Decodificação Turbo de Código Produto de Paridade Simples

Dayani Adionel Guimarães - INATEL

***Resumo* ¾ Esse artigo apresenta os resultados de uma investigação sobre o desempenho de códigos produto de paridade simples, sem paridade das paridades, em canal AWGN. A decodificação é iterativa, usando ao algoritmo Log-MAP simplificado. É verificado que excelentes resultados podem ser alcançados com o uso desses códigos que têm como principal característica a simplicidade de codificação e de decodificação. Uma forma especial de implementação de paridades diagonais é apresentada nesse trabalho e mostra ser uma alternativa que traz consideráveis melhorias de desempenho, mantendo ainda baixas a complexidade da codificação e da decodificação.**

Abstract* ¾ *This article presents the results of an investigation about the performance of single parity check product codes (SPC-PC), without check-on-checks, in AWGN channels. These codes are iteratively decoded using a simplified version of the Log-MAP algorithm. It is shown that very good performance results can be obtained with these powerful and simple codes. A special form of diagonal parity is presented here and shows considerable performance improvements, without a significant increasing in the coding and decoding complexity.

***Palavras chave* ¾ Codificação de canal; (de)codificação Turbo; decodificação iterativa; FEC; códigos produto de paridade simples; códigos concatenados; códigos de bloco Turbo; paridade diagonal.**

I. INTRODUÇÃO

Os Códigos Turbo [Ber96] correspondem a implementações onde há a concatenação de códigos componentes de tal sorte que a decodificação destes códigos possa ser realizada de forma independente e iterativa, sendo utilizado o resultado da decodificação de um dos códigos com o objetivo de melhorar a confiabilidade da decisão das etapas seguintes no processo iterativo. Em uma das possíveis estruturas de concatenação, a informação de entrada de um de dois codificadores corresponde a uma versão entrelaçada da entrada do outro codificador componente (concatenação paralela), com processos de decodificação iterativa utilizando algoritmos do tipo SISO (*Soft-Input Soft-Output*). Nesses processos de decodificação SISO, informações sobre a confiabilidade ou qualidade da decodificação de um dos códigos componentes (*soft output* ou informação de estado do canal) são passadas para o processo de decodificação do outro código, de forma iterativa, de tal sorte que a cada ciclo tem-se maior precisão na estimação do bit transmitido.

Em [Ber96][1] pôde-se obter, a 10^{-5} de taxa de erro de bit, uma E_b/N_o distante apenas 0,7 dB da Capacidade de Shannon e, mais recentemente, em [Nic97a], um desempenho resultando em uma E_b/N_o distante 0,27 dB da capacidade de canal foi obtido. Mas apesar dos incríveis ganhos de desempenho que podem ser obtidos, um dos grandes obstáculos a serem transpostos ainda se refere à simplificação dos algoritmos de decodificação. Aqueles considerados ótimos (segundo o critério MAP símbolo-a-símbolo), como regra geral são também complexos, demandando altas velocidades de processamento para que aplicações reais possam ser vislumbradas. Várias pesquisas têm sido então encaminhadas no sentido de desenvolver novos algoritmos ótimos ou sub-ótimos ou ainda no sentido de modificar algoritmos ótimos, tornando-os sub-ótimos, porém com menor grau de complexidade.

Esquemas de codificação de canal com decodificação Turbo permitem ainda que seja possível, a um moderado grau de complexidade, operar com valores de E_b/N_o abaixo da taxa de corte do canal. Até então sabia-se que esta se tratava de uma tarefa possível, mas de alta complexidade [Hag96], pois a taxa de corte era considerada como a "capacidade prática" do canal.

Basicamente tem-se duas grandes famílias de códigos Turbo: uma baseada na concatenação de códigos convolucionais (CTC, *Convolutional Turbo Code*) e outra baseada na concatenação de códigos de bloco (BTC, *Block Turbo Code*). Grande atenção passou a ser dirigida a implementações de códigos Turbo a partir de códigos convolucionais após a publicação da reconhecidas contribuições de C. Berrou e A. Glavieux em 1993 e 1996 [Ber96]. Recentemente, grande interesse tem sido demonstrado por implementações de processos de decodificação iterativa (incluindo decodificação Turbo) baseados em códigos de bloco. Dentre várias iniciativas podem-se mencionar [Nic97a], [Nic97b], [Dav01], [Kan94], [Pyn98], [Hag96], [Ran01], [Hun98a] e [Hun98b].

Várias empresas já adotaram códigos de bloco Turbo como base de seus *chips* codificadores e decodificadores. Dentre elas podem ser citadas a *Advanced Hardware Architecture* (AHA, http://www.aha.com/) e a *Efficient Channel Coding* (ECC, http://www.eccincorp.com/).

[1] O resultado de simulação apresentado em [Ber96] foi obtido com o uso de codificadores convolucionais recursivos e sistemáticos, um extenso bloco de *interleaving* pseudo-aleatório entre os codificadores (65.536 bits), 18 iterações e alguns ajustes no algoritmo BCJR.

Os principais objetivos dessas implementações se referem à possibilidade de redução na complexidade de decodificação, o aumento na velocidade de decodificação (característica intimamente ligada ao primeiro objetivo e à simplificação no processo de entrelaçamento temporal) e também o aumento de desempenho em relação aos CTCs, principalmente para códigos de altas taxas [Pyn98], [Hag96]. Esta característica é também verificada em [Dav01]. Para baixas taxas, os CTCs geralmente apresentam melhor desempenho.

Outra vantagem aparente dos BTCs se refere à considerável redução (em certos casos) do patamar de erro de bit intransponível que é percebido em todos os códigos Turbo. Esse patamar ocorre devido à existência de palavras código com baixo peso e à alta correlação entre as seqüências codificadas de cada código componente. Seu valor pode ser reduzido através da adequada implementação dos blocos de entrelaçamento temporal entre os códigos utilizados [Bar98b]. Esse processo, enquanto reduz a correlação entre as saídas codificadas de cada código componente, também reduz o número de palavras código de baixo peso.

A concatenação dos códigos componentes pode ser implementada na forma serial ou paralela. Para os BTCs, a concatenação serial é mais vantajosa que a concatenação paralela [Dav01].

Quando há concatenação serial de dois ou mais códigos de bloco separados por entrelaçadores temporais (*interleavers*), tem-se uma estrutura de código produto e, dessa forma, o processo de decodificação de BTCs pode-se valer de algoritmos derivados de algoritmos iterativos de decodificação de códigos produto, desde que estes algoritmos forneçam decisões suaves de saída.

Esse artigo encontra-se organizado da seguinte maneira: na seção II são abordados alguns conceitos fundamentais e necessários ao entendimento do processo de decodificação Turbo apresentado. Na seção III são citados, de forma resumida, alguns dos principais algoritmos de decodificação Turbo, destacando-se aqueles utilizados para decodificação Turbo de códigos de bloco. Ainda na seção III são apresentados alguns algoritmos que não são caracterizados com algoritmos de decodificação turbo, mas que podem servir à implementação de algoritmos Turbo. A Seção IV apresenta resultados de simulação de um algoritmo Log-MAP simplificado (sub-ótimo), para códigos produto de paridade simples, sem paridade das paridades, em canal AWGN, implementado com o *Mathcad® 2001 Professional*. Nesta seção também é apresentado o conceito de uma forma especial de implementação de *paridades diagonais*. A seção V conclui o trabalho, com comentários adicionais sobre os resultados apresentados e sobre futuras propostas de trabalho. Um apêndice é também fornecido e apresenta o código do programa utilizado nas simulações consideradas neste trabalho.

II. FUNDAMENTOS PARA DECODIFICAÇÃO DE CÓDIGOS TURBO

Nesta seção é apresentada uma síntese dos principais conceitos aplicados ao processo de decodificação iterativa dos códigos de bloco Turbo. Complementos a esta abordagem podem ser encontrados em várias referências, dentre elas podendo ser citadas [Ber96], [Skl97] e [Hag96].

A. Revisão sobre funções de verossimilhança

Dentre os conceitos fundamentais relacionados ao processo de decisão sobre os bits transmitidos em sistemas de comunicação digital destaca-se o Teorema de Bayes. Trata-se de uma ferramenta útil para teste de hipóteses, relacionando probabilidades condicionais e conjuntas de eventos A e B [Skl97]

$$P(A \mid B)P(B) = P(B \mid A)P(A) = P(A,B) \qquad (1)$$

Um formato mais freqüentemente encontrado para a fórmula de Bayes (1) é

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)} \qquad (2)$$

No estudo de telecomunicações a fórmula de Bayes toma sua versão contínua dada por

$$P(d = i \mid x) = \frac{p(x \mid d = i)P(d = i)}{p(x)},\ i = 1,...M \qquad (3)$$

onde $P(d = i \mid x)$ representa a probabilidade *a-posteriori* de d ter assumido o valor i ($d = i$ ter sido transmitido), tendo sido observada a "saída" x (sinal recebido). Essa probabilidade se refere a um certo tipo de "refinamento" sobre o nosso conhecimento prévio, após a observação da saída do experimento. A função $p(x \mid d = i)$ é a função de verossimilhança (*likelihood function*) da saída x, dada a ocorrência (transmissão) da classe de sinais $d = i$. O valor $P(d = i)$ é a probabilidade *a-priori* de ocorrência do evento $d = i$ (probabilidade de transmissão da classe de sinais $d = i$). Finalmente, $p(x)$ é a função densidade de probabilidade total sobre todo o espaço das M classes de sinais recebidos, ou seja

$$p(x) = \sum_{i=1}^{M} p(x \mid d = i)P(d = i) \qquad (4)$$

Os mais utilizados critérios de decisão aplicados em sistemas de comunicação digital são baseadas no teste de hipóteses instrumentalizado pelo Teorema de Bayes. São eles: o critério do máximo-a-posteriori (MAP, *Maximum A-Posteriori*) e o critério de máxima verossimilhança (ML, *Maximum Likelihood*). O critério MAP estabelece, dada como exemplo uma sinalização binária antipodal ($d \in \{+1, -1\}$, $M = 2$ e $i = 1, 2$):

$$P(d=+1\mid x) \underset{H2}{\overset{H1}{\gtrless}} P(d=-1\mid x) \qquad (5)$$

o que significa: escolha a hipótese 1 (o sinal +1 ou bit 1 foi transmitido) se a probabilidade *a-posteriori* $P(d = +1 \mid x)$ for maior que $P(d = -1 \mid x)$ e escolha a hipótese 2 (o sinal –1 ou bit 0 foi transmitido) em caso contrário.

Utilizando o teorema de Bayes pode-se escrever a expressão (5) na forma equivalente

$$p(x\mid d=+1)P(d=+1) \underset{H2}{\overset{H1}{\gtrless}} p(x\mid d=-1)P(d=-1)$$

ou

$$\frac{p(x\mid d=+1)P(d=+1)}{p(x\mid d=-1)P(d=-1)} \underset{H2}{\overset{H1}{\gtrless}} 1 \qquad (6)$$

Na expressão (6) tem-se, no termo da esquerda, a conhecida razão de verossimilhança. Para sinais transmitidos com mesma probabilidade (equiprováveis), a expressão (6) reduz-se a

$$\frac{p(x\mid d=+1)}{p(x\mid d=-1)} \underset{H2}{\overset{H1}{\gtrless}} 1 \qquad (7)$$

o que corresponde ao critério de máxima verossimilhança (ML). Verifica-se, portanto, que para sinais equiprováveis[2], o critério ótimo MAP se reduz ao critério ML (também ótimo neste caso).

A título de exemplo, a Fig. 1 ilustra o aspecto das funções de verossimilhança para sinais recebidos na saída do detetor (entrada do elemento ou bloco de decisão) em um sistema de comunicação com sinalização antipodal. Os sinais transmitidos, $\in \{+1, -1\}$, estão contaminados por ruído aditivo gaussiano branco (AWGN, *Additive White Gaussian Noise*). A escolha pelas hipóteses 1 ou 2 anteriormente definidas restringe-se à observação da magnitude do sinal de saída do detetor, ou seja, se este sinal é maior que o limiar de decisão γ, decida por +1 (bit 1, ou 0, conforme o mapeamento na transmissão); caso contrário decida por –1 (bit 0, ou 1). Em caso de igualdade, decida arbitrariamente por 0 ou por 1. Se as probabilidades *a-priori* são iguais, ou seja $P(d = +1) = P(d = -1)$, o critério ML se torna equivalente ao critério MAP e, neste caso, o limiar γ se encontra no ponto intermediário entre as médias das funções de verossimilhança (na Fig. 1, $\gamma = 0$).

[2] A equiprobabilidade dos bits de informação normalmente ocorre nas típicas fontes de em sistemas de comunicação e, quando não ocorre, esta equiprobabilidade é aproximadamente garantida através do uso de randomizadores (*scramblers*).

Ao contrário do que acontece com o critério MAP, o critério de máxima verossimilhança é comumente utilizado na decodificação de seqüências inteiras, e não símbolo-a-símbolo como ilustrado pela Fig. 1. Neste caso o critério ML minimiza a probabilidade de decisão pela seqüência incorreta.

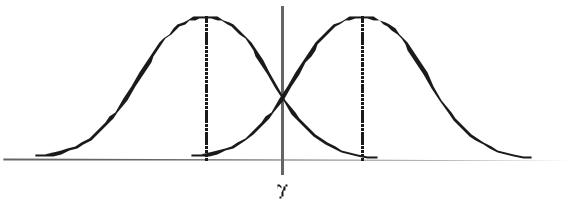

Figura 1 - *Ilustração das funções de verossimilhança para sinalização antipodal.*

B. A Informação Extrínseca

Por razões de simplificação em cálculos, às vezes torna-se mais vantajoso operar com as funções de verossimilhança no domínio logarítmico no qual multiplicações e divisões são convertidas em adições e subtrações, respectivamente. Tomando-se o logaritmo do termo da esquerda da Eq. 6 obtém-se a métrica denominada Razão de Log-Verossimilhança (LLR, *Log-Likelihood Ratio*)

$$L(d\mid x) = \log\left[\frac{p(x\mid d=+1)}{p(x\mid d=-1)}\right] + \log\left[\frac{P(d=+1)}{P(d=-1)}\right] \qquad (8)$$

que pode ser expressa como

$$L(d\mid x) = L(x\mid d) + L(d) \qquad (9)$$

A quantidade $L(x\mid d)$ é a LLR das medidas do canal x, dadas as condições de transmissão $d = +1$ ou $d = -1$, e $L(d)$ é a LLR *a-priori* do bit d.

Uma pequena simplificação na notação da expressão (9) resulta em [Skl97]

$$L'(\hat{d}) = L_c(x) + L(d) \qquad (10)$$

onde $L'(\hat{d})$ é a LLR estimada de um determinado bit d, apresentada à entrada do decodificador de canal e $L_c(d)$ é a LLR da medida do estado de canal, correspondente ao bit d, obtida na saída do detetor.

Pode-se provar [Ber96] que a LLR na saída do decodificador de canal (saída suave) é dada por [Skl97]

$$L(\hat{d}) = L'(\hat{d}) + L_e(\hat{d}) \qquad (11)$$

onde $L'(\hat{d})$ é a LLR de um bit de dados (valor *a-posteriori*) e $L_e(\hat{d})$ é definida como a *LLR Extrínseca*, ou *Informação Extrínseca* [Ber96], desse bit.

A informação extrínseca pode ser interpretada como a quantidade de informação adicionada ao valor real de entrada do decodificador (*soft input*) para formar o valor real de saída (*soft otput*) [Hun98] ou a

quantidade de informação obtida a partir do processo de decodificação [Skl97], porém independente dos valores antes deste processo (na saída do detetor) [Ber96].

A informação extrínseca pode ainda ser definida como a diferença entre a LLR calculada na saída do estágio de decodificação (saída suave) e a informação intrínseca representada por uma LLR realimentada à entrada do estágio de decodificação; é a informação adicional obtida através da exploração das dependências que existem entre o bit de informação de interesse e os demais bits codificados em um bloco [Hay01], segundo cada regra de codificação específica.

Combinando (10) e (11) pode-se escrever a LLR obtida através do processo de decodificação SISO como

$$L(\hat{d}) = L_c(x) + L(d) + L_e(\hat{d}) \qquad (11)$$

Esta LLR contém a informação necessária para decodificação abrupta do bit *d*, dada pela polaridade de $L(\hat{d})$, bem como a informação de confiabilidade dessa decisão, dada pela magnitude de $L(\hat{d})$. Relembrando, $L_c(x)$ representa a LLR obtida a partir da medida do canal, na saída do detetor, $L(d)$ é a LLR *a-priori* do bit *d* e $L_e(\hat{d})$ é a informação extrínseca gerada pelo, e dependente do, processo de decodificação.

C. Decodificação Turbo

A Fig. 2 representa o diagrama de um decodificador SISO para um código sistemático, usado na construção de um decodificador iterativo (Turbo).

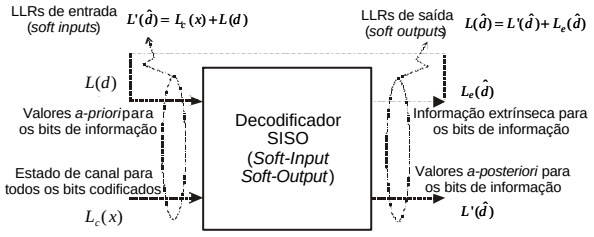

Figura 2 - *Ilustração do decodificador Turbo (adaptado de* [Hag96] *e* [Skl97]*).*

A saída do detetor é a LLR que contém a informação de canal obtida para todos os bits codificados, mais a informação *a-priori* dos bits de informação. No processo de decodificação iterativa, estes valores *a-priori* não são conhecidos antes da primeira iteração completa e normalmente são considerados nulos (probabilidades *a-priori* iguais) nesta etapa. A saída do decodificador SISO é composta pelos valores *a-posteriori* dos bits de informação, mais a informação extrínseca obtida pelo processo de decodificação. Esse valor de informação extrínseca é realimentado à entrada do SISO, como o valor da LLR *a-priori* para a próxima iteração. Este procedimento faz com que os novos valores de LLR de entrada possam produzir valores de LLR de saída mais confiáveis, iteração a iteração.

Como será justificado na seção IV, com o auxílio de um exemplo, a magnitude da informação extrínseca de um bit em particular é igual à menor das magnitudes das LLRs dos outros bits envolvidos; e o sinal dessa informação extrínseca é igual ao próprio sinal dessa LLR, se a decisão abrupta levar a uma palavra código válida (se for verificada a paridade), e tem seu sinal invertido se esta paridade não é verificada. Esta operação faz com que, a cada iteração, os valores das LLRs de saída do decodificador SISO sejam reduzidos ou fortalecidos, dependendo da verificação ou não da paridade; e a quantidade de redução ou fortalecimento dependerá do menor valor de LLR dos demais bits envolvidos [Hun98]. Esse menor valor pode ser interpretado como a menor confiabilidade obtida no resultado de verificação da paridade.

As Figs. 3(a) e 3(b) ilustram o que está registrado no parágrafo anterior. Nelas estão sendo mostradas, a cada iteração (de um total de 3 iterações), as alterações nos valores das saídas suaves (LLRs de saída) do processo de decodificação Turbo de um código produto 2D (bidimensional) de paridade simples sem paridade das paridades, formado a partir de códigos componentes idênticos (3, 2).

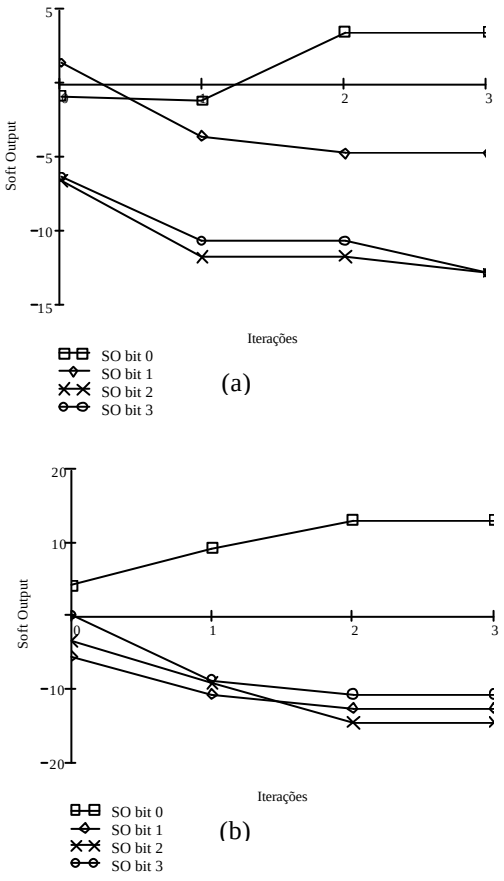

Figura 3 - *Ilustração da evolução das soft outputs a cada iteração no processo de decodificação Turbo (seqüência transmitida = 1, 0, 0, 0).*

No exemplo ilustrado pela Fig. 3(a) os bits de informação transmitidos correspondem à seqüência {1, 0, 0, 0}. Observa-se que antes da primeira iteração (instante zero, no eixo horizontal) os valores das LLRs (neste instante iguais $L_c(x)$, as medidas ou informações de canal) levariam a uma decisão abrupta pela seqüência incorreta {0, 1, 0, 0}. Ao final da primeira iteração verifica-se que o sinal da LLR para o bit 1 (*soft output* SO bit 1) foi invertido. Ao final da segunda iteração verifica-se que o sinal da LLR para o bit 0 (SO bit 0) foi invertido. Todas as outras LLRs tiveram seus sinais inalterados. A magnitude de todas as LLRs foram, contudo, alteradas, a cada iteração, para um valor igual à soma da informação de canal com a informação extrínseca correspondente. Para o exemplo dado, após a segunda iteração os valores de LLR de saída já levam à decodificação correta da seqüência original. De fato, como observado em [Ran01], após a *D*-ésima iteração as alterações nas LLRs de saída já não alterarão a decisão, caso a quantidade de erros induzidos esteja dentro da capacidade de correção do código produto de dimensão *D*.

A Fig. 3(b) mostra uma situação na qual a decisão abrupta a partir das saídas do detector (informação de canal) levaria a um erro, mas já na primeira iteração este erro seria corrigido. A partir daí percebe-se, a cada iteração, um "reforço" nas LLRs de saída do decodificador SISO, numa tendência média de confirmar a decisão que já poderia ser tomada logo após a primeira iteração. Nenhuma melhoria estaria sendo obtida em termos da decisão final neste caso. Este fato indica a necessidade de se definir um critério de parada no processo iterativo de tal sorte que seja reduzido o tempo total de decodificação e se possa aumentar a vazão (*throughput*) de dados em um sistema de comunicação real. Em [Hag96] são abordados alguns desses critérios e em [Hun98a, p. 43, item 4.7] é sugerido um método extremamente simples de verificação de convergência no processo iterativo e de interrupção desse processo caso a convergência seja verificada. A idéia em [Hun98a] é adequada à decodificação iterativa de códigos produto com paridade simples.

A Fig. 4 mostra, de forma complementar às Figs. 3(a) e 3(b), o refinamento das LLRs de saída de um decodificador Turbo usando algoritmo Log-MAP sub-ótimo. O código utilizado neste exemplo é um código SPC 4D, formado a partir de códigos componentes iguais (7, 6), e os bits codificados são considerados todos nulos (símbolos +1).

Percebe-se que, com o aumento no número de iterações, até aproximadamente um número igual à dimensão do código produto, há aumento na dispersão da distribuição da LLR de saída, porém a sua área à esquerda do limiar de decisão 0 é reduzida (melhoria das confiabilidades), levando a uma menor probabilidade de erro na decisão dos símbolos – percebe-se que há um deslocamento para a direita da média das LLRs à medida que é completada cada iteração.

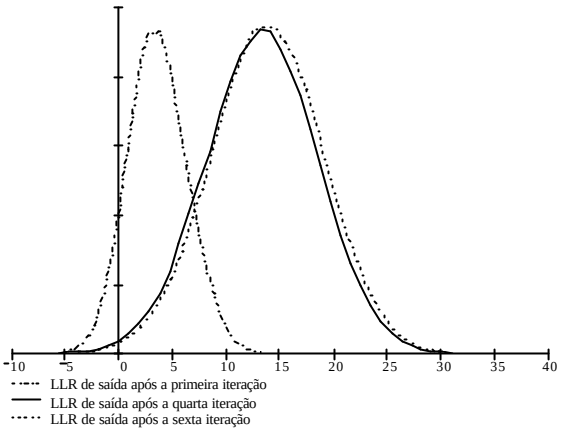

Figura 4 - *Variação da dispersão da função densidade de probabilidade de LLRs de saída de um decodificador Turbo em função do número de iterações.*

III. ALGORITMOS DE DECODIFICAÇÃO TURBO E ALGORITMOS SISO PARA COMPOSIÇÃO DE ALGORITMOS DE DECODIFICAÇÃO TURBO

Nesta seção é apresentada uma síntese sobre algumas das propostas de algoritmos de decodificação Turbo publicadas em veículos ou eventos de divulgação especializados, com destaque para aqueles utilizados para decodificação Turbo de códigos de bloco. Alguns algoritmos não específicos para decodificação Turbo são também citados como base na implementação da decodificação Turbo por operarem com entradas suaves e por permitirem que decisões suaves sejam obtidas em suas saídas. Pelo que está exposto a seguir, nota-se que quase todos os algoritmos de decodificação iterativa para códigos de bloco sugeridos nesses últimos anos são baseados em variações e/ou derivações dos algoritmos BCJR [Bah74] ou Chase [Cha72].

O uso de processos de decodificação iterativa (incluindo decodificação Turbo) é possível para qualquer código convolucional ou de bloco na forma sistemática[3], ou ainda para uma combinação entre estes. Os algoritmos de decodificação Turbo podem ser implementados a partir de qualquer regra de decodificação que tenha como princípio a geração de decisões suaves a partir de entradas suaves (medidas de canal na saída de um filtro casado, por exemplo) e que permita que alguma informação adicional obtida pelo processo de decodificação possa ser realimentada à entrada para, a cada iteração, melhorar a confiabilidade das saídas suaves e, portanto, das decisões.

Em alguns algoritmos a informação extrínseca é extraída das probabiliades *a-posteriori* obtidas e somente ela é realimentada como forma de melhorar a confiabilidade das decisões. Em outros, a informação extrínseca não é separada das probabiliades *a-posteriori*. Neste último caso as próprias

[3] Embora seja possível a implementação de decodificação turbo para códigos não sistemáticos, existem várias razões (veja, por exemplo, [Hag96, p.433) para que seja adotado o uso de códigos sistemáticos.

probabilidades *a-posteriori* atualizadas são realimentadas à entrada a cada iteração de modo a gerar novas probabilidades.

A. Algoritmo de Chase (1972)

O algoritmo de Chase não é um algoritmo específico para decodificação Turbo, mas, por ser um algoritmo que pode produzir decisões suaves (SISO), seu princípio pode ser aplicado em algoritmos de decodificação Turbo

Na verdade trata-se de um conjunto de três algoritmos similares [Cha72], e é um algoritmo sub-ótimo (desempenho próximo daquele obtido segundo o critério de máxima verossimilhança, ML) adequado ao processo de decodificação suave de códigos de bloco lineares.

Sabe-se que quando é utilizada uma procura exaustiva pela palavra código ótima que minimiza a distância euclidiana entre a palavra recebida e o conjunto de possíveis palavras-código, segundo o critério ML, a complexidade de decodificação aumenta exponencialmente em n, o número de bits de um bloco de saída do codificador de canal. Por esta razão tem-se procurado desenvolver algoritmos de decodificação com o propósito de reduzir esta complexidade.

Em 1972, D. Chase propôs um algoritmo utilizando informação de estado de canal (*soft output*), baseado seguinte observação: para altos valores de RSR, a palavra-código correta se encontra localizada, com alta probabilidade, numa esfera de raio $(d - 1)$, centrada nas coordenadas do vetor $\boldsymbol{Y} = (y_1, ..., y_l, ..., y_n)$, onde $y_l = 0.5[1 + sign(r_l)]$, $y_l \in \{0,1\}$, $\boldsymbol{R} = (r_1, ... r_l, ..., r_n)$ é o vetor recebido e d é a distância de Hamming mínima do código. Utilizando-se esta regra reduz-se o número de palavras-código investigadas àquelas localizadas dentro da esfera de raio $(d - 1)$. O processo de procura dessas palavras-código mais prováveis utiliza informações de confiabilidade obtidas a partir do vetor $\boldsymbol{R}$; o processo é descrito detalhadamente em [Cha74] e em [Pyn98] tem-se a aplicação deste algoritmo num processo de decodificação iterativa. Vale citar que em algumas situações específicas, as medidas de confiabilidade do algoritmo de Chase coincidem com aquelas utilizadas no algoritmo de Wagner [Sil54] (veja [Pyn98, p. 1005, *Step 3*]). O algoritmo de Wagner, em termos de efeito, se assemelha ao processo de inversão de polaridades na decodificação Turbo, porém atuando por uma única vez (o algoritmo de Wagner não é iterativo), não possuindo, por essa razão, a potencialidade de melhoria das LLRs.

B. Algoritmo BCJR (1974)

Trata-se de um algoritmo ótimo para decodificação iterativa, baseado no critério MAP símbolo-a-símbolo, para códigos convolucionais e de bloco sistemáticos. O algoritmo BCJR possui este nome devido às iniciais dos autores [Bah74]. O conhecido trabalho de C. Berrou *et. al.* [Ber96] utilizou uma versão ligeiramente modificada do algoritmo BCJR para produzir os surpreendentes resultados já mencionados neste artigo. Em [Hag96], J. Hagenauer apresenta uma abordagem sobre algoritmo BCJR, incluindo análises de desempenho em comparação a outros algoritmos de decodificação Turbo, como o SOVA e o Log-MAP, descritos aqui, nas seções III-*C* e III-*E*.

O Algoritmo BCJR é um dos mais utilizados na composição de processos de decodificação iterativa. Dentre as inúmeras referências que abordam o assunto podem ainda serem citadas: [Bar96], [Ber96], [Hay01] e [Ran01], além da publicação original [Bah74].

C. Algoritmo SOVA (1989)

O algoritmo SOVA (*Soft-Input Soft-Output Viterbi Algorithm*) [Hag89] corresponde a uma modificação do algoritmo de Viterbi convencional de tal sorte que decisões suaves apropriadas sejam geradas e se possa implementar processos iterativos de decodificação. Esse algoritmo é analisado em [Hag96] para códigos convolucionais sistemáticos utilizando treliça. Trata-se de uma implementação sub-ótima, mas que apresenta aproximadamente (e em certos casos) a metade da complexidade de alguns algoritmos Log-MAP.

D. Algoritmo de Kaneko (1994)

O algoritmo de Kaneko [Kan94] se presta à decodificação suave de códigos de bloco lineares. Seu desempenho é equivalente àquele obtido segundo processos de decodificação de máxima verossimilhança, nos quais é minimizada a probabilidade de decodificação errada de uma palavra-código, para palavras código eqüiprováveis.

Similar ao algoritmo 2 de Chase, o algoritmo de Kaneko também tem como princípio a geração de um conjunto reduzido de palavras código candidatas, dentro do qual, com elevada probabilidade (1, neste algoritmo), a palavra código correta estaria contida. Contudo, apresenta melhor desempenho e menor complexidade que o algoritmo de Chase.

Assim como o algoritmo de Chase, o algoritmo de Kaneko não é um algoritmo específico para decodificação Turbo, mas seu princípio pode ser utilizado na construção de algoritmos de decodificação Turbo. O algoritmo de Kaneko, apesar de produzir decisões abruptas, permite que sejam calculadas confiabilidades dessas decisões, transformando-as em decisões suaves, como por exemplo em [Dav01].

E. Algoritmo Log-MAP (1995)

Algoritmos ótimos ou sub-ótimos no domínio das log-verossimilhanças (para simplificação e transformação de algoritmos MAP símbolo-a-símbolo) são conhecidos como algoritmos Log-MAP [Rob95]. Suas versões sub-ótimas podem apresentar desempenho muito próximo ao desempenho do algoritmo MAP símbolo-a-símbolo, principalmente quando há um refinamento na simplificação anteriormente citada,

através de um fator de correção [Hag96, p. 435]. Desempenhos próximos ao de um algoritmo MAP símbolo-a-símbolo também podem ser obtidos às custas da inclusão do cálculo de um certo tipo de informação extrínseca também para os bits de paridade e o uso de paridade das paridades na estrutura de um código produto [Ran01]. Em[Hag96] são apresentadas discussões sobre a aplicação dos algoritmos MAP símbolo-a-símbolo BCJR e Log-MAP em Códigos de Bloco Turbo, genericamente, e em Códigos Produto de Paridade Simples, especificamente. Em [Hag96] e [Skl97], um algoritmo de decodificação no domínio logarítmico (versão sub-ótima do Log-MAP) é proposto e ilustrado em simples exemplos de decodificação Turbo. Esse algoritmo possui complexidade de implementação extremamente baixa e pode apresentar excelente desempenho, como será verificado mais à diante.

F. Algoritmo de Pyndiah (1998)

Em [Pyn98] é apresentado um algoritmo sub-ótimo para decodificação iterativa de códigos de bloco baseados em códigos produto. Esse algoritmo utiliza um decodificador SIHO (*Soft-Input Hard-Output*) baseado no algoritmo 2 de Chase [Cha72], seguido de cálculos para obter informações de confiabilidade (*soft decisions*) a partir das saídas abruptas (*hard decisions*) do decodificador. O desempenho desse processo é aproximadamente ótimo e a proposta apresenta uma boa solução de compromisso entre desempenho e complexidade, sendo bastante atrativa para aplicações práticas.

G. Algoritmo de Dave (2001)

No algoritmo de Dave [Dav01] é utilizado o algoritmo de Kaneko [Kan94] (*Soft-Input Hard-Output*), seguido de um procedimento similar ao de Pyndiah [Pyn98] para transformar em decisões suaves as decisões abruptas do algoritmo de Kaneko. O algoritmo de Dave apresenta certa flexibilidade na relação complexidade *versus* desempenho, possibilitando sua implementação em versões mais complexas que apresentam desempenho ótimo (como o algoritmo de Kaneko) até versões menos complexas com desempenho sub-ótimo (como aquele obtido pelo algoritmo de Pyndiah). Em todos os casos, contudo, a proposta de Dave é menos complexa que as propostas de Kaneko e Pyndiah.

IV. ALGORITMO LOG-MAP APLICADO NA DECODIFICAÇÃO TURBO DE CÓDIGOS PRODUTO DE PARIDADE SIMPLES

Introduzidos por Elias [Eli54], os códigos produto pertencem à família de códigos de arranjo [Hon97] que podem ser decodificados iterativamente e, além de grande simplicidade de implementação, apresentam enorme flexibilidade em termos de adequação da taxa e do comprimento do bloco e, em certos casos, grande capacidade de correção de erros.

Um código produto de dimensão D pode ser definido a partir do comprimento do bloco de informação em cada dimensão $\{k_1, k_2, \ldots k_i, \ldots k_D\}$, $i = 1, \ldots, D$, onde são utilizados códigos sistemáticos $(n_i, k_i, d_{\min i})$. O código produto completo resultante possui blocos de comprimento

$$N = \prod_{i=1}^{D} n_i \tag{11}$$

Se $r = k_i/n_i$ é a taxa do código componente na dimensão i, a taxa do código produto será

$$r = \prod_{i=1}^{D} r_i \tag{12}$$

E a distância mínima será (para o código completo)

$$d_{\min} = \prod_{i=1}^{D} d_{\min_i} \tag{13}$$

O código produto completo possui, além dos bits de paridade gerados pelo processo de codificação em cada dimensão, os bits de paridade gerados a partir da paridade das paridades (*check on checks*) – pode-se interpretar essa implementação como sendo a concatenação serial dos códigos componentes, separados por entrelaçadores temporais. Um código produto incompleto, sem a paridade das paridades, é mais simples de ser implementado, mas possui desempenho inferior, assintoticamente. Isto se deve ao fato principal da distância mínima do código incompleto ser menor que a do código completo [Ran01] – pode-se interpretar essa implementação como sendo a concatenação paralela dos códigos componentes, separados por entrelaçadores temporais.

Em [Ben96] e [Ben98] são introduzidas úteis ferramentas para análise, em termos de previsão e/ou validação, do desempenho de códigos concatenados nas formas paralela e serial, tanto para códigos componentes de bloco quanto para convolucionais. A análise apresentada nestes artigos é baseada em limitantes de probabilidade de erro calculados a partir de uma "distribuição média" de pesos do código concatenado. Esta distribuição considera que o *interleaver* existente entre os códigos componentes é um dispositivo probabilístico (entrelaçamento temporal uniforme), levando, então, a uma análise de comportamento médio do desempenho da concatenação.

A concatenação serial de códigos componentes segundo estrutura D-dimensional de um código produto completo permite que códigos não sistemáticos sejam também utilizados, apesar dessa alternativa não ser freqüentemente citada na literatura especializada. Tanto a taxa final do código quanto a distância mínima e também o comprimento do bloco serão os mesmos. As maiores vantagens de se utilizar códigos sistemáticos como componentes se refere à possibilidade de codificação em paralelo, a menos da geração da paridade das paridades que deve aguardar que todas as paridades sejam geradas, e certa facilidade no processo de decodificação, no

mapeamento reverso das palavras-código nos respectivos bits de informação.

A concatenação paralela de códigos componentes segundo estrutura de dimensão *D* de um código produto incompleto é de implementação mais simples, mas traz consigo considerável redução de desempenho, e redução na taxa quando não são utilizados códigos sistemáticos. Em geral percebe-se (e isto não vale somente para códigos de bloco) que a concatenação paralela é melhor naquelas situações onde deseja-se a operação do sistema em questão a baixos valores de E_b/N_o, porém podendo suportar altos valores de taxa de erro de bit, acima do patamar ou "joelho" da curva taxa de erro *versus* E_b/N_o. Esse "joelho" corresponde à um comportamento de saturação na taxa de erro de bit, fazendo com que esta seja reduzida apenas de forma bastante suave, mesmo com significativos aumentos na relação E_b/N_o. Esse fenômeno ocorre devido, principalmente, à presença de um número significativo de palavras-código de baixo peso. Portanto, em valores mais altos de E_b/N_o, a taxa de erro de bit passa a ser governada predominantemente por estas palavras de baixo peso.

Na concatenação serial tem-se maiores valores de E_b/N_o a altos valores de taxa de erro de bit, mas o decréscimo dessa taxa com o aumento de E_b/N_o é consideravelmente mais abrupto, possibilitando transmissões praticamente "livres de erro" a valores de relação sinal-ruído mais altos. Em [Ben96] e [Ben98] são apresentados ricos comentários e critérios de projeto baseados nas constatações aqui reproduzidas.

A Fig. 5 ilustra a estrutura de um código produto bi-dimensional (2D) incluindo-se a paridade das paridades (código completo, ou concatenação serial)[4].

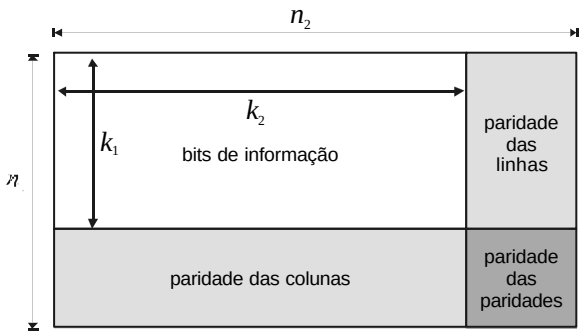

Figura 5 – *Estrutura do código produto bidimencional completo.*

Dentre os códigos produto, os códigos produto de paridade simples (SPC-PC, *Single Parity Check Product Code*) têm merecido destaque, pois, apesar de serem pouco eficientes quando utilizados isoladamente, representam ótima oportunidade para a construção de códigos multidimensionais concatenados bastante eficientes [Hag01], [Ran01]. Além disso, a oportunidade para utilização de algoritmos de decodificação Turbo para esses códigos traz novas perpectivas de melhoria de desempenho aliadas a complexidades de implementação reduzidas.

Para um SPC-PC de dimensão *D* formado a partir da concatenação serial dos códigos componentes em cada dimensão, tem-se a distância mínima

$$d_{\min} = 2^D \tag{14}$$

Se os códigos componentes são códigos (*n*, *k*) idênticos, a taxa do SPC-PC de dimensão *D* completo será

$$r = \left(\frac{n-1}{n}\right)^D \tag{15}$$

Para um SPC-PC de dimensão *D* incompleto (sem a paridade das paridades, ou formado a partir de concatenação paralela) tem-se a distância mínima

$$d_{\min} = D+1 \tag{16}$$

e se os códigos componentes são códigos (*n*, *k*) idênticos, a taxa do SPC-PC de dimensão *D* incompleto será

$$r = \frac{1}{1+D/(n-1)} \tag{17}$$

Como já citado, em [Hag96] e [Skl97] são apresentados exemplos de decodificação Turbo de um código produto de paridade simples bidimensional utilizando uma versão sub-ótima do critério MAP no domínio das log-verossimilhanças. Contudo, o desempenho desse tipo de implementação não é satisfatório, principalmente se o código produto utilizado possui baixa dimensionalidade e se não há transmissão da(s) paridade(s) das paridades (*check on checks*), caso dos exemplos de [Hag96] e [Skl97]. A justificativa para a não utilização da(s) paridade(s) das paridades nesses exemplos se deve ao fato de ser necessária somente a informação extrínseca dos bits de informação para implementação do algoritmo, pois as probabilidades de envio das palavras código são determinadas somente pelas probabilidades *a-priori* desses bits de informação. De fato, o uso da paridade das paridades em um código produto de paridade simples 2D não resulta em melhorias de desempenho. Esse fato é comprovado através de resultados de simulação apresentados em [Ran01] e confirmados nas simulações realizadas para a composição desse trabalho.

Segundo outros resultados apresentados em [Ran01], os conceitos utilizados em [Hag96] e [Skl97] podem ser facilmente estendidos para códigos produto SPC com mais de duas dimensões, incluindo a paridade das paridades no processo de codificação e a obtenção de um certo tipo de informação extrínseca sobre essa paridade das paridades de tal sorte que a todos os bits, não somente aos bits de informação,

[4] Em [Hon97, p. 7] é descrito o processo de implementação da matriz geradora de um código produto de paridade simples completo através do produto de *Kronecker* entre as matrizes geradoras dos códigos componentes em cada dimensão.

como em [Hag96] e [Skl97], estejam associadas a informação *a-priori* e a extrínseca. Dessa forma o algoritmo se torna ótimo, inclusive ultrapassando, em alguns casos, o desempenho do algoritmo MAP símbolo-a-símbolo BCJR.

Considerando-se que um algoritmo apropriado, com desempenho ótimo ou pouco distante do ótimo, esteja sendo utilizado, com o aumento da dimensão do código produto de paridade simples os retornos em termos de desempenho são sempre menores entre cada aumento de dimensão, fato que pode ser constatado pelos resultados de simulação apresentados em [Ran01] e confirmados aqui. Maiores retornos podem ser obtidos com o uso de códigos componentes mais potentes.

Os ganhos de desempenho em função do aumento da dimensionalidade ocorrem principalmente devido ao aumento da distância mínima do código. Adicionalmente, a cada iteração em um esquema de decodificação Turbo de um código produto multidimensional, percebe-se maiores retornos em desempenho em relação a esquemas bidimensionais. Este fato é devido principalmente à redução na correlação entre as probabilidades envolvidas nos cálculos das LLRs a cada iteração. Melhores desempenhos também podem ser obtidos às custas do aumento do tamanho dos blocos para um código de dimensão *D* [Hag96]. Esse fato é mais claramente percebido quando os códigos componentes possuem, individualmente, maior capacidade de correção de erros.

Como verificado através de simulações, para códigos produto de paridade simples 2D o aumento no tamanho dos blocos não traz grandes ganhos em termos de desempenho. Há inclusive uma redução de desempenho com o uso de blocos de tamanho muito elevado, o que é justificado pela predominância da influência do número de palavras código de baixo peso, nestes casos. Para baixos valores de relação sinal-ruído percebe-se também uma redução no desempenho quando aumenta-se o tamanho dos blocos em códigos 2D.

Uma outra interessante constatação, verificada também em [Ran01], se refere a uma melhoria (não muito significativa[5], contudo) no desempenho do processo de correção de erros, para códigos produto SPC completos, de dimensão maior que 2, quando é incluído um processo de entrelaçamento temporal aleatório (*random interleaving*) entre as etapas de codificação em cada dimensão. Para dimensão igual a 2 não é percebida nenhuma melhoria de desempenho; ao contrário, é percebida uma redução nesse desempenho. Um importante resultado adicional se refere à possibilidade de redução do patamar de erro de bit. Contudo, o entrelaçamento temporal aleatório, apesar de reduzir o número de palavras código de baixo peso reduz, também, a distância mínima do código.

[5] O código SPC-PC utilizado em [Ran01] é um código completo que, por si só, já apresenta baixos valores de patamar de erro de bit e um decréscimo mais abrupto da taxa de erro de bit (BER) com o aumento da relação E_b/N_o.

É importante relembrar que, ao contrário do que acontece com os códigos de bloco Turbo com concatenação paralela, o processo de entrelaçamento é crucial à melhoria do desempenho dos códigos Turbo convolucionais e dos códigos de bloco Turbo com concatenação serial. Para estes últimos casos, tanto a profundidade quanto a "aleatoriedade" do processo de entrelaçamento temporal tem relação direta com o desempenho do processo de decodificação Turbo [Bar96], [Ben96], [Ben98]. Para os códigos de bloco com concatenação paralela, a profundidade do entrelaçamento temporal pode se restringir ao comprimento do bloco de mensagem ou, no pior caso, do bloco completo (informação mais paridade), sem melhorias de desempenho significativas a partir daí.

Em casos onde a dimensionalidade do código produto de paridade simples deva ser aumentada em demasia, acarretando em um excessivo aumento no tamanho dos blocos, alternativamente pode-se implementar o código com códigos componentes mais eficientes, por exemplo códigos Hamming. Como já citado, surpreendentes resultados utilizando códigos componentes Hamming, operando com blocos de tamanho não muito elevado (1023,1013), são reportados em [Nic97].

Em [Hun98a] Andrew Hunt sugere uma forma adicional para melhorar a relação de compromisso entre complexidade e desempenho dos códigos produto de paridade simples *D*-dimensionais. Trata-se da inclusão de *D*-1 conjuntos de paridade obtidos através de operações de codificação "diagonais" em cada dimensão. A essa implementação foi dado o nome de *Hyper Codes*.

A. Implementação de um algoritmo Log-MAP para decodificação Turbo de códigos produto SPC

Nesta seção é apresentado o processo de construção de um código produto de paridade simples (SPC-PC) e a implementação de um SPC-PC de dimensão *D* incompleto, seguindo uma idéia similar à de [Hun98a] para geração de paridades diagonais, utilizando o algoritmo de decodificação Log-MAP sugerido em [Hag96], com algumas das generalizações citadas em [Ran01]. Os resultados das simulações são apresentados na seção IV-B.

O algoritmo MAP símbolo-a-símbolo no domínio logarítmico (Log-MAP) aplicado à decodificação Turbo de códigos produto de paridade simples apresenta complexidade extremamente pequena e ainda permite que, com um ligeiro aumento de complexidade, códigos potentes sejam construídos. Estes dois motivos tornam atrativas estas implementações, principalmente em aplicações onde complexidade e desempenho devam apresentar grande flexibilidade em termos da realização prática do processo.

Como já citado, a razão de verossimilhança, LLR (saída suave), obtida através do processo de decodificação SISO pode ser expressa pela equação (11), repetida aqui por conveniência:

$$L(\hat{d}) = L_c(x) + L(d) + L_e(\hat{d}) \quad (18)$$

onde, recordando, $L_c(x)$ representa a LLR obtida da medida do canal, na saída do detetor; $L(d)$ é a LLR *a-priori* do bit d e $L_e(\hat{d})$ é a informação extrínseca gerada a partir do processo de decodificação.

Como ilustração desse processo de decodificação, considere um código produto de paridade simples bidimensional. O algoritmo de decodificação opera na seguinte seqüência:

1. Inicialmente faça todas as log-probabilidades *a-priori* dos bits de informação iguais a zero (probabilidades *a-priori* iguais a 0,5), $L(d) = 0$;
2. Conhecida a lógica de codificação, que é a relação de dependência entre os bits de informação e a paridade, obtenha as informações extrínsecas na dimensão horizontal, $L_{eh}(\hat{d})$;
3. Faça as probabilidades *a-priori* iguais às informações extrínsecas calculadas no passo anterior, $L(d) = L_{eh}(\hat{d})$;
4. Obtenha as informações extrínsecas na dimensão horizontal, $L_{ev}(\hat{d})$, levando em conta as novas probabilidades *a-priori* atualizadas no passo 3.
5. Faça as probabilidades *a-priori* iguais às informações extrínsecas calculadas no passo anterior, $L(d) = L_{ev}(\hat{d})$;
6. Se houver mais iterações, vá ao passo 2, levando em conta os últimos valores de probabilidades *a-priori* atualizados; se não houver mais iterações, vá ao passo 7;
7. A decisão suave será $L(\hat{d}) = L_c(x) + L_{eh}(d) + L_{ev}(d)$.

Essa seqüência pode facilmente ser identificada no algoritmo fornecido no APÊNDICE desse artigo. Para um código de dimensão *D*, deve-se lembrar que a atualização da informação *a-priori* a cada dimensão é feita através da soma das informações extrínsecas calculadas para as *outras* dimensões até aquele momento.

Para realizar os cálculos do processo de decodificação, há que se utilizar um conjunto de ferramentas apropriadas ao problema. A esse conjunto foi dado o nome de álgebra no domínio das log-verossimilhanças (*log-likelihood algerbra*) [Ber96], [Hag96], [Skl97]. Os principais resultados dessa álgebra são resumidos em seguida.

Partindo de uma função de log-verossimilhança tal qual aquela da expressão (8), pode-se demonstrar que [Hag96], [Skl96][6], para variáveis estatisticamente independentes *d*

$$L(d_1) \boxplus L(d_2) = L(d_1 \oplus d_2) = \log_e\left[\frac{e^{L(d_1)} + e^{L(d_2)}}{1 + e^{L(d_1)}e^{L(d_2)}}\right] \quad (19)$$
$$\approx \text{sign}\left[L(d_1)\right] \times \text{sign}\left[L(d_2)\right] \times \min\left[\,|L(d_1)|, |L(d_2)|\,\right]$$

com as regras adicionais

$$L(d) \boxplus \infty = L(d),\ L(d) \boxplus -\infty = -L(d) \text{ e } L(d) \boxplus 0 = 0 \quad (20)$$

A expressão (19) pode ser generalizada, levando ao resultado [Hag96]

$$\sum_{j=1}^{J} \boxplus L(d_j) = \log_e\left[\frac{1 + \prod_{j=1}^{J} \tanh\left(L(d_j)/2\right)}{1 - \prod_{j=1}^{J} \tanh\left(L(d_j)/2\right)}\right] \quad (21)$$
$$= 2\,\text{arctanh}\left(\prod_{j=1}^{J} \tanh\left(L(d_j)/2\right)\right)$$

Esta expressão também pode ser aproximada para

$$\sum_{j=1}^{J} \boxplus L(d_j) = L\left(\sum_{j=1}^{J} \oplus d_j\right)$$
$$\approx \left(\prod_{j=1}^{J} \text{sign}[L(d_j)]\right) \min_{j=1\ldots J} \left|L(d_j)\right| \quad (22)$$

O valor de $L_c(x)$, em canal AWGN, pode ser determinado a partir da função

$$L_c(x_k) = \log_e\left[\frac{p(x_k \mid d_k = +1)}{p(x_k \mid d_k = -1)}\right]$$
$$= \log_e\left[\frac{\frac{1}{s\sqrt{2p}}\exp\left(-\frac{1}{2}\left(\frac{x_k - E_s}{s}\right)^2\right)}{\frac{1}{s\sqrt{2p}}\exp\left(-\frac{1}{2}\left(\frac{x_k + E_s}{s}\right)^2\right)}\right] \quad (23)$$
$$= \frac{4E_s}{2s^2}x_k = \frac{4E_s}{N_0}x_k$$

onde σ^2 é a variância do ruído gaussiano na saída do filtro casado do receptor, no instante de decisão, e E_s é a componente de sinal nesse mesmo instante, considerada igual à energia do símbolo recebido.

B. Resultados de simulação

Nesta seção são fornecidos alguns resultados de simulação utilizando o algoritmo proposto no APÊNDICE na forma apresentada, com alguma adaptação ou pequena modificação conforme o caso analisado.

Em [Hun98a], a implementação de paridade diagonal em estruturas SPC *D*-dimensionais é apresentada como processo chave para construção dos denominados *Hyper-codes*. Contudo, para que essa implementação leve a um aumento da distância mínima do código, certas regras de combinação dos bits devem ser seguidas. À melhor das regras foi dado o nome de geometria preferida em [Hun98a], mas com as restrições: os códigos componentes em cada dimensão devem ter o mesmo comprimento e impar. Na regra sugerida no presente artigo, basta que os códigos componentes sejam de mesmo comprimento em cada dimensão, o que efetivamente aumenta a

[6] A expressão (19) apresenta ligeiras diferenças nas publicações [Hag96] e [Skl97]. Este fato é devido à adoção do elemento nulo = 1 no primeiro caso e 0 no segundo. Neste segundo caso, deve ser utilizado um fator multiplicador $(-1)^{k-1}$ na expressão (19), sendo k o número de bits de informação do código componente ($k_1 = k_2 = k$), de forma a levar em conta se k é par ou ímpar. Isto deve ser feito se o mapeamento dos bits nos símbolos codificados for {0, 1} → {-1, +1) (elemento nulo = 0). Se for {0, 1} → {+1, -1) (elemento nulo = 1), não é necessário o fator multiplicador

flexibilidade em termos de escolha da taxa do código e do comprimento dos blocos.

A Fig. 6(a) mostra, como exemplo, um arranjo cujos elementos correspondem aos índices dos bits de informação de um código SPC 2D formado a partir de códigos componentes $(n, k) = (6, 5)$. A Fig. 6(b) mostra um novo arranjo obtido a partir de deslocamentos circulares nas linhas do arranjo (a). A primeira linha (linha 0) não sofre nenhum deslocamento e a linha l, $l = 1, 2, \ldots (k - 1)$ sofre l deslocamentos circulares à direita. Se aplicarmos paridades nas colunas do arranjo da Fig. 6(b), estaremos aplicando paridades diagonais à direita no arranjo (a). É importante notar que essa simples regra faz com que nenhuma operação de geração de paridade envolva mais de um bit de uma mesma linha e de uma mesma coluna do arranjo original, o que garante máxima eficácia do processo em termos de aumento da distância mínima do código e, consequentemente, de desempenho.

(a) $\begin{pmatrix} 0 & 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 & 9 \\ 10 & 11 & 12 & 13 & 14 \\ 15 & 16 & 17 & 18 & 19 \\ 20 & 21 & 22 & 23 & 24 \end{pmatrix}$ (b) $\begin{pmatrix} 0 & 1 & 2 & 3 & 4 \\ 9 & 5 & 6 & 7 & 8 \\ 13 & 14 & 10 & 11 & 12 \\ 17 & 18 & 19 & 15 & 16 \\ 21 & 22 & 23 & 24 & 20 \end{pmatrix}$

Figura 6 – *Ilustração da regra de implementação da paridade diagonal à direita*

Esse conceito de paridades diagonais à direita pode ser aplicado também às diagonais à esquerda do arranjo da Fig. 6(a). Para tanto, basta que os deslocamentos à direita realizados na implementação das paridades diagonais à direita sejam realizados à esquerda, para as paridades diagonais à esquerda. A Fig. 7 ilustra esse processo. Se aplicarmos paridades nas colunas do arranjo da Fig. 7(b), estaremos aplicando paridades diagonais à esquerda no arranjo (a). Quando ambas as formas de paridade diagonal foram implementadas nas simulações, o termo "paridade diagonal dupla" foi utilizado para discriminá-las.

(a) $\begin{pmatrix} 0 & 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 & 9 \\ 10 & 11 & 12 & 13 & 14 \\ 15 & 16 & 17 & 18 & 19 \\ 20 & 21 & 22 & 23 & 24 \end{pmatrix}$ (b) $\begin{pmatrix} 0 & 1 & 2 & 3 & 4 \\ 6 & 7 & 8 & 9 & 5 \\ 12 & 13 & 14 & 10 & 11 \\ 18 & 19 & 15 & 16 & 17 \\ 24 & 20 & 21 & 22 & 23 \end{pmatrix}$

Figura 7 - *Ilustração da regra de implementação da paridade diagonal à esquerda*

A Fig. 8 mostra o desempenho (taxa de erro de bit, BER, & probabilidade de erro de bit, BEP, *versus* relação sinal ruído média por bit, E_b/N_o em dB) de códigos produto de paridade simples 2D incompletos de várias taxas, com decodificação iterativa utilizando o algoritmo Log-MAP sub-ótimo. Três das quatro situações mostradas (excetuando-se o caso sem codificação, incluído como referência para comparação) utilizam paridades horizontal e vertical, sem paridade das paridades. Uma quarta situação mostra o desempenho de um código similar, contendo paridades obtidas através de combinações diagonais dos bits de informação. O número de iterações em todos os casos é 3.

Através da Fig. 8 pode-se perceber que o desempenho de um código produto SPC bidimensional incompleto não é satisfatório. Mesmo aumentando-se o tamanho dos blocos percebe-se apenas uma melhoria marginal, sendo que esse comportamento é invertido para blocos maiores, situação na qual tem-se desempenhos ainda piores que códigos bidimensionais mais curtos. No caso ilustrado na Fig. 8, o comprimento do bloco para o código de taxa 0,5 é de apenas 8 bits. Para o código de taxa 0,882, esse comprimento cresce para 255 bits e para o código de taxa 0,909 tem-se blocos de tamanho 440 bits. O aumento marginal de desempenho para códigos componentes $(n, k) = (n, n - 1)$ é conseguido até por volta de $k = 15$, com retornos cada vez menores. A partir de $k = 15$ o desempenho começa a sofrer redução. Isto se deve principalmente ao aumento do número de palavras-código de baixo peso, sem que seja alterada a distância mínima do código resultante, que no caso é sempre $D + 1 = 3$.

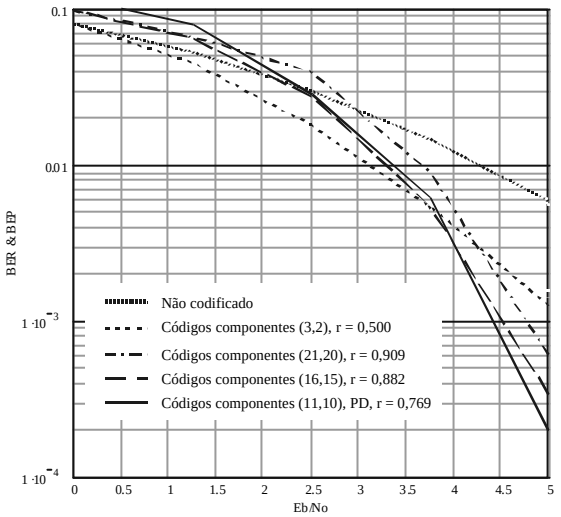

Figura 8 – *Desempenho do código produto bidimensional sem paridade das paridades, para várias taxas, usando algoritmo Log-MAP sub-ótimo.*

A inclusão de uma paridade diagonal foi realizada e mostrou, para o caso 2D, algum ganho de desempenho, mas não a ponto de levar o desempenho de códigos produto SPC bidimensionais com decodificação Turbo a patamares comparáveis a outros esquemas com decodificação iterativa apresentados na literatura. Na Fig. 8 percebe-se que, às custas de uma ligeira redução na taxa do código, pôde-se obter, com códigos componentes (11, 10), praticamente o mesmo desempenho que aquele obtido com códigos componentes (16, 15) sem paridade diagonal. Além disso pôde-se obter uma considerável redução no comprimento dos blocos (de 255 para 130 bits).

Uma segunda paridade diagonal foi incluída em caráter de teste nos esquemas da Fig. 8, mas não apresentou ganhos de desempenho em relação ao esquema com as paridades vertical, horizontal e diagonal simples, além de levar a uma menor taxa do código. Por esta razão os resultados obtidos não foram apresentados.

A Fig. 9 apresenta alguns resultados de simulação para códigos produto de paridade simples *D*-dimensionais, sem paridade das paridades, com e sem paridades diagonais, decodificados com o algoritmo Log-MAP simplificado [Hag96]. Como previsto, com o aumento da dimensão do código tem-se grandes melhorias de desempenho. Os esquemas considerados são códigos produto SPC com 5 dimensões e códigos componentes (8, 7), com e sem paridades diagonais; as taxas dos códigos e comprimentos dos blocos são, respectivamente, 0,538 e 31.213 bits e 0,583 e 28.812 bits. São também considerados os códigos produto SPC com 4 dimensões e códigos componentes (13, 12), com e sem paridades diagonais; as taxas e comprimentos dos blocos para estes casos são 0,706 e 29.380 bits e 0,750 e 27.650 bits, respectivamente. O número de iterações foi feito sempre igual à dimensão do código.

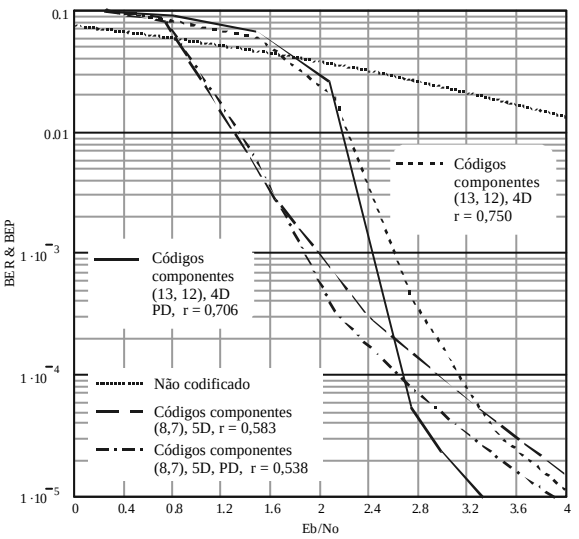

Figura 9 - *Desempenho de códigos produto SPC 5D e 4D com e sem paridades diagonais, usando algoritmo Log-MAP sub-ótimo.*

Como pode-se concluir a partir da Fig. 9, a inclusão de paridades diagonais melhora sensivelmente o desempenho dos códigos sem um significativo aumento no tamanho do bloco ou redução na taxa do código, principalmente em canais com baixa relação sinal-ruído – o problema que ainda persiste é a ocorrência de uma certa saturação na taxa de erro de bit para relações sinal-ruído mais elevadas, causada pela existência de um elevado número de palavras-código de baixo peso. Para o esquema 5D com paridades diagonais percebe-se através da Fig. 9 que, a 10^{-5} de taxa de erro de bit, um desempenho que dista cerca de 3,5 dB da capacidade de canal (0,3 dB para BPSK em AWGN e taxa do código 0,538) pôde ser obtido.

O resultado apresentado na Fig. 9 para o código 5D com componentes (8, 7) e paridades diagonais, para taxas de erro maiores que 10^{-4}, tem desempenho que dista pouco mais de 1 dB daquele apresentado em [Ran01]. Em [Ran01] foram utilizados os mesmos códigos componentes (8, 7), num esquema também 5D, porém com o uso de paridades das paridades (código produto completo, ou concatenação serial) e entrelaçamento temporal aleatório no processo de codificação entre cada dimensão. O algoritmo usado em [Ran01] também sofre uma ligeira modificação em relação ao utilizado nesse trabalho, pois opera com as informações extrínsecas e as probabilidades *a-priori* para todos os bits do bloco (e não somente para os bits de informação, como é o caso do algoritmo aqui considerado). Ainda em [Ran01], a taxa do código é ligeiramente reduzida para aproximadamente 0,513 e o comprimento dos blocos é elevado para 32.768 bits.

O que observa-se através dos resultados aqui apresentados e aqueles em [Ran01] é que, como verificado em [Ben98], pode-se obter desempenhos mais próximos da capacidade de canal a altos valores de BER, acima do "joelho" da curva BER x E_b/N_o, se a concatenação é paralela (código SPC incompleto). Porém, para valores de BER mais baixos, a concatenação serial (código SPC completo) apresenta desempenho mais próximo da capacidade de canal, pois o "joelho" ocorre em um patamar menor de BER.

A Fig. 9 também apresenta o desempenho de um código SPC 4D, composto a partir de códigos componentes (13, 12), com paridades diagonais. Esse código apresentou, para baixos valores de relação sinal-ruído, desempenho inferior aos outros casos com codificação de canal ilustrados na Fig. 9. No entanto, este comportamento é invertido para relações sinal-ruído mais elevadas. Dado que o valor mínimo de E_b/N_o para sinalização BPSK em canal AWGN à taxa do código de 0,706 é de aproximadamente 1,3 dB, esse código apresenta um excelente desempenho, pois opera em uma taxa de erro de bit de 10^{-5} a apenas 2 dB desse limite mínimo. Dada a simplicidade dos processos de codificação e decodificação, o código SPC 4D com componentes (13, 12) é bastante atrativo para implementações práticas onde blocos de tamanho relativamente elevado sejam permitidos.

O desempenho de um código SPC 4D, composto a partir de códigos componentes (13, 12), sem paridades diagonais, também é mostrado na Fig. 9. Nota-se a considerável redução no desempenho se a técnica de paridades diagonais não é utilizada.

O uso de paridade diagonal dupla também foi testado e não trouxe ganhos de desempenho para os códigos *D*-dimensionais considerados na Fig. 9. Por essa razão não foram registrados seus desempenhos.

As Figs. 10(a) e 10(b) correspondem a gráficos de funções de distribuição de pesos de um código SPC 2D formado a partir de códigos componentes (4, 3), com paridades diagonais (Fig. 10(a)) e sem paridades diagonais (Fig. 10(b)). A Fig.10(b) demonstra que a inclusão da paridade diagonal eleva a distância mínima do código e elimina, nos casos analisados para a composição deste artigo, algumas palavras-código de baixo peso. Estas são as principais razões para as

melhorias de desempenho verificadas através da Fig. 9.

Investigações mais detalhadas devem ainda ser efetuadas no sentido de verificar se a redução no número de palavras-código de baixo peso é válida para qualquer dimensão e qualquer código componente do código produto, quando é implementada paridade diagonal.

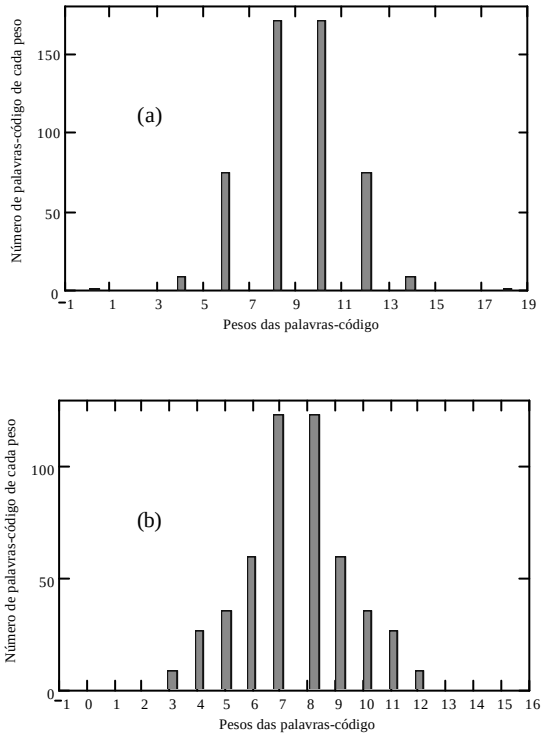

Figura 10 - *Gráfico da função de distribuição de pesos de um código 2D formado a partir de códigos componentes (4, 3), (a) com paridades diagonais (d_{min} = 4) e (b) sem paridades diagonais (d_{min} = 3).*

V. COMENTÁRIOS FINAIS

Esse artigo apresentou uma síntese sobre alguns dos principais algoritmos de decodificação Turbo, com enfoque naqueles utilizados para decodificação iterativa de códigos de bloco, genericamente, e de códigos produto de paridade simples, especificamente. Foram apresentadas várias razões para a utilização dos códigos de bloco Turbo, em vez dos códigos Turbo convolucionais, em situações onde a relação entre complexidade de decodificação e desempenho é crítica. A principal dessas razões se refere à reduzida complexidade de implementação de alguns dos algoritmos citados, com destaque especial para o Log-MAP aplicado na decodificação de códigos produto de paridade simples (SPC-PC). Foi ainda apresentado um algoritmo para decodificação Turbo de códigos produto SPC *D*-dimensionais e introduzida uma forma especial de paridades diagonais. Resultados de simulação foram então apresentados e discutidos.

Nesse trabalho não foi considerado nenhum processo de entrelaçamento temporal dos bits de informação (ou informação mais paridade) entre as etapas de codificação em cada dimensão dos códigos utilizados, diferente daquele que é inerente à estrutura de um código produto sem paridade das paridades. Em [Ran01] pode-se perceber que o uso de entrelaçamento temporal aleatório, para códigos de dimensão maior que 2 e com paridade das paridades, reduz drasticamente o patamar de erro de bit percebido em todos os códigos lá analisados. Em trabalhos futuros pretende-se investigar a influência da variação do comprimento do bloco de entrelaçamento temporal em códigos SPC sem e com paridade das paridades de forma a ratificar, para os SPCs, as informações sobre o desempenho de códigos concatenados segundo [Ben96] e [Ben98]. Pretende-se também investigar a combinação de códigos SPC completos com o uso de paridade diagonal.

Conforme relatado nesse texto, não somente o aumento da dimensão pode melhorar o desempenho de códigos produto com decodificação Turbo. O uso de códigos componentes mais potentes também é um forte fator de melhoria. Como continuidade dos trabalhos aqui apresentados pretende-se investigar o uso de códigos de arranjo generalizado (GAC) [Hon97] na composição de códigos produto *D*-dimensionais, com decodificação Turbo segundo o algoritmo Log-MAP [Hag96] e/ou Dave [Dav01] e/ou BCJR [Bah74].

No algoritmo fornecido no APÊNDICE não foi utilizado um critério de parada, ou seja, o processo iterativo é executado até o fim da última iteração programada. Sabe-se, contudo, que critérios como esse podem ser utilizados para aumentar a vazão de informação em momentos em que a relação sinal-ruído é mais elevada, não sendo necessário que sejam completadas todas as iterações para se atingir uma taxa de erro de bit ou de bloco alvo. Quando algum critério de interrupção do processo iterativo é utilizado, a LLR de saída deve ser atualizada a cada iteração, e não ao final delas. Neste caso, atenção especial deve ser dispensada para que a informação extrínseca calculada a cada iteração não seja somada mais de uma vez na LLR total. No algoritmo apresentado no APÊNDICE, a LLR de saída é formada depois que todas as iterações são completadas; e como não foi utilizado nenhum critério de parada, a atualização dessa LLR poderia ser feita a cada iteração, também tendo-se o cuidado para que a informação extrínseca calculada em uma iteração passada fosse subtraída da nova informação extrínseca, antes que esta última fosse adicionada à LLR de saída, atualizando-a. A adição de um critério de verificação de convergência e de parada poderá ser objeto de estudos futuros.

Observando-se o algoritmo apresentado no APÊNDICE, pode-se perceber que não foi utilizada a expressão simplificada para o cálculo das informações extrínsecas (veja equações (21) e (22)). Contudo, verificou-se através de simulações que a diferença de desempenho é pequena o bastante para que a expressão (22) possa ser utilizada de forma a facilitar uma eventual implementação prática do algoritmo Log-MAP.

Esse trabalho pode ser visto como uma combinação de um texto introdutório (tutorial) com a apresentação de uma aplicação sobre o processo de

decodificação Turbo, sendo ainda apresentada uma proposta de implementação para simulação e análise de resultados a partir da combinação de algumas idéias identificadas na literatura especializada. Os códigos Turbo são um tema relativamente recente e, sem dúvida, representam um campo fascinante e promissor. Apesar do grande volume de textos que têm sido publicados sobre o assunto nos últimos anos, ainda há muito por explorar tanto em termos de possibilidades de implementação quanto em tratamentos matemáticos mais rigorosos, à luz da Teoria da Informação. Esse texto sequer tangencia a real vastidão do assunto, mas, deseja o autor, que ele sirva para que os leitores possam entender os conceitos básicos da decodificação Turbo de códigos de bloco concatenados e sejam motivados a estudos mais avançados sobre o tema.

REFERÊNCIAS

[Bah74] Bahl, L. R., Cocke, J., Jelinek, F, and Raviv, J., "Optimal decoding of linear codes for minimizing symbol error rate", *IEEE Transactions on Information Theory*, pp. 284-287, March 1974.

[Bar96] Barbulescu, S. A., "Iterative Deconding of Turbo Codes and Other Concatenated Codes", *Ph.D. Thesis, Faculty of Engineering, University of South Australia*, February 1996.

[Bar98a] Barbulescu, S. A., and Pietrobon, S. S., "Turbo Codes: a tutorial on a new class of powerful error corecting coding schemes. Part I: Code Structures and Interleaver Design", *Institute for Telecommunications Research, University of South Australia*, October 1998.

[Bar98b] Barbulescu, S. A., and Pietrobon, S. S., "Turbo Codes: a tutorial on a new class of powerful error corecting coding schemes. Part II: Decoder Design and Performance", *Institute for Telecommunications Research, University of South Australia*, October 1998.

[Ben96] Benedetto, S. and Montorsi, G., "Unveiling Turbo Codes: Some Results on Parallel Concatenated Coding Schemes", *IEEE Transactions on Information Theory*, Vol. 42, No 2, March 1996.

[Ben98] Benedetto, S. *et. al*., "Serial Concatenation of Interleaved Codes: Performance Analysis, Design, and Iterative Decoding", *IEEE Transactions on Information Theory*, Vol. 44, No 3, May 1998.

[Ber96] Berrou, C.; Glavieux, A., "Near optimum error correcting coding and decoding: Turbo-Codes", *IEEE Transactions on Communications*, Vol. 44 10, pp. 1261-1271, October 1996.

[Cha72] Chase, D., "A class of algorithms for decoding block codes with channel measurement information", *IEEE Transactions on Information Theory*, Vol. IT-18, No. 1, January 1972.

[Dav01] Dave, S.; Junghwan Kim; Kwatra, S.C., "An efficient decoding algorithm for Block Turbo Codes", *IEEE Transactions on Communications*, Vol. 49 1, pp. 41-46, January 2001.

[Eli54] Elias, P., "Error–free coding" *IRE Transactions on Information Theory*, PGIT–4, pp. 29–37, September 1954.

[Hag89] Hagenauer, J., and Hoeher, P., "A Viterbi Algorithm with soft–decision outputs and its applications", In *Proc. Globecom '89* , (Dallas, USA), pp. 1680–1686, 1989

[Hag96] Hagenauer, J., Offer, E., and Papke, L., "Iterative decoding of binary block and convolutional codes", *IEEE Transactions on Information Theory*, Vol. 42 2, pp. 429-445, March 1996.

[Hay01] Haykin, S., "*Communication Systems*", 4d edition: John Wiley and Sons, Inc. New York, 2001.

[Hon97] Honary, B. and Markarian, G., "*Trellis Decoding of Block Codes: A practical approach*": Kluwer Academic Publishers, 1997.

[Hun98a] Hunt, A., "*Hyper-codes: High-performance low-complexity error-correcting codes*", Master's Thesis, Carleton University, Ottawa, Canada, March 25, 1998.

[Hun98b] Hunt, A., Crozier, S. and Falconer, D., "Hyper-codes: High-performance low-complexity error-correcting codes", *Proceedings of the 19th Biennial Symposium on Communications*, Kingston, Canada, pp. 263-267, June 1998.

[Kan94] Kaneko, T.; Nishijima, T.; Inazumi, H.; Hirasawa, S., "An efficient maximum-likelihood-decoding algorithm for linear block codes with algebraic decoder", *IEEE Transactions on Information Theory*, Vol. 40 2, pp. 320-327, March 1994.

[Nic97a] Nickl, H., Hagenauer, J., and Burket, F., "Approaching Shannon`s capacity limit by 0.27 dB using Hamming codes in a `Turbo`-decoding scheme", In *Proceedings of the 1997 IEEE International Symposium on Information Theory*, Germany, June 29 – July 4, 1997.

[Nic97b] Nickl, H., Hagenauer, J., and Burkert, F., "Approaching Shannon's capacity limit by 0.2 dB using simple Hamming codes", *IEEE Communications Letters*, Vol. 1 5, pp. 130-132, September 1997.

[Pyn98] Pyndiah, R. M. "Near-optimum decoding of product codes: Block Turbo Codes", *IEEE Transactions on Communications*, Vol. 46 8, pp. 1003-1010, August 1998.

[Ran01] Rankin, D. and Gulliver, T. A., "Single Parity Check Product Codes", *IEEE Transactions on Communications*, Vol. 49, No. 8, pp. 1354-1362, August 2000.

[Rob95] Robertson, P.; Villebrun, E.; Hoeher, P., "A comparison of optimal and sub-optimal MAP decoding algorithms operating in the log

domain", In *1995 IEEE International Conference on Communications*, ICC '95, Seattle, Vol. 2, pp. 1009-1013, 1995.

[Sil54] Silverman, R. A., and M. Balser, "Coding for Constant-Data-Rate Systems", *IRE Transactions on Information Theory*, PGIT-4, pp. 50-63, 1954.

[Skl97] Sklar, B., "A primer on Turbo Code concepts", *IEEE Communication Magazine*, pp. 94-101, December 1997.

[Wol78] Wolf, J. K., "Efficient maximum likelihood decoding of linear block codes using a trellis", *IEEE Transactions on Information Theory*, Vol. IT-24, No. 1, pp. 76-80, January 1978.

SOBRE O AUTOR

Dayani Adionel Guimarães nasceu em Carrancas, MG, em março de 1969. Em 1987, 1994, 1996 e 1998 obteve os títulos de Técnico em Eletrônica, Engenheiro Eletricista, Especialista em Administração com ênfase em Gerência de RH e Mestre em Engenharia Elétrica, pela ETE "FMC", INATEL, FAI e UNICAMP, respectivamente. Em atividades profissionais anteriores desenvolveu sensores e equipamentos para instrumentação industrial e controle e também foi Supervisor de Produção e Supervisor de Engenharia de Produtos na SENSE Sensores e Instrumentos, de 1988 a 1993. Desde de janeiro de 1995 é Professor do INATEL, onde é responsável pela estrutura que dá suporte às atividades de ensino prático dos Departamentos de Telecomunicações e Eletrônica & Eletrotécnica. Encontra-se em fase de elaboração de tese de Doutorado na UNICAMP. Suas pesquisas incluem aspectos gerais das comunicações móveis, especificamente sistemas CDMA Multiportadoras e esquemas de codificação para canais com desvanecimento.

APÊNDICE

O apêndice mostra a implementação de um esquema de codificação de canal com decodificação Turbo. O código é do tipo SPC-PC (*Single Parity Check Product Code*) de dimensão *D* com concatenação paralela (sem paridade das paridades), seguindo uma idéia similar à de [Hun98a] para geração de paridades diagonais, e utilizando o algoritmo de decodificação Log-MAP sugerido em [Hag96], com algumas das generalizações citadas em [Ran01].

A partir desse algoritmo foram gerados alguns dos resultados mostrados neste artigo. Outros resultados foram obtidos através de modificações e/ou adaptações no algoritmo fornecido no sentido de levar em conta as considerações em cada caso. Estas modificações e/ou adaptações, contudo, não alteraram o processamento principal do algoritmo. Dentre elas podem ser citadas: a modificação para códigos sem paridade diagonal; a modificação para códigos 2D com paridade diagonal dupla; a adaptação para obtenção do histograma de distribuição de pesos do código e a adaptação para obtenção dos histogramas das LLRs.

Definição de algumas variáveis

Número de erros esperado em cada RUN (referência BPSK não codificada): $Nr_{erros} := 200$

Dimensão do código: $D := 5$

Número de bits de informação dos códigos componentes nas D dimensões: $m := 7$

Numero de pontos do gráfico: $N := 6$ $ponto := 0..N-$

Número de iterações desejado: $I := D$

Mínimo Eb/No, dB: $MinEbNo := 0$ Máximo Eb/No, dB: $MaxEbNo := 3.5$

$Eb := 1$

$$SNR_{ponto} := (ponto) \cdot \frac{MaxEbNo - MinEbNo}{(N-1)} + MinEbNo$$

Variância de ruído necessária a cada RUN: $$Var_{ponto} := \frac{Eb}{2 \cdot 10^{\frac{SNR_{ponto}}{10}}}$$

Eb/No em cada RUN (verificando): $$SNR_{ponto} := 10 \cdot \log\left(\frac{Eb}{2 \cdot Var_{ponto}}\right)$$

Block length: $BL := \left(m^D + m^{D-1} \cdot D\right) + m \cdot m^{D-2}$

Message length: $m^D = 1.681 \times 10^4$

Taxa do código: $$r := \frac{m^D}{\left(m^D + m^{D-1} \cdot D\right) + m \cdot m^{D-2}}$$

Correção da energia por símbolo em função da taxa do código: $Es := Eb \cdot r$

Number of parity blocks: $$\frac{m^{D-1} \cdot D}{m} + m^{D-2}$$

Para geração de bits com distribuição uniforme: $RND\_INT(x, y) := x + floor(rnd(y - x + 1))$

$$G := \left| \begin{array}{l} \text{for } y \in 0..m^2 - 1 \\ \quad C_y \leftarrow y \\ \text{for } x \in 0..m-1 \\ \quad SC^{\langle x \rangle} \leftarrow submatrix(C, x \cdot m, x \cdot m + m - 1, 0, 0) \\ \quad \text{for } u \in 0..m-1 \\ \quad\quad SS_u \leftarrow \left(SC^{\langle x \rangle}\right)_{mod(u+m-x,\, m)} \\ \quad\quad G^{\langle x \rangle} \leftarrow SS \\ G \end{array} \right.$$

$$Aux := \left| \begin{array}{l} \text{for } j \in 0..m^D - 1 \\ \quad Aux_j \leftarrow j \\ Aux \end{array} \right.$$

Algoritmo de Codificação & Decodificação

```
(BER S1 S2) := | for ponto ∈ 0.. N – 1
                |   C_blocos ← 0
                |   C_erros ← 0
                |   while C_erros < Nr_erros
                |     for j ∈ 0.. m^D – 1
                |       Dados_j ← RND_INT(0, 1)
                |     PD_{m–1, m^(D–2)} ← 0
                |     P_{m–1, D·m^(D–2)–1} ← 0
                |     for d ∈ 0.. (D) – 1
                |       for p ∈ 0.. (m^(D–2)) – 1
                |         for i ∈ 0.. m – 1
                |           [P^⟨p+(d·m^(D–2))⟩]_i ← mod[ Σ_{k=0}^{m–1} Dados_{mod[m^d·[(i·m+k)+p·m^2], m^D–1]} + [[(i·m+k)+p·m^2] = m^D–1]·(m^D–1), 2 ]
                |     B ← Dados
                |     for k ∈ 0.. m^(D–2)·D – 1
                |       B ← augment(B^T, P^⟨k⟩^T)^T
                |     for y ∈ 0.. m^(D–2) – 1
                |       Dpd ← submatrix[Dados, y·m^2, (y + 1)·m^2 – 1, 0, 0]
                |       for x ∈ 0.. m – 1
                |         S^⟨x⟩ ← submatrix(Dpd, x·m, x·m + m – 1, 0, 0)
                |         for u ∈ 0.. m – 1
                |           SS_u ← (S^⟨x⟩)_{mod(u+m–x, m)}
                |           SSS^⟨x⟩ ← SS
                |       for i ∈ 0.. m – 1
                |         (PD^⟨y⟩)_i ← mod( Σ_{j=0}^{m–1} SSS_{i, j}, 2 )
                |     for y ∈ 0.. m^(D–2) – 1
                |       B ← augment(B^T, PD^⟨y⟩^T)^T
                |     for j ∈ 0.. [(m^D + m^(D–1)·D) + m·m^(D–2)] – 1
                |       Lc_j ← (2·Es / Var_ponto)·[(B_j·–2 + 1)·√Es + √(–(2·ln(rnd(1)))·Var_ponto)·cos(2·π·rnd(1))]
                |     for d ∈ 0.. D – 1
                |       for j ∈ 0.. m^D – 1
                |         Le_{j, d} ← 0
                |         Ld_{j, d} ← 0
                |     for j ∈ 0.. m^D – 1
                |       LeD_j ← 0
```

Algoritmo de Codificação & Decodificação (continuação)

.

for $i \in 0..I-1$

for $d \in 0..D-1$

$$Ld \leftarrow \sum_{h=0}^{D-1} Le^{\langle h \rangle} \cdot \left(Le^{\langle h \rangle} \neq Le^{\langle d \rangle}\right) + LeD$$

for $j \in 0..m^D-1$

$$\Phi_j \leftarrow Aux_{mod\left[\left(m^d\right)\cdot j, m^D-1\right]+\left(m^D-1=j\right)\cdot\left(m^D-1\right)}$$

for $j \in 0..m^D-1$

$$W \leftarrow floor\left[\frac{mod\left[m^{(D-d)}\cdot j, m^D-1\right]+\left(m^D-1=j\right)\cdot\left(m^D-1\right)}{m}\right]$$

for $n \in 0..m-1$

$$L_n \leftarrow Lc_{\Phi_{(n+W\cdot m)}} \ldots + Ld_{\Phi_{(n+W\cdot m)}}$$

$$S \leftarrow \sum_{n=0}^{m-1} n\cdot\left[\Phi_{(n+W\cdot m)}=j\right]$$

$$L_S \leftarrow Lc_{W+m^D+d\cdot m^{(D-1)}}$$

$$Le_{j,d} \leftarrow 2\cdot atanh\left(\prod_{n=0}^{m-1} \tanh\left(\frac{L_n}{2}\right)\right)$$

for $j \in 0..m^D-1$

for $n \in 0..m-1$

$$LL_n \leftarrow Lc_{G\left[\left(match\left(mod\left(j,m^2\right),G\right)_{0,0}\right),n}+floor\left(\frac{j}{m^2}\right)\cdot m^2} + \left(Ld + Le^{\langle D-1 \rangle}\right)_{G\left[\left(match\left(mod\left(j,m^2\right),G\right)_{0,0}\right),n}+floor\left(\frac{j}{m^2}\right)\cdot m^2}$$

$$LL_{\left(match\left(mod\left(j,m^2\right),G\right)_{0,0}\right)_1} \leftarrow Lc_{\left(match\left(mod\left(j,m^2\right),G\right)_{0,0}\right)_0+\left(m^D+m^{D-1}\cdot D\right)+floor\left(\frac{j}{m^2}\right)\cdot m}$$

$$LeD_j \leftarrow 2\cdot atanh\left(\prod_{n=0}^{m-1} \tanh\left(\frac{LL_n}{2}\right)\right)$$

for $i \in 0..m^D-1$

$$Lt_i \leftarrow Lc_i + \left(\sum_{d=0}^{D-1} Le_{i,d}\right) + LeD_i$$

$$DadosRx_i \leftarrow \begin{cases} 0 & \text{if } Lt_i > 0 \\ 1 & \text{otherwise} \end{cases}$$

$$E \leftarrow \sum_{i=0}^{m^D-1} \left(Dados_i \oplus DadosRx_i\right)$$

$$C_{blocos} \leftarrow C_{blocos} + 1$$

$$C_{erros} \leftarrow C_{erros} + E$$

$$ber_{ponto} \leftarrow \frac{C_{erros}}{m^D \cdot C_{blocos}}$$

$$\left(ber \quad C_{erros} \quad C_{blocos}\right)$$