EET-49 Comunicações II

Parte 10 - Códigos convolucionais

May 30, 2012

1 Códigos convolucionais

- Códigos convolucionais são diferentes de códigos de bloco pois geram a sequência-código (e não palavra-código) se forma sequencial e continuamente[1].
- A sequência é transmitida na medida em que ela é gerada. Não é necessário aguardar que todos os bits de entrada sejam recebidos para definirmos qual serão os bits de saída. A cada instante de tempo, o codificador gera n bits de saída a partir de k bits de entrada. Logo, a taxa é k/n.
- Se a cada instante de tempo os k bits de informação fornecidos aparecem *in natura* na saída do codificador, o código é sistemático.
- Códigos convolucionais são semelhantes aos códigos cíclicos no sentido de que a saída é obtida através da operação entre um polinômio de entrada e um polinômio fixo (no caso de códigos cíclicos, o polinômio gerador). A diferença reside no fato de que, enquanto que os códigos cíclicos realizam o produto de um polinômio finito por outro polinômio finito, os códigos convolucionais realizam o produto entre um polinômio de entrada (que pode ser infinito) com um polinômio fixo. Esta operação pode ser vista como a convolução entre dois vetores.
- Matematicamente, isto equivale a dizer que $\mathbf{v} = \mathbf{u} * \mathbf{g}$. Os valores do vetor $\mathbf{v}$ são calculados através da seguinte expressão:

$$v_l = \sum_{i=0}^{m} u_{l-i} g_i = u_l g_0 + u_{l-1} g_1 + u_{l-2} g_2 + \cdots + u_{l-m+1} g_{m-1} + u_{l-m} g_m \tag{1}$$

- O vetor $\mathbf{g}$ define a resposta ao impulso do código e é chamado de sequência geradora.
- É possível que haja mais de uma saída. Neste caso,

$$\begin{aligned} \mathbf{v}^0 &= \mathbf{u} * \mathbf{g}^0 \\ \mathbf{v}^i &= \mathbf{u} * \mathbf{g}^1 \\ &\vdots \\ \mathbf{v}^{n-1} &= \mathbf{u} * \mathbf{g}^{n-1} \end{aligned} \tag{2}$$

  onde o superscrito é o índice da saída. Neste caso, a sequência de saída pode ser interpretada como:

$$\mathbf{v} = \left[v_0^0 v_0^1 v_0^2 \cdots v_0^{n-2} v_0^{n-1} v_1^0 v_1^1 v_1^2 \cdots v_1^{n-2} v_1^{n-1} v_2^0 v_2^1 v_2^2 \cdots v_2^{n-2} v_2^{n-1} \cdots \right] \tag{3}$$

- Exemplo: Seja $\mathbf{u} = [100010000\cdots]$, $\mathbf{g}^0 = [111]$ e $\mathbf{g}^1 = [101]$. Neste caso teríamos $\mathbf{v}^0 = [11101110\cdots]$ e $\mathbf{v}^1 = [10101010\cdots]$, resultando no vetor $\mathbf{v} = [11 \cdot 10 \cdot 11 \cdot 00 \cdot 11 \cdot 10 \cdot 11 \cdot 00 \cdots]$. Os pontos entre os pares de bits são apenas para visualização.

[1] É possível interpretar códigos convolucionais como códigos de bloco, se os primeiros forem por projeto finitos

- Equivalente à codificação convolucional é a utilização de uma matriz geradora infinita. Havendo uma saída teríamos a matriz geradora $\mathbf{G}$. Havendo duas saídas teríamos a matriz geradora $\mathbf{G}'$:

$$\mathbf{G} = \begin{bmatrix} g_0 & g_1 & g_2 & g_3 & \cdots & g_m & 0 & 0 & \cdots \\ 0 & g_0 & g_1 & g_2 & \cdots & g_{m-1} & g_m & 0 & \cdots \\ 0 & 0 & g_0 & g_1 & \cdots & g_{m-2} & g_{m-1} & g_m & \cdots \\ \ddots & & & & \ddots & & & & \ddots \end{bmatrix}$$

$$\mathbf{G}' = \begin{bmatrix} g_0^0 & g_0^1 & g_1^0 & g_1^1 & g_2^0 & g_2^1 & \cdots & g_m^0 & g_m^1 & 0 & 0 & 0 & 0 & \cdots \\ 0 & 0 & g_0^0 & g_0^1 & g_1^0 & g_1^1 & \cdots & g_{m-2}^1 & g_{m-1}^0 & g_{m-1}^1 & g_m^0 & g_m^1 & 0 & \cdots \\ 0 & 0 & 0 & 0 & g_0^0 & g_0^1 & \cdots & g_{m-2}^0 & g_{m-2}^1 & g_{m-1}^0 & g_{m-1}^1 & g_m^0 & g_m^1 & \cdots \\ \ddots & & & & & & \ddots & & & & & & & \ddots \end{bmatrix}$$

- Assim como a multiplicação e a divisão, a convolução pode ser facilmente implementada através de circuitos lógicos.

- Exemplo(cont.): A estrutura que realiza a codificação é mostrada na figura 1. Veja que há duas saídas e uma chave que alterna entre elas.

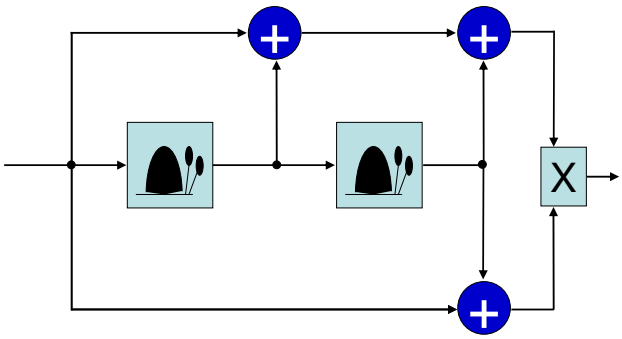

Figure 1: Codificador convolucional simples.

- As operações não precisam ser necessariamente sobre uma única entrada. As possibilidades podem ser tornar cada vez mais complicadas ao permitir por exemplo sequências (de memória) paralelas, com comprimento (grau) diferentes, etc., com mostra a figura 2, que tem 2 entradas e 3 saídas.

- Independentemente da complexidade do codificador, as saídas são obtidas através das combinações lineares das entradas. Logo, os códigos convolucionais são lineares: se $\mathbf{v}^1$ e $\mathbf{v}^2$ são sequências-código, $\mathbf{v}^3 = \mathbf{v}^1 + \mathbf{v}^2$ também será. O conjunto de sequências geradsa por um codificador com esta estrutura forma o código convolucional.

- Para caracterizar os codificadores, dois números são importantes: a quantidade de memórias e a ordem do codificador:

  – A quantidade de memórias m esta relacionada com a complexidade de decodificação, como será visto posteriormente.

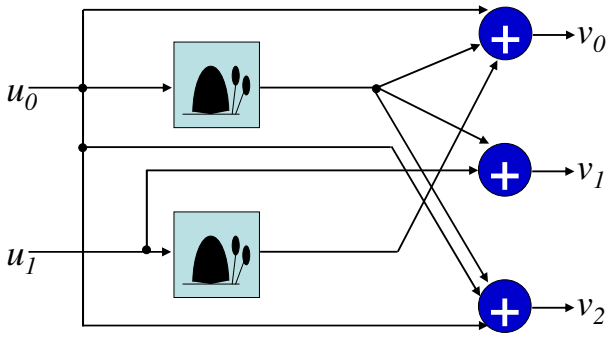

Figure 2: Codificador convolucional com 2 entradas e 3 saídas.

  - A maior cadeia de registradores de deslocamento lineares é o tamanho de restrição (*constraint lenght*). Este número indica até quantos bits de entrada passados estão sendo utilizados para gerar o bit de saída atual.

- Como visto pelos exemplos anteriores, a implementação da convolução vetorial é semelhante à implementação de códigos cíclicos, pois a convolução vetorial entre **u** e **g** é equivalente ao produto entre $u(D)$ e $g(D)$. A diferença é que neste caso o polinômio $u(D)$ pode ser uma sequência infinita. O polinômio com maior grau define a ordem do codificador.

- Por ter grau possivelmente infinito, não é possível utilizar estruturas que realizam a multiplicação começando pelos termos associados a graus menores. Por isso, utiliza-se a estrutura onde os **termos associados a graus menores são alimentados e gerados antes**.

- Ao utilizar polinômios, a multiplexação necessária para se gerar $v(D)$ pode ser facilmente tratada matematicamente através das seguintes equações:

$$\begin{aligned} v^i(D) &= u(D)g^i(D) \\ v(D) &= \textstyle\sum_{i=0}^{n-1} v^i(D^n)D^i \end{aligned} \tag{4}$$

  onde o termo D^n (argumento de $v^i(D^n)$)espalha os vetores de saída de forma que somente 1 em cada N termos seja a saída correta e o termo externo D^i desloca as sequências de saída entre si para que elas não se sobreponham.

- De forma genérica, podemos descrever os códigos convolucionais através da matriz de polinômios(finita):

$$\mathbf{G}(D) = \begin{bmatrix} g_0^0(D) & g_0^1(D) & g_0^2(D) & \cdots & g_0^{n-2}(D) & g_0^{n-1}(D) \\ g_1^0(D) & g_1^1(D) & g_1^2(D) & \cdots & g_1^{n-2}(D) & g_1^{n-1}(D) \\ g_2^0(D) & g_2^1(D) & g_2^2(D) & \cdots & g_2^{n-2}(D) & g_2^{n-1}(D) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ g_{k-2}^0(D) & g_{k-2}^1(D) & g_{k-2}^2(D) & \cdots & g_{k-2}^{n-2}(D) & g_{k-2}^{n-1}(D) \\ g_{k-1}^0(D) & g_{k-1}^1(D) & g_{k-1}^2(D) & \cdots & g_{k-1}^{n-2}(D) & g_{k-1}^{n-1}(D) \end{bmatrix}$$

  onde o termo $g_i^j(D)$ é o polinômio que diz qual é a relação entre a i-ésima entrada com a j-ésima saída.

- Pode haver possivelmente mais de um codificador para uma mesma matriz $\mathbf{G}(D)$. A complexidade de decodificação depende do número de memórias utilizadas, como será visto posteriormente. Logo, seria bom saber qual é a implementação de $\mathbf{G}(D)$ com o menor número de memórias. O codificador com esta característica é chamado de codificador mínimo.

- O codificador mínimo pode ser obtido através da análise do diagrama de estados.

- Exemplo: a matriz de polinômios associada ao codificador da figura 2 é igual a :

$$\mathbf{G}(D) = \begin{bmatrix} 1+D & D & 1+D \\ D & 1 & 1 \end{bmatrix}$$

  Se $u(D) = \begin{bmatrix} 1+D & D \end{bmatrix}$, a saída $v(D)$ seria igual a

$$\begin{aligned} v(D) = & \quad u(D)\mathbf{G}(D) \\ = & \quad \begin{bmatrix} (1+D)(1+D)+D\cdot D & (1+D)D+D & (1+D)(1+D)+D \end{bmatrix} \\ = & \quad \begin{bmatrix} 1+D+D+D^2+D^2 & D+D^2+D & 1+D+D+D^2+D \end{bmatrix} \\ = & \quad \begin{bmatrix} 1 & D^2+1+D^2 \end{bmatrix} \end{aligned} \tag{5}$$

- Códigos convolucionais também podem ser sistemáticos, isto é, com os bits de informação compondo de forma ordenada os bits de saída. Como os bits são gerados sequencialmente, códigos convolucionais sistemáticos fazem com que as primeiras k saídas (do total de n) sejam iguais as entradas. Matematicamente, isto implicaria em dizer que a matriz geradora teria o seguinte formato:

$$\mathbf{G}(D)_{sis} = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & g_0^k(D) & g_0^{k+1}(D) & \cdots & g_0^{n-1}(D) \\ 0 & 1 & 0 & \cdots & 0 & g_1^k(D) & g_1^{k+1}(D) & \cdots & g_1^{n-1}(D) \\ 0 & 0 & 1 & \cdots & 0 & g_2^k(D) & g_2^{k+1}(D) & \cdots & g_2^{n-1}(D) \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & g_{k-1}^k(D) & g_{k-1}^{k+1}(D) & \cdots & g_{k-1}^{n-1}(D) \end{bmatrix}$$

- Algumas matrizes $\mathbf{G}(D)$ podem ser postas na forma sistemática através da recombinação de linhas de forma semelhante como é feito para códigos de bloco. Entretanto, além de somar linhas, podemos multiplicar uma linha ou a matriz inteira por algum termo linear, como por exemplo um polinômio, antes de recombina-las.

- Operações de recombinação de linhas podem fazer com que os fatores $g_i^j(D)$ sejam escritos no formato $a(D)/b(D)$, onde $a(D)$ e $b(D)$ são polinômios. Neste caso a resposta ao impulso não é necessariamente finita.

- Exemplo: Seja a matriz

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

  Podemos obter $\mathbf{G}'(D)$ como:

$$\mathbf{G}'(D) = \mathbf{G}(D)\frac{1}{1+D+D^2} = \begin{bmatrix} 1 & \dfrac{1+D^2}{1+D+D^2} & \dfrac{1+D}{1+D+D^2} \end{bmatrix}$$

  As duas matrizes geram o mesmo código convolucional. Se a saída $v(D)$ é gerada por $u(D)$ através do produto $u(D)\cdot\mathbf{G}(D)$, ela também seria gerada pelo produto de $u'(D)\cdot\mathbf{G}'(D)$ fazendo com que $u'(D) = (1+D+D^2)\mathbf{G}'(D)$.

- Como os primeiros termos a entrar no codificador são aqueles associados a graus menores de $u(D)$, a divisão a ser feita é a divisão longa.

- Na divisão tradicional, ao dividirmos $a(D) = D^w + D^x + \cdots + 1$ por $b(D) = D^y + D^z + \cdots + 1$ ($w > x$, $y > z$, $y \leq w$), multiplicamos $b(D)$ por D^{w-y} e subtraímos este produto de $a(D)$. O procedimento é repetido com o resultado desta subtração até que o grau do resto seja menor do que o grau de $b(D)$.
- Na divisão longa, ao dividirmos o polinômio $a(D) = D^w + D^x + \cdots$ por $b(D) = D^y + D^z$ ($w < x$, $y < z$, $y \leq w$), multiplicamos $b(D)$ por D^{w-y} e subtraímos este produto de $a(D)$. O procedimento é repetido até que o resto seja igual a zero. Devido às restrições da divisão ($w < x$, $y < z$, $y \leq w$), o processo pode ser infinito. O resultado da divisão longa de 1 por $1 + D + D^2$ resulta em $1 + D + D^3 + D^4 + D^6 + D^7 + D^9 + \cdots$.

- A divisão longa de $a(D)$ por $b(D) = 1 + b_1 D + b_2 D^2 + \cdots + b_m D^m$ pode ser feita pelo circuito da figura 3.

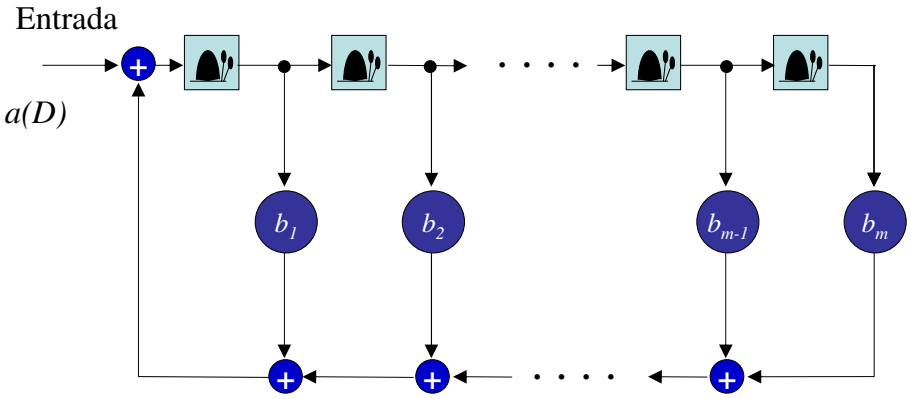

Figure 3: Circuito que implementa a divisão longa.

- O termo $a(D)/b(D)$ pode ser decomposto na forma $a_0/b(D) + a_1 D/b(D) + \cdots + a_l D^l b(D)$. Assim, ele pode ser calculado através da combinação linear do resultado da divisão $1/b(D)$ com os seus atrasos. Desta forma, o circuito que realiza a codificação do exemplo anterior por ser desenhado como mostrado na figura 4

2 Representações equivalentes

- Os codificadores podem ser representados de algumas formas:
  - Diagrama de estados
  - Diagrama de estados modificado
  - Seção de trelica

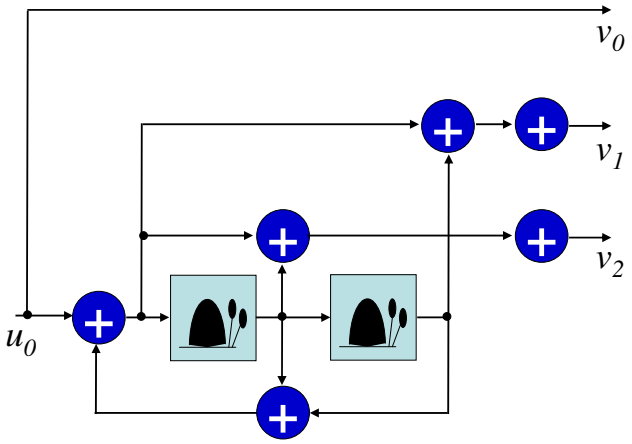

Figure 4: Codificador recursivo.

- As três formas são descritas a seguir e mostradas na figura 5 para a matriz $\mathbf{G}'(D)$ do exemplo anterior.

2.1 Diagrama de estados

- Se o codificador possui m memórias binárias (registradores de deslocamento), há então 2^m estados possíveis
- Na medida em que os bits de informação são fornecidos para o sistema, ocorrem transições entre estados.
- Desta forma, é possível descrever o código convolucional através de um diagrama de estados que contém:
  - Nós (círculos) indicando todos os estados possíveis
  - Ramos que indicam quais transições podem ocorrer
  - Rótulos de ramos que indicam qual entrada causa aquela transição e qual saída ocorre quando aquela transição ocorre, dado o estado inicial
- Se há k entradas, cada estado possui no máximo 2^k ramos de saída. É possível que duas entradas causem a mesma transição de estados. Nesta situação, os rótulos dos dois ramos paralelos serão diferentes.
- Dois estados são equivalentes e eles são equivalentes no que se refere aos ramos (e rótulos associados) de entrada e saída. Neste caso eles podem ser unidos e considerados um único estado.
- Se existe um ciclo (caminho que sai de um estado e retorna ao mesmo) que possui peso de Hamming de saída igual a zero mas peso de Hamming de entrada diferente de zero, o código é catastrófico. Isto quer dizer que um número finito de erros de recepção pode causar um número infinito de erros de decodificação.

2.2 Diagrama de estados modificado

- O diagrama de estados modificado pode ser obtido através do diagrama de estados.
- Para obte-lo, basta, a partir do diagrama de estados:
  - Separar o estado nulo em dois: um onde só há ramos de saída e outro onde só há ramos de chegada
  - Eliminar a transição do estado nulo para o estado nulo.

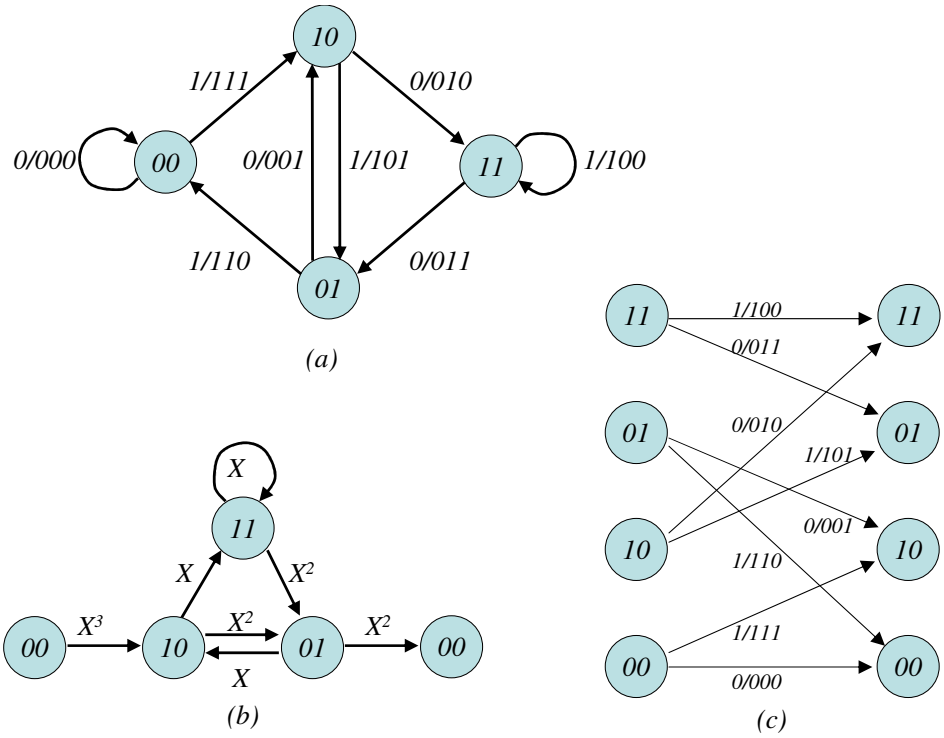

Figure 5: Representações para um código convolucional:(a)-Diagrama de estados. (b)-Diagrama de estados modificado. (c)-Seção de treliça.

- Os ramos podem ser alternativamente rotulados pelo valor X^h, onde X é uma variável fantasma e h indica o peso de Hamming de saída do ramo. Isto permite o cálculo da função de transferência do código, que será vista posteriormente. A figura mostrada

2.3 Seção de treliça

- A seção de treliça é obtida representando os estados em dois instantes: no instante t e no instante $t+1$.
- Os estados são conectados por ramos que saem dos estados pertencentes ao instante t e chegam aos estados pertencentes ao instante $t+1$
- Os ramos podem ser rotulados pelos bits de entrada, pelos bits (ou símbolos) de saída ou pela combinação destes.
- A seção de treliça não é necessariamente a mesma para todos os instantes t. Em particular, se no início da codificação ($t=0$) o codificador encontra-se no estado nulo, é possível que nem todos os estados presentes no diagrama de estados existam no instante $t=1$.
- A concatenação de diversas seções de treliça fornece a treliça do codificador, que mostra todas as sequências de estados e de saída possíveis.
- A treliça é utilizada para realizar a decodificação pelo algoritmo de Viterbi.

3 Decodificação: Algoritmo de Viterbi

- O algoritmo de Viterbi usa a treliça do código para realizar a decodificação.
- O algoritmo armazena métricas de percurso (custo) para tomar a decisão sobre qual sequência foi transmitida. A métrica do percurso é obtida através da soma das métricas dos ramos que, concatenados, formam o percurso.
- Há algumas possibilidades para a métrica:
  - No caso de transmissão através um canal BSC, pode-se utilizar a distância de Hamming entre o que foi recebido e os rótulos binários de saída dos ramos.
  - No caso de transmissão através de um canal Gaussiano, pode-se utilizar a distância Euclidiana quadrática entre o símbolo recebido e os símbolos associados aos ramos de saída.
- A cada instante, o número de percursos possíveis cresce exponencialmente. Se, por exemplo, o codificador tiver k entradas, n saídas e m memórias, há 2^{kt} sequências possíveis do instante 0 até o instante t.
- Por outro lado, somente um pequeno conjunto de sequências são as mais prováveis. O algoritmo de Viterbi aproveita-se deste fato para reduzir a complexidade de decodificação. Em um dado instante, há vários percursos que levam ao mesmo estado σ_i^t. Dentre os vários possíveis, um deles será mais provável do que os outros. Logo, se o percurso mais provável passa pelo estado σ_i no instante t, o caminho que levou até este estado deve ser o mais provável. Assim, em cada instante, é necessário armazenar, pra cada estado, somente o caminho mais provável de chegar até ele. Isto é, precisamos armazenar 2^m caminhos, o que é uma redução de complexidade se comparado com o número de sequências possíveis(2^{kt}).
- O algoritmo de Viterbi pode ser descrito em palavras desta forma:
  1. Inicie todos os estados com custo igual a zero.
  2. Dado o símbolo recebido, calcule o custo de cada uma das transições possíveis no instante atual[2]

[2]No instante inicial, por exemplo, o único estado possível é o nulo. Logo, todas as transições devem sair deste estado.

3. Para cada estado futuro, calcule o custo de todos os percursos que chegam ao estado somando o custo do estado anterior com o custo do ramo que causa a transição entre estados. O caminho sobrevivente para um estado futuro é aquele com menor custo. Este custo torna-se também o custo do estado.
4. Enquanto houver símbolos a serem processados, retorne ao passo 2.
5. Escolha o caminho que tiver menor custo, obedecendo a restrições sobre o estado final, se houver[3].

- O algoritmo de Viterbi também pode ser descrito matematicamente. Para isto precisamos apresentar algumas variáveis:
  - σ_i^k: i-ésimo estado no instante k, $i = 0, 1, 2, ..., 2^m - 1$ e $k = 0, 1, 2, ..., K$
  - $C(\sigma_i^k)$: custo do i-ésimo estado no instante k
  - $\rho_{i,j}^k$: ramo que causa a transição de σ_i^k para σ_j^{k+1}
  - $C(\rho_{i,j}^k)$: custo do ramo que causa a transição de σ_i^k para σ_j^{t+1}[4].
  - $s(\rho_{i,j}^k)$: símbolo de entrada associado ao ramo $\rho_{i,j}^k$.
  - $t(\rho_{i,j}^k)$: símbolo de saída (transmitido) associado ao ramo $\rho_{i,j}^k$.
  - r^k símbolo recebido no k-ésimo instante.
  - $\mathbf{s}_i^k = [s_0, s_1, s_2, ..., s_{k-1}]$: Sequência de símbolos que formam o caminho sobrevivente que chega ao estado σ_i^k

- Desta forma, o algoritmo de Viterbi é:

1. Iniciação de variáveis:
$$\begin{gathered} C(\sigma_0^0) = 0 \\ C(\sigma_i^0) = \infty, i = 1, 2, 3, ..., 2^m - 1 \\ k = 0 \\ \mathbf{s}_0^0 = \emptyset \end{gathered} \tag{6}$$
2. Cálculo dos custos dos ramos:
$$C(\rho_{i,j}^k) = d(r^k, t(\rho_{i,j}^k)) \tag{7}$$
3. Cálculo dos custos dos estados futuros e seleção de caminho sobrevivente.
$$\begin{aligned} i(j)* = &\quad \arg\min_i \left\{ C(\sigma_i^k) + C(\rho_{i,j}^k) \right\} \\ C(\sigma_j^{k+1}) = &\quad C(\sigma_{i(j)*}^k) + C(\rho_{i(j)*,j}^k) \\ \mathbf{s}_j^{k+1} = &\quad [\mathbf{s}_{i(j)*}^k \quad s(\rho_{i(j)*,j}^k)] \end{aligned} \tag{8}$$
onde a última linha indica a concatenação do vetor $\mathbf{s}_{i(j)*}^k$ com o símbolo $s(\rho_{i(j)*,j}^k)$.
4. Se $k < K$, $k = k + 1$. Retorne ao passo 2.
5. $\mathbf{s}* = \max_i \left\{ \mathbf{s}_i^K \right\}$

- A figura 6 mostra a evolução do algoritmo de Viterbi quando a sequência $000101100000\cdots$ é recebida, utilizando o codificador da figura 4. Na decodificação assumiu-se que a sequência $111101110000\cdots$ foi transmitida, correspondendo aos bits de informação $111000\cdots$. Note que, com o aumento de k, todos os outros caminhos continuarão a ter os seus pesos aumentados, enquanto que o caminho que prossegue com zeros manterá o seu peso igual a 3.

- Um problema do algoritmo de Viterbi é que, para tomar a decisão, precisamos processar toda a sequência recebida, o que pode levar um tempo muito grande.

[3] É possível por projeto garantir que o último estado seja igual ao estado nulo, por exemplo

[4] Pode haver mais de um ramo que causa a mesma transição entre estados. Neste caso haverá transições paralelas. Por simplicidade de notação omitimos esta situação

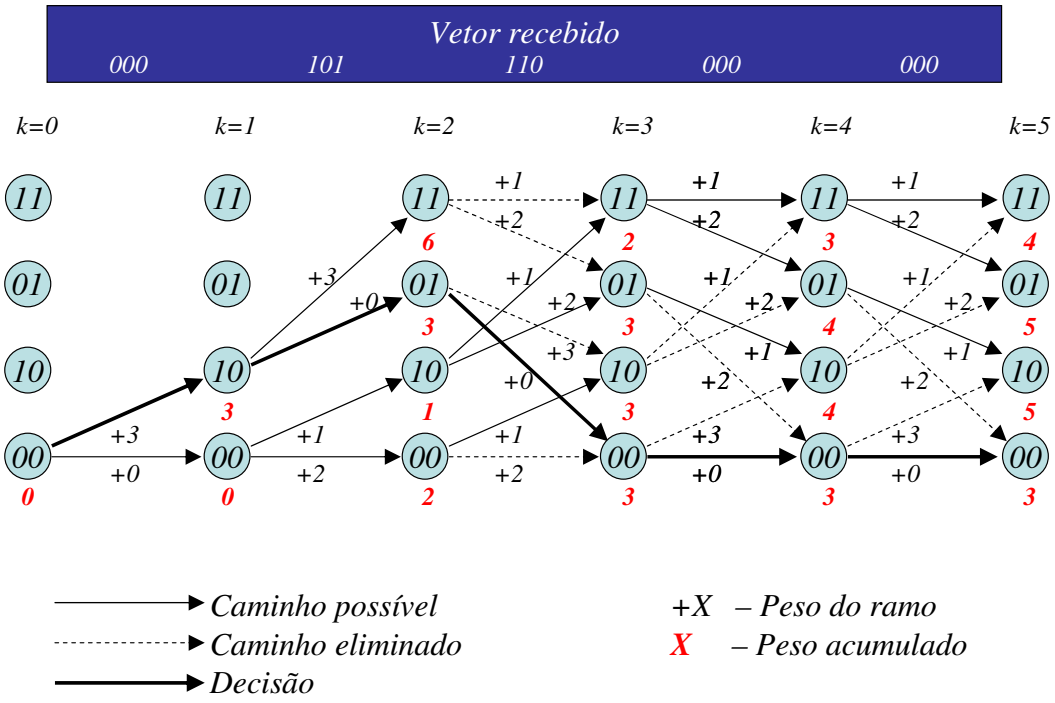

Figure 6: Execução do algoritmo de Viterbi para o exemplo.

- Alternativamente, pode-se utilizar uma variante sub-ótima onde observa-se os sinais recebidos entre os instantes k e $k+T$ para se tomar a decisão pelo símbolo que foi transmitido no instante k. Isto é, após T instantes, a sequência mais provável dirá qual foi o símbolo transmitido no instante T.
- O valor de T pode ser escolhido por simulação.
- É possível que, ao utilizar o método sub-ótimo, a decisão final seja uma sequência de ramos que não seja possível. Isto pode acontecer por exemplo se a sequência vencedora no instante $k+T$ exija que o estado no instante $k+1$ seja diferente do estado no instante $k+1$ da sequência vencedora no instante $k+T+1$.

4 Análise de desempenho

- Como não há um tamanho pré-definido para a sequência transmitida, a capacidade de correção de erro não pode ser definida da mesma forma como para códigos de bloco.
- Para que um erro ocorra, em algum momento o caminho com menor métrica deve ser diferente do caminho correto. Neste momento a decisão divergirá do caminho correto. Por outro lado, a não ser que o código seja catastrófico, a continuação da decodificação pelo caminho errado implicará no aumento constante do seu custo. Eventualmente o algoritmo retornará ao caminho correto.
- O evento em que a decisão desvia do caminho correto e retorna ao mesmo posteriormente é chamado de evento de erro. O evento de erro faz com que o caminho errado tenha métrica menor entre os instantes A e B. Para baixas probabilidades de erro, é baixa a probabilidade de dois eventos de erro ocorrerem muito próximos.
- Um evento de erro ocorre quando, dada a sequência recebida, o custo do desvio errado é menor do que o custo do caminho correto. Utilizando a métrica binária, isto implica que a distância de Hamming entre a sequência recebida e a sequência correta é maior do que a distância de Hamming entre a sequência recebida e alguma outra sequência possível. Logo, a capacidade de correção de erro depende da menor distância de Hamming entre duas sequências quaisquer.
- Considerando códigos convolucionais lineares, podemos substituir a análise das distâncias entre sequências pela análise dos pesos de Hamming das sequências, de forma semelhante como foi feita com códigos de bloco. Isto é, podemos substituir as análises dos eventos de erro pelas análises dos pesos de sequências que saem e retornam ao estado nulo (que são os eventos de erro possíveis quando a sequência toda nula é transmitida).
- A capacidade de correção do código convolucional é então definida pela menor peso de Hamming de uma sequência que sai e retorna ao estado nulo, definido como d_{min}. Qualquer evento de erro que tenha menos do que a metade deste valor pode ser corrigido.
- Uma interpretação da substituição das distâncias de Hamming pelos pesos de Hamming é a seguinte: imagine que as sequências são pontos num espaço K-dimensional. Além disso, há uma uniformidade geométrica da disposição dos pontos neste espaço de forma que qualquer ponto tem vizinhos distanciados de si de forma semelhante. Assim, para descobrir a menor distância entre pontos, podemos a distância dos vizinhos de um ponto qualquer. O ponto mais conveniente é aquele localizado na origem (sequência toda nula), pois desta forma podemos simplesmente calcular a distância dos pontos da origem, ou seja, o seu peso.
- Além da distância, também é relevante saber quantos pontos estão localizados a esta distância. Por exemplo, se um ponto tem somente um vizinho a uma distância d e outro tem 999 pontos a uma distância d, é mais provável que a probabilidade de erro seja maior ao se transmitir o segundo ponto, pois há 999 (ao invés de 1 no primeiro caso) erros com peso maior que $d/2$ que fazem com que a decisão seja errada.
- Uma forma de calcular o peso e quantidades de sequências possíveis para um código podemos utilizar o diagrama de estados modificado e a regra de Mason. Isto permite calcular a função de transferência $T(X)$.

- A função de transferência de um código convolucional é um polinômio na seguinte forma:

$$T(X) = \sum_{i=0}^{\infty} A_i X^i$$

onde X é uma variável fantasma e A_i é um inteiro que diz quantas sequências com peso i saem e retornam ao estado todo nulo.

- Para calcular a função de transferência precisamos saber:
  - Os ganhos F_j de todos os caminhos diretos do estado nulo para o estado nulo, em função de X^i.
  - Os ganhos de todos os ciclos C_k existentes no diagrama de estados modificados
- A função de trasnferência pode ser calculada através da seguinte equação:

$$T(X) = \frac{\sum_i F_j \Delta_j}{\Delta}$$

onde o somatório é feito para todos os caminhos diretos

- O valor de Δ é obtido através da expressão:

$$\Delta = 1 - \sum_k C_k + \sum_{k,l} C_k C_l - \sum_{k,l,m} C_k C_l C_m + \cdots$$

onde o primeiro somatório é para todos os ciclos, o segundo somatório é para todos os pares de ciclos que não se encontram, o terceiro é sobre todos os trios de ciclos que não se encontram e assim por diante.

- O cálculo de Δ_j é semelhante ao cálculo de Δ quando retiramos todos os estados pelo qual o j-ésimo caminho direto passa, incluindo todas as conexões destes estados.
- Para o cálculo da função de transferência também utilizamos a divisão longa. Note que neste caso não estamos utilizando variáveis binárias, de forma que as operações de + e - não são equivalentes.
- Exemplo: A função de transferência do diagrama da figura 5-(b) pode ser calculada através das seguintes etapas:
  - Há dois caminhos diretos:
    - F_1 - sequência de estados $00 \to 10 \to 01 \to 00$, com ganho igual a X^7
    - F_2 - sequência de estados $00 \to 10 \to 11 \to 01 \to 00$, com ganho igual a X^8
  - Há três ciclos:
    - C_1 - sequência de estados $11 \to 11$, com ganho igual a X
    - C_2 - sequência de estados $11 \to 01 \to 10 \to 11$, com ganho igual a X^4
    - C_3 - sequência de estados $10 \to 01 \to 10$, com ganho igual a X^3
  - Os ciclos C_1 e C_3 não se tocam. Retirando o caminho F_1 resta o ciclo C_3. Retirando o caminho F_2 não resta nenhum ciclo. Desta forma, temos:
    - $\Delta = 1 - (X + X^4 + X^3) + (X \cdot X^3) = 1 - X - X^3$
    - $\Delta_1 = 1 - X$
    - $\Delta_2 = 1$
  - Logo, a função de transferência pode ser calculada através da divisão:

$$T(X) = \frac{X^7(1-X) + X^8}{1 - X - X^3} = \frac{X^7}{1 - X - X^3} = X^7 + X^8 + X^9 + 2X^{10} + X^{11} + 2X^{12} + 4X^{13} + \cdots$$

5 Bibliografia recomendada

- S. Haykin, “Communication Systems”, capítulo 10
- Shu Lin, “Error control coding : fundamentals and applications”, capítulos 11 e 12