INSTITUTO DE PESQUISAS ESPACIAIS

DISSERTAÇÃO

CONTRIBUIÇÃO AO ESTUDO DE SISTEMAS DE MODULAÇÃO CODIFICADA EM CÓDIGOS CONCATENADOS

SUBMETIDA POR

Marco Antonio Chamon

Em cumprimento parcial dos requisitos exigidos para obtenção do título de Mestre em Eletrônica e Telecomunicações

1989

Aprovada pela Banca Examinadora
em cumprimento a requisito exigido
para a obtenção do Título de Mestre
Eletrônica e Telecomunicações

Dr. Nelson Delfino d'Ávila Mascarenhas — Presidente

Dr. Plinio Tissi — Orientador

Dr. Dalton Soares Arantes — Orientador

Dr. Áydano Barreto Carleial — Membro da Banca

Dr. Fernando Walter — Membro da Banca
- convidado -

Candidato: Marco Antonio Chamon

São José dos Campos, 30 de maio de 1989

À minha esposa Edna e a
meus pais Luiz e Odette,
com amor e carinho.

AGRADECIMENTOS

Ao Dr. Max Costa, que definiu a trajetória deste trabalho e que sempre o acompanhou. Meu agradecimento pelas discussões, pelas idéias, pelo incentivo, pela orientação enfim.

Ao Dr. Dalton Arantes, pelo apoio nos momentos de indecisão e pela valiosa bibliografia cedida, inencontrável de outra forma.

Ao Dr. Plínio Tissi, que aceitou a difícil tarefa de orientar a fase final deste trabalho e em quem encontrei o auxílio para os últimos obstáculos.

Por fim, um agradecimento especial àquela que soube suportar pacientemente as horas que este trabalho roubou ao convívio familiar.

RESUMO

Este trabalho apresenta um estudo de sistemas de modulação codificada aplicados a codificação concatenada. As análises de esquemas de modulação codificada de bloco e de treliça são mostradas bem como as simulações realizadas para escolher o código interno mais adequado ao esquema de concatenação. As simulações mostram o compromisso entre os vários parâmetros dos códigos de treliça analisados, tais como comprimento de decodificação, número de níveis de quantização (para decisão suave) e número de estados na treliça do código. O programa desenvolvido para realizar essas simulações está incluido no apêndice. O desempenho do código concatenado é mostrado e comparado com aquele do esquema concatenado padrão do CCSDS.

ABSTRACT

This work presents a study of coded modulation systems applied to concatenated codes. The analyses of some trellis and block coded modulation schemes are showed and simulations are performed to choose the most suitable internal code for the concatenated scheme. The simulations show the trade-off among several import parameters of the trellis codes analysed, such as code decision depht, number of quantization levels (for soft decision) and number of states in the trellis. The program developed for the simulations are included in the Appendix. The performance of the concatenated code is presented and compared with the CCSDS standard code for concatenated schemes.

SUMÁRIO

LISTA DE FIGURAS

Pág.

LISTA DE SÍMBOLOS

C	- Capacidade do canal.
d, d_{min}	- Distância mínima do código.
d_p	- Distância entre transições paralelas.
d_{red}	- Distância mínima do código reduzido.
d_{ref}	- Distância mínima da constelação reduzido.
Eb/N_o	- Energia de bit por densidade espectral de ruído.
Es	- Energia média de símbolo de uma constelação.
Gass	- Ganho assintótico.
n, N	- Comprimento de uma palavra de código.
(n,k,d)	- (comprimento da palavra, número de bits de informação, distância mínima).
Pb	- Probabilidade de erro de bit
Pe	- Probabilidade de erro de palavra.
Pe,n	- Probabilidade de erro de palavra para um código de tamanho n.
Pev	- Probabilidade de erro de evento.
Ps	- Probabilidade de erro de símbolo.
Q(x)	- Função erro.
R	- Taxa (total) do código.
R_{ext}	- Taxa do código externo.
R_{int}	- Taxa do código interno.
Ro	- Taxa efetiva do canal (cut-off rate).
SNR	- Relação sinal-ruído.
t	- Capacidade de correção de um código.
Δ_0	- Distância mínima entre símbolos do 16_QAM.
σ	- Desvio padrão.

CAPÍTULO 1

CÓDIGOS CONCATENADOS

1.1 - INTRODUÇÃO

Códigos concatenados têm sido frequentemente usados em sistemas de comunicação espacial como um meio prático de obter altos ganhos de codificação, para uma complexidade moderada. Isso é obtido às custas de uma taxa de codificação relativamente baixa, o que equivale a uma grande expansão da banda utilizada para uma determinada taxa de transmissão de informação. Esses sistemas, portanto, atingem alta eficiência em termos de utilização de potência com o sacrifício da eficiência espectral.

Recentemente tem-se desenvolvido um esforço muito intenso, dentro do qual se insere o presente trabalho, no sentido de se resgatar essa perda espectral, quer por meio de novos esquemas de códigos com taxas mais elevadas [1], quer por meio do emprego de sistemas de modulação multi_nível [2].

O conceito de codificação concatenada foi introduzido originalmente por Forney como um meio de atingir taxas de erro exponencialmente decrescentes, em função do comprimento das palavras do código, sem incorrer em complexidade exponencialmente crescente. Por meio da concatenação de códigos, Forney foi capaz de demonstrar o seguinte teorema [3]:

Para qualquer $R<C$, existe uma sequência de códigos C1, C2, ...Cn (onde Cn tem comprimento n), cada um tendo taxa maior ou igual a R, tal que

(a) $Pe,n < 2^{-E(R).N}$,

(b) a complexidade de codificação e decodificação de Cn é $O(n^4)$.

Na demonstração, Forney utiliza o argumento da codificação aleatória para o código interno, fixando para o código externo uma classe particular de códigos lineares - os códigos de Reed-Solomon.

Esse resultado diminui grandemente a distância entre a Teoria de Informação (que mostra o que é possível atingir) e a Teoria de Codificação (que mostra o que é prático de se atingir). Entretanto, a demonstração não é construtiva, no sentido de que o código interno não é explicitamente caracterizado. Dessa forma, o problema de obter uma sequência de códigos práticos para os quais a Pe seja exponencialmente decrescente permanece em aberto. Caso esse último teorema seja obtido, finalmente estarão unidas ambas as áreas de Teoria de Informação e Codificação (que provavelmente desaparecerão em seguida).

O interesse em códigos concatenados, no entanto, não se restringe apenas à busca de teoremas construtivos. Eles já se tornaram padrão recomendado pelo CCSDS em longas missões espaciais, tais como a de sondas de exploração de outros planetas [4].

Justifica-se assim a busca de novas técnicas e algoritmos que tornem a implementação desses códigos mais simples e/ou eficiente, bem como adequados aos sistemas de comunicação que já existam.

1.2 - ESTRUTURA DOS CÓDIGOS

Sistemas que utilizam codificação concatenada geralmente são implementados com dois níveis de codificação, como indicado na Figura 1.1. Dados binários são gerados por uma fonte de informação e constituem a

entrada para o primeiro codificador (código externo). A saída desse codificador pode então sofrer um interleaving (entrelaçamento), sendo em seguida apresentada à entrada do segundo codificador (código interno).

A sequência de símbolos assim obtida é enviada através do canal de transmissão. A decodificação se dá de modo inverso, como mostrado na mesma figura.

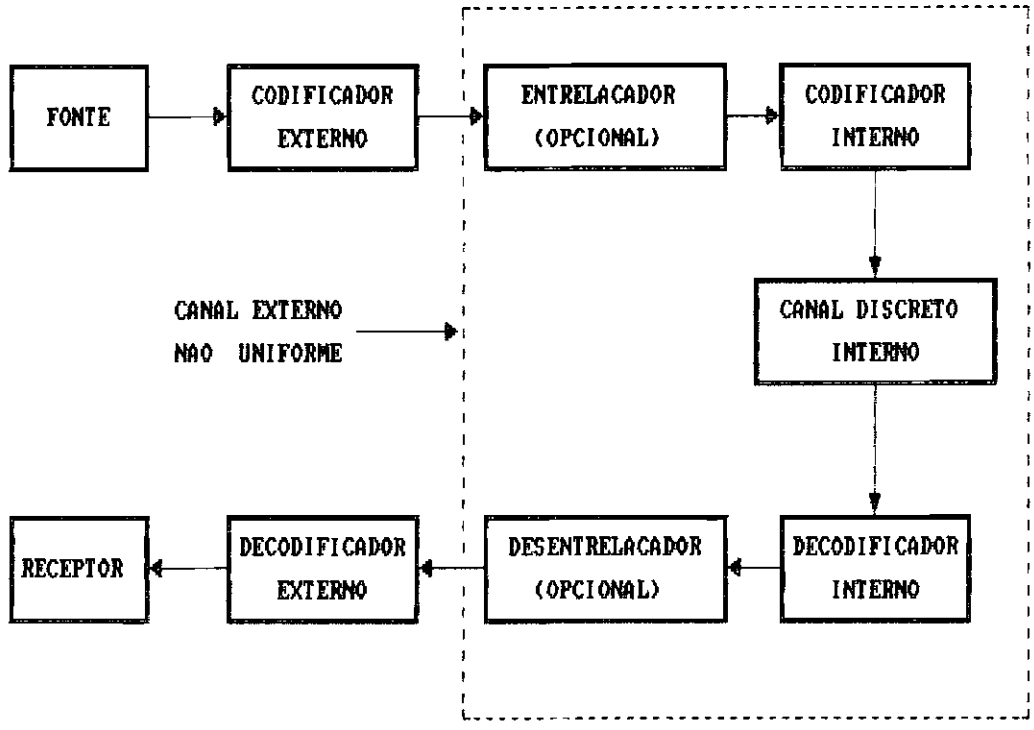


Fig. 1.1 - Diagrama de blocos de um esquema de codificação concatenada.

O conjunto em destaque (codificador interno/canal/ decodificador interno) normalmente é visto como um novo canal - super canal ou canal externo, como é chamado - que se apresenta ao codificador externo. Enquanto o canal interno é considerado um canal discreto sem

memória,o canal externo é não-uniforme, pois o "ruído" que ele acrescenta aos símbolos do codificador é gerado pelas falhas na decodificação do código interno.

Num esquema típico, o código externo é de Reed-Solomon (RS), um código cíclico não-binário, que agrupa m bits da fonte por vez e opera sobre o conjunto dos 2^m símbolos resultantes. Como o código tem capacidade para corrigir um certo número t de **símbolos** errados, isso equivale a corrigir um surto de mt **bits** errados. Essa capacidade de correção de surtos faz com que esses códigos RS sejam muito adequados à aplicação em esquemas concatenados, onde as falhas do decodificador interno tornam, de um modo geral, os erros que são passados ao decodificador externo fortemente correlacionados.

Os códigos internos utilizados podem ser de bloco ou convolucionais e normalmente são binários. Um esquema, devido a Odenwalder [5], que se tornou muito popular, usa um código convolucional binário de taxa R = 1/2 e comprimento de memória v = 7. No caso de códigos de bloco, usa-se em geral um que seja "casado" ao código externo, isto é, tal que o número de bits de informação que compõem o bloco seja idêntico ao número de bits que formam o símbolo do código RS (ou seja, m).

A Figura 1.2 ilustra uma palavra de código que foi gerada a partir da concatenação de um código externo RS (31,15), com m = 5 e um código binário interno BCH (15,5). Aqui se vê claramente a operação conjunta dos códigos : o código externo protege grupos de bits da fonte (informação) e o código interno protege cada símbolo (m-bits) do código externo, seja ele informação ou redundância.

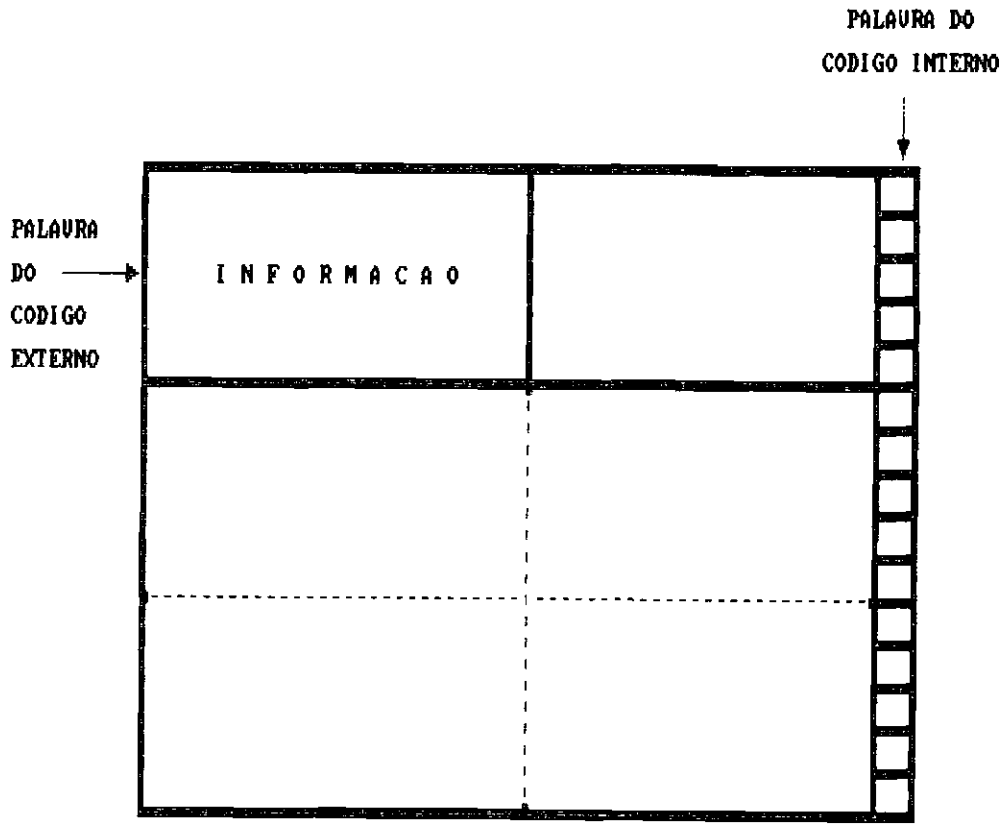


Fig. 1.2 - Palavra de código para concatenação em dois níveis.

O conjunto interleaver/de-interleaver (entrelaçador/ desentrelaçador), apresentado como opcional na Figura 1.1, cumpre dupla função. Por um lado quebra surtos de erro muito grandes em blocos menores, capazes de serem tratados pelo código externo, isto é, procura adequar o tamanho dos surtos à capacidade de correção do código. De um modo geral, funciona como meio de diminuir a correlação entre os símbolos errados que deixam o super canal. A segunda função é promover o "casamento" entre o código externo e o interno quando isso é necessário.

A forma mais simples de implementar o entrelaçador está mostrada na Figura 1.3. Consiste em montar uma matriz de palavras de código, onde as **colunas** da matriz são formadas por palavras do código externo e as **linhas** da matriz são codificadas pelo código interno. No

receptor, após a decodificação interna, o desentrelaçador monta a matriz por **linhas** e apresenta as **colunas** ao decodificador externo [6].

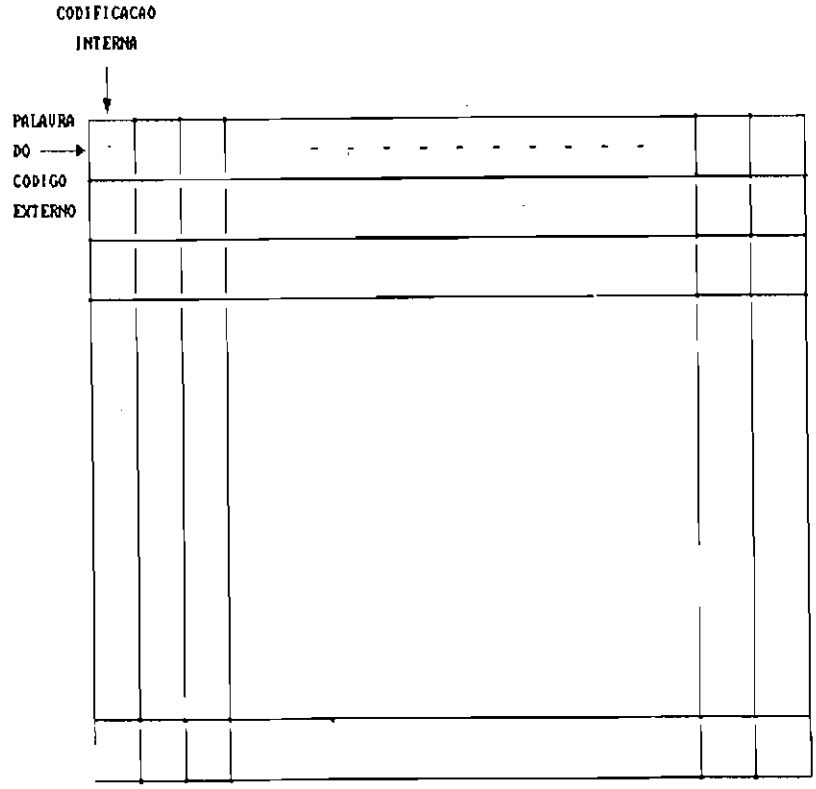


Fig. 1.3 - Técnica de entrelaçamento de símbolos para transmissão de palavras de um código concatenado.

O modulador, que não está indicado explicitamente na Figura 1.1, utiliza em geral uma constelação BPSK, transmitindo uma fração de bit de informação (equivalente à taxa total do código utilizado) por símbolo do canal. O demodulador geralmente oferece alguma informação de confiabilidade do sinal demodulado (decisão suave).

1.3 - ESTIMATIVA DE DESEMPENHO

A estrutura imposta a um código pelo processo de concatenação permite simplificar grandemente o trabalho de decodificação, que pode ser realizado por dois

decodificadores em cascata. A análise de desempenho desses códigos também é bastante simplificada, tendo em vista que se pode imaginar o código interno como integrante de um supercanal e fazer a análise do código externo separadamente.

Assumindo que o código externo é de Reed-Solomon, com capacidade para corrigir **t** erros e que o decodificador externo não corrige além desse limite, podemos calcular a probabilidade de erro de uma palavra de código, P_e, como

$$P_e = \sum_{i=t+1}^{N} \binom{N}{i} P_s^i \, (1-P_s)^{N-i} \qquad (1.1)$$

onde

- N - comprimento da palavra de código;
- t - capacidade de correção do código;
- P_s - probabilidade de erro de símbolo na entrada do decodificador externo, ou seja, na saída do decodificador interno.

Essa expressão para Pe representa simplesmente a probabilidade de ocorrerem mais de **t** erros num conjunto de **N** símbolos. Nela se considera que os símbolos errados que saem do supercanal são estatísticamente independentes.

De um modo geral, no entanto, é a probabilidade de erro de bit de informação, P_b, o fator mais importante na análise de desempenho de um código. Para estimá-la consideraremos, numa hipótese pessimista, que um padrão de **i** erros (i > t) do canal leva à decodificação de uma palavra que difere em i+t posições da palavra transmitida, ou seja, uma fração de (i+t)/N símbolos errados. Assim teremos

$$P_b < \frac{2^{m-1}}{2^m-1} \sum_{i=t+1}^{N} \frac{t+i}{N} \binom{N}{i} P_s^i \, (1-P_s)^{N-i} \qquad (1.2)$$

onde **m** é o número de bits que representam um símbolo (um código RS opera com símbolos de um corpo de Galois GF(2)) e o fator $2^{m-1}/2^{m}-1$ leva em conta o número médio de bits de informação errados por símbolo [6].

Uma expressão alternativa pode ser obtida se admitirmos que os erros de decodificação ocorram apenas no sentido de confundir palavras separadas pela distância mínima **d**, i.e., o número de erros é exatamente t+1. Nesse caso, a Expressão (1.2) torna-se

$$Pb \sim \frac{d}{2N} \sum_{i=t+1}^{N} \binom{N}{i} P_s^i (1-P_s)^{N-i} \qquad (1.3)$$

A probabilidade de erro de símbolo, P_s, que é utilizada nessas expressões, deve ser obtida independentemente através de análise ou simulação.

1.4 - ESQUEMAS ALTERNATIVOS

Baseados na idéia de se recuperar a perda na eficiência espectral já mencionada, esquemas alternativos para códigos concatenados têm sido buscados.

As pesquisas de Brine e Farrell [1] orientam-se no sentido de utilizar um código externo RS de taxa R = 7/8 e códigos internos de taxas elevadas, situando o resultado final na faixa de 0.65 (para códigos internos convolucinais) até 0.78 (para códigos internos de bloco), o que se compara favoravelmente com o esquema padrão que usa um código interno de taxa 0.5 e atinge taxa efetiva de 0.44. Nesse caso, os autores foram capazes de identificar esquemas que conjugavam altas taxas, complexidade moderada e ganhos satisfatórios para aplicação em sistemas de comunicação via satélite.

O trabalho de Deng e Costello Jr. [2] é mais específico, concentrando-se em códigos internos de treliça que sejam eficientes em faixa. Os autores analisaram

códigos variantes no tempo, com altas taxas e baixa complexidade, sistemas de modulação em quatro dimensões e sistemas de modulação codificada.

O presente trabalho se alinha com o último, procurando analisar as características de sistemas de modulação codificada em constelações retangulares.

CAPÍTULO 2

MODULAÇÃO CODIFICADA

2.1 - ORIGENS

A rigor, a modulação codificada teve suas origens nos trabalhos de Shannon [7] que assentaram a base das atuais teoria de informação e codificação. Isso porque nesses estudos jamais chegou-se a distinguir entre modulação e codificação, considerando-se sempre que o modulador/demodulador geram um determinado canal discreto através do qual o codificador envia as palavras de código.

Massey [8,9] chamou a atenção para esse fato, apontando para algumas interpretações errôneas que existiam sobre codificação e indicando a função correta do sistema de modulação : criar um canal discreto adequado (buscando maximizar a taxa efetiva do canal, Ro) que possa ser aproveitado pelo codificador. Ele propunha assim, que modulação e codificação fossem conjugadas, numa antevisão dos atuais esquemas de modulação codificada.

Mais tarde Digeon [10] investigou um esquema de codificação - que utilizava códigos convolucionais, decisão analógica (quantização infinita) e um alfabeto maior que o número de mensagens a ser enviadas - onde não havia redução na taxa de bits/transmissão e ganhos efetivos da ordem de 2 dB podiam ser obtidos. A modulação codificada tomava corpo.

Finalmente, com o trabalho de Ungerboeck [11] a área é sistematizada. Ungerboeck mostrou o que é teoricamente possível atingir em termos de ganho de codificação sem sacrificar a taxa de informação transmitida, apresentou sistemas práticos de modulação

codificada e apontou o caminho para a busca de novos e melhores códigos para esses sistemas.

Mostraremos a seguir no que consiste essa sistematização, isto é, qual a técnica desenvolvida por Ungerboeck, além de alternativas a ela que surgiram depois, dentro da mesma filosofia.

2.2 - ALFABETOS REDUNDANTES

A fim de aumentar a eficiência espectral dos sistemas de comunicação, esquemas de modulação multi-nível (fase e/ou amplitude) podem ser utilizados. Entretanto isso torna necessário um gasto maior de energia para se manter a probabilidade de erro constante. Em ambientes com restrição de potência, novas técnicas devem ser empregadas para atingir o desempenho desejado com potências mais baixas. Uma solução possível é o uso de códigos corretores de erro; no caso clássico, essa habilidade de corrigir erros só pode ser obtida através do acréscimo de bits extras (redundantes) à sequência de símbolos de informação. Com isso, reduz-se a taxa efetiva de informação por Hz pois torna-se necessário diminuir a taxa de transmissão para se acomodar à mesma faixa ou expandir a faixa para manter constante a taxa de informação. Em outras palavras, o aumento da eficiência em potência é conseguido com a diminuição da eficiência em faixa.

Uma solução alternativa, que evita a perda na relação taxa / banda de transmissão, é usar um conjunto de sinais maior no sistema de modulação. Nesse caso a redundância necessária ao processo de codificação é fornecida pelo aumento no número de símbolos codificados. Essa solução implica necessariamente no uso de modulação não-binária. Os argumentos originais de Shannon [7] já mostravam que à medida que o tamanho do alfabeto fosse aumentando, mantando-se a taxa de informação inalterada, aumentaria a eficiência na utilização da potência. A esse

alfabeto aumentado daremos o nome de alfabeto redundante (AR) e ao código a ele associado o nome de código de alfabeto redundante.

2.3 - GANHOS DE CODIFICAÇÃO

Admitindo que se pretenda trabalhar com decodificação suave no demodulador, pode-se calcular a capacidade do canal com entrada discreta, multi_nível, e saída contínua, considerando o ruído como aditivo, Gaussiano e branco. Se isso for feito para vários tipos de constelação, obteremos as curvas da Figura 2.1 [11].

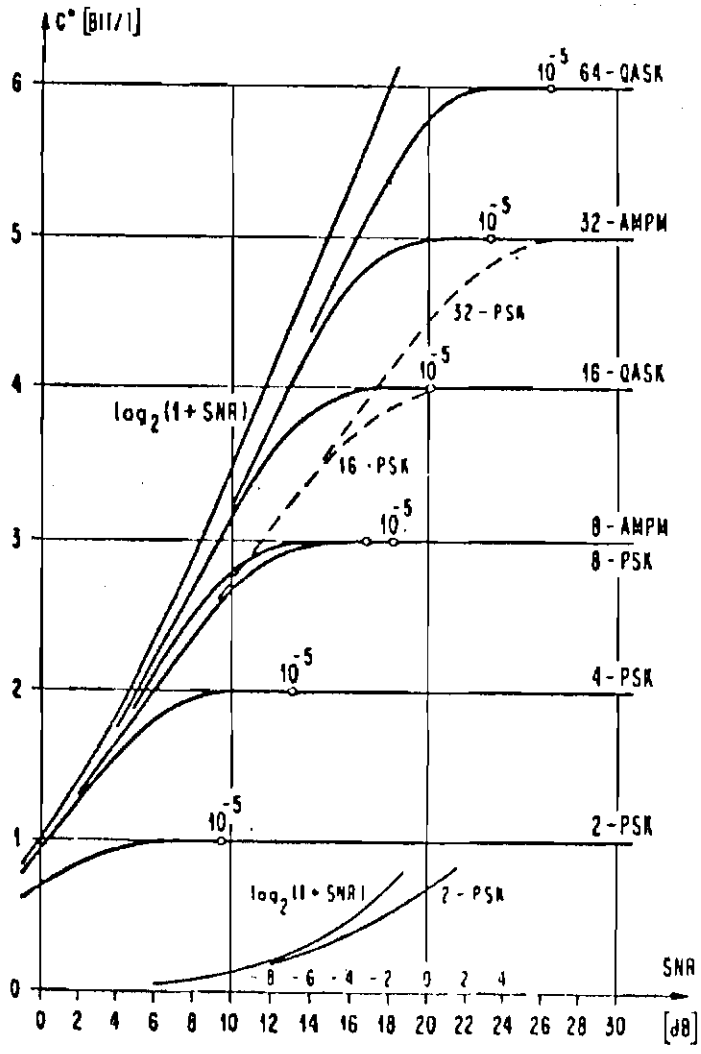


Fig. 2.1 - Capacidade de canal para canais limitados em faixa.

A partir dessas curvas pode-se verificar que enquanto uma taxa de 2 bits/T com $P_e = 10^{-5}$ pode ser obtida com uma constelação 4-PSK e uma relação sinal-ruído de 12.9 dB, o uso de um alfabeto maior como o 8-PSK pode atingir teoricamente uma taxa de 2 bits/T livre de erro já em 5.9 dB (supondo-se um esforço de codificação e decodificação ilimitado). Pode-se observar também que praticamente todo o ganho possível já foi atingido, uma vez que apenas 1.2 dB extra poderia ser conseguido caso se aumentasse indefinidamente as constelações (mantendo-se a limitação de potência).

Dessa forma, duas conclusões de caráter eminentemente prático podem ser feitas : o uso de AR é capaz de oferecer ganhos consideráveis - da ordem de 7 a 8 dB - sem perdas na taxa de transmissão e, além disso, a maior parte desse ganho pode ser atingida expandindo-se a constelação por um fator de apenas duas vezes, minimizando assim o aumento na complexidade que poderia acompanhar essa expansão do alfabeto.

2.4 - A PARTIÇÃO DE CONJUNTOS

A utilização de AR e códigos corretores de erro da forma clássica, isto é, com modulação e codificação operando independentemente, tem eficiência extremamente baixa [11]. Isso ocorre porque a distância Euclidiana (e não a de Hamming) é a métrica mais adequada para representar a separação entre símbolos em uma constelação multi-nível. Embora isso não apresente maiores inconvenientes quando se utiliza uma transmissão BPSK ou 4-PSK, nas constelações maiores não há uma maneira óbvia de associar as palavras de código aos símbolos da constelação (ou seja, fazer a modulação) de maneira a preservar a estrutura de distância entre essas palavras. Dito de outra forma, se as distâncias de Hamming entre as palavras de código são maximizadas, isso não significa que as distâncias euclidianas entre as sequências de símbolos que

saem do modulador também o sejam. Um mapeamento de Gray não faz isso sempre e não é simples, a princípio, obter qualquer função de mapeamento que transforme monotonicamente distâncias de Hamming em distâncias euclidianas, exceto em casos triviais.

A técnica de partição de conjunto é uma forma sistemática de realizar esse mapeamento, que foi desenvolvida por Ungerboeck. Nela, um alfabeto de 2M símbolos do canal é sucessivamente dividido em 2, 4, 8 ... subconjuntos de tamanhos M, M/2, M/4 ... que possuem distâncias mínimas inter-símbolos cada vez maiores. A distribuição dos símbolos nos subconjuntos é simétrica e a partição segue um esquema de árvore binária. A Figura 2.2 mostra os dois primeiros passos de partições realizadas em diferentes constelações de símbolos. Nesses exemplos pode-se notar que a distância (euclidiana) entre os pontos de um mesmo subconjunto aumenta progressivamente à medida que se prossegue na partição.

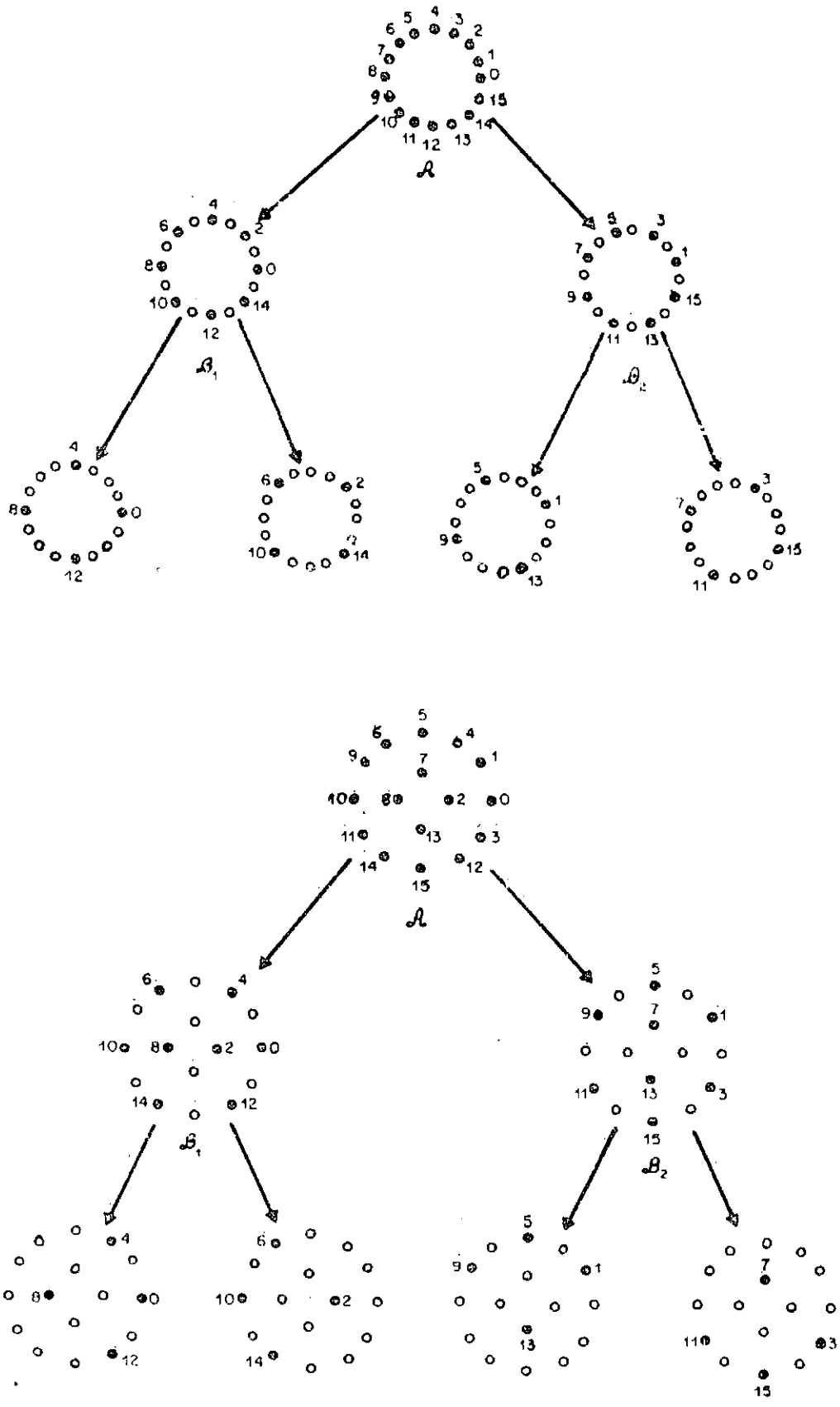


Fig. 2.2 - Partição de um alfabeto de canal em subconjuntos. a) 16_PSK b) 12/4_PSK.

2.5 - CÓDIGOS DE ALFABETO REDUNDANTE

Os subconjuntos obtidos através dessa estratégia de partição podem ser utilizados na implementação de esquemas de codificação relativamente simples mas efetivos, cuja representação geral pode ser vista na Figura 2.3 [12].

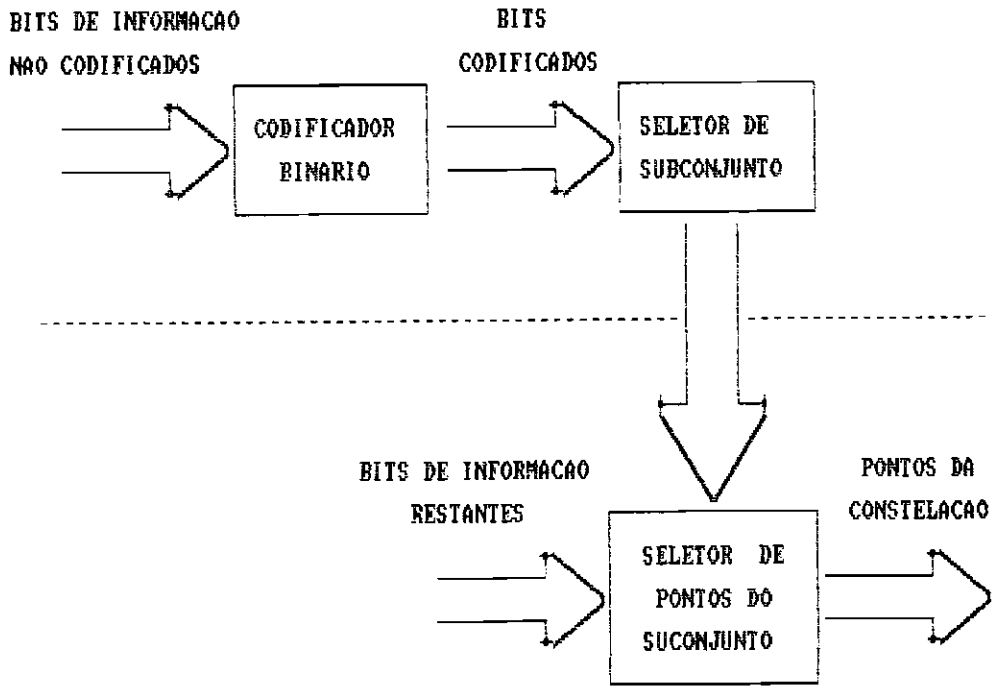


Fig 2.3 - Esquema geral de modulação codificada.

Alguns bits de entrada atravessam um codificador binário, resultando em um número maior de bits, devido ao acréscimo de redundância. Esses bits codificados são a seguir utilizados para definir um subconjunto da partição, isto é, o subconjunto de pontos da constelação que será usado na transmissão seguinte. Os bits de entrada restantes não são codificados, mas apenas definem o ponto

do subconjunto selecionado que será finalmente transmitido. A constelação é escolhida de modo adequado para comportar todos os bits.

Dessa forma, o esquema de codificação está separado da escolha da constelação. O ganho devido ao código é determinado pelas propriedades de distância entre os subconjuntos, enquanto que as propriedades de simetria e distância da própria constelação de pontos são exploradas pelos bits não codificados.

2.6 - CÓDIGOS EXISTENTES

Dentro desse esquema, vários códigos já foram propostos, utilizando tanto codificação de bloco como convolucional. Fazemos a seguir um apanhado dos de maior destaque e que se relacionam com o presente trabalho.

2.6.1 - CÓDIGOS DE UNGERBOECK

Uma vez que a análise das curvas de capacidade mostra que é suficiente dobrar o tamanho da constelação para atingir ganhos já próximos do máximo, Ungerboeck concentrou sua atenção em códigos convolucionais binários de taxa $m/(m+1)$. Os códigos convolucionais já têm inclusive um algoritmo de decodificação de máxima verossimilhança - o algoritmo de Viterbi [13] - que se adapta perfeitamente à decodificação suave.

O esquema do codificador pode ser visto na Figura 2.4, onde se nota a existência de um mapeamento já embutido na estrutura. Esse mapeamento é baseado na partição de conjuntos desenvolvida por Ungerboeck.

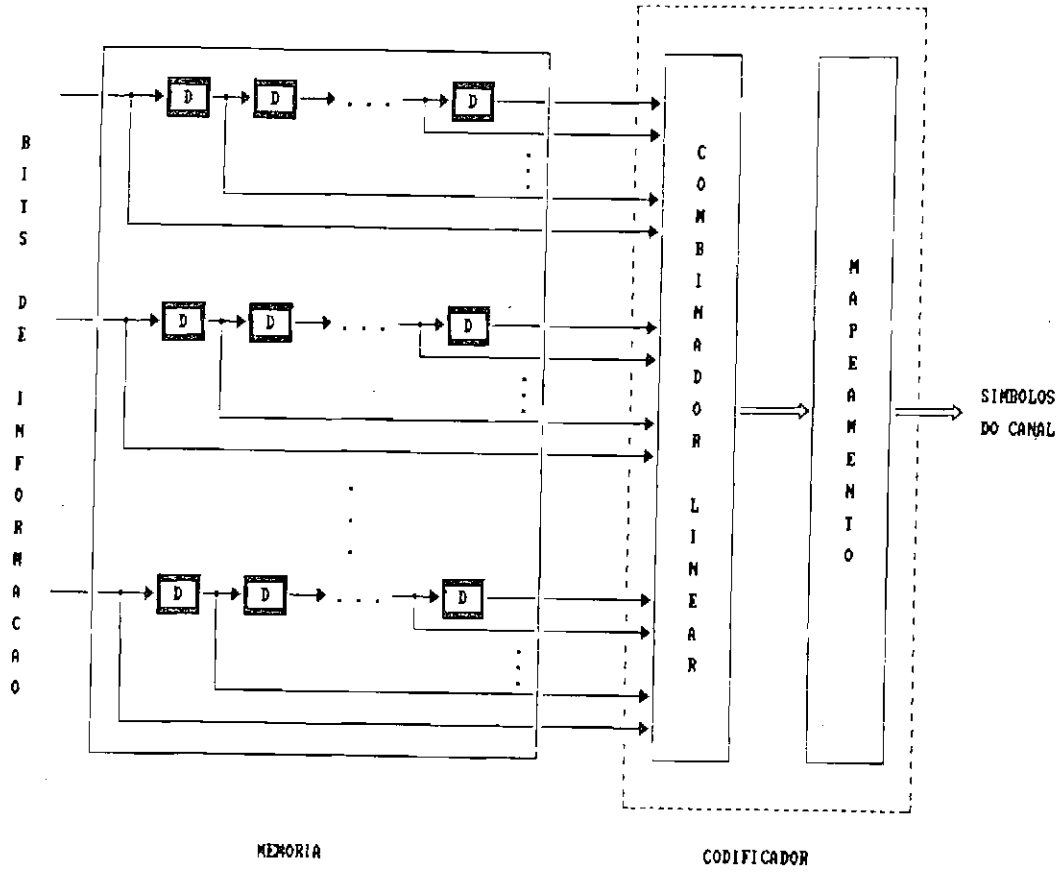


Fig 2.4 - Codificador convolucional - mapeamento por partição de conjuntos.

O diagrama da memória do codificador define o número de estados e a característica das transições entre estados na treliça que representa o código. Assim, quando alguma memória está presente em todos os bits de entrada as transições são únicas, ao passo que transições paralelas estão associadas aos bits de entrada que não contém elementos de memória. A Figura 2.5 traz um exemplo simples de um codificador onde se pode notar a existência de transições paralelas entre estados.

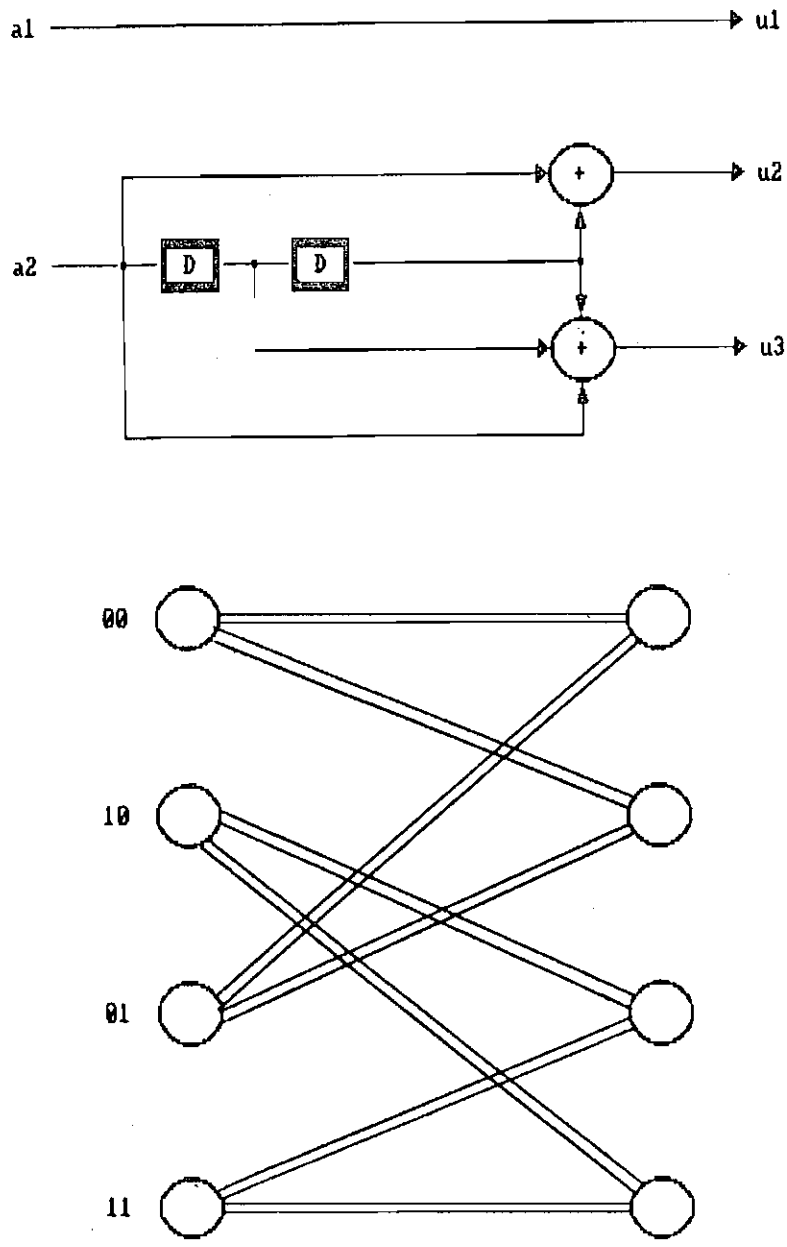


Fig 2.5 - Código convolucional com transições paralelas entre os estados.

A busca de códigos convolucionais ótimos para esse tipo de estrutura, isto é, de códigos com distâncias euclidianas maximizadas, baseia-se em três regras

a - Todos os sinais da constelação devem ocorrer com a mesma frequência;

b - A transições simples que partem de um mesmo estado ou chegam a um mesmo estado devem ser atribuídos sinais com mínima distância euclidiana.

c - A transições paralelas devem ser atribuídos sinais com distância euclidiana máxima, ou seja, do último nível da partição.

Essa regras refletem a idéia geral da cooperação mútua entre o código e o sistema de modulação : quando os símbolos recebidos são próximos o bastante para que o demodulador pudesse cometer um erro, a distância entre sequências do código garante a decodificação correta; por outro lado, quando o decodificador corre o risco de decidir errado entre duas sequências próximas, a distância entre os pontos na constelação é grande o bastante para o demodulador indicar o símbolo correto.

Utilizando uma constelação 8-PSK, esquemas de codificação construídos segundo as regras acima garantem, mesmo para códigos bastantes simples, ganhos da ordem de 3 dB do sistema codificado em relação ao não-codificado (que transmita o mesmo número m de bits de informação por símbolo do canal). Esse ganho é assintótico, no sentido de que representa um aumento na distância mínima entre sequências de palavras do código. Este ganho estará limitado ao espaçamento máximo entre pontos da constelação (e não à capacidade de correção do código) se existirem transições paralelas na treliça, pois isso gera uma sequência de erro de tamanho unitário. Por outro lado, transições paralelas diminuem a conectividade da treliça,

aumentando a distância entre sequências de erro de tamanho maior do que um. De um modo geral, os melhores códigos até agora encontrados apresentam transições paralelas [14].

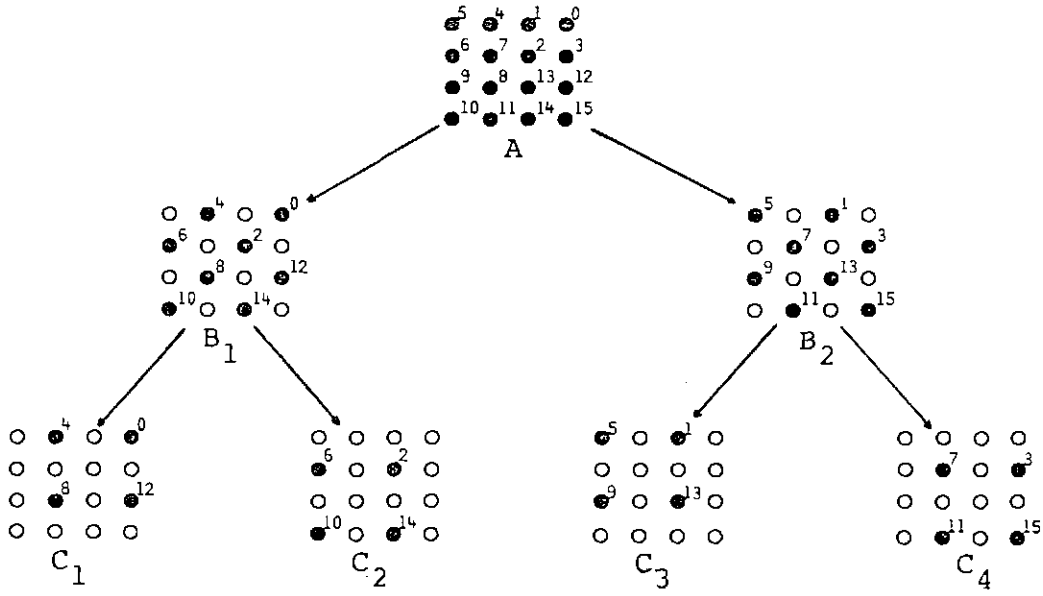
Esses códigos associam diretamente símbolos do canal às transições entre estados da treliça. Quando existem transições paralelas, isto é, quando o último nível da partição é formado por subconjuntos que contêm mais de um símbolo da constelação, esses subconjuntos podem ser considerados como super-símbolos de um código reduzido [14] cuja representação numa treliça não contém transições paralelas. A Figura 2.6 mostra um exemplo de um código de Ungerboeck para uma constelação 16-QAM que apresenta transições paralelas. Nesse caso apenas um bit de entrada é codificado, gerando dois bits na saída que são responsáveis pela escolha do subjunto Ci. Os dois bits restantes definem qual dos símbolos de Ci será efetivamente transmitido.

Para esses casos, onde transições paralelas estão presentes, a análise de desempenho dos códigos pode ser simplificada - pelo menos no que diz respeito ao cálculo da distância mínima - considerando-se apenas a avaliação no código reduzido e depois analisando-se a distância entre transições paralelas. Pode-se mostrar que [14]

$$d_{min} = \min (d_p, d_{red}) \qquad (2.1)$$

onde d_p é a distância entre transições paralelas e
d_{red} é a distância mínima do código reduzido.

Os resultados obtidos até o presente tanto na busca como na avaliação de códigos para utilização nesses esquemas mostram que um perfeito entendimento de sua estrutura algébrica está longe de ser obtido. Isso complica consideravelmente o trabalho de classificação e medida de desempenho, ainda mais se considerarmos que nem sempre os melhores códigos obtidos são superlineares, mesmo que o



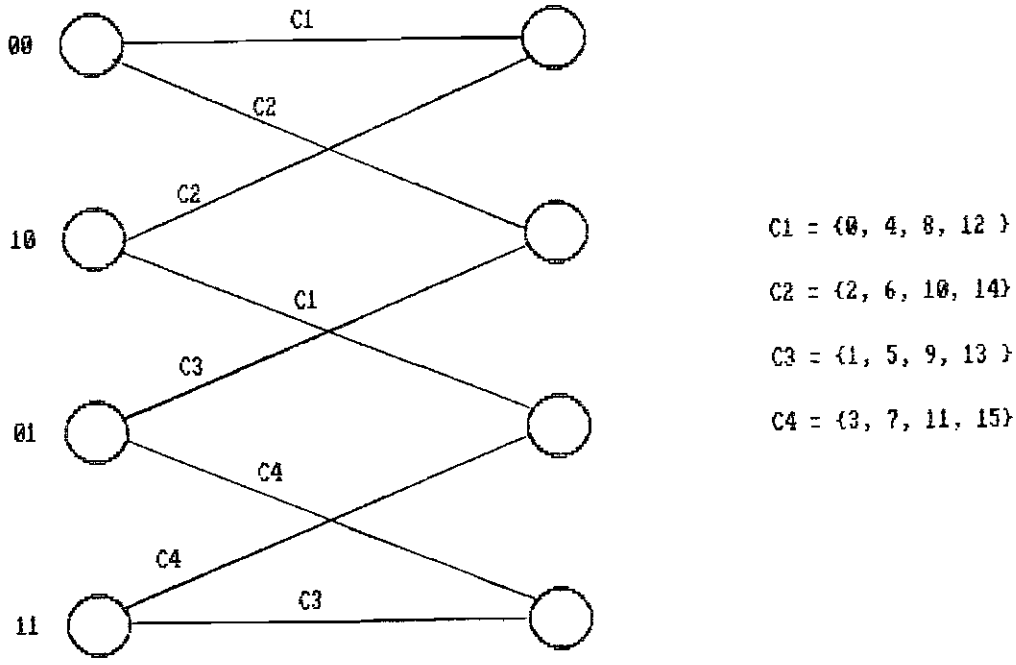


Fig. 2.6 - a) Partição do 16_QAM.
b) Diagrama de estado, mostrando o subconjunto associado à transição.

código reduzido associado o seja : as características geométricas da constelação podem destruir a superlinearidade.

Apesar disso, bons códigos para as constelações mais usuais em duas dimensões já foram identificados [14,15]. A análise desses códigos mostra que ganhos da ordem de 3-4 dB podem ser obtidos com baixa complexidade (codificadores com quatro e oito estados). Entretanto, para ganhos maiores - cerca de 6 dB - são necessários codificadores com 1024 estados, o que já é praticamente o limite do que pode ser manuseado por um decodificador de Viterbi.

2.6.2 - CÓDIGOS DE CUSACK-SAYEGH

O trabalho original de Ungerboeck não faz menção ao uso de códigos de bloco associados a esquemas de modulação e isso por dois motivos : de um modo geral ele considera os códigos convolucionais como tendo estrutura mais rica e flexível - embora alguns autores discordem categoricamente [4] - possibilitando melhor desempenho; também o uso de decodificação suave, que tem implementação imediata para os códigos convolucionais (através do algoritmo de Viterbi, por exemplo), não tem equivalente para códigos de bloco, isto é, as técnicas existentes implicam com freqüência em profundas mudanças no algoritmo de decodificação, com aumento considerável de complexidade.

Além disso, os códigos convolucionais, por sua própria natureza, fazem uso das transmissões já passadas para decidir sobre as futuras; isso é inteligentemente explorado nos esquemas de Ungerboeck, que utiliza essa memória para definir, a cada instante, qual subconjunto deve ser utilizado na transmissão seguinte. Assim, o subconjunto de símbolos "candidatos" a serem transmitidos pode ser escolhido de modo a maximizar a

distância entre sequências transmitidas. Não é claro como a técnica de partição de conjuntos pode ser aproveitada por códigos de bloco.

Os primeiros passos no sentido de viabilizar essa aplicação foram dados por Cusack [16], que combinou vários códigos de bloco com a constelação de sinais. Aqui novamente aparece a idéia original de Ungerboeck : o código mais poderoso protegerá o demodulador, impedindo-o de decidir errado sobre pontos próximos da constelação; por outro lado, o demodulador protegerá os códigos mais fracos, que deverão ser usados para símbolos mais distantes, vale dizer, símbolos do último nível da partição.

O esquema idealizado por Cusack, que pode ser visto na Figura 2.7, consiste em formar uma matriz onde as linhas são palavras dos vários códigos escolhidos. Às colunas da matriz são associados os símbolos da constelação, que serão transmitidos. Desse modo, se a constelação for composta de 2^m símbolos, deve haver **m** linhas na matriz, isto é, **m** códigos devem ser escolhidos para formar as linhas da matriz. Além disso, o comprimento de cada palavra de código (número de colunas da matriz) é 2^p, onde **p** é o número de vezes que a constelação foi particionada. Com isso é possível garantir que, com uma escolha adequada dos códigos, a distância mínima entre as "matrizes" seja de $\sqrt{(2)^p}.d$, onde **d** é a distância entre os símbolos da constelação.

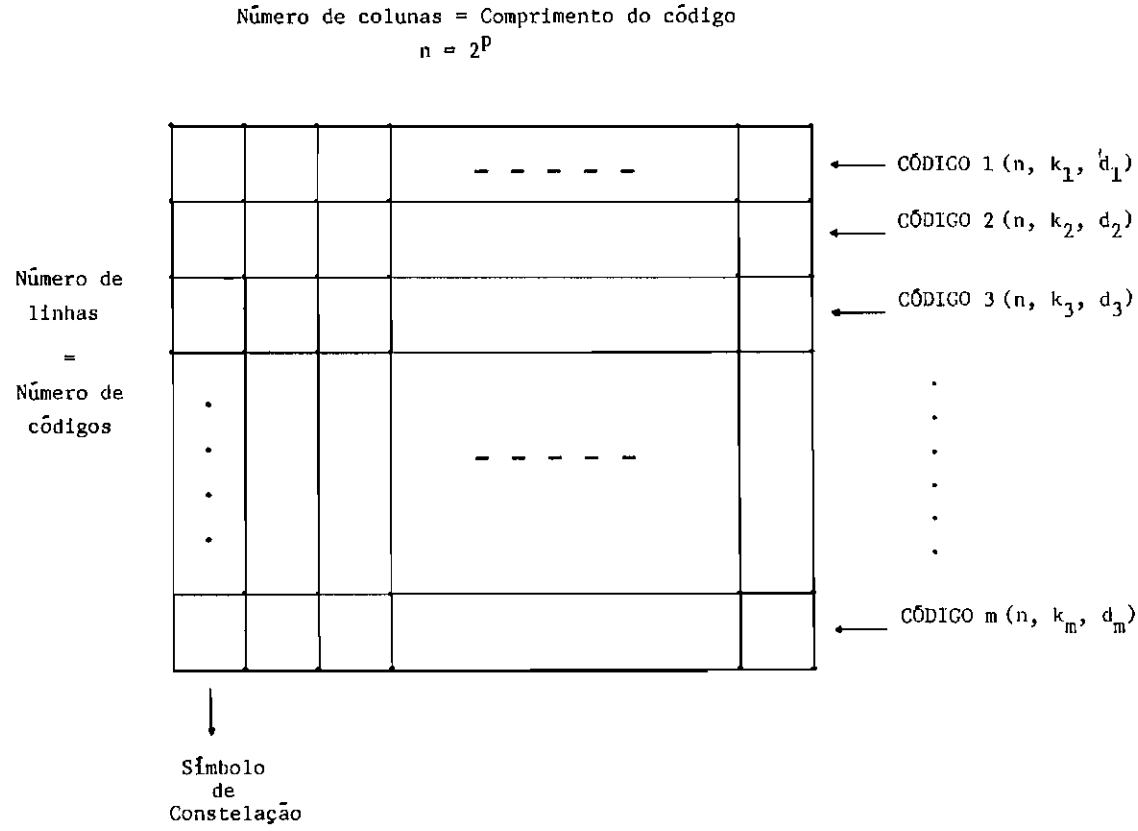
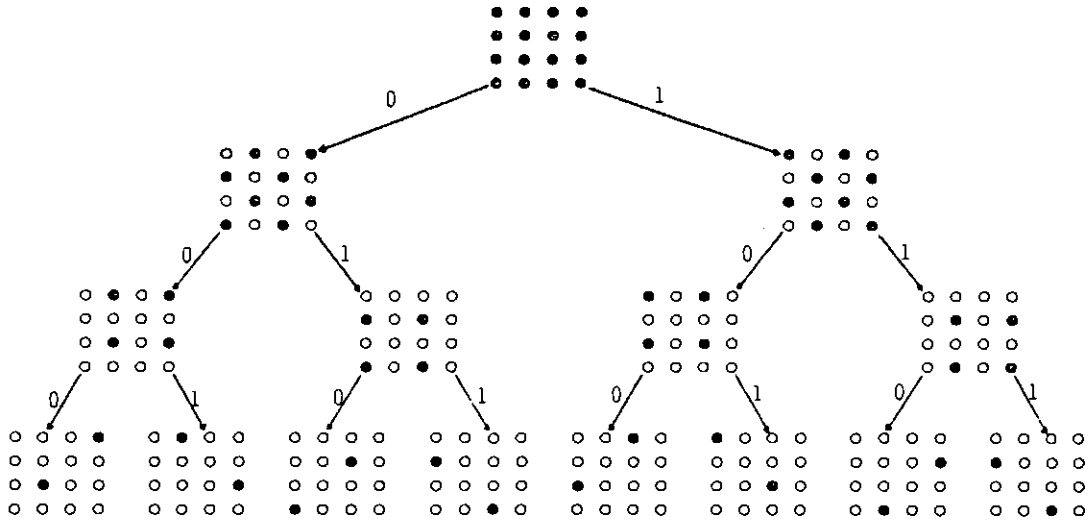


Fig 2.7 - Matriz de código para codificação de bloco.

Em seu trabalho original, Cusack usou uma sequência de códigos de Reed-Muller para as várias linhas da matriz que, devido a forma como são construídos, acomodam-se automaticamente a constelações de qualquer tamanho, garantindo a distância mínima desejada. Consideremos o exemplo abaixo (Figura 2.8), onde se aplica essa técnica de codificação a uma constelação 16_QAM particionada três vezes, isto é, p = 3.

Nesse exemplo, o comprimento de cada linha da matriz é $2^3 = 8$. Os códigos de Reed-Muller utilizados são R(0,3), R(1,3) e R(2,3), onde R(r,p) representa um código binário de Reed-Muller de comprimento 2^p e distância mínima de Hamming 2^{p-r}. Esses códigos em particular são bastante simples :



(a)

0	0	0	0	0	0	0	0
1	1	0	1	0	0	0	1
0	0	0	0	0	1	1	0
0	1	0	1	1	1	1	0

(b)

Fig 2.8 - Código de Sayegh.
a) Partição utilizada para o16_QAM.
b) Matriz de código resultante.
Bits de redundância sublinhados.

R(0,3) é o código de repetição;
R(1,3) é o código de Hamming extendido;
R(2,3) é o código com bit de paridade;
A última linha da matriz é composta de bits não codificados.

Na Figura 2.8b, está indicada a matriz resultante desse esquema de codificação com os bits reduntantes sublinhados.

Podemos notar nesse exemplo como funciona a associação entre o codificador e o modulador. Da forma como a partição de conjuntos foi feita, dois símbolos vizinhos na constelação (ainda não particionada) diferem no primeiro bit; símbolos vizinhos no primeiro nível da partição diferem pelo menos no segundo bit e assim por diante. Ora, as distâncias euclidianas crescem à medida que se avança na partição e, desse modo, os pontos dos últimos níveis estão mais protegidos (porque mais distantes uns dos outros) do que os dos primeiros níveis. Justifica-se assim, códigos com maior capacidade de correção para proteger os primeiros bits e códigos com menor capacidade (ou mesmo nenhuma codificação) para os últimos bits.

Mais tarde percebeu-se que essa técnica de codificação era um método para construir-se reticulados N-dimensionais densos tomando como base reticulados bi-dimensionais [12]. Essa abordagem unifica os vários códigos aqui estudados.

O trabalho de Sayegh [17] é uma generalização do esquema apresentado por Cusack. Nele utiliza-se a mesma matriz construída a partir de vários códigos, que já fora explorada por Cusack, mas são relaxadas as restrições quanto aos códigos ou número de dimensões.

Consideremos o mesmo exemplo da Figura 2.8a. Admitindo duas "matrizes" de códigos distintas (na realidade são palavras de um código N-dimensional) que difiram na primeira linha, a distância euclidiana D entre elas deverá ser, no mínimo,

$$D^2 = d_1$$

pois o código que forma a primeira linha tem distância mínima d_1, e os símbolos da constelação têm distância unitária.

Caso haja coincidência na primeira linha e as matrizes divirjam a partir da segunda, devemos ter

$$D^2 = 2d_2,$$

pois o segundo código tem distância mínima d_2 e os símbolos transmitidos têm distância euclidiana mínima igual a $\sqrt{2}$, equivalente ao primeiro nível da partição (observe que, como houve coincidência na primeira linha, todos os símbolos pertencem a um mesmo subconjunto do nível 1 da partição).

Prosseguindo nessa análise verifica-se que

$$D^2_{min} >= \min (d_1, 2d_2, 4d_3, 8d_4) \qquad (2.2).$$

Assim, o esquema proposto por Sayegh flexibiliza o anterior, pois permite utilizar um universo maior de códigos para a maximização de (2.2).

2.6.3 - CÓDIGOS DE IMAI-HIRAKAWA-YAMAGUCHI

Esses códigos são um desenvolvimento da idéia apresentada originalmente por Imai e Hirakawa [18], onde já apareciam, de forma ainda nebulosa, a partição de conjuntos de Ungerboeck e o esquema de codificação modulada de Sayegh.

Nesse trabalho os autores combinaram códigos de bloco e decisão abrupta com o esquema de partição de conjunto, obtendo ganhos elevados e baixa complexidade. Entretanto, o uso da métrica euclidiana em contraposição à de Hamming e o próprio conceito da partição de conjuntos não estavam perfeitamente claros.

Num trabalho posterior, Yamaguchi e Imai [19] retomaram a idéia usando códigos convolucionais e decisão suave, obtendo códigos que se comparam favoravelmente em relação aos de Ungerboeck, em termos de ganho e complexidade de decodificação.

CAPÍTULO 3

CÓDIGOS ANALISADOS

3.1 - INTRODUÇÃO

O presente capítulo traz uma análise detalhada do desempenho de alguns esquemas de codificação modulada. Serão estudados códigos convolucionais e de bloco associados aos esquemas de Ungerboeck e Sayegh anteriormente descritos. Esses códigos serão em seguida concatenados com um código de Reed-Solomon RS(255,223), que é normalmente utilizado em técnicas convencionais de codificação. Nesse estudo uma constelação de sinais 16_QAM será utilizada para modulação dos bits a serem transmitidos.

3.2 - A CONSTELAÇÃO DE SINAIS

Conforme pode ser verificado na Figura 3.1 [20], as constelações retangulares são mais eficientes em termos de potência do que constelações do tipo M-PSK, isto é, com amplitude constante e fase múltipla.

Apesar dessa maior eficiência, os sistemas M-PSK são geralmente preferidos em aplicações onde os sinais devam ser processados por sistemas que apresentam não-linearidades, tais como as válvulas amplificadoras no estágio de saída de sistemas de transmissão via satélite. A razão disso é a envoltória constante que a sequência de sinais das constelações M-PSK apresenta, que minimiza as distorções causadas por essas não-linearidades. Os ganhos

de codificação que podem ser obtidos com constelações retangulares nesse ambiente são mínimos [15].

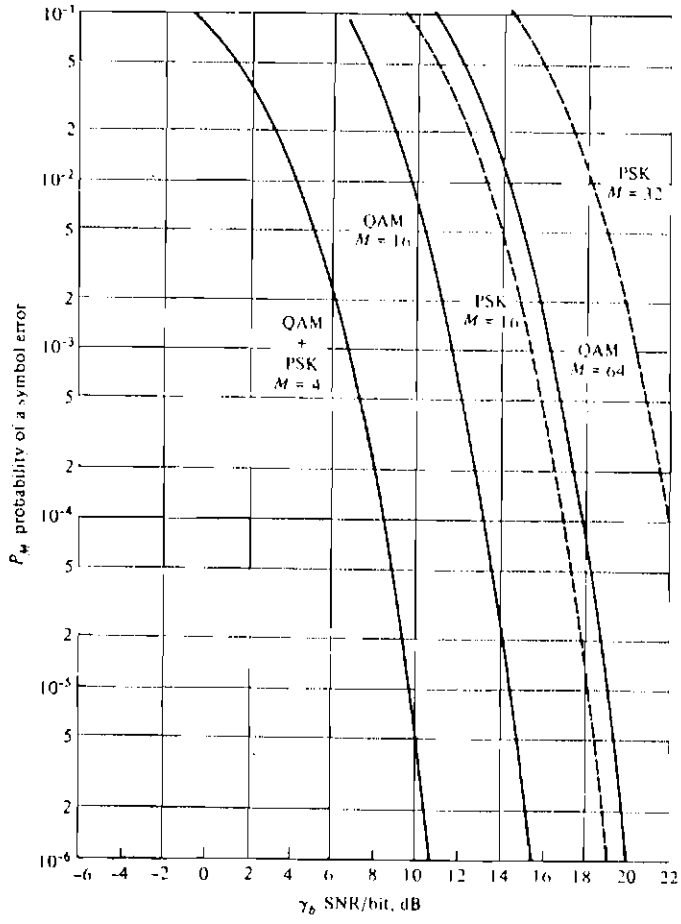


Fig 3.1 - Probabilidade de erro de símbolo para constelações QAM e PSK.

Por outro lado, em sistemas que não apresentam tais limitações, as constelações retangulares têm maior destaque, como é o caso das transmissões digitais através de redes telefônicas [12].

Neste trabalho limitaremos nossa atenção a constelações retangulares, em particular a de 16 pontos. A partição com a qual se vai trabalhar é a da Figura 3.2 adiante. Nela estão indicados os níveis da partição e as distâncias entre pontos em cada nível.

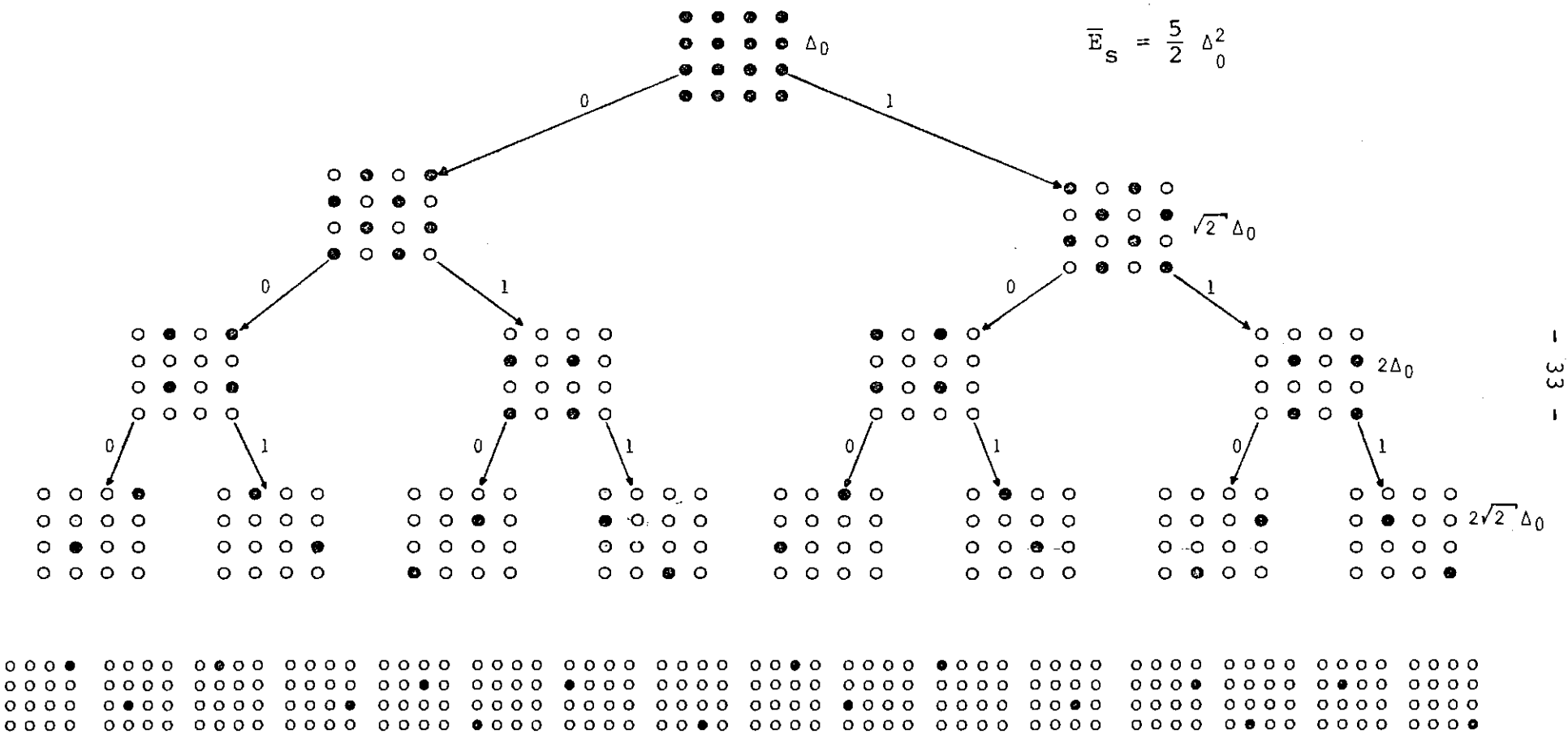


Fig. 3.2 - Partição para a constelação 16_QAM.

3.3 - CÓDIGOS DE UNGERBOECK

Utilizando busca computacional, auxiliada por alguns limitantes eficientes, Ungerboeck identificou códigos convolucionais ótimos (no sentido de máxima distância mínima) para a consteleção 16_QAM [11]. Entre eles selecionamos dois para este estudo, escolhidos em função de um compromisso entre capacidade de proteção e complexidade. A Figura 3.3 mostra um diagrama de blocos com uma implementação possível para os codificadores escolhidos.

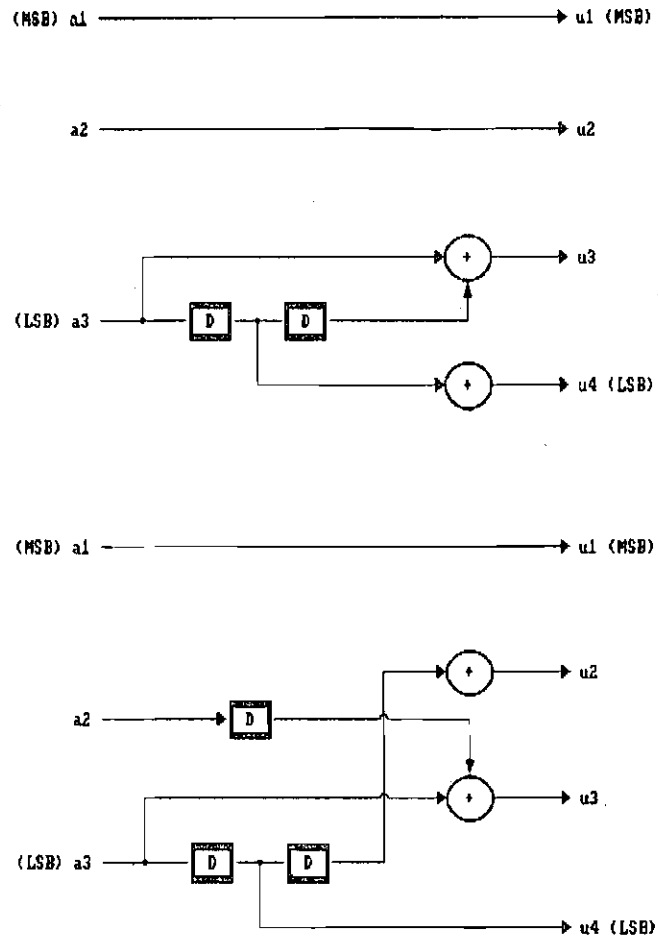


Fig. 3.3 - Implementação do codificador convolucional.
a) quatro estados
b) oito estados.

Baseando-se no fato de que grande parte do ganho atingível já é obtido com uma constelação expandida por um fator dois, os códigos escolhidos têm taxa R = 3/4. Além disso, pode-se perceber que um ou dois dentre os três bits de entrada aparecem inalterados na saída, o que caracteriza uma treliça com transições paralelas.

Para poder comparar o desempenho desses códigos com o que seria normalmente obtido num sistema sem codificação podemos utilizar uma constelação 8_PSK, que transmite 3 bits de informação por símbolo, como o nosso esquema codificado, e que tenha uma energia média por símbolo transmitido idêntica à da constelação 16_QAM. A Figura 3.4 mostra essa constelação e seus principais parâmetros.

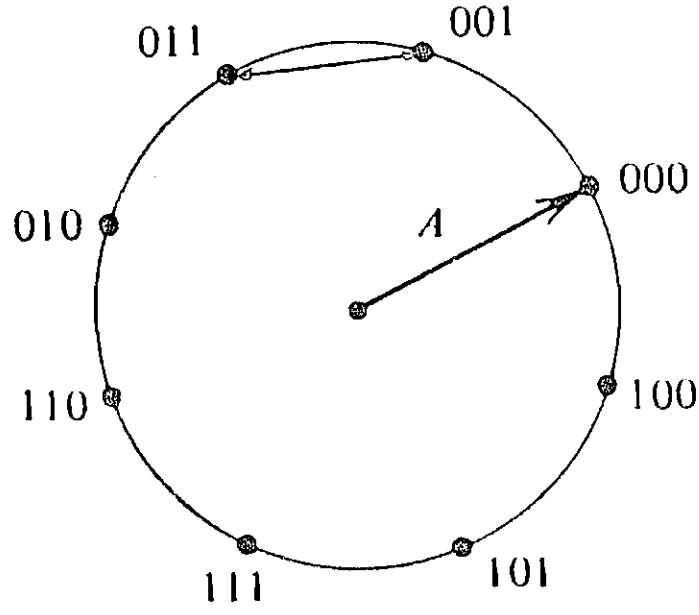


Fig 3.4 - Constelação 8_PSK com mapeamento de Gray, usada na avaliação dos códigos convolucionais.

A fim de facilitar referências futuras designaremos o esquema indicado na Figura 3.3a como Código I e o da Figura 3.3b como Código II.

3.3.1 - O CÓDIGO I

A Figura 3.5 mostra a partição que se está utilizando nesse código. Como se pode notar, a constelação foi dividida duas vezes, restando no último nível quatro subconjuntos contendo quatro pontos cada um.

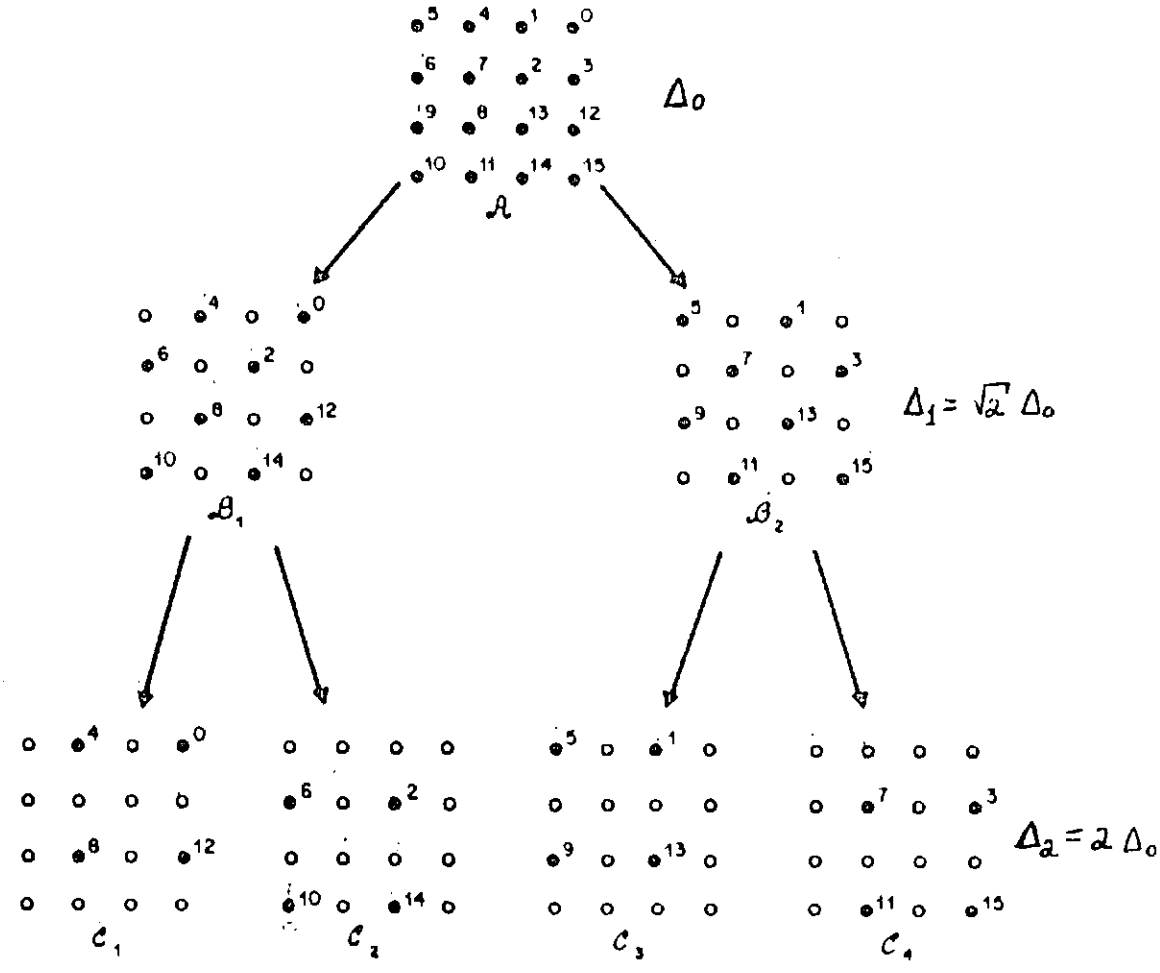


Fig. 3.5 - Partição usada para o código I.

A treliça que representa o código I está esquematizada na Figura 3.6. Aqui os símbolos Ci indicados nas transições representam os subconjuntos do último nível da partição, ou, em outras palavras, os super-símbolos do código reduzido. Desse modo, cada transição mostrada representa na realidade quatro transições paralelas, uma para cada um dos pontos do subconjunto Ci.

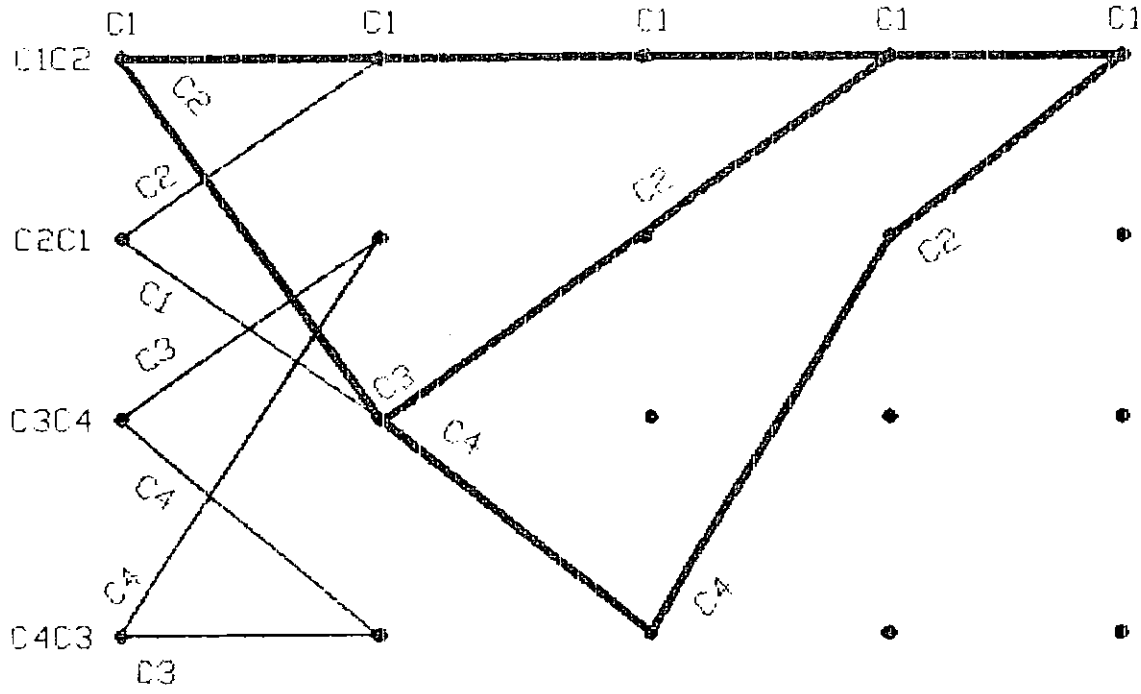


Fig. 3.6 - Representação do código I por meio de uma treliça.

Como se pode perceber esse código apresenta quatro estados cuja designação é feita com o auxílio dos subconjuntos associados às transições que partem de cada estado. A treliça mostra também duas trajetórias separadas pela distância mínima do código reduzido. Desse modo, já é possível obter o valor da distância mínima do código original por meio da Equação 2.1 da Seção 2.6.1. Para esse caso temos

$d_p = 2\Delta_0$ como pode ser visto na Figura 3.5 e

$$d_{red} = d\,(C_1C_1C_1,\ C_2C_3C_2) = \sqrt{\Delta_1^2 + \Delta_0^2 + \Delta_1^2} = \Delta_0\sqrt{5}$$

conforme a figura 3.6

Temos portanto $d_{min} = \min\,(d_p,\ d_{red}) = 2\Delta_0$.

Com base nesse resultado pode-se obter algumas estimativas para o desempenho desse código.

3.3.1.1 - GANHO ASSINTÓTICO

Entende-se como ganho de codificação **G** a diferença em dB entre as relações sinal-ruído SNR necessárias para se atingir a mesma probabilidade de erro de símbolo Ps , no sistema codificado e no não-codificado. Este último deve ser adequadamente definido a fim de garantir uma comparação em bases iguais.

O ganho assintótico é definido como sendo este ganho de codificação quando a relação SNR tende para infinito. Pode-se mostrar que

$$Gass = G\Big|_{SNR\to\infty} = 10.\log\ \frac{d^2min}{d^2ref} \qquad (3.1)$$

Para esse código a constelação de referência será aquela da Figura 3.4 anteriormente descrita. Assim temos

$$Gass = 10.\log\ \frac{(2\Delta_0)^2/\frac{5}{2}\ \Delta_0^2}{(0,765A)^2\ /A^2} = 10.\log\frac{8}{5x(0,765)^2} = 4,37dB$$

3.3.1.2 - PROBABILIDADE DE ERRO

Para obter-se um limitante inferior para a probabilidade de erro de bit, uma série de hipóteses simplificadoras devem ser feitas:

a - consideraremos apenas os erros mais prováveis, ou seja os que ocorrem entre sequências separadas pela distância mínima.

Como a distância mínima separa transições paralelas da treliça - e por isso eventos de erro de tamanho unitário - podemos já estimar a probabilidade de erro de símbolo como sendo [21]

$$P_s > Q(\frac{dmin}{2\sigma_0}) = Q\ (\frac{\sqrt{2E_s/5}}{\sigma_0}) = Q(1{,}55\ \sqrt{\frac{Eb}{N_0}})$$

Como as transições paralelas existem devido aos bits não-codificados, cada erro incide diretamente sobre bits de informação, que se encontram no último nível da partição. Observemos agora que este nível é equivalente a uma constelação 4_PSK. De um modo geral portanto, assumiremos uma segunda hipótese :

b - cada erro de símbolo acarreta um bit de informação errado

Com isso podemos finalmente escrever uma primeira estimativa de desempenho para este código I, que é tanto melhor quanto maior for a relação sinal-ruído com a qual se opera :

$$P_b = Q(1{,}55\ \sqrt{\frac{Eb}{N_0}}) \qquad (3.2)$$

Finalmente, poderíamos utilizar esse resultado para avaliar o desempenho do código I concatenado ao código RS(255,223). Para isso bastaria utilizar as Expressões (1.2) ou (1.3) da Seção 1.3, considerando a probabilidade de erro de símbolo Ps (aqui para o código concatenado) como sendo dada por

$$P_s = (1-P_b)^8 \qquad (3.3)$$

onde P_b seria obtida a partir de (3.2).

3.3.2 - O CÓDIGO II

Para esse código a constelação foi particionada como indicado na Figura 3.7. Podemos verificar que a partição vai um nível além daquela utilizada no primeiro código e, desse modo, obtemos no final oito subconjuntos contendo dois pontos cada um.

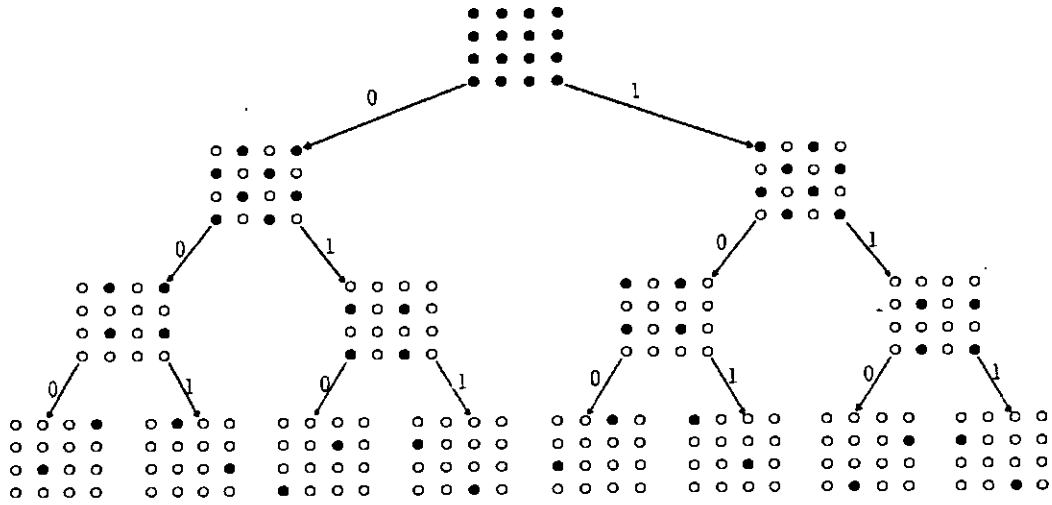


Fig. 3.7 - Partição usada para o código II.

A treliça para o código reduzido, isto é, sem as transições paralelas, está mostrada na Figura 3.8 a seguir. Esse é um código com oito estados, designados pelos subconjuntos associados às transições em cada nó da treliça, da mesma forma como foi feito para o código I. Também estão indicadas duas trajetórias separadas pela distância mínima do código reduzido.

De maneira análoga ao caso anterior, podemos utilizar as expressões já apresentadas para obter a distância mínima do código II. Temos então

$d_p = 2\sqrt{2} \cdot \Delta_0$ conforme indicado na Figura 3.7 e

$d_{red} = d\,(D_1D_1D_1D_1\ ,\ D_4D_7D_1D_3) = \sqrt{5}.\Delta_0$

como mostrado na Figura 3.8.

Temos assim $d_{min} = \min\,(d_p,\ d_{red}) = \sqrt{5}\,.\,\Delta_0$. Observemos que, nesse esquema, a distância mínima está amarrada pelo codificador e não pela constelação, como no caso anterior.

Podemos notar ainda que a treliça desse código II é muito mais conexa que a do código anterior, em função do menor número de transições paralelas. Para esse código, existem quatro trajetórias que distam dmin de uma dada trajetória de referência (essas trajetórias são $D_3\,D_6\,D_4$, $D_4D_8\,D_4$, $D_4D_7D_1D_3$ e $D_3D_5D_1D_3$, para a referência $D_1D_1D_1D_1$).

Com base nessas observações podemos imaginar que, para relações sinal-ruído baixas, onde eventos de erro associados a trajetórias de grande comprimento têm maior probabilidade de ocorrer, o código I pode ter melhor desempenho do que o código II, devido a essa diferença na

conectividade das treliças. Isso deve ocorrer apesar de o código II ter, como veremos a seguir, um ganho assintótico maior.

3.3.2.1 - GANHO ASSINTÓTICO

Aqui também utilizaremos a mesma constelação 8_PSK como sistema de referência para avaliação do código. De maneira análoga ao cálculo feito anteriormente para o código I temos

$$G_{ass} = 10\log \frac{(\Delta_0 \sqrt{5})^2/\frac{5}{2}\Delta_0^2}{(0,765A)^2/A^2} = 10\log \frac{5 \times 2}{5x(0,765)^2} = 5,34dB.$$

3.3.2.2 - PROBABILIDADE DE ERRO

A estimativa de desempenho do código II não é tão imediata como no caso anterior, uma vez que agora os eventos de erro com distância mínima não são unitários. A análise, entretanto, segue o mesmo caminho. Assumiremos inicialmente a mesma hipótese simplificadora :

a - consideraremos que os erros ocorrem apenas entre sequências separadas pela distância mínima.

A partir daí podemos calcular a probabilidade de um evento de erro, isto é, a probabilidade do decodificador considerar como correta uma trajetória vizinha da originalmente transmitida :

$$P_{ev} >= Q\left(\frac{dmin}{2\sigma_0}\right) = \left(\frac{\sqrt{2Es}}{2\sigma_0}\right) = Q\left(1,73\sqrt{\frac{Eb}{N_0}}\right).$$

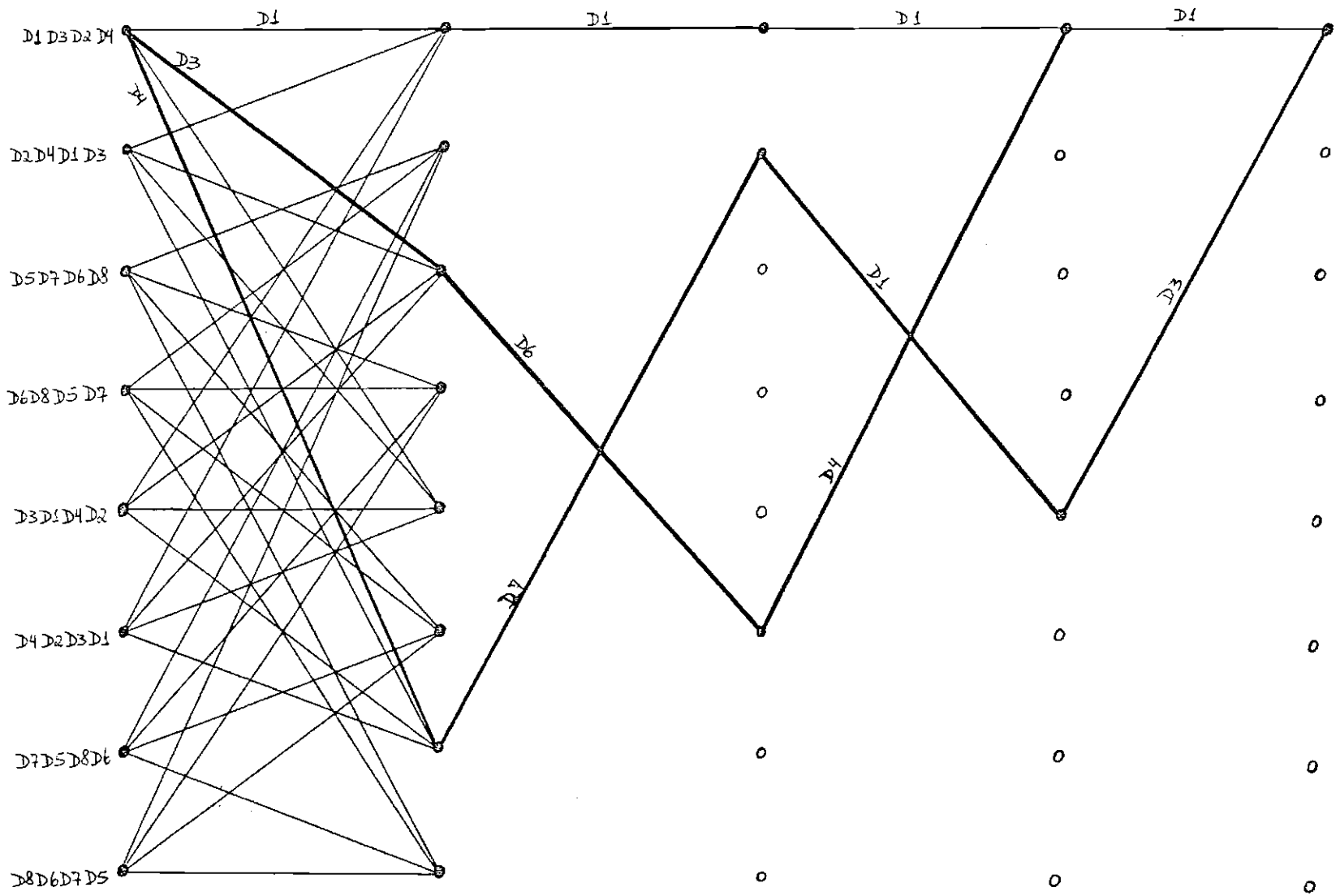


Fig. 3.8 - Treliça para o Código II.

A cada evento de erro atribuiremos três símbolos errados, considerando que dentre as quatro trajetórias vizinhas existentes, duas delas (as de comprimento quatro) contêm o ramo D1, que será otimisticamente computado como um acerto. Os demais ramos necessariamente acarretam erro de símbolo. Assim, chegamos a um limitante da probabilidade de erro de símbolo :

$$P_s >= 3.Q(1,73 . \sqrt{\frac{Eb}{N_0}})$$

Como o código não é superlinear o cálculo da probabilidade de erro de bit não é muito simples, envolvendo médias sobre todos os pares de trajetórias. Embora existam técnicas para estimá-la [14], esse resultado será obtido mais adiante por meio de simulação.

3.4 - CÓDIGOS DE SAYEGH

Uma vez que está definida a constelação com a qual se pretende trabalhar - 16_QAM - o número de linhas da "matriz" do código, e como conseqüência o número de códigos de bloco a serem empregados, fica automaticamente estabalecido: teremos $\log_2 16 = 4$ linhas (códigos) na matriz.

Resta agora escolher o comprimento das palavras do código, que equivale a definir o número de colunas da matriz, e selecionar os códigos propriamente, em função da taxa que se quer transmitir e do desempenho - associado à capacidade de correção dos códigos - que se deseja obter.

De um modo geral é possível obter bons desempenhos - ganhos da ordem de 3 dB - e baixa complexidade, utilizando códigos bem simples, de repetição e de paridade [17]. Para obter-se desempenhos superiores, códigos mais poderosos devem ser usados. Entre os existentes, os códigos BCH oferecem a vantagem de uma estrutura já bastante estudada e conhecida, facilitando sua aplicação em esquemas de modulação codificada.

3.4.1 - CONSTELAÇÕES DE REFERÊNCIA

Para os códigos de Ungerboeck já analisados, que tinham taxa R = 3/4, a constelação de referência que se usou para fins de avaliação de ganho era o 8_PSK. Ela apresentava a mesma eficiência espectral desses códigos, ou seja, 3 bits de informação por símbolo transmitido.

Aqui, a referência utilizada vai depender da taxa com que se transmite o código. Assim, uma constelação 16_QAM codificada com taxa R = 3/4 deve usar o 8_PSK ou o 8_AMPM como referência; para taxa R = 1/2, usa-se o QPSK e para taxa R = 1/4, usa-se o BPSK. Outras taxas, tais como R = 1/3 ou 2/3 por exemplo, que não admitem uma correspondência imediata, podem ser avaliadas através de esquemas mistos com mais de uma transmissão. Em qualquer caso, a energia média por símbolo deve ser a mesma nos dois sistemas.

3.4.2 - OS CÓDIGOS

A partir de tabelas de códigos BCH existentes [23] é possível obter combinações eficientes em termos de taxa e distância mínima do código.

Como já foi verificado anteriormente, a distância euclidiana mínima entre duas palavras de um código de Sayegh, admitindo-se a constelação 16_QAM, pode ser escrita como

$$d^2min = \min (d_1, 2d_2, 4d_3, 8d_4),$$

onde d_{min} é a distância mínima (Hamming) do código de bloco utilizado na i-ésima linha da matriz.

Dessa forma, podemos avaliar d por inspeção dos códigos utilizados. Consideremos, por exemplo, os códigos BCH de comprimento n = 15. São eles

(15, 15, 1) - sem codificação
(15, 14, 2) - código com bit de paridade
(15, 11, 3) - código de Hamming
(15, 7, 5) - código corretor de 2 erros
(15, 5, 7) - código corretor de 3 erros
(15, 1, 15) - código de repetição.

A partir da combinação de quatro desses códigos, dentro do esquema proposto por Sayegh, obtemos um código para a constelação 16_QAM. Obviamente, as combinações mais eficientes reservam códigos mais poderosos para as linhas iniciais da matriz. Assim, podemos criar um total de $C_4^6 = 15$ diferentes esquemas de codificação, com diferentes taxas e capacidades de correção.

É interessente notar que se pode construir códigos equivalentes em termos de taxa de transmissão e distância euclidiana mínima, mas com distribuição de distâncias entre as palavras muito diferente. Como exemplo, consideremos os seguintes códigos :

CÓDIGO I	CÓDIGO II	CÓDIGO III
(15, 5, 7)-- 7	(15, 5, 7)-- 7	(15, 5, 7)-- 7
(15, 7, 5)-- 10	(15, 7, 5)-- 10	(15, 7, 5)-- 10
(15, 11, 3)-- 12	(15, 11, 3)-- 12	(15, 14, 2)-- 8
(15, 14, 2)-- 16	(15, 15, 1)-- 8	(15, 15, 1)-- 8
d^2min=7 R=37/60	d^2min=7 R = 38/60	d^2min=7 R = 41/60

Podemos notar que todos os esquemas têm taxa aproximadamente igual a R = 2/3 e para todos eles o ganho assintótico é o mesmo - pois todos apresentam o mesmo dmin. Entretanto o código I provavelmente apresenta um desempenho geral superior, pois as distâncias entre palavras de código são maiores, como pode ser verificado através da distribuição das demais componentes de distância desse código. Assim, com um sacrifício muito pequeno da taxa (o código I transmite 4 bits de informação a menos por palavra do que o código III) obtém-se um código mais robusto.

3.4.3 - GANHO ASSINTÓTICO

Para analisarmos o desempenho desses esquemas de codificação convêm calcular o ganho assintótico de algumas das combinações de códigos em relação à constelação de referência adequada. Desse modo conseguimos um limitante inferior desse desempenho, que indica o máximo ganho atingível, nas situações em que a relação sinal-ruído for muito alta.

Usamos, para as linhas da matriz, códigos BCH de comprimento 15 e 31, com diversas taxas, a fim de comparar os vários parâmetros que podem influenciar o desempenho dos códigos de Sayegh. Os resultados obtidos estão resumidos na Tabela 3.1 a seguir.

TABELA 3.1

GANHO ASSINTÓTICO PARA ESQUEMAS
DE MODULAÇÃO CODIFICADA DE BLOCO.

TAXA	N = 15		N = 31	
	d_{min}	G_{ass}(dB)	d_{min}	G_{ass}(dB)
1/4	—	—	30	4.8
1/2	12	3.8	15	4.8
3/4 (8_PSK)	6	6.1	8	7.8
3/4 (8_AMPM)	6	4.8	8	6.0

As constelações de referência para R = 1/4 e 1/2 foram o BPSK e o QPSK, respectivamente. Para a taxa R = 3/4, usou-se a constelação indicada na própria tabela.

Os códigos escolhidos para cada um dos exemplos estão indicados a seguir na Tabela 3.2, juntamente com suas componentes de distância e a taxa real obtida.

TABELA 3.2

ESQUEMAS DE SAYEGH USADOS PARA CÁLCULO DE GANHO ASSINTÓTICO

TAXA	N = 15	N = 31
1/4	[redacted]	(31, 1, 31) - 31 (31, 6, 15) - 30 (31, 11, 11) - 44 (31, 16, 7) - 56 R = 34/124
1/2	(15, 1, 15) - 15 (15, 5, 7) - 14 (15, 11, 3) - 12 (15, 14, 2) - 16 R = 31/60	(31, 6, 15) - 15 (31, 11, 11) - 22 (31, 16, 7) - 28 (31, 30, 2) - 16 R = 63/124
3/4	(15, 5, 7) - 7 (15, 11, 3) - 6 (15, 14, 2) - 8 (15, 15, 1) - 8 R = 3/4	(31, 11, 11) - 11 (31, 21, 5) - 10 (31, 26, 3) - 12 (31, 31, 8) - 8 R = 89/124

(n, k, d_{min}) - d^2

3.4.4 - A DECODIFICAÇÃO

A fim de se extrair o máximo da capacidade do código, deve-se implementar um esquema de decodificação suave, a exemplo do que se faz para códigos convolucionais. Essa, geralmente, é a etapa que torna muito mais atraentes os códigos convolucionais em relação aos de bloco: a simplicidade com que se adapta o decodificador de Viterbi para diferentes métricas.

Um grande esforço têm sido feito no sentido de construir-se decodificadores para códigos de bloco que implementem decodificação suave sem incorrer em exagerado aumento na complexidade desse processo de decodificação. No caso particular de modulação codificada, Farrell e Willians [24] apresentaram algumas idéias que simplificam o decodificador e se adaptam aos esquemas aqui desenvolvidos.

Para constelações retangulares, a idéia básica é codificar (e posteriormente decodificar) cada dimensão separadamente por meio de constelações de sinais modulados em amplitude. Desse modo, os esquemas que utilizam constelações 16_QAM passariam a efetuar a modulação codificada com duas constelações 4_AM. O símbolo resultante da dupla codificação seria transmitido pela constelação de 16 pontos, conforme indicado na Figura 3.9 abaixo.

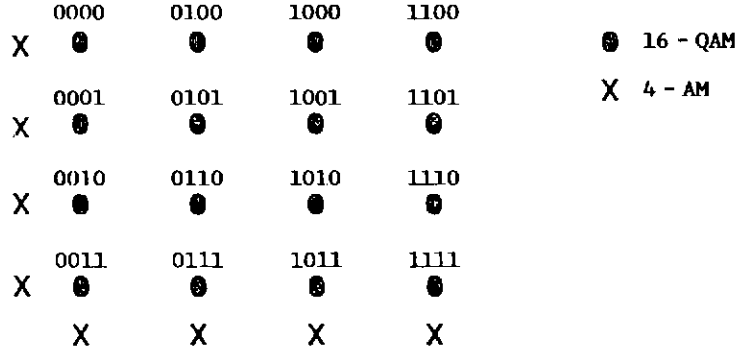


Fig. 3.9 - Constelação 16_QAM construída a partir de duas constelações 4_AM.

Na demodulação, as duas dimensões seriam separadas e cada uma, por sua vez, apresentada ao decodificador. A vantagem desse processo é diminuir o número de combinações possíveis entre os sinais, diminuindo então o espaço de busca entre palavras de código.

Consideremos, para exemplificar esse processo, um dos códigos apresentados anteriormente na Seção 3.4.3 :

CÓDIGO II

(15, 5, 7)-- 7
(15, 7, 5)-- 10
(15, 11, 3)-- 12
(15, 15, 1)-- 8

$d^2_{min} = 7$ R = 38/60

Analisemos agora um código para a mesma constelação, usando a modulação em duas etapas. Notemos inicialmente que a partição de de uma constelação 4_AM gera subconjuntos com distâncias intra-símbolos que crescem mais depressa do que numa constelação 16_QAM. Para o 4_AM vale a seguinte relação :

$$d^2_{min} = \min (d_1, 4d_2)$$

onde d_1 e d_2 são as distâncias mínimas dos códigos utilizados na primeira e segunda linha da matriz, respectivamente.

Podemos, então, gerar o seguinte código para cada dimensão:

(15, 5, 7)-- 7
(15, 14, 2)-- 8

$d^2_{min} = 7$ R = 19/30

Combinando as duas dimensões obtemos um código com a mesma taxa e distância mínima do Código II apresentado.

A grande vantagem desse esquema é a complexidade de decodificação. Supondo que fosse utilizado um decodificador de correlação, isto é, que a cada palavra recebida calcula-se sua distância a todas as palavras do código e tomasse uma decisão baseado nisso, o número de computações necessário em ambos os esquemas seria

$$1x \text{ código para } 16\_QAM = 1 \times 2^{38} \sim 2{,}7 \times 10^{11}$$
$$2x \text{ código para } 4\_AM = 2 \times 2^{19} \sim 1 \times 10^{6}.$$

Como se pode verificar, há um ganho da ordem de 100.000 vezes na quantidade de cálculo de um esquema para outro.

A desvantagem associada a esse método simplificado de codificação restringe-se a uma menor flexibilidade na escolha de taxa e distância mínima. Não é claro que o desempenho desse esquema seja pior que o original devido a distribuição de suas componentes de distância, porque apesar de ter componentes menores, o número de palavras no livro do código (para o 4_AM) também é menor, ou seja, o número de vizinhos é menor. A palavra final sobre o desempenho fica na dependência de se conhecer melhor a estrutura desses esquemas de codificação.

CAPÍTULO 4

SIMULAÇÃO DOS CÓDIGOS

4.1 - INTRODUÇÃO

A fim de caracterizar o desempenho dos códigos convolucionais foi desenvolvido um programa de computador que simula o algoritmo de decodificação de Viterbi. O programa implementa também um algoritmo para cálculo da probabilidade de erro de símbolo Ps, gerada na saída do decodificador interno, quando se está analisando o código concatenado.

Esse capítulo é dedicado à análise dos principais parâmetros utilizados nessa simulação, bem como a uma descrição sumária do programa em si. Uma listagem do programa para decodificação de Viterbi encontra-se no Apêndice A.

4.2 - A ESTRUTURA DO PROGRAMA

O programa implementa a simulação pelo método de Monte_Carlo. Uma sequência aleatória de bits de informação é gerada (simulando a fonte) e em seguida codificada. Aos símbolos resultantes é acrescentado ruído Gaussiano, gerado de maneira a se obter a relação Eb/N0 desejada. Este sinal é quantizado e decodificado através do algoritmo de Viterbi, sendo então comparado com a sequência originalmente transmitida para avaliação da probabilidade de erro.

O programa foi desenvolvido em micro-computador tipo IBM PC usando Turbo Pascal como linguagem para a simulação. Ele contém basicamente cinco grandes módulos :

. Módulo CODIFICADOR

A partir da estrutura do codificador gera uma tabela que descreve a treliça do código, armazenando os estados, as transições entre eles e os bits de informação associados.

. Módulo CANAL

Esse módulo é responsável por simular a saída do canal. Ele gera, armazena e codifica a sequência de entrada, adiciona o ruído e quantiza o sinal recebido.

. Módulo MÉTRICA

Gera a distância euclidiana, que será usada como métrica, entre os pontos da constelação e os possíveis sinais recebidos, estes últimos definidos pelo esquema de quantização do módulo anterior.

. Módulo VITERBI

É o decodificador propriamente dito. Para cada sinal recebido calcula a métrica de cada ramo e escolhe a trajetória sobrevivente em cada um dos estados. Após o comprimento de decodificação (transiente inicial), inicia a decodificação dos bits.

. Módulo COMPARADOR

Baseado na sequência original transmitida e naquela decodificada pelo algoritmo de Viterbi compara os bits e faz a estatística dos erros.

Existe ainda um sexto módulo, opcional, que é utilizado em substituição ao módulo COMPARADOR quando se deseja considerar o código concatenado. Nesse caso, as saídas do decodificador são agrupadas convenientemente em símbolos do corpo de Galois $GF(2^m)$, onde opera o código externo.

O encadeamenteo lógico dos módulos é sequencial, a exceção da tabela gerada pelo codificador que é utilizada tanto na geração de sequências codificadas (módulo CANAL) como no algoritmo de busca na treliça (módulo VITERBI).

4.2.1 - CODIFICADOR

Esse módulo gera e armazena uma treliça onde estão representados todos os nós (estados) e ramos (transições) do código.

A cada estado estão associadas 2^k transições, isto é, 2^k ramos chegam a esse estado. Cada um desses ramos traz informações sobre o estado "de partida" da transição, a saída (binária) associada à transição e a sequência de entrada que a gerou.

Dese modo, qualquer código representável por uma treliça pode ser simulado pelo programa, seja ele sistemático ou não.

4.2.2 - CANAL

Aqui são geradas as sequências de entrada que serão codificadas a seguir, com o auxílio do módulo codificador. Uma vez que o código não é superlinear, não é possível avaliar seu desempenho enviando uma única sequência - uma sequência só de zeros, por exemplo - e analisando os erros cometidos. Nesse caso em particular, devido a constelação escolhida e a designação dos pontos que foi definida (ver Figura 3.2), a sequência toda nula é extremamente privilegiada em relação a uma sequência típica. Dessa forma deve-se fazer com que a sequência de entrada seja aleatória, o que torna necessário armazená-la para comparação posterior.

Após a codificação, a sequência binária resultante é mapeada na constelação de acordo com o esquema de partição de conjuntos de Ungerboeck, gerando uma sequência de símbolos que é enviada através do canal.

Esse "enviar através do canal" consiste em adicionar ruído gaussiano branco à sequência de símbolos que foi gerada. A obtenção dessas amostras gaussianas e o processo de quantização do sinal estão descritos a seguir.

4.2.2.1 - O RUÍDO GAUSSIANO

Há várias técnicas de simulação para gerar sequências de amostras independentes com distribuição normal. Duas delas foram implementadas a fim de verificar, entre outras coisas, a validade estatística dos resultados obtidos. Ambas utilizam um gerador de números aleatórios com distribuição uniforme interno ao próprio Turbo Pascal.

A primeira técnica empregada faz uso do teorema do limite central e consiste em somar muitas variáveis aleatórias i.i.d (independentes e identicamente distribuídas) para obter uma variável gaussiana. A média é posteriormente ajustada para zero e a variância para um, da seguinte forma:

$$Y = \sum_{i=1}^{N} X_i ,$$

onde X_i são v.a. uniformemente distribuídas entre [0,1].

$$Y_m = E(Y) = \sum_{i=1}^{N} E(X_i) = N.E(X_1) = N/2$$

$$\sigma_m^2 = var(Y) = \sum_{i=1}^{N} var(X_i) = N.var(X_1) = N/12.$$

Pelo teorema do limite central, Y tende a uma gaussiana quando N tende a infinito. Então

$$Y' = (12/N)^{1/2} \cdot (Y-Ym)$$

tende a uma gaussiana padrão N (0,1).

Na prática já se obtém uma boa aproximação da gaussiana com N > 10. Nas simulações efetuadas utilizou-se N = 19, que foi um compromisso entre precisão e tempo computacional.

O segundo método usa a função de distribuição da v.a. para gerar as amostras. Como essa função é monótona crescente e tem imagem definida no intervalo [0,1], seleciona-se aleatoriamente (distribuição uniforme) pontos nesse intervalo e usa-se a função inversa para gerar as amostras desejadas. Como a distribuicão gaussiana não pode ser expressa de forma fechada, recorremos ao artifício de utilizar uma v.a. com distribuição de Rayleigh, que pode ser escrita como

$$R^2 = (X_1^2 + X_2^2)$$

onde R - v.a. com distribuição de Rayleigh e X_1, X_2 - v.a. com distribuição normal.

A Figura 4.1 ilustra a relação entre as v.a. A partir dela podemos notar que X_1 e X_2 podem ser representadas como

$$X_1 = R \cos\theta \text{ e } X_2 = R \operatorname{sen}\theta$$

com o ângulo θ uniformemente distribuído entre $[0, 2\pi]$.

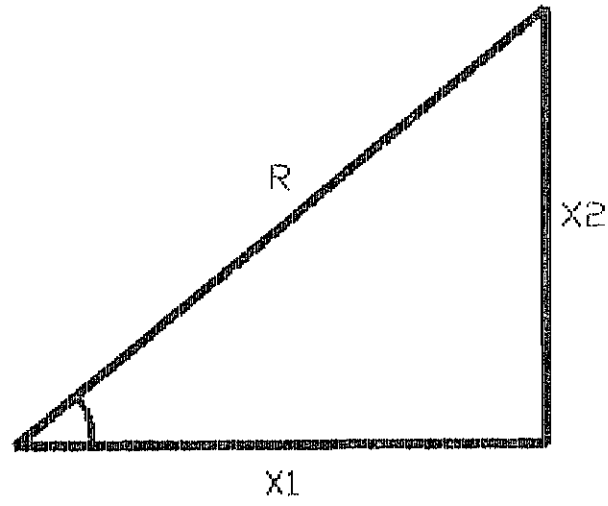


Fig. 4.1 - Relação entre as distribuições Normal e de Rayleigh.

Assim, a distribuição da Rayleigh fica sendo

$$F_R(r) = P(R < r) = 1 - \exp(-r^2/2) = 1 - \exp[-(x_1^2 + x_2^2)/2],$$

e podemos gerar duas v.a. gaussianas independentes por meio das expressões

$$X = \sqrt{-2 \cdot \ln u_1} \cdot \cos(2\pi . u_2)$$

$$X = \sqrt{-2 \cdot \ln u_1} \cdot \text{sen}(2\pi . u_2),$$ sendo u_1 e u_2 v.a. uniformemente distribuídas.

Ambas as técnicas foram usadas durante as simulações, exceto quando a probabilidade de erro a ser estimada era muito baixa e o tempo de computação tornava-se um fator limitante. Nesses casos, optou-se pela segunda implementação, por ser muito mais rápida.

Para se obter uma dada relação Eb/N0 admitiu-se que o espaçamento entre os pontos da constelação fosse

$$\Delta_0 = 2 \Rightarrow Es = 10.$$

Como essa energia média de símbolo é utilizada para transmitir 3 bits de informação, tem-se o valor de Eb fixado. Ajusta-se a variância do ruído de modo a obter-se a relação desejada, levando-se em conta que [25]

variância do ruído = $\sigma_0^2 = N_0/2$.

4.2.2.2 - A QUANTIZAÇÃO DOS SINAIS

A fim de permitir a representação das métricas num conjunto finito optou-se por quantizar os sinais recebidos, definindo antecipadamente quais seriam as métricas possíveis. Como a constelação é retangular, a alternativa natural é quantizar cada dimensão independentemente, com o mesmo número de passos e com

espaçamento linear. Isso não representa a métrica ótima mas acarreta uma perda muito pequena em relação a ela [6] e traz grandes vantagens a nível de implementação.

Foram utilizados dois quantizadores diferentes, o primeiro com 3 bits/dimensão e o segundo com 4 bits/dimensão, a fim de verificar a variação no ganho devido ao refinamento da quantização. Esses valores foram escolhidos em função dos trabalhos desenvolvidos por Snyder [26], que estudou o problema da representação da métrica para implementação de modulação codificada em constelações multi-fase, com a mesma técnica de quantização por dimensões utilizada aqui, embora as constelações não fossem retangulares.

Os resultados obtidos estão apresentados no capítulo seguinte.

4.2.3 - MÉTRICA

Esse módulo constrói uma tabela de métricas que associa a cada ponto da grade criada pela quantização a distância euclidiana a todos os pontos da constelação. Dessa forma, a cada ponto desse graticulado serão associadas 16 distâncias. O número total de pontos da grade depende do quantizador que se está usando :

- 64 pontos para o quantizador de 3 bits (8 níveis)/dim
- 256 pontos para o quantizador de 4 bits (16 níveis)/dim.

Obviamente o conjunto das métricas é menor do que 64 x 16 (ou 256 x 16). Optou-se por armazenar todos os pontos possíveis e suas combinações para tornar mais rápido o acesso a eles, acelerando a simulação.

4.2.4 - VITERBI

O algoritmo de Viterbi, citado em Viterbi [13] e Forney Jr. [21], é um algoritmo de máxima verossimilhança, de larga aplicação na decodificação de códigos convolucionais, não só por ser um algoritmo ótimo mas também pela simplicidade de sua implementação e flexibilidade no emprego de várias métricas.

Para os códigos estudados utilizou-se decisão suave e métrica euclidiana, que estão implementadas nos módulos CANAL e MÉTRICA respectivamente. O decodificador apenas faz uso dessas informações, o que torna bastante simples a simulação de diferentes esquemas. As principais características do decodificador são analisadas a seguir.

4.2.4.1 - ARMAZENAMENTO DAS MÉTRICAS

O conjunto das métricas existentes, que foi gerado em outro módulo, é formado por números inteiros que ficam armazenados em uma matriz, para que o acesso possa ser feito rapidamente. Dessa forma, ele só precisa ser calculado uma vez, no início da simulação, e armazenado para consulta posterior.

As métricas de cada uma das trajetórias sobreviventes são armazenadas num vetor associado aos nós (estados) da treliça. De tempos em tempos, esse vetor de métricas é normalizado para evitar que se exceda a capacidade de memória (overflow).

4.2.4.2 - O COMPRIMENTO DE DECODIFICAÇÃO

O comprimento de decodificação é definido como o número de passos na treliça além do qual todos os pares de trajetórias que ainda não se uniram têm distância maior do que a distância mínima d [14]. Isso está relacionado com as necessidades de armazenamento das trajetórias sobreviventes e com a complexidade de decodificação. Dele depende o atraso inicial na decodificação.

Normalmente utiliza-se como comprimento de decodificação um valor entre 4 e 6 vezes o maior dos atrasos sofridos por qualquer dos bits de entrada [27]. Nas simulações realizadas utilizou-se três valores distintos - 6, 9 e 18 passos - para ambos os códigos, a fim de verificar a degradação no desempenho e avaliar até onde pode-se sacrificar esse desempenho em favor de uma menor complexidade de decodificação.

4.2.4.3 - ARMAZENAMENTO DAS TRAJETÓRIAS

As trajetórias sobreviventes são armazenadas numa tabela indexada pelo estado final de cada trajetória. Na mesma tabela aparecem as métricas associadas a cada sobrevivente.

A fim de facilitar o processo de decodificação, armazena-se os bits de informação associados às transições - pois a forma de construir a treliça do código a partir da estrutura do codificador permite saber quais são esses bits. Com isso, a decodificação de códigos não-sistemáticos é bastante facilitada, tornando-se tão rápida quanto a dos sistemáticos.

4.2.5 - COMPARADOR

A estatística dos erros é feita de forma convencional: compara-se a sequência originalmente transmitida com a decodificada pelo módulo VITERBI, contando-se os bits não coincidentes.

Quanto ao módulo para estatística do código concatenado, ele opera de uma forma um pouco mais sofisticada, incorporando o entrelaçador/desentrelaçador na análise. Dois esquemas de entrelaçamento foram considerados: um a nível de bit e outro a nível de símbolo. O segundo é mais comum e, de um modo geral, tem desempenho superior. Isso ocorre devido a natureza do código externo que opera com símbolos e não bits. Entretanto, existem casos em que se obtém maiores ganhos com o primeiro esquema [1] e dessa forma ambos foram analisados.

Como o código externo utilizado opera com símbolos de $GF(2^8)$, é necessário agrupar oito bits da saída do decodificador para interpretá-los como um símbolo do código de Reed-Solomon. Entretanto, o código interno tem taxa 3/4, o que significa que cada etapa de decodificação gera três bits de saída. Para efetuar o "casamento" entre o código externo e o interno, usa-se o entrelaçador.

No programa desenvolvido, o decodificador interno não passa nenhuma informação de confiabilidade ao decodificador externo. A estatística é feita por comparação entre os bits do símbolo transmitido e do recebido, considerando-se que houve um erro quando pelo menos um dos bits está errado.

No programa desenvolvido, o decodificador interno não passa nenhuma informação de confiabilidade ao decodificador externo. A estatística é feita por comparação entre os bits do símbolo transmitido e do recebido, considerando-se que houve um erro quando pelo menos um dos bits está errado.

4.2.5.1 - O ENTRELAÇADOR DE BITS

A Figura 4.2 mostra o esquema do entrelaçador a nível de bit. Aqui considera-se uma palavra do código externo como um conjunto de 255 x 8 bits. Monta-se uma matriz onde cada linha é uma dessas palavras e o número de colunas, que dá a profundidade do entrelaçamento, é um múltiplo de três.

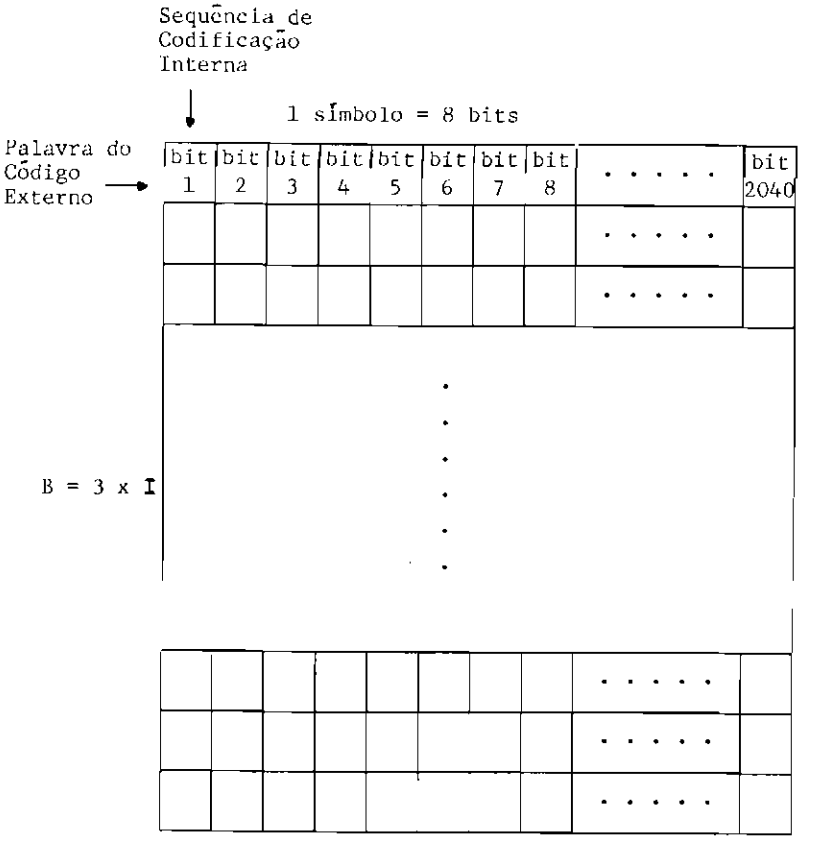


Fig. 4.2 - Esquema para entrelaçamento de bits na transmissão de códigos concatenados.

Essa matriz será, então, codificada por colunas pelo codificador interno, que tomará conjuntos de três bits - **um de cada palavra de código** - e fará o mapeamento adequado na constelação 16_QAM. Para a decodificação segue-se o processo inverso.

A profundidade de entrelaçamento I é dada pelo número de colunas B dividido por 3.

4.2.5.2 - O ENTRELAÇADOR DE SÍMBOLOS

Essa forma de entrelaçamento, mais comum, é similar à anterior, exceto que toma símbolos ao invés de bits para montar a matriz de palavras. A Figura 4.3 exemplifica esse esquema de entrelaçamento.

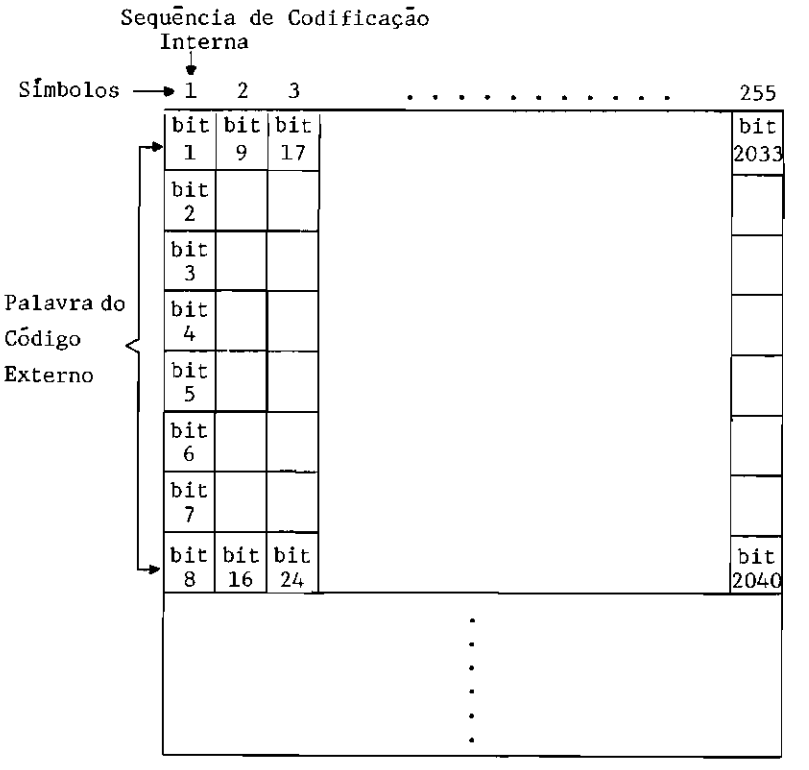


Fig. 4.3 - Esquema para entrelaçamento de símbolos na transmissão de códigos concatenados.

Aqui o número de colunas é um múltiplo de 3 símbolos, ou seja, de 24 bits. Cada grupo de oito linhas da matriz forma uma palavra do código. O codificador interno continua tomando conjuntos de 3 bits para mapear na constelação, apenas que agora ele transmite um símbolo de cada palavra antes de considerar a seguinte.

CAPÍTULO 5

RESULTADOS OBTIDOS

5.1 - INTRODUÇÃO

Apresentamos nesse capítulo os resultados obtidos na simulação dos códigos de Ungerboeck e dos códigos concatenados, em função dos vários parâmetros apresentados no capítulo anterior.

Esses resultados estão divididos em dois grupos: os resultados da simulação dos códigos de Ungerboeck, que caracterizam o código interno, e os resultados do código concatenado.

Para os esquemas de Ungerboeck, comparam-se os desempenhos dos códigos analisados em função do comprimento de decodificação e do número de níveis do quantizador usado. Além disso, esses desempenhos são confrontados com aquele obtido para a constelação de referência (8_PSK não codificado), e com o limitante inferior deduzido anteriormente.

Para o código concatenado, os desempenhos são comparados em função do comprimento de decodificação e do tipo de entrelaçamento (bit ou símbolo). É apresentado também um esquema convencional de concatenação, que utiliza a constelação BPSK, e comparado com os códigos aqui analisados com respeito a uma mesma taxa efetiva de transmissão (bits/dimensão).

5.2 - OS CÓDIGOS CONVOLUCIONAIS

Apresentamos em primeiro lugar os resultados de simulação dos códigos de Ungerboeck separadamente, analisados em função dos parâmetros já descritos. Em seguida comparamos o desempenho dos códigos entre si.

Nesses casos, faixa de operação dos códigos, isto é, a região de Eb/No onde se pretende operar, foi definida em função dos resultados que se pretende obter para o esquema concatenado.

5.2.1 - CÓDIGO I

A Figura 5.1 mostra o desempenho do código I (esse código foi definido na Seção 3.3) em função do comprimento de decodificação, para um quantizador que utiliza 8 níveis por dimensão. Estão indicados na mesma figura o desempenho da constelação de referência - o 8_PSK não codificado - e o limitante inferior obtido na Seção 3.3.1.2.

A Figura 5.2 apresenta o mesmo código com um quantizador de 16 níveis por dimensão.

De início podemos observar que a redução por um fator de 3 no comprimento de decodificação (de 18 para 6) acarreta uma perda da ordem de 0,5 dB independente do quantizador usado. Desse modo, um substancial ganho na complexidade pode ser obtido com um sacrifício pequeno no desempenho. Uma solução de compromisso parece ser o comprimento intermediário de 9 passos na treliça, com perdas próximas a 0,25 dB.

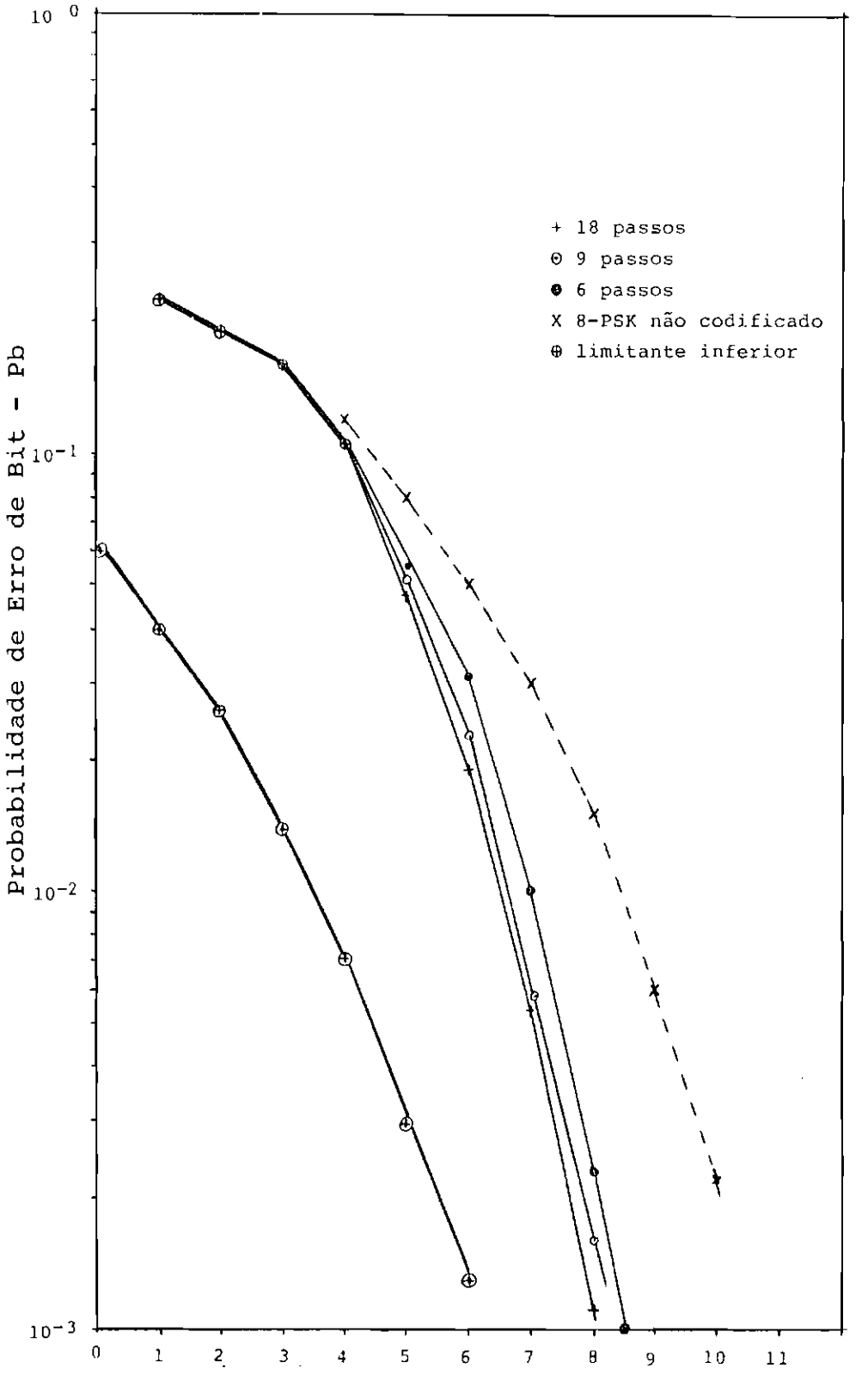


Fig. 5.1 - Desempenho do codificador de 4 estados. Quantizador de 8 níveis/dimensão.

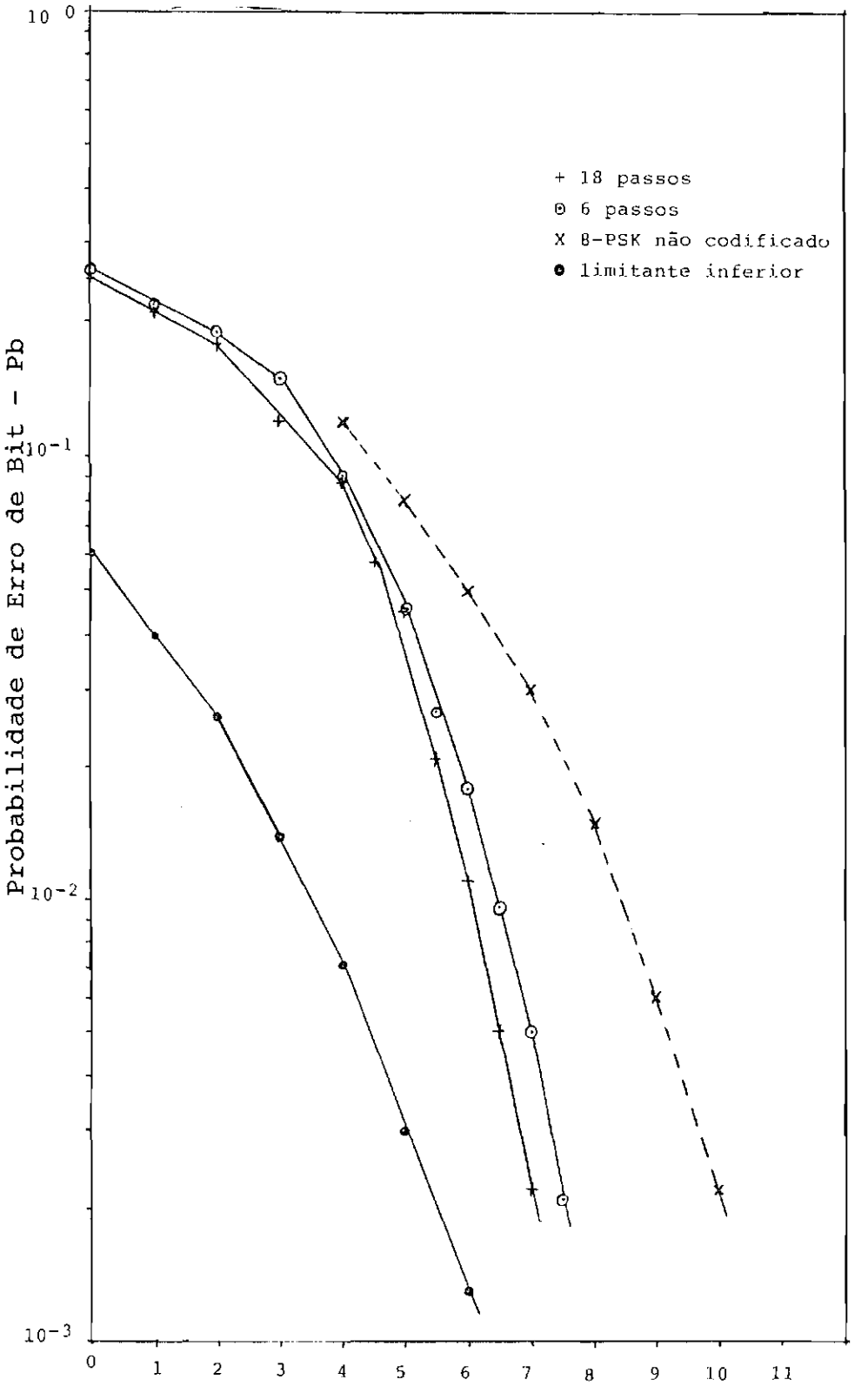


Fig. 5.2 - Desempenho do codificador de 4 estados. Quantizador de 16 níveis/dimensão.

Para relações sinal-ruído muito baixas, os erros se tornam tão frequentes que o desempenho passa a ser independente do comprimento de decodificação utilizado.

Em seguida podemos observar o desempenho em relação ao sistema não codificado: ganhos da ordem de 3 dB podem ser obtidos (2,5 dB se considerarmos o quantizador de 8 níveis) para probabilidades de erro de bit em torno de 10^{-3}. Embora esse ganho seja relativamente pequeno, deve-se observar que ele foi obtido para uma taxa de erros elevada, onde o desempenho de um código é geralmente baixo. Pode-se ainda perceber que para canais muito ruidosos o desempenho do sistema não codificado é superior ao do codificado: o código, nesse caso, atrapalha.

Finalmente nota-se que o refinamento do quantizador gera um ganho de 0,5 dB no desempenho, compatível com o que foi observado por Snyder [26], ainda que para constelações diferentes.

5.2.2 - CÓDIGO II

As Figuras 5.3 e 5.4 são equivalentes às duas anteriores para o código II. Nelas pode-se perceber as mesmas características já apontadas para o código I, no que diz respeito a comprimento de decodificação e níveis de quantização.

Entretanto, as degradações no desempenho são mais acentuadas nesse caso. A utilização de um comprimento de decodificação de 6 passos ao invés de 18 causa uma perda de 1 dB (um pouco menos - 0,8 dB - para um quantizador de 16 níveis). Um comprimento intermediário de 9 passos parece novamente ser o melhor compromisso, acarretando uma degradação da ordem de 0,5 dB.

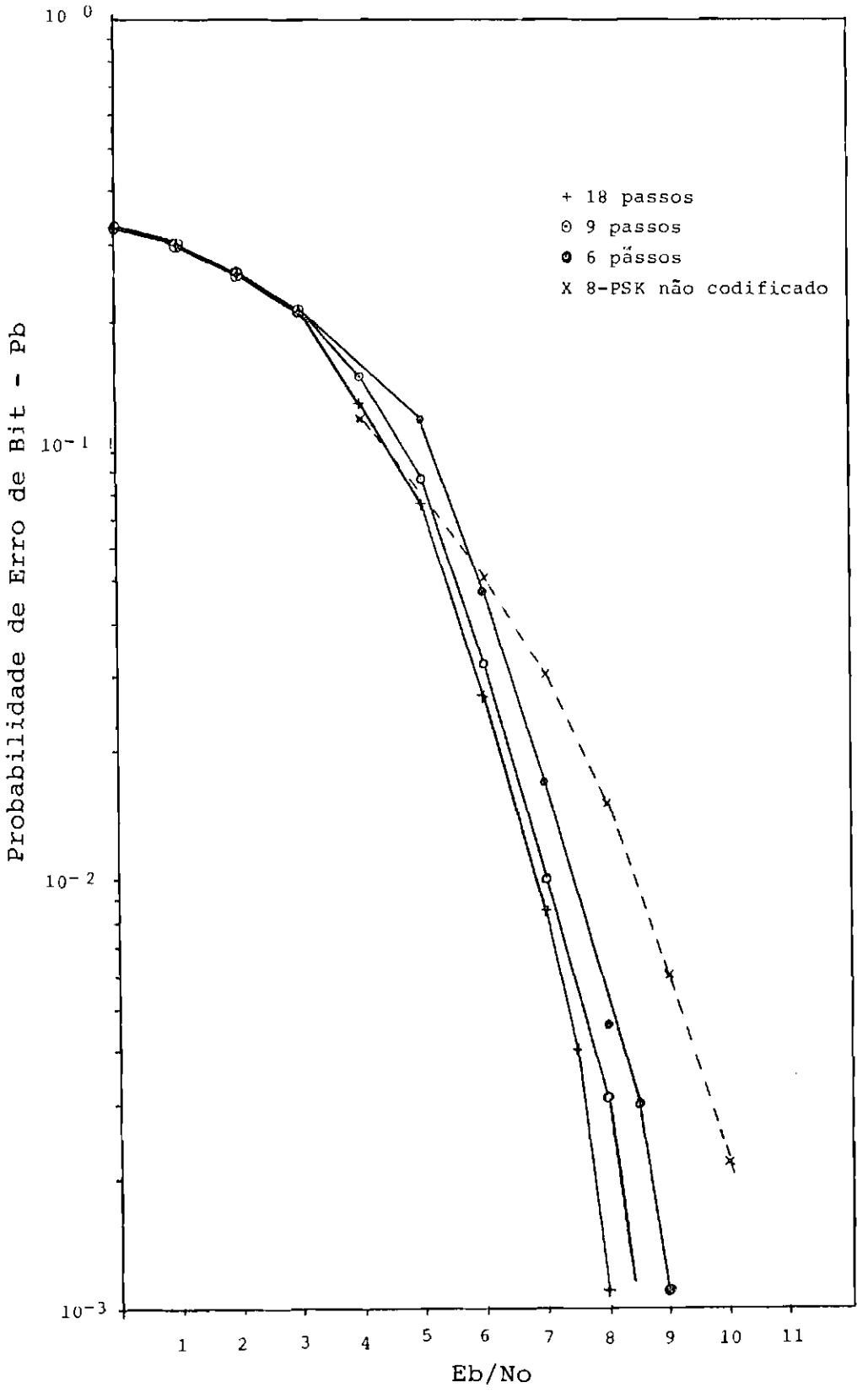


Fig. 5.3 - Desempenho do codificador de 8 estados. Quantizador de 8 níveis/dimensão.

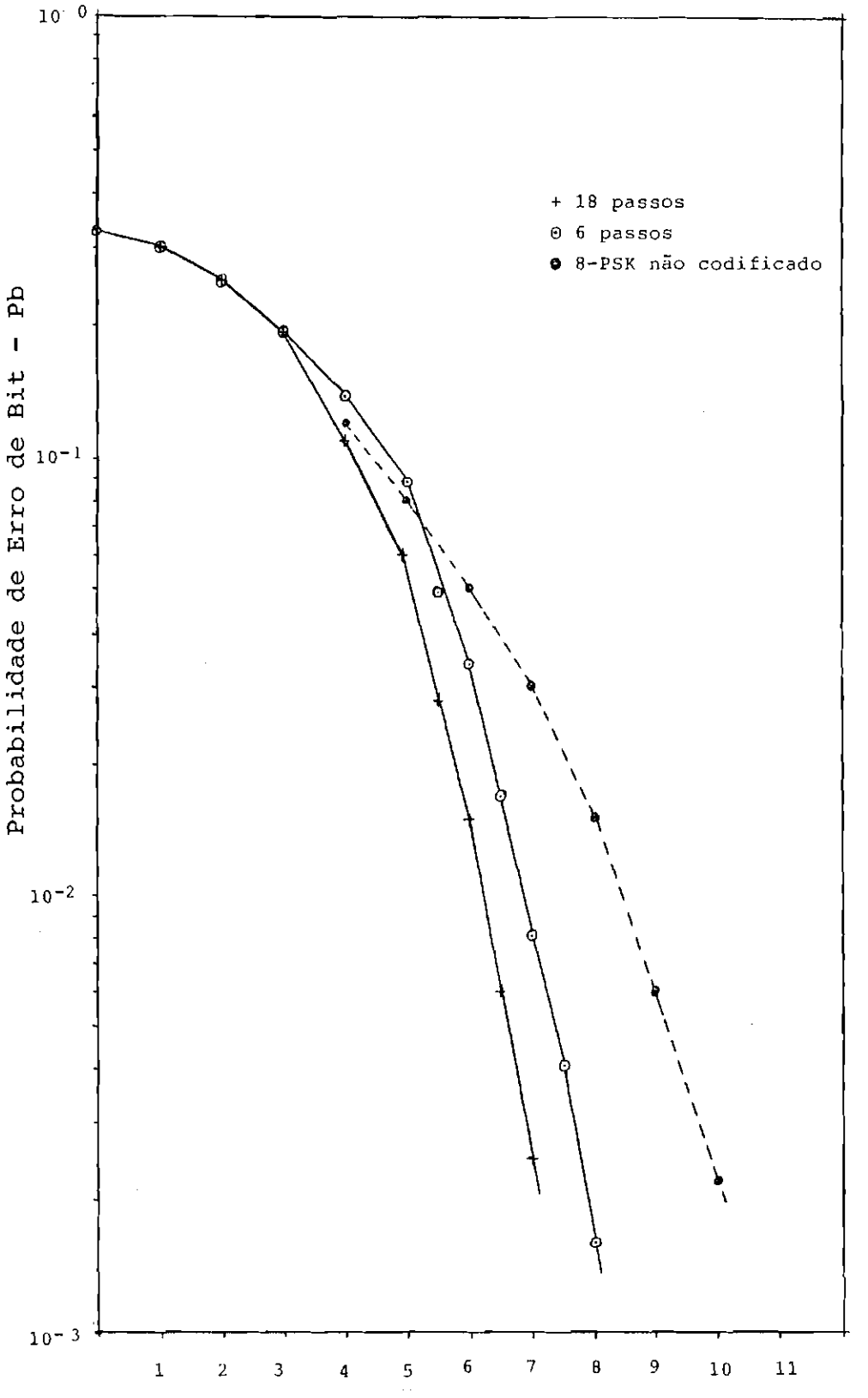


Fig. 5.4 - Desempenho do codificador de 8 estados. Quantizador de 16 níveis/dimensão

Também as melhoras relativas ao refinamento da quantização são maiores nesse código: da ordem de 0,7 dB.

O ganho em relação ao 8_PSK não codificado, que é a constelação de referência, é de 3 dB (2,3 dB para o quantizador de 8 níveis) na região onde a probabilidade de erro de bit é da ordem de 10^{-3}. Pode-se notar claramente aqui o melhor desempenho do sistema não codificado quando a relação sinal-ruído é baixa, especialmente quando o quantizador de 8 níveis é usado.

5.2.3 - COMPARAÇÃO ENTRE OS CÓDIGOS

Se compararmos o desempenho de ambos os códigos, na faixa de Eb/No onde se pretende operar, observamos que o código I é uniformemente superior ao código II, para os mesmos parâmetros do quantizador e comprimento de decodificação. Isso ocorre, apesar do maior ganho assintótico, devido à maior conectividade da treliça do código II, que permite eventos de erro (com distância maior do que a mínima) de comprimento menor do que seus equivalentes no código I.

Essa relação entre os desempenhos tende a se inverter à medida que aumenta a relação Eb/No, como se pode depreender da Figura 5.5, onde estão mostradas curvas de desempenho de ambos os códigos. Nessa figura observa-se desempenhos equivalentes em torno de uma relação Eb/No = 7 dB.

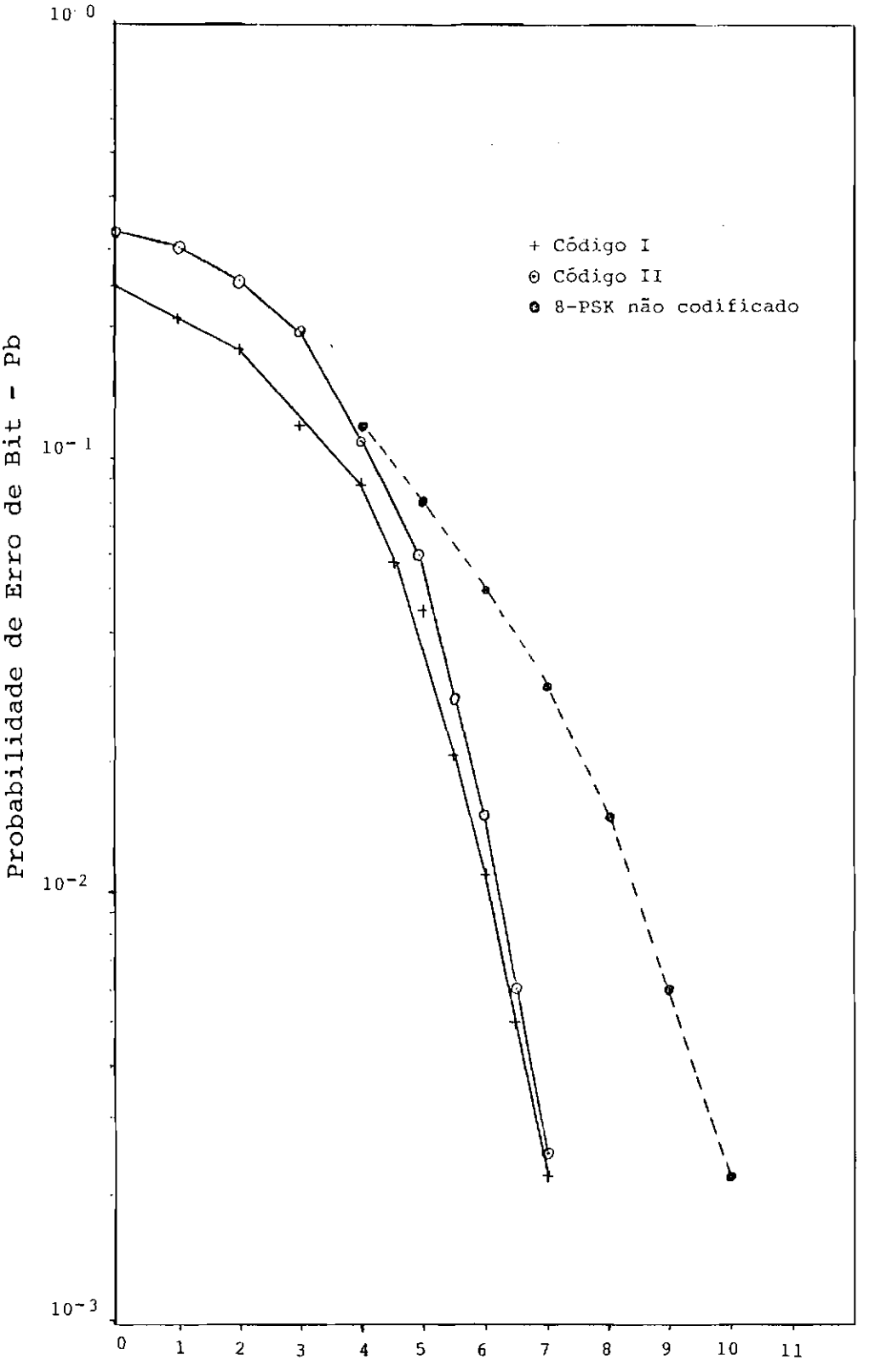


Fig. 5.5 - Comparação do desempenho dos códigos I e II.
Comprimento de decodificação: 18.
Quantizador: 16 níveis/dimensão.

A partir desse ponto, as curvas se cruzam e o código mais complexo (código II) passa a apresentar desempenho superior.

Uma vez que esses códigos serão usados como código interno de um esquema de codificação concatenada, que deverá operar na faixa indicada nas figuras, o código I é a escolha mais adequada, pois conjuga melhor desempenho e maior simplicidade.

5.3 - CÓDIGO CONCATENADO

Consideraremos aqui os resultados da concatenação do código I, escolhido como a melhor opção, com um código de Reed-Solomon definido em $GF(2^8)$. A seleção do tamanho do código foi baseada exclusivamente em considerações de caráter prático: é o padrão utilizado em esquemas convencionais, com implementações já testadas, inclusive em VLSI.

Apresentaremos inicialmente o resultado da concatenação do código I, para os vários parâmetros que foram definidos, com o código RS (255,223), que tem capacidade de corrigir até 16 símbolos errados. Estudaremos ambos os esquemas de entrelaçamento apresentados - bit e símbolo.

Em seguida analisaremos o desempenho do sistema proposto comparado aos resultados que existem para um esquema de concatenação convencional, que utiliza transmissão em BPSK. Para isso, utilizaremos códigos de Reed-Solomon diferentes (quanto à capacidade de correção) de modo a comparar os esquemas em igualdade de condições no tocante a taxa de informação transmitida.

5.3.1 - ANÁLISE DO CÓDIGO

Uma vez que o código interno tem taxa $R_{int} = 3/4$ e está-se utilizando constelação bidimensional, temos uma taxa efetiva de 1,5 bits/dimensão por uso do canal, isto é, por símbolo transmitido. Como o código RS tem taxa $R_{ext} = 7/8$, concluímos que a taxa efetiva de transmissão é de 1,3 bits/dimensão. Dessa forma obtivemos um esquema cuja eficiência espectral é superior à do BPSK não codificado - que pode transmitir apenas 1 bit/dim.

Na análise da probabilidade de erro de bit usaram-se as expressões desenvolvidas na Seção 1.3, em particular a Equação 1.2. A simulação consistiu em avaliar a probabilidade de erro de símbolo P_s para uma dada profundidade de entrelaçamento.

5.3.2 - OS ENTRELAÇADORES

As simulações confirmaram os resultados obtidos em esquemas convencionais quanto ao uso dos entrelaçadores: o desempenho do código com o entrelaçador de bits é inferior àquele com o entrelaçador de símbolos. O estudo foi levado a efeito em função dos resultados obtidos por Battail (citado em [1]), que obtivera resultados positivos com o entrelaçador de bits em alguns esquemas de concatenação. Entretanto, para o código analisado, que utiliza decodificação suave apenas no canal interno, a técnica de entrelaçamento convencional mostrou-se superior, havendo equivalência no desempenho apenas para relações sinal-ruído muito baixas, onde, inclusive, o ganho de codificação do código interno é nulo ou até mesmo negativo.

Em vista disso, concentraremos nossa atenção no entrelaçador de símbolos, que foi descrito na Seção 4.2.5.2.

Nos resultados que apresentamos a seguir considera-se o embaralhamento de símbolos com profundidade ideal, ou seja, de tal modo que os símbolos errados sejam independentes.

5.3.3 - RESULTADOS DA SIMULAÇÃO

Para simulação do código concatenado usou-se o código I, com quantizador de oito níveis, para código interno, e o código RS (255,223) como código externo.

As Figuras 5.6 e 5.7 mostram o desempenho do código concatenado em duas regiões distintas, que poderiam ser grosseiramente designadas como de "alta" e "baixa" probabilidade de erro.

Pode-se perceber através desses gráficos que, para relações Eb/No baixas (Figura 5.6), a concatenação pouco acrescenta ao desempenho. Nesses casos, códigos mais simples oferecem desempenho idêntico ou até superior [4].

Para valores mais altos de Eb/No (Figura 5.7) pode-se perceber a rápida variação (declividade elevada) da curva de desempenho. É nessa região que as vantagens do uso desse esquema concatenado se manifestam. Apenas para ilustrar esse ponto, podemos observar que probabilidades de erro de bit da ordem de 10^{-6} são obtidas para relações Eb/No em torno de 7 dB, enquanto que um sistema de transmissão BPSK não codificado, que tem eficiência espectral menor que o esquema analisado (1:1,3), somente atinge essa probabilidade para valores de Eb/No próximos a 10,5 dB. É possível que um ganho de 3,5 dB não justifique o aumento da complexidade necessário para obtê-lo.

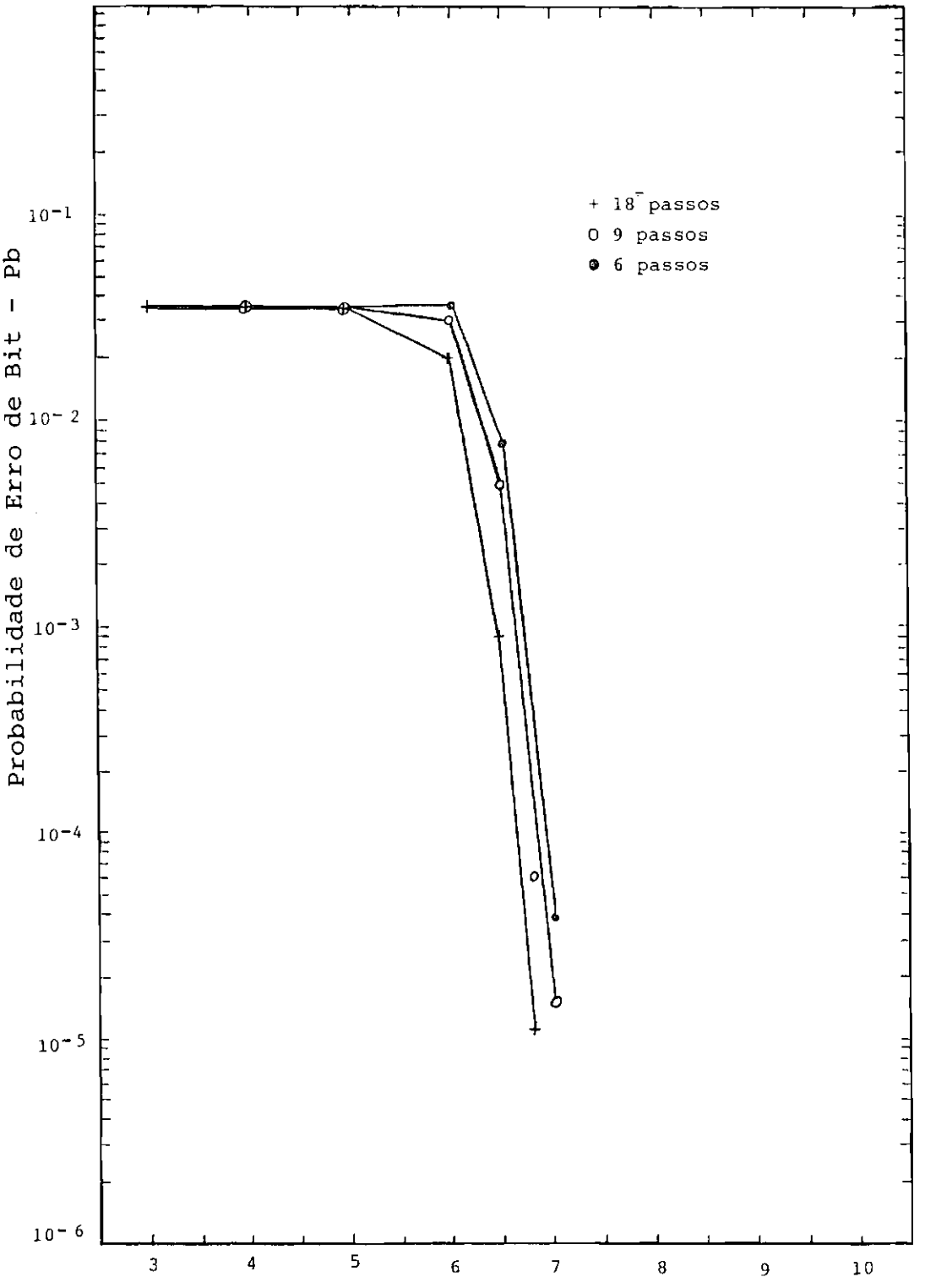


Fig. 5.6 - Desempenho do código concatenado.
"Baixa" relação Eb/No.

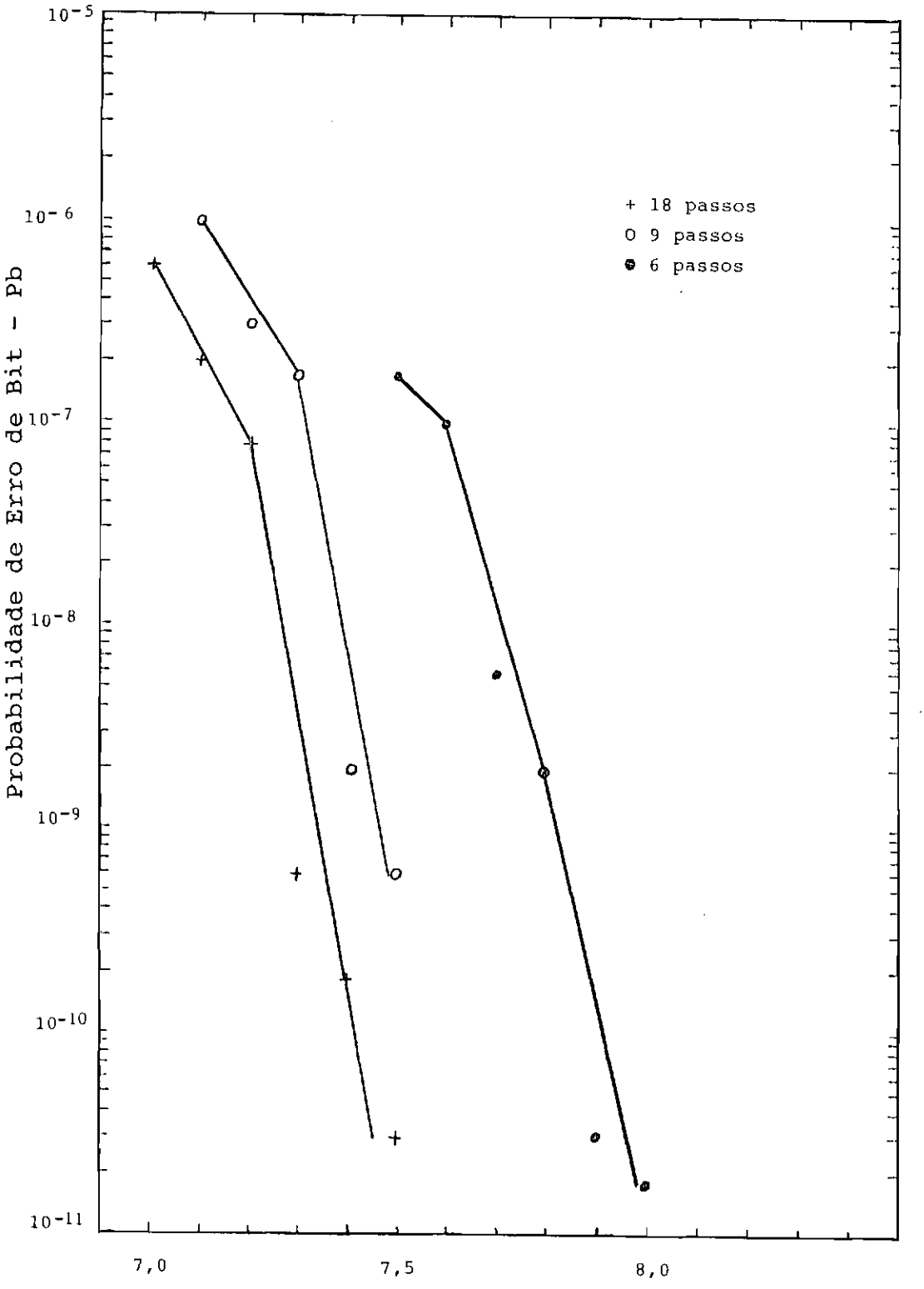


Fig. 5.7 - Desempenho do código concatenado.
"Alta" relação Eb/No.

Por outro lado, havendo necessidade de alta confiabilidade na transmissão, como no caso de sinais com baixa redundância, pode-se perceber a vantagem de um esquema que atinge probabilidade de erro de bit de 10^{-10}com Eb/No = 7,5 dB.

Finalmente, vale observar a influência do comprimento de decodificação do código interno no desempenho geral do esquema de codificação. Como se pode perceber através dos gráficos apresentados, particularmente a Figura 5.7, uma diferença de 0,5 dB no desempenho do código é obtida quando se altera o comprimento de decodificação de 18 para 6 passos na treliça. Aqui, o valor de 9 passos é a escolha mais adequada, implicando em perdas menores do que 0,2 dB.

5.3.4 - COMPARAÇÃO ENTRE ESQUEMAS CONCATENADOS

Para analisarmos as características do esquema proposto, faremos a seguir uma comparação com o código concatenado padrão do CCSDS - código interno convolucional (2,1,7) e código externo RS (255,223), com transmissão BPSK. A fim de facilitar as referências chamaremos esse esquema de transmissão BPSK de convencional, e o esquema com código interno de Ungerboeck de codimodulado.

A fim de tentar manter a comparação em bases iguais devemos adequar a taxa de transmissão dos códigos de modo a termos a mesma quantidade de informação transmitida por símbolo do canal em ambos os casos. Para isso, considerando que a transmissão BPSK ocorre com taxa R = 0,437, devemos ajustar o código externo do esquema codimodulado para igualar essa taxa. Como o código de Ungerboeck que se está utilizando transmite na taxa de 1,5 bits/dimensão, usaremos para código externo um RS

(255,75), que resulta na mesma taxa final. Esse código tem capacidade para corrigir 90 símbolos errados.

A Figura 5.8 mostra o desempenho de ambos os códigos. O desempenho do código convencional foi obtido em [1]; no cálculo do desempenho do esquema codimodulado utilizaram-se os resultados já obtidos na Seção 5.3.3 para a probabilidade de erro de símbolo Ps - considerando um comprimento de decodificação de 18 passos no código interno, ajustando-se a Equação 1.2 da Seção 1.3 para os novos parâmetros do código externo.

A comparação entre os dois códigos mostra uma vantagem de quase 2 dB para o esquema convencional. Isso indica que o potencial de esquemas de concatenação codimodulados, pelo menos para códigos internos de baixa complexidade, é melhor aproveitado em sistemas com limitação de faixa, onde sua maior eficiência espectral se impõe sobre outros esquemas.

Deve-se levar em conta, entretanto, que a complexidade de decodificação do esquema codimodulado tende a ser menor que a do esquema convencional, dado o menor número de estados na treliça do código interno (4 estados para o código de Ungerboeck contra 64 para o padrão CCSDS). Dessa forma, e uma vez que complexidade de implementação influencia diretamente no custo, talvez as bases nas quais os dois esquemas são comparados não sejam absolutamente idênticas. Dado que um pequeno incremento no desempenho do código interno tem grandes reflexos no desempenho do esquema concatenado, o uso de um código de Ungerboeck mais sofisticado poderia alterar a conclusão acima.

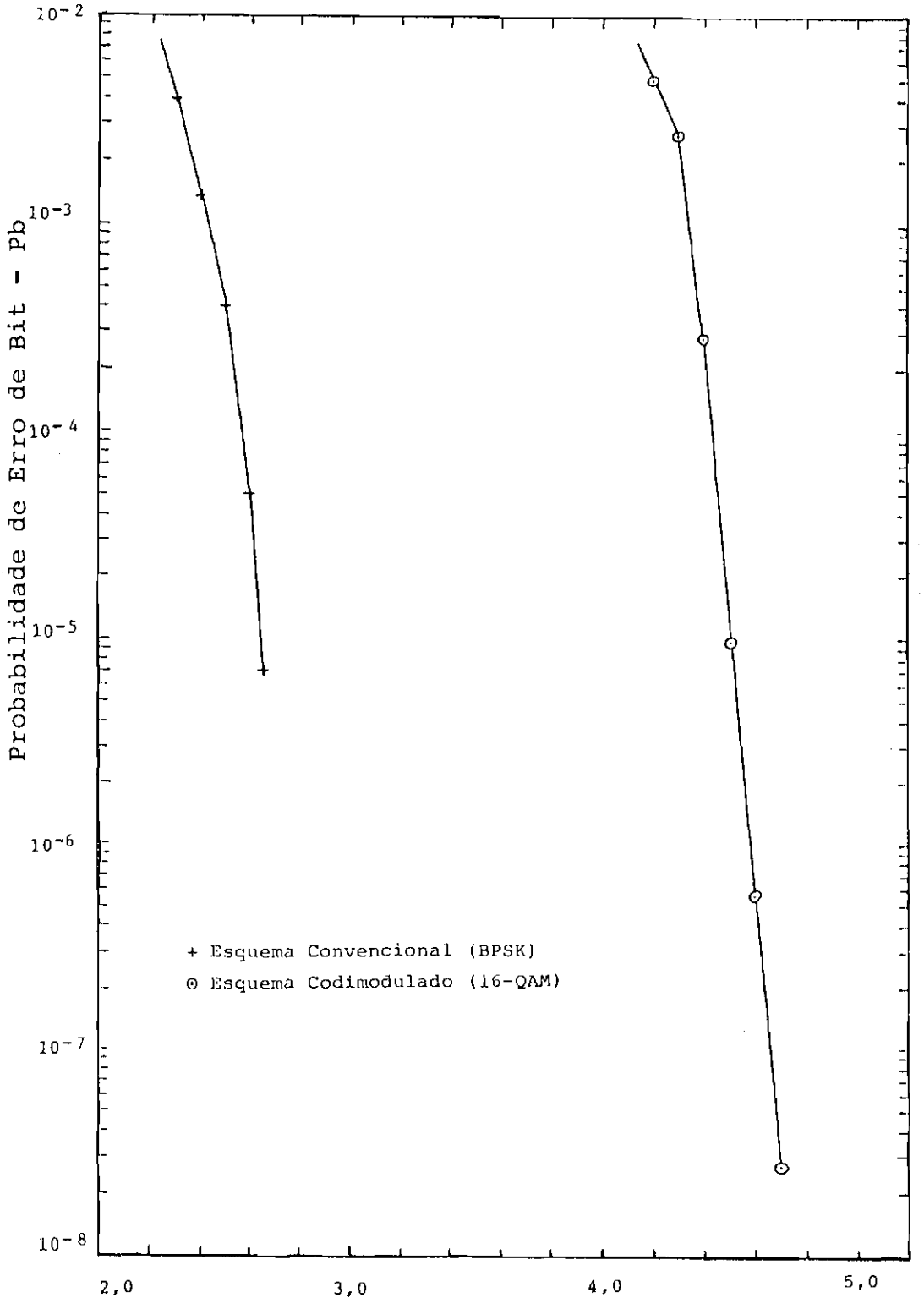


Fig. 5.8 - Comparação entre sistemas de codificação concatenada.

CAPÍTULO 6

COMENTÁRIOS

O presente trabalho procurou explorar as possibilidades de uso de modulação codificada em esquemas concatenados, para constelações retangulares do tipo 16_QAM.

Vários esquemas de modulação codificada foram avaliados a fim de se verificar o seu potencial e encontrar "candidatos" a código interno do sistema concatenado que se pretendia estudar.

Foram analisados códigos internos de bloco e convolucional, buscando-se obter os melhores compromissos entre desempenho e complexidade.

Para os códigos de bloco, foi esboçado o esquema de codificação de Sayegh e apresentados alguns resultados assintóticos para vários códigos BCH, que representam o melhor compromisso entre taxa e ganho de codificação para a constelação escolhida (16_QAM) e os tamanhos de bloco estudados.

Para os códigos convolucionais, as simulações realizadas constataram a existência de excelentes opções, do ponto de vista de desempenho, com baixa complexidade (códigos convolucionais de taxa R = 3/4 e treliça de 4 estados), bastante adequadas para código interno de um esquema concatenado. Essas simulações mostraram ainda que para relações Eb/N_O baixas (onde a probabilidade de erro de bit é da ordem de 10^{-3}) treliças de menor conectividade, correspondendo a códigos com mais transições paralelas, apresentam melhor desempenho. Estes códigos geralmente apresentam menor complexidade - isto é, menor número de

estados - de modo que, em sistemas que operam em canais muito ruidosos, eles oferecem um compromisso ótimo entre desempenho e complexidade

Esse trabalho de simulação, que visava à obtenção da probabilidade de erro de bit dos códigos internos, apresentou, como subproduto, uma avaliação do compromisso entre os vários parâmetros dos códigos, tais como comprimento de decodificação e números de níveis nos quantizadores, que pode fornecer subsídios para uma possível implementação prática desses códigos.

A comparação dos códigos estudados com esquemas já existentes mostrou, numa primeira análise, que sistemas de concatenação com modulação codificada não apresentam grandes vantagens quando não há limitação de faixa. Essa análise, entretanto, não levou em conta a complexidade de decodificação dos sistemas comparados.

Uma extensão óbvia do trabalho aqui desenvolvido seria a simulação dos esquemas de Sayegh propostos, para obter-se a curva de desempenho dos mesmos. Neste caso, seria necessário desenvolver um decodificador suave para esses códigos. Com essa ferramenta em mãos seria possível estudar a influência de parâmetros como taxa, componentes de distância e outros no desempenho dos códigos, além de testar a eficiência da técnica de codificação por dimensões que foi esboçada neste trabalho.

Outra possibilidade está no estudo das técnicas de simulação em si mesmas. O programa desenvolvido para obter os resultados que foram apresentados é bastante lento, o que dificulta a simulação precisa de probabilidades de erro muito pequenas. Novas técnicas, usando "importance sampling", podem reduzir esses tempos por fatores de 100 e até mesmo 1000 vezes. Com isso, o comportamento dos vários códigos, tanto de bloco como os convolucionais, poderia ser estudado nas regiões de alta

relação sinal-ruído, e novos compromissos entre taxa, complexidade, desempenho e outros fatores, poderiam ser obtidos.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] FARRELL, P.G.; BRINE, A. Study of reduced complexity concatenated coding schemes: final report to ESA/ESTEC, (ESA/ESTEC 7131/87/DG), Manchester, ENGLAND, Jun. 1988.

[2] DENG, R.H.; COSTELLO JR., D.J. High rate concatenated coding systems using bandwidth efficient trellis inner codes. s.n.t. Este texto apareceu, com algumas modificações, no International Symposium of Information Theory (ISIT) - IEEE 1986.

[3] BERLEKAMP, E.R. Key papers in the development of coding theory. New York, IEEE Press, 1974.

[4] BERLEKAMP, E.R.; PEILE, R.E.; POPE, S.P. The application of error control to communications. IEEE Communications Magazine, 25(4):44-57, Apr. 1987.

[5] ODENWALDER, J.P. Optimal decoding of convolutional codes. Ph.D. Dissertation. California, School of Engineering and Applied Science, UCLA, 1970.

[6] CLARK JR., G.C.; CAIN, J.B. Error correction codes for digital communications. New York, Plenum, 1981.

[7] SHANNON, C.E. A mathematical theory of communications, Part I. Bell System Technical Journal, 27(3):379-423, Jul. 1948.

[8] MASSEY, J.L. Coding and modulation in digital communications. In: International Zurich Seminar on Digital Communications, Zurich, 1974. Proceedings. Zurich, 1974, p. E2(1)-E2(4).

[9] _____ The how and why of channel coding. In: International Zurich Seminar on Digital Communications, Zurich, 1984. Proceedings. Zurich, 1984, p. E1(1)-E1(7).

[10] DIGEON, A. On improving bit error probability of QPSK and 4-level amplitude modulation systems by convolutional coding. IEEE Transactions on Communications, 25(10):1238-1239, Oct. 1977.

[11] UNGERBOECK, G. Channel coding with multilevel/phase signals. IEEE Transactions on Information Theory, 28(1):55-67, Jan. 1982.

[12] FORNEY JR., G.D.; GALLAGER, R.G.; LANG, G.R.; LONGSTAFF, F.M.; QURESHI, S.U. Efficient modulation for band-limited channels. IEEE Journal of Selected Areas in Communications, 2(5):632-646, Sept. 1984.

[13] VITERBI, A.J. Convolutional codes and their performance in communication systems. IEEE Transactions on Communication Technology, 19(5):751-772, Oct. 1971.

[14] BENEDETTO, S.; MARSAN, M.A.; ALBERTENGO, G.; GIACHIN, E. Combined coding and modulation: theory and applications. IEEE Transactions on Information Theory, 34(2):223-236, Mar. 1988.

[15] BIGLIERI, E. High-level modulation and coding for nonlinear satellite channels. IEEE Transactions on Communications, 32(5):616-626, May 1984.

[16] CUSACK, E.L. Error control codes for QAM signaling. Electronics Letters, 20(2):62-63, Jan. 1984.

[17] SAYEGH, S.I. A class of optimum block codes in signal space. IEEE Transactions on Communications, 34(10):1043-1045, Oct. 1986.

[18] IMAI, H.; HIRAKAWA, S. A new multilevel coding method using error-correcting codes. IEEE Transactions on Information Theory, 23(3):371-377, May 1977.

[19] YAMAGUCHI, K.; IMAI, H. Higly reliable multilevel channel coding system using binary convolutional codes. Electronics Letters, 23(18):939-941, Aug. 1987.

[20] PROAKIS, J.G. Digital communications. Tokio, McGraw-Hill, 1983.

[21] FORNEY JR, G.D. The Viterbi algorithm. Proceedings of the IEEE, 61(3):268-278, Mar. 1973.

[22] BLAHUT, R.E. Theory and practice of error control codes. Addison-Wesley, 1983.

[23] MAcWILLIANS, F.J.; SLOANE, N.J.A. The theory of error-correcting codes. New York, North Holland, 1977.

[24] FARREL, P.G.; WILLIANS, R.G.C. Reducing the decoding complexity of block coded modulation schemes. s.n.t. Os autores estão no Department of Electrical Enginnering, University of Manchester, M13 9PL, UK.

[25] WOZENCRAFT, J.M.; JACOBS, I.M. Principles of communication engineering. New York, Wiley, 1965.

[26] SNYDER, J. High-speed Viterbi decoding for high-rate codes. In: International Conference on Digital Communication, 6., Phoenix, AZ, 1983. Proceedings. Phoenix, AZ, 1983, p. XII(16)-XII(23).

[27] HELLER, J.A.; JACOBS, I.M. Viterbi decoding for satellite and space communication. IEEE Transactions on Communication Technology, 19(50):835-848, Oct. 1971.

APÊNDICE A

PROGRAMA DE SIMULAÇÃO DOS CÓDIGOS

O programa VITERBI faz a simulação dos códigos de Ungerboeck estudados. Ele é basicamente a implementação dos módulos descritos no Capítulo 4 deste trabalho.

O programa tem capacidade para simular códigos que utilizam o 16_QAM para sinalizar através do canal. Ele faz decisão suave nas duas dimensões (4 bits por dimensão). O comprimento de decodificação é um dado de entrada do programa, assim como o valor desejado de Eb/No e o número de bits de informação que devem ser transmitidos.

O programa está organizado de modo a orientar o usuário quanto ao fornecimento dos dados do código a ser simulado que são necessários para a inicialização dos vários módulos. O programa assume que o usuário já tem definida uma certa arquitetura do codificador (em termos de elementos de memória, somadores e ligações entre eles). Dessa forma as implementações de códigos sistemáticos, com realimentação e outros mais, podem ser tratadas igualmente.

Para indicar ao programa a maneira como os elementos do codificador se interligam foi criada uma matriz de ligação, semelhante à matriz de incidência da teoria dos grafos: as linhas da matriz são organizadas de modo a conter, na sequência, os bits de entrada (iniciando pelo mais significativo), as memórias e os somadores. As colunas devem conter as memórias, os somadores e os bits de saída. A matriz contém apenas zeros, exceto nas posições que indicam ligação entre um item da linha e um da coluna. Essas posições o usuário deve indicar ao programa.

```
program VITERBI;
uses crt;

const
     Lim = 15; Dim = 20;
     Exp_Lim = 260;
     a = 0.33333; b = 1;


type
     vector = array[1..Lim] of integer;
     vector_exp = array[0..Exp_Lim] of integer;
     matrix = array[1..Dim,1..Dim] of integer;
     tridim = array[0..32,1..32,1..3] of integer;
     tabela = array[0..32,-1..41,0..1] of integer;
     metrica = array[0..15,-7..7,-7..7] of integer;


var
     k, n, m, s, Start, Stop, Pres, Fut    : integer;
     Iter, EbNo, Desvio, Counter, Num_Iter: real;
     Link                                  : matrix;
     HypoThesis                            : tridim;
     Table                                 : tabela;
     Est_Inic, Info_b                      : vector;
     Reg                                   : vector_exp;
     Output, Info_d, I, J, Control, Error : integer;
     ExpM, ExpK, Out, Res, Delay           : integer;
     Metrica_16QAM                         : metrica;
     Sai                                   : text;
     Arq                                   : string[12];
     Alfa                                  : char;



(* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *                                                                             *
 *            O GRUPO DE PROCEDURES QUE SE SEGUE IMPLEMENTA UMA SERIE           *
 *            DE FUNCOES AUXILIARES DE CALCULO:                                *
 *                                                                             *
 *                                                                             *
 *       _ EXPON : calcula a exponencial de uma base e expoente inteiros.      *
 *                                                                             *
 *       _ BINARY: converte um numero decimal em binario.                      *
 *                                                                             *
 *       _ DECIMAL: converte um numero binario em decimal.                     *
 *                                                                             *
 *       _ INCREMENTA: efetua um incremento unitario modulo o DELAY.           *
 *                                                                             *
 *                                                                             *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *


function EXPON (Base, Pot : integer): integer;
begin
     EXPON:= round(exp(Pot*ln(Base)))
end;
```

```
procedure BINARY (Dec, Tam: integer; var Bin: vector);
var
     J : integer;
begin
     For J:= Tam downto 1 do
     begin
          Bin[J]:= Dec mod 2;
          Dec:= Dec div 2
     end
end;

procedure DECIMAL (var Bin: vector; var Tam, Dec: integer);
var
     I : integer;
begin
     Dec:= 0;
     For I:= 1 to Tam do
     Dec:= Dec + Bin[I]*Expon(2,(Tam-I));
end;

procedure INCREMENTA (var Num : integer);
begin
     Num:= (Num + 1) mod Delay;
end;


(* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *                                                                                               *
 *           AS PROCEDURES SEGUINTES IMPLEMENTAM O CODIFICADOR.                                  *
 *                                                                                               *
 *           Elas geram uma tabela  - hypothesis -  que armazena                                 *
 *           a treliça, com informaçoes sobre  estado de chegada                                 *
 *           de um ramo, codigo associado a transiçao e bits  de                                 *
 *           informaçap que geraram a transiçao.                                                 *
 *                                                                                               *
 *           As procedures  ACTUAL_ADDER, CALC_OUT , ACTUAL_MEMO                                 *
 *           e CODER geram o codificador.                                                        *
 *           As procedures STORE_BRANCH e GEN_HYPOTHESIS geram a                                 *
 *           treliça e a armazenam.                                                              *
 *                                                                                               *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *)


procedure ACTUAL_ADDER (var u, Memo, Add: vector);
var
     Line , L : integer;
begin
     For L:= 1 to s do
     begin
          Add[L]:= 0;
          For Line:= 1 to k do
          If Link[Line, L+m] = 1
          then Add[L]:= Add[L] + u[Line];
          For Line:= 1 to m do
          begin
               If Link[Line+k, L+m] = 1
               then Add[L]:= Add[L] + Memo[Line]
          end;
```

```
            Add[L]:= Add[L] mod 2
        end
end;

procedure CALC_OUT (var u, Memo, Add, v: vector);
var
     Line, J: integer;
begin
     For J:= 1 to n do
     begin
          Line:= 0;
          repeat   Line:= Line + 1
          until    Link[Line, J+m+s] = 1;
          If Line <= k
          then  v[J]:= u[Line]
          else  If Line <= k+m
                then  v[J]:= Memo[Line-k]
                else  v[J]:= Add[Line-k-m]
     end
end;

procedure ACTUAL_MEMO (var u, Memo, Add: vector);
var
     Line, c: integer;
     Memo_Aux: vector;
begin
     For c:= 1 to m do
     begin
          Line:= 0;
          repeat  Line:= Line+1
          until   Link[Line, c] = 1;
          If  Line <= k
          then  Memo_Aux[c]:= u[Line]
          else  If  Line <= k+m
                then  Memo_Aux[c]:= Memo[Line-k]
                else  Memo_Aux[c]:= Add[Line-k-m]
     end;
     For c:= 1 to m do
     Memo[c]:= Memo_Aux[c];
end;

procedure CODER (var u, Memo, v: vector);
var
     Add: vector;
begin
     ACTUAL_ADDER (u, Memo, Add);
     CALC_OUT (u, Memo, Add, v);
     ACTUAL_MEMO (u, Memo, Add)
end;

procedure STORE_BRANCH (a: integer; var b: vector_exp;
                        v, E, u: integer; var Store: tridim);
begin
     b[a]:= b[a] + 1;
     Store[a, b[a], 1]:= v;
     Store[a, b[a], 2]:= E;
     Store[a, b[a], 3]:= u;
end;

procedure GEN_HYPOTHESIS (var Hip: tridim);
```

```
var
     I, Est_Inic, Est_Fim, Ent, Sai: integer;
     State, Aux_State, Input, Output: vector;
     Count: vector_exp;
begin
     For I:= 0 to Exp_Lim do
     Count[I]:= 0;
     For Est_Inic:= 0 to (EXPON (2,m) - 1) do
     begin
          BINARY (Est_Inic, m, State);
          For Ent:= 0 to (EXPON (2,k) - 1) do
          begin
               For I:= 1 to m do
               Aux_State[I]:= State[I];
               BINARY (Ent, k, Input);
               CODER (Input, Aux_State, Output);
               DECIMAL (Output, n, Sai);
               DECIMAL (AUX_State, m, Est_Fim);
               STORE_BRANCH (Est_Fim, Count, Sai, Est_Inic, Ent, Hip)
          end
     end
end;



(* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *                                                                             *
 *           AS PROCEDURES SEGUINTES GERAM UM RUIDO GAUSSIANO COM              *
 *           MEDIA ZERO  E  VARIANCIA ADEQUADA  PARA  OBTER-SE  A              *
 *           RELAÇAO Eb/No ESPECIFICADA PARA O SINAL.                          *
 *                                                                             *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *)


procedure SIGMA (var Desvio: real);
begin
     Desvio:= sqrt(10/(9*k))*exp((-(EbNo+3.01)/20)*ln(10));
end;


procedure GAUSS_NOISE (var Desvio, Ruido1, Ruido2: real);
var
     R1, R2: real;
begin
     R1:= random;
     R2:= random;

     Ruido1:= sqrt(-2*ln(R1))*cos(2*PI*R2)*Desvio;
     Ruido2:= sqrt(-2*ln(R1))*sin(2*PI*R2)*Desvio;

end;


(* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *                      PROCEDURES AUXILIARES                                  *
 *                                                                             *
 *          As procedures abaixo auxiliam a geracao de uma saida               *
 *          aleatoria, que eh necessaria quando a  codimodulacao               *
 *          nao eh superlinear. Dessa forma, gera-se uma entrada               *
```

```
 *          aleatoria (que eh armazenada para comparacao apos  a          *
 *          decodificacao) que entra no codificador. O ruido e a          *
 *          quantizacao aparecem posteriormente.                          *
 *                                                                        *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                                                                          }

procedure GERA_INFO (var U_bin: vector; var U_dec: integer);
var
      I: integer;
begin
      U_dec:= 0;
      For I:= 1 to k do
        begin
              U_bin[I]:= trunc(round(random));
              U_dec:= U_dec + Expon (2,k-I)*U_bin[I];
        end;
end;

procedure ARMAZENA_INFO (var Info, Pos: integer; var Reg: vector_exp);
begin
      Reg[Pos]:= Info;
end;

procedure GERA_SAIDA (var U_b, E: vector; var U_d, S: integer);
var
      Out, Soma: vector;
      Est      : integer;
begin
      ACTUAL_ADDER (U_b, E, Soma);
      CALC_OUT (U_b, E, Soma, Out);
      ACTUAL_MEMO (U_b, E, Soma);
      DECIMAL (Out, n, S);
end;


(* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *                                                                        *
 *          A PROCEDURE ABAIXO GERA A SAIDA DO CANAL                      *
 *                                                                        *
 *    Sao levados em consideraçao  o ruido,  gerado pelas procedures      *
 *    anteriores, e a constelaçao particular que se esta utilizando,      *
 *    que no caso eh uma constelaçao 16_QAM. Assume-se decisao suave.     *
 *                                                                        *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *)


procedure CHANNEL_OUT_16QAM_SOFT (var Saida: vector; S:integer);
var
     N1, N2 : real;
     Constel: array[0..15,1..2] of real;
begin
      Constel[0,1]:=  b;   Constel[0,2]:=  b;
      Constel[1,1]:=  a;   Constel[1,2]:=  b;
      Constel[2,1]:=  a;   Constel[2,2]:=  a;
      Constel[3,1]:=  b;   Constel[3,2]:=  a;
      Constel[4,1]:= -a;   Constel[4,2]:=  b;
```

```
        Constel[5,1]:= -b;   Constel[5,2]:=  b;
        Constel[6,1]:= -b;   Constel[6,2]:=  a;
        Constel[7,1]:= -a;   Constel[7,2]:=  a;
        Constel[8,1]:= -a;   Constel[8,2]:= -a;
        Constel[9,1]:= -b;   Constel[9,2]:= -a;
       Constel[10,1]:= -b;  Constel[10,2]:= -b;
       Constel[11,1]:= -a;  Constel[11,2]:= -b;
       Constel[12,1]:=  b;  Constel[12,2]:= -a;
       Constel[13,1]:=  a;  Constel[13,2]:= -a;
       Constel[14,1]:=  a;  Constel[14,2]:= -b;
       Constel[15,1]:=  b;  Constel[15,2]:= -b;


       GAUSS_NOISE (Desvio, N1, N2);

       N1:= (N1 + Constel[S,1])*7;
       N2:= (N2 + Constel[S,2])*7;

       If N1 >= 6.5
       then Saida[1]:= 7
       else If N1 < -6.5
            then Saida[1]:= -7
            else Saida[1]:= round (N1);

       If N2 >= 6.5
       then Saida[2]:= 7
       else If N2 < -6.5
            then Saida[2]:= -7
            else Saida[2]:= round (N2);
end;



(* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *                                                                           *
 *        AS PROCEDURES  ABAIXO  CALCULAM  A  METRICA  ASSOCIADA  A          *
 *        CADA RAMO DA TRELICA,  LEVANDO EM CONTA O TIPO DE DECISAO          *
 *        (SUAVE, NO CASO), O TIPO DE DISTANCIA ( EUCLIDES )  E   O          *
 *        TIPO DE CONSTELACAO UTILIZADA NA TRANSMISSAO (16_QAM).             *
 *                                                                           *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *)


procedure GERA_METRICA_16QAM (var MET: metrica);
var
      CONSTEL: array[0..15,1..2] of real;
      I, J, K: integer;
begin

      CONSTEL[0,1]:=  b;   CONSTEL[0,2]:=  b;
      CONSTEL[1,1]:=  a;   CONSTEL[1,2]:=  b;
      CONSTEL[2,1]:=  a;   CONSTEL[2,2]:=  a;
      CONSTEL[3,1]:=  b;   CONSTEL[3,2]:=  a;
      CONSTEL[4,1]:= -a;   CONSTEL[4,2]:=  b;
      CONSTEL[5,1]:= -b;   CONSTEL[5,2]:=  b;
```

```
      CONSTEL[6,1]:= -b;   CONSTEL[6,2]:=  a;
      CONSTEL[7,1]:= -a;   CONSTEL[7,2]:=  a;
      CONSTEL[8,1]:= -a;   CONSTEL[8,2]:= -a;
      CONSTEL[9,1]:= -b;   CONSTEL[9,2]:= -a;
     CONSTEL[10,1]:= -b;  CONSTEL[10,2]:= -b;
     CONSTEL[11,1]:= -a;  CONSTEL[11,2]:= -b;
     CONSTEL[12,1]:=  b;  CONSTEL[12,2]:= -a;
     CONSTEL[13,1]:=  a;  CONSTEL[13,2]:= -a;
     CONSTEL[14,1]:=  a;  CONSTEL[14,2]:= -b;
     CONSTEL[15,1]:=  b;  CONSTEL[15,2]:= -b;

     For I:= 0 to 15 do
       For J:= -7 to 7 do
         For K:= -7 to 7 do
           MET[I, J, K]:= round (sqrt (sqr(7*CONSTEL[I,1] - J) +
                                       sqr(7*CONSTEL[I,2] - K))*2);
end;


procedure BRANCH_MET_16QAM_UNGERB (var Out1: vector; var Hip, Res: integer);
begin
      Res:= (METRICA_16QAM (Hip, Out1[1], Out1[2]))
end;



{* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *                                                                                                                     *
 *                                                                                                                     *
 *          AS PROCEDURES ABAIXO AUXILIAM A GERAÇAO DAS TRAJETORIAS:                                                   *
 *                                                                                                                     *
 *          PATH_MET: calcula a  metrica da trajetoria ate o momento                                                   *
 *                    atual, incorporando o valor obtido para o ramo                                                   *
 *                    em questao.                                                                                      *
 *                                                                                                                     *
 *          SELECTION: seleciona o sobrevivente, ou seja a trajetoria                                                  *
 *                     com minima metrica.                                                                             *
 *                                                                                                                     *
 *          ACTUALIZE: atualiza a tabela de trajetorias, para cada                                                     *
 *                     estado, com base na decisao da procedure acima.                                                 *
 *                     Nessa tabela, a procedure ACTUALIZE armazena as                                                 *
 *                     trajetorias atraves dos  bits de informaçao ja                                                  *
 *                     decodificados.                                                                                  *
 *                                                                                                                     *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *)


procedure  PATH_MET (var Einic, Peso_Ramo, Res: integer);
begin
     Res:= Table[Einic, -1, Pres] + Peso_Ramo
end;


procedure SELECTION (var Ramo, Metrica, Min_Metrica, Min_Ramo: integer);
begin
     If Metrica < Min_Metrica
     then  begin
               Min_Metrica:= Metrica;
```

```
                Min_Ramo:= Ramo
            end
      else  If Metrica = Min_Metrica
            then  begin
                        randomize;
                        If round(random) = 0
                        then  begin
                                    Min_Metrica:= Metrica;
                                    Min_Ramo:= Ramo;
                              end;
                  end;
end;


procedure ACTUALIZE (Ramo, Metrica, Efin: integer; var Tab: tabela;
                     var Hip: tridim);
var
     Traj, Einic : integer;
begin
     TAB[Efin, -1, Fut]:= Metrica;
     Einic:= Hip[Efin, Ramo, 2];
     Traj:= Start;
     while Traj <> Stop do
     begin
          Tab[Efin, Traj, Fut]:= Tab[Einic, Traj, Pres];
          INCREMENTA (Traj);
     end;
Tab[Efin, Stop, Fut]:= Hip[Efin, Ramo, 3];
end;



(* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *                                                                                               *
 *                          GERAÇAO DE TRAJETORIAS                                               *
 *                                                                                               *
 *                                                                                               *
 *         A procedure seguinte gera a tabela de trajetorias para                                *
 *         todos os estados a partir da saida do canal, dos dados                                *
 *         sobre o codificador e das procedures anteriores.                                      *
 *                                                                                               *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *)


procedure GEN_PATH (var Tab: tabela; var Hip: tridim; var Sai: integer);
var
     Efin, Branch, Min, Decision, Metric, Weight: integer;
     Estimate: vector;
begin
     CHANNEL_OUT_16QAM_SOFT (Estimate, Sai);
     For Efin:= 0 to ExpM do
     begin
          Min:= MaxInt;
          For Branch:= 1 to ExpK do
          begin
               BRANCH_MET_16QAM_UNGERB (Estimate, Hip[Efin, Branch, 1], Weight
               PATH_MET (Hip[Efin, Branch, 2], Weight, Metric);
```

```
                SELECTION (Branch, Metric, Min, Decision)
            end;
            ACTUALIZE (Decision, Min, Efin, Tab, Hip);
      end
end;
```

```
(* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *                                                                             *
 *         AS PROCEDURES SEGUINTES AUXILIAM A MANIPULAÇAO DA TABELA             *
 *         DE TRAJETORIAS:                                                     *
 *                                                                             *
 *         INIC_TAB  : inicializa a metrica de cada trajetoria com o           *
 *                     mesmo valor (zero);                                     *
 *                                                                             *
 *         CHANGE_TAB: a tabela "presente" armazena o conjunto das             *
 *                     trajetorias atuais e a tabela "futuro" o das            *
 *                     trajetorias extendidas por mais um ramo. A              *
 *                     procedure troca uma pela outra ao final de cada         *
 *                     iteraçao;                                               *
 *                                                                             *
 *         REDUCTION : de tempos em tempos os valores das metricas das         *
 *                     trajetorias devem ser normalizados. Esta proce-         *
 *                     dure realiza a normalizaçao.                            *
 *                                                                             *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *)
```

```
procedure INIC_TAB ( var T: tabela);
var

     I : integer;
begin
     For I:= 0 to (EXPON (2,m) - 1) do
     begin
          T[I, -1, Pres]:=0;
          T[I, -1, Fut]:= 0
     end
end;
```

```
procedure CHANGE_TAB (var P, F: integer);
begin
     P:= F;
     F:= (F + 1) mod 2
end;
```

```
procedure REDUCTION (var TAB: tabela);
var
     E, Med : integer;
begin
     Med:= 0;
     For E:= 0 to (EXPON (2,m) - 1) do
       Med:= Med + (Tab[E,-1,Fut] div (ExpM + 1));
```

```
      For E:= 0 to (EXPON (2,m) - 1) do
        Tab[E,-1,Fut]:= Tab[E,-1,Fut] - Med;
end;



(* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *                                                                                               *
 *         AS PROCEDURES QUE SE SEGUEM REALIZAM A DECODIFICAÇAO DOS                              *
 *         BITS DE INFORMAÇAO E A ESTATISTICA DOS ERROS COMETIDOS.                               *
 *                                                                                               *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *)


procedure FRAME_DECOD (var Tab: tabela; var Saida: integer);
var
     E, Dec, Min : integer;
begin
     Min:= MaxInt;
     For E:= 0 to  ExpM do
     If Tab[E,-1,Fut] < Min
     then  begin
                Min:= Tab[E,-1,Fut];
                Dec:= Tab[E,Start,Fut]
           end;
     Saida:= Dec;
end;


procedure ERROR_COUNT (var Erro: integer; var Decod: integer);
var
     I           : integer;
     Estim, Info: vector;
begin
     If Decod <> Reg[Start]
     then  begin
                BINARY (Decod, k, Estim);
                BINARY (Reg[Start], k, Info);
                For I:= 1 to k do
                  If Estim[I] <> Info[I]
                  then  Erro:= Erro + 1;
           end;
end;



(* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *                          ENTRADA DE DADOS                                                     *
 *                                                                                               *
 *          Estas procedures fazem a leitura dos dados de entrada,                               *
 *          referentes  a taxa do codigo,  as caracteristicas  do                                *
 *          codificador e a relaçao Eb/No. Admite-se a constelaçao                               *
 *          16_QAM  e  uma decodificaçao suave,  com 16 niveis por                               *
 *          dimensao.                                                                            *
 *                                                                                               *
```

```
end. * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                                                                             }

procedure CODE_INPUT;
var
      Alfa: char;
begin
      writeln ('':10,'DADOS DO CODIGO A SER SIMULADO'); writeln;
      writeln ('':10,'Taxa do codigo - k/n');
      write ('':10,'k = ');  readln (k);
      write ('':10,'n = ');  readln (n);
      writeln;
      writeln ('':10,'Numero de memorias no codificador');
      write ('':10,'m = ');  readln (m);
      writeln;
      writeln ('':10,'Numero de somadores no codificador');
      write ('':10,'s = ');  readln (s);

      For I:= 1 to Dim do
        For J:= 1 to Dim do
          Link[I,J]:= 0;
      writeln;
      writeln ('':14,'Entre com a matriz de ligaçao');
      writeln ('':10,'Indique as posiçoes (I,J) iguais a 1');
      writeln;

      repeat
             write ('':10,'I = ');  readln (I);
             write ('':10,'J = ');  readln (J);
             Link [I,J]:= 1;
             writeln ;

             writeln ('':10,'Outra posiçao?  S)im  N)ao');
             Alfa:= UpCase(ReadKey);
             while  (Alfa <> 'N') and (Alfa <> 'S') do
             begin
                  write(^G,^G,'COMANDO INVALIDO - Outra posiçao?  S)im  N)ao '
                  Alfa:= UpCase(ReadKey); writeln;
             end;
      until  Alfa = 'N';
end;

procedure DATA_INPUT;
var
      Alfa    : char;
      Num_Bits: real;
begin
      write ('Deseja alterar comprimento de decodificaçao?');
      writeln (' (Valor atual = ',Delay:2,')');
      write ('':25,'S)im   N)ao');
      Alfa:= UpCase(ReadKey);
      while (Alfa <> 'S') and (Alfa <> 'N')  do
      begin
            writeln;
            writeln (^G,^G,'COMANDO INVALIDO  -  S)im  N)ao');
            Alfa:= UpCase(ReadKey);
      end;
      If Alfa = 'S'
      then  begin
```

```
                writeln;
                write ('':10,'Comprimento de decodificaçao = ');
                readln (Delay);
          end;
      writeln;
      write ('':10,'Valor de Eb/No = ');  readln (EbNo);
      writeln; writeln;
      writeln ('Entre com o numero total de bits para a simulaçao');
      writeln;
      write ('':10,'Numero de bits = '); readln (Num_Bits);
      If Num_Bits < (k*2501)
      then  Num_Iter:= 1
      else  Num_Iter:= round (Num_Bits/(5000*k));
end;




{* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *                                                                                                 *
 *                          PROGRAMA PRINCIPAL                                                     *
 *                                                                                                 *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                                                                                                    }



begin

      ClrScr;
      CODE_INPUT;

      writeln; writeln;
      writeln ('':20,'MATRIZ DE LIGAÇAO');
      For I:= 1 to (k + m + s) do
        begin
              writeln; write ('':20);
              For J:= 1 to (n + m + s) do
                write (Link[I,J]:2);
        end;
      writeln; writeln; writeln;

repeat

      DATA_INPUT;

      writeln; writeln; writeln;
      writeln ('':10,'Entre com o tipo de saida desejada:');
      writeln ('':10,'Con      -  para saida na tela');
      writeln ('':10,'Prn      -  para saida na impressora');
      writeln ('':10,'Outros   -  para saida em arquivo (nome do arquivo)');
      writeln;
      write ('':10,'=======) '); readln (Arq);
      Assign (Sai, Arq);
      Rewrite (Sai);
```

```
      randomize;

(* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *                                                                         *
 *                    INICIALIZAÇAO DE VARIAVEIS AUXILIARES                *
 *                                                                         *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *)

      SIGMA (Desvio);

      ExpM:= EXPON(2,m)-1;
      ExpK:= EXPON(2,k);

      Pres := 0; Fut:= 1;
      Start:= 0; Stop:= -1;
      Error:= 0;

      For I:= 1 to m do
        Est_Inic[I]:=  0;



(* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *                                                                         *
 *                        GERAÇAO DAS TABELAS                              *
 *                                                                         *
 *          Aqui sao geradas as varias tabelas usadas ao longo da          *
 *          simulaçao. Ver a descriçao das respectivas procedures.         *
 *                                                                         *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *)


      GEN_HYPOTHESIS (Hypothesis);
      GERA_METRICA_16QAM (Metrica_16QAM);
      INIC_TAB (Table);

      repeat
            Stop:= Stop + 1;
            CHANGE_TAB (Pres, Fut);
            GERA_INFO (Info_b, Info_d);
            GERA_SAIDA (Info_b, Est_Inic, Info_d, Output);
            ARMAZENA_INFO (Info_d, Stop, Reg);
            GEN_PATH (Table, HypoThesis, Output);
      until   Stop = (Delay-1);
      REDUCTION (Table);

      writeln;
      writeln;
      writeln ('':10,^G,'Pre-processamento Terminado - Iniciando Iteraçoes');
      writeln ('':20,'Numero de Iteracoes = 20');
      write(n;
```

```
{*                          INICIO DAS ITERAÇOES                              *
 *                                                                            *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *}


      For J:= 1 to 20 do
      begin
            Counter:= 0;
            While (Counter < Num_Iter) do
            begin
                  Counter:= Counter + 1;
                  Control:= 0;
                  repeat
                        Control:= Control + 1;
                        INCREMENTA (Start);
                        INCREMENTA (Stop);
                        CHANGE_TAB (Pres, Fut);
                        GERA_INFO (Info_b, Info_d);
                        GERA_SAIDA (Info_b, Est_Inic, Info_d, Output);
                        ARMAZENA_INFO (Info_d, Stop, Reg);
                        GEN_PATH (Table, HypoThesis, Output);
                        FRAME_DECOD (Table, Out);
                        ERROR_COUNT (Error, Out);

                  until  (Control = 250);

                  REDUCTION (Table);
            end;


            randomize;

{* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *                                                                            *
 *                    SAIDA DOS RESULTADOS DA SIMULAÇAO                       *
 *                                                                            *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                                                                              }
            writeln;
            writeln (^G,^G,'':20,'Iteraçao numèro  ',J:3);

            Iter:= Counter*J*k*250;
            writeln (Sai);
            writeln (Sai,'':20,'Numero de bits         =  ',Iter:8:1);
            writeln (Sai,'':20,'Numero de bits errados =  ',Error:5);
            writeln (Sai,'':20,'Probabilidade de Erro  =  ',(Error/Iter):11);
            writeln (Sai);

      end;

      For l:= 1 to 20 do
        write (^G);

      writeln; writeln;
      writeln ('':10,'PROCESSAMENTO TERMINADO');
      writeln ('':10,'DESEJA NOVA SIMULAÇAO?  S)im   N)ao');
      Alfa:= UpCase(ReadKey);
      while  (Alfa <> 'N') and (Alfa <> 'S')  do
```

```
      begin
            writeln (^G,^G,'COMANDO INVALIDO  -  S)im  N)ao');
            Alfa:= UpCase(ReadKey);
      end;
      writeln; writeln;

until  Alfa = 'N';
```