Speech Compression

Métodos Numéricos Avanzados

Juan Pablo Orsay - 49373
Horacio Miguel Gomez - 50825
Sebastián Andrés Maio - 50386
Federico Bond - 52247
Braulio Sespede - 51074

Noviembre 2016

En este trabajo implementaremos un simple compresor del habla. Nos basaremos en *2nd International Conference on Computer and Communication Technology*, donde se propone un esquema muy simple de compresión basado en tres pasos: transformación, cuantificación y codificación Huffman.

Índice

1. Procedimiento

Para realizar la compresión seguimos los siguientes pasos:

1. Obtener la FFT de la muestra de audio.
2. De los coeficientes obtenidos se guarda la mitad +1.
3. Los coeficientes cuyo valor absoluto es menor que *Epsilon* se los hace 0.
4. Cuantificamos la parte real y la imaginaria de los coeficientes con L bits.
5. Codificamos con el algoritmo de Huffman los coeficientes.

Para recuperar la grabación, hace falta de-codificar la información generada previamente y aplicar IFFT inversa.

2. Ejemplo compresión y descompresión

Para realizar la compresión y descompresión de un archivo de sonido, abrir octave dentro del directorio *src* del proyecto y ejecutar lo siguiente:

```
wavName = "01"
epsilon = 0.1
L = 16
[compressed, scale] = compress(wavName, epsilon, L)
```

Siendo:

1. wav_name el nombre del archivo dentro de resources/wav a comprimir
2. *epsilon* el valor a utilizar para eliminar ruido
3. L el número de bits a utilizar en la cuantización

En *compressed* obtenemos el wav comprimido, cuantizado y truncado que luego puede ser descomprimido mediante el siguiente código:

```
wav_name = "01" %01_recompressed.wav
uncompressed(compressed, "01")
```

3. Resultados

Al ejecutar el código para comprimir y descomprimir el archivo *11.wav*, obtuvimos los siguientes resultados.

3.1. Compresión

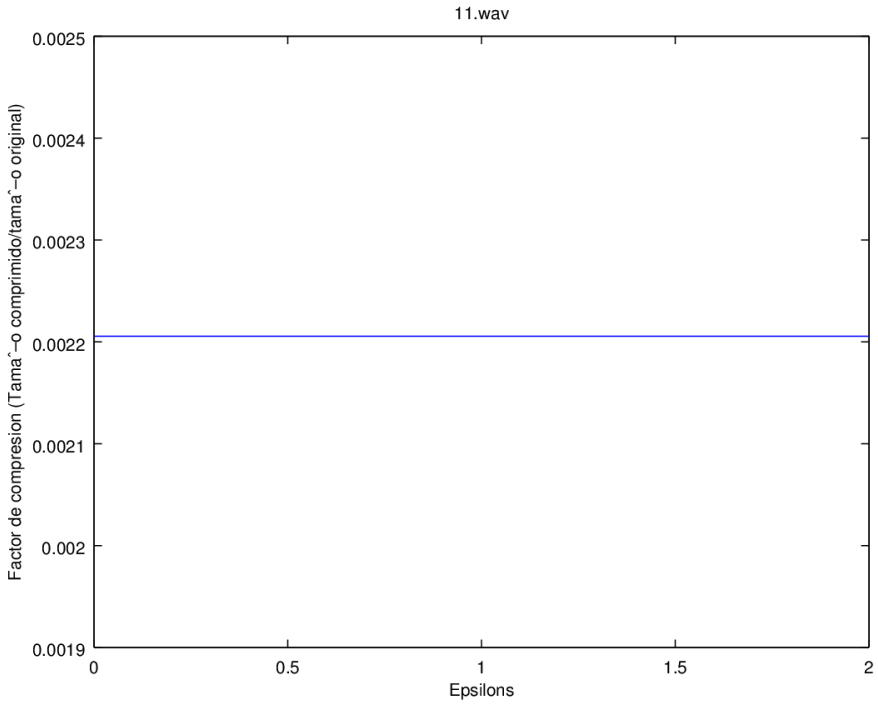

Figura 1: Factor de compresión

3.2. Relación entre bits mapeados y compresión

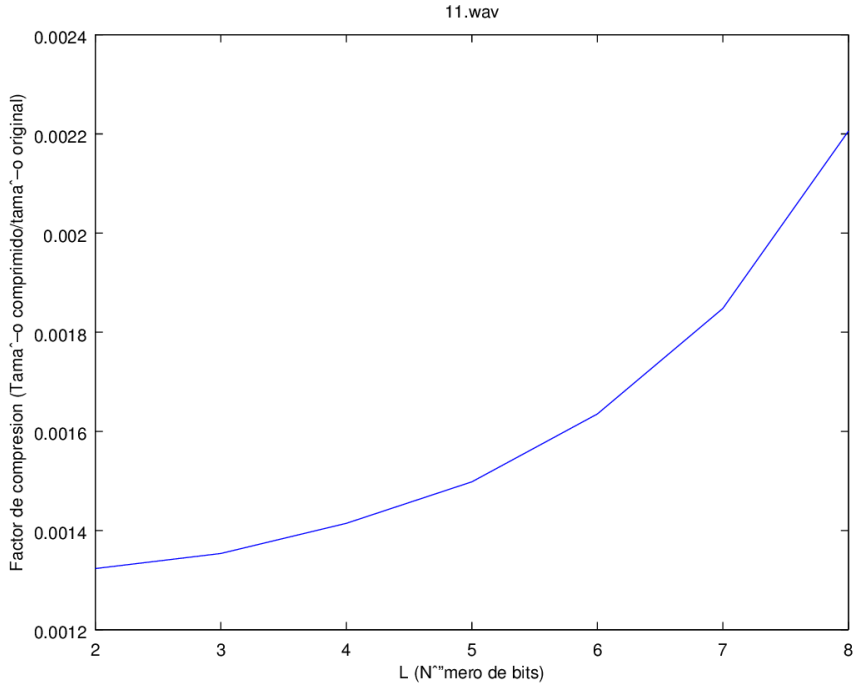

Figura 2: Relación entre factor de compresión y bits utilizados para el mapeo

3.3. Distorsión cuadrática media

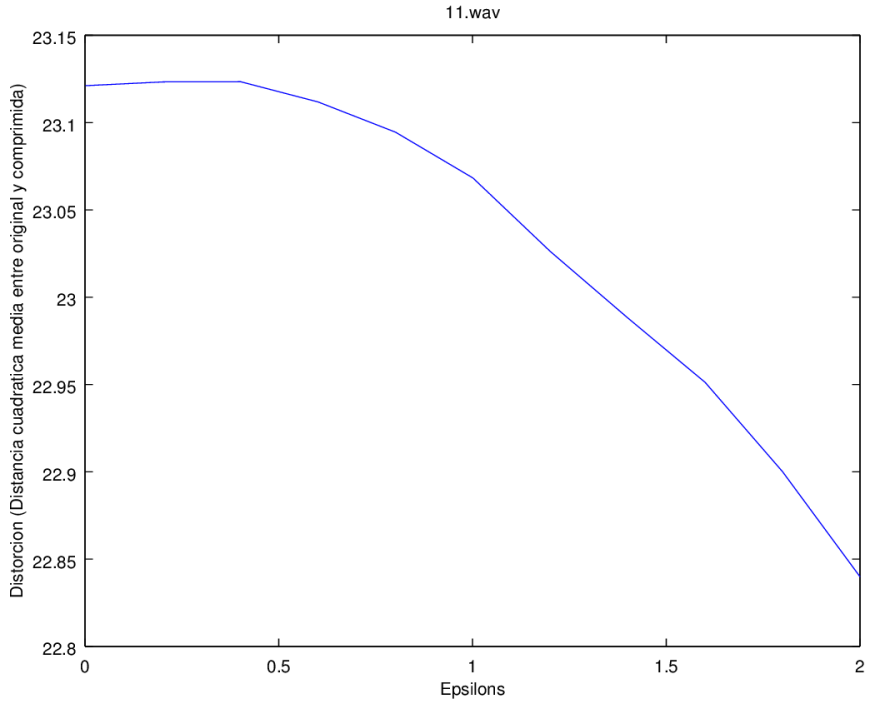

Figura 3: Distorsión cuadrática media respecto de *epsilon*

3.4. Distorsión

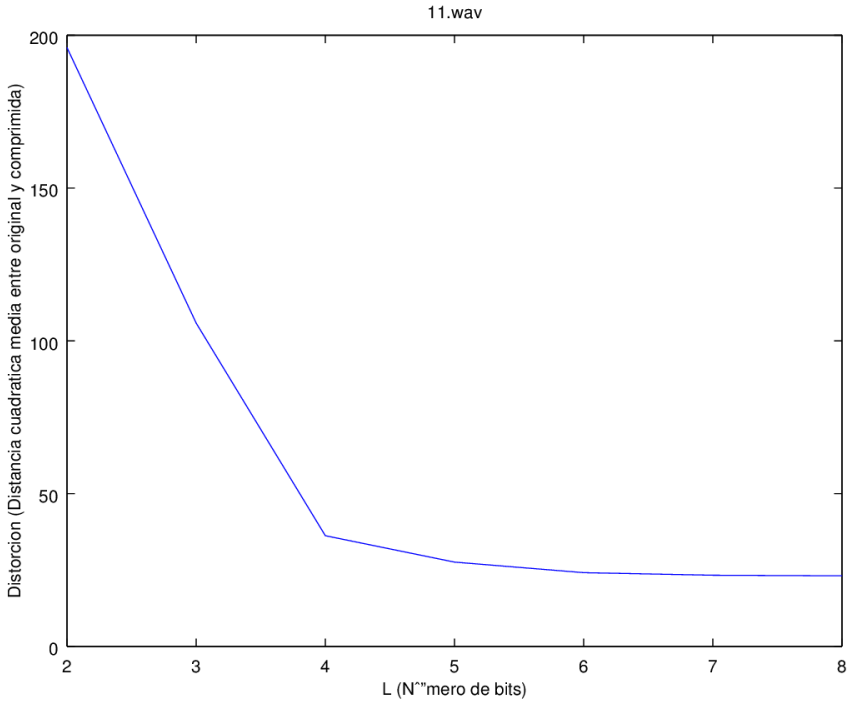

Figura 4: Distorcion cuadratica media respecto de la cantidad de bits utilizados

4. Conclusión

Pudimos observar que cuantos menos bits utilizamos a la hora de cuantizar los coeficientes resultantes de la FFT de las muestras presentadas obtenemos un factor de compresión considerable pero, a su vez, encontramos que tras descomprimir lo obtenido previamente el sonido resultante tiene un gran número de ruido.
Por otro lado encontramos que a mayores números de épsilon, más apagado se escucha el audio descomprimido ya que al tener números altos de épsilon estamos filtrando una mayor amplitud del sonido resultando en la pérdida de contenido y definición.
Por último, encontramos que por alguna razón que desconocemos, todos los audios que comprimimos, al ser descomprimidos tienen una gran caída de volumen en el medio de la grabación.

5. Anexo

5.1. Código

Dentro de la carpeta *src* se encuentran los archivos de octave utilizados en la implementación del trabajo práctico:

- compress.m: código para comprimir un archivo de sonido
- uncompress.m: código para descomprimir un archivo de sonido
- main.m: archivo que comprime y descomprime los archivos de prueba
- plot_fixed_bits.m: código para generar los gráficos
- plot_fixed_epsilon.m: código para generar los gráficos
- stats.m: código que genera estadísticas sobre la compresión/descompresión a ser utilizadas en los gráficos
- uncompress.m: código para descomprimir un archivo comprimido mediante el método propuesto
- huffman.m y myhuffmandict.m: código relacionado a la codificación huffman[1]

Referencias

[1] *Gryllos Prokopis' nhuff*. URL: https://github.com/PGryllos/nhuff (visitado 17-11-2016).