Do algoritmo BCJR à descodificação turbo

© Sílvio A. Abrantes[*]

Abril 2004

Em 1974 Bahl, Cocke, Jelinek e Raviv publicaram um algoritmo de descodificação de códigos baseado em probabilidades a posteriori. O algoritmo passou depois a ser conhecido como *algoritmo BCJR* (das iniciais dos autores), *algoritmo MAP* (de "maximum a posteriori") ou ainda *"forward-backward algorithm"*. O procedimento pode ser aplicado a códigos de blocos ou a códigos convolucionais mas, como é bastante mais complexo que o algoritmo de Viterbi, durante cerca de vinte anos não foi usado na prática, situação que foi substancialmente alterada com o advento dos turbo-códigos em 1993. Nessa altura os seus inventores, Berrou, Glavieux e Thithimajshima, usaram uma versão modificada do algoritmo BCJR, que dessa forma renasceu com grande vigor.

Existem diversas versões simplificativas do algoritmo MAP, nomeadamente as designadas como log-MAP e max-log-MAP. O propósito deste texto é mostrar, sem cálculos intermédios, como é que todos estes algoritmos funcionam e são aplicados na descodificação turbo. Os pormenores teóricos poderão ser consultados no Apêndice.

1 Introdução

Considere-se um codificador de blocos ou convolucional de taxa *k/n* descrito por uma treliça, que apresente na sua saída uma sequência de *N* palavras de código, ou símbolos, de *n* bits cada. Designemos essa sequência por $\mathbf{x} = x_1 x_2 \ldots x_N$, onde x_k representa o símbolo produzido pelo codificador no instante *k*. O bit de informação que nesse instante lhe deu origem, u_k, pode tomar os valores[1] -1 ou +1 com uma probabilidade de ocorrência *a priori* $P(u_k)$. Associada a esta probabilidade define-se a quantidade

$$L(u_k) = \ln \frac{P(u_k = +1)}{P(u_k = -1)}, \tag{1}$$

a que se dá o nome, em inglês, de LLR (de "log-likelihood ratio"[2]). Se os bits $u_k = \pm 1$ forem equiprováveis esta LLR a priori é nula.

A sequência codificada **x** atravessa um canal de ruído gaussiano sem memória (canal AWGN) que a transforma numa sequência de símbolos reais $\mathbf{y} = y_1 y_2 \ldots y_N$, como se mostra na Fig. 1. É esta sequência que o descodificador recebe e a partir dela o algoritmo BCJR, ou qualquer outro, vai tentar estimar a sequência de bits originais u_k. Para isso o algoritmo calcula a LLR *a posteriori* $L(u_k|\mathbf{y})$, um valor real definido pela razão

[*] Faculdade de Engenharia da Universidade do Porto (FEUP), Porto, Portugal.

[1] Substituiremos os habituais valores lógicos 0 e 1 pelos valores reais simétricos -1 e +1, respectivamente, pois vai ser usada a modulação BPSK.

[2] Uma tradução possível é "log-razão de verosimilhança", apesar de ser tentador usar "razão de log-verosimilhança". Esta última hipótese é incorrecta pois o logaritmo é aplicado à razão e não o contrário.

$$L(u_k|\mathbf{y}) = \ln \frac{P(u_k = +1|\mathbf{y})}{P(u_k = -1|\mathbf{y})}. \quad (2)$$

O sinal, positivo ou negativo, da LLR, indicia que o bit enviado foi +1 ou -1, respectivamente, e o seu valor absoluto traduz a maior ou menor confiança, ou fiabilidade, que temos nessa presunção, isto é, quanto mais afastado o valor de $L(u_k|\mathbf{y})$ estiver do limiar de decisão nulo mais confiança teremos na estimativa do bit. Esta informação branda[3] contida em $L(u_k|\mathbf{y})$ pode ser passada a outro elemento de descodificação, se o houver, ou convertida em decisões rígidas: como se disse, se $L(u_k|\mathbf{y}) < 0$ o descodificador estimará que foi enviado o bit u_k = -1, caso contrário estimará que foi enviado o bit +1.

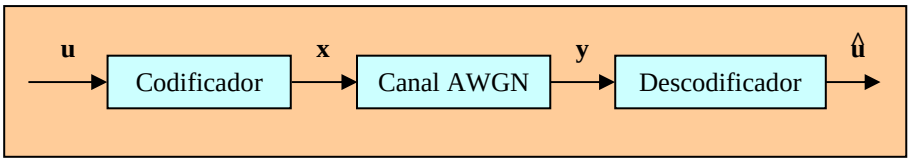


Fig. 1 Diagrama de blocos simplificado do sistema

No numerador e no denominador da Eq. (2) temos probabilidades condicionais a posteriori, ou seja, probabilidades calculadas após o conhecimento de **y**. Por outras palavras, temos as probabilidades de no instante *k* o bit de entrada do codificador ter sido u_k = +1 ou u_k = -1, respectivamente, dado que conhecemos toda a sequência recebida. São estas as probabilidades condicionais que o algoritmo BCJR original [1] de facto calcula e compara. Porém, é completamente equivalente usar a formulação logarítmica da Eq. (2).

É conveniente trabalhar com treliças. Admitamos que temos um codificador convolucional de taxa 1/2, com n = 2, e quatro estados S = {0, 1, 2, 3}, e uma treliça entre dois instantes de tempo consecutivos como a apresentada na Fig. 2. Por convenção arbitramos que um bit de mensagem -1 na entrada do codificador produz um ramo de traço contínuo e um bit +1 produz um ramo tracejado. Cada ramo está rotulado com a correspondente saída de dois bits x_k, em que 0 e 1 correspondem a -1 e +1, respectivamente.

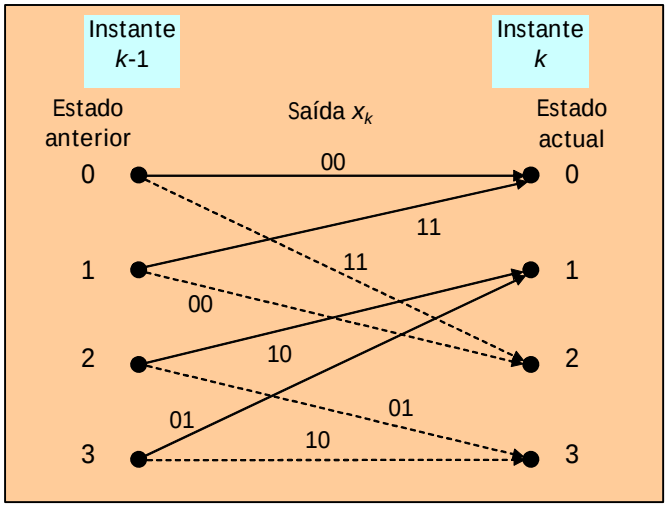


Fig. 2 A treliça do codificador convolucional usado

[3] Na literatura de língua inglesa um valor brando é designado como “soft” e um valor rígido (0 ou 1) é designado como “hard”.

Suponhamos que estamos no instante *k*. O estado correspondente é $S_k = s$, o estado anterior é $S_{k-1} = s'$ e o símbolo recebido no descodificador é y_k. Antes deste instante já tinham sido recebidos *k*-1 símbolos e depois irão ser recebidos *N*-*k* símbolos. Ou seja, a sequência completa **y** pode ser dividida em três subsequências, uma representando o passado, outra o presente e outra o futuro:

$$\mathbf{y} = \underbrace{y_1 y_2 \cdots y_{k-1}}_{\mathbf{y}_{<k}} y_k \underbrace{y_{k+1} \cdots y_N}_{\mathbf{y}_{>k}} = \mathbf{y}_{<k}\mathbf{y}_k\mathbf{y}_{>k} \tag{3}$$

No Apêndice mostra-se que a LLR a posteriori $L(u_k|\mathbf{y})$ é dada pela expressão

$$L(u_k|\mathbf{y}) = \ln \frac{\sum_{R_1} P(s', s, \mathbf{y})}{\sum_{R_0} P(s', s, \mathbf{y})} = \ln \frac{\sum_{R_1} \alpha_{k-1}(s')\gamma_k(s', s)\beta_k(s)}{\sum_{R_0} \alpha_{k-1}(s')\gamma_k(s', s)\beta_k(s)} \tag{4}$$

$P(s', s, \mathbf{y})$ representa a probabilidade conjunta de no instante *k*-1 estarmos no estado s', no instante corrente, *k*, estarmos no estado *s* e de a sequência de *N* bits recebida ser **y**. No numerador R_1 significa que o somatório se estende às transições entre estados s' e *s* provocadas por um bit de mensagem u_k = +1 (ou seja, ramos tracejados). Da mesma maneira, no denominador R_0 designa os outros ramos, os que são originados por um bit de mensagem u_k = -1. As variáveis α, γ e β são probabilidades definidas a seguir.

2 Cálculo da probabilidade conjunta $P(s', s, \mathbf{y})$

Esta probabilidade pode ser calculada como o produto de três probabilidades,

$$P(s', s, \mathbf{y}) = \alpha_{k-1}(s')\gamma_k(s', s)\beta_k(s), \tag{5}$$

definidas da seguinte maneira:

$$\alpha_{k-1}(s') = P(s', \mathbf{y}_{<k}) \tag{6}$$

$$\gamma_k(s', s) = P(\mathbf{y}_k, s|s') \tag{7}$$

$$\beta_k(s) = P(\mathbf{y}_{>k}|s) \tag{8}$$

No instante *k* as probabilidades α, γ e β estão associadas ao passado, ao presente e ao futuro da sequência **y**, respectivamente. Vejamos como as calcular começando por γ.

2.1 Cálculo de γ

A probabilidade $\gamma_k(s', s) = P(\mathbf{y}_k, s|s')$ é a probabilidade condicional de, no instante *k*, o símbolo recebido ser $\mathbf{y}_k$ e o estado actual ser $S_k = s$, sabendo-se que o estado de onde se proveio foi $S_{k-1} = s'$. É dada pelo produto de probabilidades

$$\gamma_k(s', s) = P(\mathbf{y}_k|x_k)P(u_k).$$

No caso especial de canais de ruído branco gaussiano (AWGN) $\gamma_k(s', s)$ é dada por

$$\gamma_k(s',s) = C_k e^{u_k L(u_k)/2} \exp\left(\frac{L_c}{2}\sum_{l=1}^{n} x_{kl} y_{kl}\right), \tag{9}$$

onde C_k representa uma quantidade que, surgindo no numerador e no denominador da Eq. (4), vai desaparecer ao calcularmos a LLR condicional $L(u_k|\mathbf{y})$. Quer dizer que o valor de C_k é irrelevante para o que desejamos. Na segunda exponencial temos a quantidade L_c e um somatório. Denominado *valor*, ou *medida, de fiabilidade do canal*, L_c é igual a

$$L_c = 4a\frac{E_c}{N_0} = 4aR_c\frac{E_b}{N_0}$$

$N_0/2$ é a densidade espectral de potência bilateral do ruído, E_c e E_b são as energias transmitidas por bit codificado e bit de mensagem, respectivamente, R_c é a taxa do código e a é a amplitude de *fading*. Se no canal não existir *fading* então $a = 1$.

2.2 Cálculo recursivo de α e β

As probabilidades α e β podem (e devem) ser calculadas recursivamente. As respectivas fórmulas recursivas são as seguintes:

$$\alpha_k(s) = \sum_{s'} \gamma_k(s',s)\alpha_{k-1}(s') \qquad \text{Condições iniciais: } \alpha_0(s) = \begin{cases} 1 & s = 0 \\ 0 & s \neq 0 \end{cases} \tag{10}$$

$$\beta_{k-1}(s') = \sum_{s} \gamma_k(s',s)\beta_k(s) \qquad \text{Condições iniciais: } \beta_N(s) = \begin{cases} 1 & s = 0 \\ 0 & s \neq 0 \end{cases} \tag{11}$$

Repare-se no seguinte:

- Em ambos os casos necessitamos da mesma quantidade $\gamma_k(s',s)$, que terá de ser calculada primeiro.
- No caso de $\alpha_k(s)$ os somatórios são efectuados para todos os estados anteriores $S_{k-1} = s'$ dos quais saem ramos que convergem no estado s, enquanto que no caso de $\beta_{k-1}(s')$ os somatórios são efectuados para todos os estados seguintes $S_k = s$ que se atingem do estado s'. Com códigos binários os somatórios têm duas parcelas apenas.
- A probabilidade α vai sendo calculada à medida que se for recebendo a sequência **y**. Ou seja, no cálculo de α vamos indo do início para o fim da treliça (sentido "forward").
- A probabilidade β só pode ser calculada depois de termos recebido toda a sequência **y**. Ou seja, no cálculo de β vamos regressando do fim para o princípio (sentido "backward").
- Veremos que na treliça os valores de α e β estão associados aos estados do codificador e os valores de γ estão associados aos ramos ou transições entre estados.
- Os valores iniciais $\alpha_0(s)$ e $\beta_N(s)$ pressupõem que o percurso na treliça começa e termina no estado nulo (diz-se que se trata de uma treliça *terminada*). Para isso será preciso acrescentar ao fim da mensagem alguns bits de cauda que obriguem o percurso a regressar ao estado nulo inicial.

A razão do nome pelo qual o algoritmo BCJR também é conhecido, "*forward-backward algorithm*", torna-se agora evidente.

3 Combatendo a instabilidade numérica

São bem conhecidos os problemas de instabilidade numérica associados ao algoritmo BCJR. De facto, a natureza iterativa de alguns cálculos pode conduzir a indesejáveis situações de "underflow" ou "overflow" que devem ser evitadas. Nesse sentido, em vez de na Eq. (5) usar α e β calculadas directamente das equações recursivas (10) e (11) essas probabilidades devem ser previamente normalizadas pela soma de todos os α e β em cada instante, respectivamente. O mesmo se aplica à probabilidade conjunta, $P(s', s, \mathbf{y})$, como segue. Em cada instante de tempo k vamos definir as variáveis auxiliares não normalizadas α' e β':

$$\alpha'_k(s) = \sum_{s'} \alpha_{k-1}(s')\gamma_k(s', s)$$

$$\beta'_{k-1}(s') = \sum_{s} \beta_k(s)\gamma_k(s', s)$$

Depois de todos os M valores de α' e β' terem sido calculados somamo-los: $\sum_s \alpha'_k(s)$ e $\sum_{s'} \beta'_{k-1}(s')$. Depois normalizamos α e β dividindo-os por essas somas:

$$\alpha_k(s) = \frac{\alpha'_k(s)}{\sum_s \alpha'_k(s)} \tag{12}$$

$$\beta_{k-1}(s') = \frac{\beta'_{k-1}(s')}{\sum_{s'} \beta'_{k-1}(s')} \tag{13}$$

Do mesmo modo, depois de todos os $2M$ produtos $\alpha_{k-1}(s')\gamma_k(s', s)\beta_k(s)$ em todos os ramos da treliça terem sido calculados no instante k a sua soma,

$$\begin{aligned}\Sigma_{Pk} &= \sum_{R_o, R_1} \alpha_{k-1}(s')\gamma_k(s', s)\beta_k(s) = \\ &= \sum_{R_o} \alpha_{k-1}(s')\gamma_k(s', s)\beta_k(s) + \sum_{R_1} \alpha_{k-1}(s')\gamma_k(s', s)\beta_k(s)\end{aligned}$$

irá normalizar $P(s', s, \mathbf{y})$:

$$P_{norm}(s', s, \mathbf{y}) = \frac{P(s', s, \mathbf{y})}{\Sigma_{Pk}}$$

Assim garantimos que as somas de todos os α, β e $P_{norm}(s', s'\mathbf{y})$ são sempre iguais a 1 em cada instante k. Nenhuma destas somas de normalização afecta o LLR $L(u_k|\mathbf{y})$ final visto que todas aparecem no numerador e denominador:

$$L(u_k|\mathbf{y}) = \ln \frac{\sum_{R_1} P(s', s, \mathbf{y})}{\sum_{R_0} P(s', s, \mathbf{y})} = \ln \frac{\sum_{R_1} P_{norm}(s', s, \mathbf{y})}{\sum_{R_0} P_{norm}(s', s, \mathbf{y})} \tag{14}$$

4 Cálculo de α e β usando a treliça

A figura seguinte apresenta a treliça da Fig. 2 mas agora com outros rótulos. Recorda-se que um ramo tracejado é devido a um bit de entrada +1 e um ramo contínuo é devido a um bit de entrada -1.

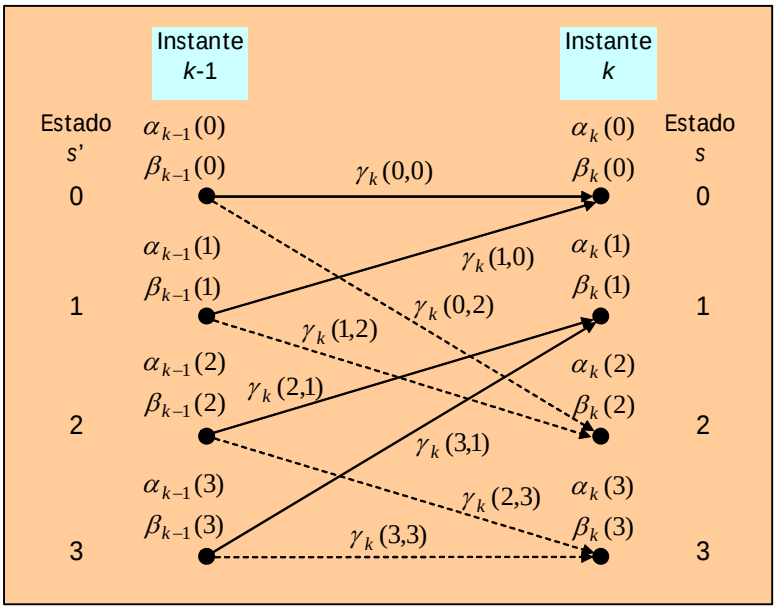


Fig. 3 Treliça com os valores de α, β e γ.

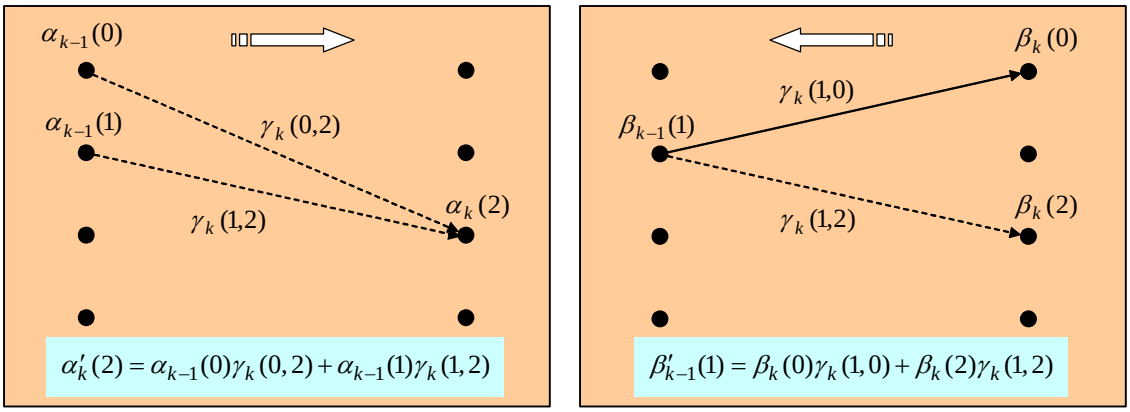
À medida que os cálculos se forem efectuando façamos o seguinte:

- em cada ramo colocamos um rótulo com o valor de $\gamma_k(s',s)$ calculado de acordo com a Eq. (9).
- em cada nó de estado colocamos o valor de $\alpha_k(s)$ calculado de acordo com a Eq. (10) ou (12) a partir das condições iniciais $\alpha_0(s)$.
- em cada nó de estado e por baixo do valor de $\alpha_k(s)$ colocamos o valor de $\beta_k(s)$ calculado de acordo com a Eq. (11) ou (13) a partir das condições iniciais $\beta_N(s)$.

4.1 Cálculo de α

Suponhamos então que conhecemos $\alpha_{k-1}(s')$. A probabilidade $\alpha'_k(s)$ (sem normalização) é obtida somando os produtos de $\alpha_{k-1}(s')$ e $\gamma_k(s',s)$ dos ramos que convergirem em *s*. Por exemplo, e olhando para a treliça da Fig. 3, vemos que no instante *k* chegam dois ramos ao estado $S_k = 2$, um que vem do estado 0 e outro que vem do estado 1, como se salienta na Fig. 4a. Depois de todos os *M* valores de $\alpha'_k(s)$ terem sido calculados devem ser divididos pela sua soma para obtermos $\alpha_k(s)$.

O procedimento repete-se até chegarmos ao fim da sequência recebida e termos calculado $\alpha_N(0)$ (recorda-se que a treliça termina no estado nulo).



a) b)

Fig. 4 Cálculo recursivo de α e β através da treliça.

4.2 Cálculo de β

A quantidade β só pode ser calculada recursivamente depois de se ter recebido toda a sequência **y**. Conhecendo $\beta_k(s)$ o valor de $\beta_{k-1}(s')$ é calculado de maneira semelhante à de $\alpha_k(s)$: vê-se que ramos saem do estado $S_{k-1} = s'$, somam-se os correspondentes produtos de $\gamma_k(s)$ por $\beta_k(s)$ e dividem-se pela soma $\sum_{s'} \beta'_{k-1}(s')$. Por exemplo, vemos na Fig. 3 que do estado $S_{k-1} = 1$ saem dois ramos, um dirigido ao estado $S_k = 0$ e outro dirigido ao estado $S_k = 2$, como se realça na Fig. 4b. O processo repete-se até ao cálculo de $\beta_0(0)$.

4.3 Cálculo de $P(s', s, \mathbf{y})$ e $L(u_k|\mathbf{y})$

Tendo todos os valores de α, β e γ estamos prontos para calcular a probabilidade conjunta $P_{norm}(s', s, \mathbf{y}) = \alpha_{k-1}\gamma_k(s', s)\beta_k(s)/\Sigma_{Pk}$. Por exemplo, quanto vale *P*(1,2,**y**) não normalizado? Como se mostra na Fig. 5, vale $\alpha_{k-1}(1)\gamma_k(1,2)\beta_k(2)$.

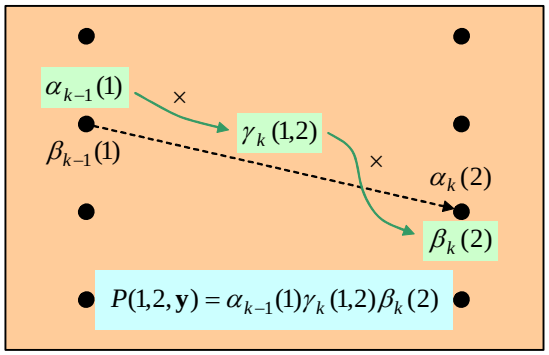


Fig. 5 A probabilidade *P*(*s*',*s*,**y**) como produto de três factores $\alpha\gamma\beta$

Só nos falta calcular a LLR a posteriori $L(u_k|\mathbf{y})$. Ora observemos a treliça da Fig. 3 de novo: vemos que um bit de mensagem +1 origina as transições de estados seguintes: 0→2, 1→2, 2→3 e 3→3. São estas as transições R_1 da Eq. (4). As restantes quatro transições de estados, representados por traço

contínuo (conjunto R_0), são devidas, é claro, a um bit de mensagem -1. Logo, as primeiras quatro transições estão associadas ao numerador da Eq. (4) e as restantes ao denominador:

$$L(u_k|\mathbf{y}) = \ln \frac{\sum_{R_1} P(s', s, \mathbf{y})}{\sum_{R_0} P(s', s, \mathbf{y})} = \\ = \ln \frac{P(0,2,\mathbf{y}) + P(1,2,\mathbf{y}) + P(2,3,\mathbf{y}) + P(3,3,\mathbf{y})}{P(0,0,\mathbf{y}) + P(1,0,\mathbf{y}) + P(2,1,\mathbf{y}) + P(3,1,\mathbf{y})}$$

A Fig. 6 apresenta um resumo com as expressões não normalizadas necessárias à obtenção da LLR condicional.

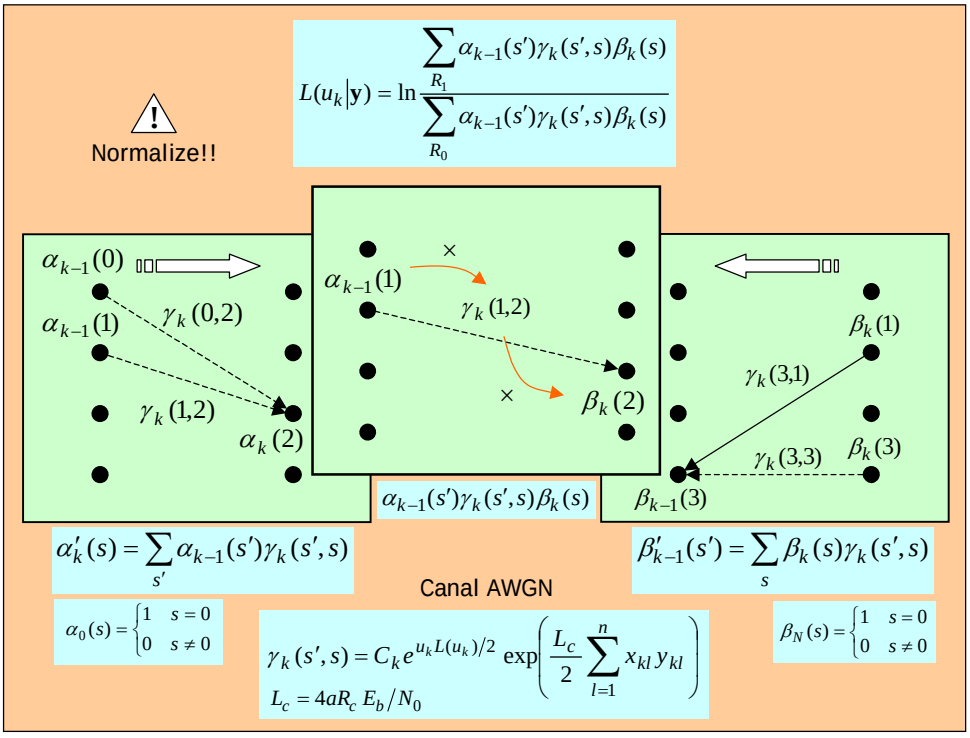


Fig. 6 Expressões das variáveis do algoritmo BCJR

Um exemplo numérico mais adiante ajudará a compreender todo o algoritmo MAP.

5 Modificações simplificativas do algoritmo MAP

O algoritmo BCJR, ou MAP, apresenta uma desvantagem importante: necessita de efectuar muitas multiplicações. No intuito de reduzir a complexidade dos cálculos surgiram algoritmos mais simples, nomeadamente os algoritmos SOVA ("Soft-Output Viterbi Algorithm"), em 1989 [2], max-log-MAP, em 1990-1994 [3] [4], e log-MAP, em 1995 [5]. Neste texto só iremos abordar os dois últimos, nos quais as multiplicações do algoritmo MAP são substituídas por adições.

São definidas três novas variáveis, A, B e Γ:

$$\Gamma_k(s',s) = \ln \gamma_k(s',s) =$$
$$= \ln C_k + \frac{u_k L(u_k)}{2} + \frac{L_c}{2}\sum_{l=1}^{n} x_{kl} y_{kl}$$

$$A_k(s) = \ln \alpha_k(s) = \qquad A_0(s) = \begin{cases} 0 & s = 0 \\ -\infty & s \neq 0 \end{cases}$$
$$= \max_{s'}{}^{*}\left[A_{k-1}(s') + \Gamma_k(s',s)\right]$$

$$B_{k-1}(s') = \ln \beta_{k-1}(s') = \qquad B_N(s) = \begin{cases} 0 & s = 0 \\ -\infty & s \neq 0 \end{cases}$$
$$= \max_{s}{}^{*}\left[B_k(s) + \Gamma_k(s',s)\right]$$

em que

$$\max{}^{*}(a,b) = \begin{cases} \max(a,b) + \ln(1+e^{-|a-b|}) & \text{algoritmo log} - \text{MAP} \\ \max(a,b) & \text{algoritmo max} - \log - \text{MAP} \end{cases}$$

bastando guardar os valores da função $\ln(1+e^{-|a-b|})$ numa tabela com apenas oito valores de $|a-b|$ entre 0 e 5.

A parcela $\ln C_k$ na expressão de $\Gamma_k(s',s)$ não vai ser usada no cálculo da LLR. Esta é igual a

$$L(u_k|\mathbf{y}) = \max_{R_1}{}^{*}\left[A_{k-1}(s') + \Gamma_k(s',s) + B_k(s)\right] - \max_{R_0}{}^{*}\left[A_{k-1}(s') + \Gamma_k(s',s) + B_k(s)\right] \qquad (15)$$

A Fig. 7 apresenta o conjunto de expressões necessárias para calcular a LLR da Eq. (15).

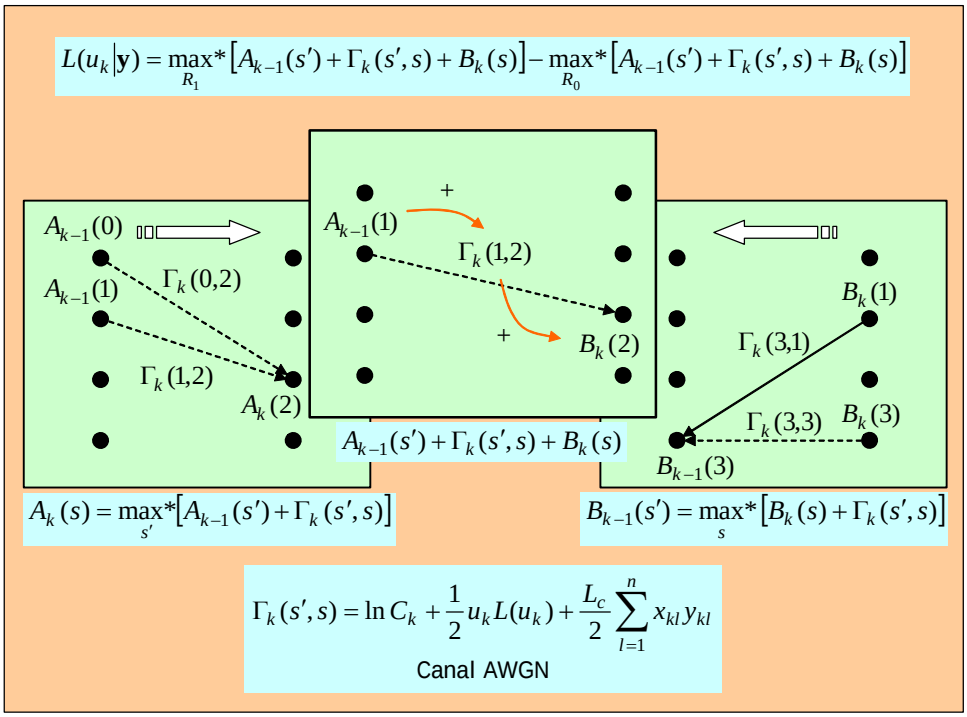


Fig. 7 Sumário de expressões usadas nos algoritmos MAP simplificados

A Eq. (15) não é de cálculo imediato no algoritmo log-MAP pois a função max* nessa equação tem mais de duas variáveis. No Apêndice pode ver-se como a calcular de forma recursiva.

Como usa fórmulas exactas o algoritmo log-MAP tem um desempenho igual ao do algoritmo BCJR mas é mais simples pelo que lhe é preferido na prática. O algoritmo max-log-MAP, por sua vez, usa aproximações e por isso o seu desempenho é ligeiramente inferior.

6 Aplicação do algoritmo BCJR à descodificação iterativa

Consideremos um codificador convolucional sistemático de taxa $1/n$ no qual o primeiro bit codificado, x_{k1}, é igual ao bit de informação u_k. Nesse caso a razão LLR a posteriori $L(u_k|\mathbf{y})$ pode ser decomposta numa soma de três parcelas (os pormenores estão no Apêndice):

$$L(u_k|\mathbf{y}) = L(u_k) + L_c y_{k1} + L_e(u_k)\,. \tag{16}$$

As primeiras duas parcelas estão relacionadas com o bit de informação u_k. Pelo contrário a terceira, $L_e(u_k)$, só depende dos bits de paridade da palavra de código. É por isso que a $L_e(u_k)$ se dá o nome de informação *extrínseca*. Esta informação extrínseca é uma estimativa da LLR a priori $L(u_k)$. Mas como? É fácil: fornecemos $L(u_k)$ e $L_c y_{k1}$ a um descodificador MAP (ou outro), que nos apresenta $L(u_k|\mathbf{y})$ na saída. Depois, por subtracção, obtemos a estimativa de $L(u_k)$:

$$L_e(u_k) = L(u_k|\mathbf{y}) - L(u_k) - L_c y_{k1} \tag{17}$$

Esta estimativa de $L(u_k)$ é, presumivelmente, um valor mais preciso da LLR a priori, que desconhecemos, pelo que deve substituir o anterior valor de $L(u_k)$. Se repetirmos o procedimento anterior de um modo iterativo fornecendo a um outro descodificador $L_c y_{k1}$ (de novo) e a nova $L(u_k) = L_e(u_k)$ esperamos obter uma $L(u_k|\mathbf{y})$ mais rigorosa na sua saída. Este facto é explorado no nosso próximo assunto, a descodificação turbo.

Os inventores dos turbo-códigos [6] usaram dois códigos convolucionais recursivos e sistemáticos[4], de taxa 1/2, concatenados em paralelo[5] e entrelaçados, e realizaram a descodificação iterativamente com dois descodificadores MAP (ver as Figs. 8 e 9, onde P e P^{-1} representam o entrelaçador e o desentrelaçador, respectivamente. P é também um vector-linha com o padrão de entrelaçamento). Ambos os codificadores da Fig. 8 são iguais e os seus bits de paridade $x_{kp}^{(1)}$ e $x_{kp}^{(2)}$, na saída, são frequentemente recolhidos alternadamente, por perfuração, de modo a obter-se uma taxa de código global de 1/2. O elemento de ordem i da sequência entrelaçada, $u_i^{(P)}$, é simplesmente igual ao elemento de ordem P_i da sequência original, $u_i^{(P)} = u_{P_i}$.

A Eq. (16) é a base da descodificação iterativa. Na primeira iteração a LLR a priori $L(u_k)$ é nula se considerarmos que os bits de entrada são equiprováveis. A informação extrínseca $L_e(u_k)$ que cada descodificador fornece será usada para actualizar $L(u_k)$ de iteração para iteração e desse descodificador para o outro. Desta maneira o descodificador turbo ganha progressivamente mais confiança nas decisões rígidas ± 1 que o decisor forçosamente terá de tomar no fim do processo iterativo. A Fig. 9 mostra um diagrama de blocos simplificado de um descodificador turbo que ilustra os procedimentos iterativos. Ora vimos já como

[4] Designados em inglês pela sigla RSC ("recursive systematic convolutional").
[5] A concatenação em série também é usada, embora não tão frequentemente.

obter uma sequência entrelaçada $\mathbf{u}^{(P)}$ a partir de $\mathbf{u}$ (calculamos $\mathbf{u}_i^{(P)} = \mathbf{u}_{P_i}$). Para desentrelaçar a sequência arbitrária $\mathbf{w}$ calculamos simplesmente $\mathbf{w}_{P_i}^{(P^{-1})} = \mathbf{w}_i$ (veja-se o exemplo numérico na Secção 7.4).

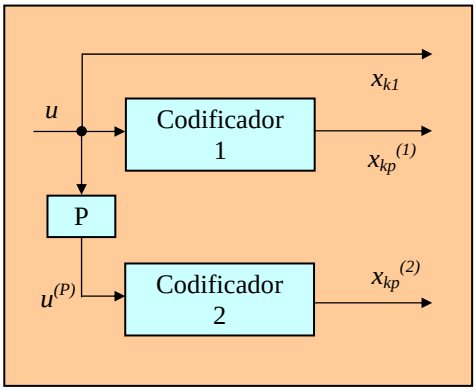


Fig. 8 O codificador turbo

A descodificação iterativa prossegue assim:

- na primeira iteração assumimos que $L(u_k) = 0$. O descodificador 1 fornece então a informação extrínseca $L_{e1}(u_k|\mathbf{y})$ sobre o bit sistemático, ou de mensagem, que obteve do primeiro bit de paridade. (note-se que, de facto, o descodificador 2 não precisa da LLR $L_1(u_k|\mathbf{y})$!);
- Após entrelaçamento apropriado a informação extrínseca $L_{e1}(u_k|\mathbf{y})$ do descodificador 1, calculada através da Eq. (17), é entregue ao descodificador 2 como $L_1(u_k)$, que é um "palpite" melhor e mais actual sobre $L(u_k)$. Depois, o descodificador 2 fornece $L_{e2}(u_k|\mathbf{y})$, que é a sua própria informação extrínseca sobre o bit sistemático, mas agora baseada no outro bit de paridade (note-se de novo que continuamos a desprezar a LLR!). Após desentrelaçamento conveniente esta informação é entregue ao descodificador 1 como $L_2(u_k)$, a qual é um palpite ainda mais bem elaborado sobre $L(u_k)$. Começará então uma nova iteração.
- Após um número pré-determinado de iterações ou após se ter atingido um determinado critério de paragem a LLR $L_2(u_k|\mathbf{y})$ na saída do descodificador 2 é desentrelaçada e entregue como $L(u_k|\mathbf{y})$ ao dispositivo de decisão rígida o qual, por sua vez, estima o bit de informação de acordo exclusivamente com o sinal, positivo ou negativo, da LLR desentrelaçada,

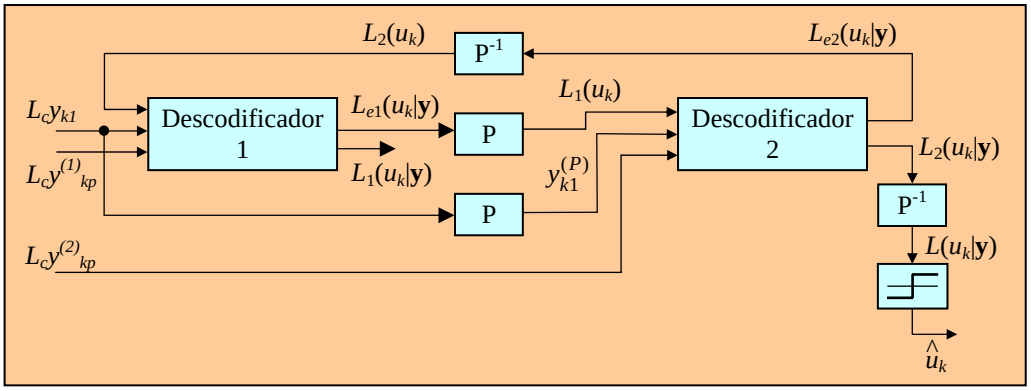


Fig. 9 Um diagrama de blocos simplificado de um descodificador turbo

$$\hat{u}_k = sign\left[L(u_k\,|\mathbf{y})\right] = sign\left\{P^{-1}\left[L_2(u_k\,|\mathbf{y})\right]\right\}.$$

Em vez do algoritmo BCJR também se poderá usar qualquer dos seus algoritmos simplificativos.

7 Exemplos numéricos

Nos próximos exemplos do algoritmo BCJR e suas simplificações vamos usar o mesmo codificador convolucional que usámos até agora. Suponhamos então que uma sequência **x** de seis símbolos codificados atravessa um canal AWGN sendo recebida a sequência de doze números reais

$$\mathbf{y} = \overbrace{0.3 \quad 0.1}^{y_1} \quad \overbrace{-0.5 \quad 0.2}^{y_2} \quad \overbrace{0.8 \quad 0.5}^{y_3} \quad \overbrace{-0.5 \quad 0.3}^{y_4} \quad \overbrace{0.1 \quad -0.7}^{y_5} \quad \overbrace{1.5 \quad -0.4}^{y_6}$$

Os bits de entrada do codificador, $u_k = \pm 1$, são equiprováveis e o percurso da sequência codificada **x** começa e termina no estado nulo da treliça (para o que é necessário acrescentar dois bits -1 à cauda da mensagem). O canal AWGN é tal que $\frac{E_c}{N_0} = 1 \;\; dB$ e $a = 1$. Devido à cauda da mensagem já sabemos quais são os valores dos dois últimos bits a estimar. Qual é a estimativa MAP, log-MAP e max-log-MAP dos outros quatro?

7.1 Exemplo com o algoritmo BCJR, ou MAP

A medida de fiabilidade do canal vale $L_c = 4a\frac{E_c}{N_0} = 4\times 1\times 10^{0,1} = 5.0$. Como $P(u_k = \pm 1) = 1/2$ então $L(u_k) = 0$ pelo que $e^{u_k L(u_k)/2} = 1$ (ver Eq. (9)), independentemente do sinal de u_k. Na mesma Eq. (9) vamos considerar $C_k = 1$ (porque não? qualquer valor serve… mas, atenção! assim as "probabilidades" que vamos calcular poderão ser superiores a um! Como o que queremos é calcular a razão de probabilidades esse facto não vai ter importância no resultado final[6]).

No instante $k = 1$ temos a situação retratada na Fig. 10a. Os valores de γ são, assim,

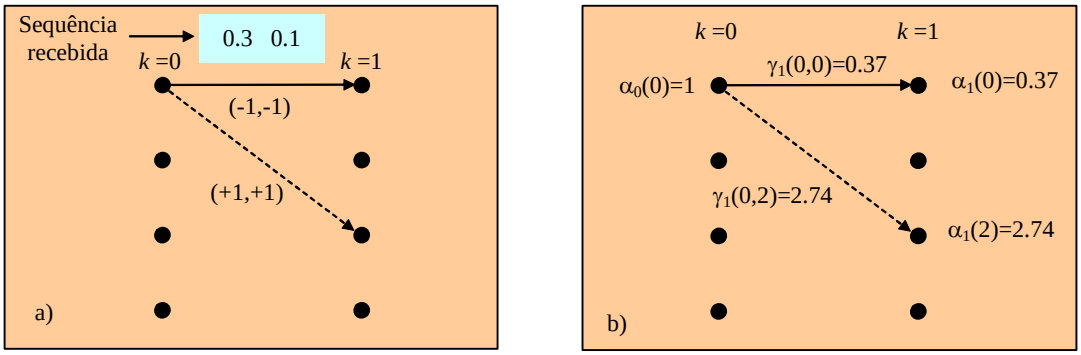


Fig. 10 Valores recebidos e enviados em $k = 1$ e probabilidades α e γ correspondentes.

$$\gamma_1(0,0) = C_k e^{u_k L(u_k)/2} \exp\left[\frac{L_c}{2}\left(x_{11}y_{11} + x_{12}y_{12}\right)\right] = e^{\left[2.5\times(-1\times 0.3 - 1\times 0.1)\right]} = e^{-1} = 0.37$$

[6] Contudo, se puderem surgir problemas de "underflow" ou de "overflow" será preferível usar os valores verdadeiros de C_k.

$$\gamma_1(0,2) = e^{\left[2.5\times(1\times0.3+1\times0.1)\right]} = e = 2.74$$

e, portanto (ver Fig. 10b),

$$\alpha_0(0)\gamma_1(0,0) = 1\times0.37 = 0.37 \quad \Rightarrow \quad \alpha_1(0) = \frac{0.37}{0.37+2.74} = 0.12$$

$$\alpha_0(0)\gamma_1(0,2) = 1\times2.74 = 2.74 \quad \Rightarrow \quad \alpha_1(2) = \frac{2.74}{0.37+2.74} = 0.88$$

Os cálculos de α e γ vão prosseguindo de maneira idêntica à medida que a sequência for sendo recebida[7]. Quando tiver chegado ao fim far-se-ão os cálculos de β em direcção ao início da treliça. A situação final com todos os valores de α, β e γ é a retratada na Fig. 11. Vejamos apenas mais um exemplo de cálculo de α, β e γ.

$$\gamma_3(2,3) = e^{\left[2.5\times(-1\times(0.8)+1\times0.5)\right]} = e^{-0.75} = 0.47$$

$$\alpha_3(3) = \frac{\alpha_2(2)\gamma_3(2,3)+\alpha_2(3)\gamma_3(3,3)}{4.31} = \frac{0.01\times0.47+0.92\times2.13}{4.31} = 0.45$$

$$\beta_2(2) = \frac{\beta_3(1)\gamma_3(2,1)+\beta_3(3)\gamma_3(2,3)}{9.96} = \frac{0.69\times2.13+0.001\times0.47}{9.96} = 0.15$$

Os valores dos denominadores provêm das somas

$$\sum_s \alpha'_3(s) = \alpha'_3(0)+\alpha'_3(1)+\alpha'_3(2)+\alpha'_3(3) = 0.716+0.452+1.182+1.959 = 4.31$$

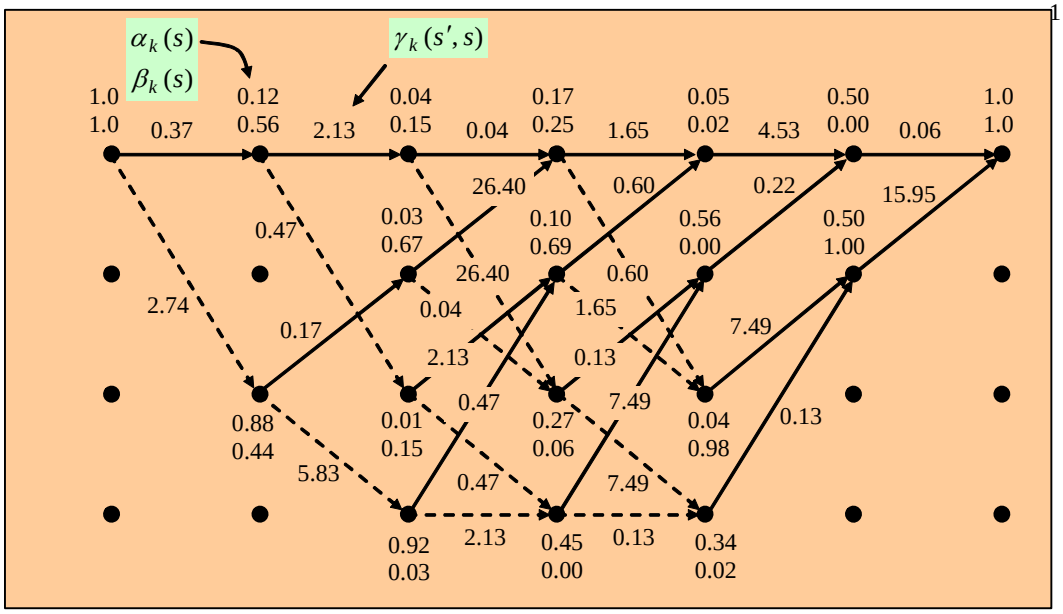


Fig. 11 Valores de α, β e γ ao fim de toda a sequência de seis símbolos recebida.

[7] Há,no máximo, $2^n = 4$ valores diferentes de γ em cada instante k pois há, no máximo, $2^n = 4$ palavras de código diferentes.

$$\sum_{s'} \beta'_2(s') = \beta'_2(0) + \beta'_2(1) + \beta'_2(2) + \beta'_2(3) = 1.476 + 6.689 + 1.469 + 0.327 = 9.96 .$$

No cálculo de $\gamma_3(2,3)$ teve-se em conta que o ramo em questão está associado ao símbolo $x_3 = \{-1,+1\}$.

Os valores normalizados das diversas probabilidades $P(s', s, \mathbf{y})$, que se calculam facilmente com a ajuda da Fig. 5 ou 6, são apresentados na Fig. 13. Por exemplo (veja-se a Fig. 12), no instante $k = 3$ $P_{norm}(2,3,\mathbf{y}) = P(2,3,\mathbf{y})/\Sigma_{P3}$ é igual a $P_{norm}(2,3,\mathbf{y}) \approx (0.01 \times 0.47 \times 0.001)/0.56 \approx 1.1.10^{-5}$, pois a soma dos produtos $\alpha\gamma\beta$ dos oito ramos é

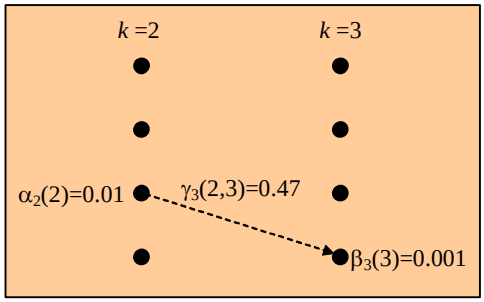


Fig. 12 Valores usados no cálculo de $P(2,3,\mathbf{y})$

$$\Sigma_{P3} = \sum_{R_o, R_1} \alpha_2(s')\gamma_3(s', s)\beta_3(s) =$$
$$= \sum_{R_o} \alpha_2(s')\gamma_3(s', s)\beta_3(s) + \sum_{R_1} \alpha_2(s')\gamma_3(s', s)\beta_3(s) = 0.493 + 0.068 \approx 0.56$$

Os valores de Σ_{Pk}, $\sum_{R_0 \text{ or } R_1} P(s', s, \mathbf{y})$ e $\sum_{R_0 \text{ or } R_1} P_{norm}(s', s, \mathbf{y})$ são apresentados na Tabela 1.

Tabela 1 – Valores de Σ_{Pk}, $\Sigma P(s',s,\mathbf{y})$ e $\Sigma P_{norm}(s',s,\mathbf{y})$

	$k = 1$	$k = 2$	$k = 3$	$k = 4$	$k = 5$	$k = 6$
Não normalizados						
$\sum_{R_1} P(s', s, \mathbf{y})$	1.214	0.177	0.068	0.306	0.000	0.000
$\sum_{R_0} P(s', s, \mathbf{y})$	0.203	0.139	0.493	0.001	0.379	8.084
Σ_{Pk}	1.417	0.316	0.562	0.307	0.379	8.084
Normalizados	⇓	⇓	⇓	⇓	⇓	⇓
$\sum_{R_1} P_{norm}(s', s, \mathbf{y})$	0.857	0.560	0.122	0.996	0.000	0.000
$\sum_{R_0} P_{norm}(s', s, \mathbf{y})$	0.143	0.440	0.878	0.004	1.000	1.000

Estamos a um passo de obter os valores pretendidos de $L(u_k|\mathbf{y})$ dados pela Eq. (14): recolhemos os valores das duas últimas linhas da Tabela 1 e obtemos

$$L(u_1|\mathbf{y}) = \ln \frac{P_{norm}(0,2,\mathbf{y})}{P_{norm}(0,0,\mathbf{y})} = \ln \frac{0.857}{0.143} = 1.79$$

$$L(u_2|\mathbf{y}) = \ln \frac{P_{norm}(0,2,\mathbf{y}) + P_{norm}(2,3,\mathbf{y})}{P_{norm}(0,0,\mathbf{y}) + P_{norm}(2,1,\mathbf{y})} = \ln \frac{0.560}{0.440} = 0.24$$

$$L(u_3|\mathbf{y}) = \ln \frac{P_{norm}(0,2,\mathbf{y}) + P_{norm}(1,2,\mathbf{y}) + P_{norm}(2,3,\mathbf{y}) + P_{norm}(3,3,\mathbf{y})}{P_{norm}(0,0,\mathbf{y}) + P_{norm}(1,0,\mathbf{y}) + P_{norm}(2,1,\mathbf{y}) + P_{norm}(3,1,\mathbf{y})} = \ln \frac{0.122}{0.878} = -1.98$$

$$L(u_4|\mathbf{y}) = \ln \frac{P_{norm}(0,2,\mathbf{y}) + P_{norm}(1,2,\mathbf{y}) + P_{norm}(2,3,\mathbf{y}) + P_{norm}(3,3,\mathbf{y})}{P_{norm}(0,0,\mathbf{y}) + P_{norm}(1,0,\mathbf{y}) + P_{norm}(2,1,\mathbf{y}) + P_{norm}(3,1,\mathbf{y})} = \ln \frac{0.996}{0.004} = 5.56$$

$$L(u_5|\mathbf{y}) = \ln \frac{0}{P(0,0,\mathbf{y}) + P(1,0,\mathbf{y}) + P(2,1,\mathbf{y}) + P(3,1,\mathbf{y})} = -\infty$$

$$L(u_6|\mathbf{y}) = \ln \frac{0}{P(0,0,\mathbf{y}) + P(1,0,\mathbf{y})} = -\infty$$

Os dois últimos valores são provocados pela terminação forçada da treliça. Em face dos valores obtidos a estimativa rígida da sequência **u** é $\hat{\mathbf{u}} = +1 \quad +1 \quad -1 \quad +1 \quad -1 \quad -1$ ou, em termos de valores 0 e 1,

1 1 0 1 0 0.

O percurso estimado pelo algoritmo BCJR está assinalado na Fig. 13.

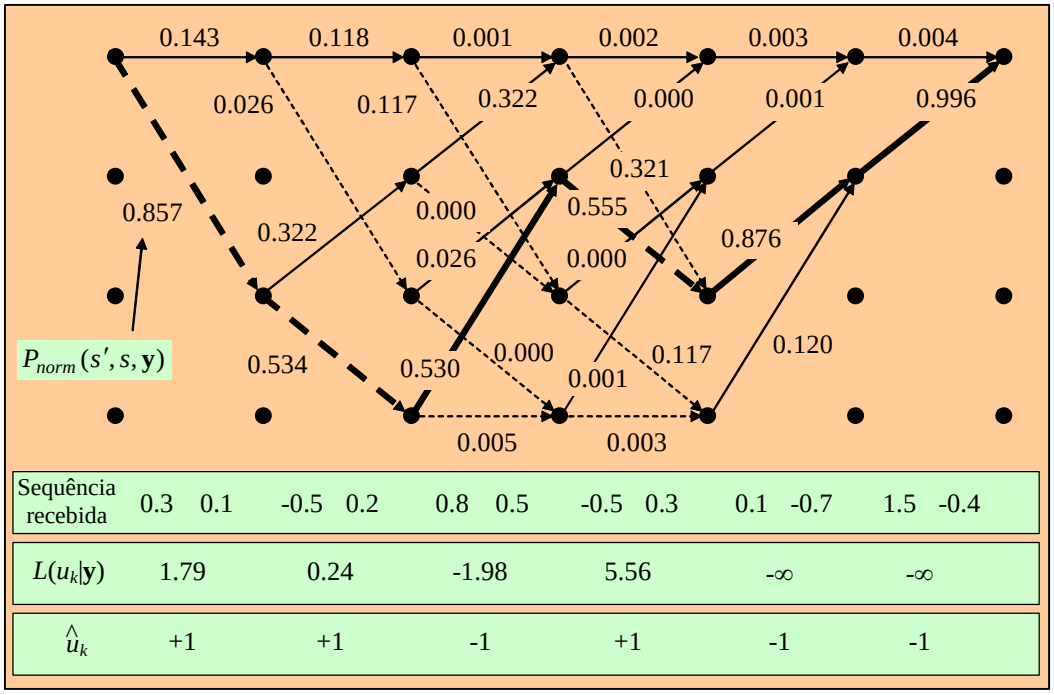


Fig. 13 Valores da probabilidade conjunta $P_{norm}(s', s, \mathbf{y})$ e da LLR $L(u_k|\mathbf{y})$ e estimativa de u_k.

7.2 Exemplo com o algoritmo max-log-MAP

Seguindo os procedimentos descrito atrás obter-se-iam os valores de A, B e Γ que estão na treliça da Fig. 14. Por exemplo, $A_4(1)$ e $B_3(2)$ foram calculados assim:

$$A_4(1) = \max\left[A_3(2) + \Gamma_4(2,1),\, A_3(3) + \Gamma_4(3,1)\right] = \\ = \max(2.01 - 2.01, 3.52 + 2.01) = 5.54$$

$$B_3(2) = \max\left[B_4(1) + \Gamma_4(2,1),\, B_4(3) + \Gamma_4(2,3)\right] = \\ = \max(-4.28 - 2.01, 0.76 + 2.01) = 2.77$$

As somas $A_{k-1}(s') + \Gamma_k(s', s) + B_k(s)$ calculam-se seguindo a Fig. 5 mas substituindo α, β e γ por A, B e Γ, respectivamente, e substituindo as multiplicações por adições, como na Fig. 7. Agora só há que calcular o valor máximo dessas somas para cada instante k e para cada conjunto R_0 e R_1. Subtraindo o primeiro valor do segundo obtemos a LLR da Tabela 2 e a respectiva estimativa de u_k.

Tabela 2 – Valores do algoritmo max-log-MAP

	$k = 1$	$k = 2$	$k = 3$	$k = 4$	$k = 5$	$k = 6$
$\max_{R_1}(\ )$	7.302	7.302	5.791	7.302	-∞	-∞
$\max_{R_0}(\ )$	5.791	6.798	7.302	1.762	7.302	7.302
$L(u_k\|\mathbf{y})$	1.511	0.504	-1.511	5.539	-∞	-∞
Estimativa de u_k	+1	+1	-1	+1	-1	-1

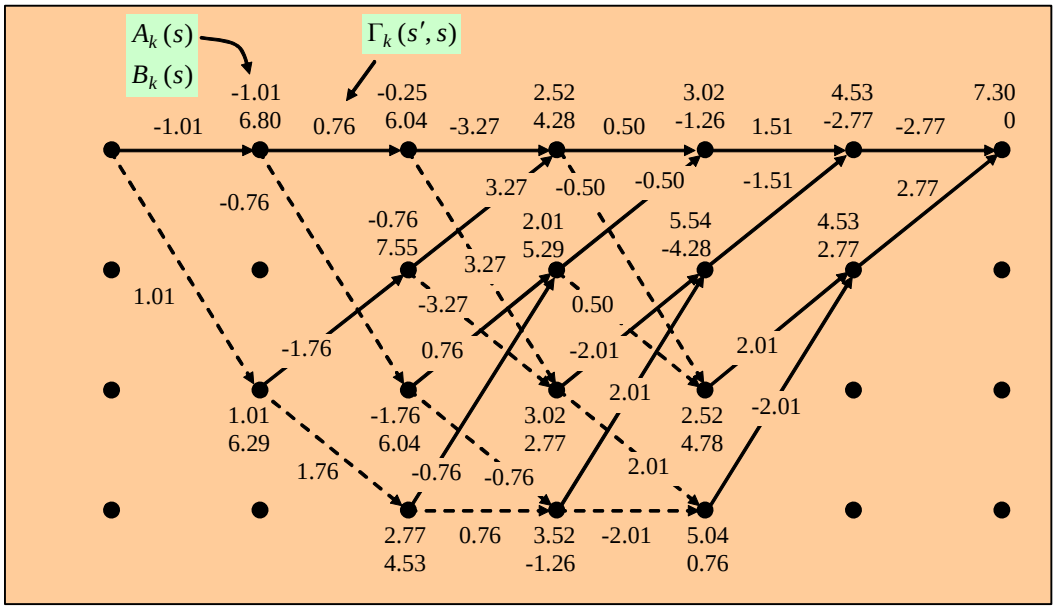


Fig. 14 Valores de *A*, *B* e Γ ao fim de toda a sequência de seis símbolos recebida.

A Fig. 15 ilustra os ramos correspondentes aos valores máximos em cada conjunto. O único percurso sem interrupções é o que corresponde aos bits de informação estimados.

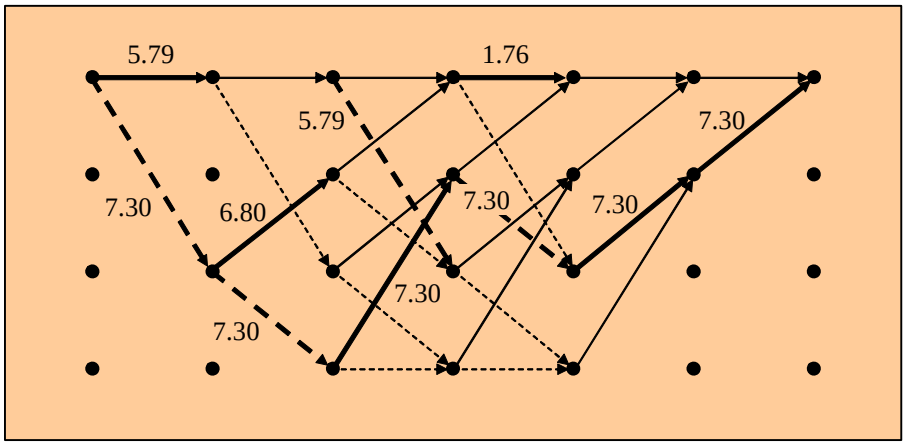


Fig. 15 Valores máximos das somas "$A + \Gamma + B$" em cada conjunto R_0 e R_1 no algoritmo max-log-MAP

7.3 Exemplo com o algoritmo log-MAP

Neste caso os valores de A e B são obtidos sem aproximações e a partir deles e de Γ (que é igual nos dois algoritmos simplificativos) obtêm-se os valores da Tabela 3. Como era de esperar, os valores de $L(u_k|\mathbf{y})$ são exactamente iguais aos do algoritmo BCJR.

Tabela 3 – Valores do algoritmo log-MAP

	$k=1$	$k=2$	$k=3$	$k=4$	$k=5$	$k=6$
$\max_{R_1}(\ )$	7.783	7.311	5.791	7.349	-∞	-∞
$\max_{R_0}(\ )$	5.996	6.805	7.303	1.765	7.805	7.934
$L(u_k\|\mathbf{y})$	1.79	0.24	-1.98	5.56	-∞	-∞
Estimativa de u_k	+1	+1	-1	+1	-1	-1

7.4 Exemplo de descodificação turbo

Vamos considerar a seguinte situação:

- Uma mensagem de nove bits nulos é aplicada a dois codificadores convolucionais sistemáticos e recursivos iguais, cada um com matriz geradora $\mathbf{G}(x) = [1\ (1+x^2)/(1+x+x^2)]$ e a treliça apresentada na Fig. 16. A sequência de saída do codificador turbo é obtida por perfuração de modo que a taxa de código global é de 1/2.
- O padrão de entrelaçamento é $\mathbf{P}$ = [1 4 7 2 5 9 3 6 8]. Assim, por exemplo, o terceiro elemento da versão entrelaçada da sequência arbitrária $\mathbf{m}$ = [2 4 3 1 8 5 9 6 0] é $\mathbf{m}_3^{(P)} = \mathbf{m}_{P_3} = \mathbf{m}_7 = 9$. Portanto, $\mathbf{m}^{(P)}$ = [2 1 9 4 8 0 3 5 6].
- O canal AWGN é tal que $E_c/N_0 = 0.25$ e $a = 1$, pelo que $L_c = 1$.
- A sequência de dezoito valores reais recebida é

$$\mathbf{y} = 0.3\ -4.0\ -1.9\ -2.0\ -2.4\ -1.3\ 1.2\ -1.1\ 0.7\ -2.0\ -1.0\ -2.1\ -0.2\ -1.4\ -0.3\ -0.1\ -1.1\ 0.3.$$

- Arbitra-se que a LLR a priori inicial é $L(u_k) = 0$.

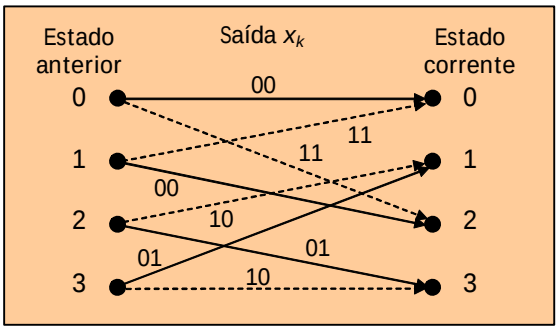


Fig. 16 A treliça do codificador RSC caracterizado por $\mathbf{G}(x) = [1\ (1+x^2)/(1+x+x^2)]$

A tabela seguinte contém as sequências de entrada do descodificador turbo da Fig. 9:

k	1	2	3	4	5	6	7	8	9
$L_c\mathbf{y}_{k1}$	0.3	-1.9	-2.4	1.2	0.7	-1.0	-0.2	-0.3	-1.1
$L_c\mathbf{y}_{k1}^{(P)}$	0.3	1.2	-0.2	-1.9	0.7	-1.1	-2.4	-1.0	-0.3
$L_c\mathbf{y}_{kp}^{(1)}$	-4.0	0	-1.3	0	-2.0	0	-1.4	0	0.3
$L_c\mathbf{y}_{kp}^{(2)}$	0	-2.0	0	-1.1	0	-2.1	0	-0.1	0

Na primeira iteração de descodificação a saída do descodificador 1 apresentaria os seguintes valores da LLR a posteriori $L_1(u_k|\mathbf{y})$ e da informação extrínseca $L_{e2}(u_k)$, com $L(u_k)$ e $L_c\, y_{k1}$ à entrada:

$L_1(u_k\|\mathbf{y})$	-4.74	-3.20	-3.66	1.59	1.45	-0.74	0.04	0.04	-1.63
$L(u_k)$	0	0	0	0	0	0	0	0	0
$L_c\, y_{k1}$	0.30	-1.90	-2.40	1.20	0.70	-1.00	-0.20	-0.30	-1.10
$L_{e1}(u_k)$	-5.04	-1.30	-1.26	0.39	0.75	0.26	0.24	0.34	-0.53
$L_1(u_k)$	-5.04	0.39	0.24	-1.30	0.75	-0.53	-1.26	0.26	0.34

Os valores da informação extrínseca, na quarta linha, foram calculados subtraindo a segunda e a terceira linha da primeira (Eq. (17).

Vemos que, se agora se estimasse a sequência de informação enviada, os valores da informação branda $L_1(u_k|\mathbf{y})$ fornecida pelo descodificador 1 originariam quatro bits errados, aqueles que são estimados quando $L_1(u_k|\mathbf{y})$ é positiva (nos instantes k = 4, 5, 7 e 8). Este mesmo descodificador transfere a informação extrínseca $L_{e1}(u_k)$ para o descodificador 2 após entrelaçamento – ou seja, o descodificador 2 recebe os valores $L_1(u_k)$ da última linha da tabela. É de notar, entretanto, que depois desta meia iteração e devido aos valores negativos elevados de $L_1(u_1|\mathbf{y})$ e $L_{e1}(u_1)$, já ganhámos uma confiança elevada acerca da decisão sobre o primeiro bit da sequência – muito provavelmente será u_1 = -1 (bem, já o sabíamos…). Porém, não estamos tão confiantes acerca dos outros bits, especialmente aqueles para os quais os valores absolutos de $L_1(u_1|\mathbf{y})$ e $L_{e1}(u_1)$ são baixos.

O descodificador 2 vai agora lidar com os valores sistemáticos entrelaçados $L_c y_{k1}^{(P)}$, os valores de paridade $\mathbf{y}_{kp}^{(2)}$ e as novas LLRs a priori $L_1(u_k)$. Os resultados, obtidos de acordo com a Eq. (17), são apresentados na tabela seguinte, onde a sequência $L_2(u_k)$ representa a versão desentrelaçada de $L_{e2}(u_k)$ que vai servir de estimativa mais refinada da LLR a priori $L(u_k)$. Isto significa que $L_2(u_k)$ vai ser usada como uma das entradas do descodificador 1 na iteração seguinte.

$L_2(u_k\|\mathbf{y})$	-3.90	0.25	0.18	-3.04	1.23	-1.44	-3.65	-0.72	0.04
$L_1(u_k)$	-5.04	0.39	0.24	-1.30	0.75	-0.53	-1.26	0.26	0.34
$L_c y_{k1}^{(P)}$	0.30	1.20	-0.20	-1.90	0.70	-1.10	-2.40	-1.00	-0.30
$L_{e2}(u_k)$	0.85	-1.34	0.14	0.16	-0.22	0.19	0.01	0.02	0.00
$L_2(u_k)$	0.85	0.16	0.01	-1.34	-0.22	0.02	0.14	0.00	0.19

Como é que $L_2(u_k)$ foi calculada? Como se disse antes o elemento de ordem P_i de $L_2(u_k)$ é o elemento de ordem i da sequência $L_{e2}(u_k)$. Por exemplo, o quarto elemento de $L_2(u_k)$ é igual a

$$\left[L_2(u_k)\right]_4 = \left[L_2(u_k)\right]_{P_2} = \left[L_{e_2}(u_k)\right]_2 = -1.34$$

pois $4 = P_2$.

Podemos ver de novo que ainda obteríamos quatro bits errados se tomássemos decisões rígidas na sequência $L_2(u_k|\mathbf{y})$ depois de esta ser reorganizada na ordem correcta através de desentrelaçamento. A sequência reordenada é -3.90 -3.04 -3.65 0.25 1.23 -0.72 0.18 0.04 -1.44 e provoca erros nas posições 4, 5, 7 e 8.

O que atrás se descreve deve ser repetido iteração após iteração. Por exemplo, com cinco iterações obteríamos a Tabela 4, onde os valores sombreados, positivos, indicam decisões erradas, se tomadas.

Bastam três iterações para que os quatro bits inicialmente errados sejam corrigidos. A partir daí não vale a pena continuar. De facto, como os valores das razões LLR a posteriori estabilizam muito depressa não ganhamos nada em prosseguir com a descodificação.

Tabela 4 – Saídas dos descodificadores turbo durante cinco iterações

Iteração	$k \rightarrow$	1	2	3	4	5	6	7	8	9
1	$L_1(u_k\|\mathbf{y})$	-4.74	-3.20	-3.66	1.59	1.45	-0.74	0.04	0.04	-1.63
2	$L_1(u_k\|\mathbf{y})$	-3.64	-2.84	-3.28	0.11	0.27	-0.95	-0.17	-0.25	-1.40
3	$L_1(u_k\|\mathbf{y})$	-3.65	-3.00	-3.35	-0.58	-0.34	-1.07	-0.61	-0.63	-1.53
4	$L_1(u_k\|\mathbf{y})$	-3.85	-3.21	-3.49	-1.02	-0.74	-1.20	-0.93	-0.90	-1.75
5	$L_1(u_k\|\mathbf{y})$	-4.08	-3.42	-3.64	-1.35	-1.05	-1.32	-1.18	-1.11	-1.95
1	$L(u_k\|\mathbf{y})$	-3.90	-3.04	-3.65	0.25	1.23	-0.72	0.18	0.04	-1.44
2	$L(u_k\|\mathbf{y})$	-3.61	-2.96	-3.29	-0.41	0.13	-0.97	-0.43	-0.25	-1.48
3	$L(u_k\|\mathbf{y})$	-3.75	-3.11	-3.35	-0.87	-0.45	-1.08	-0.80	-0.63	-1.66
4	$L(u_k\|\mathbf{y})$	-3.98	-3.32	-3.50	-1.22	-0.85	-1.21	-1.07	-0.90	-1.86
5	$L(u_k\|\mathbf{y})$	-4.21	-3.52	-3.65	-1.51	-1.15	-1.33	-1.28	-1.11	-2.06

A Fig. 17 mostra a evolução de $L_1(u_k|\mathbf{y})$ e $L(u_k|\mathbf{y})$ ao longo das iterações. Observamos que todas as mudanças de sinal ocorrem no início da descodificação.

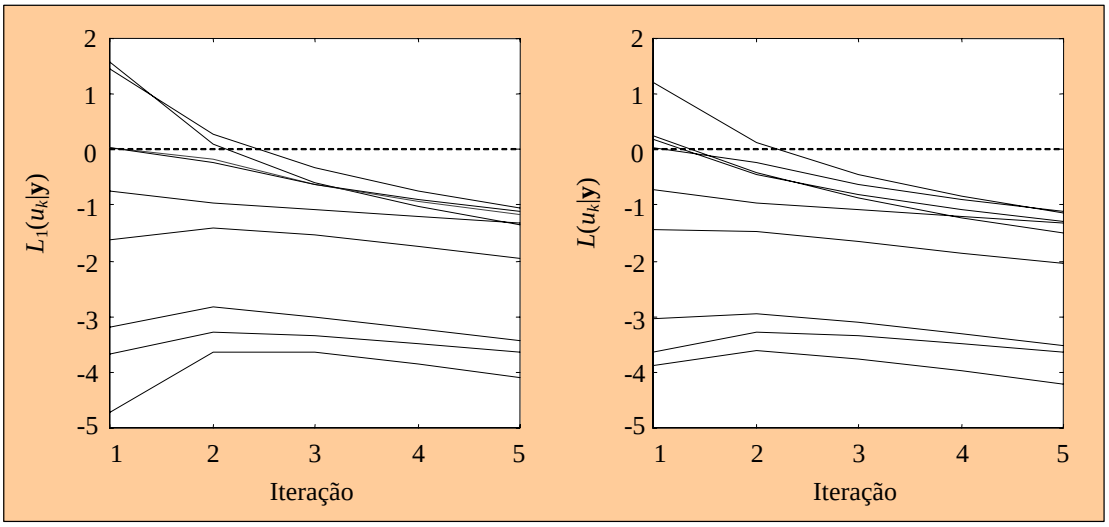


Fig. 17 As LLR a posteriori $L_1(u_k|\mathbf{y})$ (primeiro descodificador) e $L(u_k|\mathbf{y})$ (segundo descodificador)

8 Apêndice: a teoria dos algoritmos BCJR, log-MAP e max-log-MAP

Neste Apêndice são apresentados os desenvolvimentos matemáticos dos algoritmos BCJR, log-MAP e max-log-MAP. Todos eles fazem a estimação dos bits de entrada do codificador instante a instante, ao contrário do algoritmo de Viterbi, que encontra o percurso de máxima verosimilhança e através dele estima a sequência enviada.

Começamos por uma relação de probabilidades condicionais que vai ser usada varia vezes:

$$P(A,B|C) = P(A|B,C)P(B|C) \tag{18}$$

Esta relação prova-se recorrendo à regra de Bayes $P(A,B) = P(A|B)P(B)$ e definindo os conjuntos de eventos $D = \{A,B\}$ e $E = \{B,C\}$:

$$\begin{aligned} P(A,B|C) = P(D|C) = \frac{P(D,C)}{P(C)} = \\ = \frac{P(A,B,C)}{P(C)} = \frac{P(A,E)}{P(C)} = \\ = \frac{P(A|E)P(E)}{P(C)} = \frac{P(A|E)P(B,C)}{P(C)} = \\ = \frac{P(A|E)P(B|C)P(C)}{P(C)} = P(A|B,C)P(B|C) \end{aligned}$$

8.1 O algoritmo BCJR, ou MAP

Nas secções seguintes mostra-se como determinar teoricamente as várias variáveis deste algoritmo. Numa secção final mostra-se como o algoritmo pode ser aplicado na descodificação iterativa de códigos concatenados, de que os turbo-códigos são o exemplo mais preponderante [6].

8.1.1 A LLR a posteriori $L(u_k|\mathbf{y})$

Desejamos calcular $L(u_k|\mathbf{y}) = \ln \dfrac{P(u_k = +1|\mathbf{y})}{P(u_k = -1|\mathbf{y})}$. Se $L(u_k|\mathbf{y}) > 0$ estima-se que foi enviado o bit u_k = +1, se não estima-se que foi u_k = -1. Ora observemos a Fig. 18, que mostra o troço de uma treliça de quatro estados entre os instantes k-1 e k. Existem oito transições entre os estados $S_{k-1} = s'$ e os estados seguintes $S_k = s$: quatro são devidas a um bit de entrada -1 (as que na figura estão realçadas a traço grosso) e as outras quatro são devidas a um bit de entrada +1. Cada uma dessas transições é devida inequivocamente a um bit conhecido (basta observar se o ramo é contínuo ou tracejado). Logo, as probabilidades de u_k = -1 ou u_k = +1, dado que conhecemos a sequência **y**, são iguais à probabilidade de a transição (s',s) entre estados corresponder a um dos ramo contínuos ou tracejados, respectivamente. Como as transições são mutuamente exclusivas, pois só uma é que pode ocorrer em cada instante, a probabilidade de ocorrer qualquer delas é igual à soma das probabilidades individuais, isto é,

$$P(u_k = -1|\mathbf{y}) = \sum_{\mathbf{R}_0} P(s', s|\mathbf{y})$$

$$P(u_k = +1|\mathbf{y}) = \sum_{\mathbf{R}_1} P(s', s|\mathbf{y}),$$

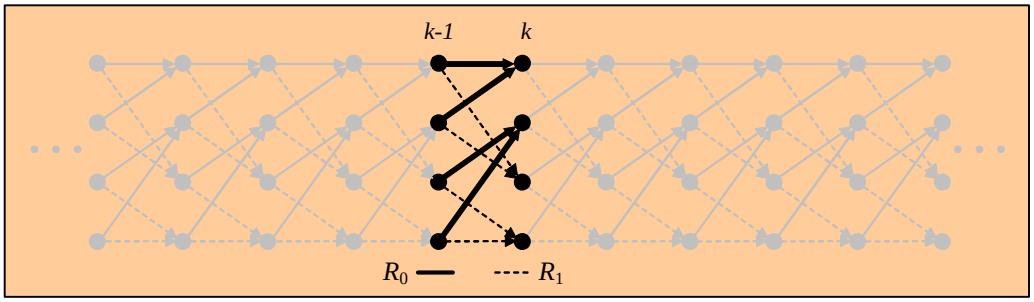


Fig. 18 Troço da treliça entre os instantes k-1 e k

em que R_0 e R_1 representam, respectivamente, o conjunto das transições do estado $S_{k-1} = s'$ para o estado $S_k = s$ provocadas por u_k = -1ou u_k = +1. A Fig. 18 mostra precisamente os conjuntos de quatro transições R_0 e R_1. Assim, obtemos para $L(u_k|\mathbf{y})$:

$$\begin{aligned} L(u_k|\mathbf{y}) = \ln \frac{P(u_k = +1|\mathbf{y})}{P(u_k = -1|\mathbf{y})} = \ln \frac{\sum_{\mathbf{R}_1} P(s', s|\mathbf{y})}{\sum_{\mathbf{R}_0} P(s', s|\mathbf{y})} = \\ = \ln \frac{\sum_{\mathbf{R}_1} P(s', s, \mathbf{y})/P(\mathbf{y})}{\sum_{\mathbf{R}_0} P(s', s, \mathbf{y})/P(\mathbf{y})} = \ln \frac{\sum_{\mathbf{R}_1} P(s', s, \mathbf{y})}{\sum_{\mathbf{R}_0} P(s', s, \mathbf{y})} \end{aligned} \tag{19}$$

8.1.2 $P(s', s, \mathbf{y})$ é um produto $\alpha\gamma\beta$

A sequência recebida **y**, de N símbolos, pode ser seccionada em três subsequências: uma sequência passada, $\mathbf{y}_{<k}$, uma sequência presente (um símbolo), $\mathbf{y}_k$, e uma sequência futura, $\mathbf{y}_{>k}$, como está na Eq. (3). Escrevendo $P(s', s, \mathbf{y}) = P(s', s, \mathbf{y}_{<k}, \mathbf{y}_k, \mathbf{y}_{>k})$ e usando a regra de Bayes temos

$$P(s', s, \mathbf{y}) = P(s', s, \mathbf{y}_{<k}, \mathbf{y}_k, \mathbf{y}_{>k}) = \\ = P(y_{>k} \left| s', s, \mathbf{y}_{<k}, \mathbf{y}_k \right.) P(s', s, \mathbf{y}_{<k}, \mathbf{y}_k)$$

Acontece que se conhecermos o estado corrente $S_k = s$ a sequência recebida após o instante k, $\mathbf{y}_{>k}$, não vai depender nem do estado anterior s' nem das sequências passada ou presente, $\mathbf{y}_{<k}$ e $\mathbf{y}_k$, visto o canal não ter memória. Assim,

$$P(s', s, \mathbf{y}) = P(y_{>k} \left| s \right.) P(s', s, \mathbf{y}_{<k}, \mathbf{y}_k) .$$

Desenvolvendo o segundo membro e notando de novo que num canal sem memória se conhecermos o estado anterior s' o estado seguinte s e o símbolo corrente y_k não dependem da sequência passada $\mathbf{y}_{<k}$, obtemos

$$P(s', s, \mathbf{y}) = P(y_{>k} \left| s \right.) P(\mathbf{y}_k, s \left| s', \mathbf{y}_{<k} \right.) P(s', \mathbf{y}_{<k}) = \\ = \underbrace{P(y_{>k} \left| s \right.)}_{\beta_k(s)} \underbrace{P(\mathbf{y}_k, s \left| s' \right.)}_{\gamma_k(s', s)} \underbrace{P(s', \mathbf{y}_{<k})}_{\alpha_{k-1}(s')} = \\ = \alpha_{k-1}(s') \gamma_k(s', s) \beta_k(s)$$

A probabilidade $\alpha_{k-1}(s') = P(s', \mathbf{y}_{<k})$ representa a probabilidade de no instante $k-1$ se estar no estado s' e de a sequência entretanto recebida ser $y_{<k}$. A probabilidade $\gamma_k(s', s) = P(\mathbf{y}_k, s \left| s' \right.)$ representa a probabilidade de, dado que o estado anterior é s', o estado seguinte ser s e o símbolo recebido ser y_k. A probabilidade $\beta_k(s) = P(\mathbf{y}_{>k} \left| s \right.)$, finalmente, representa a probabilidade de, dado que estamos no estado s, a sequência futura ser $y_{>k}$.

Teremos, portanto,

$$L(u_k \left| \mathbf{y} \right.) = \ln \frac{\sum_{R_1} P(s', s, \mathbf{y})}{\sum_{R_0} P(s', s, \mathbf{y})} = \ln \frac{\sum_{R_1} \alpha_{k-1}(s') \gamma_k(s', s) \beta_k(s)}{\sum_{R_0} \alpha_{k-1}(s') \gamma_k(s', s) \beta_k(s)}$$

Como se explica no texto principal, é conveniente normalizar $P(s', s, \mathbf{y})$. Assim, em vez de usar o simples produto de três factores $\alpha\gamma\beta$ é mais seguro usar a expressão

$$P_{norm}(s', s, \mathbf{y}) = \frac{P(s', s, \mathbf{y})}{\sum_{R_0, R_1} P(s', s, \mathbf{y})} = \frac{P(s', s, \mathbf{y})}{\sum_{R_0} P(s', s, \mathbf{y}) + \sum_{R_1} P(s', s, \mathbf{y})}$$

Esta substituição não tem qualquer efeito no valor final de $L(u_k|\mathbf{y})$, naturalmente.

8.1.3 O presente: determinação de γ

Usando a relação de probabilidades condicionais da Eq. (18) obtemos, para $\gamma_k(s', s)$,

$$\gamma_k(s', s) = P(\mathbf{y}_k, s \left| s' \right.) = \\ = P(\mathbf{y}_k \left| s', s \right.) P(s \left| s' \right.) =$$

Ora se conhecermos o estado *s'* a probabilidade de no instante seguinte atingirmos um dos dois estados *s* possíveis é igual à probabilidade de o bit de entrada ser $u_k = \pm 1$ (ou seja, o ramo da treliça ser contínuo ou tracejado). Portanto, $P(s|s') = P(u_k)$. Por exemplo, se $P(u_k = \pm 1) = 1/2$ e estivermos num dado estado, é tão provável atingir um dos dois estados seguintes como o outro. Porém, se $P(u_k = +1) = 3/5$ então $P(s' \underset{tracejado}{\rightarrow} s) = 3/5$. Quanto ao outro factor, $P(\mathbf{y}_k|s', s)$, repare-se que a ocorrência conjunta dos estados consecutivos $S_{k-1} = s'$ e $S_k = s$ equivale à ocorrência do símbolo codificado x_k correspondente, isto é, $P(\mathbf{y}_k|s', s) = P(\mathbf{y}_k|x_k)$. Ou seja,

$$\gamma_k(s', s) = P(\mathbf{y}_k|x_k)P(u_k) \tag{20}$$

x_k pode ser uma das 2^n palavras de *n* bits produzidas em cada instante no codificador. Assim, alguns dos valores de $\gamma_k(s', s)$ terão de ser iguais, só havendo, no máximo, 2^n valores diferentes.

Voltemos a $P(u_k)$. Da definição $L(u_k) = \ln \frac{P(u_k = +1)}{P(u_k = -1)} = \ln \frac{P(u_k = +1)}{1 - P(u_k = +1)}$ tira-se imediatamente

$$P(u_k = \pm 1) = \frac{e^{u_k L(u_k)}}{1 + e^{u_k L(u_k)}}$$

que pode ainda ser escrita como

$$\begin{aligned} P(u_k = \pm 1) &= = \frac{e^{L(u_k)/2}}{1 + e^{L(u_k)}} e^{u_k L(u_k)/2} = \\ &= C_{1k} e^{u_k L(u_k)/2} \end{aligned} \tag{21}$$

A fracção $C_{1k} = e^{L(u_k)/2} \Big/ \left[1 + e^{L(u_k)}\right]$, que vale 1/2 se os bits $u_k = \pm 1$ forem equiprováveis, não depende de u_k ser +1 ou -1 pelo que, surgindo no numerador e no denominador da expressão da LLR a posteriori (Eq. (19)), vai desaparecer nesse cálculo. Voltaremos ao assunto.

A probabilidade $P(\mathbf{y}_k|x_k)$ de se receberem *n* valores $y_k = y_{k1} y_{k2} \ldots y_{kn}$ dado que se enviaram *n* valores $x_k = x_{k1} x_{k2} \ldots x_{kn}$ será igual ao produto das probabilidades individuais $P(y_{kl}|x_{kl})$, $l = 1, 2, \ldots, n$, pois não havendo memória no canal as sucessivas transmissões são estatisticamente independentes:

$$P(\mathbf{y}_k|\mathbf{x}_k) = \prod_{l=1}^{n} P(y_{kl}|x_{kl}) . \tag{22}$$

No caso muito vulgar de $n = 2$ temos simplesmente $P(\mathbf{y}_k|\mathbf{x}_k) = P(y_{k1}|x_{k1})P(y_{k2}|x_{k2})$.

Estas probabilidades dependem do canal e da modulação usados. Tratando-se de modulação BPSK os sinais transmitidos têm amplitudes $x_{kl}E_c = \pm E_c$, em que E_c é a energia transmitida por bit codificado. Suponhamos que o canal é um canal de ruído branco gaussiano de densidade espectral de potência bilateral $N_0/2$ e amplitude de *fading a*. Na saída do filtro adaptado do receptor o sinal apresenta uma amplitude igual a $y'_{kl} = a x_{kl} \sqrt{E_c} + n' = \pm a \sqrt{E_c} + n'$, em que *n'* representa uma amostra de ruído branco de média nula e variância $\sigma^2_{n'} = N_0/2$. Normalizando as amplitudes no receptor obtemos

$$y_{kl} = \frac{y'_{kl}}{\sqrt{E_c}} = a x_{kl} + \frac{n'}{\sqrt{E_c}} = a x_{kl} + n \,,$$

onde agora a variância do ruído $n = n' \big/ \sqrt{E_c}$ é $\sigma_n^2 = \sigma_{n'}^2 \big/ E_c = N_0 / 2E_c$. Teremos então, finalmente,

$$P(y_{kl} \,|\, x_{kl}) = \frac{1}{\sqrt{\pi\, N_0 / E_c}} e^{-\frac{E_c}{N_0}(y_{kl} - a x_{kl})^2} \,.$$

Desenvolvendo a Eq. (22) obtemos

$$P(\mathbf{y}_k \,|\, \mathbf{x}_k) = \left[\frac{1}{\left(\sqrt{\pi\, N_0 / E_c}\right)^n} \exp\left(-\frac{E_c}{N_0} \sum_{l=1}^{n} y_{kl}^2 \right) \exp\left(-\frac{E_c}{N_0} a^2 \sum_{l=1}^{n} x_{kl}^2 \right) \right] \exp\left(2a \frac{E_c}{N_0} \sum_{l=1}^{n} x_{kl} y_{kl} \right) =$$

$$= C_{2k} \exp\left(2a \frac{E_c}{N_0} \sum_{l=1}^{n} x_{kl} y_{kl} \right)$$

O produto de factores

$$C_{2k} = \frac{1}{\left(\sqrt{\pi\, N_0 / E_c}\right)^n} \exp\left(-\frac{E_c}{N_0} \sum_{l=1}^{n} y_{kl}^2 \right) \exp\left(-\frac{E_c}{N_0} a^2 \sum_{l=1}^{n} x_{kl}^2 \right)$$

não depende nem do sinal de u_k nem da palavra de código x_k. De facto, o primeiro factor do produto só depende do canal, o segundo depende do canal e da sequência y_k e o terceiro, no qual $\sum_{l=1}^{n} x_{kl}^2 = n$, depende do canal e da amplitude de *fading*. Isto significa que C_{2k} vai estar quer no numerador quer no denominador da Eq. (19) e por isso vai desaparecer mais tarde.

O somatório $\sum_{l=1}^{n} x_{kl} y_{kl}$ representa a soma dos produtos elemento-a-elemento das palavras x_k (enviada) e y_k (recebida). Se encararmos cada uma dessas palavras como um vector de n elementos o somatório é igual ao respectivo produto interno:

$$\begin{aligned} \mathbf{x}_k &= [x_{k1} x_{k2} \ldots x_{kn}] \\ \mathbf{y}_k &= [y_{k1} y_{k2} \ldots y_{kn}] \end{aligned} \qquad \Rightarrow \qquad \sum_{l=1}^{n} x_{kl} y_{kl} = \mathbf{x}_k \bullet \mathbf{y}_k$$

Disse-se atrás que $L_c = 4a \frac{E_c}{N_0}$. Logo,

$$P(\mathbf{y}_k \,|\, x_k) = C_{2k} \exp\left(\frac{L_c}{2} \sum_{l=1}^{n} x_{kl} y_{kl} \right) =$$

$$= C_{2k} \exp\left(\frac{L_c}{2} \mathbf{x}_k \bullet \mathbf{y}_k \right)$$

Voltando à Eq. (20) podemos agora escrever a expressão final de $\gamma_k(s',s)$ tal como aparece na Eq. (9):

$$\begin{aligned}\gamma_k(s',s) &= P(\mathbf{y}_k|x_k)P(u_k) = \\ &= C_{2k}\exp\left(\frac{L_c}{2}\sum_{l=1}^{n}x_{kl}y_{kl}\right)C_{1k}e^{u_kL(u_k)/2} = \\ &= C_k e^{u_kL(u_k)/2}\exp\left(\frac{L_c}{2}\sum_{l=1}^{n}x_{kl}y_{kl}\right)\end{aligned}$$

onde fizemos $C_k = C_{1k}C_{2k}$. C_k representa uma quantidade que vai desaparecer ao calcularmos $L(u_k|\mathbf{y})$. O seu valor é, pois, irrelevante.

Já temos $\gamma_k(s',s)$. Falta-nos α e β. Comecemos por α.

8.1.4 O passado: determinação de α

Por definição $\alpha_{k-1}(s') = P(s', \mathbf{y}_{<k})$, que escreveremos $\alpha_k(s) = P(s, \mathbf{y}_{<k+1})$ por conveniência. Ora

$$\begin{aligned}\alpha_k(s) &= P(s, \mathbf{y}_{<k+1}) = \\ &= P(s, \mathbf{y}_{<k}, \mathbf{y}_k)\end{aligned}$$

Sabe-se da Teoria das Probabilidades que $P(A) = \sum_B P(A,B)$, em que o somatório se estende a todos os valores de *B* possíveis. Então

$$\begin{aligned}\alpha_k(s) &= P(s, \mathbf{y}_{<k}, \mathbf{y}_k) = \\ &= \sum_{s'} P(s, s', \mathbf{y}_{<k}, \mathbf{y}_k)\end{aligned}$$

Usando mais uma vez o argumento da falta de memória no canal teremos

$$\begin{aligned}\alpha_k(s) &= \sum_{s'} P(s, \mathbf{y}_k|s', \mathbf{y}_{<k})P(s', \mathbf{y}_{<k}) = \\ &= \sum_{s'} P(s, \mathbf{y}_k|s')P(s', \mathbf{y}_{<k}) = \sum_{s'}\gamma_k(s',s)\alpha_{k-1}(s')\end{aligned}$$

Contudo, para evitar problemas numéricos é conveniente normalizar α dividindo-o pela soma dos α em todas as transições de estados, ou seja, calculando

$$\alpha_k(s) = \frac{\sum_{s'}\alpha_{k-1}(s')\gamma_k(s',s)}{\sum_s\sum_{s'}\alpha_{k-1}(s')\gamma_k(s',s)} = \frac{\alpha'_k(s)}{\sum_s\alpha'_k(s)}$$

Esta normalização não afecta o resultado final de LLR. Agora só temos que partir das condições iniciais adequadas a uma treliça que começa no estado nulo, e que são:

$$\alpha_0(s) = \begin{cases} 1 & s = 0 \\ 0 & s \neq 0 \end{cases}$$

8.1.5 O futuro: determinação de β

A fórmula recursiva de β obtém-se de modo semelhante. Por definição é $\beta_k(s) = P(\mathbf{y}_{>k}|s)$, que escreveremos $\beta_{k-1}(s') = P(\mathbf{y}_{>k-1}|s')$ por conveniência. Usando o já habitual argumento de falta de memória do canal teremos

$$\begin{aligned}
\beta_{k-1}(s') &= P(\mathbf{y}_{>k-1}|s') = \\
&= \sum_s P(s, \mathbf{y}_{>k-1}|s') = \sum_s P(s, \mathbf{y}_k, \mathbf{y}_{>k}|s') = \\
&= \sum_s P(\mathbf{y}_{>k}|s', s, \mathbf{y}_k) P(s, \mathbf{y}_k|s') = \sum_s P(\mathbf{y}_{>k}|s) P(s, \mathbf{y}_k|s') = \\
&= \sum_s \beta_k(s) \gamma_k(s', s)
\end{aligned}$$

É de novo conveniente normalizar β pela sua soma em todas as transições de estados pelo que, em vez das equações anteriores, devemos usar

$$\beta_{k-1}(s') = \frac{\sum_s \beta_k(s) \gamma_k(s', s)}{\sum_{s'} \sum_s \beta_k(s) \gamma_k(s', s)} = \frac{\beta'_{k-1}(s')}{\sum_{s'} \beta'_{k-1}(s')}.$$

A partir do momento em que conhecermos $\beta_N(s)$ podemos calcular os outros valores recursivamente. Os valores iniciais são os seguintes, para uma treliça terminada no estado nulo:

$$\beta_N(s) = \begin{cases} 1 & s = 0 \\ 0 & s \neq 0 \end{cases}$$

Várias alternativas foram propostas no intuito de aliviar a desvantagem de ter de se esperar pelo fim da sequência **y** para se começar a calcular β [7].

8.1.6 Aplicação do algoritmo BCJR na descodificação iterativa

Já vimos na Secção 6 como podemos aplicar o algoritmo BCJR e as suas simplificações à descodificação iterativa de códigos turbo. Vamos ver agora como se chega à Eq. (16), aquela que exprime $L(u_k|\mathbf{y})$ como uma soma de três parcelas se o código for sistemático.

Suponhamos que o primeiro bit codificado, x_{k1}, é igual ao bit de informação u_k. Se expandirmos a Eq. (9) de forma a evidenciar esse bit sistemático obtemos

$$\gamma_k(s',s) = C_k e^{u_k L(u_k)/2} \exp\left(\frac{L_c}{2}\sum_{l=1}^{n} x_{kl} y_{kl}\right) =$$

$$= C_k e^{u_k L(u_k)/2} \exp\left[\frac{L_c}{2}\left(x_{k1} y_{k1} + \sum_{l=2}^{n} x_{kl} y_{kl}\right)\right] =$$

$$= C_k e^{\frac{u_k}{2}\left[L(u_k) + L_c y_{k1}\right]} \exp\left(\frac{L_c}{2}\sum_{l=2}^{n} x_{kl} y_{kl}\right)$$

Definindo a segunda exponencial como a nova variável

$$\chi_k(s',s) = \exp\left(\frac{L_c}{2}\sum_{l=2}^{n} x_{kl} y_{kl}\right) \tag{23}$$

(que se simplifica em $\chi_k(s',s) = e^{L_c x_{k2} y_{k2}/2}$ se $n = 2$) obtemos para $\gamma_k(s',s)$:

$$\gamma_k(s',s) = C_k e^{\frac{u_k}{2}\left[L(u_k) + L_c y_{k1}\right]} \chi_k(s',s,\mathbf{y}) . \tag{24}$$

Introduzindo $\chi_k(s',s)$ na Eq. (4) passamos a ter

$$L(u_k|\mathbf{y}) = \ln \frac{\sum_{R_1} \alpha_{k-1}(s')\gamma_k(s',s)\beta_k(s)}{\sum_{R_0} \alpha_{k-1}(s')\gamma_k(s',s)\beta_k(s)} =$$

$$= \ln \frac{\sum_{R_1} C_k e^{\frac{u_k}{2}\left[L(u_k) + L_c y_{k1}\right]} \alpha_{k-1}(s')\chi_k(s',s)\beta_k(s)}{\sum_{R_0} C_k e^{\frac{u_k}{2}\left[L(u_k) + L_c y_{k1}\right]} \alpha_{k-1}(s')\chi_k(s',s)\beta_k(s)}$$

Como os conjuntos R_1 e R_0 estão associados a bits de entrada +1 e -1, respectivamente, podemos substituir u_k no numerador e denominador por estes respectivos valores. Obtemos então

$$L(u_k|\mathbf{y}) = \ln\left\{e^{L(u_k)} e^{L_c y_{k1}} \frac{\sum_{R_1} \alpha_{k-1}(s')\chi_k(s',s)\beta_k(s)}{\sum_{R_0} \alpha_{k-1}(s')\chi_k(s',s)\beta_k(s)}\right\} =$$

$$= L(u_k) + L_c y_{k1} + \ln \frac{\sum_{R_1} \alpha_{k-1}(s')\chi_k(s',s)\beta_k(s)}{\sum_{R_0} \alpha_{k-1}(s')\chi_k(s',s)\beta_k(s)}$$

Definindo agora

$$L_e(u_k) = \ln \frac{\sum_{R_1} \alpha_{k-1}(s')\chi_k(s',s)\beta_k(s)}{\sum_{R_0} \alpha_{k-1}(s')\chi_k(s',s)\beta_k(s)} \tag{25}$$

obtemos finalmente

$$L(u_k|\mathbf{y}) = L(u_k) + L_c y_{k1} + L_e(u_k), \qquad \text{c.q.d.} \tag{26}$$

ou ainda

$$L_e(u_k) = L(u_k|\mathbf{y}) - L(u_k) - L_c y_{k1}. \tag{27}$$

Recorda-se que $L_e(u_k)$ é a informação extrínseca que um descodificador fornece ao seguinte na descodificação iterativa (veja-se a Fig. 9 de novo), $L(u_k)$ é a LLR a priori dos bits de informação e L_c é a medida de fiabilidade do canal.

Note-se que não é verdadeiramente necessário calcular $\chi_k(s',s)$ para obtermos a informação extrínseca, a única informação que cada descodificador fornece ao outro. Digamos que há duas maneiras de calcular $L_e(u_k)$:

- Calcula-se $\chi_k(s',s)$ pela Eq. (23), $\gamma_k(s',s)$ pela Eq. (24), $\alpha_{k-1}(s')$ e $\beta_k(s)$ pelas equações recursivas (10)) e (11) (ou ainda melhor, pelas Eqs. (12) e (13)) e $L_e(u_k)$ pela Eq. (25) – e se necessário calcula-se também $L(u_k|\mathbf{y})$ (Eq. (26));
- Calculam-se $\gamma_k(s',s)$, $\alpha_{k-1}(s')$, $\beta_k(s)$ e $L(u_k|\mathbf{y})$ tal e qual como no algoritmo BCJR (ou seja, usando as Eqs. (9), (10) (ou (12)), (11) (ou (13)) e (4), respectivamente) e, por subtracção, calcula-se $L_e(u_k)$ (Eq. (27).

A descodificação iterativa é descrita na Secção 6.

8.2 Os algoritmos log-MAP e max-log-MAP

Nestes algoritmos substituem-se as multiplicações do algoritmo BCJR por adições. Para isso usa-se o chamado logaritmo jacobiano

$$\ln(e^a + e^b) = \max(a,b) + \ln(1 + e^{-|a-b|}) \tag{28}$$

O logaritmo jacobiano é aproximadamente igual a $\max(a,b)$ funcionando a parcela $\ln(1+e^{-|a-b|})$ essencialmente como um termo corrector. A diferença entre os algoritmos log-MAP e max-log-MAP está no modo como lidam com este logaritmo. Se usarmos a fórmula exacta da Eq. (28) temos o algoritmo log-MAP; se usarmos a expressão aproximada $\ln(e^a + e^b) \approx \max(a,b)$ temos o algoritmo max-log-MAP. Unifiquemos a notação através da função $\max^*(a,b)$, assim definida:

$$\max^*(a,b) = \begin{cases} \ln(e^a + e^b) = \max(a,b) + \ln(1+e^{-|a-b|}) & \log - \text{MAP} \\ \max(a,b) & \max - \log - \text{MAP} \end{cases} \tag{29}$$

Por causa da aproximação naturalmente que o algoritmo max-log-MAP terá um desempenho inferior ao do outro, que se comporta exactamente como o algoritmo BCJR original pois não usa aproximações. Além disso e segundo os seus proponentes [5], no algoritmo log-MAP basta usar uma tabela

com oito valores do termo corrector $\ln(1+e^{-x})$, com x entre 0 e 5, não se notando melhoria do desempenho com mais valores.

Definem-se novas variáveis:

$$A_k(s) = \ln \alpha_k(s)$$
$$B_k(s) = \ln \beta_k(s)$$
$$\Gamma_k(s', s) = \ln \gamma_k(s', s)$$

Aplicando logaritmos a ambos os membros da Eq. (9) obtemos

$$\Gamma_k(s', s) = \ln C_k + \frac{u_k L(u_k)}{2} + \frac{L_c}{2} \sum_{l=1}^{n} x_{kl} y_{kl} \ .$$

A parcela $\ln C_k$ não vai ter importância no cálculo de $L(u_k|\mathbf{y})$.

A partir das equações recursivas de α e β obtemos

$$\begin{aligned} A_k(s) &= \ln\left[\sum_{s'} \alpha_{k-1}(s')\gamma_k(s', s)\right] = \\ &= \ln\left\{\sum_{s'} \exp\left[A_{k-1}(s') + \Gamma_k(s', s)\right]\right\} = \\ &= \max_{s'}{}^{*}\left[A_{k-1}(s') + \Gamma_k(s', s)\right] \end{aligned} \tag{30}$$

$$\begin{aligned} B_{k-1}(s') &= \ln\left[\sum_{s} \beta_k(s)\gamma_k(s', s)\right] = \\ &= \ln\left\{\sum_{s} \exp\left[B_k(s) + \Gamma_k(s', s)\right]\right\} = \\ &= \max_{s}{}^{*}\left[B_k(s) + \Gamma_k(s', s)\right] \end{aligned} \tag{31}$$

Note-se que com códigos binários cada um dos somatórios das Eqs. (30) e (31) tem apenas duas parcelas. Analisando o que se passa com $A_k(s)$ vemos que dos dois ramos que chegam a cada estado s o algoritmo max-log-MAP escolhe aquele para o qual a soma $A_{k-1}(s') + \Gamma_k(s', s)$ é maior, ou seja, tal como no algoritmo de Viterbi há um ramo sobrevivente e um que é desprezado. O mesmo se passa com $B_{k-1}(s')$.

Os valores iniciais de A e B em treliças terminadas são os seguintes:

$$A_0(s) = \begin{cases} 0 & s = 0 \\ -\infty & s \neq 0 \end{cases} \qquad\qquad B_N(s) = \begin{cases} 0 & s = 0 \\ -\infty & s \neq 0 \end{cases}$$

Os valores $A_{k-1}(s) + \Gamma_k(s', s)$ e $B_k(s) + \Gamma_k(s', s)$ podem ser interpretados como uma métrica de ramo: no caso de $A_k(s)$ a métrica calcula-se do princípio para o fim da treliça, como no algoritmo de Viterbi; no caso de $B_k(s)$ calcula-se do fim para o princípio. Ou seja, de facto o algoritmo max-log-MAP funciona como dois algoritmos de Viterbi, um a percorrer a treliça no sentido habitual e outro a percorrê-la no sentido oposto.

Da Eq. (5) obtemos imediatamente

$$\ln P(s',s,\mathbf{y}) = \ln\left[\alpha_{k-1}(s')\gamma_k(s',s)\beta_k(s)\right] = \\ = A_{k-1}(s') + \Gamma_k(s',s) + B_k(s)$$

A LLR condicional da Eq. (4) passa então a ser igual a

$$L(u_k|\mathbf{y}) = \ln \frac{\sum_{R_1} P(s',s,\mathbf{y})}{\sum_{R_0} P(s',s,\mathbf{y})} = \ln \frac{\sum_{R_1} \exp\left[A_{k-1}(s') + \Gamma_k(s',s) + B_k(s)\right]}{\sum_{R_0} \exp\left[A_{k-1}(s') + \Gamma_k(s',s) + B_k(s)\right]} = \\ = \max_{R_1}{}^{*}\left[A_{k-1}(s') + \Gamma_k(s',s) + B_k(s)\right] - \max_{R_0}{}^{*}\left[A_{k-1}(s') + \Gamma_k(s',s) + B_k(s)\right] \tag{32}$$

Refira-se que no algoritmo max-log-MAP só um dos ramos de cada conjunto R_0 e R_1 vai contribuir para o cálculo de $L(u_k|\mathbf{y})$.

Como calcular $L(u_k|\mathbf{y})$ no algoritmo log-MAP sabendo nós que o logaritmo jacobiano se aplica à soma de duas exponenciais e aqui temos mais de duas? O processo de cálculo é recursivo e descrito de acordo com Robertson et al. [5]. Assim, considere-se a função $z = \ln\left(e^{x_1} + e^{x_2} + \ldots + e^{x_n}\right)$ e efectuemos os seguintes passos:

1) $z = \ln\left(\underbrace{e^{x_1} + e^{x_2}}_{e^{y_1}} + \ldots + e^{x_n}\right) \Rightarrow e^{y_1} = e^{x_1} + e^{x_2} \Rightarrow y_1 = \ln(e^{x_1} + e^{x_2}) = \max(x_1, x_2) + \ln\left(1 + e^{-|x_1 - x_2|}\right)$

2) $z = \ln\left(\underbrace{e^{y_1} + e^{x_3}}_{e^{y_2}} + \ldots + e^{x_n}\right) \Rightarrow e^{y_2} = e^{y_1} + e^{x_3} \Rightarrow y_2 = \ln(e^{y_1} + e^{x_3}) = \max(y_1, x_3) + \ln\left(1 + e^{-|y_1 - x_3|}\right)$

$\vdots$

n-1) $z = \ln\left(e^{y_{n-2}} + e^{x_n}\right) = \max(y_{n-2}, x_n) + \ln\left(1 + e^{-|y_{n-2} - x_n|}\right)$

Transportando estes desenvolvimentos para a função max* seria fácil concluir que também esta pode ser calculada recursivamente se tiver mais de duas variáveis independentes [8]. Por exemplo, $\max^*(x_1,x_2,x_3) = \max^*[\max^*(x_1,x_2),x_3]$.

9 Conclusão e agradecimentos

Este é mais um artigo sobre descodificação MAP. No entanto, apesar de haver uma imensidão de artigos sobre o assunto – quem descrer que proceda a uma pesquisa no Google… – o autor espera que a sua abordagem torne a matéria um pouco mais fácil de entender. No fim de contas, os conceitos necessários à compreensão dos algoritmos não são assim tão difíceis de apreender, de facto.

Este trabalho foi realizado no "Information and Telecommunication Technology Center" (ITTC) da Universidade do Kansas, em Lawrence, EUA. O autor agradece ao ITTC, à Fundação Calouste Gulbenkian e à Fundação para a Ciência e Tecnologia (ambas de Lisboa), sem cujo apoio a sua estadia em Lawrence não teria sido possível.

10 Referências

[1] L. R. Bahl, J. Cocke, F. Jelinek e J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate", *IEEE Trans. on Information Theory*, pp. 284-287, Março de 1974.

[2] J. Hagenauer e P. Hoeher, “A Viterbi Algorithm With Soft-Decision Outputs and Its Applications”, *Proceedings of GLOBECOM ’89*, Dallas, Texas, pp. 47.1.1-47.1.7, Novembro de 1989.

[3] W. Koch e A. Baier, “Optimum and sub-optimum detection of coded data disturbed by time-varying inter-symbol interference,” *IEEE Globecom*, pp. 1679–1684, Dezembro de 1990.

[4] J. A. Erfanian, S. Pasupathy e G. Gulak, “Reduced complexity symbol detectors with parallel structures for ISI channels,” *IEEE Trans. Communications*, vol. 42, pp. 1661–1671, 1994.

[5] P. Robertson, E. Villebrun e P. Hoeher, “A comparison of optimal and sub-optimal MAP decoding algorithms operating in the log domain”, *Proc. Intern. Conf. Communications (ICC)*, pp. 1009–1013, Junho de 1995.

[6] C. Berrou, A. Glavieux e P. Thitimajshima, “Near Shannon limit error-correcting coding and decoding: Turbo codes”, *Proc. Intern. Conf. Communications (ICC)*, pp. 1064–1070, Maio de 1993.

[7] S. Benedetto, D. Divsalar, G. Montorsi e F. Pollara, “Soft-Output Decoding Algorithms in Iterative Decoding of Turbo Codes”, *The Telecommunications and Data Acquisition Progress Report 42-124*, Jet Propulsion Laboratory, Pasadena, California, pp. 63-87, 15 de Fevereiro de 1996.

[8] A. J. Viterbi, “An Intuitive Justification and a Simplified Implementation of the MAP Decoder for Convolutional Codes”, *IEEE Journal on Selected Areas in Communications*, Vol. 16, no. 2, pp. 260-264, Fevereiro de 1998.