Codificador/decodificador convolucional

João Carlos Becker, Leonardo Bianchini, Marco Aurélio Alves Puton

Junho 2017

1 Introdução

Este artigo busca descrever detalhadamente a implementação do código de um codificador/decodificador baseado no algoritmo de Viterbi. A linguagem de programação escolhida foi *Java 8*, pois por ser orientada a objetos, mostrou-se de maior facilidade na implementação.

2 Descrição Geral do Algoritmo

O algoritmo consiste em, fornecida uma entrada e um ruído, codificar a entrada de acordo com a máquina de estados, aplicar o ruído e enviar a mensagem codificada ao decodificador, este que tentará fazer o caminho de volta, traduzindo a mensagem afim de obter a mensagem original fornecida ao programa.

Inicialmente, a partir do estado *00*, percorre-se a entrada fornecida ao programa, codificando índice por índice de acordo com o que está definido na tabela de estados. A tabela de estados foi colocada em uma classe que facilita a obtenção do valor a ser emitido e do próximo estado. Após percorrer toda a entrada, continua executando a máquina de estados para *00*, para manter a propriedade de que cada bit influencie na geração de 3 pares de saída. No final dessa etapa, obtemos a codificação da entrada com o *00* no final.

Após a fase de codificação, aplica-se o ruído que foi fornecido para o algoritmo. Esse ruído é aplicado bit a bit, ou seja, a cada bit é gerado um número aleatório entre 0 e 1 e se o número gerado for menor que o ruído informado, o valor do bit é alterado, caso contrário não se altera, fazendo assim para toda a mensagem codificada. Dessa forma, todos os bits tem a chance de serem trocados.

Com a mensagem codificada e seu ruído aplicado, chama-se o decodificador, que é o responsável por ler a mensagem tentando traduzi-la a fim de obter a entrada original. A partir do estado *00*, vai-se fazendo as transições e calculando para cada caminho o erro acumulado, sendo este erro o número de bits diferentes entre o que seria emitido pelo estado e os bits recebidos.

Como a cada estado há duas possibilidades de decodificação, haverá mais de uma possibilidade de solução. Dessa forma, as possibilidades paralelas são

guardadas em uma lista. Quando houver mais de um caminho para o mesmo estado, escolhe-se o caminho com menor erro ou, caso o valor de erro seja igual para ambos os caminhos, o primeiro é escolhido. Como as possibilidades de decodificação são guardadas em uma lista, basta conferir na lista procurando o estado, adicionando se não encontrar e substituindo caso contrário. As transições são armazenadas em objetos do tipo *line*, que armazenam o estado atual, o erro e a mensagem até então codificada. Faz esse processo até o fim e ignora-se os 2 últimos valores, depois comparando os valores da entrada originais com os codificados.

No final da decodificação, haverão as 4 decodificações com menor erro, das quais é escolhida a que possui menor diferença com a entrada fornecida na execução do programa. Após achar a solução, o programa gera um arquivo *html* exibindo o resultado, além de mostrar como ficou a codificação da entrada, as 4 possibilidades de solução finais e mais algumas informações. Também é exibido os bits diferentes em vermelho, facilitando a visualização da solução encontrada

3 Descrição dos problemas e soluções usadas

O desenvolvimento do codificador e da função que atribui ruído ao que foi codificado foi relativamente tranquilo, pois nessa parte é tratada uma máquina de estados que define o que será emitido de acordo com a entrada, que posteriormente aplica-se o ruído de maneira simples.

Durante o desenvolvimento do decodificador foram encontrados problemas acerca da compreensão do problema. A solução proposta inicialmente considerava todos os casos possíveis, deste modo, sua execução era exponencial em relação ao tamanho da entrada, e para entradas muito grandes o algoritmo não acabava. Com o entendimento do problema, foi possível "podar" a árvore resultante das transições com base na transição para um mesmo estado e no erro até então acumulado. Deste modo, foram obtidos os resultados em complexidade linear ao tamanho da entrada.

4 Exemplos de codificação/decodificação alcançados pelo programa

A seguir, encontra-se algumas imagens exibindo codificações, decodificações e algumas outras informações sobre a execução do algoritmo para algumas entradas. As imagens foram retiradas do arquivo *html* gerado pelo algoritmo.

erro: 0,100000

entrada: 10100

Codificado
s/ erro: 1110001011
c/ erro: 1110001001

Decodificado	Diferença	Erro
101	0	1
100	1	3
101	0	2
101	0	1

Resultado

101

Figure 1: Imagem gerada com a entrada *101*

erro: 0,100000

entrada: 11111111111111111100

Codificado
s/ erro: 11011010101010101010101010101010101010100111
c/ erro: 11011000101011100101010101010101110001110111

Decodificado	Diferença	Erro
11111111111111111111	0	6
11111111111111110001	3	6
11111111111111110001	3	6
11111111111111111100	2	6

Resultado

11111111111111111111

Figure 2: Imagem gerada com a entrada *11111111111111111111*

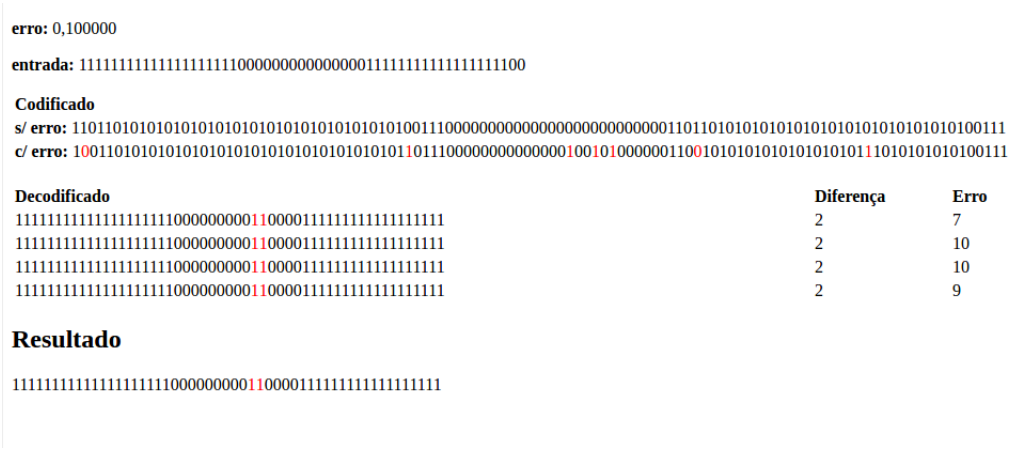

Figure 3: 11111111111111111111000000000000000111111111111111111

Para testar o algoritmo, foi criado um *script* em *python* executando o programa para diferentes entradas e valores de ruído. Os valores de entrada e resultados podem ser vistos na imagem 4, a seguir. Para cada entrada e ruído, o programa foi executado 20 vezes, e o resultado exibe a diferença média de bits entre a entrada original e a solução encontrada.

<Erro>	<Tamanho da entrada>	<Media de diferença>
0.1	10	0.350000
0.1	20	0.700000
0.1	40	3.500000
0.1	80	5.750000
0.1	160	11.450000
0.15	10	1.100000
0.15	20	3.300000
0.15	40	5.700000
0.15	80	17.150000
0.15	160	27.550000
0.2	10	1.300000
0.2	20	4.800000
0.2	40	9.700000
0.2	80	24.250000
0.2	160	51.550000
0.5	10	4.100000
0.5	20	8.950000
0.5	40	19.550000
0.5	80	38.700000
0.5	160	79.500000

Figure 4: Execuções do algoritmo para diferentes valores de entrada

5 Conclusão

Ao término da implementação foi obtido um código capaz de codificar, inserir ruído e decodificar, devolvendo o valor da maneira mais próxima possível da entrada fornecida. Foi possível observar que o algoritmo apresenta resultados satisfatórios com ruído de até 0.1 (10%). Também pode se observar que quanto maior for a entrada fornecida, maior é a média de diferença quando comparada a entrada com a solução.