Speech compression
Métodos Numéricos Avanzados

Enzo Altamiranda, Cristian Ontiver, Valeria Serber

19 de noviembre de 2014

Resumen

El siguiente informe explora el uso de la transformada rápida de Fourier, en combinación con la codificación de Huffman para la compresión de voz. Se realiza un estudio comparativo de diferentes niveles de cuantificación y sus efectos tanto en el tamaño como la calidad - en términos de distorción - del archivo comprimido.

1. Palabras Clave

Transformada Rápida de Fourier, Compresión de voz, Codificación de Huffman

2. Introducción

La voz, entendida como el sonido generado por el aparato fonador humano, se encuentra en frencuencias de entre 80 a 1100 Hz. Teniendo esto en cuenta, a la hora de almacenar digitalmente la señal de voz, es posible utilizar métodos para reducir el tamaño necesario para guardar la información correspondiente a esta señal. Tradicionalmente estos se dividen en dos. Por un lado métodos "lossless"(sin pérdida), es decir, métodos que almacenan la información de modo tal que esta pueda ser recuperada en su totalidad. Por otro, métodos "lossy"(con pérdida), en los cuales, tomando ventaja de que el rango de audición humano suele situarse entre los 20 y 20,000 Hz (con variaciones de individuo a individuo), descartan información considerada redundante (aquella por encima o por debajo de ese rango).
Aquí exploramos una forma de lograr lo segundo, mediante el uso de la transformada rápida de Fourier, en conjunto con la codificación de Huffman, para lograr almacenar diferentes archivos de sonidos conteniendo voces, de manera que se reduzca el tamaño necesario.
Como se verá más adelante, una reducción en la distorición y aumento en la fidelidad del audio comprimido implican la necesidad de mayor información, y por ende, una menor reducción en relación al tamaño (en bytes) original.

3. Metodología

3.1. Cálculo de la matriz del Modelo de Ising

3.1.1. Primera Iteración

En la primera iteración se optó por almacenar cada matriz en un arreglo de arreglos. Se comenzó realizando una implementación del algoritmo estándar de multiplicación de matrices, debido a la simpleza del código correspondiente. Se probó esta versión creando las matrices K y L y multiplicándolas. El orden temporal de este algoritmo es cúbico, es decir, $O(n^3)$. Debido a esto, al calcular matrices pequeñas el algoritmo termina de forma rápida, sin embargo, al probar matrices de mayor tamaño, el tiempo crece en gran medida.

3.1.2. Segunda Iteración

Debido a los resultados anteriores, se hizo evidente que se debía mejorar el programa para disminuir su complejidad. Para ello, se utilizó el hecho de que las matrices K y L son **ralas**, es decir, matrices de gran tamaño, en la que la mayoría de los elementos son cero. Para optimizar el algoritmo, se optó por cambiar la estructura de datos que almacena las matrices, de modo que los ceros no se almacenen, y el algoritmo pueda saltear las operaciones que den como resultado cero. La estructura utilizada en esta iteración se conoce como *"Compressed column storage" (CCS)*, o alternativamente *"Compressed sparse column"*, y es la representación tradicional utilizada en *MATLAB* al usar la función *"sparse"*.

Esta representación cuenta con tres arreglos. El primero, al cual llamaremos ***values***, contiene los valores no nulos de la matriz, de tamaño ***nnz*** (del inglés *"number of nonzeros"*). El segundo, $\boldsymbol{r_i}$, indica el índice de la fila del elemento que se encuentra en el primer arreglo y en la misma posición. Dicho arreglo también tiene tamaño *nnz*. El tercero, ***cp***, tiene como tamaño la cantidad de columnas más uno, donde en la posición j del arreglo se guarda el índice en el arreglo de valores, en el que se encuentra el primer elemento no nulo de la columna j. El último elemento de *cp* es el valor *nnz*. Si alguna columna tuviera todos ceros, en el índice de esa columna se coloca el mismo valor que en la próxima columna. Utilizando el arreglo *cp* se puede conocer la cantidad de elementos no nulos presentes en cada columna j, para ello basta con calcular la resta: *cp[j+1] - cp[j]*.

Ejemplo de almacenamiento de una matriz utilizando Compressed column storage

$$M = \begin{pmatrix} 0 & 3 & 0 & 5 & 7 \\ 0 & 0 & 0 & 3 & 8 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 5 \end{pmatrix}$$

Notar que la tercer columna no tiene valores. Esto se representa usando el mismo valor en *cp* que en la próxima columna. Esto es consistente con el hecho de que la cantidad de elementos no nulos presentes en la columna j se puede conocer con la resta

$$cp[j+1] - cp[j].$$

$$values = \begin{bmatrix} 1, & 3, & 2, & 5, & 3, & 7, & 8, & 5 \end{bmatrix}$$

$$ri = \left[\begin{array}{cccccccc} 2, & 0, & 3, & 0, & 1, & 0, & 1, & 3 \end{array} \right]$$

$$cp = \left[\begin{array}{cccccc} 0, & 1, & 3, & 3, & 5, & 8 \end{array} \right]$$

Complejidad espacial de la representación CCS

Para una **matriz densa**, es decir, que no tiene elementos nulos, de m x n, se necesitan m punteros en memoria, cada uno apuntando a los m arreglos de n elementos. En comparación con una matriz en representación *CCS*, se necesita espacio para *nnz·sizeof(elems) + (nnz + cols)·sizeof(índices)*. Para una aproximación más fácil de entender, si suponemos que el tamaño de índices, elementos, y punteros en memoria usan todos a misma cantidad de bytes, se tiene que la complejidad espacial de la matriz densa es $m+m·n$, es decir, $O(m·n)$. Como en el caso del Modelo de Ising las matrices son cuadradas, queda de orden espacial $O(n^2)$. En cambio, la complejidad espacial de la **matriz rala** es $2·nnz+n$, es decir orden $O(nnz+n)$, que es *lineal*.

Algoritmo de multiplicación CCS

4. Resultados

4.1. Comparación entre ambas iteraciones

Se realizaron pruebas en ambas iteraciones para descubrir las diferencias entre ellas. Se utilizaron matrices de diferentes tamaños, y se utilizó el comando *"time"* de *Linux* para medir el tiempo de cada caso. A continuación se muestra una tabla con los valores que se utilizaron en las pruebas, y luego los gráficos que se realizaron a partir de ellas.

Multiplicación estándar		Multiplicación CCS	
m	**tiempo (seg)**	**m**	**tiempo (seg)**
100	0.02	100	0
150	0.05	150	0
200	0.13	200	0
250	0.25	250	0.01
300	1.32	300	0.01
350	2.88	350	0.01
400	4.24	400	0.01
450	6.51	450	0.02
500	8.90	500	0.02
550	12.05	550	0.02
600	15.81	600	0.03
650	20.22	650	0.03
700	25.25	700	0.04
750	32.36	750	0.04
800	36.33	800	0.05
850	45.21	850	0.06
900	46.49	900	0.06
950	54.95	950	0.06
1000	62.82	1000	0.07
		10000	3.19
		20000	7.14
		25000	19.59
		30000	28.22
		35000	39.30
		40000	52.50
		45000	63.79

Tabla 1: Muestra el tiempo en segundos, que tardaron los dos algoritmos implementados respectivamente, a partir de distintos valores de m.

4.1.1. Primera iteración

4.1.2. Segunda iteración

4.2. Descripción de los resultados

El resultado más importante que se puede obtener de los gráficos anteriores es que en el intervalo de 0 a 70 segundos, el primer algoritmo puede resolver hasta matrices de $m = 1000$, mientras que el segundo puede resolver hasta matrices de $m = 500000$ aproximadamente. Además, se puede observar que la curva del primer algoritmo crece mucho más rápido que la segunda. Esto puede corresponderse con el hecho de que el primer algoritmo es de $O(n^3)$, mientras que el otro es de orden menor.

5. Conclusiones

5.1. Cálculo de A

Si bien se logró optimizar bastante el algoritmo que calcula la matriz A, mientras se realizaban pruebas se pudo observar un patrón en la construcción de A. Se notó que a partir de $m = 3$, la matriz A puede escribirse genéricamente, ya que pequeños bloques de elementos se repiten a lo largo de la estructura, aumentando la cantidad de bloques lógicamente mientras m aumenta. Por lo tanto, se podría evitar tener que calcular A a partir del producto entre K y L, y en vez de eso, seguir la siguiente regla para representarla:

Cuando $m = 3$, la matriz comienza a expandirse dejando ceros en varias posiciones. Además puede verse que el bloque del medio compuesto por Q, R, S y T comienza a repetirse.

Entonces, para $m >= 3$, la matriz puede escribirse de forma genérica de la siguiente manera:

$$A = \begin{pmatrix} T & Q & R & & & & & & & S \\ & S & T & Q & R & & & & & \\ & & & S & T & \ddots & & & & \\ & & & & & \ddots & & & & \\ & & & & & & Q & R & & \\ & & & & & & S & T & Q & R \\ R & & & & & & & S & T & Q \end{pmatrix} \tag{1}$$

Siendo

$$Q = \begin{pmatrix} \sin\alpha & \cos\beta \\ \cos\alpha & \cos\beta \end{pmatrix} \tag{2}$$

$$R = \begin{pmatrix} \sin\alpha & \sin\beta \\ \cos\alpha & \sin\beta \end{pmatrix} \tag{3}$$

$$S = \begin{pmatrix} -\cos\alpha & \sin\beta \\ \sin\alpha & \sin\beta \end{pmatrix} \tag{4}$$

$$T = \begin{pmatrix} \cos\alpha & \cos\beta \\ -\sin\alpha & \cos\beta \end{pmatrix} \tag{5}$$

6. Bibliografía