192
(1,7)	0,6793	3/5	1,20	2	1	96	96	192
(1,8)	0,6853	3/5	1,20	2	1	96	96	192
(1,9)	0,6888	3/5	1,20	2	1	96	96	192
(2,6)	0,4979	2/5	1,20	3	1	224	32	256
(2,7)	0,5174	2/5	1,20	2	1	56	32	88
(2,8)	0,5293	2/5	1,20	3	1	224	32	256
(2,9)	0,5369	2/5	1,20	2	1	56	32	88
(1,7)	0,6793	2/3	1,33	4	3	80	1024	1104
(1,9)	0,6888	2/3	1,33	4	3	80	1024	1104
(3,6)	0,3746	1/3	1,33	6	4	96	4096	4192
(3,6)	0,3746	2/6	1,33	4	2	128	8192	8320
(3,7)	0,4057	1/3	1,33	3	2	40	64	104
(3,8)	0,4251	2/6	1,33	3	1	128	128	256
(2,7)	0,5174	1/2	1,50	7	4	80	256	336
(2,8)	0,5293	1/2	1,50	5	4	80	256	336
(2,9)	0,5369	2/4	1,50	4	2	96	512	608
(2,9)	0,5369	2/4	1,50	3	3	96	8192	8288
(2,10)	0,5418	1/2	1,50	7	4	80	256	336
(2,10)	0,5418	2/4	1,50	3	2	96	512	608
(2,11)	0,5450	2/4	1,50	3	2	96	512	608
(5,10)	0,3158	1/4	1,50	5	2	112	256	368
(3,9)	0,4376	2/5	1,60	4	2	112	2048	2160
(3,10)	0,4460	2/5	1,60	3	2	112	2048	2160
(3,11)	0,4517	2/5	1,60	4	2	112	2048	2160

Capítulo 6

CONCLUSÃO

É, sem dúvida, significativa a contribuição do algoritmo dos blocos deslizantes (*sliding block algorithm*) como um procedimento sistemático na geração de códigos RLL.

Os métodos de divisão (*splitting*) e fusão (*merging*) de estados, permitem a construção de codificadores com o grau de saída desejado, respeitando-se, é claro, a condição de Shannon $R \leq C$.

O método da fusão, em particular, permite uma grande diminuição no número de estados do codificador, se aplicado tanto antes, quanto depois da fase de divisões de estado. Esta simplificação acarretará, por sua vez, em uma sensível redução no número de estados necessários à implementação do codificador.

O algoritmo constitui-se numa ferramenta eficaz na busca por códigos RLL, uma vez que é sempre possível encontra-los para uma dada restrição dk e taxa R.

Os exemplos apresentados mostraram claramente que uma boa escolha na manipulação de ramos na divisão/fusão de estados, possibilitam a obtenção de codificadores com complexidade ainda menor que os já obtidos.

Com relação ao decodificador também é importante mencionar que, dependendo das opções feitas durante a fase de divisão de estados, é possível obter códigos com mínimo tamanho da janela de decodificação. Fazendo-se uma atribuição de etiquetas cuidadosa aos ramos do diagrama de estados, pode-se obter decodificadores com reduzida janela de decodificação. A atribuição de etiquetas consegue somente alterar o valor do número de memórias, mas nunca o valor da antecipação do código. Este parâmetro pode ser alterado no momento do descarte dos ramos em excesso, e com isso, pode-se reduzir, ainda mais, o tamanho da janela.

Pode-se citar ainda o caso do algoritmo para cálculo do autovetor aproximado. Foi observado que os autovetores obtidos via algoritmo de Franaszek não são os

autovetores menores. À primeira vista poder-se-ia esperar que um autovetor menor poderia gerar codificadores mais simples, já que, ao final da fase de divisões de estado, o codificador teria um menor número de estados. Contudo, esta conclusão não parece ser suficiente, já que nada garante que esses mesmos codificadores não poderiam também ser obtidos com um autovetor mais complexo. Um exemplo disso é o código (1,7), onde foi obtido o mesmo codificador para dois autovetores com componentes máximas diferentes (3 e 5)[8].

Foi implementada e testada uma versão na subrotina AVA que obtinha o autovetor menor possível para um dado valor inicial (L). Os resultados obtidos permitiram que se constatasse uma sensível redução no tempo de execução do programa. Chegou-se, em alguns casos, a códigos tão bons quanto os anteriormente obtidos; em alguns casos até melhores, entretanto, em outros, os resultados pioraram em termos de número de estados do codificador. Uma possível explicação para estes casos, é que a técnica de simplificação do autovetor reduz o peso de alguns estados que poderiam se fundir a outros, com pesos anteriormente idênticos a estes. Nessas condições, a fusão não ocorreu e obteve-se, ao final, um codificador mais complexo. Este fato permite reforçar a hipótese de que não há garantia nenhuma que autovetores menores levem a codificadores melhores. Decidiu-se, neste caso, por deixar em aberto esta questão, podendo-se na geração de códigos optar-se por uma ou outra alternativa.

Estas questões induzem ao questionamento se, de fato, o algoritmo dos blocos deslizantes gera todos os códigos RLL possíveis para uma dada restrição dk. Alguns testes neste sentido foram realizados, sem que, contudo, fossem encontrados alguns códigos. Talvez fosse mais correto afirmar que, se fossem usados todos os autovetores possíveis isso fosse verdade, mas também não há garantias a respeito. Na verdade, a teoria deixa bem vagas estas questões, já que em nenhum ponto da bibliografia consultada, essas questões foram abordadas.

Quanto ao software, pode-se afirmar que os códigos, por ele obtidos, obedecem às restrições iniciais dk e grau de saída. Os resultados quanto ao número de estados podem ser considerados satisfatórios, em vista de terem sido encontrados na bibliografia resultados similares (à exceção do código (2,7) que foi encontrado com 5 estados).

Foi também tentada a obtenção de códigos RLL com capacidade de correção de erros. O processo consistia na redução da taxa do código fazendo-se os descartes de ramo, após a fase de divisão de estados, visando a obtenção de códigos com distância livre de Hamming maior que 1 [10]. Sabe-se, da teoria de códigos corretores de erros, que quanto maior o produto $R \times dfree$ de um código, maior será o ganho de codificação, onde $dfree$ é a distância livre de Hamming e R é a taxa do código.

Algumas tentativas foram realizadas sem grande sucesso. Foram encontrados

alguns códigos RLL com *dfree* igual a 3 ou 4, indicando uma capacidade de correção de apenas um erro, para baixas restrições. O maior problema que se enfrentou nessa fase foi a elevada complexidade dos codificadores. Não houve tempo para implementar uma rotina para cálculo do *dfree* e a operação teve que ser manual. Dessa maneira não foi possível obter bons códigos.

Finalizando, deve-se ressaltar que o software aqui apresentado serve não apenas para a obtenção de códigos RLL. Com algumas pequenas alterações nas rotinas que se relacionam com a atribuição de rótulos, ele pode ser utilizado para gerar outros códigos restritos a sistemas de estados finitos.

Como sugestão para trabalhos futuros, seria interessante a implementação de uma rotina para o cálculo do *dfree* do código empregando, por exemplo, a idéia discutida acima. Este procedimento reduziria significativamente o tempo de busca por códigos corretores de erros, e talvez conduzisse a resultados bem mais interessantes aos aqui obtidos acerca deste tipo de código.

Outra rotina interessante a ser obtida, seria para o cálculo do tamanho da janela de decodificação. Esta rotina também reduziria o tempo gasto no cálculo manual do tamanho desta janela, além de tornar completamente automática a busca por codificadores com mínima complexidade total, reunindo, numa só busca, os parâmetros, número de estados do codificador, taxa e comprimento da janela de decodificação.

Bibliografia

[1] Siegel, P.H., : " Recording codes for digital magnetic storage ", *IEEE Trans. Magn.*, vol. Mag-21, pp. 1344-1349, no. 5, Sept. 1985.

[2] Heemskerk, J.P.J. e Immink, K.A.S., : " Compact Disc: System aspects and modulation ", *Phillips Techn. Rev.*, vol. 40, no. 6, pp. 157-164, 1982.

[3] Howwell, T.D.,: " Analysis of correctable errors in the IBM 3380 disk file ", *IBM J. Res. Develop.*, vol. 28, no. 2, pp. 206-211, Mar. 1984.

[4] Lempel, A. and Cohn, M..: " Look-ahead coding for input-restricted channels ", *IEEE Trans. Inf. Theory*, vol. IT-28, pp. 933-937, Nov. 1982.

[5] Franaszek, P.A. ,: " Sequence-state encoding for digital transmission ", *Bell Syst. Tech. J.*, vol 47, pp. 143-157, jan. 1968.

[6] Immink, K.A.S.,: " Runlength-limited sequences ", *Proc. IEEE*, vol. 78, no.11, pp. 1745-1759, Nov. 1990.

[7] Adler, R.L., Coppersmith, D. and Hassner, M.,: " Algorithms for sliding block codes ", *IEEE Trans. Inf. Theory*, vol. IT-29, no. 1, pp. 5-22, Jan. 1983.

[8] Marcus, B.H., Siegel, P.H., Wolf, J.K.,: " Finite-state modulation codes for data storage ", *IEEE J. Sel. Ar. Com.*, vol. 10, no. 1, pp. 5-37, Jan. 1992.

[9] Siegel, P.H., Wolf, J.K.: "Modulation and coding for information storage",*IEEE Communications Magazine*, vol. , pp. 68-86, Dez. 91.

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

[11] Shannon, C.E., : " A mathematical theory of communication ", *Bell Syst. Tech. J.*, vol. 27, pp. 379-423, Jul. 1948.

[12] Franaszek, P.A.,: " A general method for channel coding ", *IBM J. Res. Develop.*, vol.24, pp. 638-691, 1980.

[13] Haykin, S.S.: *Digital Communications*, John Wiley & Sons, New York, 1988.

[14] Marcus, B.H. and Roth, R.M.: "Bounds on the number of states in encoder graphs for input-constrained chsnnels", *IEEE Trans. Inf. Theory*, vol. IT-37, no. 3, pp. 742-758, May. 1991.

[15] Seneta, E.: *Non-Negative Matrices and Markov Chains*, 2nd Edition, Springer-Verlag, New York, 1981.

[16] Costa, R.T. e Almeida, C.: "Algoritmo dos Blocos Deslizantes para a Geração de Códigos RLL", Anais do 11^0 Simpósio de Telecomunicações, pp. , Natal, 1993.

[17] Costa, R.T.: "Um laboratório de comunicações no computador", Anais do 10^0 Simpósio de Telecomunicações, pp. , Brasília, 1992.

[18] Marcellin,M.W. and Weber, H.J.: "Two-dimensional modulation codes", *IEEE J. Sel. Ar. Com.*, vol. 10, no. 1, pp. 254-265, Jan. 1992. RLL Capacidade

[19] Hehl, M.E.: *Linguagem de Programação Estruturada Fortran* 77, McGraw-Hill, Rio de Janeiro, 1987.

[20] Costa, R.T.: "Um laboratório de comunicações no computador", Anais do 10^0 Simpósio de Telecomunicações, pp. , Brasília, 1992.

[21] Schwartz, H. and Shaw, L.:*Signal Processing - Discrete Spectral Analysis Detection and Estimation*, McGraw-Hill, Tokyo, 1975.

[22] Wolf, J.K. and French, C.: " Alternative modulation codes for the Compact Disc ", *IEEE Trans. Cons. Eletr.*, vol. 34, no. 4, Nov. 1988.