UNIVERSIDADE FEDERAL DE PERNAMBUCO

CENTRO DE TECNOLOGIA E GEOCIÊNCIAS

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

ISAAC BENJAMIM BENCHIMOL

MÓDULO DE TRELIÇA MÍNIMO PARA CÓDIGOS CONVOLUCIONAIS

RECIFE, NOVEMBRO DE 2012.

UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE TECNOLOGIA E GEOCIÊNCIAS
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

MÓDULO DE TRELIÇA MÍNIMO PARA CÓDIGOS CONVOLUCIONAIS

ISAAC BENJAMIM BENCHIMOL

Tese submetida ao Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Pernambuco como parte dos requisitos para a obtenção do grau de **Doutor em Engenharia Elétrica.**

ORIENTADOR: CECILIO JOSÉ LINS PIMENTEL, PH.D.

Recife, Novembro de 2012.

Catalogação na fonte
Bibliotecário Marcos Aurélio Soares da Silva, CRB-4 / 1175

B457m Benchimol, Isaac Benjamim.
Módulo de treliça mínimo para códigos convolucionais / Isaac Benjamim Benchimol. - Recife: O Autor, 2012.
xii, 119 folhas, il., gráfs., tabs.

Orientador: Profº Drº. Cecilio José Lins Pimentel.
Tese (Doutorado) – Universidade Federal de Pernambuco. CTG. Programa de Pós-Graduação em Engenharia Elétrica, 2012.
Inclui Referencias e Apêndices.

1. Engenharia Elétrica. 2.Códigos Convolucionais. 3. Treliça Mínima. 4.Algoritmo de Viterbi. 5.Códigos Turbo. I. Pimentel, Cecilio José Lins (Orientador). II. Título.

621.3 CDD (22. ed.) UFPE BCTG/2012-291

Universidade Federal de Pernambuco
Pós-Graduação em Engenharia Elétrica

PARECER DA COMISSÃO EXAMINADORA DE DEFESA DE TESE DE DOUTORADO

ISAAC BENJAMIM BENCHIMOL

TÍTULO

"MÓDULO DE TRELIÇA MÍNIMO PARA CÓDIGOS CONVOLUCIONAIS"

A comissão examinadora composta pelos professores: CECÍLIO JOSÉ LINS PIMENTEL, DES/UFPE; RICHARD DEMO SOUZA, DAE/UTFPR; BARTOLOMEU FERREIRA UCHÔA FILHO, CT/UFSC; DANIEL CARVALHO DA CUNHA, CIN/UFPE e JULIANO BANDEIRA LIMA, DM/UFPE**,** sob a presidência do primeiro, consideram o candidato **ISAAC BENJAMIM BENCHIMOL APROVADO**.

Recife, 22 de novembro de 2012.

GERALDO LEITE TORRES Vice-Coordenador do PPGEE	**CECÍLIO JOSÉ LINS PIMENTEL** Orientador e Membro Titular Interno
BARTOLOMEU FERREIRA UCHÔA FILHO Membro Titular Externo	**RICHARD DEMO SOUZA** Membro Titular Externo
DANIEL CARVALHO DA CUNHA Membro Titular Externo	**JULIANO BANDEIRA LIMA** Membro Titular Externo

à memória de meus pais,

Benjamin e Gracília

Shall memory restore
The steps and the shore,
The face and the meeting place;

W. H. AUDEN
(1907-1973)

AGRADECIMENTOS

A minha filha Julianna pelo carinho, amor e dedicação aos estudos.

Aos amigos do doutorado pelo companheirismo. Em especial ao Victor e Ednelson pelos estudos e trabalhos em grupo.

À Aldevandra pelo apoio e torcida.

À UFPE e UEA pela oportunidade que me concederam.

Aos professores do PPGEE da UFPE pelos conhecimentos transmitidos e momentos de descontração.

Ao IFAM pelo incentivo e oportunidade.

À FAPEAM pelo apoio financeiro.

Em especial ao professor Cecilio José Lins Pimentel pela constante e valorosa orientação, disponibilidade e paciência, o meu muito obrigado.

ISAAC BENJAMIM BENCHIMOL

Recife, Novembro de 2012.

Resumo da Tese apresentada à UFPE como parte dos requisitos necessários
para a obtenção do grau de Doutor em Engenharia Elétrica

MÓDULO DE TRELIÇA MÍNIMO PARA CÓDIGOS CONVOLUCIONAIS

Isaac Benjamim Benchimol

Novembro/2012

Orientador: Cecilio José Lins Pimentel, Ph.D.
Área de Concentração: Comunicações
Palavras-chave: Códigos convolucionais, treliça mínima, complexidade de decodificação, algoritmo de Viterbi, seccionamento de treliça, codificadores sistemáticos recursivos, códigos turbo.
Número de Páginas: XII+119

Esta tese apresenta uma medida de complexidade computacional para códigos convolucionais adequada para receptores que implementam o algoritmo de Viterbi em software. A definição desta complexidade envolve a determinação do número de operações aritméticas executadas em um módulo de treliça durante a decodificação, a implementação destas em uma arquitetura de processadores digitais de sinais e a avaliação do respectivo custo computacional de cada operação. Na sequência, esta medida é utilizada para avaliar o impacto do seccionamento do módulo de treliça mínimo. Um conjunto de regras é introduzido para construir padrões de seccionamento que resultem em estruturas de treliça mais compactas e regulares e de mesma complexidade da treliça mínima, constituindo uma alternativa de interesse em aplicações práticas. Finalmente, este trabalho apresenta um método para a construção do módulo de treliça mínimo para codificadores convolucionais sistemáticos recursivos adotados em esquemas turbo. Esta abordagem contribui para a redução da complexidade de decodificação de um decodificador turbo típico operando com codificadores constituintes de taxas altas. Uma busca de códigos é realizada e obtém-se um refinamento da relação complexidade de decodificação versus distância livre efetiva do código turbo.

Abstract of Thesis presented to UFPE as a partial fulfillment of the requirements for the degree of Doctor in Electrical Engineering.

MINIMAL TRELLIS MODULE FOR CONVOLUTIONAL CODES

Isaac Benjamim Benchimol

November/2012

Supervisor: Cecilio José Lins Pimentel, Ph.D.

Area of Concentration: Communications

Keywords: Convolutional codes, minimal trellis, decoding complexity, Viterbi algorithm, trellis sectionalization, recursive systematic encoders, turbo codes.

Number of Pages: XII+119

This thesis presents a computational complexity measure for convolutional codes suitable for receivers that implement the Viterbi algorithm by software. The definition of this complexity involves the determination of the number of arithmetic operations performed by a trellis module during decoding, the implementation of these operations by a digital signal processor architecture and the evaluation of the computational cost of each operation. Then, this measure is used to analyze the impact of sectionalization of the minimal trellis module. A set of rules is introduced to construct sectionalization patterns which yield more compact and regular trellis structures of same complexity of the minimal trellis module, providing interesting alternatives for practical applications. Finally, this work presents a method for construction of the minimal trellis module for a systematic recursive convolutional encoding to be adopted in turbo schemes. This approach contributes for reduction of the decoding complexity of a typical turbo decoder operating with high-rate constituent encoders. A code search is conducted and a more refined decoding complexity versus effective free distance of the turbo code is achieved.

SUMÁRIO

LISTA DE TABELAS

LISTA DE FIGURAS

CAPÍTULO 1

INTRODUÇÃO

A transmissão de dados na forma digital permite o uso de técnicas de processamento digital de sinais. Um exemplo destas técnicas são os códigos corretores de erros que aumentam a confiabilidade da informação transmitida [1]. Sistemas como os de telefonia celular, TV digital, redes sem fio, WIMAX, entre outros, utilizam diversos tipos de códigos corretores de erros, tais como, códigos de bloco, códigos convolucionais, códigos turbo e LDPC. Códigos convolucionais e códigos turbo estão entre os mais usados em aplicações práticas, inclusive em padrões mais recentes de comunicações sem fios como WIMAX [2], EDGE [3] e LTE [4].

A decodificação possui um papel importante no desempenho de códigos corretores de erros. Em particular, a decodificação de códigos convolucionais ainda requer um alto custo de processamento e consumo de energia pelo receptor [5][6]. Se o algoritmo de Viterbi (VA) for utilizado, a decodificação de um código convolucional consome 76% do processamento de um receptor HYPERLAN/2 [7][8]. Diferentes implementações de receptores compatíveis com o padrão IEEE 802.11 foram analisadas em [9] e mostram que o VA contribui com 35% do consumo de energia total. Este consumo está fortemente relacionado com a complexidade de decodificação requerida pela estrutura de treliça que representa o código. Desta forma, buscar alternativas de decodificação menos complexas é essencial para alguns sistemas de comunicações, sobretudo para os que têm severas limitações de energia, como aqueles que envolvem implantes biomédicos e redes de sensores sem fio [10]. Uma pesquisa recente na literatura mostra uma série de trabalhos

relacionados com esta questão [10-24]. Pode-se dividir estes trabalhos em três grupos diferentes, de acordo com a abordagem utilizada. O primeiro grupo trata de questões específicas da implementação em hardware cujo projeto acelera o cálculo das operações envolvidas na decodificação [10][11]. O segundo grupo propõe aplicações ou extensões de métodos de decodificação sub-ótimos [12-15]. O terceiro trata de projeto de códigos baseado em representações mais simples de treliça [16-24]. Este trabalho tem enfoque no terceiro grupo: representação de treliça de baixa complexidade.

Um código convolucional de taxa $R = k/n$ e comprimento de restrição v pode ser representado por várias treliças. Esta é uma estrutura periódica, sendo o menor período denominado de módulo de treliça [17][21][25]. Tradicionalmente, um código é representado por um módulo de treliça convencional com 2^k ramos divergindo de cada estado e n bits rotulando cada ramo. O número de operações aritméticas associadas ao módulo de treliça, tais como somas e comparações, executadas pelo VA constitui a complexidade computacional da operação de decodificação de um código [26]. A complexidade do VA operando no módulo convencional cresce exponencialmente com k e v.

O módulo de treliça mínimo foi desenvolvido para códigos convolucionais não-sistemáticos não-recursivos por Sidorenko e Zyablov [27] e por McEliece e Lin [25] e é baseado na construção de treliças BCJR para códigos de blocos [28]. Este módulo tem uma estrutura irregular com número de estados em cada seção periodicamente variante no tempo em que várias medidas de complexidade de módulo de treliça são minimizadas, como, por exemplo, o número total de estados e o número total de ramos [29]. Como consequência, as operações aritméticas realizadas pelo VA sobre este módulo também são minimizadas. Tal módulo apresenta n seções, em que cada ramo que conecta 2 estados numa seção é rotulado por 1 bit da palavra código. A busca de códigos convolucionais com boas propriedades de distância e complexidade do módulo mínimo fixa foi realizada em [21][23].

Um código em particular pode ter uma representação de treliça mais compacta e regular (do ponto de vista do número máximo de estados e do número de seções) com mesmo espectro de distâncias em relação ao módulo de treliça mínimo. Isso pode ser alcançado através da aplicação sistemática da técnica de seccionamento [30] com pouco ou nenhum impacto na complexidade de decodificação do VA. Várias topologias de treliça de

códigos propostas na literatura, tais como códigos convolucionais puncionados (PCCs) [16] e aquelas propostas em [23], podem ser membros de famílias de módulos de treliça mínimo seccionados. Uma nova topologia de treliça que utiliza o PCC juntamente com a técnica de supressão de ramos é abordada em [26]. Este trabalho utiliza o seccionamento e obtém uma variedade de novas topologias de treliça que podem ser menos complexas e, consequentemente, reduzir o consumo de energia total do receptor (se comparadas com a implementação em treliça convencional), um tema importante abordado na literatura atual [6][31]. Além disso, os módulos de treliça seccionados podem reduzir a complexidade de decodificação quando utilizadas por algoritmos de decodificação como SOVA [32], BCJR [33] e Algoritmo-M [5].

Outro enfoque deste trabalho são os códigos turbo. Introduzidos mais recentemente, em 1993, estes códigos apresentam excelente desempenho, próximo ao limite teórico estabelecido pela Teoria da Informação e, por este motivo, são utilizados por aplicações que requerem alta confiabilidade. Para aumentar a taxa do esquema de codificação turbo típico pode-se utilizar a técnica de puncionamento ou utilizar codificadores convolucionais sistemáticos recursivos de taxas altas como codificadores constituintes. Em [34,] são apresentadas algumas vantagens destes sobre aqueles, tais como melhor convergência do processo iterativo, maiores distâncias mínimas e maior robustez do decodificador. Entretanto, até este momento não há na literatura uma abordagem que permita utilizar o módulo de treliça mínimo apresentado em [25][27] para decodificação em esquemas turbo. Este trabalho apresenta um novo método para a construção de um módulo de treliça mínimo, de tal forma que os requisitos de mapeamento sistemático recursivo da codificação turbo sejam satisfeitos, possibilitando sua adoção por códigos turbo de taxas altas e permitindo reduzir a complexidade de decodificação associada a estes códigos.

1.1 Objetivos

Há na literatura várias definições de complexidade de um módulo de treliça para códigos convolucionais. A mais adotada foi definida por McEliece e Lin e baseia-se no número total de bits que rotulam os ramos de um módulo de treliça (normalizado pelo número de bits de informação), denominada de complexidade de treliça de um módulo M, denotada por $TC(M)$. Esta medida representa a complexidade aditiva do VA.

Neste trabalho, propomos uma medida de complexidade do VA, denominada de complexidade computacional de M, denotada por $TCC(M)$, que reflete mais adequadamente o esforço computacional de decodificação do VA em aplicações em que o receptor é implementado por software. Esta medida é baseada no número de operações aritméticas de soma e comparação executadas pela decodificação. Consideramos a implementação destas operações utilizando-se a arquitetura de processadores digitais de sinais de ponto-fixo TMS320C55xx da Texas Instruments, e seus custos (complexidades) computacionais são medidos em termos do número de ciclos de máquina consumidos pela execução. Uma relação entre a complexidade definida neste trabalho e a complexidade de treliça definida por McEliece e Lin é investigada sobre os módulos de treliça convencional e mínimo. Uma busca por melhores códigos (em termos de espectro de distâncias) é realizada utilizando-se a $TCC(M)$ como referência.

O seccionamento do módulo de treliça mínimo também é avaliado. Há 2^{n-1} possibilidades de seccionar um módulo de treliça mínimo, obtendo-se uma treliça seccionada com n' seções, $n' = n-1,...,1$. O problema principal desta técnica consiste em encontrar a melhor escolha, dentre todas as possibilidades, que minimize certa medida de complexidade. Pretendemos avaliar o impacto que o seccionamento produz sobre as complexidades $TC(M)$ e $TCC(M)$. Um estudo é conduzido para determinar a treliça mais compacta (menor n') e regular com as mesmas complexidades do módulo de treliça mínimo. A construção de um conjunto de regras que norteiam o seccionamento auxilia esta tarefa. Novas topologias de treliças propostas neste trabalho constituem boas alternativas às treliças usadas em aplicações práticas. Várias taxas são consideradas neste estudo.

A construção do módulo de treliça mínimo para codificadores convolucionais sistemáticos recursivos utilizados em códigos turbo é abordada. Deseja-se reduzir a complexidade de decodificação destes códigos operando com codificadores constituintes de taxas altas. A complexidade de decodificação do algoritmo max-log-MAP é avaliada em termos do número de comparações, adições e multiplicações. Finalmente, com o intuito de obter um refinamento em termos de complexidade de decodificação-desempenho do código, realizamos uma busca com base no procedimento proposto em [21] e listamos bons códigos (em termos de distância livre efetiva) com novas complexidades em relação aos listados em [18]. As publicações geradas por esta tese são listadas no Apêndice D.

1.2 Estrutura da Tese

Este trabalho é dividido em seis capítulos. Este primeiro capítulo traz uma abordagem geral sobre o trabalho e apresenta os objetivos a serem buscados. O segundo capítulo apresenta algumas definições sobre códigos convolucionais e suas representações em treliça. Em particular, três estruturas são consideradas: a treliça convencional, a treliça puncionada e a treliça mínima. O conceito de complexidade de treliça é revisado. No terceiro capítulo, analisamos as operações requeridas pelo VA, seus respectivos custos computacionais e definimos uma nova medida de complexidade computacional de um módulo de treliça. Tabelas são apresentadas com o resultado da busca de bons códigos convolucionais com base nesta complexidade. O quarto capítulo apresenta a técnica de seccionamento do módulo de treliça mínimo e utiliza a complexidade computacional definida no capítulo anterior como critério de avaliação da complexidade da treliça resultante. A construção das regras de seccionamento é descrita. Tabelas são apresentadas com padrões de seccionamento de treliças mais compactas e de mesma complexidade da treliça mínima, para códigos de diversas taxas. No quinto capítulo, apresentamos os fundamentos de códigos e codificadores convolucionais sistemáticos recursivos em esquemas turbo. Um método utilizado para a construção da treliça mínima para estes codificadores é introduzido e a complexidade de decodificação do algoritmo Max-log-MAP é analisada. Também mostramos neste capítulo o resultado da busca de bons codificadores sistemáticos recursivos de taxas diferentes de $(n-1)/n$ para serem utilizados em códigos turbo. Finalmente, o sexto capítulo apresenta as conclusões e as sugestões de trabalhos futuros.

CAPÍTULO 2

MÓDULOS DE TRELIÇA DE CÓDIGOS CONVOLUCIONAIS

Este capítulo apresenta algumas definições sobre códigos convolucionais e suas representações em treliça. Em particular, três estruturas serão consideradas: a treliça convencional, a treliça puncionada e a treliça mínima. O conceito de complexidade de treliça é introduzido.

2.1 Codificador e Código Convolucional

A sequência de informação de um codificador convolucional pode ser escrita na forma $\mathbf{u} = (\mathbf{u}_0, \mathbf{u}_1, ...) = (u_0^{(1)}, u_0^{(2)}, ..., u_0^{(k)}, u_1^{(1)}, u_1^{(2)}, ..., u_1^{(k)}, ...)$, em que $\mathbf{u}_t = (u_t^{(1)}, ..., u_t^{(k)})$ é o bloco de informação de comprimento k no instante de tempo t. Similarmente, a sequência codificada, chamada de palavra-código, é $\mathbf{v} = (\mathbf{v}_0, \mathbf{v}_1, ...) = (v_0^{(1)}, v_0^{(2)}, ..., v_0^{(n)}, v_1^{(1)}, v_1^{(2)}, ..., v_1^{(n)}, ...)$ em que $\mathbf{v}_t = (v_t^{(1)}, ..., v_t^{(n)})$ é o bloco codificado de comprimento n no instante de tempo t. Cada bloco $\mathbf{v}_t$ depende do bloco $\mathbf{u}_t$ bem como de m blocos de informação passados $\mathbf{u}_{t-1}, ..., \mathbf{u}_{t-m}$. O parâmetro m é chamado de memória do codificador. O número total de elementos de memória para armazenar os bits passados utilizados pelo codificador é denotado por ν. O código gerado por um codificador convolucional é chamado de código convolucional $C(n,k,\nu)$. O código convolucional é definido como o

conjunto de palavras-código geradas pelo codificador para todas as possíveis sequências de informação. A taxa de um código é $R = k/n$.

A Figura 2.1 mostra um codificador convolucional de um código $C(3,2,2)$. Este codificador é formado por elementos de memória e somadores módulo-2. Neste exemplo, a cada instante de tempo t, o codificador recebe como entrada um bloco de informação $\mathbf{u}_t = (u_t^{(1)}, u_t^{(2)})$ e gera na saída um bloco codificado $\mathbf{v}_t = (v_t^{(1)}, v_t^{(2)}, v_t^{(3)})$, portanto $k = 2$, $n = 3$ e $R = 2/3$. A operação do codificador convolucional da Figura 2.1 pode ser representada através das seguintes equações:

$$
\begin{aligned}
v_t^{(1)} &= u_t^{(1)} + u_{t-1}^{(1)} + u_{t-1}^{(2)} \\
v_t^{(2)} &= u_t^{(1)} + u_{t-1}^{(1)} \\
v_t^{(3)} &= u_t^{(1)} + u_t^{(2)} + u_{t-1}^{(2)}
\end{aligned}
$$

em que o símbolo (+) significa adição módulo-2.

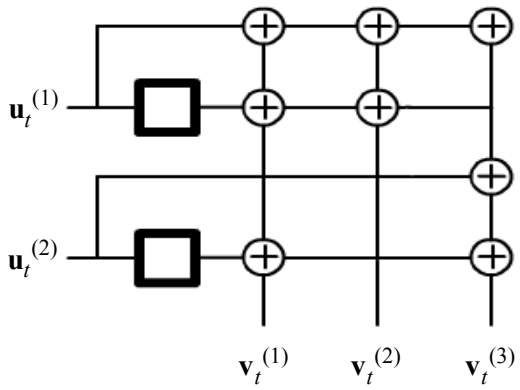


Figura 2.1 – Codificador convolucional de um código $C(3,2,2)$.

Um codificador convolucional de taxa $R = k/n$ é representado por nk sequências de geradores de comprimento $m+1$, $g_i^{(j)} = (g_{i,0}^{(j)}, g_{i,1}^{(j)}, ..., g_{i,m}^{(j)})$ para $i = 1,...,k$, $j = 1,...,n$. As sequências de geradores $g_i^{(j)}$ são as respostas ao impulso do codificador. Estas respostas são obtidas quando a entrada é $\mathbf{u}^{(i)} = (1000...)$ para $i = 1,...,k$ e observa-se a sequência de saída $\mathbf{v}^{(j)}$ de comprimento $m+1$ para $j = 1,...,n$. Para o codificador da Figura 2.1 tem-se

$$
\begin{aligned}
g_1^{(1)} &= (1,1) \quad g_1^{(2)} = (1,1) \quad g_1^{(3)} = (1,0) \\
g_2^{(1)} &= (0,1) \quad g_2^{(2)} = (0,0) \quad g_2^{(3)} = (1,1).
\end{aligned}
$$

Assim, a codificação convolucional pode ser escrita como

$$\mathbf{v} = \mathbf{u}\mathrm{G}_{escalar}$$

em que a matriz $G_{escalar}$ é uma matriz geradora semi-infinita da forma

$$G_{escalar} = \begin{pmatrix} G_0 & G_1 & G_2 & \cdots & G_m & & & \\ & G_0 & G_1 & \cdots & G_{m-1} & G_m & & \\ & & G_0 & \cdots & G_{m-2} & G_{m-1} & G_m & \\ & & & \ddots & \ddots & \ddots & \ddots & \ddots \end{pmatrix} \tag{2.1}$$

em que os espaços em branco indicam zeros e as n x k submatrizes geradoras são da forma

$$G_l = \begin{pmatrix} g_{1,l}^{(1)} & g_{1,l}^{(2)} & \cdots & g_{1,l}^{(n)} \\ g_{2,l}^{(1)} & g_{2,l}^{(2)} & \cdots & g_{2,l}^{(n)} \\ \vdots & \vdots & \cdots & \vdots \\ g_{k,l}^{(1)} & g_{k,l}^{(2)} & \cdots & g_{k,l}^{(n)} \end{pmatrix}$$

$l = 0,...,m.$

Exemplo 2.1: Para o codificador da Figura 2.1, tem-se $m = 1,$ portanto as matrizes G_0, G_1 e $G_{escalar}$ são dadas por

$$G_0 = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} \qquad G_1 = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}$$

$$G_{escalar} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 0 & & & & & & \\ 0 & 0 & 1 & 1 & 0 & 1 & & & & & & \\ & & & 1 & 1 & 1 & 1 & 1 & 0 & & & \\ & & & 0 & 0 & 1 & 1 & 0 & 1 & & & \\ & & & & & & 1 & 1 & 1 & 1 & 1 & 0 \\ & & & & & & 0 & 0 & 1 & 1 & 0 & 1 \\ & & & & & & & & & \ddots & & \end{pmatrix}.$$

2.1.1 Representação Polinomial

As sequências de geradores $g_i^{(j)}$ com $i = 1,...,k$ e $j = 1,...,n$ são finitas e podem ser representadas como polinômios de grau finito em um operador de retardo *D*. Os polinômios geradores para um codificador $C(n,k,\nu)$ são $g_i^j(D) = g_{i,0}^{(j)} + g_{i,1}^{(j)}D + ... + g_{i,m}^{(j)}D^m$ para $i = 1,...,k$ e $j = 1,...,n$.

As sequências de entrada e saída do codificador podem ser escritas como polinômios da forma $\mathbf{u}^{(i)}(D) = u_0^{(i)} + u_1^{(i)}D + u_2^{(i)}D^2 + ...$ e $\mathbf{v}^{(j)}(D) = v_0^{(j)} + v_1^{(j)}D + v_2^{(j)}D^2 + ...$, respectivamente. A operação de codificação é dada por

$$\mathbf{v}(D) = \mathbf{u}(D)\mathrm{G}(D)$$

em que o vetor $\mathbf{u}(D) = [\mathbf{u}_t^{(1)}(D),...,\mathbf{u}_t^{(k)}(D)]$ tem k componentes $\mathbf{u}^{(i)}(D)$, para $i = 1,...,k$ e o vetor $\mathbf{v}(D) = [\mathbf{v}_t^{(1)}(D),...,\mathbf{v}_t^{(n)}(D)]$ tem n componentes $\mathbf{v}^{(j)}(D)$, para $j = 1,...,n$. A matriz $\mathrm{G}(D)$ de dimensão $k \times n$ é chamada de matriz geradora polinomial, ou simplesmente de matriz geradora, e é dada por

$$\mathrm{G}(D) = \begin{pmatrix} g_1^{(1)}(D) & g_1^{(2)}(D) & \cdots & g_1^{(n)}(D) \\ g_2^{(1)}(D) & g_2^{(2)}(D) & \cdots & g_2^{(n)}(D) \\ \vdots & \vdots & \cdots & \vdots \\ g_k^{(1)}(D) & g_k^{(2)}(D) & \cdots & g_k^{(n)}(D) \end{pmatrix}.$$

O codificador da Figura 2.1 possui matriz geradora dada por

$$G(D) = \begin{pmatrix} 1+D & 1+D & 1 \\ D & 0 & 1+D \end{pmatrix}.$$

2.1.2 Parâmetros de Codificadores Convolucionais

A seguir, são apresentadas algumas definições relativas a um codificador convolucional que serão úteis no restante deste trabalho.

Definição 2.1: O comprimento do *i-ésimo* elemento de memória é

$$\nu_i = \max_j [grau(g_i^{(j)}(D))].$$

Definição 2.2: O comprimento de restrição total é

$$\nu = \sum_{i=1}^{k} \nu_i .$$

Definição 2.3: A memória do codificador é

$$m = \max_{1 \le i \le k} [\nu_i].$$

2.1.3 Diagrama de Estados

O diagrama de estados é outra forma de representar um codificador convolucional. Um estado representa os bits armazenados nos elementos de memória do codificador no instante de tempo t. O número de estados distintos é igual a 2^{ν}. O diagrama de estados mostra a transição entre os estados, os k bits do bloco de informação que provocam cada transição e os n bits do bloco de saída do codificador [35][36]. Os rótulos dos estados são definidos pelos valores atuais das memórias. A Figura 2.2 mostra o diagrama de estados do codificador da Figura 2.1 com 4 estados. Neste diagrama, foram suprimidos os bits de informação que causam as transições. Os valores dos $n = 3$ bits nos ramos, formados por $v_t^{(1)} v_t^{(2)} v_t^{(3)}$, representam o bloco de saída gerado pela transição de estados entre o tempo $t-1$ e t.

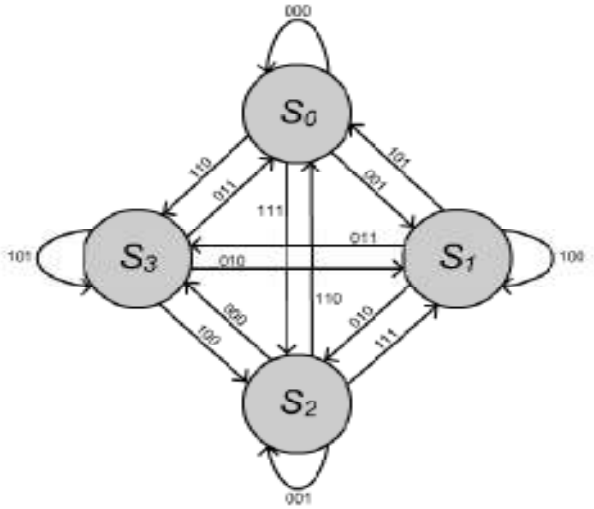


Figura 2.2 - Diagrama de estados do código $C(3,2,2)$.

A distância livre d_f de um código convolucional é definida como o caminho de menor peso de *Hamming* que diverge do estado S_0 em $t=0$ e volta para S_0 pela primeira vez em algum instante futuro e permanece neste estado [35]. No diagrama da Figura 2.2, a palavra-código equivalente às transições de estados $S_0 \to S_1 \to S_0 \to S_0 \to ...$ é dada por 001101000... Qualquer outro caminho que inicie e retorne a S_0 possui peso de *Hamming* igual ou maior que 3, portanto $d_f = 3$. Também pode-se determinar a distância livre mínima de um código convolucional a partir de sua representação em treliça, a ser apresentada na Seção 2.2.

O espectro de distância de um código convolucional é definido pela sequência $\boldsymbol{N} = (N_{d_f}, N_{d_f+1}, N_{d_f+2}, ...)$ em que o $i-ésimo$ número nesta sequência corresponde ao número de palavras-códigos com peso de *Hamming* $d_f + i - 1$. O melhor código é aquele que apresentar a maior distância de *Hamming* d_f. Empates são resolvidos tomando-se o código com o menor espectro de distância em ordem lexicográfica.

2.1.4 Codificadores Catastróficos, Sistemáticos e Básico-Mínimo

Codificadores convolucionais que geram uma palavra-código de peso finito para uma sequência de informação de peso infinito são chamados de codificadores catastróficos [37]. Nestes codificadores, um número finito de erros de canal pode causar um número infinito de erros de decodificação.

Definição 2.4: Seja $wt(\mathbf{u})$ o peso de *Hamming* da sequência de informação $\mathbf{u}$. Um codificador convolucional com matriz geradora $G(D)$ é catastrófico se existir uma sequência $\mathbf{u}'$ tal que $wt(\mathbf{u}') = \infty$ e o peso de *Hamming* da palavra-código $wt(\mathbf{u}'G(D)) < \infty$.

Teorema 2.1 [37][38]: Uma matriz geradora $G(D)$ é não-catastrófica se, e somente se, existir uma matriz inversa à direita $G^{-1}(D)$ apresentando apenas entradas polinomiais.

Exemplo 2.2: Considere a matriz geradora

$$G(D) = \begin{pmatrix} 1 & D^2 & D \\ D & 1 & 0 \end{pmatrix}.$$

Uma matriz inversa à direita para $G(D)$ é

$$G^{-1}(D) = \begin{pmatrix} 1 & D \\ D^2 & 1+D^2 \\ D^2 & 1+D+D^3 \end{pmatrix}.$$

Observa-se que $G^{-1}(D)$ é uma matriz polinomial, ou seja, apresenta apenas entradas polinomiais, portanto $G(D)$ é não-catastrófica.

Definição 2.5 [39]: Duas matrizes geradoras $G(D)$ e $G(D)'$ são equivalentes se ambas geram o mesmo código convolucional. Duas matrizes geradoras $G(D)$ e $G(D)'$ são equivalentes se existir uma matriz inversível $T(D)$ tal que $G(D) = T(D)G(D)'$.

Definição 2.6: Uma matriz geradora $G(D)$ é básica se $G(D)$ é polinomial e existir pelo menos uma matriz inversa à direita $G^{-1}(D)$ polinomial.

A matriz geradora $G(D)$ do Exemplo 2.2 é básica.

Definição 2.7: Uma matriz geradora é básica-mínima se v é mínimo sobre todas as matrizes geradoras equivalentes básicas [37][38].

Teorema 2.2 [37][38]: A matriz binária $\left[G(D)\right]_h$ possui elemento (i, j) igual a 1 se $grau[g_i^{(j)}(D)] = v_i$, ou 0 caso contrário. A matriz geradora $G(D)$ é básica-mínima se $\left[G(D)\right]_h$ é de posto cheio (do inglês, *full-rank).*

Exemplo 2.3: Considere a matriz geradora básica

$$G(D) = \begin{pmatrix} 1+D^3 & 0 & 0 & 1+D+D^3 \\ D^2+D^3+D^4+D^5 & 0 & 1 & 1+D+D^4+D^5 \\ D^3 & 1 & 1 & D+D^3 \end{pmatrix}.$$

Observa-se que $G(D)$ apresenta $v = 11$ e

$$[G(D)]_h = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \end{pmatrix}$$

não é de posto cheio. Logo, $G(D)$ não é básica-mínima. Aplicando o algoritmo proposto em [37][38] para construir a matriz básica-mínima $G_{mb}(D)$ equivalente a $G(D)$, temos

$$G_{mb}(D)=\begin{pmatrix} 1 & 0 & 1 & 1 \\ 1+D & 0 & 1 & D \\ 1+D^2 & 0 & D^2 & 1+D \end{pmatrix} \quad \text{e} \quad [G_{mb}(D)]_h=\begin{pmatrix} 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix}.$$

$G_{mb}(D)$ apresenta $v=3$ e $[G_{mb}(D)]_h$ é de posto cheio, logo $G_{mb}(D)$ é básica-mínima equivalente a $G(D)$.

Definição 2.8 [39]: Dado um codificador $G(D)$, pode-se encontrar um codificador equivalente sistemático com matriz $G_{sys}(D)$ tal que $G_{sys}(D)=T(D)G(D)$, sendo $T^{-1}(D)$ a submatriz k x k de $G(D)$. A matriz $G_{sys}(D)$ deve ser da forma

$$G_{sys}(D)=\begin{bmatrix} I & P(D) \end{bmatrix}$$

em que I é a matriz identidade k x k e $P(D)$ é uma matriz racional k x $(n-k)$.

Exemplo 2.4: Considere a matriz geradora $G(D)$ dada por

$$G(D)=\begin{pmatrix} 1+D^2 & 0 & 0 & 1 \\ 1+D+D^2 & 0 & 1 & 0 \\ D^2 & 1 & 1 & 0 \end{pmatrix}$$

e $T^{-1}(D)$ é a submatriz formada pelas três primeiras colunas de $G(D)$, tal que

$$T^{-1}(D)=\begin{pmatrix} 1+D^2 & 0 & 0 \\ 1+D+D^2 & 0 & 1 \\ D^2 & 1 & 1 \end{pmatrix}.$$

Com isto

$$T(D)=\frac{1}{1+D^2}\begin{pmatrix} 1 & 0 & 0 \\ 1+D & 1+D^2 & 1+D^2 \\ 1+D+D^2 & 1+D^2 & 0 \end{pmatrix}.$$

Portanto,

$$G_{sys}(D)=T(D)G(D)=\begin{pmatrix} 1 & 0 & 0 & \dfrac{1}{1+D^2} \\ 0 & 1 & 0 & \dfrac{1+D}{1+D^2} \\ 0 & 0 & 1 & \dfrac{1+D+D^2}{1+D^2} \end{pmatrix}$$

é o codificador sistemático equivalente a $G(D)$.

Nas próximas seções serão abordadas as representações em treliça de códigos convolucionais. Três estruturas de treliça serão consideradas: a treliça convencional, a treliça puncionada e a treliça mínima.

2.2 Módulo de Treliça e Complexidade

Todo código convolucional $C(n,k,\nu)$ pode ser representado por uma treliça semi-infinita que (afora um transitório inicial) é periódica, sendo o menor período chamado de módulo da treliça. Em geral, um módulo de treliça M de um código convolucional consiste de n' seções, 2^{v_t} estados na profundidade t, 2^{b_t} ramos conectando os estados entre as profundidades t e $t+1$, e l_t bits rotulando cada ramo entre as profundidades t e $t+1$, para $0 \le t \le n'-1$ [21].

Códigos convolucionais são, em geral, decodificados através do algoritmo de Viterbi (VA) [35][36]. Em [25], foi proposta uma medida de complexidade de treliça que representa o esforço da decodificação por bit requerido pelo VA sobre um módulo de treliça M. Essa medida é o número total de bits que rotulam os ramos de M (normalizada por k) e é chamada de complexidade de treliça, denotada por $TC(M)$, e é definida por [25]

$$TC(M) = \frac{1}{k}\sum_{t=0}^{n'-1} l_t 2^{v_t+b_t} \tag{2.2}$$

símbolos por bit.

2.2.1 Módulo Convencional

Um módulo de treliça convencional, denominado M_{conv}, consiste de uma seção com 2^v estados iniciais e finais e 2^k ramos divergindo de cada estado inicial (e convergindo para um estado final). Cada ramo é rotulado por uma sequência de n bits, que representa um bloco codificado produzido pelo codificador em resposta a uma transição de estados. Assim, para o módulo de treliça convencional, $n'=1$, $\nu_0 = \nu_1 = \nu$, $l_t = n$. A complexidade de treliça $TC(M_{conv})$ pode ser obtida reescrevendo-se (2.2) da seguinte forma:

$$TC(M_{conv}) = \frac{n}{k} 2^{k+v}. \tag{2.3}$$

Exemplo 2.5: Considere o código convolucional $C(5,2,2)$ com matriz geradora dada por

$$G(D) = \begin{pmatrix} 1+D & 1+D & 1 & 1 & 0 \\ 0 & D & 1+D & 1+D & 1+D \end{pmatrix}.$$

Para esse código, $n=5$, $k=2$ e $\nu=2$. O módulo de treliça convencional M_{conv} é mostrado na Figura 2.3 e possui 4 estados iniciais conectando-se com 4 estados finais. Os rótulos dos ramos são compostos por *n*=5 bits e foram suprimidos da Figura 2.3. Para o código $C(5,2,2)$, obtemos $TC(M_{conv}) = 40$ símbolos por bit.

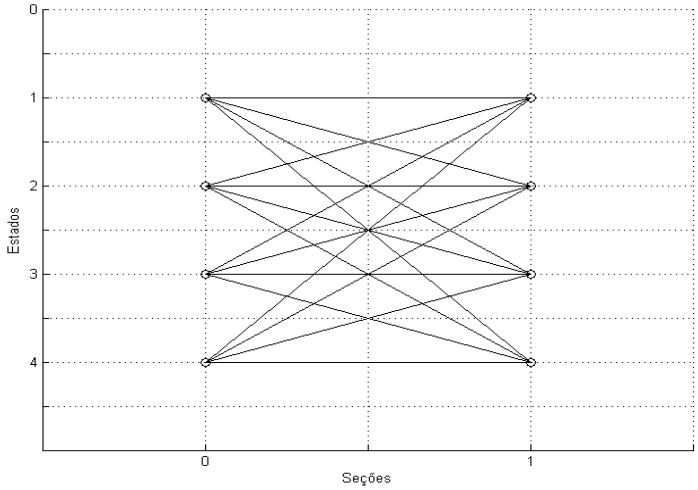


Figura 2.3 – Módulo de treliça convencional do código $C(5,2,2)$ do Exemplo 2.5.

2.2.2 Módulo Puncionado

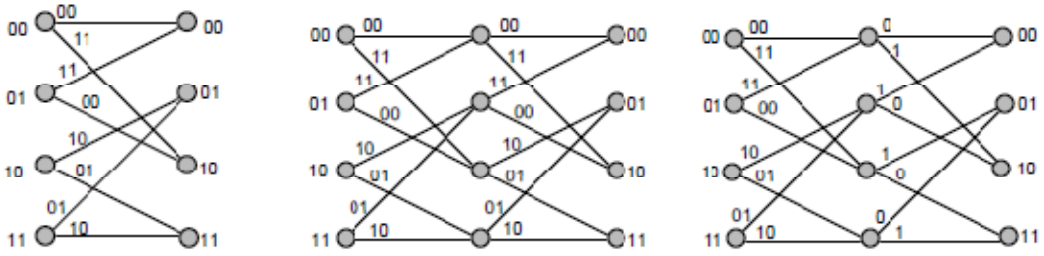
Alguns códigos podem ser representados por um módulo de treliça puncionado. Para obter um módulo de treliça puncionado, denominado M_{PCC}, parte-se de um módulo convencional de um código convolucional $C(N,1,\nu)$, chamado de código-mãe, e replica-se este módulo k vezes, resultando num código convolucional $C(Nk,k,\nu)$. Se forem puncionados (removidos) $Nk-n$ bits de cada bloco codificado, é obtido um código convolucional puncionado $C(n,k,\nu)$. Portanto, o código puncionado é representado por um módulo de treliça formado por *k* cópias do módulo de treliça do código-mãe $C(N,1,\nu)$. A complexidade de treliça $TC(M_{PCC})$ pode ser obtida a partir de (2.2) da seguinte forma

$$TC(M_{PCC}) = \frac{n}{k} 2^{\nu+1}. \tag{2.4}$$

Exemplo 2.6: Considere o código convolucional $C(2,1,2)$ [25] com matriz geradora dada por

$$G(D) = \begin{pmatrix} 1+D+D^2 & 1+D^2 \end{pmatrix}. \tag{2.5}$$

O módulo de treliça convencional para este código é mostrado na Figura 2.4(a). Depois, mais um módulo deste mesmo código é concatenado ao primeiro obtendo-se o módulo para um código $C'(4,2,2)$, mostrado na Figura 2.4(b), formado por duas seções idênticas à mostrada na Figura 2.4(a). Puncionando-se o segundo bit dos ramos da segunda seção da Figura 2.4(b) obtemos o código $C''(3,2,2)$ e a treliça puncionada mostrada na Figura 2.4(c). De acordo com (2.3), a complexidade de treliça convencional de um código $C(3,2,2)$ é $(3/2)2^4 = 24$ símbolos por bit. Mas se usarmos a treliça puncionada com $k = 2$ cópias do código-mãe $C(2,1,2)$, obtemos, a partir de (2.4) a complexidade de $(3/2)2^3 = 12$ símbolos por bit.



(a) $C(2,1,2)$ (b) $C'(4,2,2)$ (c) $C''(3,2,2)$

Figura 2.4 – (a) Módulo de treliça para o código $C(2,1,2)$ com matriz geradora em (2.5); (b) Módulo de treliça para o código $C'(4,2,2)$ formado por duas seções idênticas à da Figura 2.4(a); (c) Módulo de treliça para o código $C''(3,2,2)$ formado pelo puncionamento do quarto bit do módulo de treliça da Figura 2.4(b).

Entretanto, é possível obter para um dado código convolucional um módulo de treliça que minimize a complexidade de treliça. A esse módulo chamamos de módulo mínimo de treliça, o qual será abordado na próxima seção.

2.2.3 Módulo Mínimo

A teoria sobre treliça mínima para códigos convolucionais foi desenvolvida por Sidorenko e Zyablov [27] e por McEliece e Lin [25]. O módulo mínimo de treliça apresenta uma estrutura topológica irregular. Para esta estrutura mínima vamos assumir a complexidade

de estados $\tilde{v}_t$ e a complexidade de ramos $\tilde{b}_t$. Um módulo de treliça mínimo de um código convolucional $C(n,k,\nu)$ consiste de $n' = n$ seções (profundidades de 0 a n), das quais k possuem $\tilde{b}_t = 1$ e as $(n-k)$ sessões restantes não apresentam informação, ou seja, um único ramo diverge de cada estado $(\tilde{b}_t = 0)$. Há $2^{\tilde{v}_t}$ estados na profundidade t e $l_t = 1$ bit por ramo, $\forall t$ [25]. Definem-se os perfis de complexidade de estados e de ramos da treliça mínima por $\tilde{\boldsymbol{v}} = (\tilde{v}_0, ..., \tilde{v}_{n-1})$ e $\tilde{\boldsymbol{b}} = (\tilde{b}_0, ..., \tilde{b}_{n-1})$, respectivamente.

A Figura 2.5 mostra o módulo de treliça mínimo do código $C(5,2,2)$ do Exemplo 2.5. Este módulo possui $n = 5$ seções, com 4 ou 8 estados cada. As seções 1 e 3 (profundidades 0-1 e 2-3) apresentam $\tilde{b}_t = 1$ e as demais, $\tilde{b}_t = 0$. Todos os ramos são rotulados com um único bit. As complexidades de estados e de ramos são, respectivamente, $\tilde{\boldsymbol{v}} = (2,3,2,3,3)$ e $\tilde{\boldsymbol{b}} = (1,0,1,0,0)$. Na Figura 2.5, as linhas cheias representam bit codificado 0, e as tracejadas, bit codificado 1. Para as seções em que $\tilde{b}_t = 1$, a transição de baixo corresponde ao bit de informação 1, e a de cima, ao bit de informação 0.

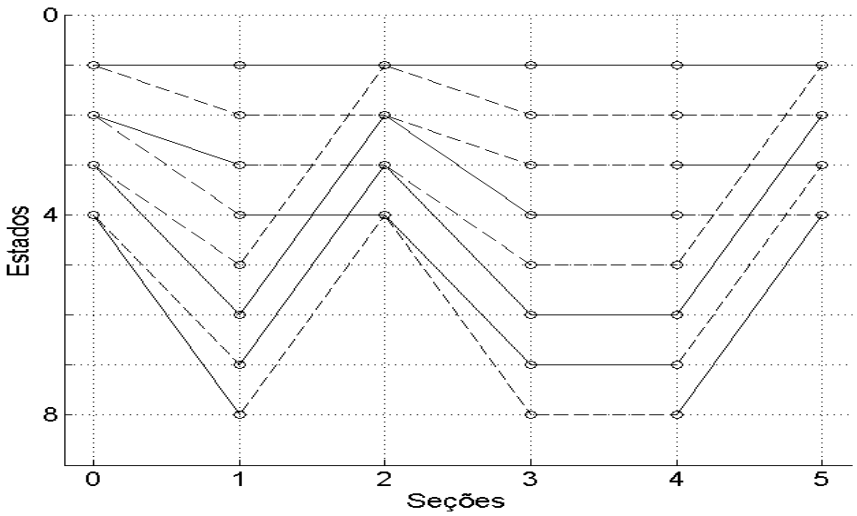


Figura 2.5 – Módulo de treliça mínimo do código $C(5,2,2)$ do Exemplo 2.5.

Para o caso particular de um módulo de treliça mínimo, $M_{\min}$, podemos reescrever (2.2) como

$$TC(M_{\min}) = \frac{1}{k}\sum_{t=0}^{n-1} 2^{\tilde{v}_t + \tilde{b}_t}. \tag{2.6}$$

Portanto, o módulo de treliça mínimo da Figura 2.5 possui $TC(M_{\min}) = 20$ símbolos por bit, a metade da complexidade da treliça convencional. O teorema a seguir relaciona as medidas de complexidade minimizadas pelo módulo de treliça mínimo [29, Teorema 4.26].

Teorema 2.3: [29, Teorema 4.26] Seja M_{min} um módulo de treliça mínimo com perfis $\tilde{\boldsymbol{v}}$ e $\tilde{\boldsymbol{b}}$ para um código C, e M um outro módulo de treliça com perfis $\boldsymbol{v}$ e $\boldsymbol{b}$ para C, ambos com um bit por ramo, então:

$$
\begin{aligned}
2^{\tilde{v}_i} &\le 2^{v_i} \qquad i = 0,\ldots,n \\
2^{\tilde{v}_i+\tilde{b}_i} &\le 2^{v_i+b_i} \quad i = 1,\ldots,n \\
\sum_{i=0}^{n-1} 2^{\tilde{v}_i} &\le \sum_{i=0}^{n-1} 2^{v_i} \\
\sum_{i=0}^{n-1} 2^{\tilde{v}_i+\tilde{b}_i} &\le \sum_{i=0}^{n-1} 2^{v_i+b_i} \\
\max_i 2^{\tilde{v}_i} &\le \max_i 2^{v_i} \\
\max_i 2^{\tilde{v}_i+\tilde{b}_i} &\le \max_i 2^{v_i+b_i}.
\end{aligned}
$$

A obtenção do módulo de treliça mínimo a partir de $G(D)$ requer que esta matriz faça parte de uma classe de matrizes geradoras chamada de matrizes geradoras de cobertura mínima (MGCM). Portanto, torna-se essencial para a busca de códigos de baixa complexidade de decodificação que a matriz geradora associada esteja na forma MGCM.

Em [28], foi definida uma técnica eficiente para se determinar a MGCM de um código de bloco e a construção do módulo de treliça mínimo associado. Em [25], este método foi adaptado para códigos convolucionais. Na próxima seção, utilizaremos a técnica proposta em [25] [28] para obter a MGCM a partir de uma $G(D)$ qualquer.

2.3 Matriz Geradora de Cobertura Mínima

Iniciaremos com algumas definições. Seja $\boldsymbol{x} = (x_1, x_2, x_3, \ldots)$ uma sequência não-nula. O índice-esquerda de $\boldsymbol{x}$, denotado por $L(\boldsymbol{x})$, é o menor índice j tal que $x_j \neq 0$. De forma análoga, o índice-direita de $\boldsymbol{x}$ (se existir), denotado por $R(\boldsymbol{x})$, é o maior índice j tal que $x_j \neq 0$. A cobertura de $\boldsymbol{x}$, denotada por $Span(\boldsymbol{x})$, é o intervalo discreto $[L(\boldsymbol{x}), R(\boldsymbol{x})]$. O comprimento de cobertura de $\boldsymbol{x}$ é definido por $S(\boldsymbol{x}) = R(\boldsymbol{x}) - L(\boldsymbol{x})$. Por simplicidade,

consideramos nesta seção apenas matrizes geradoras finitas (matrizes geradoras de códigos de bloco), embora códigos convolucionais apresentem dimensão não-finita.

Se G é uma matriz geradora binária $k\,x\,n$ com linhas $\boldsymbol{x}_1,...,\boldsymbol{x}_k$, denotada por $G=(\boldsymbol{x}_1,...,\boldsymbol{x}_k)$, o conjunto de cobertura de G é o conjunto de cobertura das linhas de G, ou seja

$$\{[L(\boldsymbol{x}_1),R(\boldsymbol{x}_1)],...,[L(\boldsymbol{x}_k),R(\boldsymbol{x}_k)]\}.$$

O comprimento de cobertura de G, denotado por $S(G)$, é a soma dos comprimentos de cobertura de suas linhas.

Exemplo 2.7: Considere a matriz geradora $G_1=(\boldsymbol{x}_1,\boldsymbol{x}_2,\boldsymbol{x}_3)$ dada por

$$G_1=\begin{pmatrix}\mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{1} & 0\\ \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{1} & 0 & 0 & 0\\ 0 & 0 & \mathbf{1} & \mathbf{0} & \mathbf{1} & 0 & 0\end{pmatrix}. \tag{2.7}$$

Então, $L(\boldsymbol{x}_1)=1,\ R(\boldsymbol{x}_1)=6$, e $S(\boldsymbol{x}_1)=5$; $L(\boldsymbol{x}_2)=1,\ R(\boldsymbol{x}_2)=4$, e $S(\boldsymbol{x}_2)=3$; $L(\boldsymbol{x}_3)=3,\ R(\boldsymbol{x}_3)=5$ e $S(\boldsymbol{x}_3)=2$. O conjunto de cobertura de G_1 é $\{[1,6],[1,4],[3,5]\}$ e $S(G_1)=10$. Os elementos compreendidos entre $L(\boldsymbol{x}_i)$ e $R(\boldsymbol{x}_i),\ i=1,...,3,$ estão mostrados em negrito.

Em [28], é definida a propriedade esquerda-direita (LR, do inglês *left-right*) de um conjunto de sequências binárias. Se $L(\boldsymbol{x}_i)\neq L(\boldsymbol{x}_j)$ e $R(\boldsymbol{x}_i)\neq R(\boldsymbol{x}_j)\ \ \forall i\neq j$, então as sequências binárias possuem a propriedade LR. Daqui em diante, usaremos o termo "forma LR" para nos referirmos à propriedade LR.

Exemplo 2.8: As linhas da matriz $G_1=(\boldsymbol{x}_1,\boldsymbol{x}_2,\boldsymbol{x}_3)$ em (2.7) não estão na forma LR já que $L(\boldsymbol{x}_1)=L(\boldsymbol{x}_2)=1$. Entretanto, as linhas da matriz $G_2=(\boldsymbol{x}_1+\boldsymbol{x}_2,\boldsymbol{x}_2,\boldsymbol{x}_3)$ estão na forma LR, como pode ser observado em

$$G_2=\begin{pmatrix}0 & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{1} & \mathbf{1} & 0\\ \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{1} & 0 & 0 & 0\\ 0 & 0 & \mathbf{1} & \mathbf{0} & \mathbf{1} & 0 & 0\end{pmatrix}. \tag{2.8}$$

O conjunto de cobertura de G_2 é $\{[2,6],[1,4],[3,5]\}$ e $S(G_2)=9$.

Pode-se mostrar que, para duas sequências $\boldsymbol{x}$ e $\boldsymbol{y}$ quaisquer, $Span(\boldsymbol{x}+\boldsymbol{y}) \subseteq Span(\boldsymbol{x}) \bigcup Span(\boldsymbol{y})$, com a igualdade verificada se, e somente se, $L(\boldsymbol{x}) \neq L(\boldsymbol{y})$ e $R(\boldsymbol{x}) \neq R(\boldsymbol{y})$ [28]. De maneira similar, $S(\boldsymbol{x}+\boldsymbol{y}) \leq S(\boldsymbol{x}) + S(\boldsymbol{y})$, com a igualdade verificada se, e somente se, $L(\boldsymbol{x}) \neq L(\boldsymbol{y})$ e $R(\boldsymbol{x}) \neq R(\boldsymbol{y})$ [29].

Definição 2.9 [28]*:* Seja C um código linear. Dentre todas as matrizes geradoras para C, aquelas cujo comprimento de cobertura é o menor possível são chamadas de matrizes geradoras de cobertura mínima (MGCM*)*.

Teorema 2.4 [40]: Uma matriz geradora binária *G* é MGCM se, e somente se, esta estiver na forma LR. Quaisquer duas matrizes MGCMs possuem o mesmo conjunto de cobertura para o mesmo código.

Um algoritmo prático, definido em [28], que produz uma MGCM utilizando uma sequência de operações com as linhas de uma matriz geradora é descrito através do seguinte pseudocódigo:

```
/* Algoritmo para encontrar a matriz geradora de cobertura mínima (MGCM) */

enquanto (forma LR não satisfaz) faça
     encontrar um par (i, j) tal que (L(x_i) = L(x_j) e R(x_i) ≤ R(x_j)) ou
     (R(x_i) = R(x_j) e L(x_i) ≥ L(x_j));
     x_j = x_j + x_i ;
fim_enquanto
```

Este algoritmo é baseado no fato de que se a forma LR não for satisfeita, por exemplo, se $L(\boldsymbol{x}_1) = L(\boldsymbol{x}_2)$ e $R(\boldsymbol{x}_1) \geq R(\boldsymbol{x}_2)$, então, substituindo-se $\boldsymbol{x}_1$ (maior comprimento de cobertura) por $\boldsymbol{x}_1 + \boldsymbol{x}_2$, o comprimento de cobertura $S(G)$ será reduzido. Caso contrário, se a forma LR for satisfeita, a matriz *G* está automaticamente em sua forma MGCM (Teorema 2.4).

Definição 2.10 [28]*:* Uma sequência não-nula $\boldsymbol{x}$ é dita ativa na coordenada j se $j \in Span(\boldsymbol{x})$, ou seja, se $L(\boldsymbol{x}) \leq j$ e $R(\boldsymbol{x}) \geq j$. Similarmente, $\boldsymbol{x}$ é dita ativa na profundidade j se ambas as coordenadas j e $j+1$ estão no intervalo $[L(\boldsymbol{x}), R(\boldsymbol{x})]$, ou seja, se $L(\boldsymbol{x}) \leq j$ e $R(\boldsymbol{x}) \geq j+1$.

Nesta seção, verificamos que, se uma matriz geradora binária *G* qualquer estiver na forma LR, então esta possui comprimento de cobertura mínimo. Também mostramos um

algoritmo eficiente para encontrar a forma LR de uma matriz geradora qualquer. Na próxima seção, veremos como obter os perfis de estados e de ramos da treliça mínima a partir da matriz geradora na forma LR.

2.4 Perfis de Estados e de Ramos

Nesta seção, apresentamos o método proposto em [25] para obter os perfis de estados e de ramos da treliça mínima de um código convolucional C a partir da matriz geradora de cobertura mínima (MGCM) de C. Sendo $G(D)$ a matriz geradora de um código convolucional $C(n,k,\nu)$, pode-se escrever $G(D)$ na forma

$$G(D) = G_0 + G_1 D + ... + G_m D^m \tag{2.9}$$

em que $G_0, G_1, ..., G_m$ são submatrizes geradoras k x n e m é a memória do codificador. À matriz k x $(m+1)n$ obtida pela concatenação das $m+1$ matrizes $G_0, G_1, ..., G_m$, chamamos de $\tilde{G}$, isto é,

$$\tilde{G} = (G_0 \ G_1 \ ... \ G_m). \tag{2.10}$$

A matriz geradora semi-infinita $G_{escalar}$ foi mostrada em (2.1). Se $G_{escalar}$ estiver na forma MGCM, podemos extrair desta a treliça mínima para C. A matriz $(m+1)k$ x n, denotada por $\hat{G}$, é definida em [21] por

$$\hat{G} = \begin{pmatrix} G_m \\ G_{m-1} \\ \vdots \\ G_0 \end{pmatrix} \tag{2.11}$$

que é uma fatia vertical em $G_{escalar}$. Se $\hat{G}$ estiver na forma MGCM, consequentemente, $G_{escalar}$ também satisfará esta condição.

Definição 2.11: Sejam $\boldsymbol{A} = (A_1, A_2, ..., A_n)$ e $\boldsymbol{B} = (B_1, B_2, ..., B_n)$ os vetores cujos elementos A_i e B_i representam o número de elementos ativos na coluna j, e, simultaneamente nas colunas $j-1$ e j, $j = 1, ..., n$ de $\hat{G}$, respectivamente.

Se $\hat{G}$ estiver na forma LR, o perfil de estados da treliça mínima é definido pelo vetor $\tilde{\boldsymbol{v}} = (\tilde{v}_0, \tilde{v}_1, ..., \tilde{v}_{n-1})$ em que $\tilde{v}_t = B_{t+1}$, $t = 0, ..., n-1$, e significa que a treliça mínima possui $2^{\tilde{v}_t}$ estados na profundidade t. O perfil de ramos da treliça mínima é definido pelo vetor $\tilde{\boldsymbol{b}} = (\tilde{b}_0, \tilde{b}_1, ..., \tilde{b}_{n-1})$ em que $\tilde{b}_t = A_{t+1} - B_{t+1}$, $t = 0, ..., n-1$, e significa que $2^{\tilde{b}_t}$ ramos divergem de cada estado da profundidade t. O número total de ramos da profundidade t é $2^{\tilde{v}_t + \tilde{b}_t}$, $t = 0, ..., n-1$. A relação entre o perfil de estados da treliça mínima e o comprimento de cobertura de $\hat{G}$ na forma LR é dada por [28]

$$\sum_{t=0}^{n-1} \tilde{v}_t = S(\hat{G}).$$

Exemplo 2.9: Considere o código convolucional C de taxa $R = 2/3$ com matriz geradora

$$G(D) = \begin{pmatrix} 1+D & 1+D & 1 \\ D & 0 & 1+D \end{pmatrix}. \tag{2.12}$$

O grau máximo de $G(D)$ é igual a 1 $(m = 1)$, portanto podemos escrevê-la da forma $G(D) = G_0 + G_1 D$ com as matrizes G_0 e G_1 dadas por:

$$G_0 = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} \quad \text{e} \quad G_1 = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}.$$

As matrizes $\tilde{G}$ e $\hat{G}$ são dadas por:

$$\tilde{G} = \begin{pmatrix} \underline{\mathbf{1}} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \bar{\mathbf{1}} & 0 \\ 0 & 0 & \underline{\mathbf{1}} & \mathbf{1} & \mathbf{0} & \bar{\mathbf{1}} \end{pmatrix} \qquad \hat{G} = \begin{pmatrix} \mathbf{1} & \bar{\mathbf{1}} & 0 \\ \mathbf{1} & \mathbf{0} & \bar{\mathbf{1}} \\ \underline{\mathbf{1}} & \mathbf{1} & \mathbf{1} \\ 0 & 0 & \underline{\mathbf{1}} \end{pmatrix}.$$

O primeiro elemento (índice-esquerda) e o último elemento (índice-direita) do intervalo de cobertura de cada linha de $\tilde{G}$ é mostrado com $\underline{1}$ e $\bar{1}$, respectivamente. Podemos observar que $\hat{G}$ está na forma LR e $S(\tilde{G}) = 7$. Observe ainda que as colunas de $\hat{G}$ são indexadas entre 1 e 3, enquanto que o primeiro elemento do vetor $\boldsymbol{B}$ necessita dos elementos ativos na coluna "0". Os elementos ativos nesta coluna podem ser interpretados como os elementos ativos da última seção do módulo anterior e correspondem aos elementos ativos na coluna 1 de $\hat{G}$ que não sejam um índice-esquerda. Portanto, os vetores de elementos ativos (destacados em negrito) de $\hat{G}$ são $\boldsymbol{A} = (3,3,3)$ e $\boldsymbol{B} = (2,3,2)$, e os perfis de estados

e de ramos da treliça mínima são respectivamente, $\tilde{\boldsymbol{v}} = (2,3,2)$ e $\tilde{\boldsymbol{b}} = (1,0,1)$. A Tabela 2.1 mostra o número de estados e de ramos em cada profundidade da treliça mínima associada ao código C e a Figura 2.6 mostra a treliça mínima.

Tabela 2.1 – Número de estados e de ramos em cada profundidade da treliça mínima de C.

Profundidade (t)	Número de estado ($2^{\tilde{v}_t}$)	Número de ramos ($2^{\tilde{v}_t+\tilde{b}_t}$)
0	4	8
1	8	8
2	4	8

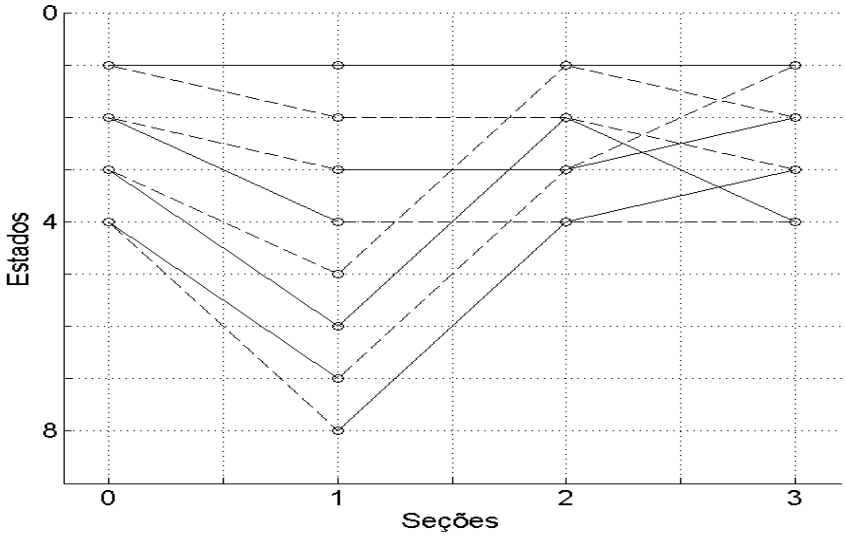


Figura 2.6 - Treliça mínima do código C, de taxa $R = 2/3$ e $G(D)$ dada em (2.12).

No Exemplo 2.9, a matriz geradora $G(D)$ está na forma LR, logo nenhuma operação com as linhas de $\tilde{G}$ foi necessária. Essa situação não ocorre no Exemplo 2.10 a seguir.

Exemplo 2.10: Considere o código C de taxa $R = 2/3$ e $m = 1$ com matriz geradora dada por

$$G(D) = \begin{pmatrix} 1 & 0 & 1 \\ 1 & 1+D & 1+D \end{pmatrix}.$$

A matriz $\tilde{G}$ extraída diretamente de $G(D)$ é dada por

$$\tilde{G} = \begin{pmatrix} \underline{\mathbf{1}} & \mathbf{0} & \bar{\mathbf{1}} & 0 & 0 & 0 \\ \underline{\mathbf{1}} & \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{1} & \bar{\mathbf{1}} \end{pmatrix}.$$

Percebe-se que $\tilde{G}$ e, consequentemente, $\hat{G}$ não estão na forma LR, pois $L(\boldsymbol{x}_1) = L(\boldsymbol{x}_2)$ e $S(\tilde{G}) = 7$. Vamos utilizar o algoritmo proposto em [28] para encontrar as matrizes $\tilde{G}'(\boldsymbol{x}_1', \boldsymbol{x}_2') = \tilde{G}(\boldsymbol{x}_1, \boldsymbol{x}_1 + \boldsymbol{x}_2)$ e $\hat{G}'$ a seguir

$$\tilde{G}' = \begin{pmatrix} \underline{\mathbf{1}} & \mathbf{0} & \bar{\mathbf{1}} & 0 & 0 & 0 \\ 0 & \underline{\mathbf{1}} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \bar{\mathbf{1}} \end{pmatrix} \qquad \text{e} \qquad \hat{G}' = \begin{pmatrix} 0 & 0 & 0 \\ \mathbf{0} & \mathbf{1} & \bar{\mathbf{1}} \\ \underline{\mathbf{1}} & \mathbf{0} & \bar{\mathbf{1}} \\ 0 & \underline{\mathbf{1}} & \mathbf{0} \end{pmatrix}.$$

Percebe-se que $S(\tilde{G}') = 6$, mas $\hat{G}'$ ainda não está na forma LR (a terceira coluna possui dois índices-direita). Novamente vamos utilizar o algoritmo proposto em [28] e encontrar as matrizes $\tilde{G}''(\boldsymbol{x}_1'', \boldsymbol{x}_2'')$ e $\hat{G}''$ a seguir

$$\tilde{G}'' = \begin{pmatrix} \underline{\mathbf{1}} & \mathbf{0} & \bar{\mathbf{1}} & 0 & 0 & 0 \\ 0 & \underline{\mathbf{1}} & \mathbf{0} & \mathbf{1} & \bar{\mathbf{1}} & 0 \end{pmatrix} \qquad \text{e} \qquad \hat{G}'' = \begin{pmatrix} 0 & 0 & 0 \\ \mathbf{1} & \bar{\mathbf{1}} & 0 \\ \underline{\mathbf{1}} & \mathbf{0} & \bar{\mathbf{1}} \\ 0 & \underline{\mathbf{1}} & \mathbf{0} \end{pmatrix}.$$

Ambas as matrizes $\tilde{G}''$ e $\hat{G}''$ estão na forma LR e $S(\tilde{G}'') = 5$. Os vetores de elementos ativos (destacados em negrito) de $\hat{G}''$ são $\boldsymbol{A} = (2,3,2)$ e $\boldsymbol{B} = (1,2,2)$. Portanto, os perfis de estados e de ramos da treliça mínima são respectivamente, $\tilde{\boldsymbol{v}} = (1,2,2)$ e $\tilde{\boldsymbol{b}} = (1,1,0)$.

Nesta seção, encontramos o perfil de estados e de ramos da treliça mínima, sendo este o primeiro passo para a sua construção. As conexões entre os estados e a determinação do rótulo de cada ramo são descritas no Apêndice A, que utiliza o método proposto em [25] para a construção da treliça mínima.

CAPÍTULO 3

COMPLEXIDADE COMPUTACIONAL DO ALGORITMO DE VITERBI

O Algoritmo de Viterbi (VA) utiliza a treliça de um código convolucional para estimar a palavra-código transmitida, dada uma seqüência de símbolos recebida através de um canal de comunicação ruidoso. As operações envolvidas nesse processo dependem da topologia da treliça e podem ser usadas para determinar a complexidade computacional (TCC) do VA para um código específico. A complexidade computacional é a medida apropriada da complexidade de decodificação, caso o decodificador seja implementado em software [23]. No caso de decisão abrupta, as operações envolvidas são apenas soma e comparação.

Neste capítulo, analisamos as operações requeridas pelo VA e seus respectivos custos computacionais em termos de ciclos de máquina, utilizando como base a família de Processadores Digitais de Sinais de ponto fixo TMS320C55xx da Texas Instruments. Uma vez determinado o custo de cada operação do VA, definiremos uma medida para a TCC de um módulo de treliça.

Finalmente, uma busca de códigos será conduzida para as taxas 2/4, 3/5, 4/7 e 5/7. O objetivo destas buscas é listar códigos com melhor espectro de distâncias para vários valores da TCC definida neste trabalho.

3.1 Operações realizadas pelo Algoritmo de Viterbi

A cada seção de um módulo de treliça M, o VA operando com decisão abrupta realiza as quatro etapas listadas a seguir:

1. Uma palavra binária de comprimento l_t é recebida.
2. A distância de *Hamming* entre a palavra recebida e a palavra-código em cada ramo da treliça é calculada. O resultado é conhecido como incremento de métrica.
3. É executada a soma de cada incremento de métrica com a métrica atual de seu respectivo ramo.
4. A menor métrica acumulada (incremento de métrica + métrica atual do ramo) entre os ramos que convergem para um estado na profundidade *t+1* é considerada a vencedora. O rótulo do ramo da métrica vencedora é armazenado na memória (chamada de *Traceback RAM*) como caminho sobrevivente deste estado e a métrica é atualizada. Repete-se esse procedimento para cada estado da profundidade *t+1*.

Após um quadro ter sido decodificado, a operação de leitura do caminho sobrevivente na memória determina a sequência de informação decodificada. Essa operação é chamada de *Traceback*.

Para realizar a decodificação usando o VA, são necessários três componentes: o calculador de distância de *Hamming* (HDC, do inglês *Hamming distance calculator*), o somar-comparar-armazenar (ACS, do inglês *add-compare-store*) e o *RAM Traceback.* A Figura 3.1 ilustra o diagrama em blocos do VA.

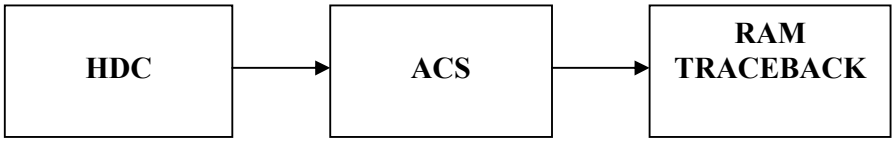


Figura 3.1 – Diagrama em blocos do VA.

Nas próximas subseções, serão analisadas as operações requeridas pelo HDC e ACS sobre um módulo de treliça para o VA operando com decisão abrupta. Como o *RAM Traceback* não envolve operações de soma nem comparações, este não será considerado nesse trabalho.

3.1.1 Operações do HDC

A etapa de HDC consiste em calcular a distância de *Hamming* entre a palavra recebida e a palavra-código em cada ramo de uma seção t do módulo de treliça M. Como cada ramo é rotulado com l_t bits, são necessárias l_t operações de comparação de bit cujos resultados são somados com $l_t - 1$ operações de soma. O resultado final é a distância de *Hamming* desejada. O número total de ramos na seção t é dado por $2^{v_t+b_t}$, portanto são necessárias $l_t 2^{v_t+b_t}$ operações de comparações de bit e $(l_t - 1)2^{v_t+b_t}$ operações de soma. O cálculo detalhado do HDC para uma seção t de um módulo de treliça M é mostrado na Tabela 3.1. Definindo o número de operações de soma por S e o número de comparações de bit por C_b, concluímos da Tabela 3.1 que o número total de operações por seção do HDC, T_t^{HDC}, é dado por:

$$T_t^{HDC} = (l_t - 1)2^{v_t+b_t}(S) + l_t 2^{v_t+b_t}(C_b). \tag{3.1}$$

Tabela 3.1 – Operações do HDC sobre uma seção t do módulo de treliça M.

Operações por ramo:	l_t comparações de bit $l_t - 1$ somas
Número de ramos:	$2^{v_t+b_t}$
Total de operações do HDC (T_t^{HDC}):	$l_t 2^{v_t+b_t}$ comparações de bit $(l_t - 1)2^{v_t+b_t}$ somas

3.1.2 Operações do ACS

A etapa de ACS consiste em atualizar a métrica dos estados da treliça. Primeiramente, o incremento de métrica de cada ramo é somado com a métrica de seu estado inicial. O resultado é a métrica acumulada do ramo. Essa operação é repetida para cada um dos $2^{v_t+b_t}$ ramos da seção t, portanto são necessárias $2^{v_t+b_t}$ operações de soma. O próximo passo dessa etapa consiste em comparar as métricas acumuladas dos ramos que convergem para cada um dos estados da seção $t+1$ e selecionar a menor. O resultado é a métrica vencedora e esta é armazenada na memória. Há $2^{v_{t+1}}$ estados na seção $t+1$ e $2^{v_t+b_t}$ ramos entre as seções t e $t+1$, portanto são comparados $2^{v_t+b_t} / 2^{v_{t+1}}$ ramos por estado, que utilizam $(2^{v_t+b_t} / 2^{v_{t+1}}) - 1$ operações de comparação. Considerando todos os estados,

teremos $[(2^{v_t+b_t}/2^{v_{t+1}})-1]2^{v_{t+1}}$,ou ainda $2^{v_t+b_t}-2^{v_{t+1}}$ operações de comparação no total. Essas operações de comparação são realizadas sobre valores inteiros. O cálculo detalhado do ACS para uma seção t de um módulo de treliça M é mostrado na Tabela 3.2. Definindo o número de operações de soma por S e o número de comparações de valores inteiros por C_i, concluímos da Tabela 3.2 que o número total de operações por seção do ACS, T_t^{ACS}, é dado por:

$$T_t^{ACS} = 2^{v_t+b_t}(S) + [2^{v_t+b_t} - 2^{v_{t+1}}](C_i). \tag{3.2}$$

Tabela 3.2 – Operações do ACS sobre uma seção t do módulo de treliça M .

Operação por ramo:	1 soma
Número de ramos:	$2^{v_t+b_t}$
Operação por estado convergente:	$(2^{v_t+b_t}/2^{v_{t+1}})-1$ comparações de valores inteiros
Número de estados convergentes:	$2^{v_{t+1}}$
Total de operações do ACS (T_t^{ACS}):	$2^{v_t+b_t}$ somas $2^{v_t+b_t}-2^{v_{t+1}}$ comparações de valores inteiros

3.2 Algoritmo de Viterbi sobre as Treliças Convencional e Mínima

Nas subseções anteriores, determinamos que as operações envolvidas nas etapas de HDC e ACS de um módulo de treliça são: operações de soma (S), comparações de bits (C_b) e comparações de valores inteiros (C_i). A partir de (3.1) e (3.2) definimos o número total de operações do VA por bit de informação de um módulo de treliça M conforme a seguir

$$\begin{aligned} T(M) &= \frac{1}{k}\sum_{t=0}^{n'-1}(T_t^{HDC} + T_t^{ACS}) \\ T(M) &= \frac{1}{k}\sum_{t=0}^{n'-1} l_t 2^{v_t+b_t}[C_b + S] + (2^{v_t+b_t} - 2^{v_{t+1}})(C_i). \end{aligned} \tag{3.3}$$

Considerando que a complexidade de treliça por bit de informação $TC(M)$ de um módulo de treliça M para um código convolucional C é definida por [25]

$$TC(M) = \frac{1}{k}\sum_{t=0}^{n'-1} l_t 2^{v_t+b_t} \tag{3.4}$$

e definindo a complexidade comparativa por bit de informação $MC(M)$ de um módulo de treliça M para um código convolucional C por

$$MC(M) = \frac{1}{k}\sum_{t=0}^{n'-1}(2^{v_t+b_t} - 2^{v_{t+1}}) \tag{3.5}$$

em que $v_{n'} = v_0$, reescrevemos (3.3) usando (3.4) e (3.5) da seguinte forma

$$T(M) = TC(M)(C_b + S) + MC(M)C_i. \tag{3.6}$$

A seguir, reescreveremos (3.4) e (3.5) para três casos particulares:

- Para o módulo de treliça convencional, M_{conv}, temos $l_t = n$, $n' = 1$, $v_0 = v_1 = v$ e $b_0 = k$. Assim, obtemos

$$TC(M_{conv}) = \frac{n}{k}2^{k+v} \tag{3.7}$$

e

$$MC(M_{conv}) = \frac{2^v(2^k - 1)}{k}. \tag{3.8}$$

- Para o módulo de treliça puncionado, M_{PCC}, temos $n' = k$, $v_t = v \ \ \forall t$, $b_t = 1$ $\forall t$ e $\sum l_t = n$. Assim, obtemos

$$TC(M_{PCC}) = \frac{n}{k}\cdot 2^{v+1} \tag{3.9}$$

e

$$MC(M_{PCC}) = \frac{2^v}{k}. \tag{3.10}$$

- Para o módulo de treliça mínimo, $M_{\min}$, temos $n' = n$, $l_t = 1$, $v_t = \tilde{v}_t \ \forall t$, $b_t = \tilde{b}_t \ \forall t$ e $v_n = v_0$. Desta forma,

$$TC(M_{\min}) = \frac{1}{k}\sum_{t=0}^{n'-1}2^{\tilde{v}_t+\tilde{b}_t} \tag{3.11}$$

e

$$MC(M_{\min}) = \frac{1}{k}\sum_{t=0}^{n'-1}(2^{\tilde{v}_t+\tilde{b}_t} - 2^{\tilde{v}_{t+1}}). \tag{3.12}$$

Observe que a expressão (3.12) pode ainda ser reescrita na forma

$$MC(M_{\min}) = TC(M_{\min}) - S(M_{\min}) \tag{3.13}$$

em que

$$S(M_{\min}) = \frac{1}{k}\sum_{t=0}^{n'-1} 2^{v_t} \tag{3.14}$$

é o número total de estados por bit de informação da treliça mínima. Uma fórmula alternativa para $MC(M_{\min})$ é desenvolvida a seguir

$$\begin{aligned} MC(M_{\min}) &= \frac{1}{k}\sum_{t=0}^{n'-1} (2^{\tilde{v}_t+\tilde{b}_t} - 2^{\tilde{v}_t}) \\ &= \frac{1}{k}\sum_{t=0}^{n'-1} 2^{\tilde{v}_t}(2^{\tilde{b}_t} - 1) \\ &= \frac{1}{k}\sum_{\substack{t=0 \\ \tilde{b}_t=1}}^{n'-1} 2^{\tilde{v}_t} \end{aligned} \tag{3.15}$$

de onde conclui-se que a complexidade comparativa $MC(M_{\min})$ depende do número total de estados apenas nas seções da treliça mínima em que $\tilde{b}_t = 1$.

Exemplo 3.1: Considere o código convolucional $C_1(7,3,3)$ com matriz geradora $G_1(D)$ dada por:

$$G_1(D) = \begin{pmatrix} 1+D & 1+D & 1 & 1 & 0 & 1 & 1 \\ D & 0 & 1+D & 1+D & 1 & 1 & 0 \\ D & D & 0 & D & 1+D & 1+D & 1+D \end{pmatrix}. \tag{3.16}$$

As complexidades de treliça e comparativa do módulo de treliça convencional para este código são $TC(M_{conv}) = 149,33$ e $MC(M_{conv}) = 18,66$. Portanto, obtemos de (3.6):

$$T(M_{conv}) = 149,33(S + C_b) + 18,66(C_i).$$

O módulo da treliça mínimo para o mesmo código é mostrado na Figura 3.2. As complexidades de estado e de ramos são, respectivamente, $\tilde{\boldsymbol{v}} = (3,4,3,4,3,4,4)$ e $\tilde{\boldsymbol{b}} = (1,0,1,0,1,0,0)$, o que implica em $TC(M_{\min}) = 37,33$ e $MC(M_{\min}) = 8$. Analogamente, obtemos de (3.6):

$$T(M_{\min}) = 37,33(S + C_b) + 8(C_i).$$

No exemplo analisado, o número relativo de operações requeridas pela treliça mínima em relação à treliça convencional é 25% para S e C_b e 42,8% para C_i.

Foi discutido nesta seção que a complexidade do VA operando com decisão abrupta depende de três operações básicas denotadas por S, C_b e C_i. Para que a comparação de complexidade seja mais significativa, o custo destas operações é determinado nas próximas seções, com base numa arquitetura particular.

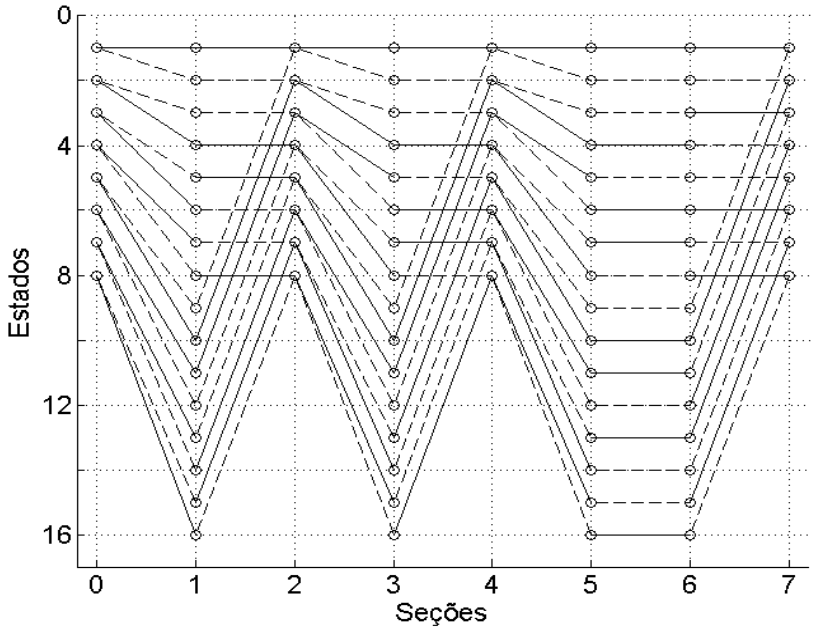


Figura 3.2 – Módulo de treliça mínimo para o código $C_1(7,3,3)$. As linhas sólidas representam bit codificado "0" e as tracejadas o bit codificado "1".

3.3 Simulação das Operações do HDC e ACS

Para determinar a complexidade computacional do VA em termos de ciclos de máquina usamos a família de Processadores Digitais de Sinais 320TMSC55xx da Texas Instruments (TI). Estes processadores são específicos para aplicações de comunicação, utilizam processamento com ponto fixo, possuem desempenho otimizado e baixo consumo de energia [41]. Mais detalhes sobre esses processadores são encontrados em [42]. O ambiente de desenvolvimento integrado (IDE) utilizado foi o Code Composer Studio (CCStudio) versão 4.1.1.00014 da TI. Esse IDE incorpora uma série de ferramentas como compiladores para cada família de dispositivos da TI, editor de programas, ambiente para montagem de projetos, depurador, simuladores e outros recursos [43]. Para simular a

operação do processador, utilizamos o simulador *C55xx Rev2.x CPU Cycle Accurate Simulator* que fornece com precisão o número de ciclos de máquina consumidos por cada instrução do programa. No restante deste trabalho, vamos usar a abreviação C55x para nos referirmos à família de DSPs 320TMSC55xx.

Essa etapa do trabalho envolveu implementar da forma mais eficiente possível cada operação definida em (3.3), ou seja, as operações S, C_b e C_i usando o C55x, e determinar seus respectivos custos computacionais em termos de ciclos de máquina. Com base nesses valores, foi possível definir uma complexidade computacional do VA aplicada a um código convolucional operando em um módulo de treliça M.

3.4 Custo Computacional do VA

O custo computacional do VA é uma função do custo computacional isolado das operações S, C_b e C_i e será determinado com base em simulações utilizando-se linguagem C. Operações com a memória, como leitura de operandos e escrita de resultados são incluídas automaticamente pelo compilador, e têm seus custos embutidos no custo geral da operação. Em todas as operações simuladas, o compilador C do C55x utilizou o modo de endereçamento direto com a pilha do programa para armazenar as variáveis. O registrador ponteiro de pilha (SP) foi usado para acessá-las. Nas próximas subseções, descreveremos como foi implementada cada uma dessas operações.

3.4.1 Implementação da Operação S

A Tabela 3.3 mostra os detalhes da implementação da operação S. Foram usadas variáveis de tipo inteiro. Na primeira coluna, aparece a operação soma de dois valores em linguagem C; na segunda, o código em linguagem *Assembly* do C55x gerado pelo compilador; na terceira, uma breve descrição do código, em que AR1 é o registrador acumulador utilizado para receber o resultado da soma; na quarta, os respectivos ciclos de máquina. No total, são necessários 3 ciclos de máquina para implementar a operação S.

Tabela 3.3 – Detalhes da implementação da operação S.

Implementação em C	Assembly C55x	Descrição	Ciclos
S = code1 + code2;	MOV *SP(#01h),AR1	AR1 ← code1	1
	ADD *SP(#00h),AR1,AR1	AR1 ← AR1 + code2	1
	MOV AR1,*SP(02h)	S ← AR1	1
		Total	3

3.4.2 Implementação da Operação C_b

A operação C_b foi implementada com instruções lógicas XOR bit a bit, assumindo que cada bit da palavra recebida e da palavra-código em cada ramo do módulo da treliça tenha sido previamente armazenado em variável de tipo inteiro.

Apesar de gerar um maior consumo de memória e, consequentemente, um maior número de operações de leitura e escrita, essa opção foi a que apresentou menor custo computacional dentre várias simuladas. Como estamos buscando uma medida da complexidade computacional baseada em operações aritméticas, fatores relacionados à eficiência do uso da memória não foram considerados.

A Tabela 3.4 mostra os detalhes da implementação da operação C_b. Na primeira coluna, aparece a instrução XOR bit a bit em linguagem C; na segunda, o código em linguagem *Assembly* do C55x gerado pelo compilador; na terceira, uma breve descrição do código, em que AR1 é o registrador acumulador utilizado para receber o resultado da operação XOR; na quarta, os respectivos ciclos de máquina. No total, são necessários 3 ciclos de máquina para implementar a operação C_b.

Tabela 3.4 – Detalhes da implementação da operação C_b.

Implementação em C	Assembly C55x	Descrição	Ciclos
Cb = code1 ^ code2;	MOV *SP(#01h),AR1	AR1 ← code1	1
	XOR *SP(#00h),AR1,AR1	AR1 ← AR1 XOR code2	1
	MOV AR1,*SP(02h)	Cb ← AR1	1
		Total	3

3.4.3 Implementação da Operação C_i

A operação C_i foi implementada com uma instrução de seleção composta *if* incluindo o armazenamento do valor da menor métrica acumulada. A Tabela 3.5 mostra os detalhes da implementação da operação C_i. Na primeira coluna, aparece a instrução *if* que compara duas métricas acumuladas, sendo a menor armazenada na variável de tipo inteiro *menor*. Na segunda coluna, o código em linguagem *Assembly* do C55x gerado pelo compilador; na terceira, uma breve descrição do código explicada pelos seguintes passos: AR1 e AR2 são registradores acumuladores que recebem os valores das métricas acumuladas, aqui representadas pelas variáveis A e B, respectivamente. A seguir, as métricas são comparadas; se B < A, então o bit de *status* TC1 é setado, o fluxo do programa é desviado para o rótulo @L1 e o valor de B é armazenado em *menor*. Seguindo esse caminho, o código consome 10 (=1+1+1+6+1) ciclos de máquina. Caso A < B o código segue armazenando o valor de A em *menor* e desviando o fluxo do programa para o rótulo @L2 onde está a próxima instrução a ser executada. Um detalhe da arquitetura é que o valor em AR2 não pode ser diretamente transferido para a memória, precisando ser copiado primeiro para AR1 e depois então para a memória. Seguindo esse caminho, o código consome 16 (=1+1+1+5+1+1+6) ciclos de máquina.

Tabela 3.5 – Detalhes da implementação da operação C_i.

Implementação em C	**Assembly C55x**	**Descrição**	**Ciclos**	
If (A < B) menor = A; else menor = B;	MOV *SP(#01h),AR1	AR1 ← B	1	1
	MOV *SP(#00h),AR2	AR2 ← A	1	1
	CMP AR2>=AR1, TC1	TC1 ← (AR2>=AR1)	1	1
	BCC @L1,TC1	se (TC1) vá para @L1	6	5
	MOV AR2,AR1	AR1 ← AR2	--	1
	MOV AR1, *SP(#02h)	menor ← AR1	--	1
	B @L2	vá para @L2	--	6
	@L1: MOV AR1, *SP(02h)	@L1: menor ← AR1	1	--
	@L2: ... (próxima instrução)	@L2: ... (próxima instrução)	--	--
		Total	10 ou 16	

É comum que instruções de seleção apresentem variação do custo computacional em função do desvio do fluxo do programa requerido para executar as instruções relativas ao bloco verdadeiro ou falso associados ao resultado da operação. Levamos em consideração o valor médio consumido pela operação, ou seja, 13 ciclos de máquina,

justificado pela aleatoriedade dos valores. Finalmente, na quarta coluna da Tabela 3.5, aparecem os ciclos de máquina por instrução. Em resumo, o custo computacional das operações do VA é mostrado na Tabela 3.6.

Tabela 3. 6 - Custo computacional das operações do VA.

Operação	Ciclos
Soma S	3
Comparação de bit C_b	3
Comparação de inteiro C_i	13

3.5 Complexidade Computacional do Algoritmo de Viterbi

Substituindo os resultados da Tabela 3.6 em (3.6), definimos uma complexidade computacional, denotada por $TCC(M)$, de um módulo de treliça M por

$$TCC(M) = TC(M)6 + MC(M)13. \tag{3.17}$$

Portanto, a complexidade computacional depende das duas medidas de complexidade do módulo, a complexidade de treliça e a comparativa, sendo que o peso da última é aproximadamente o dobro do peso da primeira.

Para o código $C_1(7,3,3)$ do Exemplo 3.1, obtemos $TCC(M_{conv}) = 1138,5$ e $TCC(M_{\min}) = 328$. Logo, a complexidade computacional da treliça mínima é 28,8% da complexidade computacional da treliça convencional, mas em termos de complexidade comparativa, esse percentual é 42,87%, enquanto que para complexidade de treliça é 25%.

Exemplo 3.2: Considere os códigos convolucionais $C_2(5,2,2)$ e $C_3(5,3,3)$ com matrizes geradoras $G_2(D)$ e $G_3(D)$ dadas por

$$G_2(D) = \begin{pmatrix} 1+D & 1+D & 1+D & 1 & 0 \\ D & 0 & 1+D & 1+D & 1+D \end{pmatrix}$$

$$G_3(D) = \begin{pmatrix} 1+D & 1 & 0 & 0 & 1 \\ 0 & 1+D & 1+D & 0 & 1 \\ D & 0 & D & 1+D & 1 \end{pmatrix}$$

e complexidades de estados e de ramos da treliça mínima dadas por $\tilde{\boldsymbol{v}} = (2,3,3,3,3)$ e $\tilde{\boldsymbol{b}} = (1,0,1,0,0),$ para C_2, e $\tilde{\boldsymbol{v}} = (3,3,4,3,3)$ e $\tilde{\boldsymbol{b}} = (1,1,0,1,0),$ para C_3. As complexidades de

treliça, comparativa e computacional do módulo de treliça mínimo para os códigos C_2 e C_3 são mostradas na Tabela 3.7.

Tabela 3.7 – Complexidades de treliça, comparativa e computacional do módulo de treliça mínimo para os códigos C_2 e C_3.

Código	$TC(M_{\min})$	$MC(M_{\min})$	$TCC(M_{\min})$
$C_2(5,2,2)$	24	6	222
$C_3(5,3,3)$	24	8	248

Observa-se que ambos os códigos possuem a mesma complexidade de treliça mínima $TC(M_{\min}) = 24$. A Tabela 3.7 mostra que C_2 apresenta 75% da complexidade comparativa e 89,5% da complexidade computacional de C_3. Logo, dois códigos de taxas diferentes que apresentam complexidades de treliça iguais podem apresentar complexidades computacionais diferentes. Este exemplo evidencia que a complexidade de treliça não deve ser adotada para comparar a complexidade computacional de códigos de taxas diferentes. Nos próximos exemplos, vamos analisar o impacto das complexidades de treliça, comparativa e computacional sobre códigos de taxas iguais.

Exemplo 3.3: Considere os códigos convolucionais $C_4(4,3,3)$ e $C_5(4,3,4)$ de taxa $R = 3/4$ e matrizes geradoras $G_4(D)$ e $G_5(D)$ dadas por

$$G_4(D) = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & D & 1+D & 1 \\ D^2 & 1+D^2 & 1+D & 0 \end{pmatrix}$$

$$G_5(D) = \begin{pmatrix} D & D & 1 & 1 \\ 1+D & 1 & 0 & 1 \\ D+D^2 & 0 & D^2 & 1+D^2 \end{pmatrix}$$

e complexidades de estados e de ramos da treliça mínima dadas por $\tilde{\boldsymbol{v}} = (1,1,1,0)$ e $\tilde{\boldsymbol{b}} = (3,4,4,4)$ para C_4 e $\tilde{\boldsymbol{v}} = (1,0,1,1)$ e $\tilde{\boldsymbol{b}} = (4,4,3,4)$ para C_5. Nesse caso, ambos os códigos apresentam $TC(M_{min}) = 32$, $MC(M_{\min}) = 13,33$ e $TCC(M_{\min}) = 365,3$, embora tenham comprimento de restrição total distintos. Os módulos de treliça mínimos de C_4 e C_5 são mostrados nas Figuras 3.3 e 3.4, respectivamente. Por outro lado, códigos de mesma taxa e mesmo comprimento total de restrição podem apresentar complexidades comparativas diferentes. O Exemplo 3.4 mostra essa situação.

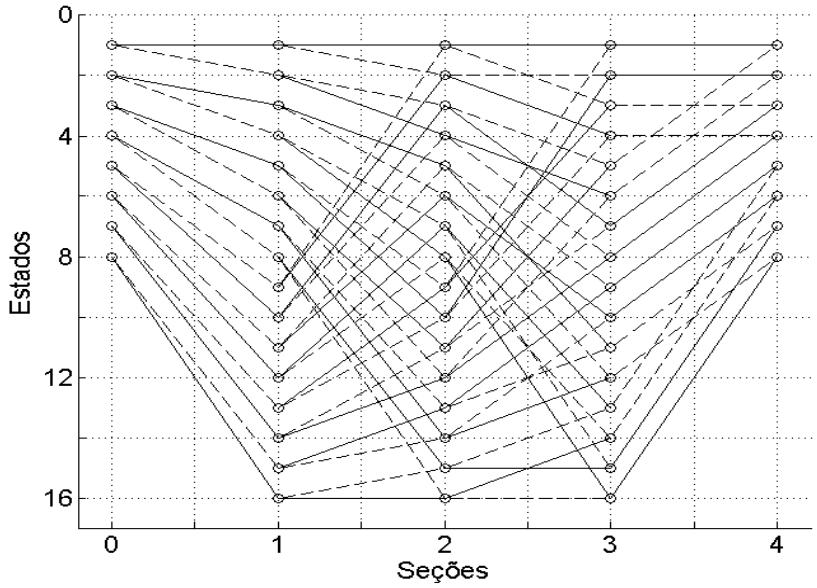


Figura 3.3 – Módulo de treliça mínimo para o código $C_4(4,3,3)$. As linhas sólidas representam o bit codificado "0" e as tracejadas o bit codificado "1".

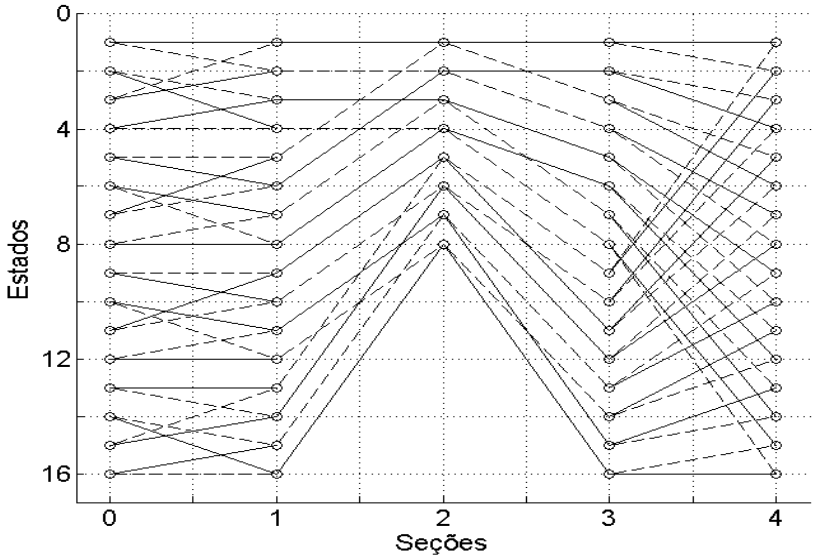


Figura 3.4 – Módulo de treliça mínimo para o código $C_5(4,3,4)$. As linhas sólidas representam o bit codificado "0" e as tracejadas o bit codificado "1".

Exemplo 3.4: Considere os códigos convolucionais $C_6(4,2,3)$ com perfis $\tilde{\boldsymbol{v}} = (3,2,3,4)$ e $\tilde{\boldsymbol{b}} = (0,1,1,0)$ e $C_7(4,2,3)$ com perfis $\tilde{\boldsymbol{v}} = (3,3,3,3)$ e $\tilde{\boldsymbol{b}} = (1,0,1,0)$, ambos de taxa $R = 2/4$ e matrizes geradoras $G_6(D)$ e $G_7(D)$ dadas por

$$G_6(D) = \begin{pmatrix} D+D^2 & 1+D & D & 0 \\ D & 0 & 1+D & 1+D \end{pmatrix}$$

$$G_7(D) = \begin{pmatrix} D^2 & D & 1+D & 1+D \\ 1+D & 1+D & D & 1 \end{pmatrix}.$$

As complexidades de treliça, comparativa e computacional do módulo de treliça mínimo para os códigos C_6 e C_7 são mostradas na Tabela 3.8. Embora ambos apresentem a mesma complexidade de treliça, isso não se reflete na complexidade comparativa gerando complexidades computacionais diferentes. Esse fato reforça a importância de se adotar a complexidade computacional para comparar também códigos de mesma taxa.

Tabela 3.8 – Complexidades de treliça, comparativa e computacional do módulo de treliça mínimo para os códigos C_6 e C_7.

Código	$TC(M_{\min})$	$MC(M_{\min})$	$TCC(M_{\min})$
$C_6(4,2,3)$	24	6	222
$C_7(4,2,3)$	24	8	248

Tabelas de códigos convolucionais para uma dada taxa e várias $TC(M_{\min})$ são encontradas em [19][21][23], porém estes trabalhos consideram apenas a complexidade de treliça em suas buscas.

Exemplo 3.5: Considere os códigos convolucionais $C_8(4,3,4)$ com perfis $\tilde{\boldsymbol{v}} = (4,4,4,4)$ e $\tilde{\boldsymbol{b}} = (0,1,1,1)$ e $C_9(4,3,4)$ com perfis $\tilde{\boldsymbol{v}} = (4,5,4,4)$ e $\tilde{\boldsymbol{b}} = (1,0,1,1)$, ambos de taxa $R = 3/4$ e mesmo comprimento de restrição v e matrizes geradoras $G_8(D)$ e $G_9(D)$ dadas por

$$G_8(D) = \begin{pmatrix} D & D & D & 1+D \\ D & 1+D & 1 & 1 \\ D & D+D^2 & 1+D+D^2 & 0 \end{pmatrix}$$

$$G_9(D) = \begin{pmatrix} 1 & 1 & 1 & 1 \\ D & D^2 & D+D^2 & 1 \\ D^2 & D+D^2 & 1+D & 0 \end{pmatrix}.$$

As complexidades de treliça, comparativa e computacional do módulo de treliça mínimo para os códigos C_8 e C_9 são mostradas na Tabela 3.9. Embora ambos apresentem a mesma

complexidade comparativa, isso não se reflete nas complexidades de treliça e computacional. Este fato evidencia que a complexidade comparativa sozinha também não é suficiente para medir o esforço computacional de decodificação. Observa-se ainda que o número de estados nas seções em que $\tilde{b}_t = 1$ é o mesmo para as duas treliças, porém o número total de estados $S(M_{\min})$ é diferente.

Tabela 3.9 – Complexidades de treliça, comparativa e computacional do módulo de treliça mínimo para os códigos C_8 e C_9.

Código	$TC(M_{\min})$	$MC(M_{\min})$	$TCC(M_{\min})$
$C_8(4,3,4)$	37,33	16	432
$C_9(4,3,4)$	42,67	16	464

A análise da complexidade computacional do VA para decisão suave é apresentada no Apêndice B deste trabalho. Esta análise mostra que, se um processo de quantização dos símbolos de entrada for considerado, a complexidade computacional da decisão suave é praticamente a mesma da decisão abrupta (para uma implementação usando o mesmo processador de ponto fixo considerado neste capítulo).

3.6 Busca de Códigos baseadas no par (TC, MC)

Buscas de bons códigos convolucionais são comuns na literatura. No entanto, geralmente apenas a $TC(M)$ é considerada como parâmetro destas buscas, como em [21], por exemplo. O objetivo das buscas de códigos deste trabalho é encontrar bons códigos (em relação ao espectro de distâncias) considerando-se tanto a $TC(M)$ quanto a $MC(M)$, o que permite a listagem de códigos com base na complexidade computacional $TCC(M)$. O procedimento definido em [21] é utilizado nestas buscas. O primeiro passo é propor *templates* a partir da matriz $G(D)$ com complexidades de treliça $TC(M_{min})$ e comparativa $MC(M_{min})$ fixas para a matriz módulo de um código. Definido o *template* e fixados os bits 1's dos índices-esquerda e índices-direita em suas posições específicas, os demais bits do intervalo de cobertura podem assumir livremente qualquer combinação de 0's e 1's. Cada uma destas combinações produz uma matriz $G(D)$ diferente.

Exemplo 3.6: Considere o código $C_1(7,3,3)$ com matriz geradora $G_1(D)$ em (3.16) com $TC(M_{\min}) = 37,33$ e $MC(M_{\min}) = 8$. O *template* associado a $G_1(D)$ é dado por

$$\begin{pmatrix} * & \bar{1} & & & & & \\ * & * & * & \bar{1} & & & \\ * & * & * & * & * & * & \bar{1} \\ \underline{1} & * & * & * & * & * & * \\ & & \underline{1} & * & * & * & * \\ & & & & \underline{1} & * & * \end{pmatrix} \tag{3.18}$$

em que os índices-esquerda são representados por $\underline{1}$ e os índices-direita por $\bar{1}$. Os demais elementos do intervalo de cobertura são representados por asterisco.

Para cada matriz geradora produzida a partir do *template,* calcula-se o espectro de distâncias. Este procedimento é realizado de forma exaustiva para cada combinação de 0's e 1's do *template* e o código com melhor espectro é selecionado. Fazendo a busca exaustiva no *template* em (3.18), obtemos o código com $G(D)$ dada por

$$G(D) = \begin{pmatrix} 1+D & 1+D & 1 & 0 & 1 & 0 & 1 \\ D & D & 1+D & 1+D & 1 & 1 & 0 \\ D & 0 & D & D & 1+D & 1+D & 1+D \end{pmatrix} \tag{3.19}$$

$d_f = 7$ e N=(3,6,10,13,31). Este procedimento foi repetido para *templates* de vários valores de $TC(M_{min})$ e $MC(M_{min})$ para as taxas 2/4, 3/5, 3/7 e 4/7. As Tabelas 3.10-3.13 listam os códigos com melhores espectros para cada par $(TC(M_{min}), MC(M_{min}))$ resultante das buscas realizadas neste trabalho. Os códigos encontrados com $TC(M_{min})$ iguais mas $MC(M_{min})$ diferentes são destacados em negrito.

3.7 Conclusões

Neste capítulo, foi definida uma nova medida de complexidade computacional de decodificação de códigos convolucionais baseada no VA operando com decisão abrupta. A complexidade computacional depende das complexidades de treliça e comparativa, sendo que o peso da última é aproximadamente o dobro do peso da primeira. Para se obter esta medida, foram calculados o número de operações aritméticas realizadas pelo VA e seus custos computacionais de execução com base numa plataforma de processadores digitais de sinais. A complexidade computacional proposta representa o número de ciclos de máquina consumidos pelas operações aritméticas sobre um módulo de treliça. Foram considerados os módulos de treliça convencional e mínimo.

As complexidades de treliça, comparativa e computacional foram calculadas para códigos de diversas taxas. No universo de códigos de mesma taxa, aqueles que apresentam a mesma complexidade de treliça podem apresentar complexidade comparativa e, consequentemente, complexidade computacional diferentes. Uma busca de códigos baseada nas complexidades de treliça e comparativa foi conduzida para as taxas 2/4, 3/5, 4/7 e 5/7. A medida de complexidade computacional proposta neste capítulo representa mais fielmente o esforço computacional requerido pela decodificação implementada por software.

Tabela 3.10 – Bons códigos convolucionais de taxa 2/4.

$TC(M_{min})$	$MC(M_{min})$	d_f	N	$G(D)$
18	6	4	1,3,5,9,17	[1 3 0 1; 2 3 3 0]
24	**6**	**5**	**2,4,8,16,32**	**[6 3 2 0; 2 0 3 3]**
24	**8**	**6**	**10,0,26,0,142**	**[4 2 3 3; 6 6 4 2]**
32	8	6	2,6,10,22,50	[3 3 3 2; 6 4 3 3]
48	**12**	**6**	**1,8,9,16,57**	**[3 7 4 1; 2 3 3 2]**
48	**16**	**7**	**6,10,14,39,92**	**[7 1 3 2; 4 6 7 3]**
56	16	7	4,9,16,38,86	[6 3 1 3; 5 7 7 0]
64	16	7	2,8,18,35,70	[6 4 3 3; 3 3 6 4]
96	**24**	**8**	**12,0,52,0,260**	**[12 7 0 7; 6 6 7 4]**
96	**32**	**8**	**4,15,16,36,104**	**[14 6 1 7; 3 7 7 2]**
112	32	8	4,14,17,36,114	[16 4 2 7; 2 16 15 6]
128	32	8	2,10,16,31,67	[16 3 7 4; 6 16 14 7]
192	**48**	**8**	**1,9,15,33,80**	**[13 14 13 1; 6 7 3 6]**
192	**64**	**9**	**4,12,27,46,109**	**[11 7 2 7; 14 16 15 3]**

Tabela 3.11 – Bons códigos convolucionais de taxa 3/5.

$TC(M_{min})$	$MC(M_{min})$	d_f	N	$G(D)$
12	4	4	11,0,52,0,279	[0 1 1 1 1; 2 2 2 0 1; 2 2 0 3 0]
13,33	4	4	3,12,24,56,145	[1 0 1 1 1; 2 1 1 1 0; 2 2 2 1 1]
26,66	**8**	**4**	**2,6,20,54,140**	**[2 2 1 0 1; 6 2 2 3 1; 2 3 2 1 0]**
26,66	**9,33**	**4**	**1,12,32,68,173**	**[1 0 1 1 1; 2 1 1 2 1; 2 2 3 1 0]**
32	10,66	4	1,0,34,0,211	[1 0 1 1 1; 2 3 3 0 2; 4 2 3 3 0]
37,33	**10,66**	**5**	**6,18,40,103,320**	**[2 2 3 0 1; 6 0 2 2 3; 6 6 0 1 0]**
37,33	**13,33**	**5**	**4,11,29,90,254**	**[3 3 0 1 0; 2 1 3 2 1; 2 2 2 1 3]**
42,66	13,33	5	1,16,29,78,217	[1 3 0 1 1; 2 3 3 3 0; 4 0 2 3 3]
48	16	6	18,0,139,0,1202	[1 3 2 1 1; 2 1 3 3 0; 6 2 2 3 3]
53,33	16	6	15,0,136,0,1208	[3 2 3 2 3; 2 3 3 1 0; 6 2 0 3 1]
74,66	**21,33**	**6**	**13,0,122,0,1014**	**[4 0 3 3 2; 2 1 3 2 1; 3 3 0 2 2]**
74,66	**26,66**	**6**	**4,18,48,114,374**	**[3 2 3 2 1; 0 3 1 3 2; 4 4 2 3 3]**
96	32	6	2,18,,58,132,338	[1 1 1 3 1; 6 4 3 2 1; 0 4 6 3 3]
106,66	32	6	1,16,34,79,292	[3 3 2 3 2; 4 1 3 2 1; 4 6 6 1 1]
149,33	**42,66**	**7**	**19,48,99,319,988**	**[1 3 3 0 3; 6 3 2 3 0; 6 0 7 4 1]**
149,33	**53,33**	**7**	**9,36,65,236,671**	**[3 3 0 3 2; 4 4 2 3 3; 4 3 7 5 0]**

Tabela 3.12 – Bons códigos convolucionais de taxa 4/7.

$TC(M_{min})$	$MC(M_{min})$	d_f	N	$G(D)$
20	5	4	6,12,21,58,143	[1 1 1 1 0 1 0; 0 1 1 1 1 0 1; 2 2 1 1 0 0 0; 2 0 0 1 1 1 0]
22	**6**	**4**	**3,7,18,54,125**	**[0 2 2 2 0 1 1; 2 0 2 0 1 1 0; 0 0 0 1 1 1 1; 3 3 1 0 1 0 0]**
22	**7**	**4**	**1,9,21,45,118**	**[1 1 0 1 1 0 1; 0 1 1 0 1 1 0; 2 2 0 1 1 1 0; 0 0 2 2 1 1 1]**
26	8	5	7,17,39,96,249	[1 1 0 1 1 1 1; 2 1 1 1 0 0 1; 0 2 2 1 1 1 0; 2 2 0 2 0 1]
28	8	5	3,14,29,72,205	[0 1 1 1 1 1 1; 3 1 0 0 1 1 0; 2 3 1 0 0 2 3; 2 3 3 1 1 0 0]
40	**12**	**5**	**3,11,31,70,176**	**[1 0 1 0 1 1 1; 2 3 1 1 1 1 0; 2 0 2 1 0 1 1; 2 2 2 2 1 3 0]**
40	**14**	**6**	**24,0,144,0,1021**	**[3 1 1 0 1 0 1; 0 0 3 3 0 1 1; 2 2 2 0 1 1 1; 0 2 2 2 2 3 0]**
44	14	6	23,0,140,0,993	[3 1 0 1 0 1 1; 2 3 1 2 1 1 1; 2 2 2 1 1 1 0; 0 2 2 2 2 1 1]
48	**14**	**6**	**17,0,133,0,855**	**[1 1 1 0 1 1 1; 2 1 2 1 3 0 0; 2 2 1 1 1 1 0; 2 0 0 2 2 3 1]**
48	**16**	**6**	**15,0,120,0,795**	**[3 0 1 0 1 1 1; 0 1 3 2 1 0 1; 2 2 2 1 1 1 0; 2 2 2 2 2 3 1]**
56	16	6	7,21,47,132,359	[3 1 1 1 1 1 0; 2 3 0 2 3 1 0; 2 2 2 1 1 1 1; 2 0 0 0 2 3 3]
72	20	6	5,22,54,136,389	[0 3 3 3 1 0 0; 3 1 1 2 2 2 2; 1 1 0 1 3 2 0; 0 1 1 0 1 3 3]
80	24	6	3,18,36,96,291	[1 3 2 1 1 0 1; 2 3 1 2 1 1 1; 2 2 0 1 3 1 0; 0 0 2 2 2 3 3]
88	**28**	**6**	**1,18,35,73,258**	**[3 3 2 1 0 1 0; 2 1 1 3 1 1 1; 0 2 0 3 3 3 0; 2 2 2 2 2 1 3]**
88	**32**	**7**	**18,44,77,234,703**	**[3 3 1 1 1 0 0; 0 1 3 2 1 1 1; 0 2 0 3 3 3 0; 4 0 2 2 2 3 3]**
104	32	7	15,34,72,231,649	[1 3 2 1 1 1 1; 0 1 1 3 2 3 1; 2 2 0 3 3 1 0; 6 2 2 0 0 3 3]

Tabela 3.13 – Bons códigos convolucionais de taxa 5/7.

$TC(M_{min})$	$MC(M_{min})$	d_f	N	$G(D)$
17,6	7,2	3	4,23,87,299,1189	[1 0 1 1 0 1 0; 0 1 0 1 0 0 1; 2 0 0 1 1 0 0; 2 2 0 0 0 1 0; 0 2 2 0 0 0 1]
22,4	8	4	17,49,205,773,3090	[1 1 0 1 1 0 0; 0 1 1 1 0 1 1; 2 0 0 1 1 1 0; 2 2 2 0 1 0 0; 0 2 0 2 0 1 1]
28,8	11,2	4	15,40,174,658,2825	[2 2 2 2 2 0 1; 0 2 0 2 0 1 1; 2 2 2 0 1 0 0; 0 0 0 1 1 1 1; 3 1 0 1 0 1 0]
32	**11,2**	**4**	**10,52,169,712,3060**	**[0 2 2 0 2 0 1; 2 2 0 2 0 1 0; 2 2 2 0 1 0 0; 0 0 0 1 1 1 1; 1 2 1 0 1 1 0]**
32	**12,8**	**4**	**9,43,169,629,2640**	**[1 1 0 1 0 1 0; 0 0 1 1 1 1 1; 2 2 2 1 0 1 0; 2 0 0 2 1 1 0; 2 0 2 2 2 0 1]**
35,2	**12,8**	**4**	**6,40,137,544,2318**	**[2 2 2 2 2 0 1; 0 0 2 2 0 1 1; 2 2 2 0 1 0 0; 0 0 1 1 1 1 1; 1 2 1 0 1 1 0]**
35,2	**14,4**	**4**	**6,36,134,586,2528**	**[1 1 0 1 1 1 0; 0 1 1 1 0 0 1; 2 2 0 1 1 0 0; 2 0 0 2 1 1 0; 0 2 2 0 2 1 1]**
44,8	16	4	2,27,109,445,1955	[1 1 0 0 1 1 1; 2 1 1 1 0 1 0; 2 2 2 1 0 0 1; 0 2 0 2 1 1 0; 2 0 0 0 2 1 1]
64	**22,4**	**4**	**1,28,113,434,1902**	**[3 2 1 0 1 0 1; 2 0 3 1 1 1 0; 2 2 2 2 1 0 0; 2 2 2 0 2 1 0; 0 2 0 2 2 2 1]**
64	**25,6**	**4**	**1,21,91,331,1436**	**[3 0 1 0 1 0 1; 2 3 2 1 0 1 0; 2 2 0 1 1 1 1; 0 2 2 2 2 1 0; 0 2 2 2 0 2 3]**
70,4	28,8	5	16,88,314,1320,5847	[3 0 1 1 1 0 0; 2 3 1 1 0 1 0; 0 2 2 3 1 0 1; 0 2 2 0 2 1 1; 0 2 0 2 2 2 3]
76,8	**28,8**	**5**	**15,71,274,1179,5196**	**[1 0 1 1 1 0 1; 0 3 3 1 1 1 0; 2 2 0 3 0 1 0; 2 2 2 2 2 3 1; 2 2 0 0 3 0 1]**
76,8	**32**	**5**	**14,59,256,1078,4649**	**[3 0 1 1 1 1 1; 0 3 1 1 0 1 1; 2 2 2 3 0 1 0; 0 2 0 2 3 1 0; 2 0 2 0 0 3 1]**
89,6	32	5	9,54,236,987,4502	[3 0 1 1 1 0 1; 2 3 3 1 1 1 1; 0 0 2 3 1 1 0; 2 2 2 2 3 0 1; 0 2 0 2 2 3 2]
153,6	57,6	6	69,0,1239,0,2478	[1 2 3 0 1 0 1; 2 3 1 3 1 0 1; 2 2 2 1 2 1 0; 2 2 2 2 3 0 3; 0 2 2 2 0 3 1]

CAPÍTULO 4

SECCIONAMENTO DO MÓDULO DE TRELIÇA MÍNIMO

O seccionamento é uma técnica que altera a estrutura de um módulo de treliça mínimo. Uma característica deste módulo é sua estrutura irregular, podendo apresentar seções com número de estados diferente. Em algumas referências, o número máximo de estados é a medida de complexidade de treliça adotada [19][24] e está relacionada com a complexidade de implementação em hardware do algoritmo de Viterbi [28][44][45][46], bem como o número de seções do módulo [47][48]. Por exemplo, os registradores de memória que armazenam o caminho sobrevivente na etapa de *Traceback* e os vetores que armazenam as métricas de cada estado na etapa de ACS do decodificador de Viterbi, devem ter sua capacidade baseada no número máximo de estados da treliça. Entretanto, é possível seccionar o módulo de treliça mínimo chegando-se num módulo de treliça mais compacto e regular (do ponto de vista do número máximo de estados e do número de seções), mais vantajoso para implementações em hardware [49][50]. Isso pode ser alcançado sem nenhum custo adicional nas complexidades de treliça ou computacional. O impacto positivo da reestruturação do módulo é a redução ou padronização dos requisitos do decodificador. Além disso, a variedade de novas topologias obtidas pelo seccionamento pode resultar em módulos de treliça menos complexos para serem usados com outros algoritmos, como SOVA [32], BCJR [33] e Algoritmo-M [5]. Isto pode reduzir o consumo de energia do receptor, uma abordagem importante na literatura atual [6][31]. Este capítulo apresenta a técnica de seccionamento do módulo de treliça mínimo e utiliza a

complexidade computacional $TCC(M)$ definida no capítulo anterior como critério de avaliação da complexidade do módulo resultante. Identificam-se alguns padrões de seccionamento do módulo mínimo capazes de diminuir o número máximo de estados sem comprometer a $TCC(M)$ e o espectro de distâncias em relação ao módulo de treliça mínimo. Tabelas são apresentadas com padrões de seccionamento de módulos de treliças de códigos de diversas taxas que satisfazem esta condição. Esta análise também mostra como a busca de códigos, ou a listagem dos melhores códigos para uma dada complexidade, pode ser consideravelmente ampliada se o seccionamento for considerado. Várias topologias de treliça de códigos propostas na literatura, tais como PCCs [16] e aquelas propostas em [23], podem ser membros desta família de módulos de treliça mínimo seccionados.

4.1 Seccionamento do Módulo de Treliça

Um módulo de treliça mínimo M_{min} possui n seções compreendidas entre os tempos 0 a n. O seccionamento de M_{min} no tempo i, para $i = 1,...,n-1$, significa a remoção dos estados no tempo i e a conexão dos estados no tempo $i-1$ diretamente com os estados no tempo $i+1$ se existir um caminho entre estes em M_{min}. O rótulo dos ramos no módulo seccionado é formado pela concatenação dos rótulos dos ramos dos caminhos entre os tempos $i-1$ e $i+1$ em M_{min}. O módulo de treliça seccionado no tempo i possui $n' = n-1$ seções. Este procedimento pode ser aplicado sequencialmente em quaisquer tempos, portanto, há 2^{n-1} formas distintas de seccionar M_{min} e o número de seções do módulo de treliça seccionado varia entre $n-1$ e 1 (treliça convencional). O problema de seccionamento de um módulo consiste em encontrar a melhor escolha dentre todas as possibilidades que minimize uma medida de complexidade de treliça [30].

Define-se um vetor binário de seccionamento, denotado por ***vetsec***. O i-ésimo elemento deste vetor, $vetsec_i$, $i = 1,...,n-1$, indica que há seccionamento no tempo i se $vetsec_i = 1$; caso contrário $vetsec_i = 0$. Se $vetsec_i = 1,\ i = 1,...,n-1,$ o módulo resultante é o módulo convencional e se $vetsec_i = 0,\ i = 1,...,n-1,$ resulta no módulo de treliça mínimo. Os perfis de estados e de ramos do módulo de treliça seccionado M_{sec} são denotados, respectivamente, por $\boldsymbol{v}_{sec}$ e $\boldsymbol{b}_{sec}$. O perfil de rótulos da treliça seccionada,

denotado por $\boldsymbol{l}_{sec}$, define o número de bits que rotulam os ramos por seção. A seguir, é apresentado um exemplo de seccionamento de um módulo de treliça mínimo em que são listados os perfis e as complexidades do módulo de treliça seccionado.

Exemplo 4.1: Considere o código $C_1(5,3,3)$ com $TC(M_{min}) = 26,67$, perfis $\tilde{\boldsymbol{v}} = (3,3,4,3,4)$ e $\tilde{\boldsymbol{b}} = (1,1,0,1,0)$ com o módulo de treliça mínimo mostrado na Figura 4.1. A matriz geradora deste código é dada por [19, Tabela II]

$$G(D) = \begin{pmatrix} 1+D & 1 & 0 & 1 & 0 \\ 0 & 1 & 1+D & D & D \\ D & D & D & 1 & 1 \end{pmatrix}. \tag{4.1}$$

Este código possui $d_{free} = 4$ e $N = (1,5,13,39,111)$. Os perfis $\boldsymbol{v}_{sec}$, $\boldsymbol{b}_{sec}$, $\boldsymbol{l}_{sec}$ e as complexidades $TC(M_{sec})$, $MC(M_{sec})$ e $TCC(M_{sec})$ do módulo de treliça seccionado de algumas das 16 possibilidades de seccionamento do módulo de treliça mínimo de C_1 são mostrados na Tabela 4.1.

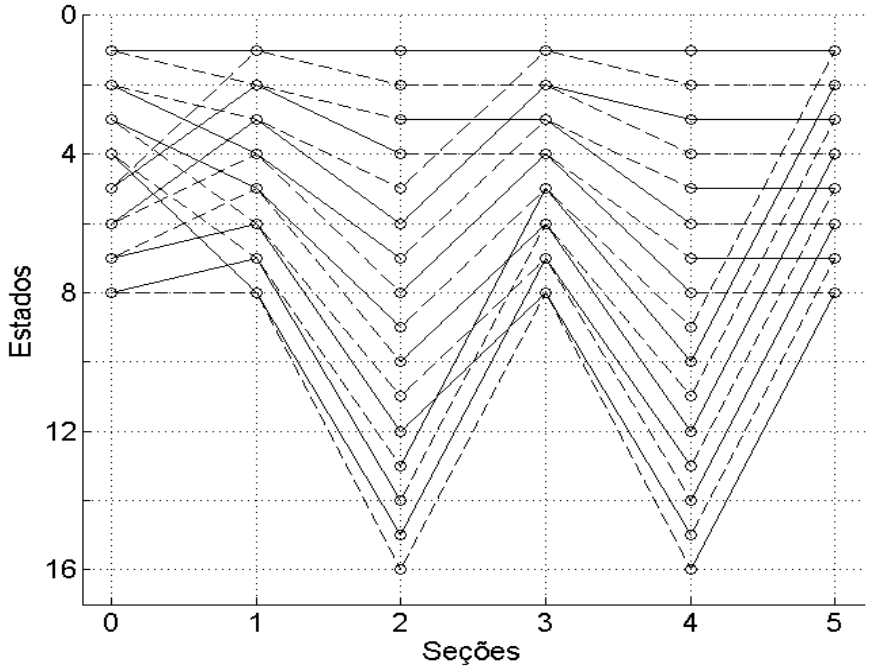


Figura 4.1 – Módulo de treliça mínimo de $C_1(5,3,3)$ considerado no Exemplo 4.1.

Tabela 4.1 – Resultados do seccionamento do módulo de treliça mínimo da Figura 4.1.

vetsec	$\boldsymbol{v}_{sec}$	$\boldsymbol{b}_{sec}$	$\boldsymbol{l}_{sec}$	$TC(M_{sec})$	$MC(M_{sec})$	$TCC(M_{sec})$
(0,0,0,0)	(3,3,4,3,4)	(1,1,0,1,0)	(1,1,1,1,1)	26,67	8	264
(0,1,0,0)	(3,3,3,4)	(1,1,1,0)	(1,2,1,1)	26,67	8	264
(0,1,0,1)	(3,3,3)	(1,1,1)	(1,2,2)	26,67	8	264
(1,0,0,0)	(3,4,3,4)	(2,0,1,0)	(2,1,1,1)	37,33	10,67	363
(1,1,0,0)	(3,3,4)	(2,1,0)	(3,1,1)	42,67	10,67	395
(1,1,0,1)	(3,3)	(2,1)	(3,2)	42,67	10,67	395
(1,0,1,0)	(3,4,4)	(2,1,0)	(2,2,1)	48	13,33	461
(1,0,1,1)	(3,4)	(2,1)	(2,3)	53,33	13,33	493
(1,1,1,0)	(3,4)	(3,0)	(4,1)	90,67	18,67	786
(1,1,1,1)	(3)	(3)	(5)	106,67	18,67	883

Observa-se na Tabela 4.1 que, para $\boldsymbol{vetsec} = (0,1,0,1)$ o número máximo de estados é reduzido de 16 para 8 e o número de seções é reduzido para $n' = 3$, permitindo até 2 bits por ramo, sem qualquer alteração nas complexidades. Não é possível reduzir n' além deste valor sem que haja um aumento nas complexidades. A Figura 4.2 mostra a estrutura resultante deste seccionamento. Os perfis $\boldsymbol{v}_{sec} = (3,3,3)$, $\boldsymbol{b}_{sec} = (1,1,1)$, $\boldsymbol{l}_{sec} = (1,2,2)$ e $TC(M_{sec}) = 26,67$ mostrados na Tabela 4.1 poderiam indicar que se trata da representação de um PCC de C_1, como listado em [4, Tabela XIII]. De fato, se o perfil de estados $\tilde{\boldsymbol{v}}$ do módulo de treliça mínimo é formado por elementos v e $v+1$, e nas seções em que $\tilde{v}_i = v$ implicar em $\tilde{b}_i = 1$, bem como nas seções em que $\tilde{v}_i = v+1$ implicar em $\tilde{b}_i = 0$, o seccionamento deste módulo de treliça nas seções em que $\tilde{b}_i = 0$ produz um módulo de treliça com os perfis $\boldsymbol{v}_{sec}$ e $\boldsymbol{b}_{sec}$ de um PCC. Entretanto, observando-se a Figura 4.2, o padrão de conexões de ramos não é igual em todas as seções da treliça, o que descaracteriza um PCC. Isto permite definir uma nova classe de códigos, mais geral que o PCC.

Definição 4.1: A classe de códigos que pode ser representada por um módulo de treliça que apresenta os perfis $\boldsymbol{v}_{sec}$, $\boldsymbol{b}_{sec}$ e $\boldsymbol{l}_{sec}$ compatíveis com a de um PCC, mas com padrões de conexões de ramos variando nas $n' = k$ seções da treliça, é definida como tipo-PCC ou simplesmente, t-PCC. A treliça da Figura 4.2 representa um código t-PCC.

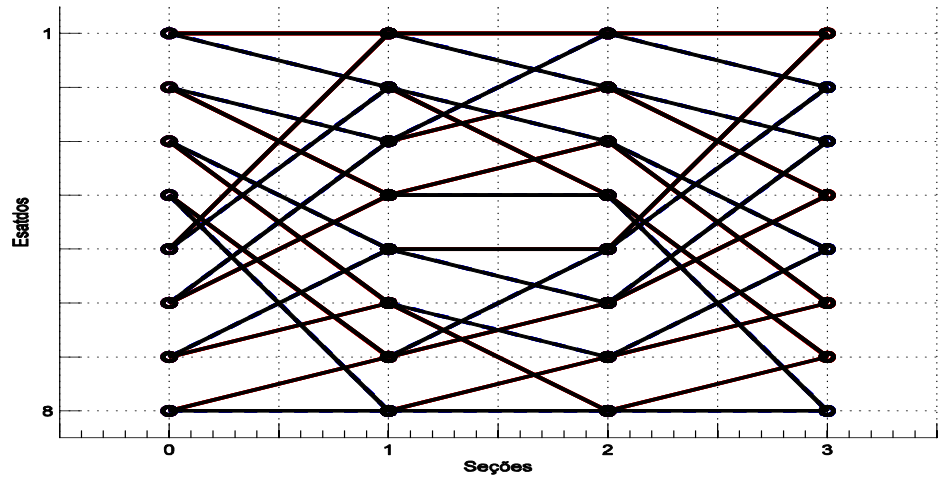


Figura 4.2 – Treliça resultante do seccionamento $vetsec_i = (0,1,0,1)$ do Exemplo 4.1.

Considerando-se ainda o código $C_1 = (5,3,3)$ do Exemplo 4.1, pode-se reduzir sucessivamente o número de seções mantendo-se o número máximo de estados igual a 8, ao custo de um aumento das complexidades, conforme é comentado a seguir.

- Para $\boldsymbol{vetsec} = (1,1,0,1)$, obtém-se $\boldsymbol{v}_{sec} = (3,3)$, $\boldsymbol{b}_{sec} = (2,1)$ e $\boldsymbol{l}_{sec} = (3,2)$. O número de seções é $n' = 2$, permitindo-se até 3 bits por ramo, $TC(M_{\text{sec}}) = 42,67$, $MC(M_{sec}) = 10,67$ e $TCC(M_{sec}) = 395$.
- Para $\boldsymbol{vetsec} = (1,1,1,1)$, obtém-se $\boldsymbol{v}_{sec} = (3)$, $\boldsymbol{b}_{sec} = (3)$ e $\boldsymbol{l}_{sec} = (5)$, equivalente ao módulo de treliça convencional de C_1, com $TC(M_{\text{sec}}) = 106,67$, $MC(M_{sec}) = 18,67$ e $TCC(M_{sec}) = 883$.

Em alguns casos, o seccionamento da treliça mínima pode produzir um PCC, como pode ser visto no próximo exemplo.

Exemplo 4.2: Considere o código $C_2(7,3,3)$ com $TC(M_{min}) = 37,33$ e perfis $\tilde{\boldsymbol{v}} = (3,4,3,4,3,4,4)$ e $\tilde{\boldsymbol{b}} = (1,0,1,0,1,0,0)$ [19, Tabela IV]. Este código possui $d_{free} = 7$ e $N = (3,6,10,13,31)$. A matriz geradora $G(D)$ é dada por

$$G(D) = \begin{pmatrix} 1+D & 1+D & 1 & 1 & 0 & 1 & 1 \\ D & 0 & 1+D & 1+D & 1 & 1 & 0 \\ D & D & 0 & D & 1+D & 1+D & 1+D \end{pmatrix}. \qquad (4.2)$$

O módulo de treliça mínimo de C_2 é mostrado na Figura 4.3. Considerando o vetor $\boldsymbol{vetsec} = (1,0,1,0,1,1)$, obtemos um módulo de treliça seccionado com perfis $\boldsymbol{v}_{sec} = (3,3,3)$, $\boldsymbol{b}_{sec} = (1,1,1)$, $\boldsymbol{l}_{sec} = (2,2,3)$, $TC(M_{sec}) = TC(M_{min}) = 37,33$ e $MC(M_{sec}) = MC(M_{min}) = 8$. A treliça seccionada representa um PCC de C_2, como pode ser observado na Figura 4.4.

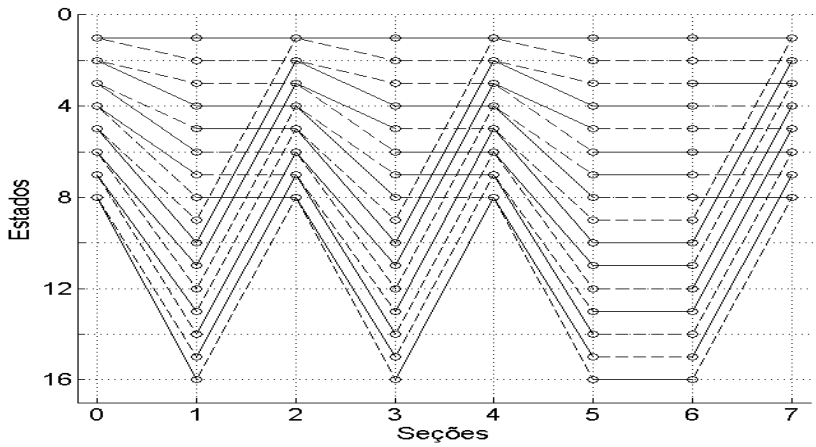


Figura 4.3 – Módulo de treliça mínimo de $C_2(7,3,3)$ considerado no Exemplo 4.2.

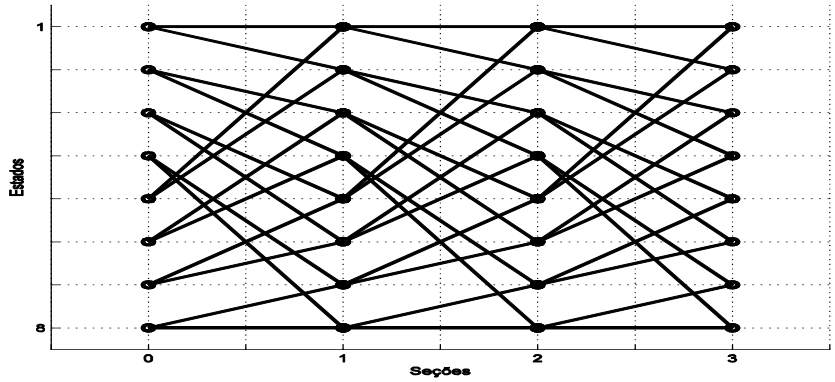


Figura 4.4 – Treliça PCC de C_2 resultante do seccionamento $\boldsymbol{vetsec} = (1,0,1,0,1,1)$ do Exemplo 4.2.

O código considerado neste exemplo possui espectro melhor do que o PCC de taxa $R = 3/7$ com matriz geradora $G(D)$ dada por

$$G(D) = \begin{pmatrix} 1+D & 1+D & 1 & 1 & 1 & 0 & 1 \\ D & 0 & 1+D & 1+D & 1 & 1 & 0 \\ 0 & 0 & D & 0 & 1+D & 1+D & 1+D \end{pmatrix}$$

e $TC(M_{PCC}) = 37,33,$ com $d_{free} = 7$ e $N = (3,7,12,14,31)$ [16, Tabela XIV].

A classe de códigos PCC pode ser identificada a partir da matriz geradora do código convolucional. A partir de (4.2), obtemos as submatrizes geradoras

$$G_0 = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix} \quad \text{e} \quad G_1 = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}.$$

A matriz $\hat{G}$ deste código é

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

em que os elementos ativos estão destacados em negrito. Esta é uma matriz típica de um PCC com três seções derivadas de um código mãe de taxa $R = 1/3$ com oito estados, sendo as duas primeiras seções puncionadas. Todas as colunas de $\hat{G}$ possuem uma sequência de quatro elementos ativos sem lacunas.

A classe de códigos t-PCC descrita na Definição 4.1 apresenta a matriz $\hat{G}$ com características diferentes. Considere a matriz geradora do código $C_1 = (5,3,3)$ do Exemplo 4.1 dada por

$$G(D) = \begin{pmatrix} 1+D & 1 & 0 & 1 & 0 \\ 0 & 1 & 1+D & D & D \\ D & D & D & 1 & 1 \end{pmatrix}$$

e submatrizes geradoras G_0 e G_1 dadas por

$$G_0 = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix} \quad \text{e} \quad G_1 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 \end{pmatrix}.$$

A matriz $\hat{G}$ deste código é

$$\hat{G} = \begin{pmatrix} \mathbf{1} & 0 & 0 & 0 & 0 \\ \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{1} & \mathbf{1} \\ \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 & 0 \\ \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{0} \\ 0 & \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{0} \\ 0 & 0 & 0 & \mathbf{1} & \mathbf{1} \end{pmatrix} \tag{4.3}$$

em que os elementos ativos estão destacados em negrito. Esta matriz não representa um PCC com três seções derivadas de um código mãe de taxa $R = 1/2$ com a primeira seção puncionada, visto que ocorre uma lacuna nos elementos ativos das duas últimas colunas. Observa-se que o número de elementos ativos por coluna de $\hat{G}$ em (4.3) é fixo, o que representa uma treliça seccionada com 8 estados em cada seção.

O Exemplo 4.1 ilustra o resultado do seccionamento do módulo de treliça mínimo do código $C_1(5,3,3)$ para vários valores de ***vetsec*** diferentes. Na Tabela 4.1, podemos verificar que as complexidades $TC(M_{sec})$, $MC(M_{sec})$ e $TCC(M_{sec})$ podem aumentar ou não em relação às complexidades do módulo mínimo. Dado um módulo M_{min}, analisamos todas as possibilidades de seccionamento de uma seção isolada deste módulo e construímos uma série de regras que norteiam os efeitos do seccionamento sobre as complexidades $TC(M_{sec})$, $MC(M_{sec})$ e $TCC(M_{sec})$. Por exemplo, é possível estabelecer as condições necessárias para que o módulo de treliça seccionado tenha as mesmas complexidades do módulo de treliça mínimo. Estas regras são mostradas na próxima seção, bem como o procedimento adotado para a construção destas.

4.2 Regras Gerais de Seccionamento do Módulo de Treliça

Primeiramente, vamos analisar os efeitos do seccionamento sobre as complexidades $TC(M_{sec})$, $MC(M_{sec})$ e $TCC(M_{sec})$ para um tipo específico de seção. Definimos um setor de treliça consistindo de duas seções consecutivas de um módulo de treliça mínimo M_{min}. Por exemplo, considere o setor de treliça S com os perfis $(\tilde{v}_{i-1}, \tilde{v}_i, \tilde{v}_{i+1}) = (v, v, v)$, $(\tilde{b}_{i-1}, \tilde{b}_i) = (1,0)$ e $(\tilde{l}_{i-1}, \tilde{l}_i) = (1,1)$. As complexidades de treliça e comparativa do setor S, denotadas por $TC(M_S)$ e $MC(M_S)$, respectivamente, são dadas por

$$TC(M_S) = \frac{1}{k}\sum_{m=-1}^{0} \boldsymbol{l}_{i+m} 2^{v_{i+m}+b_{i+m}} = \frac{1}{k} 3(2^v) \tag{4.4}$$

$$MC(M_S) = \frac{1}{k}\sum_{m=-1}^{0} (2^{v_{i+m}+b_{i+m}} - 2^{v_{i+1+m}}) = \frac{1}{k}(2^v). \tag{4.5}$$

Considere o seccionamento da seção i de S, resultando em uma única seção com $\boldsymbol{v}_{sec} = (v, v)$, $\boldsymbol{b}_{sec} = (1)$ e $l_{sec} = (2)$. As complexidades $T(M_{S'})$ e $MC(M_{S'})$ do setor seccionado $M_{S'}$ são dadas por

$$TC(M_{S'}) = \frac{1}{k} 4(2^v) \tag{4.6}$$

$$MC(M_{S'}) = \frac{1}{k}(2^v). \tag{4.7}$$

Portanto, para este caso específico de seccionamento da seção i em que $\tilde{b}_i = 0$, $\tilde{b}_{i-1} = 1$ e $\tilde{v}_i = \tilde{v}_{i-1}$, obtém-se $TC(M_{S'}) > TC(M_S)$ e $MC(M_{S'}) = MC(M_S)$. Repetimos este procedimento de forma exaustiva para todos os perfis de setores de um módulo de treliça mínimo M_{min} e com base nos resultados, criamos um conjunto de regras que se aplicam ao seccionamento de M_{min}. Seja o módulo de treliça mínimo M_{min} com perfis $\tilde{\boldsymbol{v}}$ e $\tilde{\boldsymbol{b}}$ e complexidades $TC(M_{min})$, $MC(M_{min})$ e $TCC(M_{min})$ e seja o módulo de treliça seccionado M_{sec} resultante do seccionamento de M_{min} na seção i, para $i = 1, ..., n-1$. As seguintes regras são aplicadas:

1. Se a seção i não contiver informação, ou seja, somente um ramo diverge de cada estado $(\tilde{b}_i = 0)$, então

   a) $MC(M_{sec}) = MC(M_{min})$.

   b) $TC(M_{sec}) = TC(M_{min})$ se $(\tilde{v}_i = \tilde{v}_{i-1} + 1$ e $\tilde{b}_{i-1} = 1)$ ou $(\tilde{v}_i = \tilde{v}_{i-1}$ e $\tilde{b}_{i-1} = 0)$.

   c) $TC(M_{sec}) > TC(M_{min})$ se $(\tilde{v}_i = \tilde{v}_{i-1} - 1$ e $\tilde{b}_{i-1} = 0)$ ou $(\tilde{v}_i = \tilde{v}_{i-1}$ e $\tilde{b}_{i-1} = 1)$.

2. Se a seção i contiver informação, ou seja, dois ramos divergem de cada estado $(\tilde{b}_i = 1)$, então

   a) $TC(M_{sec}) > TC(M_{min})$.

   b) $MC(M_{sec}) = MC(M_{min})$ se $(\tilde{v}_i = \tilde{v}_{i-1}$ e $\tilde{b}_{i-1} = 0)$ ou $(\tilde{v}_i = \tilde{v}_{i-1} + 1$ e $\tilde{b}_{i-1} = 1)$.

   c) $MC(M_{sec}) > MC(M_{min})$ se $(\tilde{v}_i = \tilde{v}_{i-1}$ e $\tilde{b}_{i-1} = 1)$ ou $(\tilde{v}_i = \tilde{v}_{i-1} - 1$ e $\tilde{b}_{i-1} = 0)$.

O caso analisado em (4.4)-(4.7) é refletido pela regra 1c. Observa-se pelas regras anteriores que o seccionamento produz módulos de treliça seccionados com complexidades de treliça e comparativa iguais ou maiores em relação ao módulo de treliça mínimo. Por exemplo, é possível manter ambas as complexidades seccionando-se as seções que não têm informação (pela regra 1), mesmo assim apenas em alguns casos específicos. O seccionamento pode reduzir a complexidade de decodificação caso o número máximo de estados seja utilizado como medida de complexidade. Estas regras permitem selecionar as seções de um módulo de treliça mínimo, de tal modo que o seccionamento destas resulte num módulo seccionado que satisfaça certos requisitos de complexidade. O próximo exemplo utiliza estas regras para obter o módulo de treliça seccionado mais compacto possível mantendo as complexidades do módulo de treliça mínimo. Isto é possível selecionando-se as seções em que $\tilde{b}_i = 0$ para $i = 1,...,n-1,$ de modo que a regra 1b seja satisfeita.

Exemplo 4.3: Considere o código $C_3(5,2,4)$ com $TC(M_{min}) = 96$, $MC(M_{min}) = 24,$ perfis $\tilde{\boldsymbol{v}} = (4,5,5,5,5)$ e $\tilde{\boldsymbol{b}} = (1,0,1,0,0)$, $d_{free} = 10,$ $N = (7,0,32,0,66)$ e matriz geradora dada por [19, Tabela I]

$$G(D) = \begin{pmatrix} 1+D+D^2 & 1+D^2 & 0 & 1+D^2 & 1+D+D^2 \\ D^2 & D+D^2 & 1+D+D^2 & 1+D & 1+D \end{pmatrix}. \tag{4.8}$$

O módulo de treliça mínimo M_{min} deste código é mostrado na Figura 4.5. A Tabela 4.2 mostra os perfis $\boldsymbol{v}_{sec}$, $\boldsymbol{b}_{sec}$, $\boldsymbol{l}_{sec}$ e as complexidades $TC(M_{sec})$, $MC(M_{sec})$ e $TCC(M_{sec})$ de alguns casos de seccionamento de M_{min} de C_3.

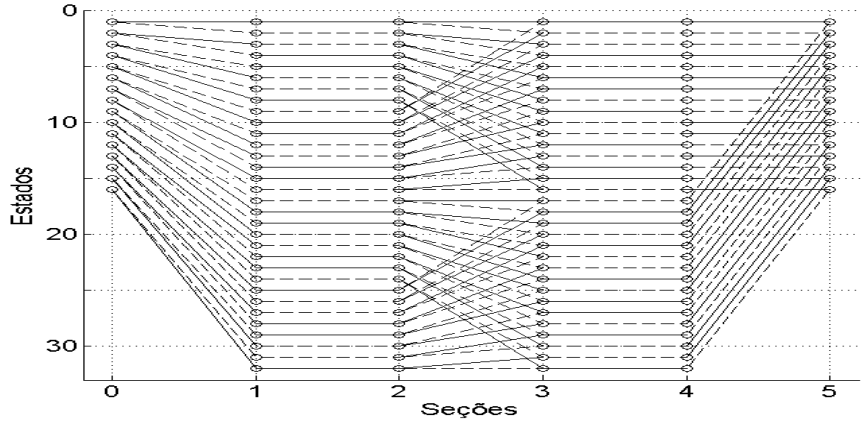


Figura 4.5 – Módulo de treliça mínimo de $C_3(5,2,4)$ considerado no Exemplo 4.3.

Tabela 4.2 – Resultados de seccionamento do módulo de treliça mínimo da Figura 4.5.

vetsec	$\boldsymbol{v}_{sec}$	$\boldsymbol{b}_{sec}$	$\boldsymbol{l}_{sec}$	$TC(M_{sec})$	$MC(M_{sec})$	$TCC(M_{sec})$
(1,0,0,1)	(4,5,5)	(1,1,0)	(2,1,2)	96	24	888
(0,1,0,0)	(4,5,5,5)	(1,1,0,0)	(1,2,1,1)	112	24	984
(0,0,1,0)	(4,5,5,5)	(1,0,1,0)	(1,1,2,1)	112	24	984
(1,0,1,1)	(4,5)	(1,1)	(2,3)	128	24	1080

Note que as seções 1, 3 e 4 do módulo de treliça mínimo da Figura 4.5 apresentam $\tilde{b}_i = 0$ e a seção 2, $\tilde{b}_i = 1$. Para manter as complexidades nos valores mínimos, o seccionamento das seções 2 e 3 é descartado, pois isto ocasiona um aumento da complexidade de treliça, conforme as regras 2a e 1c, respectivamente. Portanto, o seccionamento ótimo neste caso (pela regra 1b) é obtido com $\boldsymbol{vetsec} = (1,0,0,1)$ resultando no módulo de treliça seccionado da Figura 4.6 com $\boldsymbol{v}_{sec} = (4,5,5)$, $\boldsymbol{b}_{sec} = (1,1,0)$, $\boldsymbol{l}_{sec} = (2,1,2)$ e complexidades $TC(M_{sec}) = 96$, $MC(M_{sec}) = 24$ e $TCC(M_{sec}) = 888$, as mesmas do módulo de treliça mínimo. Por outro lado, se decidirmos pelo seccionamento das seções que apresentam $\tilde{b}_i = 0$, ou seja, dado $\boldsymbol{vetsec} = (1,0,1,1)$, obtemos $TC(M_{sec}) = 128$ e $MC(M_{sec}) = MC(M_{min}) = 24$ com $\boldsymbol{v}_{sec} = (4,5)$, $\boldsymbol{b}_{sec} = (1,1)$ e $\boldsymbol{l}_{sec} = (2,3)$, o mesmo módulo listado em [17, Tabela II]. A complexidade comparativa mantém-se inalterada (pela regra 1a), enquanto que a complexidade de treliça é aumentada devido ao seccionamento da seção 3 (pela regra 1c).

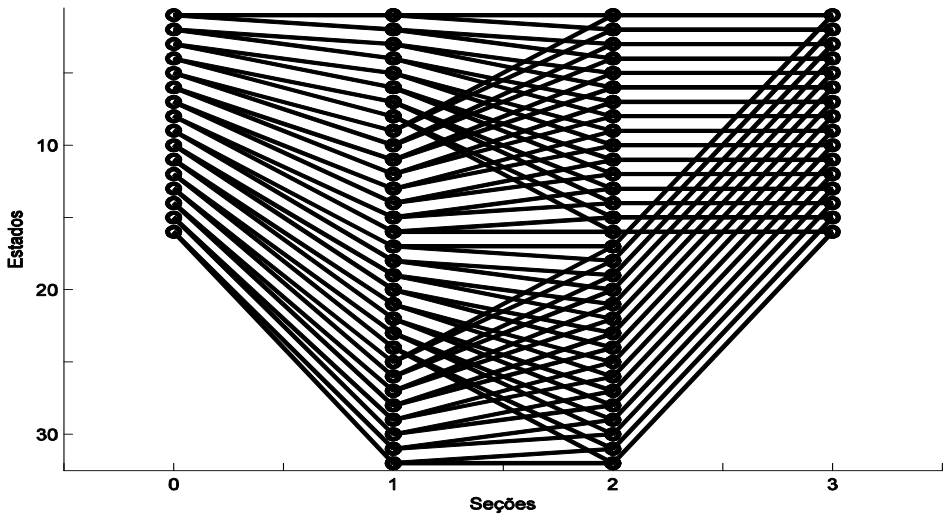


Figura 4.6 – Treliça resultante do seccionamento $\boldsymbol{vetsec} = (1,0,0,1)$ do Exemplo 4.3.

Embora no último caso, $\boldsymbol{vetsec} = (1,0,1,1)$, a complexidade de treliça apresente um aumento, o número de seções n' é reduzido de 5 para 2, produzindo uma treliça mais compacta, uma possível vantagem para algoritmos de decodificação subótimos, como o algoritmo M, que seleciona M caminhos sobreviventes em cada seção da treliça.

Exemplo 4.4: Considere o código $C_4(7,4,4)$ com $TC(M_{min}) = 80$, $MC(M_{min}) = 24$, perfis $\tilde{\boldsymbol{v}} = (4,5,6,5,5,4,5)$ e $\tilde{\boldsymbol{b}} = (1,1,0,1,0,1,0)$. Este código possui $d_{free} = 6$, $N = (4,20,45,116,306)$ e matriz geradora dada por [19, Tabela V]

$$G(D) = \begin{pmatrix} 1+D & 1+D & D & 0 & 1 & 1 & 1 \\ D & 1 & 1+D & 1+D & 0 & 1 & 0 \\ 0 & D & D & 1+D & 1+D & 1 & 0 \\ 0 & D & D & D & 0 & 1+D & 1+D \end{pmatrix}. \tag{4.9}$$

O módulo de treliça mínimo para este código é mostrado na Figura 4.7. A Tabela 4.3 mostra os perfis $\boldsymbol{v}_{sec}$, $\boldsymbol{b}_{sec}$, $\boldsymbol{l}_{sec}$ e as complexidades $TC(M_{sec})$, $MC(M_{sec})$ e $TCC(M_{sec})$ de algumas possibilidades de seccionamento da treliça mínima de C_4.

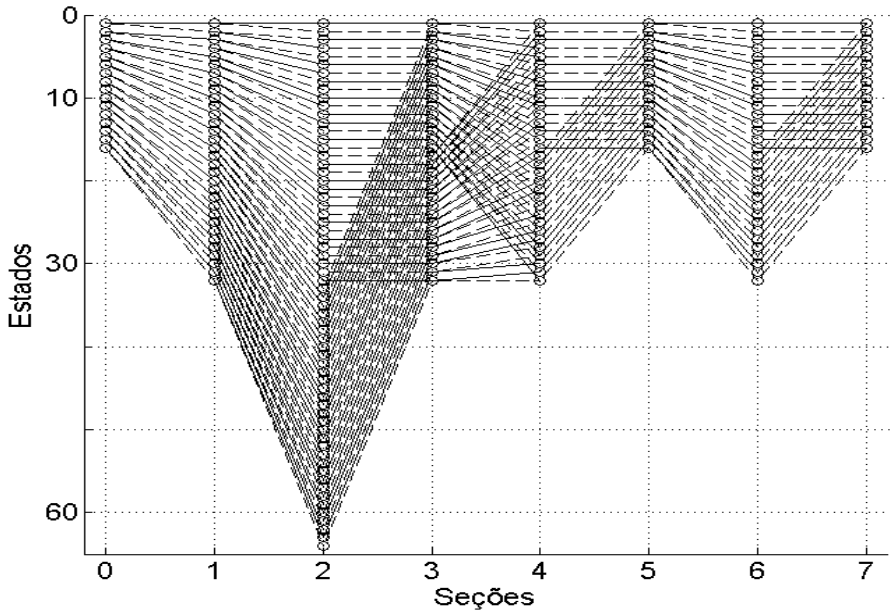


Figura 4.7 – Módulo de treliça mínimo de $C_4(7,4,4)$ considerado no Exemplo 4.4.

Tabela 4.3 – Resultados do seccionamento do módulo de treliça mínimo da Figura 4.7.

vetsec	$\boldsymbol{v}_{sec}$	$\boldsymbol{b}_{sec}$	$\boldsymbol{l}_{sec}$	$TC(M_{sec})$	$MC(M_{sec})$	$TCC(M_{sec})$
(0,1,0,0,0,1)	(4,5,5,5,4)	(1,1,1,0,1)	(1,2,1,1,2)	80	24	792
(0,1,0,1,0,1)	(4,5,5,4)	(1,1,1,1)	(1,2,2,2)	88	24	840
(1,1,0,1,0,1)	(4,5,4)	(2,1,1)	(3,2,2)	96	24	888
(1,1,1,1,0,1)	(4,4)	(3,1)	(5,2)	176	32	1472
(1,1,1,1,1,1)	(4)	(4)	(7)	448	60	3468

Primeiramente, considere o seccionamento do módulo de treliça mínimo da Figura 4.7 nas seções 2 e 6. Dado $\boldsymbol{vetsec} = (0,1,0,0,0,1)$, obtemos $TC(M_{sec}) = 80$ e $MC(M_{sec}) = 24$ com $\boldsymbol{v}_{sec} = (4,5,5,5,4)$, $\boldsymbol{b}_{\text{sec}} = (1,1,1,0,1)$ e $\boldsymbol{l}_{\text{sec}} = (1,2,1,1,2)$, resultando na estrutura mostrada na Figura 4.8. Este módulo de treliça seccionado é o mais compacto obtido com as mesmas complexidades do módulo de treliça mínimo (pela aplicação das regras 1a e 1b). Observe que o número máximo de estados é reduzido de 64 para 32. Agora assumimos o seccionamento das seções em que $\tilde{b}_i = 0$, ou seja, seções 2, 4 e 6. Dado $\boldsymbol{vetsec} = (0,1,0,1,0,1)$, obtemos o módulo de treliça seccionado com $TC(M_{sec}) = 88$, $MC(M_{sec}) = 24$, $\boldsymbol{v}_{sec} = (4,5,5,4)$, $\boldsymbol{b}_{sec} = (1,1,1,1)$ e $\boldsymbol{l}_{sec} = (1,2,2,2)$. Neste caso, o seccionamento das seções 4 e 6 causou um aumento da complexidade de treliça pela regra 1c, reduzindo-se n' para 4. Se incluirmos o seccionamento da seção 1, $\boldsymbol{vetsec} = (1,1,0,1,0,1)$, n' é reduzido para 3, obtendo-se $TC(M_{sec}) = 96$ pela regra 1a. Pode-se novamente reduzir o número máximo de estados para 16 (o mesmo da treliça convencional) e n' para 2, aplicando-se $\boldsymbol{vetsec} = (1,1,1,1,0,1)$, resultando $TC(M_{sec}) = 176$ e $MC(M_{sec}) = 32$. Ambas as complexidades de treliça e comparativa são aumentadas pelas regras 2a e 2c, respectivamente. Finalmente, obtemos a treliça convencional, $\boldsymbol{vetsec} = (1,1,1,1,1,1)$, resultando num aumento significativo das complexidades com $TC(M_{sec}) = 448$ e $MC(M_{sec}) = 60$. A Tabela 4.3 lista todos os casos discutidos anteriormente e o comportamento da $TCC(M_{sec})$ em cada um destes.

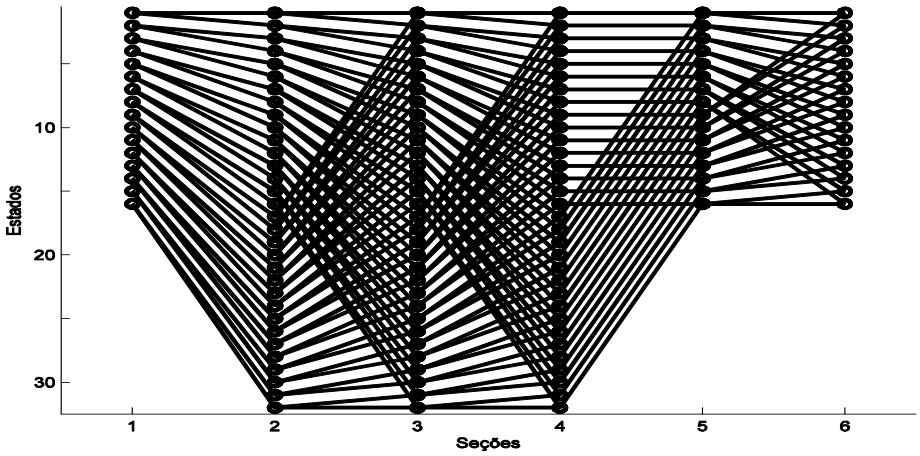


Figura 4.8 – Treliça resultante do seccionamento $\boldsymbol{vetsec} = (0,1,0,0,0,1)$ do Exemplo 4.4.

Há casos em que o seccionamento sempre ocasiona aumento da complexidade de treliça, como mostrado no próximo exemplo.

Exemplo 4.5: Considere o código $C_5(5,3,4)$ com $TC(M_{min}) = 74,67$, $MC(M_{min}) = 26,67$ e perfis $\tilde{\boldsymbol{v}} = (4,5,5,5,5)$ e $\tilde{\boldsymbol{b}} = (1,1,0,1,0)$. Este código possui $d_{free} = 6$, $N = (4,18,48,114,374)$ e matriz geradora dada por [19, Tabela II]

$$G(D) = \begin{pmatrix} 1 & 1+D & D & 1+D & 1 \\ D & 1+D & 1+D & 1 & D \\ D^2 & D^2 & D & 1+D & 1+D \end{pmatrix}. \tag{4.10}$$

O módulo de treliça mínimo para este código é mostrado na Figura 4.9. Note que a regra 1b não pode ser aplicada neste exemplo, o que ocasiona um aumento da complexidade de treliça para qualquer escolha de $\boldsymbol{vetsec}$. A Tabela 4.4 lista os perfis $\boldsymbol{v}_{sec}$, $\boldsymbol{b}_{sec}$ e $\boldsymbol{l}_{sec}$ e as complexidades $TC(M_{sec})$ e $MC(M_{sec})$ do módulo de treliça seccionado resultante de alguns casos de seccionamento de M_{min}.

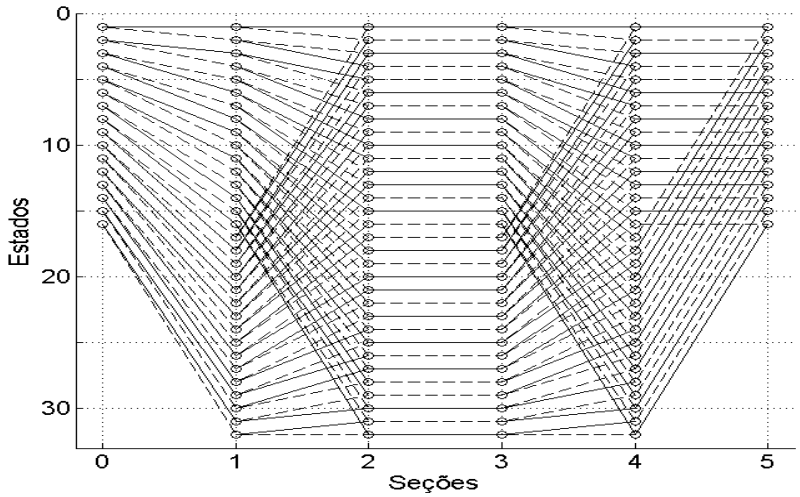


Figura 4.9 – Módulo de treliça mínimo de $C_5(5,3,4)$ considerado no Exemplo 4.5.

Tabela 4.4 – Resultados do seccionamento do módulo de treliça mínimo do Exemplo 4.5.

vetsec	$\boldsymbol{v}_{sec}$	$\boldsymbol{b}_{sec}$	$\boldsymbol{l}_{sec}$	$TC(M_{sec})$	$MC(M_{sec})$	$TCC(M_{sec})$
(1,0,0,0)	(4,5,5,5)	(2,0,1,0)	(2,1,1,1)	85,33	26,67	859
(0,1,0,1)	(4,5,5)	(1,1,1)	(1,2,2)	96	26,67	923
(1,1,0,1)	(4,5)	(2,1)	(3,2)	106,67	26,67	987
(1,1,1,1)	(4)	(3)	(5)	213,33	37,33	1765

Exemplo 4.6: Considere os códigos $C_6(7,3,3)$ com $TC(M_{min}) = 29,33$, $\tilde{\boldsymbol{v}} = (3,3,3,3,3,4,3)$, $\tilde{\boldsymbol{b}} = (1,0,1,0,1,0,0)$ e $C_7 = (7,3,3)$ com $TC(M_{min}) = 48$, $\tilde{\boldsymbol{v}} = (3,4,4,4,4,4,4)$, $\tilde{\boldsymbol{b}} = (1,0,1,0,1,0,0)$ [19, Tabela IV]. As matrizes geradoras destes códigos são mostradas, respectivamente, em (4.11) e (4.12). A Tabela 4.5 mostra os perfis $\boldsymbol{v}_{sec}$, $\boldsymbol{b}_{sec}$ e $\boldsymbol{l}_{sec}$ e as complexidades $TC(M_{sec})$, $MC(M_{sec})$ e $TCC(M_{sec})$ relativas ao seccionamento do módulo da treliça mínimo de C_6 e C_7 para $\boldsymbol{vetsec} = (1,1,0,0,0,0)$ e $\boldsymbol{vetsec} = (1,0,0,0,0,1)$, respectivamente.

$$G(D) = \begin{pmatrix} 1+D & 1 & 0 & 1 & 1 & 0 & 1 \\ D & D & 1+D & 1 & 0 & 1 & 1 \\ D & 0 & D & D & 1+D & 1+D & 1 \end{pmatrix} \tag{4.11}$$

$$G(D) = \begin{pmatrix} 1+D & 1+D & 1+D & 0 & 0 & 1 & 1 \\ 0 & D & 1+D & 1+D & 1+D & 1 & 0 \\ D & 0 & D & D & 1 & 1+D & 1+D \end{pmatrix}. \tag{4.12}$$

Tabela 4.5 – Resultado do seccionamento de $C_6(7,3,3)$ e $C_7=(7,3,3)$ do Exemplo 4.6.

Código	$\boldsymbol{v}_{sec}$	$\boldsymbol{b}_{sec}$	$\boldsymbol{l}_{sec}$	$TC(M_{sec})$	$MC(M_{sec})$	$TCC(M_{sec})$
$C_6(7,3,3)$	(3,3,3,4,3)	(2,0,1,0,0)	(3,1,1,1,1)	48	10,67	427
$C_7(7,3,3)$	(3,4,4,4,4)	(1,1,0,1,0)	(2,1,1,1,2)	48	13,33	461

Pela Tabela 4.5, podemos observar que os seccionamentos realizados neste exemplo produzem treliças com $TC(M_{sec})=48,$ mas com $MC(M_{sec})$ e, consequentemente, $TCC(M_{sec})$ diferentes. Isto indica que a complexidade computacional $TCC(M_{sec})$ também deve ser considerada na seleção de novas treliças geradas a partir do seccionamento.

4.3 Seleção do Número de Seções da Treliça Seccionada

Nesta seção, apresentaremos os novos perfis de treliças resultantes do seccionamento da treliça mínima de códigos de diversas taxas listados em [17][21][51][52]. Para uma dada taxa e uma dada $TC(M_{min}),$ partimos da treliça mínima com $n'=n$ seções e listamos os perfis $\boldsymbol{v}_{sec}$, $\boldsymbol{b}_{sec}$ e $\boldsymbol{l}_{sec}$ da treliça seccionada com menor n' e com as mesmas complexidades $TC(M_{sec})$ e $MC(M_{sec})$ e, consequentemente, com mesma $TCC(M_{sec})$ e espectro do módulo de treliça mínimo, ou seja, selecionamos o módulo de treliça seccionado mais compacto possível que satisfaça a regra de seccionamento 1b. As Tabelas 4.6-4.7 listam os perfis com menor valor de n' para cada $TCC(M_{sec})$ obtidos a partir do seccionamento dos módulos de treliça mínimos de taxas R=2/4, 2/5 e 3/5 e R=3/7 e 4/7, respectivamente, de códigos com matrizes geradoras $G(D)$ listadas em [21][51]. O menor valor de n' está compreendido entre n (treliça mínima) e k (PCC ou t-PCC). Outras topologias diferentes das apresentadas na literatura (com valores intermediários de n') também são obtidas e listadas nas Tabelas 4.6-4.7. Um aspecto importante da treliça seccionada é o número máximo de estados. Em geral, o menor valor de n' é desejável se isto ocasionar a redução do número máximo de estados, mantendo-se a $TCC(M_{sec})$ e o espectro. Por exemplo, para $R=2/5$ e $TCC(M_{sec})=688,$ obtemos $n'=2$ e redução do número máximo de estados de 32 para 16. Este módulo de treliça seccionado é um t-PCC com espectro melhor do que o respectivo PCC listado em [16] e mesma complexidade do módulo de treliça mínimo de partida. Pode ocorrer redução de n' sem uma redução do

número máximo de estados, mas ainda assim mantendo-se a $TCC(M_{sec})$ e o espectro. Por exemplo, para $R = 2/5$ e $TCC(M_{sec}) = 222$ e $TCC(M_{sec}) = 272$, obtemos $n' = 3$ e $n' = 4$, respectivamente, mantendo-se o número máximo de estados em 8. Em alguns casos, não é possível nenhuma redução de n' sem que isto ocasione um aumento de $TCC(M_{sec})$ em relação à treliça mínima. Nestes casos, a melhor escolha recai sobre a própria treliça mínima, ou seja, $n' = n$, como pode ser observado para $R = 3/5$ e $TCC(M_{sec}) = 464$. A Tabela 4.6 também indica os casos em que o seccionamento produz uma treliça de código PCC.

Na Tabela 4.8, listamos os perfis com menor valor de n' para cada $TCC(M_{sec})$ obtidos a partir do seccionamento dos módulos de treliça mínimos de taxas $(n-1)/n$, $n = 4,5,6$ listados em [17][52].

Observa-se nas Tabelas 4.6-4.8 que o seccionamento tem o potencial de produzir treliças mais vantajosas do que a treliça mínima em aplicações práticas, em virtude da diminuição do número máximo de estados e do número de seções, sem ocasionar nenhum acréscimo na $TCC(M_{min})$.

4.4 Seleção de Perfis de Melhor Espectro

A partir do seccionamento do módulo de treliça mínimo de códigos de diversas taxas listados em [17][21][51][52], selecionamos os perfis de treliça de melhor espectro para cada valor de n' e $TCC(M_{sec})$. Para uma dada taxa e uma dada $TC(M_{min})$, partimos da treliça mínima com $n' = n$ seções e realizamos todos os 2^{n-1} seccionamentos possíveis. Cada seccionamento gera uma nova treliça com perfis $\boldsymbol{\nu}_{sec}$, $\boldsymbol{b}_{sec}$ e $\boldsymbol{l}_{sec}$ e complexidades $TC(M_{sec})$ e $MC(M_{sec})$. Selecionamos várias $TC(M_{min})$ para cada taxa e repetimos este procedimento gerando uma lista de perfis de treliças seccionadas. Para cada novo n', $n' = n,...,2$, listamos os perfis $\boldsymbol{\nu}_{sec}$, $\boldsymbol{b}_{sec}$ e $\boldsymbol{l}_{sec}$ de melhor espectro para cada par $TC(M_{sec})$ e $MC(M_{sec})$ encontrado. Finalmente, alguns perfis foram descartados porque implicam num aumento de $TC(M_{sec})$ sem a correspondente melhoria no espectro. As Tabelas 4.9-4.11 listam os padrões de treliça de melhor espectro para cada valor de n' e $TCC(M_{sec})$ para taxas $R = k/5$ $k = 2,3$, $R = k/7$, $k = 3,4$, e $R = (n-1)/n$, $n = 4,5,6$.

Tabela 4.6 – Menor valor de n' para cada $TCC(M_{sec})$ resultante do seccionamento de códigos de taxas R=2/4, 2/5 e 3/5 listados em [21][51].

k / n	$TC(M_{sec})$	$MC(M_{sec})$	$TCC(M_{sec})$	n'	$\boldsymbol{v}_{sec}$	$\boldsymbol{b}_{sec}$	$\boldsymbol{l}_{sec}$	d_f	N
2/4	24^{s}	6	222	3	323	011	112	5	2,4,8,16,32
2/4	24^{m}	8	248	4	3333	1010	1111	6	10,0,26,0,142
2/4	$32^{s,p}$	8	296	2	33	11	22	6	2,6,10,22,50
2/4	48^{s}	12	444	3	344	110	121	6	1,8,9,16,57
2/4	48^{m}	16	496	4	4444	1010	1111	7	6,10,14,39,92
2/4	56^{s}	16	544	3	444	110	121	7	4,9,16,38,86
2/4	$64^{s,p}$	16	592	2	44	11	22	7	2,8,18,35,70
2/4	80^{n}	24	792	3	455	110	211	8	12,0,52,0,260
2/4	112^{m}	32	1088	4	6555	0011	1111	8	4,14,17,36,114
2/5	14^{n}	4	136	4	2222	1010	1112	5	3,2,1,7,8
2/5	16^{s}	4	148	3	222	110	212	5	2,1,3,7,4
2/5	$20^{s,p}$	4	172	2	22	11	23	6	1,4,5,8,17
2/5	24^{n}	6	222	3	233	110	212	6	1,2,4,8,12
2/5	28^{n}	8	272	4	3333	1010	1112	7	3,4,7,16,24
2/5	36^{n}	8	320	4	3343	1100	2111	7	1,3,6,8,12
2/5	$40^{s,p}$	8	344	2	33	11	23	8	5,0,13,0,45
2/5	64^{s}	16	592	3	444	110	212	8	1,4,6,11,18
2/5	$80^{s,t}$	16	688	2	44	11	23	9	2,5,8,10,20
2/5	96^{n}	24	888	3	455	110	212	10	7,0,32,0,66
2/5	128^{s}	32	1184	4	5555	1010	1121	10	2,8,9,13,26
3/5	12^{s}	4	124	4	2222	1011	1121	4	11,0,52,0,353
3/5	$13{,}33^{s,p}$	4	132	3	222	111	122	4	3,12,24,56,161
3/5	24^{s}	8	248	4	3333	1110	1211	4	1,7,19,48,128
3/5	$26{,}67^{s,t}$	8	264	3	333	111	122	4	1,5,13,39,111
3/5	$42{,}67^{m}$	16	464	5	4444	1101	1111	5	1,16,29,78,217
3/5	48^{s}	16	496	4	4444	1110	1211	6	18,0,139,0,1202
3/5	$53{,}33^{s,t}$	16	528	3	444	111	122	6	15,0,131,0,1216
3/5	64^{n}	21,33	661	4	4554	1101	1112	6	13,0,105,0,955
3/5	$74{,}67^{m}$	26,67	795	5	4555	1101	1111	6	4,18,48,114,374
3/5	$106{,}67^{s,t}$	32	1056	3	555	111	122	7	21,48,87,328,103
3/5	128^{n}	42,67	1327	4	5566	1110	2111	7	11,42,82,238,793

^{s} Redução de n' e do número máximo de estados em relação ao perfil da treliça mínima listado em [21][51] sem alteração da $TCC(M_{sec})$ e do espectro.

^{m} Perfil da treliça mínima listado em [21][51]. O seccionamento não produziu uma treliça com mesmo $TCC(M_{sec})$ e espectro da treliça mínima em [21][51].

^{n} Redução de n' em relação ao perfil da treliça mínima listado em [21][51], sem alteração da $TCC(M_{sec})$ e do espectro.

^{t} t-PCC com espectro melhor do que o respectivo PCC listado em [16].

^{p} Código PCC.

Tabela 4.7 – Menor valor de n' para cada $TCC(M_{sec})$ resultante do seccionamento de códigos de taxas R=3/7 e 4/7 listados em [21][51].

k/n	$TC(M_{sec})$	$MC(M_{sec})$	$TCC(M_{sec})$	n'	$\boldsymbol{v}_{sec}$	$\boldsymbol{b}_{sec}$	$\boldsymbol{l}_{sec}$	d_f	N
3/7	$21,33^{n}$	5,33	197	6	22333	11010	21111	6	3,9,4,18,40
3/7	$29,33^{s}$	8	280	6	33333	10101	11112	6	1,6,7,12,27
3/7	$37,33^{s,p}$	8	328	3	333	111	223	7	3,6,10,13,31
3/7	48^{n}	13,33	461	5	34444	11010	21112	8	10,0,28,0,106
3/7	$53,33^{n}$	16	528	6	44444	10101	11111	8	8,0,29,0,92
3/7	64^{n}	16	592	6	44445	11010	21111	8	5,11,11,27,52
3/7	$74,67^{s,p}$	16	656	3	444	111	223	8	1,9,10,16,42
3/7	$106,67^{n}$	32	1056	6	55555	10101	11111	9	6,7,13,30,43
3/7	128^{s}	32	1184	5	55555	10110	11221	10	21,0,47,0,188
4/7	22^{s}	7	223	5	33323	11011	11122	4	2,12,20,47,127
4/7	24^{s}	8	248	6	33333	11010	11111	4	1,7,18,40,110
4/7	26^{s}	8	260	5	33333	11101	12112	5	7,17,39,96,249
4/7	$28^{s,t}$	8	272	4	3333	1111	1222	5	3,14,29,72,205
4/7	48^{s}	16	496	6	44444	11101	12111	6	19,0,122,0,902
4/7	$56^{s,t}$	16	544	4	4444	1111	1222	6	8,27,46,143,380
4/7	80^{s}	24	792	5	45554	11101	12112	6	4,20,45,116,306
4/7	104^{s}	32	1040	5	55555	11110	12211	7	15,34,72,231,64
4/7	$112^{s,t}$	32	1088	4	5555	1111	2212	7	8,35,69,147,497
4/7	128^{n}	40	1288	5	56655	11011	11122	7	8,25,59,152,461

[s] Redução de n' e do número máximo de estados em relação ao perfil da treliça mínima listado em [21][51] sem alteração da $TCC(M_{sec})$ e do espectro.

[m] Perfil da treliça mínima listado em [21][51]. O seccionamento não produziu uma treliça com mesmo $TCC(M_{sec})$ e espectro da treliça mínima em [21][51].

[n] Redução de n' em relação ao perfil da treliça mínima listado em [21][51], sem alteração da $TCC(M_{sec})$ e do espectro.

[t] t-PCC com espectro melhor do que o respectivo PCC listado em [16].

[p] Código PCC.

Tabela 4.8 – Menor valor de n' para cada $TCC(M_{sec})$ resultante do seccionamento de códigos de taxas $(n-1)/n,\ n=4,5,6$ listados em [17][52].

k/n	$TC(M_{sec})$	$MC(M_{sec})$	$TCC(M_{sec})$	n'	$\boldsymbol{\nu}_{sec}$	$\boldsymbol{b}_{sec}$	$\boldsymbol{l}_{sec}$	d_f	N
3/4	$21{,}33^{s}$	8	232	3	333	111	211	4	29,0,532,0,10146
3/4	$26{,}67^{m}$	10,67	299	4	4334	0111	1111	4	10,46,202,949,4464
3/4	32^{m}	13,33	365	4	4344	0111	1111	4	5,36,152,708,3439
3/4	$42{,}67^{s,t}$	16	464	3	444	111	121	4	3,44,160,638,3558
3/4	$74{,}67^{m}$	32	864	4	5555	0111	1111	5	15,81,354,1766,9233
3/4	$85{,}33^{s,t}$	32	928	3	555	111	211	5	13,67,318,1587,8115
4/5	$20^{s,t}$	8	224	4	3333	1111	2111	3	5,36,200,1065,5693
4/5	36^{m}	16	424	5	44444	01111	11111	4	30,126,815,4822,2899
4/5	40^{s}	16	448	4	4444	1111	2111	4	31,0,1254,0,46870
4/5	48^{m}	20	548	5	54445	01111	11111	4	12,80,487,3066,19308
4/5	56^{m}	24	648	5	55445	10111	11111	4	7,73,432,2657,16795
4/5	72^{m}	32	848	5	55555	01111	11111	4	4,52,338,2022,12930
4/5	$80^{s,t}$	32	896	4	5555	1111	2111	4	3,55,305,1828,12099
5/6	$19{,}2^{s}$	8	219	5	33333	11111	12111	3	15,96,601,3903,25325
5/6	$38{,}4^{s}$	16	438	5	44444	11111	21111	4	111,0,5628,0,291251
5/6	$57{,}6^{m}$	25,6	678	6	54455	01111	11111	4	20,197,1372,10457,81
5/6	64^{m}	28,8	758	6	55455	10111	11111	4	19,160,1186,9132,707
5/6	$76{,}8^{s,t}$	32	877	5	55555	11111	12111	4	13,149,1064,8175,642

^{s} Redução de n' e do número máximo de estados em relação ao perfil da treliça mínima listado em [17][52] sem alteração da $TCC(M_{sec})$ e do espectro.

^{m} Perfil da treliça mínima listado em [17][52]. O seccionamento não produziu uma treliça com mesmo $TCC(M_{sec})$ e espectro da treliça mínima em [17][52].

^{t} t-PCC com espectro melhor do que o respectivo PCC listado em [16].

4.5 Conclusões

Neste capítulo, foi apresentado o seccionamento do módulo de treliça mínimo de códigos convolucionais. O principal objetivo consiste em determinar o melhor padrão de seccionamento dentre as 2^{n-1} possibilidades que minimize uma medida de complexidade em particular. Um conjunto de regras foi construído para avaliar os efeitos do seccionamento sobre as complexidades de treliça e comparativa. Foram identificados padrões de seccionamento capazes de reduzir o número máximo de estados e o número total de seções mantendo-se as mesmas complexidades de treliça e comparativa do módulo mínimo. Desta forma, foi estabelecido um bom compromisso entre as quatro medidas de complexidade consideradas: complexidade de treliça, complexidade comparativa, número máximo de estados e número total de seções. Tabelas são apresentadas com padrões de seccionamento de treliças mais compactas e de mesma complexidade da treliça mínima,

para códigos de diversas taxas. As topologias de treliças seccionadas propostas por este trabalho podem simplificar a implementação de decodificadores em hardware e são uma alternativa interessante sobre o módulo de treliça mínimo. Além disso, a variedade de novas topologias obtidas pelo seccionamento pode resultar em treliças com menor complexidade de decodificação a serem usadas por outros algoritmos de decodificação, como SOVA, BCJR e Algoritmo-M. Isto pode reduzir o consumo de energia total do receptor, um assunto de relevância na literatura atual.

Tabela 4.9 – Perfis de treliça de melhor espectro para cada valor de n' e $TCC(M_{sec})$ resultantes do seccionamento de códigos de taxas 2/5 e 3/5.

k/n	n'	$\boldsymbol{\nu}_{sec}$	$\boldsymbol{b}_{sec}$	$\boldsymbol{l}_{sec}$	$TC(M_{sec})$	$MC(M_{sec})$	$TCC(M_{sec})$	d_f	N
2/5	4	2222	1010	1112	14	4	136	5	3,2,1,7,8
2/5	4	2322	1010	1112	16	4	148	5	2,1,3,7,4
2/5	4	2233	1100	2111	20	4	172	6	1,4,5,8,17
2/5	4	2333	1100	2111	24	6	222	6	1,2,4,8,12
2/5	4	3333	1010	1112	28	8	272	7	3,4,7,16,24
2/5	4	3343	1100	2111	36	8	320	7	1,3,6,8,12
2/5	4	3344	1100	2111	40	8	344	8	5,0,13,0,45
2/5	4	4444	1100	2111	64	16	592	8	1,4,6,11,18
2/5	4	4454	1100	2111	72	16	640	9	2,5,10,18,30
2/5	4	4455	1100	2111	80	16	688	9	2,5,8,10,20
2/5	4	4555	1100	2111	96	24	888	10	7,0,32,0,66
2/5	4	5555	1010	1121	128	32	1184	10	2,8,9,13,26
2/5	3	222	110	212	16	4	148	5	2,1,3,7,4
2/5	3	223	110	212	20	4	172	6	1,4,5,8,17
2/5	3	233	110	212	24	6	222	6	1,2,4,8,12
2/5	3	333	110	122	32	8	296	7	3,4,7,16,24
2/5	3	334	110	212	40	8	344	8	5,0,13,0,45
2/5	3	444	110	212	64	16	592	8	1,4,6,11,18
2/5	3	445	110	212	80	16	688	9	2,5,8,10,20
2/5	3	455	110	212	96	24	888	10	7,0,32,0,66
2/5	3	555	110	131	144	32	1280	10	2,8,9,13,26
2/5	2	22	11	23	20	4	172	6	1,4,5,8,17
2/5	2	23	11	23	32	6	270	6	1,2,4,8,12
2/5	2	33	11	23	40	8	344	8	5,0,13,0,45
2/5	2	44	11	23	80	16	688	9	2,5,8,10,20
2/5	2	45	11	23	128	24	1080	10	7,0,32,0,66
3/5	4	2222	1011	1121	12	4	124	4	11,0,52,0,353
3/5	4	3333	1110	1211	24	8	248	4	1,7,19,48,128
3/5	4	3443	1101	1112	32	10,67	331	4	1,2,14,38,87
3/5	4	4444	1110	1211	48	16	496	6	18,0,139,0,1202
3/5	4	4554	1101	1112	64	21,33	661	6	13,0,105,0,955
3/5	4	4555	1101	1112	85,33	26,67	859	6	4,18,48,114,374
3/5	4	5566	1110	2111	128	42,67	1323	7	11,42,82,238,793
3/5	3	222	111	122	13,33	4	132	4	3,12,24,56,161
3/5	3	333	111	122	26,67	8	264	4	1,5,13,39,111
3/5	3	444	111	122	53,33	16	528	6	15,0,131,0,1216
3/5	3	455	210	311	96	26,67	923	6	4,18,48,114,374
3/5	3	555	111	122	106,67	32	1056	7	21,48,87,328,1037
3/5	3	566	210	311	192	53,33	1845	7	11,42,82,238,793
3/5	2	22	21	32	21,33	5,33	197	4	3,12,24,56,161
3/5	2	33	21	32	42,67	10,67	395	4	1,2,14,38,87
3/5	2	34	21	23	53,33	13,33	493	4	1,5,13,39,111
3/5	2	44	21	23	74,67	21,33	725	5	1,16,29,78,217
3/5	2	45	21	23	106,67	26,67	987	6	4,18,114,374
3/5	2	55	12	23	170,67	42,67	1579	7	11,42,82,238,793

Tabela 4.10 – Perfis de treliça de melhor espectro para cada valor de n' e $TCC(M_{sec})$ resultantes do seccionamento de códigos de taxas 3/7 e 4/7.

k/n	n'	$\boldsymbol{v}_{sec}$	$\boldsymbol{b}_{sec}$	$\boldsymbol{l}_{sec}$	$TC(M_{sec})$	$MC(M_{sec})$	$TCC(M_{sec})$	d_f	N
3/7	6	223333	110100	211111	21,33	5,33	197	6	3,9,4,18,40
3/7	6	333333	101010	111121	29,33	8	280	6	1,6,7,12,27
3/7	6	334344	110100	211111	37,33	8	328	7	3,6,10,13,31
3/7	6	344444	110100	211111	48	13,33	461	8	10,0,28,0,106
3/7	6	444444	101010	111112	53,33	16	528	8	8,0,29,0,92
3/7	6	444454	110100	211111	64	16	592	8	5,11,11,27,52
3/7	6	445455	110100	211111	74,67	16	656	8	1,9,10,16,42
3/7	6	555555	101010	111112	106,67	32	1056	9	6,7,13,30,43
3/7	6	555565	101100	112111	128	32	1184	10	21,0,47,0,188
3/7	5	23333	11010	12112	26,67	6,67	247	6	3,9,4,18,40
3/7	5	33333	10110	11131	32	8	296	6	1,6,7,12,27
3/7	5	33434	11010	21112	37,33	8	328	7	3,6,10,13,31
3/7	5	34444	11010	21112	48	13,33	461	8	10,0,28,0,106
3/7	5	44444	11010	12112	58,67	16	560	8	8,0,29,0,92
3/7	5	45444	10101	11113	69,33	16	624	8	5,11,11,27,52
3/7	5	44545	11010	21112	74,67	16	656	8	1,9,10,16,42
3/7	5	55555	10110	11221	128	32	1184	10	21,0,47,0,188
3/7	5	55655	11010	12121	138,67	32	1248	10	21,0,47,0,188
3/7	4	2333	2010	3121	32	6,67	279	6	3,9,4,18,40
3/7	4	3333	1110	1231	34,67	8	312	6	1,6,7,12,27
3/7	4	3334	1110	2221	37,33	8	328	7	3,6,10,13,31
3/7	4	3444	1110	2212	53,33	13,33	493	8	10,0,28,0,106
3/7	4	4444	1110	1312	64	16	592	8	8,0,29,0,92
3/7	4	4445	1110	2212	74,67	16	656	8	1,9,10,16,42
3/7	4	5555	1110	2212	128	32	1184	9	6,7,13,30,43
3/7	4	5556	1110	2212	149,33	32	1312	10	21,0,47,0,188
3/7	3	223	120	232	26,67	5,33	229	6	3,9,4,18,40
3/7	3	333	111	223	37,33	8	328	7	3,6,10,13,31
3/7	3	344	210	322	64	13,33	557	8	10,0,28,0,106
3/7	3	444	111	223	74,67	16	656	8	1,9,10,16,42
3/7	3	565	201	313	213,33	42,67	1834	10	21,0,47,0,188
4/7	6	333333	110101	111112	24	8	248	4	1,7,18,40,110
4/7	6	333334	111010	121111	26	8	260	5	7,17,39,96,249
4/7	6	444444	111010	121111	48	16	496	6	19,0,122,0,902
4/7	6	555655	111010	121111	104	32	1040	7	15,34,72,231,649
4/7	5	33323	11011	11122	22	7	223	4	2,12,20,47,127
4/7	5	33333	11011	11122	26	8	260	4	1,7,18,40,110
4/7	5	33334	11110	12121	28	8	272	5	7,17,39,96,249
4/7	5	44444	11110	12211	52	16	520	6	19,0,122,0,902
4/7	5	44445	11110	12211	56	16	544	6	8,27,46,143,380
4/7	5	45555	11110	12121	96	28	940	6	4,20,45,116,306
4/7	5	55555	11110	12211	104	32	1040	7	15,34,72,231,649
4/7	4	3323	1111	1222	24	7	235	4	2,12,20,47,127
4/7	4	3333	1111	1222	28	8	272	5	3,14,29,72,205
4/7	4	3334	2110	2311	36	10	346	4	1,7,18,40,110
4/7	4	4444	1111	1222	56	16	544	6	8,27,46,143,380
4/7	4	4555	2110	3121	104	28	988	6	4,20,45,116,306
4/7	4	5555	1111	2212	112	32	1088	7	8,35,69,147,497

Tabela 4.11 – Perfis de treliça de melhor espectro para cada valor de n' e $TCC(M_{sec})$ resultantes do seccionamento de códigos de taxas 3/4, 4/5 e 5/6.

k/n	n'	$\boldsymbol{v}_{sec}$	$\boldsymbol{b}_{sec}$	$\boldsymbol{l}_{sec}$	$TC(M_{sec})$	$MC(M_{sec})$	$TCC(M_{sec})$	d_f	N
3/4	3	333	111	211	21,33	8	232	4	29,0,532,0,10146
3/4	3	433	012	112	32	10,67	331	4	10,46,202,949,4464
3/4	3	434	021	121	37,33	13,33	398	4	5,36,152,708,3439
3/4	3	444	111	121	42,67	16	464	4	3,44,160,638,3558
3/4	3	555	111	211	85,33	32	928	5	13,67,318,1587,8115
4/5	4	3333	1111	2111	20	8	224	3	5,36,200,1065,5693
4/5	4	4444	1111	2111	40	16	448	4	30,126,815,4822,28996
4/5	4	5444	0112	1112	56	20	596	4	12,80,487,3066,19308
4/5	4	5545	1021	1121	72	28	796	4	7,73,432,2657,16795
4/5	4	5555	1111	2111	80	32	896	4	4,52,338,2022,12930
4/5	4	5655	1111	1211	112	40	1192	4	3,55,305,1828,12099
5/6	5	33333	11111	12111	19,20	8	219	3	15,96,601,3903,25325
5/6	5	44444	11111	21111	38,4	16	438	4	111,0,5628,0,291251
5/6	5	54455	01211	11211	64	25,6	717	4	20,197,1372,10457,811
5/6	5	54555	11111	21111	70,4	28,8	797	4	19,160,1186,9132,7070
5/6	5	55555	11111	12111	76,8	32	877	4	13,149,1064,8175,6429

CAPÍTULO 5

TRELIÇA MÍNIMA PARA CODIFICADORES CONVOLUCIONAIS SISTEMÁTICOS RECURSIVOS

Nos capítulos anteriores, abordamos aspectos da decodificação de códigos convolucionais não-sistemáticos não-recursivos relacionados à complexidade computacional do algoritmo de Viterbi e ao seccionamento do módulo de treliça mínimo. Estes estudos foram conduzidos considerando códigos de diversas taxas. Porém, uma classe de aplicações importante, como gravações magnéticas e fibras ópticas, requer taxas de dados altas e probabilidade de erro baixa [18]. Códigos turbo de taxas altas são fortemente utilizados nestes casos pelo seu excelente desempenho. O esquema de codificação turbo típico com dois codificadores constituintes de taxa $1/2$ conectados em paralelo possui taxa $1/3$. Para aumentar esta taxa, pode-se utilizar a técnica de puncionamento. Uma alternativa é utilizar codificadores convolucionais sistemáticos recursivos de taxas altas como codificadores constituintes em esquemas turbo [34]. Codificadores destes tipos são propostos em [34] para taxas $(n-1)/n$.

Neste capítulo, abordaremos os fundamentos de códigos e codificadores convolucionais sistemáticos recursivos em esquemas turbo. A construção da treliça mínima para estes codificadores será introduzida e a complexidade de decodificação do algoritmo Max-log-MAP será analisada. Finalmente, apresentamos o resultado da busca de bons

codificadores sistemáticos recursivos de taxas diferentes de $(n-1)/n$ para serem utilizados em códigos turbo.

5.1 Códigos e Codificadores Turbo

Códigos turbo, introduzidos por Berrou, Glavieux e Thitimajshima [53], apresentam desempenho próximo dos limites teóricos estabelecidos pela teoria da informação. Estes códigos consistem em duas ideias fundamentais [54]: um esquema de codificação que utiliza dois ou mais codificadores constituintes concatenados em série ou paralelo, resultando num código longo com características pseudo-aleatórias, e um procedimento de decodificação iterativo de forma a refinar uma medida de confiabilidade sobre os dados a serem decodificados.

A Figura 5.1 mostra a configuração típica de um codificador turbo formada por dois codificadores convolucionais constituintes sistemáticos recursivos (CCCSRs) de taxa $1/2$ conectados em paralelo por um entrelaçador (representado pelo bloco Π), resultando num código de taxa $1/3$.

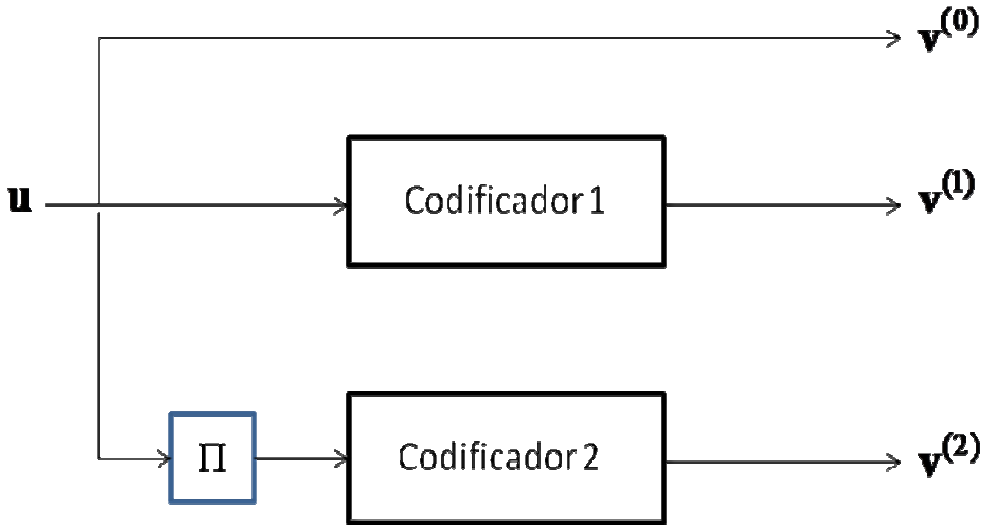


Figura 5.1 – Diagrama em blocos de um codificador turbo com dois CCCSRs conectados em paralelo.

Considere a sequência de informação de comprimento K, denotada por $\mathbf{u} = (u_0, ..., u_{K-1})$. Os primeiros $K-m$ bits de $\mathbf{u}$ são de informação e os restantes são bits de terminação. O codificador 1 deve ser terminado e retornar ao estado zero no tempo K. Como o codificador é sistemático, a primeira sequência de saída é $\mathbf{v}^{(0)} = \mathbf{u}$. O codificador 1 gera a sequência de paridade $\mathbf{v}^{(1)}$. O entrelaçador permuta os K bits de informação, produzindo a sequência pseudo-aleatória $\mathbf{u}'$ na entrada do codificador 2. A sequência de paridade do codificador 2 é $\mathbf{v}^{(2)} = (v_0^{(2)}, v_1^{(2)}, ..., v_{K-1}^{(2)})$. Os dois codificadores operam com versões permutadas da sequência de informação, embora estas tenham o mesmo peso de *Hamming*.

O espectro de um esquema turbo com dois CCCSRs conectados em paralelo por um entrelaçador é menos denso nas distâncias pequenas quando comparado com o de um CCCSR isolado. A alteração do espectro é resultado direto do entrelaçador que permuta os bits de informação para codificá-los novamente. Assim, palavras-código de peso alto na saída do codificador 1 estão associadas a palavras-código de peso alto na saída do codificador 2. Este efeito, conhecido como *spectral thinning*, é típico de um esquema concatenado em paralelo com CCCSRs.

O entrelaçador é parte integrante do codificador turbo e é responsável pela pseudo-aleatoriedade destes códigos. Devido à presença do entrelaçador, há uma grande probabilidade de o codificador 2 não ser terminado. Apesar de alguns autores terem proposto técnicas para eliminar este problema [55][56], simulações mostram que esta não terminação resulta em uma degradação imperceptível do desempenho para entrelaçadores longos. O projeto de entrelaçadores para códigos turbo é abordado em [57].

A Figura 5.2 mostra um exemplo de um codificador turbo de taxa $1/3$ em que o polinômio gerador de cada CCCSR é dado por

$$G_{sys}(D) = \left(1 \quad \frac{1 + D + D^2 + D^3 + D^4}{1 + D + D^4} \right). \tag{5.1}$$

Códigos turbo de taxas maiores do que $1/3$ são de interesse em várias aplicações. Duas técnicas diferentes são abordadas na literatura para aumentar a taxa de um código turbo: o puncionamento de um ou mais bits da saída do codificador [58][59] e a utilização de CCCSRs de taxas maiores do que 1/2 [18][34]. Na próxima seção, essas duas técnicas serão discutidas.

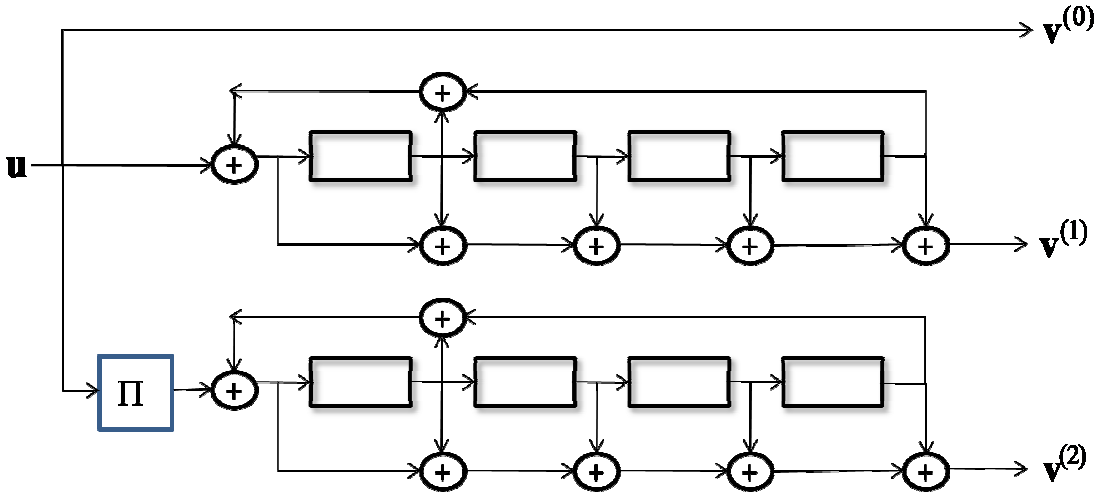


Figura 5.2 – Codificador turbo com polinômio gerador dado em (5.1).

5.2 Puncionamento x CCCSR de taxa alta

A técnica de puncionamento consiste em puncionar (apagar) um ou mais bits da saída do codificador a cada período de puncionamento p. Seja $\mathbf{P}$ uma matriz-puncionamento binária de período p associada a um código turbo com N sequências de saída dada por

$$\mathbf{P} = \begin{pmatrix} h_{11} & \cdots & h_{1N} \\ \cdots & h_{ik} & \cdots \\ h_{p1} & \cdots & h_{pN} \end{pmatrix} \tag{5.2}$$

em que cada coluna de $\mathbf{P}$ corresponde a uma sequência de saída, ou seja, a primeira coluna corresponde à sequência sistemática $\mathbf{v}^{(0)}$ do codificador e assim por diante, e cada linha de $\mathbf{P}$ representa uma sequência de informação contida no período de puncionamento p. Note que $h_{ik} \in \{0,1\}$, em que 0 significa que o bit correspondente é puncionado. Por exemplo, se considerarmos o puncioamento alternado dos bits de paridade $v_l^{(1)}$ e $v_l^{(2)}$, $l = 0,...,K-1$, do codificador turbo da Figura 5.2, temos $p = 2$ e a matriz $\mathbf{P}$ é dada por

$$\mathbf{P} = \begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}. \tag{5.3}$$

Assim, a sequência de bits de informação $\mathbf{u} = (u_0, u_1, ..., u_{K-1})$ gera a palavra-código $\mathbf{v} = (v_0^{(0)}, v_0^{(1)}, v_1^{(0)}, v_1^{(2)}, ..., v_{K-2}^{(0)}, v_{K-2}^{(1)}, v_{K-1}^{(0)}, v_{K-1}^{(2)})$, obtendo-se assim uma taxa $R = 1/2$.

Apesar de aumentar a taxa em esquemas de codificação turbo, o puncionamento causa uma possível piora no espectro de distância (ou mesmo na distância livre) do código [18]. Em contrapartida, é vantajoso do ponto de vista de flexibilidade, já que é possível obter várias taxas diferentes sem modificar a estrutura do codificador/decodificador que permanece essencialmente o mesmo do código-mãe de taxa $R = 1/2$.

Uma outra técnica para aumentar a taxa de códigos turbo é utilizar CCCSRs de taxas altas. Em [18][34], foi proposta a substituição dos códigos constituintes paralelos de taxa $R = 1/2$ por códigos de taxa $k/(k+1)$. Isso implica utilizar CCCSRs de múltiplas entradas (neste caso, k entradas) em vez de apenas uma entrada do esquema turbo clássico [34]. Por exemplo, se dois CCCSRs de taxa $R = 4/5$ forem utilizados no esquema de codificação da Figura 5.1, obtém-se um codificador turbo de taxa $R = 4/6$. Em [34], foram discutidas várias vantagens deste esquema: melhor convergência do processo iterativo, maiores distâncias mínimas, menor grau de puncionamento para atingir taxas ainda maiores, maior *throughput* e menor latência de decodificação (hardware) além de melhor robustez do decodificador.

Um aspecto negativo de CCCSRs de taxas altas é a complexidade de decodificação. Em vários algoritmos de decodificação, a complexidade de decodificação aumenta exponencialmente com k e v. Uma forma de reduzir a complexidade de decodificação é considerar a utilização da treliça mínima. Entretanto, a treliça mínima para códigos convolucionais da forma como é apresentada em [25] não pode ser diretamente adotada para códigos turbo. Isso é devido ao fato de que o mapeamento entre bits de informação e bits codificados produzido pela treliça mínima corresponde a codificadores convolucionais não-sistemáticos, enquanto que um mapeamento sistemático recursivo é necessário para CCCSRs de um código turbo. Portanto, para poder ser utilizada em códigos turbo, é necessário adaptar o mapeamento da treliça mínima para torná-lo sistemático. Na próxima seção, descrevemos um novo método para construir a treliça mínima para um CCCSR, dada a matriz geradora $G_{sys}(D)$.

5.3 Construção da Treliça Mínima a partir de $G_{SYS}(D)$

A construção da treliça mínima para um CCCSR a partir da matriz $G_{sys}(D)$ envolve duas etapas. Primeiramente, deve-se encontrar a treliça mínima para a matriz geradora não-sistemática não-recursiva básica-mínima na forma LR equivalente a $G_{sys}(D)$. Depois, o mapeamento entre os bits de informação e bits codificados é trocado para atender aos requisitos da treliça sistemática recursiva. Estas etapas são abordadas nas duas seções a seguir.

5.3.1 Matrizes Geradoras Equivalentes

Considere a matriz geradora sistemática recursiva $G_{sys}(D)$ de um código convolucional de taxa $R = k/n$, e seja $q(D)$, o mínimo múltiplo comum dos denominadores dos elementos de $G_{sys}(D)$. Para encontrar a matriz geradora não-sistemática não-recursiva básica $G_b(D)$, equivalente a $G_{sys}(D)$, usamos o seguinte procedimento [37][38]:

- Encontrar a decomposição de Smith da matriz polinomial $q(D)G_{sys}(D)$:

$$q(D)G_{sys}(D) = A(D)\Gamma'(D)B(D)$$

  em que os elementos não-zeros de $\Gamma'(D)$ são chamados de fatores invariantes de $q(D)G_{sys}(D)$, a matriz k x k $A(D)$ e a matriz n x n $B(D)$ são polinomiais. Logo, a decomposição de $G_{sys}(D)$ pode ser escrita da seguinte forma:

$$G_{sys}(D) = A(D)\Gamma(D)B(D)$$

  em que $\Gamma(D) = \Gamma'(D)/q(D)$.

- $G_b(D)$ é a matriz k x n que consiste das primeiras k linhas de $B(D)$.

Finalmente, utiliza-se o algoritmo proposto em [28] e realiza-se uma sequência de operações nas linhas de $G_b(D)$ para se obter a matriz não-sistemática não-recursiva básica-mínima na forma LR equivalente $G(D)$, como visto no Exemplo 2.10.

Exemplo 5.1: Considere a matriz geradora sistemática recursiva de um código $C(4,3,2)$ [1, Tabela IV]

$$G_{sys}(D)=\begin{pmatrix} 1 & 0 & 0 & \dfrac{1}{1+D^2} \\ 0 & 1 & 0 & \dfrac{1+D}{1+D^2} \\ 0 & 0 & 1 & \dfrac{1+D+D^2}{1+D^2} \end{pmatrix}. \tag{5.4}$$

Para $G_{sys}(D)$ em (5.4), obtemos a matriz polinomial

$$q(D)G_{sys}(D)=\begin{pmatrix} 1+D^2 & 0 & 0 & 1 \\ 0 & 1+D^2 & 0 & 1+D \\ 0 & 0 & 1+D^2 & 1+D+D^2 \end{pmatrix} \tag{5.5}$$

em que $q(D)=1+D^2$. Utilizando-se a decomposição de Smith, obtemos $G_{sys}(D)=A(D)\Gamma(D)B(D)$ a seguir

$$G_{sys}(D)=\begin{pmatrix} 1 & 0 & 0 \\ 1+D & 1 & 1 \\ 1+D+D^2 & 1 & 0 \end{pmatrix}\begin{pmatrix} \dfrac{1}{1+D^2} & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}\begin{pmatrix} 1+D^2 & 0 & 0 & 1 \\ 1+D+D^2 & 0 & 1 & 0 \\ D^2 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}. \tag{5.6}$$

Portanto, a matriz $G_b(D)$, equivalente a $G_{sys}(D)$, obtida a partir de $B(D)$ é dada por

$$G_b(D)=\begin{pmatrix} 1+D^2 & 0 & 0 & 1 \\ 1+D+D^2 & 0 & 1 & 0 \\ D^2 & 1 & 1 & 0 \end{pmatrix}. \tag{5.7}$$

Utilizando o algoritmo em [28], obtemos a seguinte matriz não-sistemática não-recursiva na forma LR equivalente a $G_{sys}(D)$ em (5.4)

$$G(D)=\begin{pmatrix} D & 1+D & 1 & 0 \\ D & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}. \tag{5.8}$$

A treliça mínima para o código convolucional com matriz dada em (5.8) é mostrada na Figura 5.3. Para este módulo de treliça, obtemos $\tilde{\boldsymbol{v}} = (2,2,2,3)$, $\tilde{\boldsymbol{b}} = (1,1,1,0)$ e $TC(M_{min}) = 10,67$.

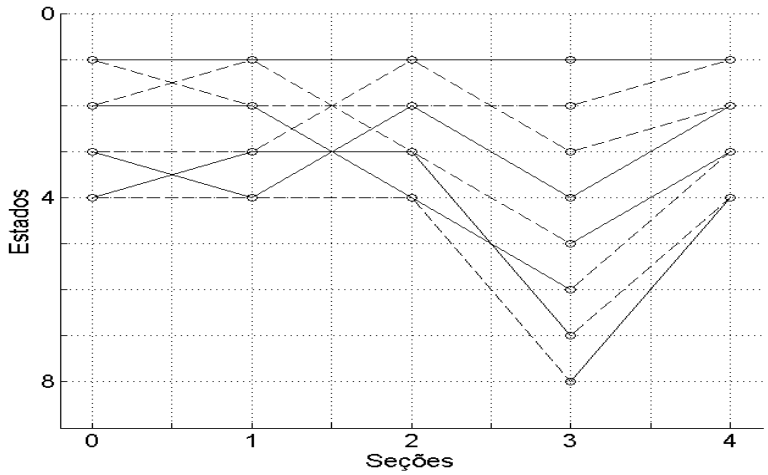


Figura 5.3 – Treliça mínima para o código convolucional $C(4,3,2)$ com $G(D)$ dada em (5.8). As linhas em cheio representam bit codificado 0 e em tracejado, bit codificado 1.

Nas três primeiras transições da treliça mínima da Figura 5.3, os ramos superiores e inferiores correspondem, respectivamente, ao bit de informação 0 e 1, conforme a convenção padronizada em [25]. As linhas em cheio representam bit codificado 0 e em tracejado, bit codificado 1. Portanto, o mapeamento entre os bits de informação e os bits codificados mostrados na Figura 5.3 não é sistemático e esta treliça não corresponde ao codificador com matriz geradora $G_{sys}(D)$ em (5.4). Na próxima seção, será mostrado como efetuar a troca de mapeamento de modo que a treliça mínima obtida nesta etapa corresponda a um codificador sistemático recursivo dado por $G_{sys}(D)$.

5.3.2 Troca de Mapeamento da Treliça Mínima

A convenção adotada em [25] para códigos não-sistemáticos não-recursivos em que os ramos superiores das transições da treliça mínima se referem ao bit de informação 0 e os inferiores, ao bit de informação 1, de uma certa forma particulariza o codificador que, geralmente, é não-sistemático. Para exemplificar, considere o ramo tracejado conectando o estado 1 na profundidade 0 ao estado 0 na profundidade 1 na treliça mínima da Figura 5.3 Pela convenção [25], este ramo (superior) corresponde ao bit de informação 0, porém a

linha (tracejada) corresponde ao bit codificado 1. Assim, esta treliça não representa um mapeamento sistemático.

A associação dos ramos cheios/tracejados com os bits codificados não pode ser alterada, pois a treliça mínima não mais representaria o mesmo código convolucional. A nova convenção altera as k seções da treliça em que há informação $(b_t = 1)$, associando os ramos cheios destas seções com bits de informação 0 e os ramos tracejados com bits de informação 1. Esta nova convenção será denominada de mapeamento sistemático. No Exemplo 5.1, é necessário aplicar o mapeamento sistemático apenas nas três $(k = 3)$ primeiras seções da treliça mínima da Figura 5.3 para convertê-la em uma treliça sistemática.

Observe que a treliça mínima da Figura 5.3 representa o código com $G(D)$ dada em (5.8). A mesma treliça com mapeamento sistemático representa o código com $G_{sys}(D)$ dada em (5.4).

O algoritmo completo para construir a treliça mínima que corresponda corretamente a um codificador sistemático recursivo é detalhado a seguir:

1. A partir da $G_{sys}(D)$, utilize a decomposição de Smith para obter a matriz geradora não-sistemática não-recursiva básica $G_b(D)$;
2. Se $G_b(D)$ não estiver na forma LR, então aplique o algoritmo em [28] para convertê-la para a forma LR. Denote a nova matriz geradora por $G(D)$. Caso contrário, faça $G(D) \leftarrow G_b(D)$;
3. Construa o módulo de treliça mínimo para $G(D)$ de acordo com o método apresentado em [25];
4. Aplique o mapeamento sistemático no módulo de treliça mínimo obtido no passo anterior. A treliça resultante é a treliça sistemática desejada.

5.4 O Algoritmo Max-log-Map

Dado um vetor recebido, $\mathbf{r}$, o algoritmo BCJR calcula as probabilidades *a posteriori* $P(u_k = i \mid \mathbf{r})$ para formar o logaritmo da razão de verossimilhança (LLR)

$$\Lambda(u_k) = \ln\left(\frac{P(u_k = 1 \mid \mathbf{r})}{P(u_k = 0 \mid \mathbf{r})}\right). \tag{5.9}$$

No algoritmo Max-log-MAP (BCJR no domínio do logaritmo), a razão de verossimilhança em (5.9) pode ser reescrita da forma simplificada a seguir

$$\Lambda(u_k) = \max_{(s,s')\in\Sigma_k^+} \left[\tilde{\beta}_{k+1}(S) + \tilde{\gamma}_k(s,s') + \tilde{\alpha}_k(s')\right] - \max_{(s,s')\in\Sigma_k^-} \left[\tilde{\beta}_{k+1}(S) + \tilde{\gamma}_k(s,s') + \tilde{\alpha}_k(s')\right] \quad (5.10)$$

em que $\tilde{\alpha}_k(s')$ é o logaritmo da probabilidade que a sequência recebida até o tempo k seja $\mathbf{r}_{t<k}$ e que o codificador esteja no estado s' em k. $\tilde{\beta}_{k+1}(s)$ é o logaritmo da probabilidade que a sequência recebida após o tempo k seja $\mathbf{r}_{t>k}$, dado que o codificador esteja no estado s em $k+1$. $\tilde{\gamma}_k(s,s')$ é o logaritmo da probabilidade de transição para o estado s' em $k+1$ com sequência recebida $\mathbf{r}_k$, dado que o codificador esteja no estado s em k.

Os termos $\tilde{\alpha}_k(s')$ e $\tilde{\beta}_{k+1}(s)$ podem ser calculados de forma recursiva para $k = 0,...,K-1$. A recursão para $\tilde{\alpha}_{k+1}$ corresponde ao algoritmo de Viterbi operando no modo direto e a recursão para $\tilde{\beta}_k$ corresponde ao algoritmo de Viterbi no modo reverso. O termo $\tilde{\gamma}_k(s,s')$ corresponde à métrica do ramo entre os estados s e s'. Esta versão simplificada do algoritmo BCJR apresenta uma perda que depende do código usado, tipicamente 0,5 dB.

Na próxima seção, vamos analisar a complexidade de decodificação do algoritmo Max-log-MAP sobre as treliças convencional e mínima.

5.4.1 Complexidade de Decodificação do algoritmo Max-log-MAP

A complexidade de decodificação do algoritmo Max-log-MAP será determinada em termos das operações aritméticas consumidas pelo algoritmo. Neste trabalho, são consideradas operações de soma (*S*), multiplicação (*M*) e comparação (*C*). Operações com a memória não são consideradas.

Considere um módulo de treliça M com n' seções, 2^{v_t} estados na profundidade t, 2^{b_t} ramos conectando os estados entre as profundidades t e $t+1$ e l_t bits rotulando cada ramo entre as profundidades t e $t+1$, para $0 \le t \le n'-1$. A decodificação pelo algoritmo Max-log-MAP é realizada em três etapas: cálculo da métrica dos ramos, atualização das métricas direta e reversa dos estados e decisão suave. A métrica do ramo entre os estados s e s' na seção t, $\tilde{\gamma}_t(s,s')$, considerando-se o canal AWGN, é dada por

$$\tilde{\gamma}_t(s,s') = \left(\left(\sum_{j=1}^{l_t} y^j x^j(s,s') \right) + L_a(s,s') \right) \quad (5.11)$$

em que y^j é o *j-ésimo* bit recebido, x^j é o *j-ésimo* bit da palavra código do ramo da transição entre os estados s e s', e L_a é a probabilidade *a priori* desta transição. Portanto, são necessárias l_t operações (M) e l_t operações (S) por ramo na seção t. O número total de ramos é $(2^{v_t+b_t})$, logo o número total de operações por seção t para o cálculo de $\tilde{\gamma}_t(s,s')$, denotado por $T(M)_t^{\gamma}$, é dado por

$$T(M)_t^{\gamma} = l_t 2^{v_t+b_t}(M+S). \tag{5.12}$$

A atualização da métrica direta $\tilde{\alpha}_t(s')$ consiste em selecionar o caminho convergente de maior probabilidade em cada estado do módulo da treliça. As operações realizadas por esta etapa são idênticas àquelas realizadas pela etapa de ACS do algoritmo de Viterbi mostradas no Capítulo 3. Logo, o número total de operações por seção t para o cálculo de $\tilde{\alpha}_t(s')$, denotado por $T(M)_t^{\alpha}$, é dado por

$$T(M)_t^{\alpha} = 2^{v_t+b_t}(S) + [2^{v_t+b_t} - 2^{v_{t+1}}](C). \tag{5.13}$$

A atualização da métrica reversa $\tilde{\beta}_{t+1}(s)$ é similar à atualização da métrica direta $\tilde{\alpha}_t(s')$, mas invertendo-se o sentido de percurso da treliça, percorrendo-a da seção n' até a seção 1. Logo, o número de operações por seção t para o cálculo de $\tilde{\beta}_{t+1}(s)$, denotado por, $T(M)_t^{\beta}$, é dada também por

$$T(M)_t^{\beta} = 2^{v_t+b_t}(S) + [2^{v_t+b_t} - 2^{v_t}](C). \tag{5.14}$$

A última etapa da decodificação pelo algoritmo Max-log-MAP é o cálculo da decisão suave $L_u = \Lambda(u_k)$ conforme a equação (5.10). Nesta etapa, calcula-se para cada estado da seção t da treliça o somatório dos termos $\tilde{\alpha}_k(s'), \tilde{\beta}_{k+1}(s)$ e $\tilde{\gamma}_k(s)$, $k = t$ e o maior valor é selecionado. Portanto, são necessárias 2^{v_t+1} operações (S) e $(2^{v_t}-1)$ operações (C). Entretanto, este cálculo deve ser repetido para cada símbolo de informação, ou seja, para cada uma das 2^k k-tuplas possíveis. Finalmente, o valor de L_u é calculado para cada k-tupla em relação à k-tupla zero (a subtração em (5.10) será considerada como soma). Logo, o número total de operações por seção t para o cálculo de L_u, denotado por $T(M)_t^{L_a}$, é dado por

$$T(M)_t^{La} = 2^k\left[2^{v_t+1}(S) + (2^{v_t}-1)(C)\right] + (2^k-1)(S). \tag{5.15}$$

Não estamos considerando em (5.15) o cálculo da decisão de qual k-tupla é mais provável de ter sido transmitida (maior L_u). Finalmente, o número total de operações requeridas por uma seção t de um módulo de treliça M é dado por

$$T(M)_{tot}^{MAP} = T(M)_t^{\gamma} + T(M)_t^{\alpha} + T(M)_t^{\beta} + T(M)_t^{L_a}. \tag{5.16}$$

Para o módulo de treliça convencional, M_{conv}, temos $l_t = n$, $n' = 1$, $v_0 = v_1 = v$ e $b_0 = k$. Assim, obtemos o número total de operações requeridas para M_{conv}, $T(M_{conv})$, dado por

$$T(M_{conv}) = \left[n2^{k+v} \right](M) + \left[(n+4)2^{k+v} + 2^k - 1 \right](S) + \left[(3)2^{k+v} - 2^k - 2^{v+1} \right](C). \tag{5.17}$$

Para o módulo de treliça mínimo, $M_{\min}$, temos $n' = n$, $l_t = 1$, $v_t = \tilde{v}_t \ \forall t$, $b_t = \tilde{b}_t \ \forall t$ e $v_n = v_0$. Desta forma reescrevemos (5.12)-(5.16) da seguinte forma

$$T(M_{\min})_t^{\gamma} = 2^{\tilde{v}_t + \tilde{b}_t}(M + \tilde{b}_t S) \tag{5.18}$$

$$T(M_{\min})_t^{\alpha} = 2^{\tilde{v}_t + \tilde{b}_t}(S) + [2^{\tilde{v}_t + \tilde{b}_t} - 2^{\tilde{v}_{t+1}}](C) \tag{5.19}$$

$$T(M_{\min})_t^{\beta} = 2^{\tilde{v}_t + \tilde{b}_t}(S) + [2^{\tilde{v}_t + \tilde{b}_t} - 2^{\tilde{v}_t}](C) \tag{5.20}$$

$$T(M_{\min})_t^{La} = 2^{\tilde{b}_t}\left[2^{\tilde{v}_t + 1}(S) + (2^{\tilde{v}_t} - 1)(C) \right] + (2^{\tilde{b}_t} - 1)(S) \tag{5.21}$$

$$T(M_{\min})_t^{MAP} = T(M_{\min})_t^{\gamma} + T(M_{\min})_t^{\alpha} + T(M_{\min})_t^{\beta} + T(M_{\min})_t^{L_a}. \tag{5.22}$$

A complexidade total do módulo de treliça mínimo, $T(M_{\min})$, é a soma das complexidades de todas as seções deste módulo

$$T(M_{\min}) = \sum_{t=0}^{n-1} T(M_{\min})_t^{MAP}. \tag{5.23}$$

Exemplo 5.2: Considere o código $C(4,3,2)$ do Exemplo 5.1. Usando a equação (5.17) obtemos

$$T(M_{conv}) = 128(M) + 263(S) + 80(C).$$

O módulo de treliça mínimo para $C(4,3,2)$, mostrado na Figura 5.3, apresenta $\tilde{\boldsymbol{v}} = (2,2,2,3)$ e $\tilde{\boldsymbol{b}} = (1,1,1,0)$. As operações requeridas em cada uma das quatro seções deste módulo são

$$T(M_{\min})_0^{MAP} = 8(M) + 41(S) + 14(C)$$
$$T(M_{\min})_1^{MAP} = 8(M) + 41(S) + 14(C)$$
$$T(M_{\min})_2^{MAP} = 8(M) + 41(S) + 10(C)$$
$$T(M_{\min})_3^{MAP} = 8(M) + 16(S) + 4(C).$$

Logo,

$$T(M_{\min}) = \sum_{t=0}^{n-1} T(M_{\min})_t^{MAP} = 32(M) + 139(S) + 42(C).$$

O número relativo de operações requeridas pela treliça mínima em relação às operações requeridas pela treliça convencional é 25% (*M*), 53% (*C*) e 52,5% (*S*).

O mesmo procedimento adotado no Capítulo 3 deste trabalho foi realizado para determinar o custo computacional das operações (M), (C) e (S) sobre números reais. Entretanto, o processador digital de sinais de ponto fixo foi substituído por um de ponto flutuante, mais adequado à natureza dos valores a serem manipulados. Os detalhes da implementação destas operações são mostrados no Apêndice C e o resultado final das simulações é mostrado na Tabela 5.1. Portanto, a complexidade computacional do algoritmo Max-log-MAP para os módulos de treliça convencional e mínimo do código do Exemplo 5.2, denotadas por $TCCm(M_{conv})$ e $TCCm(M_{min})$, respectivamente, são

$$TCCm(M_{conv}) = 128(17) + 263(17) + 80(22) = 8407$$

$$TCCm(M_{min}) = 32(17) + 139(17) + 42(22) = 3831.$$

Neste caso, a complexidade computacional da treliça mínima é 45,57% da complexidade computacional da treliça convencional.

Tabela 5. 1 - Custo computacional das operações do algoritmo Max-log-MAP.

Operação	**Ciclos**
Multiplicação real M	17
Soma real S	17
Comparação real C	22

Exemplo 5.3: Considere a matriz geradora sistemática recursiva do código $C(5,4,3)$ em [1, Tabela IV]

$$G_{sys}(D)=\begin{pmatrix} 1 & 0 & 0 & 0 & \dfrac{1+D+D^2+D^3}{1+D+D^3} \\ 0 & 1 & 0 & 0 & \dfrac{1+D+D^2}{1+D+D^3} \\ 0 & 0 & 1 & 0 & \dfrac{1+D^2+D^3}{1+D+D^3} \\ 0 & 0 & 0 & 1 & \dfrac{1+D^3}{1+D+D^3} \end{pmatrix}.$$

A treliça mínima para este código obtida utilizando o algoritmo apresentado na Seção 5.3.2 é mostrada na Figura 5.4. Para este módulo de treliça, obtemos $\tilde{\boldsymbol{v}}=(3,4,4,4,4)$, $\tilde{\boldsymbol{b}}=(1,1,1,1,0)$ e $TC(M_{min})=32$. O número de operações requeridas pelas treliças convencional e mínima, são respectivamente

$$T(M_{conv})=640(M)+1167(S)+352(C)$$

$$T(M_{\min})=112(M)+516(S)+184(C).$$

A complexidade computacional do algoritmo Max-log-MAP para os módulos de treliça convencional e mínimo do código $C(5,4,3)$ do Exemplo 5.3 é dada por $TCCm(M_{conv})=38463$ e $TCCm(M_{min})=14724$. Portanto, a complexidade computacional da treliça mínima é 38,28% da complexidade computacional da treliça convencional.

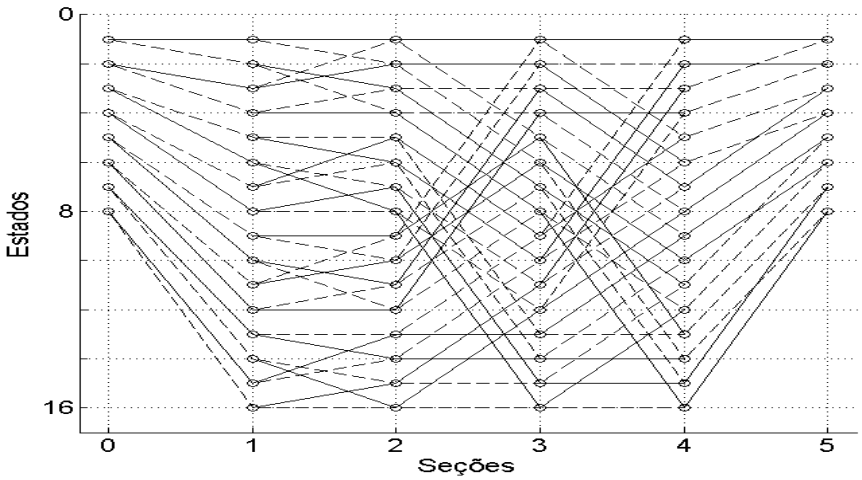


Figura 5.4 – Treliça mínima para o código convolucional $C(5,4,3)$ do Exemplo 5.3. As linhas em cheio representam bit codificado 0 e em tracejado, bit codificado 1.

5.5 Busca de bons CCCSRs em *templates*

No esquema típico de um código turbo, em que dois CCCSRs são interligados em paralelo via um entrelaçador, a probabilidade de erros de bit depende, em geral, das sequências codificadas de baixo peso geradas pelas sequências de informação de baixo peso [60]. Expressões assintóticas (alta relação sinal-ruído) para a probabilidade de erro de bit de códigos turbo com entrelaçador uniforme de comprimento elevado operando em canal AWGN revelam que o desempenho deste esquema de codificação é determinado pelo espectro (d_i, N_i) [60][61]. O parâmetro d_i é definido em [60] como o menor peso de *Hamming* de caminhos na treliça de um CCCSR originados por sequências de informação de peso i que divergem do estado 0 da treliça em um tempo fixo, por exemplo em $t = 0$, e retornam ao estado 0 pela primeira vez em algum instante de tempo futuro. A multiplicidade de caminhos com distância d_i é denominada N_i. Um termo relevante é o d_2, denominado de distância livre efetiva. Portanto, o critério de otimização para ambos os CCCSRs de um código turbo é diferente do usado para códigos convolucionais isolados, que se baseia na distância livre d_f e no espectro de distância N.

O objetivo das buscas de códigos deste trabalho é encontrar bons CCCSRs (em relação ao espectro efetivo (d_i, N_i), $i = 2,...,6)$ com complexidades intermediárias dos códigos listados em [18][62] utilizando o procedimento definido em [17]. O primeiro passo é propor *templates* a partir da matriz $G(D)$ com complexidade de treliça $TC(M_{min})$ fixa para a matriz módulo de um código. Estes *templates* devem garantir que os bits de informação estejam nas *k* primeiras seções da treliça mínima, ou seja, os elementos índices-esquerda devem estar posicionados nas *k* primeiras colunas do *template*. Definido o *template* e fixados os bits 1's dos índices-esquerda e índices-direita em suas posições específicas, os demais bits do intervalo de cobertura podem assumir livremente qualquer combinação de 0's e 1's. Cada uma destas combinações produz uma matriz $G(D)$ diferente.

Exemplo 5.4: Considere o código $C(4,3,2)$ com matriz $G_{sys}(D)$ em (5.4) e matriz equivalente $G(D)$ em (5.8) com $TC(M_{min}) = 10,67$ e $\tilde{\boldsymbol{v}} = (2,2,2,3)$, $\tilde{\boldsymbol{b}} = (1,1,1,0)$. O *template* associado à matriz módulo de $G(D)$ é dado por

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ \bar{1} & 0 & 0 & 0 \\ * & \bar{1} & 0 & 0 \\ \underline{1} & * & * & \bar{1} \\ 0 & 0 & \underline{1} & * \\ 0 & \underline{1} & * & * \end{pmatrix} \tag{5.24}$$

em que os índices-esquerda são representados por $\underline{1}$ e os índices-direita por $\bar{1}$. Observe que os três elementos índices-esquerda estão fixados nas primeiras três colunas do *template*. Os demais elementos do intervalo de cobertura são representados por asterisco.

Para cada matriz $G(D)$ produzida a partir do *template,* aplica-se o mapeamento sistemático nas seções do módulo de treliça mínimo em que há informação e calculam-se os pares (d_i, N_i) para $i = 2,...,6$. Este procedimento é realizado de forma exaustiva para cada combinação de 0's e 1's do *template.* O termo dominante é o d_2. Dentre todos os códigos selecionados com o maior d_2, seleciona-se os de menor N_2. Nos casos em que produzem o mesmo par (d_2, N_2), busca-se otimizar o par (d_3, N_3) e assim sucessivamente. Fazendo a busca exaustiva no *template* em (5.24), obtemos o código com $G(D)$ em (5.8) e $(d_2, N_2) = (3,3)$, $(d_3, N_3) = (3,3)$, $(d_4, N_4) = (4,7)$, $(d_5, N_5) = (5,15)$ e $(d_6, N_6) = (6,36)$.

Exemplo 5.5: O código $C(2,4,3)$ com aplicação prática (WiMAX), $TC(M_{min}) = 48$ e $G(D)$ dada por

$$G(D) = \begin{pmatrix} 1+D^2 & D & 1+D^2 & 1+D^2 \\ D & 1+D & 1+D & 1 \end{pmatrix} \tag{5.25}$$

apresenta $(d_2, N_2) = (7,1)$. Uma busca em *templates* com $TC(M_{min}) = 48$ revelou codificadores com $(d_2, N_2) = (10,2)$ como pode ser visto na Tabela 5.3.

O problema de enumerar os pares (d_i, N_i) para $i = 2,...,6$ é resolvido pelo método de determinação da função de transferência descrito na próxima seção.

5.5.1 Enumeração do espectro efetivo pela função de transferência $T(x, y)$

O espectro efetivo para codificadores turbo baseado no par (d_i, N_i), $i = 2,...,6$ pode ser determinado a partir da função de transferência $T(x, y)$ do codificador definida por

$$T(x,y)=\sum_{i}\sum_{j}C_{i,j}x^{i}y^{j} \tag{5.26}$$

em que $C_{i,j}$ é o número de caminhos que diverge do estado 0 em um tempo fixo e converge pela primeira vez (evento erro simples) em algum instante futuro com peso de *Hamming* da sequência de entrada igual a i e peso de *Hamming* da sequência de saída igual a j.

Uma vez obtido $T(x,y)$, o coeficiente do termo x^{i}, denotado por $[x^{i}]T(x,y)$, é uma série de potência em y cuja menor potência fornece d_i e o seu coeficiente fornece N_i. Uma técnica para calcular $T(x,y)$ a partir da matriz adjacência do codificador será discutida a seguir.

5.5.1.1 Determinação da função de transferência $T(x,y)$

O primeiro passo para determinar a função de transferência $T(x,y)$ é calcular a matriz adjacência **A**. Denota-se $[\mathbf{A}]_{i,j}$ o (i,j)-ésimo elemento de **A**. Esta matriz**,** de dimensão 2^{v} x 2^{v}, é obtida a partir do diagrama de estados do codificador. O elemento $[\mathbf{A}]_{i,j}$, é da forma $x^{p}y^{q}$ em que p e q são, respectivamente, os pesos de *Hamming* dos rótulos das sequências de informação e codificada do ramo que interliga os estados i e j do diagrama de estados, para $0\le i,j\le 2^{v}-1$. Caso haja ramos em paralelo, deve-se repetir este procedimento para cada um destes ramos e somar os resultados**.** Seja **A***, a matriz igual a **A** exceto que $[\mathbf{A}^{*}]_{0,0}=0$ e seja $\mathbf{A}'$, a matriz igual a **A** exceto que $[\mathbf{A}']_{0,j}=0,\forall j$. A matriz **A*** elimina a primeira transição da treliça do estado 0 para o estado 0, e a matriz $\mathbf{A}'$ elimina todas as transições originárias do estado 0. A função de transferência $T(x,y)$ é calculada da seguinte forma [63]

$$T(x,y)=\left[\mathbf{A}^{*}((I-\mathbf{A}')^{-1})\right]_{0,0}+[\mathbf{A}]_{0,0}-1. \tag{5.27}$$

O termo $[\mathbf{A}]_{0,0}-1$ refere-se à contribuição dos percursos paralelos (se houver) que iniciam e terminam no estado zero no diagrama de estados. I é a matriz identidade.

Exemplo 5.6: Considere o código $C(4,3,2)$ com $TC(M_{min})=10,67$ e matriz geradora sistemática recursiva $G_{sys}(D)$ dada em (5.4). A matriz básica-minima na forma LR $G(D)$ equivalente a $G_{sys}(D)$ foi determinada no Exemplo 5.1 e é dada por

$$G(D) = \begin{pmatrix} 1 & 1 & 1 & 1 \\ D & 0 & 1 & 1 \\ D & 1+D & 1 & 0 \end{pmatrix}. \tag{5.28}$$

A matriz adjacência para este código é

$$\mathbf{A} = \begin{pmatrix} 1+x^3y^4 & x^2y^2+xy^2 & x^2y^2+xy^2 & x^2y^2+xy^2 \\ x^2y^3+xy & x^2y^3+xy & x^3y^3+y & x^2y^3+xy \\ x^2y^2+xy^2 & x^3y^4+1 & x^2y^2+xy^2 & x^2y^2+xy^2 \\ x^2y^3+xy & x^2y^3+xy & x^2y^3+xy & x^3y^3+y \end{pmatrix}. \tag{5.29}$$

O elemento $[\mathbf{A}]_{1,0}$ em (5.29), ou seja, $(x^2y^3+xy),$ indica que há duas transições do estado 1 da treliça para o estado 0: a primeira é causada por uma sequência de informação de peso 2 gerando uma sequência codificada de peso 3, e a segunda é causada por uma sequência de informação de peso 1 gerando uma sequência codificada de peso 1. Observe que esta e as demais transições de estados em **A** são representadas por dois ramos paralelos entre o estado de origem e o estado de destino no módulo de treliça convencional. As matrizes **A*** e $\mathbf{A}'$ são dadas por

$$\mathbf{A}^* = \begin{pmatrix} 0 & x^2y^2+xy^2 & x^2y^2+xy^2 & x^2y^2+xy^2 \\ x^2y^3+xy & x^2y^3+xy & x^3y^3+y & x^2y^3+xy \\ x^2y^2+xy^2 & x^3y^4+1 & x^2y^2+xy^2 & x^2y^2+xy^2 \\ x^2y^3+xy & x^2y^3+xy & x^2y^3+xy & x^3y^3+y \end{pmatrix}$$

e

$$\mathbf{A}' = \begin{pmatrix} 0 & 0 & 0 & 0 \\ x^2y^3+xy & x^2y^3+xy & x^3y^3+y & x^2y^3+xy \\ x^2y^2+xy^2 & x^3y^4+1 & x^2y^2+xy^2 & x^2y^2+xy^2 \\ x^2y^3+xy & x^2y^3+xy & x^2y^3+xy & x^3y^3+y \end{pmatrix}.$$

Logo, obtemos de (5.27) a seguinte função de transferência:

$T(x,y) =$ - ((x^2*y^2 + x*y^2)*(- x^8*y^10 + x^7*y^10 + 2*x^6*y^8 + x^6*y^6 - 2*x^5*y^8 - x^5*y^6 - x^4*y^6 - 2*x^4*y^4 + x^3*y^6 + 2*x^3*y^4 + x^2*y^3 + x^2*y^2 - x*y^2 + x*y))/(- x^9*y^10 + x^7*y^10 + 2*x^7*y^8 + x^6*y^7 + x^6*y^6 - 2*x^5*y^8 - 2*x^5*y^6 - x^5*y^5 - x^4*y^5 - 3*x^4*y^4 + x^3*y^6 + x^3*y^5 + 2*x^3*y^4 + 2*x^3*y^3 + x^2*y^4 + 3*x^2*y^2 - x*y^3 + x*y - y^2 + 2*y - 1) -

((x^2*y^2 + x*y^2)*(- x^8*y^10 + x^7*y^10 + 2*x^6*y^8 + x^6*y^6 - 2*x^5*y^8 + x^5*y^7 - x^5*y^6 - x^5*y^5 - x^4*y^6 - 2*x^4*y^4 + x^3*y^6 + 2*x^3*y^4 + 2*x^2*y^2 - x*y^3 + x*y))/(- x^9*y^10 + x^7*y^10 + 2*x^7*y^8 + x^6*y^7 + x^6*y^6 - 2*x^5*y^8 - 2*x^5*y^6 - x^5*y^5 - x^4*y^5 - 3*x^4*y^4 + x^3*y^6 + x^3*y^5 + 2*x^3*y^4 + 2*x^3*y^3 + x^2*y^4 + 3*x^2*y^2 - x*y^3 + x*y - y^2 + 2*y - 1) - ((x^2*y^2 + x*y^2)*(- x^8*y^8 + x^7*y^8 + x^6*y^8 + 2*x^6*y^6 - x^5*y^8 - 3*x^5*y^6 + x^5*y^5 - x^4*y^6 - 2*x^4*y^4 + 2*x^3*y^6 - x^3*y^5 + 3*x^3*y^4 - x^3*y^3 + x^2*y^3 + x^2*y^2 - x*y^4 + x*y^3 - x*y^2 + x*y))/(- x^9*y^10 + x^7*y^10 + 2*x^7*y^8 + x^6*y^7 + x^6*y^6 - 2*x^5*y^8 - 2*x^5*y^6 - x^5*y^5 - x^4*y^5 - 3*x^4*y^4 + x^3*y^6 + x^3*y^5 + 2*x^3*y^4 + 2*x^3*y^3 + x^2*y^4 + 3*x^2*y^2 - x*y^3 + x*y - y^2 + 2*y - 1) + x^3*y^4.

A expansão de $\left[T(x,y)\right]_{0,0}$ em série de Taylor em termos dos coeficientes de *x* resulta nas seguintes expressões:

$$\left[x^2\right]\ T(x,y) = 3y^3 + 4y^4 + 5y^5 + 5y^6 + \ldots \tag{5.30}$$

$$\left[x^3\right]\ T(x,y) = 3y^3 + 11y^4 + 25y^5 + 40y^6 + \ldots \tag{5.31}$$

$$\left[x^4\right]\ T(x,y) = 7y^4 + 35y^5 + 104y^6 + 219y^7 + \ldots \tag{5.32}$$

$$\left[x^5\right]\ T(x,y) = 15y^5 + 105y^6 + 383y^7 + 998y^8 + \ldots \tag{5.33}$$

$$\left[x^6\right]\ T(x,y) = 36y^6 + 301y^7 + 1305y^8 + 4020y^9 + \ldots \tag{5.34}$$

Observando-se os termos com os menores expoentes de *y* em (5.30) - (5.34), obtemos o espectro efetivo do código $C(4,3,2)$ com matriz geradora equivalente $G(D)$ em (5.28). A Tabela 5.2 lista os pares (d_i, N_i) para $i = 2,...,6$.

Tabela 5.2 - Espectro efetivo do código $C(4,3,3)$ com matriz geradora $G(D)$ em (5.28).

(d_2, N_2)	(d_3, N_3)	(d_4, N_4)	(d_5, N_5)	(d_6, N_6)
3,3	**3,3**	**4,7**	**5,15**	**6,36**

5.5.2 Resultados das Buscas

A busca foi realizada em vários *templates* para cada $TC(M_{min})$ e o código com melhor espectro efetivo foi selecionado. Este procedimento foi repetido com várias $TC(M_{min})$ de valores intermediários aos listados em [18][62] para as taxas $R = 2/4,\ R = 3/4,\ R = 3/5$ e $R = 4/5$. A Tabela 5.3 lista a complexidade $TC(M_{min})$, o número de operações aritméticas *S*, *C* e *M,* a complexidade computacional *TCCm* do algoritmo Max-log-MAP e o espectro de distâncias dos códigos obtidos. A Tabela 5.4 lista o espectro efetivo e a matriz $G(D)$ em octal destes códigos. Por exemplo, as matrizes

$G_{sys}(D)$ listadas em [18] para $R = 3/4$ produzem $TC(M_{min}) = 10,67$ ($v = 2$), $TC(M_{min}) = 32$ ($v = 3$) e $TC(M_{min}) = 64$ ($v = 4$) e, para $R = 4/5$ produzem $TC(M_{min}) = 14$ ($v = 2$), $TC(M_{min}) = 32$ ($v = 3$) e $TC(M_{min}) = 64$ ($v = 4$). A matriz $G(D)$ mostrada na Tabela 5.4 juntamente com a aplicação do mapeamento sistemático é utilizada para construir a treliça mínima que atende aos requisitos de desempenho estabelecidos pelos pares (d_i, N_i) para $i = 2,...,6$. As Tabelas 5.3-5.4 também mostram novos códigos com complexidades de treliça intermediária aos listados em [18][62]. Estas novas complexidades, obtidas pelo procedimento de busca, produzem um refinamento em relação ao espectro efetivo. O aumento gradual da complexidade é acompanhado pela melhora da distância livre efetiva e/ou multiplicidade do código.

Simulações realizadas em [64][65] mostram que o desempenho do algoritmo Max-log-MAP sobre a treliça mínima não apresenta o mesmo desempenho do apresentado sobre a treliça convencional. Entretanto, se certos padrões de seccionamento forem aplicados à treliça mínima, é possível produzir estruturas de treliça seccionadas com o mesmo desempenho da treliça convencional.

Tabela 5.3 – Complexidade do algoritmo Max-log-MAP e espectro de distâncias de códigos obtidos para as taxas $R = 2/4, 3/4, 3/5, 4/5$.

R	$TC(M_{min})$	S	C	M	$TCCm$	d_f	N
$\frac{2}{4}$	12	98	28	24	2690	4	4,0,11,0,35
	24	194	60	48	5434	5	4,6,6,19,47
	28	210	60	56	5842	5	1,3,9,20,41
	32	226	60	64	6250	6	7,0,24,0,138
	48	386	124	96	10922	6	3,0,28,0,113
	56	418	124	112	11738	6	2,5,8,25,44
	64	450	124	128	12554	6	1,5,7,14,42
	80	610	188	160	17226	6	1,6,17,42,115
$\frac{3}{4}$	10,67*	139	42	32	3831	3	6,23,80,290,1056
	18,67	259	90	56	7335	3	3,13,64,263,1078
	21,33	275	90	64	7743	4	29,0,532,0,10146
	32*	435	154	96	12415	4	5,36,152,708,3429
	42,67	547	186	128	15567	4	3,44,160,638,3561
	53,33	707	250	160	20239	4	3,27,127,582,2833
	64*	867	314	192	24911	4	1,16,88,416
	74,67	1027	378	224	29583	4	4,23,101,552,2733
$\frac{3}{5}$	10,67	139	42	32	3831	3	3,6,12,26,53
	21,33	275	90	64	7743	4	4,11,23,73,182
	29,33	371	122	88	10487	4	1,8,18,47,127
	32	387	122	96	10895	4	1,7,14,44,126
	37,33	467	154	112	13231	4	1,4,12,35,90
	48	579	186	144	16383	5	3,12,40,100,283
	58,67	739	250	176	21055	5	3,5,23,82,195
	64	771	250	192	21871	5	4,7,18,70,205
	74,67	931	314	224	26543	5	3,7,15,48,154
$\frac{4}{5}$	14*	260	88	56	7308	2	1,9,48,240
	18	340	120	72	9644	3	10,51,244,1244
	20	356	120	80	10052	3	6,29,176,1030
	32*	596	216	128	17060	3	1,21,139,776
	48	868	312	192	24884	3	1,10,76,495,2943
	56	1028	376	224	29556	3	1,9,78,501,3096
	64*	1188	440	256	34228	4	7,64,360,2268
	72	1348	504	288	38900	4	18,0,917,0,35407

* Códigos listados em [18].

Tabela 5.4 – Espectro efetivo e matriz $G(D)$ dos códigos obtidos.

R	$TC(M_{min})$	d_2, N_2	d_3, N_3	d_4, N_4	d_5, N_5	d_6, N_6	$G(D)$
$\frac{2}{4}$	12	4,2	4,2	6,3	8,6	10,9	[2 3 0 1; 3 0 1 1]
	24	6,2	5,4	6,4	7,2	8,3	[2 3 1 1; 5 1 0 3]
	28	7,2	5,1	6,1	7,2	8,2	[2 3 3 0; 7 1 2 3]
	32	8,2	6,4	6,3	8,6	8,2	[3 2 2 3; 4 3 1 3]
	48	10,2	6,1	6,1	6,1	8,2	[7 2 3 3; 4 5 1 3]
	56	11,2	6,2	7,2	7,1	8,1	[4 3 1 3; 7 4 4 3]
	64	12,2	7,3	6,1	7,2	10,20	[7 3 1 2; 2 7 7 7]
	80	14,2	7,3	8,12	9,31	6,1	[7 4 3 3; 4 3 5 5]
$\frac{3}{4}$	10,67*	3,3	3,3	4,7	5,15	6,36	[1 1 1 1; 2 0 1 1;2 3 1 0]
	18,67	3,2	3,1	4,4	5,14	6,40	[2 0 1 1; 0 1 2 1; 3 3 1 1]
	21,33	4,4	4,15	4,10	6,215	6,86	[3 1 0 1; 2 1 3 3; 2 2 1 1]
	32*	4,1	4,3	4,1	5,7	6,19	[1 1 1 1;0 3 2 1; 6 6 1 3]
	42,67	5,2	4,2	4,1	5,9	6,23	[1 1 1 1; 6 0 1 3; 4 5 2 3]
	53,33	6,3	4,3	5,12	5,7	6,23	[2 3 2 1; 2 0 3 3; 7 1 1 1]
	64*	6,2	4,1	5,8	5,4	6,11	[1 1 1 1; 4 4 3 1; 2 5 4 3]
	74,67	7,2	4,3	4,1	5,5	6,17	[2 1 1 1; 5 7 1 0; 6 0 7 0]
$\frac{3}{5}$	10,67	3,2	3,1	5,2	7,10	8,5	[1 1 1 0 0; 0 3 0 0 1; 2 0 1 1 0]
	21,33	4,1	4,3	5,4	5,1	6,1	[0 2 1 1 1; 3 0 1 0 1; 2 1 3 1 0]
	29,33	6,3	5,8	4,1	7,33	6,1	[2 2 3 1 1; 2 3 0 3 0; 3 1 1 1 1]
	32	6,1	5,4	4,1	6,3	6,1	[0 3 2 1 1; 2 2 1 2 3; 3 1 1 0 1]
	37,33	7,2	4,1	5,2	6,2	7,3	[0 3 2 1 0; 2 2 1 1 1; 1 3 1 2 2]
	48	8,3	5,1	5,2	6,2	6,1	[4 0 3 1 3; 3 2 3 2 1; 2 3 1 1 0]
	58,67	8,2	5,2	5,1	6,1	7,2	[6 3 2 3 3; 1 3 3 0 1; 2 2 1 3 1]
	64	10,3	5,2	5,2	6,3	7,2	[6 2 3 3 3; 1 0 3 2 3; 0 3 2 1 1]
	74,67	10,2	5,1	5,2	6,2	8,10	[2 2 3 2 3; 7 6 2 3 1; 2 1 2 1 1]
$\frac{4}{5}$	14*	2,1	3,5	4,17	5,65	6,236	[1 1 0 0 1; 2 0 1 1 0; 0 1 0 1 0;2 0 2 1 1]
	18	3,4	3,6	4,23	5,80	6,284	[3 1 0 0 1; 0 2 1 1 1; 0 1 1 1 0;2 0 2 1 1]
	20	3,2	3,4	4,11	5,45	6,220	[2 0 1 1 0; 2 2 0 1 1; 0 1 1 1 1; 3 1 3 1 1]
	32*	4,2	3,1	4,8	5,42	6,179	[2 0 3 1 0; 2 2 0 3 1; 1 0 1 1 1;0 3 0 1 0]
	48	4,1	3,1	4,4	5,22	6,105	[2 1 1 1 1; 1 1 3 3 0; 2 2 0 1 1; 0 2 1 2 3]
	56	5,2	3,1	4,3	5,24	6,129	[2 1 1 1 1; 3 2 3 1 1; 0 2 1 3 0; 2 0 0 3 3]
	64*	5,2	4,4	4,3	5,14	6,86	[1 0 1 1 1; 0 2 0 3 1; 2 1 3 1 1;6 6 3 3 2]
	72	6,4	4,10	4,8	6,392	6,227	[2 0 1 1 1; 1 0 3 0 1; 2 3 1 3 0; 0 4 2 3 3]

* Códigos listados em [18].

5.6 Conclusões

Neste capítulo, foram apresentados os fundamentos de códigos e codificadores convolucionais sistemáticos recursivos utilizados em esquemas turbo. Um método para construção da treliça mínima para estes codificadores foi introduzido e a complexidade de decodificação do algoritmo Max-log-MAP foi analisada. A principal contribuição deste trabalho é a obtenção de novos codificadores sistemáticos recursivos de taxas $2/4$, $3/4$, $3/5$ e $4/5$, com certo grau de refinamento na relação complexidade versus distância livre que atendam às necessidades de aplicações práticas.

CAPÍTULO 6

CONCLUSÕES E TRABALHOS FUTUROS

Nesta tese, foi definida uma nova medida de complexidade computacional de decodificação de códigos convolucionais utilizando o VA operando com decisão abrupta. A complexidade computacional é função das complexidades de treliça e comparativa, e foi definida com base no número de ciclos de máquina consumidos pela decodificação. Isto foi alcançado determinando-se o número de operações aritméticas e seus respectivos custos computacionais de execução baseado numa plataforma de processadores digitais de sinais de ponto-fixo para módulos de treliça convencional e mínimo.

As complexidades de treliça, comparativa e computacional de códigos de várias taxas foram calculadas. No universo de códigos de mesma taxa, aqueles que apresentam a mesma complexidade de treliça podem apresentar complexidades comparativas diferentes e, consequentemente, complexidades computacionais diferentes. Uma busca de códigos baseada nas complexidades de treliça e comparativa foi conduzida para as taxas 2/4, 3/5, 4/7 e 5/7. A complexidade computacional proposta no Capítulo 3 é uma medida mais fiel em termos de esforço computacional requerido pela decodificação implementada por software.

A técnica de seccionamento do módulo de treliça mínimo para códigos convolucionais foi apresentada. Um conjunto de regras de seccionamento foi construído para auxiliar na determinação do melhor padrão de seccionamento dentre as 2^{n-1}

possibilidades que minimize uma certa medida de complexidade. Foram identificados padrões de seccionamento que resultam em módulos de treliça mais compactos e regulares que apresentam redução do número máximo de estados e do número total de seções em relação ao módulo mínimo, mantendo-se as complexidades deste módulo. Um compromisso foi estabelecido entre as quatro medidas de complexidade consideradas neste trabalho: complexidade de treliça, complexidade comparativa, número máximo de estados e número total de seções. Tabelas foram apresentadas com padrões de seccionamento de treliças mais compactas e de mesma complexidade da treliça mínima, para códigos de diversas taxas. As topologias de treliças seccionadas propostas neste trabalho podem simplificar a implementação de decodificadores em hardware e são uma alternativa interessante sobre o módulo de treliça mínimo. Além disso, a variedade de novas topologias obtidas pelo seccionamento pode resultar em treliças com menor complexidade de decodificação a serem usadas por outros algoritmos de decodificação, como SOVA, BCJR e Algoritmo-M. Isto pode reduzir o consumo de energia total do receptor, um assunto de relevância na literatura atual.

Na abordagem de esquemas turbo, foi apresentado um método para construção da treliça mínima para codificadores convolucionais sistemáticos recursivos utilizados por estes esquemas. A complexidade de decodificação do algoritmo Max-log-MAP foi analisada. Uma outra contribuição desta etapa do trabalho foi a obtenção de novos codificadores sistemáticos recursivos de taxas $2/4$, $3/4$, $3/5$ e $4/5$, com certo grau de refinamento na relação complexidade/distância livre, que atendem às necessidades de aplicações práticas.

6.1 Sugestões de Trabalhos Futuros

Esta tese apresentou uma medida de complexidade computacional do VA mais adequada para o receptor implementado em software e os efeitos do seccionamento do módulo de treliça mínimo sobre esta complexidade. Como trabalho futuro, poderíamos sugerir a análise do impacto do seccionamento sobre o VA quando este for implementado numa plataforma FPGA em hardware. Desta forma, poderíamos avaliar na prática se realmente a redução do número máximo de estados e/ou número total de seções de um

módulo de treliça seccionado influi na complexidade de decodificação. Análise similar poderia ser realizada para o algoritmo Max-log-Map.

Uma plataforma de arquitetura paralela poderia ser proposta para definição de uma medida de complexidade computacional em que processadores distintos executassem as operações do VA.

A representação mais simples de treliça foi a principal abordagem deste trabalho para análise da complexidade de decodificação. Outra abordagem discutida na literatura para reduzir a complexidade é a utilização de algoritmos sub-ótimos, como por exemplo, o algoritmo-M. Geralmente, estes algoritmos são propostos com base na representação convencional de treliça de um código, bem como na treliça mínima [5]. Seria interessante combinar a operação de um algoritmo sub-ótimo sobre uma treliça seccionada e avaliar o impacto deste arranjo na complexidade de decodificação.

A combinação das técnicas de seccionamento e supressão de ramos [26] também poderia ser considerada para análise de seu efeito sobre a complexidade de decodificação. Uma busca de códigos baseada em treliças resultantes dessas duas técnicas poderia ser conduzida.

Estudar o compromisso complexidade versus distância livre efetiva para sistemas turbo com códigos componentes com diferentes representações de treliças e operando com diversos algoritmos de decodificação, tais como, log-MAP e SOVA.

APÊNDICE A

CONSTRUÇÃO DA TRELIÇA MÍNIMA

Seja a matriz $\hat{G}$ associada a uma matriz $G_{escalar}$ na forma LR. Os índices das linhas de $\hat{G}$ são indexados por $1,...,k(m+1)$. Seja R_j o conjunto formado pelos índices das linhas de $\hat{G}$ que têm elementos ativos na coordenada j, $j=0,...,n+1$, e seja Q_j o conjunto formado pelos índices das linhas de $\hat{G}$ que têm elementos ativos simultaneamente nas coordenadas j e $j+1$, $j=0,...,n$, ou seja, $Q_j = R_j \cap R_{j+1}$. No cálculo de R_j, para $j=0$ são considerados os elementos ativos em $j=1$ que não sejam índices-esquerdas de $\hat{G}$ e, para $j=n+1$ são considerados os elementos ativos em $j=n$ que não sejam índices-direitas de $\hat{G}$. As cardinalidades de R_j e Q_j são ω_j e ε_j, respectivamente.

Exemplo A.1: Considere a matriz $\hat{G}$ do Exemplo 2.9

$$\hat{G} = \begin{pmatrix} \mathbf{1} & \bar{\mathbf{1}} & 0 \\ \mathbf{1} & \mathbf{0} & \bar{\mathbf{1}} \\ \underline{\mathbf{1}} & \mathbf{1} & \mathbf{1} \\ 0 & 0 & \underline{\mathbf{1}} \end{pmatrix}.$$

Neste código, temos $n=3$, portanto a treliça mínima possui 3 seções distintas e profundidades 0 a 3. A Tabela A.1 mostra os $R_j's$, $Q_j's$, $\omega_j's$ e $\varepsilon_j's$ para $\hat{G}$ do Exemplo A.1.

Tabela A.1 - R_j's , Q_j's , ω_j's e ε_j's da matriz $\hat{G}$ do Exemplo A.1.

j	R_j	Q_j	ω_j	ε_j
0	{1,2}	{1,2}	2	2
1	{1,2,3}	{1,2,3}	3	3
2	{1,2,3}	{2,3}	3	2
3	{2,3,4}	{3,4}	3	2
4	{3,4}	{∅}	2	0

O número de estados é 2^{ε_j}, $j = 0,...,n-1$. Cada estado na profundidade j está associado a uma sequência binária de comprimento ε_j. Para conectar os ramos entre as profundidades $j-1$ e j, $j = 1,...,n$, define-se um vetor binário $\boldsymbol{\mu}_j = (\mu_{l_1},...,\mu_{l_{\alpha_j}})$ de comprimento ω_j onde cada $l_j \in R_j$. Os estados iniciais e finais são vetores binários de comprimento ε_{j-1} e ε_j respectivamente, tais que

$$init(\boldsymbol{\mu}_j) = \boldsymbol{\mu}_j \cap \boldsymbol{Q}_{j-1} \qquad j = 1,...,n$$
$$fim(\boldsymbol{\mu}_j) = \boldsymbol{\mu}_j \cap \boldsymbol{Q}_j.$$

Em que a notação $\boldsymbol{\mu}_j \cap \boldsymbol{Q}_j$ é um vetor de comprimento ε_j formado pelos componentes de $\boldsymbol{\mu}_j$ que correspondem aos índices em $\boldsymbol{Q}_j$. Por exemplo, se $\boldsymbol{\mu}_1 = (\mu_1, \mu_2, \mu_3)$, $Q_0 = \{1,2\}$ e $Q_1 = \{1,2,3\}$ então $init(\boldsymbol{\mu}_1) = (\mu_1, \mu_2)$ e $fim(\boldsymbol{\mu}_1) = (\mu_1, \mu_2, \mu_3)$. Para obter o símbolo codificado, calculamos

$$c(\boldsymbol{\mu}_j) = \boldsymbol{\mu}_j \bullet (g_j \cap R_j)$$

em que g_j é a j- ésima coluna da matriz $\hat{G}$, $j = 1,...,n$. Realiza-se o produto interno de $\boldsymbol{\mu}_j$ com as linhas de g_j que têm elementos ativos na coordenada j.

Exemplo A.2: Considere a matriz $\hat{G}$ e a Tabela A.1 do Exemplo A.1. A definição das conexões dos ramos entre as profundidades 0-1, 1-2 e 2-3 e respectivos símbolos codificados é mostrada a seguir.

- $j = 1$, profundidade 0-1

$$\begin{aligned}
\boldsymbol{\mu}_1 &= (\mu_1, \mu_2, \mu_3) \\
init(\boldsymbol{\mu}_1) &= \boldsymbol{\mu}_1 \cap \boldsymbol{Q}_0 \\
&= (\mu_1, \mu_2, \mu_3) \cap \{1, 2\} \\
&= (\mu_1, \mu_2) \\
fim(\boldsymbol{\mu}_1) &= \mu_1 \cap \boldsymbol{Q}_1 \\
&= (\mu_1, \mu_2, \mu_3) \cap \{1, 2, 3\} \\
&= (\mu_1, \mu_2, \mu_3) \\
c(\boldsymbol{\mu}_1) &= \boldsymbol{\mu}_1 \bullet (1, 1, 1) \\
&= \mu_1 \oplus \mu_2 \oplus \mu_3.
\end{aligned}$$

- $j = 2$, profundidade 1-2

$$\begin{aligned}
\boldsymbol{\mu}_2 &= (\mu_1, \mu_2, \mu_3) \\
init(\boldsymbol{\mu}_2) &= \boldsymbol{\mu}_2 \cap \boldsymbol{Q}_1 \\
&= (\mu_1, \mu_2, \mu_3) \cap \{1, 2, 3\} \\
&= (\mu_1, \mu_2, \mu_3) \\
fim(\boldsymbol{\mu}_2) &= \mu_2 \cap \boldsymbol{Q}_2 \\
&= (\mu_1, \mu_2, \mu_3) \cap \{2, 3\} \\
&= (\mu_2, \mu_3) \\
c(\boldsymbol{\mu}_2) &= \boldsymbol{\mu}_2 \bullet (1, 0, 1) \\
&= \mu_1 \oplus \mu_3.
\end{aligned}$$

- $j = 3$, profundidade 2-3

$$\begin{aligned}
\boldsymbol{\mu}_3 &= (\mu_2, \mu_3, \mu_4) \\
init(\boldsymbol{\mu}_3) &= \boldsymbol{\mu}_3 \cap \boldsymbol{Q}_2 \\
&= (\mu_2, \mu_3, \mu_4) \cap \{2, 3\} \\
&= (\mu_2, \mu_3) \\
fim(\boldsymbol{\mu}_3) &= \mu_3 \cap \boldsymbol{Q}_3 \\
&= (\mu_2, \mu_3, \mu_4) \cap \{3, 4\} \\
&= (\mu_3, \mu_4) \\
c(\boldsymbol{\mu}_3) &= \boldsymbol{\mu}_3 \bullet (1, 1, 1) \\
&= \mu_2 \oplus \mu_3 \oplus \mu_4.
\end{aligned}$$

As Tabelas A.2-A.4 listam os estados iniciais e finais dos ramos que conectam as profundidades 0-1, 1-2 e 2-3 da treliça mínima, respectivamente, bem como os respectivos símbolos codificados. As Figuras A.1-A.3 mostram as conexões dos ramos entre estas profundidades e a Figura A.4 mostra a treliça mínima resultante.

Tabela A.2 - Conexões entre profundidades 0 e 1.

μ_1	Estado inicial $init(\mu_1)$	Estado Final $fim(\mu_1)$	Símbolo $c(\mu_1)$
000	00	000	0
001	00	001	1
010	01	010	1
011	01	011	0
100	10	100	1
101	10	101	0
110	11	110	0
111	11	111	1

Tabela A.3 - Conexões entre profundidades 1 e 2.

μ_2	Estado inicial $init(\mu_2)$	Estado Final $fim(\mu_2)$	Símbolo $c(\mu_2)$
000	000	00	0
001	001	01	1
010	010	10	0
011	011	11	1
100	100	00	1
101	101	01	0
110	110	10	1
111	111	11	0

Tabela A.4 - Conexões entre profundidades 2 e 3.

μ_3	Estado inicial $init(\mu_3)$	Estado Final $fim(\mu_3)$	Símbolo $c(\mu_3)$
000	00	00	0
001	00	01	1
010	01	10	1
011	01	11	0
100	10	00	1
101	10	01	0
110	11	10	0
111	11	11	1

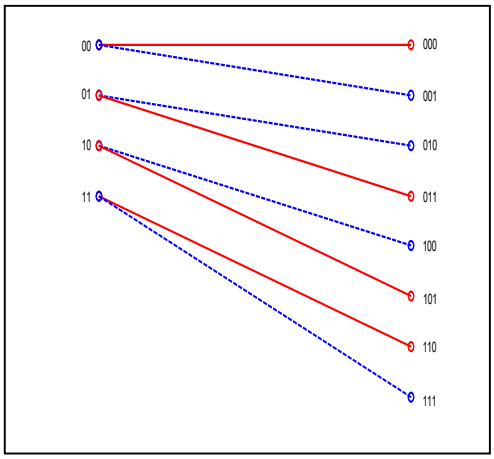


Figura A.1 – Conexões entre as profundidades 0-1 obtidas a partir da Tabela A.2. As linhas em cheio representam bit codificado 0 e em tracejado, bit codificado 1.

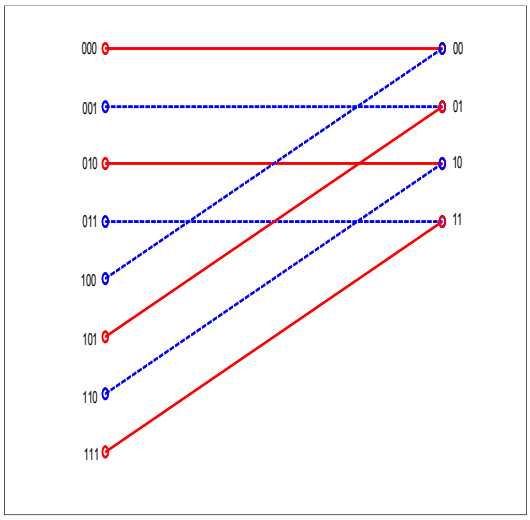


Figura A.2 – Conexões entre as profundidades 1-2 obtidas a partir da Tabela A.3. As linhas em cheio representam bit codificado 0 e em tracejado, bit codificado 1.

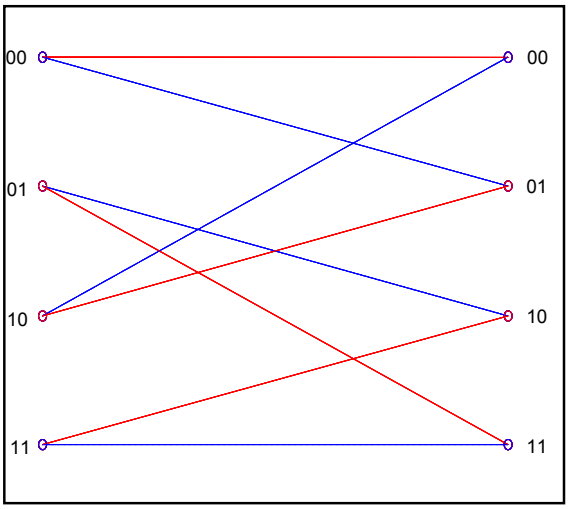


Figura A.3 - Conexões entre as profundidades 2-3 obtidas a partir da Tabela A.4. As linhas em cheio representam bit codificado 0 e em tracejado, bit codificado 1.

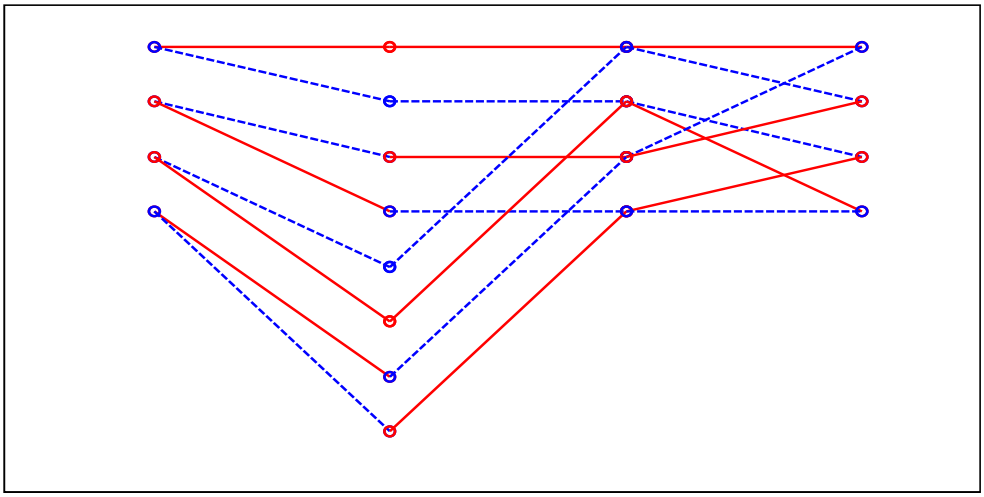


Figura A.4 – Módulo da treliça mínima resultante com as três seções interconectadas.

APÊNDICE B

ANÁLISE DA COMPLEXIDADE COMPUTACIONAL DO VA PARA DECISÃO SUAVE

A decodificação usando o VA para decisão suave é realizada com três componentes: o calculador de métrica acumulada (AMC, do inglês *acumulated metric calculator*), o comparar-armazenar (CS, do inglês *compare-store*) e o *RAM Traceback.* A Figura B.1 ilustra o diagrama em blocos do VA para decisão suave.

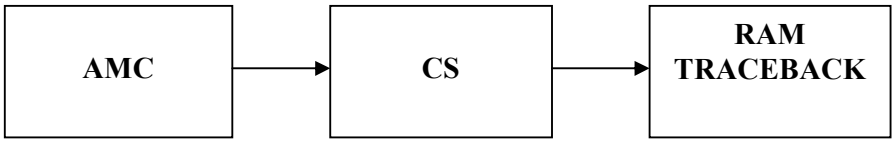


Figura B.1 – Diagrama em blocos do VA para decisão suave.

As operações requeridas pelo AMC e CS sobre um módulo de treliça para o VA operando com decisão suave serão analisadas nas subseções a seguir. O *RAM Traceback* não envolve operações de soma nem comparações, portanto este não será considerado nesse trabalho.

B.1 Operações do AMC

A etapa de AMC consiste em calcular o incremento de métrica entre a palavra recebida e a palavra-código em cada ramo de uma seção t do módulo de treliça M e somá-lo à métrica

de seu estado inicial. O resultado desta soma é a métrica acumulada do ramo. A palavra recebida é uma sequência de símbolos (valores reais) que passa por um processo de quantização em q níveis. O resultado da quantização de um símbolo é o valor inteiro p entre 0 e q-1. A palavra-código é formada pelos símbolos que rotulam cada ramo de M. Os valores dos incrementos de métrica são valores inteiros pré-definidos e armazenados na tabela de métricas **T** para cada nível de quantização e símbolo da treliça. Para um código binário e q níveis de quantização, a tabela **T** é formada por 2 linhas e q colunas. Por exemplo, se o símbolo de um ramo da seção t da treliça é igual a 0 e o resultado da quantização do símbolo recebido é igual a 3, então o elemento $T_{0,3}$ é o incremento de métrica do ramo que é somado à métrica de seu estado inicial na seção t-1. O acesso à tabela **T** juntamente com a operação de soma de métricas é chamado de soma indexada. Como cada ramo é rotulado com l_t bits, são necessárias l_t somas indexadas. O número total de ramos no módulo é dado por $2^{v_t+b_t}$, portanto são necessárias $(l_t)2^{v_t+b_t}$ operações de soma indexada. O cálculo detalhado do AMC para uma seção t de um módulo de treliça M é mostrado na Tabela B.1. Definindo o número de operações de soma indexada por S_x, concluímos da Tabela B.1 que o número total de operações por seção do AMC, T_t^{AMC}, é dado por:

$$T_t^{AMC} = (l_t)2^{v_t+b_t}(S_x). \tag{B.1}$$

Tabela B.1– Operações do AMC sobre uma seção t do módulo de treliça M.

Operações por ramo:	l_t somas*
Número de ramos:	$2^{v_t+b_t}$
Total de operações do AMC (T_t^{AMC}):	$(l_t)2^{v_t+b_t}$ somas

***** A operação de soma indexada engloba o acesso à **T** e a operação de soma propriamente dita.

B.2 Operações do CS

A etapa de CS consiste em comparar as métricas acumuladas dos ramos que convergem para cada um dos estados da seção $t+1$ e selecionar a menor. O resultado é a métrica vencedora do estado e é armazenada na memória. Há $2^{v_{t+1}}$ estados na seção $t+1$ e $2^{v_t+b_t}$ ramos entre as seções t e $t+1$, portanto são comparados $2^{v_t+b_t}/2^{v_{t+1}}$ ramos por estado, que utilizam $(2^{v_t+b_t}/2^{v_{t+1}})-1$ operações de comparação. Considerando todos os estados,

teremos $[(2^{v_t+b_t} / 2^{v_{t+1}})-1]2^{v_{t+1}}$, ou ainda $2^{v_t+b_t}-2^{v_{t+1}}$ operações de comparação no total. Essas operações de comparação são realizadas sobre valores inteiros. O cálculo detalhado do CS para uma seção t de um módulo de treliça M é mostrado na Tabela B.2. Definindo o número de comparações de valores inteiros por C_i, concluímos da Tabela B.2 que o número total de operações por seção do CS, T_t^{CS}, é dado por:

$$T_t^{CS} = [2^{v_t+b_t} - 2^{v_{t+1}}](C_i). \tag{B.2}$$

Tabela B.2 – Operações do CS sobre uma seção t do módulo de treliça M.

Operação por estado convergente:	$(2^{v_t+b_t} / 2^{v_{t+1}})-1$ comparações de valores inteiros
Número de estados convergentes:	$2^{v_{t+1}}$
Total de operações do CS (T_t^{CS}):	$2^{v_t+b_t}-2^{v_{t+1}}$ comparações de valores inteiros

A partir de (B.1) e (B.2), definimos o número total de operações do VA para decisão suave por bit de informação de um módulo de treliça M por:

$$\begin{aligned} T(M) &= \frac{1}{k}\sum_{t=0}^{n'-1}(T_t^{AMC} + T_t^{CS}) \\ T(M) &= \frac{1}{k}\sum_{t=0}^{n'-1}\left\{l_t 2^{v_t+b_t}[S_x] + (2^{v_t+b_t} - 2^{v_{t+1}})[C_i]\right\}. \end{aligned} \tag{B.3}$$

Podemos reescrever (B.3) usando-se (3.4) e (3.5) da seguinte forma:

$$T(M) = TC(M)\left[(S_x)\right] + MC(M)\left[C_i\right]. \tag{B.4}$$

B.3 Custo Computacional do VA para Decisão Suave

O custo computacional do VA para decisão suave foi determinado computando-se o custo computacional isolado das operações S_x e C_i baseado em simulações utilizando-se a família de processadores digitais de sinais de ponto fixo 320TMSC55xx da Texas Instruments. O ambiente de desenvolvimento integrado (IDE) Code Composer Studio (CCStudio) versão 4.1.1.00014 e o simulador integrado *C55xx Rev2.x CPU Cycle Accurate Simulator* foram utilizados para editar, compilar e simular as operações. O custo computacional das operações S_x e C_i foi calculado em termos do número de ciclos de máquina consumidos pela sua execução. As implementações foram realizadas em

linguagem C. Nas próximas seções, são mostrados os detalhes de implementação destas operações.

B.3.1 Implementação da Operação S_x

A Tabela B.3 mostra os detalhes da implementação da operação S_x para cada símbolo recebido. Na primeira coluna aparece a operação soma indexada em que o valor atual da variável S_x representa a métrica do estado inicial, o termo T[b][p] realiza o acesso indexado na tabela de métricas **T**, implementada com uma matriz bidimensional cujos índices *b* e *p* são o símbolo da treliça e o resultado da quantização do símbolo recebido, respectivamente. O resultado da soma S_x é a métrica acumulada do ramo. Na segunda coluna é mostrado o código em linguagem *Assembly* do TMS320CC55x gerado pelo compilador; na terceira, uma breve descrição do código; na quarta o número de ciclos de máquina consumidos por cada instrução. No total são necessárias 7 instruções e 7 ciclos de máquina para implementar a operação S_x.

Tabela B.3 – Detalhes da implementação da operação S_x.

Implementação em C	Assembly C55x	Descrição	Ciclos
Sx = Sx + T[b][p];	MOV *SP(#00h), T0[a]	T0 ← p	1
	AMAR *SP(#02h),XAR3 || MOV AR3,AC0[b]	AC0 ← &T	1
	MACMK *SP(#01h),#5,AC0,AC0[c]	AC0 ← AC0+(b*q)	1
	MOV AC0,AR3[d]	AR3 ← AC0	1
	MOV *AR3(T0),AR1[e]	AR1 ←T[b][p]	1
	ADD *SP(#0Ch),AR1,AR1[f]	AR1 ← AR1+Sx	1
	MOV AR1,*SP(#0Ch)[g]	Sx ←AR1	1
		Total	7

[a] Leitura do resultado da quantização *p* (índice de acesso à tabela **T).**

[b] Leitura do endereço base de **T.**

[c,d] Ajuste do endereço base de **T** conforme o símbolo *b* recebido.

[e] Leitura do valor referente à métrica do ramo na tabela **T** (acesso indexado com o índice *p*).

[f] Soma da métrica do ramo com a métrica do estado inicial S_x.

[g] Armazenamento da métrica acumulado do ramo S_x.

B.3.2 Implementação da Operação C_i

A operação C_i foi implementada com uma instrução de seleção composta *if* incluindo o armazenamento do valor da menor métrica acumulada. A Tabela B.4 mostra os detalhes da implementação da operação C_i. Na primeira coluna aparece a instrução *if* que compara duas métricas acumuladas, a menor é armazenada na variável de tipo inteiro *menor*. Na segunda coluna, o código em linguagem *Assembly* do TMS320CC55x gerado pelo compilador; na terceira, uma breve descrição do código explicada pelos seguintes passos: AR1 e AR2 são registradores acumuladores que recebem os valores das métricas acumuladas, aqui representadas pelas variáveis A e B, respectivamente. A seguir as métricas são comparadas, se B < A então o bit de *status* TC1 é setado e o fluxo do programa é desviado para o rótulo @L1 e o valor de B é armazenado em *menor*. Seguindo esse caminho, o código consome 10 (=1+1+1+6+1) ciclos de máquina. Caso A < B o código segue armazenando o valor de A em *menor* e desviando o fluxo do programa para o rótulo @L2 onde está a próxima instrução a ser executada. Um detalhe da arquitetura é que o valor em AR2 não pode ser diretamente transferido para a memória, precisando ser copiado primeiro para AR1 e depois então para a memória. Seguindo esse caminho, o código consome 16 (=1+1+1+5+1+1+6) ciclos de máquina. Considerando a média dos valores, o custo computacional da operação C_i é de 13 ciclos de máquina.

Tabela B.4 – Detalhes da implementação da operação C_i.

Implementação em C	**Assembly C55x**	**Descrição**	**Ciclos**	
If (A < B) menor = A; else menor = B;	MOV *SP(#01h),AR1	AR1 ← B	1	1
	MOV *SP(#00h),AR2	AR2 ← A	1	1
	CMP AR2>=AR1, TC1	TC1 ← (AR2>=AR1)	1	1
	BCC @L1,TC1	se (TC1) vá para @L1	6	5
	MOV AR2,AR1	AR1 ← AR2	--	1
	MOV AR1, *SP(#02h)	menor ← AR1	--	1
	B @L2	vá para @L2	--	6
	@L1: MOV AR1, *SP(02h)	@L1: menor ← AR1	1	--
	@L2: ... (próxima instrução)	@L2: ... (próxima instrução)	--	--
		Total	10 ou 16	

Em resumo, o custo computacional das operações do VA para decisão suave é mostrado na Tabela B.5.

Tabela B.5 - Custo computacional das operações do VA para decisão suave.

Operação	Ciclos
Soma indexada S_x	7
Comparação de inteiro C_i	13

B.4 Complexidade Computacional do VA para Decisão Suave

Substituindo os resultados apresentados na Tabela B.5 em (B.4) definimos uma complexidade computacional para decisão suave, denotada por $TCC'(M)$, de um módulo de treliça M por:

$$TCC'(M) = TC(M)[7] + MC(M)[13]. \tag{B.5}$$

APÊNDICE C

IMPLEMENTAÇÃO DAS OPERAÇÕES S_r, M_r E C_r PARA OPERANDOS REAIS

Neste apêndice, apresentamos a implementação das operações S_r (soma real), M_r (multiplicação real) e C_r (comparação real) com base na família de processadores digitais de sinais de ponto flutuante 320TMSC67xx da Texas Instruments. O ambiente de desenvolvimento integrado (IDE) Code Composer Studio (CCStudio) versão 4.1.1.00014 e o simulador integrado *C67xx CPU Cycle Accurate Simulator* foram utilizados para editar, compilar e simular as operações. Nesta implementação, foram considerados valores de precisão simples de 32 bits, ou seja, todos os operandos são números absolutos compreendidos entre $1{,}17549435.10^{-38}$ e $3{,}40282347.10^{38}$. As operações S_r, M_r e C_r são utilizadas nos Exemplos 5.2 e 5.3 no cálculo da complexidade computacional do algoritmo Max-log-MAP sobre as treliças convencional e mínima.

C.1 Operação Soma real (S_r)

A Tabela C.1 mostra os detalhes da implementação da operação S_r. A operação S_r carrega os operandos da operação soma real da pilha para os registradores de 32 bits A3 e B5 e executa a soma de precisão simples. O resultado da operação é armazenado na pilha. No total, são necessários 17 ciclos de máquina para realizar esta operação.

Tabela C.1– Detalhes da implementação da operação S_r.

Implementação em C	Assembly C55x	Descrição	Ciclos
$S_r = a + b;$	LDW *+SP[1],A3	A3 ← *a*	5
	LDW *+SP[2], B5	B5 ← *b*	5
	ADDSP B5,A3,B4	B4 ← A3+B5	4
	STW B4, *+SP[3]	S_r ← B4	3
		Total	17

C.2 Operação Multiplicação real (M_r)

A Tabela C.2 mostra os detalhes da implementação da operação M_r. Esta operação também utiliza os registradores de 32 bits A3 e B5 para armazenar os operandos e executa a multiplicação de precisão simples. O resultado é armazenado na pilha. A operação é realizada também com 17 ciclos de máquina.

Tabela C.2– Detalhes da implementação da operação M_r.

Implementação em C	Assembly C55x	Descrição	Ciclos
$M_r = a*b;$	LDW *+SP[1],A3	A3 ← *a*	5
	LDW *+SP[2], B5	B5 ← *b*	5
	MPYSP B5,A3,B4	B4 ← A3*B5	4
	STW B4, *+SP[3]	M_r ← B4	3
		Total	17

C.3 Operação Comparação real (C_r)

A operação C_r compara dois operandos reais de precisão simples e armazena aquele de menor valor. Esta operação foi implementada com uma instrução de seleção composta *if*. A Tabela C.3 mostra os detalhes desta implementação. Na primeira coluna aparece a instrução *if* que compara duas métricas acumuladas, a menor é armazenada na variável de tipo real *menor*. Na segunda coluna, o código em linguagem *Assembly* do C67xx gerado pelo compilador; na terceira, uma breve descrição do código explicada pelos seguintes

passos: B5 e B4 são registradores de 32 bits que são carregados com os valores das métricas acumuladas, aqui representadas pelas variáveis de tipo real A e B, respectivamente. A seguir, as métricas são comparadas; se A < B, então o bit de *status* B0 é setado e o fluxo do programa segue armazenando o valor de A em *menor,* desviando o fluxo do programa para o rótulo @L2 onde está a próxima instrução a ser executada. Seguindo esse caminho, o código consome 24 (=5+5+1+5+3+5) ciclos de máquina. Caso contrário, o fluxo do programa é desviado para o rótulo @L1 e o valor de B é armazenado em *menor*. Seguindo esse caminho, o código consome 19 (=5+5+1+5+3) ciclos de máquina.

Tabela C.3 – Detalhes da implementação da operação C_i .

Implementação em C	**Assembly C55x**	**Descrição**	**Ciclos**	
if (A < B) menor = A; *else menor = B;*	LDW *+SP[1], B5	B5 ← A	5	5
	LDW *+SP[2], B4	B4 ← B	5	5
	CMPLTSP B5, B4, B0	B0 ← (B5<B4)	1	1
	B @L1	se (!B0) vá para @L1	5	5
	STW B5, *+SP[3]	menor ← B5	3	--
	B @L2	vá para @L2	5	--
	@L1: STW B4, *+SP[3]	@L1: menor ← B4	--	3
	@L2: ... (próxima instrução)	@L2: ... (próxima instrução)	--	--
		Total	24 ou 19	

Levamos em consideração o valor médio consumido pela operação, ou seja, 22 ciclos de máquina, justificado pela aleatoriedade dos valores. Finalmente, na quarta coluna da Tabela C.3, aparecem os ciclos de máquina por instrução. Em resumo, o custo computacional das operações S_r, M_r e C_r para operandos reais no cálculo da complexidade computacional do algoritmo Max-log-MAP sobre as treliças convencional e mínima é mostrado na Tabela C.4.

Tabela C.4 - Custo computacional das operações do algoritmo Max-log-MAP.

Operação	**Ciclos**
Soma real S_r	17
Multiplicação real M_r	17
Comparação real C_r	22*

* Valor médio obtido.

APÊNDICE D

PUBLICAÇÕES

Publicações em revistas:

1. I. Benchimol, C. Pimentel, R. Souza, B. Uchôa-Filho, "High-rate systematic recursive convolutional encoders: minimal trellis and code search", EURASIP Journal on Advances in Signal Processing, vol. 2012:243, pp. 1-10, 2012. Doi:10.1186/1687-6180-2012-243.
2. I. Benchimol, C. Pimentel, R. Souza, "Low complexity trellis representations of convolutional codes via sectionalization of the minimal trellis", Telecommunication Systems, 2013. [submetido]
3. G. Moritz, R. Souza, I. Benchimol, C. Pimentel, M. Pellenz, B. Uchôa-Filho", Turbo decoding using the sectionalized minimal trellis of the contituent code: performance-complexity trade-off". [submetido]

Publicações em congressos:

1. I. Benchimol, C. Pimentel, R. Souza, B. Uchôa-Filho, "Complexidade computacional de módulos de treliça de códigos convolucionais", in *XXIX Simpósio Brasileiro de Telecomunicações*, Curitiba, Brasil, 2011, pp. 1-5.
2. C. Pimentel, R. Souza, B. Uchôa-Filho, I. Benchimol, "Minimal trellis for systematic recursive convolutional encoders", in *Int. Symp. Inform. Theory*, St. Petersburg, Russia, 2011, pp. 1-5.

3. I. Benchimol, C. Pimentel, R. Souza, “Sectionalization of the minimal trellis module for convolutional codes”, in *35th International on Telecommunications and Signal Processing*, Praga, Rep. Tcheca, 2012, pp. 227-232.

4. I. Benchimol, C. Pimentel, R. Souza, “Análise de complexidade do módulo de treliça seccionado de códigos convolucionais”, in *XXX Simpósio Brasileiro de Telecomunicações*, Brasília, Brasil, 2012, pp. 1-5.

5. G. Moritz, R. Souza, I. Benchimol, C. Pimentel, M. Pellenz, “Análise de desempenho e complexidade da decodificação turbo utilizando a treliça mínima seccionada”, in *XXX Simpósio Brasileiro de Telecomunicações*, Brasília, Brasil, 2012, pp. 1-5.

REFERÊNCIAS

[1] S. B. Wicker, *Error Control Systems for Digital Communication and Storage*. Englewood Cliffs, Prentice-Hall, 1995.

[2] IEEE Standard 802.16e-2005. "IEEE Standard for local and metropolitan area netrworks Part 16: Air Interface for Fixed and Mobile Broadband Wireless Access Systems", Feb. 2006.

[3] 3GPP TS 45.003, "3rd Generation Partnership Project; Tecnical Specification Group GSM/EDGE Radio Acess Network; channel Coding (Release 7)", Feb. 2007.

[4] 3GPP TS 36.212 V8.7.0, "3rd generation partnership project; technical specification group radio acess network; evolved universal terrestrial radio access (E-UTRA); multiplexing and channel coding (Release 8)", 2009.

[5] R. D. Souza, C. Pimentel, D. N. Muniz, "Reduced complexity decoding of convolutional codes based on the M-Algorithm and the minimal trellis", *Annals of Telecommunications*, v. 67, pp. 537-545, 2012.

[6] F. Kienle, N. When, H. Meyr, "On complexity energy-and-implementtion-efficiency of channel decoders", *IEEE Trans. Commun*., vol. 59, no. 12, pp. 3301-3310, Dec. 2011.

[7] K. Masselos, S. Blionas, T. Rautio, "Reconfigurability requirements of wireless communication systems", in *Proc. IEEE Workshop on Heterogeneous Reconfigurable Systems on Chip,* Apr. 2002.

[8] ETSI TS 101 475, "Broadband radio access networks (BRAN), HYPERLAN type 2, Physical (PHY) layer," 2000.

[9] B. Bougard et al, "Energy-scalability enhancement of wireless local area network transceivers," in *Proc. IEEE Workshop on Signal Processing Advances in Wireless Communications*, Lisbon, Portugal, July 2004, pp. 449-453.

[10] B. Tomatsopoulos and A. Demosthenous, "A CMOS hard-decision analog convolutional decoder employing the MFDA for low-power applications," *IEEE Trans. Circuits Syst. I*: Regular Papers, vol. 55, no. 9, pp. 2912-2923, Oct. 2008.

[11] R. A. Abdallah and N. R. Shanbhag, "Error-resilient low-power Viterbi decoders via state clustering," in *Proc. IEEE Workshop on Signal Processing Systems*, Oct. 2008.

[12] R. Henning and C. Chakrabarti, "An approach for adaptively approximating the Viterbi algorithm to reduce power consumption while decoding convolutional codes," *IEEE Trans. Signal Proc*., vol. 52, no. 5, pp. 1443-1451, May 2004.

[13] C.-C. Lin, Y. H. Shih, H.-C. Chang, and C.-Y. Lee, "Design of a power reduction Viterbi decoder for WLAN applications," *IEEE Trans. Circuits Syst. I*: Regular Papers, vol. 52, no. 6, pp. 1148-1156, Jun. 2005.

[14] R. Tessier, S. Swaminathan, R. Ramaswamy, D. Goeckel, and W. Burleson, "A reconfigurable, power-efficient adaptive Viterbi decoder," *IEEE Trans. VLSI Syst*., vol. 13, no. 4, pp. 484-488, April 2005.

[15] J. Jin and C.-Y. Tsui, "Low-Power limited-search parallel state viterbi decoder implementation based on scarce state transition," *IEEE Trans. VLSI Syst*., vol. 15, no. 10, pp. 1172-1176, Oct. 2007.

[16] I. E. Bocharova and B. D. Kudryashov, “Rational rate punctured convolutional codes for soft-decision Viterbi decoding," *IEEE Trans. Inform. Theory*, vol. 43, no. 4, pp. 1305-1313, July 1997.

[17] H.-H. Tang and M.-C. Lin, "On (n,n-1) convolutional codes with low trellis complexity," *IEEE Trans. Commun*., vol. 50, n. 1, pp. 37-47, Jan. 2002.

[18] A. G. i Amat, G. Montorsi, and S. Benedetto, “Design and decoding of optimal high-rate convolutional codes," *IEEE Trans. Inform. Theory*, vol. 50, pp. 867-881, May 2004.

[19] E. Rosnes and O. Ytrehus, “Maximum length convolutional codes under a trellis complexity constraint," *Journal of Complexity*, vol. 20, pp. 372-408, March-June 2004.

[20] B. F. Uchôa-Filho, R. D. Souza, C. Pimentel, and M.-C. Lin, “Generalized punctured convolutional codes," *IEEE Commun. Letters*, vol. 9, no. 12, pp. 1070-1072, Dec. 2005.

[21] B. F. Uchôa-Filho, R. D. Souza, C. Pimentel, and M. Jar, “Convolutional codes under minimal trellis complexity measure," *IEEE Trans. Commun*., vol. 57, pp. 1-5, Jan. 2009.

[22] C. Pimentel, R. D. Souza, B. F. Uchôa-Filho, M. E. Pellenz, "Generalized punctured convolutional codes with unequal error protection," *EURASIP Journal on Advances in Signal Processing*, vol. 2008, pp. 1-7, 2008.

[23] A. Katsiotis, P. Rizomiliotis, and N. Kalouptsidis, “New constructions of low complexity convolutional codes," in *Proc. IEEE Int. Conf. on Commu*. (ICC 2009), Dresden, Germany, 2009.

[24] F. Hug, I. Bocharova, R. Johannesson, and B. D. Kudryashov, “Searching for high-rate convolutional codes via binary syndrome trellises”, in *Proc. ISIT* 2009, Seoul, Korea, 2009, pp. 1358-1362.

[25] R. J. McEliece, W. Lin, “The trellis complexity of convolutional codes”, *IEEE Trans. Inform. Theory*, vol. 42, no. 6, pp. 1855-1864, Nov. 1996.

[26] A. Katsiotis, P. Rizomiliotis, and N. Kalouptsidis, “Flexible convolutional codes: variable rate and complexity”, *IEEE Trans. Commun*., vol. 60, no. 3, pp. 608-613, Mar. 2012.

[27] V. Sidorenko, V. Zyablov, “Decoding of convolutional codes using a syndrome trellis”, *IEEE Trans. Inform. Theory*, vol. 40, no. 5, pp. 1663-1666, Sept 1994.

[28] R. J. McEliece, “On the BCJR trellis for linear block codes”, *IEEE Trans. Inform. Theory*, vol. 42, no. 4, pp. 1072-1092, Jul. 1996.

[29] A. Vardy, “Trellis structure of codes”, in Handbook of Coding Theory, vol. II, (V. S. Pless and W.C. Huffman, eds.) Amsterdam. The Netherlands: North-Holland, 1998.

[30] A. Lafourcade, A. Vardy, “Optimal sectionalization of a trellis”, *IEEE Trans. Inform. Theory,* vol. 42, no. 3, pp. 689-703, May 1996.

[31] L. Zhao, J. Cai, and H. Zhang, “Radio-efficient adaptive modulation and coding: green communication perspective”, in *Proc. Vehicular Technology Conference*, Budapest, Hungary, 2011, pp. 1-5.

[32] F. Labeau, “Complexity evaluation of nonbinary SOVA for sectionalized trellises”, in *Proc. 41st Allerton Conference on Communication, Control and Computing*, Oct. 2003.

[33] I. Benchimol, C. Pimentel, R. Souza, B. Uchôa-Filho, “High-rate systematic recursive convolutional encoders: minimal trellis and code search”, *EURASIP Journal on Advances in Signal Processing*, vol. 2012, pp. 243, 2012.

[34] C. Douillard and C. Berrou, “Turbo codes with rate-m/(m+1) constituent convolutional codes”, *IEEE Trans. Commun*., vol. 53, no. 10, pp. 1630-1638, Oct. 2005.

[35] D. J. Costello, S. Lin, *Error Control Coding*. Upper Saddle River: Pearson Prentice Hall, 2nd ed., 2004.

[36] J. C. Moreira, P. G. Farrell, *Essentials of Error Control Coding*. John Wiley & Sons Ltd, 2006.

[37] R. Johannesson and K. S. Zigangirov, *Fundamentals of Convolutional Coding*, Wiley-IEEE Press, 1999.

[38] C. Schlegel and L. Pérez, *Trellis and Turbo Coding*. Wiley-IEEE Press, 2004.

[39] T. K. Moon, *Error Correction Coding: Mathematical Methods and Algorithms.* Wiley & Sons, 2005.

[40] F. R. Kschischang, V. Sorokine, “On the trellis structure of block codes”, *IEEE Trans. Inform. Theory*, vol. 41, pp. 1924-1937, Nov. 1995.

[41] S. M. Kuo, B. H. Lee, W. Tian, “Real-Time Digital Signal Processing: Implementations and Applications”, Second Editon, Jonh Wiley & Sons, 2006.

[42] Texas Instruments, Inc., *TMS320C55x DSP CPU Reference Guide*, Literature no. SPRU371F, 2004.

[43] Texas Instruments, Inc., “Code Composer Studio user’s guide (Rev B)”, Literature no. SPRU328B, Mar. 2000.

[44] I. E. Bocharova, R. Johannesson, B. D. Kudryashov, “Trellis complexity of short linear codes”, *IEEE Trans. Inform. Theory,* vol. 53, no. 1, pp. 361-365, Jan. 2007.

[45] D. J. Muder, “Minimal trellises for block codes”, *IEEE Trans. Inform. Theory,* vol. 34, pp. 1049-1053, Sep. 1988.

[46] A. Vardy, Y. Be’ery, “Maximum-Likelihood soft decision decoding of BCH codes”, *IEEE Trans. Inform. Theory,* vol. 40, no. 2, pp. 546-554, Mar. 1994

[47] S. Ranpara and D. Ha, “A low-power Viterbi decoder design for wireless communications applications”, in *Proc. International ASIC Conference,* Washington D.C., USA, Sep. 1999, pp. 1-5.

[48] M.-H. Chan, W.-T. Lee and L.-G. Chen, “IC design of an adaptive Viterbi Decoder”, *IEEE Trans. on Consumer Electronics*, vol. 42, no. 1, pp. 52-62, Feb. 1996.

[49] A. Katsiotis and N. Kalouptsidis, “On (n,n-1) punctured convolutional codes and their trellis modules”, *IEEE Trans. Commun*., vol. 59, no. 5, pp. 1213-1217, May 2011.

[50] B. U. Pedroni, V. A. Pedroni, and R. D. Souza, “Hardware implementation of a Viterbi decoder using the minimal trellis”, in *Proc. 4th Inter. Symp. on Commun., Control and Signal Processing* (ISCCSP'2010), pp. 1-4, Limassol, Cyprus, Mar. 2010.

[51] B. Uchôa-Filho, R. Souza, C. Pimentel, “On the behavior of the distance spectrum of convolutional codes under a minimal trellis complexity measure”, in *Proc. IEEE Information Theory Workshop*, Punta del Este, Uruguay, 2006, pp. 1-5.

[52] H.-H. Tang, M.-C. Lin, B. F. Uchôa-Filho, “Minimal trellis modules and equivalent convolutional codes”, *IEEE Trans. Inform. Theory,* vol. 52, no. 8, pp. 3738-3746, Aug. 2006.

[53] C. Berrou, A. Glavieux and P. Thitimajshima, “Near shannon limit error-correcting coding: turbo codes”, in *Proc.* 1*993 IEEE International Conference on Communication*, pp. 1064 – 1070, 1993.

[54] G. Battail, “A conceptual framework for understanding turbo codes”, *IEEE Journal on Selected Areas in Communications*, v. 16, pp. 245-254, Fevereiro 1998.

[55] S. A. Barbulescu, S. Pietrobon, “Interleaver design for turbo codes”, *Electronics Letters,* v. 30, pp. 2107-2108, Dezembro 1994.

[56] O. Joerssen, H. Meyer, “Terminating the trellis of turbo codes”, *Electronics Letters,* v. 30, pp. 1285-1286, Agosto 1994.

[57] M. J. Silva. *Projeto de entrelaçadores para códigos turbo*. Recife, 2006. Dissertação (Mestrado em Engenharia Elétrica) – Departamento de Eletrônica e Sistemas, Universidade Federal de Pernambuco.

[58] M. A. Kousa and A. H. Mugaibel, “Puncturing effects on turbo codes”, *IEEE Proc. Commun*., vol. 149, no.3, pp. 132-138, June 2002.

[59] I. Chatzigeorgiou, M. R. D. Rodrigues and I. J. Wassell, “The augmented state diagram and its application to convolutional and turbo codes”, *IEEE Trans. Commun*., vol. 57, no. 7, pp. 1948-1958, July 2009.

[60] S. Benedeto and G. Montorsi, “Design of parallel concatenated convolutional codes”, *IEEE Trans. Inform. Commu,* vol. 44, pp. 591-600, May 1996.

[61] D. Divsalar and F. Pollara, "On the design of turbo codes," TDA Prog. Rep., 1995, 42-123, pp. 99-121.

[62] S. Benedeto, R. Garello and G. Montorsi, “A search for good convolutional codes to be used in the construction of turbo codes”, *IEEE Trans. on Communications,* vol. 46, no. 9, pp. 1101-1105, Sep. 1998.

[63] R. McEliece, “How to compute weight enumerators for convolutional codes”, Communications and Coding, M Darnell and B. Honary, Eds New York: Wiley, 1998, pp. 121-141.

[64] G. Moritz, R. Souza, I. Benchimol, C. Pimentel, M. Pellenz, “Análise de desempenho e complexidade da decodificação turbo utilizando a treliça mínima e seccionada”, in *Proc*. XXX Simpósio Brasileiro de Telecomunicações, Brasília, Brasil, 2012, pp. 1-5.

[65] G. Moritz. *Análise de complexidade de códigos turbo utilizando a treliça mínima e seccionada*. Curitiba, 2012. Dissertação (Mestrado em Engenharia Elétrica e Informática Industrial) – Universidade Tecnológica Federal do Paraná.