ATIPI - Proyecto de Evaluación
Compresión de imágenes de tono gris

Felipe Tambasco - Mauro Barbosa

Marzo de 2017

Índice

1. Resumen

Este trabajo consiste en la implementación de un algoritmo de compresión y descompresión de imágenes en software. Se implementará un compresor/descompresor sin pérdidas para imágenes de tono gris (8 bpp). El algoritmo de compresión a utilizar es una variante de LOCO-I.

2. Pautas del algoritmo de compresión

El algoritmo cuenta con los siguientes elementos:

- El mismo predictor fijo MED utilizado en JPEG-LS pero sin parte adaptativa. No se usará modo de *RUN*, por lo tanto, el contexto [0,0,0] es un contexto común de codificación.

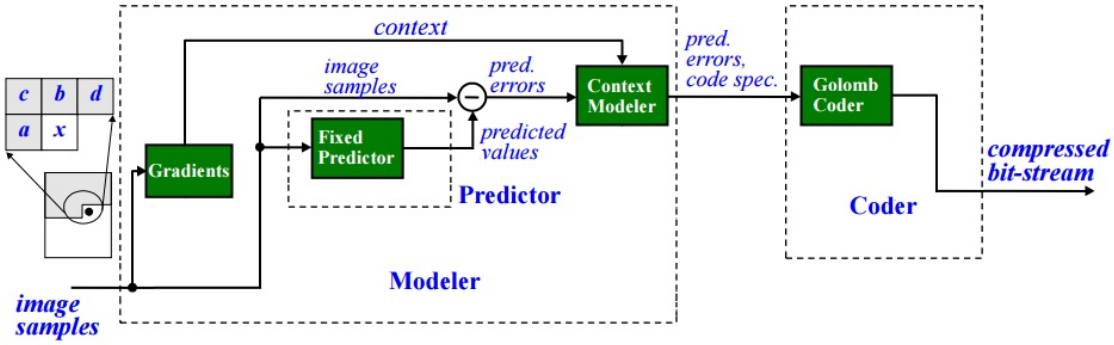

Figura 1: Diagrama de bloques

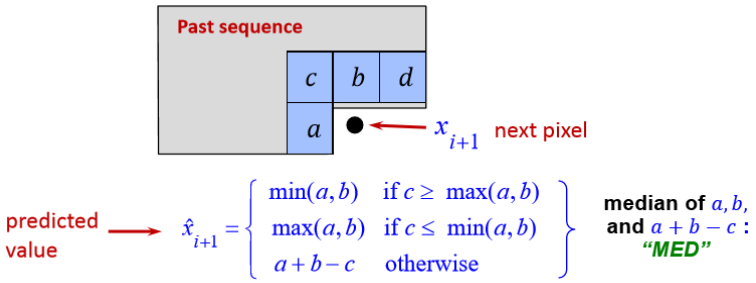

Figura 2: Pixeles y predicción. Tomado de las transparencias de clase.

- En la letra del laboratorio aconseja suponer que toda muestra que cae fuera de la imagen tiene valor 128, pero luego de analizar el tipo de imágenes a procesar, nos tomamos la libertad de aprovechar que el *Nivel de Blanco* es un dato conocido (Ver sección 3). Decidimos que los píxeles que rodean la imagen tengan el valor

$$x = \frac{NiveldeBlanco}{2} \tag{1}$$

- El modelo de contextos se construye de la manera siguiente: sean a,b y c las muestras vecinas a la muestra a comprimir x, según se define en las transparencias del curso (Figura 2). Definimos

$$p = \left[\frac{2a + 2b + 2c + 3}{6} \right] \tag{2}$$

y las diferencias

$$g_a = a - p,\ g_b = b - p,\ g_c = c - p \tag{3}$$

Las diferencias g_a y g_b se cuantifican igual que en LOCO-I (9 regiones simétricas). La diferencia g_c se cuantifica en 5 regiones simétricas que tendremos que definir. Con las diferencias cuantificadas, se forma el vector $q = [q_a, q_b, q_c]$, que se utiliza como contexto de cuantificación.

- Se codifican errores de predicción, utilizando códigos de Golomb de orden $m = 2^k, k = 0, 1, 2, ...$, y utilizando la función

$$M(e) = \begin{cases} -2e - 1 & e < 0 \\ 2e & e \geq 0 \end{cases} \tag{4}$$

para trasladar errores de predicci'on $e = 0, -1, 1, -2, 2, ...$ al rango no negativo. No se utilizan funciones alternativas y no se trata el caso $k = 0$ de forma especial.

- En cada contexto se determina el parámetro k del código de Golomb basado en dos estadísticas: A (suma de valores absolutos de errores) y N (número de muestras), con valores iniciales $A = 8, N = 1$. Procesamiento de la muestra x_i:
  - Se determina k en función de A y N como en los contextos regulares de JPEG-LS. Se codifica $x_i - \hat{x}_i$ con el código de Golomb de orden $m = 2^k$.
  - Actualización de estadísticas: Si $N == N_{max}$ (N_{max} es un parámetro de diseño),

$$\begin{array}{rcl} N & \leftarrow & N/2, \\ A & \leftarrow & A/2 \end{array} \tag{5}$$

Para cualquier N (independientemente de si se efectuó la actualización anterior):

$$\begin{array}{rcl} N & \leftarrow & N + 1, \\ A & \leftarrow & A + |x_i - \hat{x}_i|, \\ i & \leftarrow & i + 1 \end{array} \tag{6}$$

3. Tipo de imágenes a comprimir

Se trabajará sobre imágenes de formato PGM (Portable Graymap Mode). Son imágenes binarias, en escala de gris, a 8 bits per pixel.

Cada archivo PGM cuenta con un encabezado de tres líneas con la siguiente informaicón:

1. Un "Magic Number"que identifica el tipo de archivo. Esperamos imágenes con el Magic Number $P5$.
2. Ancho y alto separados por un espacio. ASCII, decimal.
3. El valor máximo de gris (nivel de blanco). ASCII, decimal.

Los píxeles son enviados siguiendo al encabezado a razón de un byte por pixel.

4. Resumen del algoritmo

4.1. Compresor

1. Obtener los píxeles de contexto a, b y c junto al pixel x.
2. Cálculo de p y las diferencias (ecuación 3).
3. Cálculo del contexto.
4. Obtener predicción $x_{pred} = pred(a, b, c)$.
5. $\epsilon = x_i - \hat{x}_i$.
6. Estimar k para ese contexto
7. Mapear ϵ utilizando la ecuación de Rice (ecuación 4) .
8. Codificar $M(\epsilon)$ usando Golomb-PO2.

4.2. Descompresor

1. Obtener los píxeles de contexto a, b y c junto al pixel x.
2. Cálculo de p y las diferencias (ecuación 3).
3. Cálculo del contexto.
4. Obtener predicción $x_{pred} = pred(a, b, c)$.
5. Calcular k para ese contexto.
6. Leer k bits. *Parte binaria.*
7. Leer la cantidad de ceros hasta el próximo uno. *Parte unaria.*
8. Obtengo $M(\epsilon) = binary + unary * 2^k$.
9. Deshago el Mapeo de Rice para obtener ϵ.
10. $x_i = \epsilon + x_{pred}$.

5. Implementación

Se implementó el algoritmo de compresión y descompresión descripto en la sección anterior. Se hicieron pruebas para distintos valores de N_{max} y variando los límites de las regiones para g_c. Se estudió en cada caso cómo cambia la relación de compresión para las imágenes propuestas. En base a los resultados se eligen valores para estos parámetros, como aquellos que optimizan el desempeño.

6. Análisis y diseño

6.1. Cuantificación y contextos

Los límites de cuantificación para clasificar los contextos son ajustables, con la excepción de que la región central debe ser cero. Aunque lo mejor sería modificar estos límites de forma adaptativa, la necesidad de bajar la complejidad nos lleva a dividir en regiones *equiprobables*. Si dividimos en 9 regiones, podemos usar los límites por defecto, que se corresponden a los que aplicamos a g_a y g_b:

$$[g < -21, -21 <= g < -7 \quad , -7 <= g < 0 \quad , g = 0 \quad , 0 < g <= 7 \quad , 7 < g <= 21 \quad , g > 21] \tag{7}$$

A pedido de la letra de este proyecto, se desea dividir la cuantificación de g_c en 5 regiones que puedan lograr una compresión aceptable en al menos las 9 imágenes de prueba (Baloons, Barbara, Bike, FaxballsL, Kodim07, Kodim20, Kodim24, Tools y Womanc).

Las regiones de g_c se pueden resumir de la siguiente forma:

$$[\quad g_c < -R \quad , \quad -R <= g_c < 0 \quad , \quad g_c = 0 \quad , \quad 0 < g_c <= R \quad , \quad g_c > R \quad] \tag{8}$$

6.2. N_{max} y Elección de las regiones cuantificación de g_c

Debido a las características diferentes de las imágenes de prueba, es difícil dividir en regiones equiprobables, por lo que realizamos un estudio estadístico.

6.2.1. Elección de N_{max}

Al analizar cómo cambia la relación de compresión al variar $Nmax$, nos encontramos con dos tipos de respuesta diferentes, que coincide con la aletoreidad y la naturaleza de las imágenes. Nos pareció apropiado dividir las imágenes en dos grupos, las *artificiales* (tipo 1) y *naturales* (tipo 2).

Las figuras 3 y 4 muestran la tasa de compresión según N_{max} para imágenes de tipo 1 y 2, con un valor de R arbitrario, fijo.

Las imágenes que llamamos *Naturales*, se corresponden con imágenes tomadas de la vida real, ricas en texturas, patrones, gradientes de gris; que además presentaron un comportamiento similar al variar N_{max} (figura 4). Todas ellas presentan buenas tasas de compresión para N_{max} grande.

Las imágenes que llamamos *Artificiales*, se corresponden a imágenes que presentan grandes zonas de un mismo tono, raras de encontrar en la naturaleza; también presentaron un comportamiento similar al variar N_{max} (figura 3). Presentan buenas tasas de compresión para N_{max} pequeño.

Para lograr un desempeño aceptable en los 2 tipos de imágenes, decidimos fijar el valor

$$N_{max} = 64 \tag{9}$$

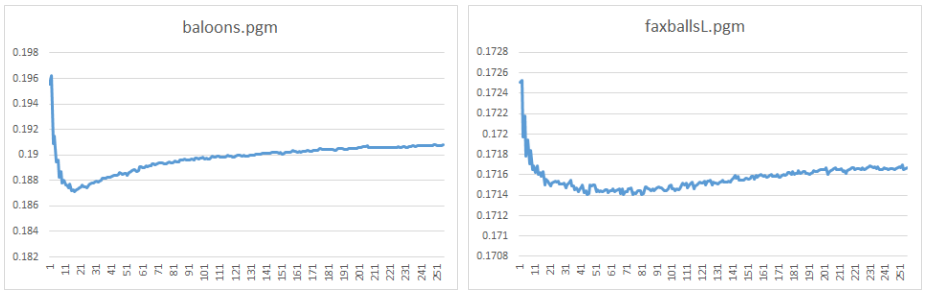

Figura 3: Tipo 1 - Imágenes Artificiales

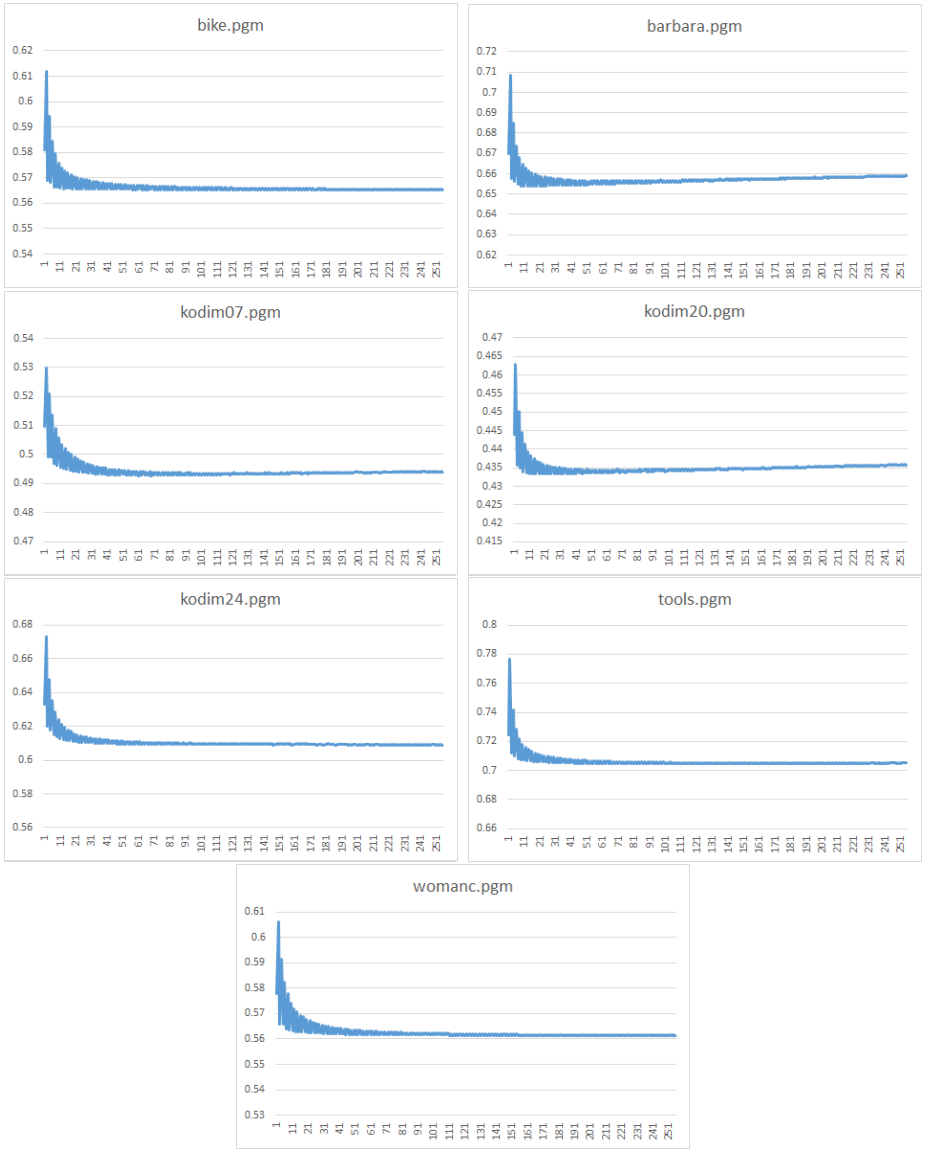

Figura 4: Tipo 2 - Imágenes Naturales

6.2.2. Elección de R

Ahora que fijamos nuestro N_{max} (ecuación 9), estudiaremos el impacto de R en las tasas de compresión.

La figura 5 muestra como varía la tasa de compresión para el N_{max} fijado cambiando R en cuatro imágenes que entendimos representativas de ambos Tipos.

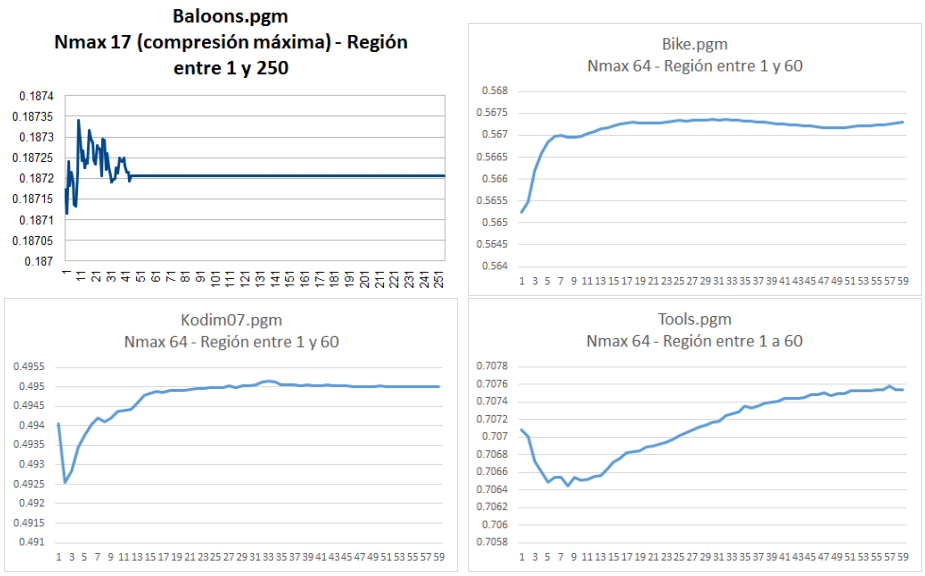

Figura 5: $N_{max} = 64$ - Variando R

A primera vista, podemos ver que las imágenes *Naturales* presentan mejores tasas de compresión para R bajo, y que por el contrario, la imagen *Artificial* se comportó de forma desordenada siendo más efectivo utilizar R grande o ninguno.

Pero no es menor notar que en la primera de ellas, la variación en la tasa de compresión fue como mucho de un $0,02\,\%$ (despreciable) permitiéndonos elegir el R basándonos en el resto de las imágenes.

Dado la información disponibles hasta el momento, decidimos fijar

$$R = 3 \tag{10}$$

6.3. ¿Por qué g_c con menos regiones?

Buscamos que el costo del modelo sea lo más bajo posible, ya que aumentar la cantidad de parámetros juega en contra de lo ganado por el estudio de la entropía. Por defecto, los tres gradientes g_a, g_b y g_c son cuantizados en 9 regiones cada uno. La cantidad de contextos que esto genera es además disminuída al hacer un *merge* entre los contextos de signo opuesto, quedando un total de $(9^3 + 1)/2 = 365$ contextos.

En nuestro caso no hicimos el *merge* de contextos de signo opuesto, por lo que la cantidad de ellos aumenta a 729. Disminuir éste número ayudaría a quitar complejidad.

Si cuantizamos uno de los gradientes en 5 regiones, obtendríamos $9 * 9 * 5 = 405$ contextos, pero para esto es conveniente optar por el que menos afecte a las tasas de compresión.

Debido a que en general se puede apreciar un nivel de dependencia mayor entre pixeles vertical u horizontalmente ubicados, la mejor opción sería manipular la cuantización del pixel de la diagonal.

A modo de ilustrar, graficamos la tasa de compresión según N_{max} para varios valores de R diferentes (figura 6). Donde puede verse que las variació en la relación de compresión es insignificante para distintos valores de R.

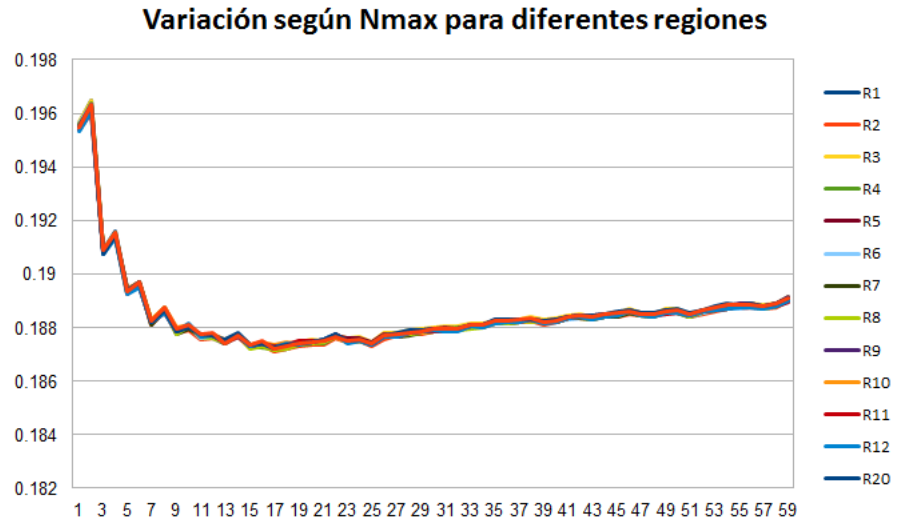

Figura 6: Baloons.pgm - $N_{max} = 64$ - Variando R

6.4. Función de N_{max} en la actualización de estadísticas de contexto

La función de N_{max} es, de alguna forma, controlar que las estadísticas del compresor no crezcan demasiado; perjudicaría el desempeño global del algoritmo. Ya que el parámetro k, que ajusta el largo de código del error a escribir, se calcula como el logaritmo del acumulado de errores absolutos (A) entre la cantidad de muestras (N). Si tanto A, como N pudieran crecer sin límites, esto incidiría fuertemente, y negativamente, sobre el valor de k, ya que si luego de estar en una zona con determinadas características, se pasa a otra, sería más difícil aprender cómo adaptarse a esta, y el algoritmo respondería con un aprendizaje que se arrastra de otra sección de la imagen, que no es igual de bueno para esta.

Tratándose de imágenes que en principio podrían tener una composición muy diversa, es bueno que el aprendizaje se dé más bien sobre el comportamiento local que se está codificando, y evitar que se impongan características de otras zonas de la imagen.

7. Resultados

7.1. Tasas dados los N_{max} pedidos en la tarea

En la figura 7 se reune la relación de compresión en bits por pixel (bpp) para cada imagen de prueba al variar N_{max}, con los valores diseñados en los límites de región de g_c.

	Original	Nmax = 4		Nmax = 8		Nmax = 64		Nmax = 256	
	(Bytes)	(Bytes)	(bpp)	(Bytes)	(bpp)	(Bytes)	(bpp)	(Bytes)	(bpp)
Ballons	328739	62972	1.532	62050	1.510	62208	1.514	62748	1.527
Barbara	262159	179444	5.476	174961	5.339	172023	5.249	172650	5.269
Bike	5242897	3111869	4.748	3034616	4.630	2968470	4.530	2962051	4.520
FaxballsL	524304	90185	1.376	90029	1.374	89865	1.371	89984	1.373
Kodim07	393231	204281	4.156	199577	4.060	193802	3.943	193732	3.941
kodim20	393231	176980	3.601	173549	3.531	170700	3.473	171367	3.486
kodim24	393231	254460	5.177	246814	5.021	239839	4.879	239325	4.869
tool	1828817	1356579	5.934	1321156	5.779	1292474	5.654	1290591	5.646
womanc	5242897	3100458	4.731	3027306	4.619	2945708	4.495	2937645	4.482

Figura 7: Tasas de compresión para $N_{max} = 4$, 8, 64 y 256

Los colores de la tabla fueron agregados para visualizar de forma rápida los diferentes escenarios según cada N_{max}. Cuanto más verde la tasa fue mayor.

La figura 7 nos permite corroborar que la elección de $N_{max} = 64$, aunque no logra la mejor tasa de compresión en 5 de las imágenes, es la que permite una tasa de compresión más universal dado el grupo de imágenes de testeo.

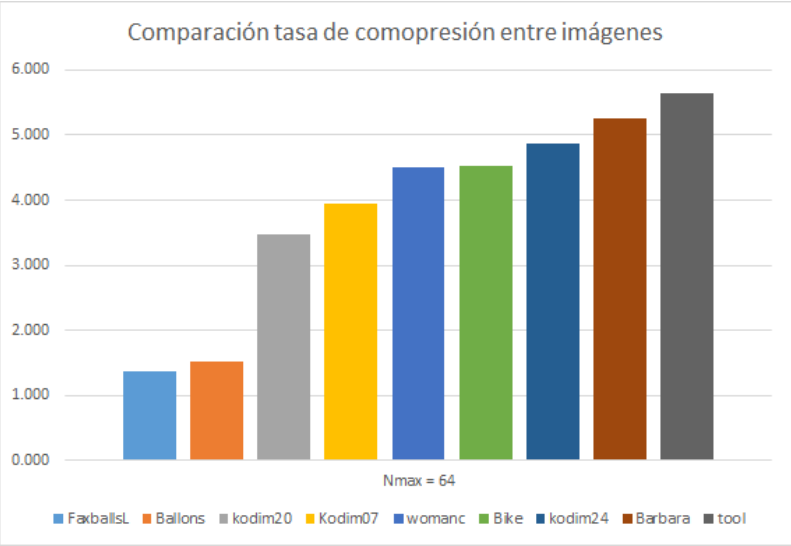

Figura 8: Comparación de tasas de compresión en bpp entre imágenes

7.2. Comparación con JPEG-LS

	Original	Nmax = 64		JPEG-LS	
	(Bytes)	(Bytes)	(bpp)	(Bytes)	(bpp)
Ballons	328739	62208	1.514	15375	0.374
Barbara	262159	172023	5.249	159340	4.862
Bike	5242897	2968470	4.530	2749295	4.195
FaxballsL	524304	89865	1.371	53232	0.812
Kodim07	393231	193802	3.943	177279	3.607
kodim20	393231	170700	3.473	154263	3.138
kodim24	393231	239839	4.879	226306	4.604
tool	1828817	1292474	5.654	1235563	5.405
womanc	5242897	2945708	4.495	2767857	4.223

Figura 9: Comparación de tasas de compresión con JPEG-LS

Si colocamos los datos en un gráfico de barras, podemos ver en qué imágenes nuestro diseño fue menos performante que el algoritmo JPEG-LS.

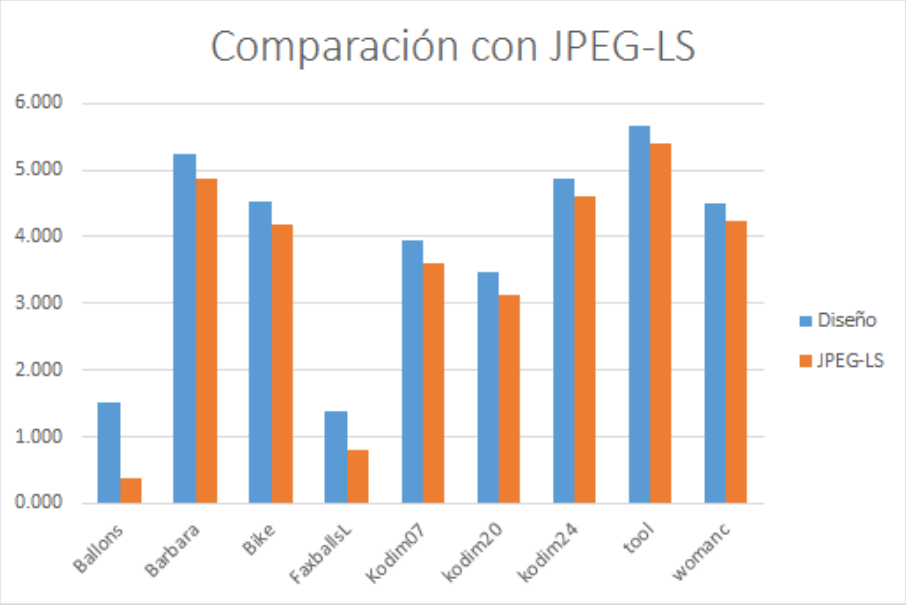

Figura 10: Comparación de tasas de compresión con JPEG-LS

La mayor diferencia se puede ver en las imágenes Artificiales, esto suponemos que es debido a la ausencia de RUN en nuestro código.

8. Conclusiones

Las imágenes *naturales*, y aquellas producidas artificialmente, tienen distintas características estadísticas, que se explica por su composición. En las imágenes artificiales pueden producirse extensas zonas de color uniforme, y luego un cambio

abrupto a otro color totalmente distinto. Esto no se da en las naturales ya que la transición es más suave entre píxeles adyacentes, y no existen colores uniformes en la naturaleza, sino que en lo aparentemente uniforme siempre hay imperfecciones y texturas. Por lo tanto, en las artificiales, la predicción es excelente en las zonas planas, pero muy mala en las transiciones, donde codifica un error con una cantidad muy grande de bits. Todo esto hace que el algoritmo responda distinto a cada grupo de imágenes, y que cambie en cada caso el conjunto de parámetros que optimizan el rendimiento.

Como se mostró en las tablas de la sección de resultados, no es posible definir un valor óptimo de N_{max} para el que todas las imágenes muestren una relación de compresión óptima. Este valor cambia incluso para imágenes del mismo tipo. Por lo que es necesario definir un valor de N_{max} en el que se logre un mejor desempeño en media.

Por las diferencias obtenidas con el algoritmo JPEG-LS, donde el desempeño de este siempre es mejor, vemos que el modo RUN es muy efectivo en imágenes artificiales con extensas zonas de color uniforme, como en baloons.pgm. En el resto de las imágenes donde la optimalidad del modo RUN no es tan evidente, la ventaja de JPEG-LS puede deberse a otros detalles no implementados, como la parte adaptiva del predictor, o las 9 zonas de cuantificación de g_c. Cualquiera sea el caso, es claro que estos aspectos no implementados logran un mejor desempeño en el algoritmo.

Contrario a lo que podía parecer intuitivamente, la relación de un pixel con el que se encuentra en su diagonal es casi nula. Tanto es así que variar los límites en las regiones de g_c tiene poco efecto en la relación de compresión del archivo. Aunque, esto podría cambiar si se consideran los aspectos no implementados de JPEG-LS, como el modo RUN.

A modo general, la predicción basada en estadísticas de contextos y la codificación de errores, resulta una poderosa herramienta de compresión. Alcanzando, en el mejor caso de las imágenes de prueba, una relación de 1.371 bpp para el algoritmo implementado[1], en faxballsL.pgm, y una relación de 0.374 bpp en JPEG-LS para baloons.pgm. Por lo que, no obstante la relativa sencillez conceptual del algoritmo, así como su fácil implementación, es un método muy eficiente de compresión de imágenes.

[1]Con N_{max} fijo en 64.

9. ANEXO - Descripción del material entregado

9.1. Comentarios sobre el código

La clases Image y CodedImage modelan una imagen, o imagen codificada, según el caso. En cada una de ellas, se carga toda la imagen en un array de enteros, o chars, para trabajar con la imagen de forma más cómoda. Esto es realizable por el tamaño reducido de las imágenes de prueba. Si las mismas fueran más grandes (cientos de MB, o algunos GB) esto no sería posible, ya que agotaría la memoria del computador, y sería necesario leer la imagen a medida que se codifica, o decodifica, según sea el caso. Por simplicidad, se adoptó esta solución.

En el codificador se reciclaron un par de métodos hechos para la tarea de Compresión de Datos Sin Pérdida (CDSP). Ya que el código generado en cada pixel es de largo variable, este tiene que ser almacenado en alguna estructura interna del programa hasta tener la cantidad suficiente de bits y poder escribirlos, de a 8. Este problema ya se había resuelto en las tareas de CDSP, por uno de los integrantes del equipo, por lo que, pareció oportuno reciclar la solución. En concreto, se refiere a los métodos writeCode() y flushEncoder().

9.2. Contenido

Entregamos un archivo comprimido con el siguiente contenido:

- **Src**: Directorio que contiene el código fuente
- **Doc**: Directorio que contiene la documentación en formato HTML.
- **Bin**: Directorio que contiene los ejecutables:
  - Codificador: Ejecutable de Linux
  - WCodificador.exe: Ejecutable de Windows 64 bits

9.3. Documentación

9.3.1. Doxygen

El software Doxygen lee un proyecto de lenguaje C++ (y otros) y en base a los comentarios de clases y archivos arma un html que permite navegar por el código fuente. Nos pareció enriquecedor agregar dicha información.

Para leer esta información se debe acceder al archivo *Index.html* dentro del directorio *Doc*.

9.3.2. Modo de uso de los ejecutables

La forma de uso es la siguiente:

Codificador <opción> <ruta y nombre de archivo> <valor de Nmax>
Las opciones disponibles son:

"–c" Para Comprimir el archivo
"–d" Para Descomprimir el archvio
"–cd" Comprime y Descomprime el archivo

Las tres opciones generan nuevos archivos de salida en la misma carpeta donde se encuentra el original. Se le agrega "$\_coded\_Nmax\_ < valordeNmaxusado > \_region\_3$" al archivo comprimido y "$\_decoded\_.pgm$" al rchivo Descomprimido. En ningún caso sobreescribe el original.