UNIVERSIDADE DE AVEIRO

DEPARTAMENTO DE ELETRÓNICA, TELECOMUNICAÇÕES E INFORMÁTICA

TEORIA ALGORITMICA DA INFORMAÇÃO - 2019/2020

Lab Work 2

Mestrado em Engenharia Informática

Authors:

Bruno Assunção, 89010

Cláudio Costa, 85113

João Artur Costa, 80390

Professor:

Prof. Armando Pinho

November 24, 2019

Contents

1 Descrição da solução 4

2 Input do Programa 4

3 Criação dos Módulos 5

4 Resultados 8

5 Conclusão 8

List of Figures

3.1 Equações utilizadas para o cálculo do ***SNR*** 6
3.2 Fórmula para o cálculo da distância entre vetores 7

Introdução

O objetivo principal deste Lab Work consiste na implementação de um sistema de identificação automática de músicas usando pequenas amostras de áudio das mesmas.
Para tal, foram desenvolvidos em C++ os módulos necessários para a implementação final do interpretador de música.

1 Descrição da solução

No problema proposto, com recurso a uma base de dados com uma grande quantidade de músicas completas e a uma gravação de *query* de input (com possível ruído) a essa base de dados, obtém-se o resultado final de classificação da gravação com uma das músicas na base de dados.
A cada música na base de dados está associada um ***codebook*** criado com recurso à quantização vetorial do ficheiro de áudio.
Quando uma *query* é feita, a amostra é codificada e descodificada usando cada um dos ***codebooks*** referidos anteriormente.
Por fim, cada versão da amostra descodificada por cada um ***codebooks*** da base de dados é comparada à versão original, medindo a distância entre cada uma delas.
A versão que tiver mais semelhança com a original é a música adivinhada.

2 Input do Programa

Os ficheiros do nosso programa a ser executados são o ***wavcp.cpp*** e o ***wavhist.cpp***.
Após compilar o programa com o comando ***make***, segue-se:

```
$../bin-example/wavcp sample.wav copy.wav
```

Este é o segundo comando a executar. O ficheiro "sample.wav" é copiado para o ficheiro "copy.wav"

```
$../bin-example/wavhist sample.wav 0
```

Este é o terceiro comando a executar. Neste caso de execução, é gerado um histograma do canal "0" (esquerdo) do ficheiro "sample.wav".

```
$../bin-example/wavquant sample2.wav sample.wav 8
```

Este é o quarto comando a executar, gerando uma cópia quantizada do ficheiro "sample.wav", com uma variável de quantização ***q*** = 8.

```
$../bin-example/wavcb sample.wav 2 10 4 1
```

Este é o quinto comando a executar. Neste caso, o programa gera o ***codebook*** do ficheiro "sample.wav" com um ***block_size*** = 2, ***overlap_factor*** = 10, com 4 ***clusters***, durante 1 iteração (último parâmatro - ***iterations***).

```
$../bin-example/wavfind sample.wav
```

Este é o sexto e último comando a executar, tendo como *output* o ***codebook*** mais provável para a *sample* fornecida no argumento.

3 CRIAÇÃO DOS MÓDULOS

- **Criação do histograma (*wavhist*) -** Para a implementação da classe ***wavhist.h*** que irá representar a média dos canais no ficheiro de som de amostra, foi criada uma função que atualiza a média dos canais à medida que itera sobre eles. O resultado é de seguida exportado para um ficheiro *.csv*, permitindo a sua visualização por uma aplicação externa (*ex.: Tabela de Excel*).

- **Quantização escalar uniforme (*wavquant*) -** De modo a reduzir o número de bits usados para representar cada amostra de áudio, foi criado na classe **C++** ***wavquant.h*** e no programa associado um quantizador escalar uniforme. Na função de quantização, os bits da *sample* sofrem um *shift right* e, posteriormente, os bits perdidos são repostos a zero, o que leva à perda de informação e quantização do ficheiro de som. Finalmente, são retornadas as amostras quantizadas.

- **Cálculo do *Signal-to-Noise Ratio* (*wavcmp*) -** Foi proposta a implementação de um programa ***wavcmp***. A sua função é calcular o *signal-to-noise ratio* (*SNR*) de um ficheiro de som em relação a um ficheiro original, usando as equações da energia de sinal e energia de ruído:

$$\mathrm{SNR} = 10\log_{10}\frac{E_s}{E_n}\quad(\mathrm{dB}),\qquad E_s = \sum_k x_k^2\qquad E_n = \sum_k (x_k - \tilde{x}_k)^2.$$

Figure 3.1: Equações utilizadas para o cálculo do ***SNR***

- **Criação de *codebooks* (*wavcb*) -** Nesta etapa do trabalho, é necessária a criação do programa responsável pela computação de um ***codebook*** de quantização vetorial usando o algoritmo the *clustering* de ***k-means***. É exigido também que o programa aceite como pârametros de entrada o tamanho de cada bloco, o fator de *overlap* e o número de *clusters* do *codebook*.

  - ***populate_blocks()*** - Função responsável pela transformação da amostra de áudio em blocos de amostra. Este processo de transformação é afetado pelo parâmetro do ***block_size***, que irá definir o tamanho de cada partição e pelo ***overlap_factor***, que indica quantos índices irá "saltar" na amostra original. O algoritmo inicia-se por iterar sobre a amostra lida, tendo como *step* da iteração o fator de *overlap*. Sequencialmente, vai-se adicionando a cada bloco **n** *samples*, sendo **n** o tamanho máximo de cada bloco, passado em argumento. Por fim, obtém-se um vetor de vetores, constituindo a amostra original separada em blocos mais pequenos.

  - ***populate_means()*** - Nesta função, inicializam-se os vetores representativos dos *clusters* obtidos na função descrita anteriormente. Começamos escolhendo um determinado número de *clusters* (passado em argumento) dos blocos da amostra, como as "***means***" iniciais.

– ***cluster_blocks()*** - Função que irá determinar a versão final dos vetores representativos, ao iterar sobre cada ***mean*** e determinar qual vetor está mais próximo da ***mean*** em questão. Tal proximidade obtém-se a partir de uma função ***dist_between_vecs()*** que obtém a distância da seguinte forma:

$$distance = \sum_{i=0}^{n} \frac{(vec1_i - vec2_i)^2}{n}$$

Figure 3.2: Fórmula para o cálculo da distância entre vetores

Por fim, o dicionário (inicializado aleatoriamente) que contêm os vetores e respetivas ***means*** é atualizado para os novos valores.

– ***get_codebooks()*** - Esta função representa a função principal para a geração do ***codebook***. Inicia-se por fazer a divisão da amostra em blocos e por popular as ***means*** com os parâmetros passados. De seguida, e durante um especificado número de iterações faz-se o *clustering* dos blocos e a atualização das ***means*** sequencialmente. Após isso, as ***means*** finais são extraídas do mapa de vetores representativos quando, após um certo número de *clusterings* e atualizações de ***means***, os ***centroids*** estão mais próximos do centro do *cluster*. Finalmente, os ***centroids*** (*means*) finais são exportados para um ficheiro para futura leitura.

– ***populate_means()*** - Nesta função, inicializam-se os vetores representativos dos *clusters* obtidos na função descrita anteriormente. Começamos escolhendo um determinado número de *clusters* (passado em argumento) dos blocos da amostra, como as "***means***" iniciais.

- **Classificador de música (*wavfind*) -** Este módulo final do sistema de reconhecimento de música é o *wavfind*, que conjuga os módulos desenvolvidos anteriormente para fazer o *match* entre o ficheiro de áudio de *input* e os ficheiros de música na base de dados. Tal obtêm-se calculando a distância entre o ***codebook*** da amostra de *input* com cada ***codebook*** de cada música da base de dados. São iterados todos os ficheiros no diretório em que estão contidos e é calculado o SNR entre a ***mean*** do ***codebook*** e o vetor associado ao áudio de input. O ficheiro áudio cujo ***codebook*** tenha um **SNR** maior em relação à amostra de input será o melhor candidato à classificação da música.

4 RESULTADOS

Ao longo do desenvolvimento do trabalho foram encontrados vários fatores que influenciam a performance computacional do programa.
De todos os módulos, a construção de ***codebooks***, (***wavcb***) foi aquele no qual foram mais notórias as variações de performance com a variância de parâmetros de input.
A título de exemplo, é notório um aumento do tempo de execução da primeira fase da construção (***populate_blocks()*** e ***populate_means()***) quanto maior for o valor do tamanho de cada bloco, enquanto que a segunda parte do programa (***cluster_blocks()*** e ***update_cluster_means()***) é mais afetada negativamente com o aumento do número de *clusters* a serem escolhidos nos blocos da amostra.

5 CONCLUSÃO

Consideramos a implementação destes interpretadores de amostra de áudio conseguem ser bastante precisos utilizando os processos que implementámos. Apesar de um dos maiores *bottlenecks* à precisão deste classificador ser o ruído presente na amostra de input, através do processo de normalização escalar é possível reduzir este fator, aproximando mais a amostra da representação real da música.