Códigos Convolucionais

Felipe Mourad Pereira
femp1999@gmail.com

Kelven Yi Chen Du
kelvenchendu@gmail.com

I. Introdução

Em sistemas de telecomunicação, a Codificação de Canal tem sempre como sua meta detectar e corrigir eventuais erros de transmissão devidos às imperfeições de canal. Dentre os outros códigos mais utilizados, os Códigos Convolucionais se conhecem como códigos não em bloco, diferentemente dos Códigos de Hamming, Códigos Cíclicos etc. Contanto que um código convolucional (n, k, N), de n bits de código, se gera sempre a partir de uma mensagem de k bits, a mensagem codificada, além de depender da mensagem transmitida, depende também das $N-1$ mensagens anteriores. Em visto dessa propriedade, pode-se construir o codificador de Códigos Convolucionais com máquinas de estado, de forma que sempre que se conhecem o bit de entrada e o estado da máquina, obtenham-se a saída codificada e o próximo estados da máquina.

Nestas práticas laboratoriais, os alunos implementaram um codificador e alguns decodificadores para Códigos Convolucionais. Vale mencionar que para o primeiro decodificador, uma ferramenta utilizada na implementação foi o Algoritmo de Viterbi, cujo principio é, através de uma comparação entre uma sequência de sinais recebidos e as sequências possíveis, considerar a sequência que apresenta a menor distância de Hamming como a sequência emitida. Enquanto os outros decodificadores se regem por variações de Algorítimo de Viterbi, mudando o critério de decisão à maior probabilidade de ocorrência e à menor distância euclidiana.

Aqui, vale-se fazer uma explicação sucinta sobre o Algoritmo de Viterbi. Para isso vejamos a **Figura 1** [1], chamado de diagrama de treliça e, neste caso, é referente à uma máquina condificadora de 4 estados.

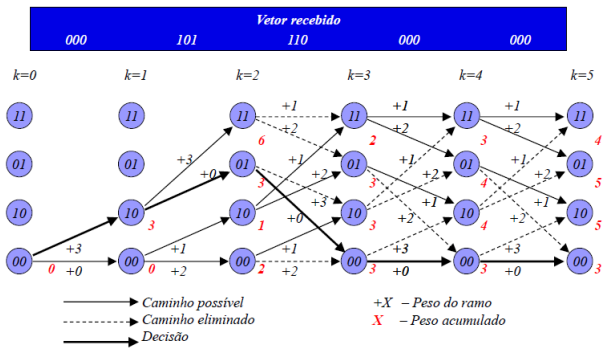

Figura 1. Diagrama de treliça.

Na **Figura 1**, temos em cada coluna quatro círculos que representam os quatro respectivos estados possíveis da máquina e temos flechas partindo de um circulo para um outro da coluna seguinte que representam transições de estados da máquina após um bit de entrada. Nota-se que a cada transição, retorna-se uma saída de três bits.

Por fim, nota-se que a cada bit de entrada temos uma saída codificada de 3 bits e esses vão ser transmitidos através de canal. Mas, por eventuais ruídos, o código recebido pode ser diferente de código de saída. Assim, o Algoritmo de Viterbi consiste em comparar a distância de Hamming acumulada entre uma sequência de sinal recebido e todas sequências de sinais possíveis de saída e, pelo fato de que normalmente a menor distância de Hamming implica a maior probabilidade de ocorrência, toma como se fosse mensagem a que corresponde a menor distância de Hamming (em caso de empate, a escolha seria arbitrária).

II. Metodologia

Inicialmente, entendeu-se o funcionamento do código convolucional, com sua máquina de estados e como funciona a transição de estados e a geração de saídas.

Assim, para cada valor de m e a tripla g_1, g_2, g_3 de polinômios, gerou-se a entrada de 10000 bits aleatórios, equiprováveis, e obteve-se a saída esperada (de 30000 bits) e uma tabela de transição de estados modelo Mealy [2], em que para um dado estado atual e uma entrada (1 bit), obtém-se uma dada saída (3 bits) e o próximo estado. Assim, para um código com m memórias, há uma tabela com 2 linhas (entrada 0 ou 1) e 2^m colunas, sendo que em cada célula desta tabela há a informação do próximo estado e da saída de 3 bits.

Após a obtenção da mensagem codificada, insere-se o ruído nesta. Dependendo do método utilizado, temos um dado ruído. Para a codificação de Viterbi a partir da distância de Hamming ou da probabilidade de ocorrência, o ruído adicionado é representado pela probabilidade de inversão de bit p. Para o método utilizando a distância euclidiana, o ruído é considerado gaussiano com uma dada variância. A relação entre a probabilidade de inversão de bit e a variância equivalente do ruído é dada pelas seguintes equações:

Ao invés de as triplas de saída serem compostas pelos bits 0 e 1, elas serão compostas pelos símbolos -1 e 1. Assim, supondo tais símbolos equiprováveis, temos que a probabilidade de inversão de bit é dada por:

$$p = \frac{1}{2} \cdot Q\left(\frac{0-(-\sqrt{E_b})}{\sqrt{N_0/2}}\right) + \frac{1}{2} \cdot Q\left(\frac{\sqrt{E_b}-(0)}{\sqrt{N_0/2}}\right) \quad (1)$$

Sendo E_b a energia do símbolo 1 ou -1. Assim:

$$p = Q\left(\sqrt{\frac{2 \cdot E_b}{N_0}}\right) = Q\left(\sqrt{\frac{2 \cdot R \cdot E_i}{N_0}}\right) \quad (2)$$

Sendo E_i a energia por bit de informação. Assim:

$$N_0/2 = \frac{E_b}{(invQ(p))^2} = \sigma_n^2 \quad (3)$$

Sendo $invQ(p)$ a função inversa de Q para um dado valor de p. Por exemplo, se $p = 0.16$, então $invQ(0.16) \approx 1$ o

que significa que $Q(1) \approx 0.16$. Assim, utilizando $E_b = 1$, temos:

$$\sigma_n^2 = \frac{1}{(invQ(p))^2} \quad (4)$$

Assim, encontramos a variância do ruído gaussiano em função da probabilidade de inversão de bit desejada p.

Após a inserção do ruído, o receptor recebe a mensagem (30000 bits, com ruído), e ele irá decodificá-la. A decodificação, como mencionado, ocorreu de acordo com o algoritmo de Viterbi explicado. Entretanto, o critério de decisão ocorreu de 3 maneiras possíveis:

1) minimizando a distância de Hamming entre a saída obtida e as possíveis saídas que a geraram, encontrando assim o melhor caminho com a menor distância de Hamming entre os 30000 bits do caminho e os 30000 bits recebidos;
2) maximizando a probabilidade de ocorrência de um dado caminho, sendo que dada uma tripla recebida $b_1b_2b_3$, uma dada tripla sugerida $b'_1b'_2b'_3$ tem probabilidade de ocorrência dada por $p^j(1-p)^k$, em que $j+k=3$, j representa a quantidade de bits invertidos e k a quantidade de bits iguais entre as triplas. Para evitar *underflow*, fez-se a normalização das probabilidades de cada caminho a cada iteração do algoritmo de Viterbi;
3) minimizando a distância euclidiana entre a saída obtida e as possíveis saídas que a geraram, encontrando assim o melhor caminho com a menor distância euclidiana entre os 30000 bits do caminho e os 30000 bits recebidos. Por exemplo, um caminho sugerido 1 1 -1 possui uma distância do caminho recebido 1.1 1.2 -0.3 dada por $\sqrt{0.1^2 + 0.2^2 + 0.3^2} \approx 0.374$.

Desta forma, após a decodificação, obtém-se o caminho mais provável de bits de entrada que ocorreram, escolhendo-o assim como a esperança da informação transmitida. Finalmente é possível encontrar a quantidade de bits errados em relação a informação que se desejava enviar, e é possível encontrar a probabilidade de erro de bit P_e.

Entretanto, percebeu-se que o resultado obtido não é muito consistente para alguns valores de p, ou seja, as vezes acerta muito bem, mas as vezes nem tão bem. Assim, achou-se ser plausível a repetição de diversos experimentos independentes, em que cada um obtém um valor de P_e, e fez-se a média de tais valores, também obtendo-se sua variância. Os gráficos expostos utilizam um total de 20 repetições para cada valor de p. É possível visualizar o desvio-padrão em cada ponto na barra de erro.

Por fim, comparou-se os resultados obtidos com os resultados dos laboratórios anteriores, mas em relação ao valor de E_i/N_0 ao invés de p, para uma comparação mais justa.

III. Resultados e discussões

A seguir, algumas perguntas sugeridas para a explicação do experimento:

1) **Quais foram as maiores dificuldades em implementar o codificador convolucional?**

   A maior dificuldade foi uma boa implementação e utilização da tabela de transição de estados. No caso, utilizou-se da técnica de *memoization*, em que, ao procurar-se por uma dada transição na tabela, se ela ainda não foi feita anteriormente, calcula-se qual é a transição correspondente e guarda-se em tal posição na tabela. Se ela já foi feita, não há o retrabalho de calcular novamente qual ela seria, e simplesmente acessa-se tal posição na tabela.

2) **Quanto tempo a sua solução demora para codificar cada bit de informação? Faça uma média.**

   Há uma razoável diferença no tempo de codificação para diferentes valores de m. Isto se deve pelo fato de o número de operações realizadas em cada codificação ser proporcional ao número de memórias. Assim, fez-se a média para $m = 3$, $m = 4$ e $m = 6$ separadamente, e então a média total. Para obter cada média, rodou-se o código 20 vezes para cada valor de p. Obtivemos:

   Tabela I
   TABELA DE TEMPO DE CODIFICAÇÃO POR M

m	tempo(us)
3	0.56 ± 0.07
4	0.60 ± 0.06
6	0.7980 ± 0.0004

   Com uma média final de 0.65 ± 0.09 us.

3) **Quais foram as maiores dificuldades em implementar o decodificador convolucional?**

   A maior dificuldade foi projetar um bom modelo de treliças, e como usá-las. No caso, utilizou-se um modelo em que em um dado passo k, tem-se uma configuração de treliça T_k representada por um vetor de configurações, com cada configuração correspondendo a um estado possível, sendo 2^m estados possíveis. Na transição de k para $k+1$, inicialmente copia-se T_k em T_{k+1}. Cada configuração da treliça foi definida por uma tripla, que contém o custo mínimo para chegar em tal estado, o passo em que ocorreu sua última atualização e o caminho das entradas que geraram esse custo mínimo. Assim, na transição de T_k para T_{k+1}, verifica-se em cada configuração de T_k quais os dois próximos estados que ela atinge, e se esta ligação é vantajosa (em termos de custo mínimo) ou não. Caso seja, atualiza-se tal configuração atingida da treliça T_{k+1}. No final, basta escolher qual das configurações da última treliça possui o menor custo, para definir o caminho vencedor.

4) **Como a probabilidade de erro de transmissão foi estimada? Qual é o seu valor? Como ela se compara com o valor de p escolhido? Como ela muda com m?**

   Para estimar a probabilidade de erro, realizou-se a decodificação e comparou-se com a informação inicialmente enviada, vendo a quantidade de bits invertidos. Assim, $P_e = \frac{\text{qtde bits invertidos}}{\text{qtde bits}}$. Entretanto, percebeu-se que este valor variava muito ao rodar o programa algumas vezes, especialmente para alguns valores de p (próximos de 10%). Desta forma, achou-se melhor rodar o programa diversas vezes (20 no total) e realizar

a média e desvio-padrão dos valores de P_e encontrados, estimando-se assim um valor acurado de P_e. Na **Figura 2** , o gráfico com os valores obtidos para cada valor de m e p.

Figura 2. Probabilidade de erro de bit utilizando a decodificação de Viterbi com critério de minimização da distância de Hamming entre o sinal recebido e o sinal esperado.

Podemos perceber que, com o aumento de m, o algoritmo de Viterbi se torna um pouco mais eficiente, corrigindo melhor o sinal recebido. Isto se deve pelo aumento do número de memórias, utilizando mais informações recentes para validar um dado passo.

Além disso, para valores de p grandes (maiores que 0.1), o algoritmo possui bastante dificuldade de correção, enquanto que após $p = 0.1$ ele rapidamente começa a acertar com uma boa eficiência, reduzindo o erro para praticamente 0 com $p < 10^{-2}$.

5) **Qual é o tamanho final do seu executável?**

Um valor razoavelmente elevado, de 37.3 MB, criado com *pyinstaller*. Muito provavelmente o tamanho elevado se deve pela presença de bibliotecas robustas, como bibliotecas de plot de gráficos (matplotlib).

6) **Quanto tempo a sua solução demora para decodificar cada bit? Faça uma média.**

Há uma clara diferença no tempo de codificação quando utilizamos diferentes valores de m. Isto se deve pois a quantidade de estados possíveis cresce exponencialmente com m, o que torna a execução mais lenta com o aumento de m. Assim, fez-se a média para $m = 3$, $m = 4$ e $m = 6$ separadamente, e então a média total. Obtivemos o resultado da tabela 2.

Tabela II
TABELA DE TEMPO DE DECODIFICAÇÃO POR M

m	tempo(ms)
3	0.134 ± 0.003
4	0.271 ± 0.006
6	1.40 ± 0.02

Levando um tempo médio de 0.60 ± 0.02 ms.

7) **É possível utilizar como métrica tanto a distância de Hamming entre os bits recebidos e os potencialmente transmitidos como a probabilidade de ter transmitido uma subsequência qualquer dados os bits recebidos quando estes passam por um canal BSC com parâmetro p. Qual apresenta melhor desempenho?**

Realizou-se diversas medidas para cada valor de m e p. O gráfico da **Figura 3**, para $m = 4$, ilustra bem a comparação. Para outros valores de m a comparação é muito semelhante.

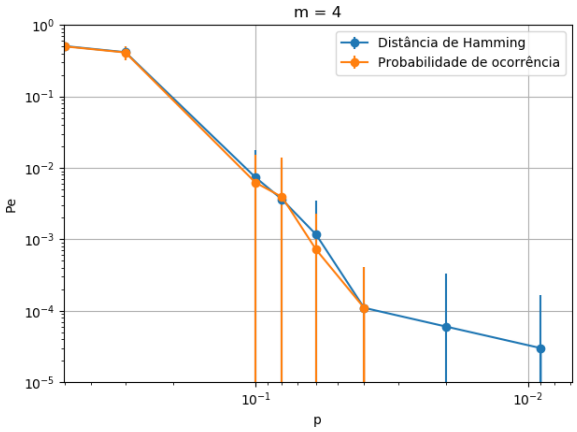

Figura 3. Comparação entre algoritmo de Viterbi e variação utilizando a probabilidade de transmissão de subsequência para $m = 4$.

Podemos perceber que o critério que maximiza a probabilidade de ocorrência é um pouco melhor que o da distância de Hamming, embora ambos se sobreponham em diversos pontos, levando em consideração a margem de incerteza. Mas o critério que utiliza a probabilidade de ocorrência consegue zerar o erro mais rapidamente do que o que utiliza a distância de Hamming, para baixos valores de p. Isto se deve principalmente pelo fato que o critério da probabilidade utiliza de mais informação útil da transmissão em seus cálculos, enquanto que o critério de Hamming funciona como uma aproximação, extremando a inversão de bit a algo puramente binário, sem levar em conta as probabilidades de ocorrência. Assim, com mais informação, o critério de probabilidade consegue ser um pouco mais eficiente.

Além disso, obteve-se a comparação entre os 3 métodos de códigos convolucionais citados. Utilizou-se $m = 6$ para comparar os métodos com a melhor eficiência possível. O resultado é visualizado na **Figura 4**.

É de se perceber que a decodificação convolucional que utiliza como métrica a distância euclidiana é a melhor dentre os métodos apresentados. Isto se deve pelo fato de ser o método que utiliza as informações mais precisas, não somente extremando a inversão de bit (inverte/não inverte), mas tratando o ruído como contínuo, podendo realizar uma análise mais precisa do acontecimento ruidoso. Assim, tal

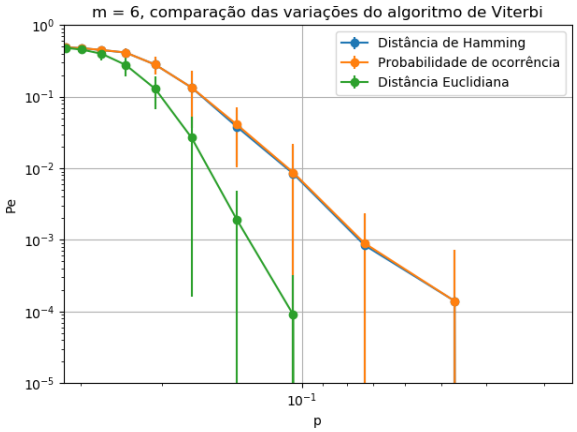

Figura 4. Comparação entre as variações do algoritmo de Viterbi.

critério consegue encontrar melhor o caminho de entrada de bits que ocorreu.

Por fim, a **Figura 5** mostra o resultado da comparação entre os desempenhos de todas codificações implementadas. Para tornar comparação justa, tomou-se inclusive como parâmetro de variação o valor de E_i/N_0, representando de fato a probabilidade de erro em função da energia gasta por bit de informação.

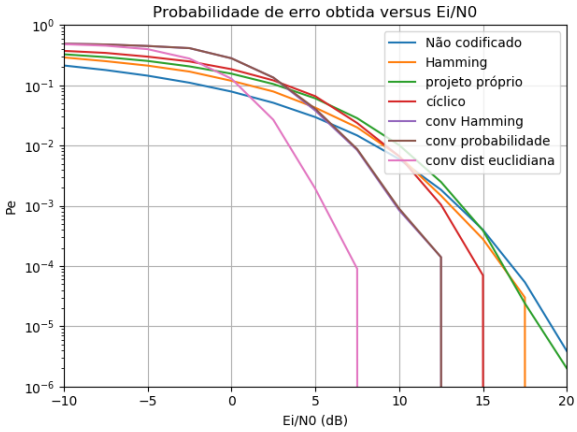

Figura 5. Comparação entre todos os algoritmos desenvolvidos. Os métodos convolucionais de Hamming e de probabilidade estão praticamente superpostos. Evitou-se colocar barras de erro para não poluir o gráfico.

Percebe-se que os métodos convolucionais apresentam melhores resultados em geral. Isto se deve pelo fato de utilizarem de memórias, que armazenam informações recentes para poder obter um resultado melhor, além de levarem em conta a minimização/maximização de algum critério ao longo de todo o caminho. Já os outros métodos são em blocos, não entrelaçando informações consecutivas entre blocos. Apesar de os códigos convolucionais obterem maior eficiência, é necessário decodificar toda uma sequência de informação de uma só vez, enquanto que os outros métodos a decodifica em partes, não sendo necessário aguardar todo o processo para se obter a informação. Além disso, o método convolucional utilizou de um razoável espaço de armazenamento, e apresentou-se mais lento que os outros métodos em geral, o que pode ser uma complicação caso deseja-se transmitir uma larga sequência de dados com rapidez. Como explicado anteriormente, o método convolucional mais eficiente é o que utiliza como critério a minimização da distância euclidiana.

Em termos de capacidade do canal, podemos interpretar que, a cada bit de informação, transmite-se $1/R$ símbolos, sendo assim a taxa do canal de R bits/símbolo. Assim:

$$\begin{aligned} R &\leq \frac{1}{2}\log_2\left(1+\frac{2\cdot E_b}{N_0}\right) \\ \implies \frac{E_i}{N_0} &\geq \frac{2^{2R}-1}{2R} \end{aligned} \tag{5}$$

A tabela abaixo mostra o valor mínimo teórico e experimental de E_i/N_0 que permite que a probabilidade de erro seja tão baixa quanto se queira para cada modelo de decodificação:

Tabela III
TABELA DO LIMITE MÍNIMO DE E_i/N_0, TEÓRICO E EXPERIMENTAL

método	bit info./bit total	$\left(\frac{E_i}{N_0}\right)(dB)_T$	$\left(\frac{E_i}{N_0}\right)(dB)_E$
Não codificado	1	3.522	19
Hamming	$4/7$	0.483	17
projeto próprio	$4/7$	0.483	18
cíclico	$9/17$	0.198	15
conv. Hamming	$1/3$	-1.099	12
conv. prob.	$1/3$	-1.099	12
conv. dist. euclid.	$1/3$	-1.099	7

Podemos perceber, pela **Figura 5** e a **Tabela III**, que os métodos empregados possuem uma razoável distância entre o limite mínimo teórico e experimental, e que o método que mais rápido se aproxima de uma probabilidade de erro da ordem de 10^{-4} é a codificação convolucional com critério de distância euclidiana, numa razão de $E_i/N_0 \approx 7dB$. Entretanto, mesmo assim percebemos tal valor bem distante do valor teórico de $\approx -1.1dB$, apresentando uma boa margem de aprimoramento de projeto. Uma sugestão seria o código Turbo [4], como sugerido em sala.

IV. CONCLUSÃO

Nas atividades realizadas, os alunos tiveram seu primeiro contato teórico com Códigos Convolucionais, assim como utilizaram dessa aprendizagem para projetar um codificador e alguns decodificadores com critérios variados. Após implementação, as análises breves das variações de Viterbi e uma comparação justa entre os desempenhos de todos códigos desenvolvidos até então foram bastante enriquecedoras para a aprendizagem dos alunos na disciplina.

Por fim, as análises e comparações justas realizadas fizeram com que os discentes conhecessem ainda melhor as propriedades em transmissão de cada decodificação, consolidando muito bem os conhecimentos sobre o tópico Codificação de Canal.

REFERÊNCIAS

[1] SHARMA, M. ELE-32 Introdução a Comunicações - Códigos Convolucionais. Disponível em: http://www.ele.ita.br/~manish/ele32/Documentos/ELE32-Lab3v5.pdf
[2] https://en.wikipedia.org/wiki/Mealy_machine
[3] https://en.wikipedia.org/wiki/Hamming_distance
[4] https://en.wikipedia.org/wiki/Turbo_code