FICH, UNL - Departamento de Informática - Ingeniería Informática

Procesamiento Digital de Imágenes

Guía de Trabajos Prácticos 8

Compresión de imágenes

2013

1. Objetivos

- Analizar las características y el desempeño de tres algoritmos muy difundidos para compresión reversible de imágenes y compresión con pérdidas.
- Evaluar subjetiva y objetivamente la calidad de la reconstrucción de imágenes comprimidas.

2. Trabajos Prácticos

Ejercicio 1: Compresión sin pérdidas por códigos de longitud variable. Huffman

El método de compresión por códigos de longitud variable asigna, a los grises con mayor probabilidad de realización, una cadena binaria de pocos bits. Mientras que, a los grises que tienen una probabilidad de realización inferior, les asigna un código de mayor longitud en bits. Este método tiene sus fundamentos en conceptos de teoría de la información, obteniendo mayor provecho en aquellas imágenes cuya distribución de grises tiene una entropía baja. Por otra parte, en aquellas imágenes cuya distribución de grises tiende a ser uniforme, o sea tiene entropía alta, este método obtiene malos resultados.
En este ejercicio se pretende que el alumno implemente el núcleo de la codificación Huffman en papel, sin embargo, algunas tareas más complejas pueden implementarse en un programa (por ejemplo: cálculo de histograma, obtención de las cadenas comprimidas, el cálculo aproximado del tamaño comprimido, la estimación de la tasa de compresión, etc.).
El cálculo de las cadenas (algoritmo de ramificación de probabilidades) debe realizarse en papel, para ésto se trabajará con una versión cuantizada de una imagen de referencia. Los pasos para el desarrollo del ejercicio serán los siguientes:

1. Cargar la imagen `'estanbul.tif'` y realizar la cuantización en 8 niveles de gris.
2. Calcular el histograma de la imagen cuantizada y guardarla en un vector p.

3. Obtener dos nuevos vectores con el ordenamiento ascendente del histograma: un vector y que tendrá los valores de probabilidad, y otro u almacenará el nivel de gris correspondiente (ó un índice). Ordenar descendentemente los vectores en base a su probabilidad (como indica el algoritmo de Huffman).
4. **En papel**: desarrollar el algoritmo de ramificación de probabilidades y obtener los códigos de longitud variable correspondientes a cada nivel de gris.

La cátedra le suministrará el archivo `'HuffmanBinaryTreeOptimized.h'`, allí se encuentra implementado el algoritmo completo para realizar la codificación de Huffman. En la cabecera de este archivo encontrará la información acerca de como utilizarlo. Los parámetros básicos que requiere la función son un array con los valores de los sucesos (ó las probabilidades) de los símbolos (niveles de gris en el caso de imágenes), un array de string donde guardar la codificación de Huffman y el tamaño del array de datos. Opcionalmente se puede obtener *la tasa de compresión* y/o *la entropía* y *la longitud de palabra media*.

Este código fue adaptado para que retorne la codificación en un array de string, sin embargo, las estructuras implementadas permiten que con pocas modificaciones se puedan obtener las ramificaciones de Huffman completas, con los índices de los símbolos, las probabilidades en cada nodo y el código Huffman.

Para continuar con los ejercicios se pide:

1. Compilar el archivo `'ExampleHuffman.cpp'` (suministrado por la cátedra).
   Para ejecutar este programa: `./ExampleHuffman valor1 valor2 valor3 ...`
2. Comparar los códigos obtenidos en papel en el ejercicio anterior con la codificación obtenida con `'ExampleHuffman.cpp'`.
3. Realizar pruebas para distintas cantidades de símbolos y diferentes distribuciones de probabilidad. Analizar y comparar la información de *entropía* y *longitud de palabra media* que se obtienen con el programa para las distintas cadenas. Observar que sucede con las medidas de información cuando la cantidad de datos es potencia de 2 (2^n) y cuando no lo es.
4. Elegir varias imágenes que le despierten interés para calcular el histograma y obtener su codificación Huffman. Analizar los datos obtenidos de las imágenes y las tasas de compresión.

Ejercicio 2: Compresión por codificación de longitud de cadena

El algoritmo de compresión RLC sin pérdida utiliza un contador que acumula la cantidad de ocurrencias sucesivas que presenta determinado nivel de gris. Al detectar otro nivel de gris, se almacena un binomio (nivel de gris, ocurrencias) y se vuelve a contar para el nuevo gris. El rendimiento de este método es bueno en imágenes sin ruido (binarias o con pocos niveles de grises) siempre que posean grandes zonas homogéneas. En este ejercicio se pide implementar el esquema completo de compresión y descompresión por el método RLC sin pérdida para imágenes binarias.

Los pasos sugeridos son:

1. Cargar una imagen de MxN y obtener una versión binaria de la misma (fijando el umbral eurísticamente).
2. Definir un vector fila C que tendrá todo el archivo comprimido.
3. Especificar qué valor de gris (0 ó 1) es el inicial, así se aprovecha que sólo hay dos valores posibles y en vez del binomio se guardará en C sólo la cuenta de grises consecutivos.
4. Recorrer la imagen y construir C. (Recordar que la suma de los valores de C debe ser igual a MxN).
5. Calcular la tasa de compresión estimada.
6. Generar un formato propio de archivo comprimido, con el qué se ha de guardar C en un archivo binario (no olvidar una cabecera con la cantidad de columnas de la imagen y el gris inicial (0 ó 1)). Calcular la tasa de compresión verificando el tamaño en disco del archivo original binarizado y del archivo comprimido.
7. Implementar la descompresión para generar la imagen binaria a partir de la versión comprimida.

Ejercicio 3: Compresión RLC con pérdidas

El algoritmo de compresión RLC sin pérdidas computa la cantidad ocurrencias consecutivas de cierto nivel de gris. La variante con pérdidas es más flexible puesto que realiza la cuenta con niveles de gris dentro de un rango ó ventana. El propósito de este ejercicio es analizar el programa `'lossy_rlc.cpp'` suministrado por la cátedra.

Para este ejercicio el alumno realizará las siguientes actividades:

1. Compilar el programa `'lossy_rlc.cpp'`. (obtendrá ayuda con `'lossy_rlc -h'`).
2. Comprimir la imagen `'estanbul.tif'` y evaluar subjetivamente la calidad obtenida.
3. Realizar una gráfica del error cuadrático medio entre la imagen original (sin comprimir) y el resultado de la descompresión utilizando diferentes anchos de ventana.
4. Analizar en qué casos es útil el método y en cuáles no presenta ventajas.
5. Repetir el análisis para las imágenes `'city.tif'` y `'camaleon.tif'`.

El programa suministrado contiene una implementación completa de un compresor/descompresor de imágenes cuyo tamaño máximo es 65536x65536. Se incluye la escritura en disco de un archivo binario comprimido con un formato definido que es el siguiente:

Fhb	Flb	Chb	Clb	AV_1	m_1	AV_2	m_2	...

donde:

Fhb	byte alto de la cantidad de filas
Flb	byte bajo de la cantidad de filas
Chb	byte alto de la cantidad de columnas

Clb	byte bajo de la cantidad de columnas
AV_i	ancho de ventana de la i-ésima ejecución
m_i	gris medio de la i-ésima ejecución

El pseudo-código para la implementación de la compresión tiene los siguientes pasos:

1. Cargar la imagen especificada como parámetro, obtener su tamaño y fijar AV como el tamaño inicial del ancho de ventana.
2. Inicializar en cero un vector renglón I_{comp} que contendrá todo el archivo comprimido según el formato especificado.
3. Expresar el número de filas de la imagen original en forma binaria con dos bytes. Por ejemplo, el número 256 se especifica con un byte alto cuyo valor es `'00000001'` y un byte bajo igual a `'00000000'`. Se guarda el byte alto en el primer elemento de I_{comp}, y el byte bajo en el segundo elemento.
4. Repetir el proceso para las columnas, y guardar los bytes en los elementos 3 y 4 del vector de compresión.
5. A continuación se explica el ciclo del algoritmo RLC con pérdidas que se ha de repetir para todas las filas.
   a) Inicializar variables: cadena codificada, ancho de ventana, columna de inicio, mínimo y máximo gris del rango.
   b) Avanzar sobre los píxeles de la fila, si el gris está dentro del rango se agrega a la cadena y se actualiza el rango. Repetir para los grises sucesivos que cumplan la condición.
   c) Calcular la longitud de ejecución mediante la diferencia de columnas final e inicial.
   d) Guardar en I_{comp} el ancho de ventana final y el promedio de grises en la ejecución.
6. Verificar una la compresión exitosa (la suma de las cantidades de grises ha de ser igual al producto MxN.
7. Guardar I_{comp} en un archivo binario.

Los pasos necesarios para cargar un archivo comprimido y reconstruir la imagen guardada, pueden enunciarse como:

1. Cargar el contenido del archivo binario en un vector.
2. Recuperar los bytes alto y bajo que indican la cantidad de filas, y calcular su equivalente en decimal. Repetir para obtener la cantidad de columnas.
3. Inicializar en cero la matriz I_{desc} que alojará la imagen descomprimida.
4. Para cada fila, los pasos que permiten la descompresión aplicados a cada binomio serían:
   a) Obtener del vector comprimido la longitud de ejecución y el gris medio.
   b) Calcular el gris mínimo, el gris máximo y la diferencia en la ventana bajo análisis.

c) Generar valores de gris aleatorios dentro del rango especificado, para la cantidad de columnas especificadas.

5. (Opcionales del programa) Calcular el tamaño en Kb del archivo comprimido dividiendo la cantidad de elementos en I_{comp} por 1024. Cargar la imagen original y calcular su tamaño en Kb. Calcular la razón de compresión.
6. (Opcionales del programa) Desplegar la imagen original y la reconstruída.

Ejercicio 4: Compresión JPEG

Este ejercicio tiene como objetivo analizar los pasos más destacados del algoritmo JPEG, que logra la compresión utilizando las características de la DCT en las imágenes. Una versión básica del algoritmo está implementada en el programa `'jpeg.cpp'` y será suministrado por la cátedra. Se requiere que analice el resultado obtenido al variar la cantidad de coeficientes utilizados para la representación, y realice un seguimiento de la calidad subjetiva de la imagen y de la tasa de compresión a medida que se quitan componentes de la DCT en el proceso.

En el programa `'jpeg.cpp'` se implementan las siguientes operaciones:

1. Se carga la imagen especificada (de tamaño 256x256), convirtiendo los valores al rango [0,1]. Se sustrae 0.5 a cada píxel para centrar los valores en el rango [-0.5,0.5].
2. Se divide la imagen en bloques no solapados de 8x8 (obteniendo $32^2 = 1024$ bloques). Se calcula la DCT de cada bloque. La información obtenida contiene 1024 muestras para cada uno de los 64 coeficientes de la DCT.
3. Las características de la imagen, ahora representadas mediante la DCT están concentradas en algunos pocos de esos 64 coeficientes. En consecuencia, la varianza de los coeficientes DCT determinará que información es más significativa y permitirá eliminar aquellos coeficientes con poca información. La varianza se calcula para cada uno de los 64 coeficiente sobre las 1024 muestras.
4. Los K coeficientes con poca información pueden ser eliminados. Aquí se introduce pérdida de información, incrementando la tasa de compresión. La imagen $\hat{f}$ se reconstruye con los 64-K coeficientes DCT.

Para este ejercicio deberá utilizar el programa `'jpeg.cpp'` y analizar el resultado obtenido al eliminar distintas cantidades de coeficientes. Elegir dos imágenes, una imagen real (reescalada a un tamaño de 256x256) y una imagen aleatoria. Con éstas se deben realizar las siguientes acciones:

1. Compilar el programa `'jpeg.cpp'` (obtendrá ayuda con `'jpeg -h'`).
2. Para la imagen real, visualizar y evaluar la imagen reconstruída con distintos valores de K, con $K \in \{1, 2, ..., 63\}$. Utilizar los siguientes criterios de evaluación:
   a) Subjetivo: observar el grado de deterioro sufrido en la imagen reconstruída.
   b) Objetivo: calcular el ECM como

$$ECM = \sqrt{\frac{1}{MN} \sum_x \sum_y \left[f(x,y) - \hat{f}(x,y) \right]^2}$$

c) Grafique el parámetro ECM en función de K.

3. Repetir el ejercicio con la imagen aleatoria y comparar los resultados con los obtenidos para la imagen real (comparar calidad de las imágenes para un mismo K y las características de la curva de error).