UNIVERSIDAD DE DEUSTO

UN NUEVO ALGORITMO DE COMPRESIÓN DE IMÁGENES DIGITALES SIN PÉRDIDA DE DATOS – ALGORITMO INA -

Tesis doctoral presentada por Juan Ignacio Larrauri

Programa de Doctorado en Ingeniería Informática y Telecomunicación

Dirigida por Dr. Javier García Zubía

El Doctorando — El Director

JUAN IGNACIO LARRAURI VILLAMOR — DR. JAVIER GARCÍA ZUBÍA

Bilbao, Julio de 2014

Resumen

El objetivo de esta tesis es el desarrollo y presentación de un nuevo método de compresión de imágenes estáticas sin pérdida de datos que permite obtener mayores ratios de compresión que los métodos universales. Este algoritmo es aplicado a imágenes fijas bidimensionales y de continuos tonos en escala de grises y en color. La mayoría de los métodos tradicionales sin pérdida de datos utilizan técnicas de eliminación o reducción de la redundancia existente en los datos (píxeles), aplicando métodos basados en modelos estadísticos (Huffman Coding, Arithmetic Coding, ...), modelos de diccionario (LZ77, LZW, etc.), modelos de predicción (FELICS, JPEG, etc.) o modelos basados en transformadas Wavelets. Los ratios de compresión alcanzados por estos métodos en imágenes fotográficas oscilan desde 1,5:1 a 2:1, considerándose estos últimos como resultados muy aceptables.

Alternativamente, proponemos un nuevo método basado fundamentalmente en tres procesos consecutivos: segmentación, estructura de árbol binario y codificación transversal. El primer proceso consiste en la segmentación de la imagen en bloques de píxeles de longitud fija o variable, el segundo proceso realiza el tratamiento de los píxeles del cada bloque obtenidos en el proceso anterior mediante un nuevo algoritmo de compresión basado en estructura de datos en forma de árbol binario y por último, una codificación transversal de la estructura de árbol genera los códigos binarios de salida. La unidad de tratamiento de la imagen es el bloque y por cada bloque se ejecutan los tres procesos secuencialmente hasta procesar la imagen completa Este método, en el caso de imágenes fotográficas de continuos tonos, se aproxima a ratios de compresión de 2:1.

El alcance del algoritmo ha sido evaluado utilizando un conjunto de imágenes estándares de diferente fuente o naturaleza (fotográficas, satélite, médicas, texto escaneadas, etc.). Los resultados experimentales han sido comparados con los métodos universales y especialmente con respecto al método JPEG-LS (ISO/IEC 14495, 2000) propuesto como estándar.

La implementación de este método en una plataforma PC se realiza completamente en niveles lógicos de programación, sin requisitos de dispositivos de hardware adicionales. Adicionalmente, el algoritmo propuesto podría ser implementado en plataformas hardware y sistemas embebidos sin necesidad de ningún cambio de funcionalidad.

Abstract

The main purpose of this thesis is the development and presentation of a new lossless compression method for still images that allows to achieve higher compression ratios than universal methods. This algorithm is applied to still images with two-dimensional continuous-tone grayscale and color. Most traditional methods use lossless techniques in order to remove or reduce the existing redundancy in the data (pixels), using methods based on statistical models (Huffman Coding, Arithmetic Coding, ...) dictionary models (LZ77, LZW, etc.), prediction models (FELICS, JPEG, etc.) or based on Wavelet transform models. The compression ratios achieved by these methods in photographic images range from 1.5:1 to 2:1, the latter being very acceptable results.

Alternatively, we propose a new method mainly based on three consecutive processes: segmentation, binary tree structure and traversal encoding. The first process consists on the segmentation of the image into blocks of pixels of fixed or variable length; the second process treats the pixels of each block obtained in the above process using a new compression algorithm based on a data structure in binary tree form; and finally, the traversal encoding of the tree structure generates binary output codes. The image processing unit is the block and the three processes are sequentially executed for each block until the whole image is processed. This method, in the case of continuous-tone photographic images, is close to 2:1 compression ratios.

The scope of the algorithm has been evaluated using a set of standard images from different sources or nature (photographic, satellite, medical, scanned text, etc.). Experimental results have been compared with the universal methods and especially with respect to the proposed JPEG-LS standard.

The implementation of this method on a PC platform is successfully performed in logic programming levels, with no requirement for additional hardware devices. In addition, the proposed algorithm could be implemented in any hardware and embedded systems platforms without any change in this functionality.

Laburpena

Tesiaren helburua da irudi estatikoak ulertzeko metodo berri bat garatzea eta aurkeztea, datuen galerarik izan gabe, eta metodo unibertsalek baino ulermen ratio hobeak izateko modua ematen duena. Algoritmo hori irudi finko bidimentsionalei eta grisen eta koloreen eskalako tonu jarraituei aplikatzen zaie. Datuen galerarik gabeko ohiko metodorik gehienek datuetan (pixelak) dauden erredundantzia murrizteko edo kentzeko teknikak erabiltzen dituzte: eredu estatistikoetan oinarritutako metodoak (Huffman Coding, Arithmetic Coding, etab.), hiztegietako metodoak (LZ77, LZW, etab.), predikzio ereduak (FELICS, JPEG, etab.) edo Wavelet moldatuetan oinarritutako ereduak aplikatzen dituzte. Metodo horiek argazkietan lortu dituzten ulermen ratioak 1,5:1 eta 2:1 bitartekoak dira, emaitza onargarriak.

Horien ordez, metodo berri bat proposatzen dugu, funtsean elkarren segidako hiru prozesutan oinarritua: segmentazioan, zuhaitz bitar egituran eta zeharkako kodetzean. Lehenengo prozesua honetan datza: irudia segmentatzen da luzera finko edo aldakorreko pixelen blokeetan. Bigarren prozesuan, aurreko prozesuan lortutako bloke bakoitzeko pixelak tratatzen dira, zuhaitz bitar itxurako datuen egituran oinarritutako beste ulermen algoritmo baten bidez. Eta, azkenik, zuhaitzaren egituraren zeharkako kodetzeak irteerako kode bitarrak sortzen ditu. Irudia tratatzeko unitatea blokea da, eta bloke bakoitzeko hiru prozesuak exekutatzen dira sekuentziari jarraituz, irudi osoa prozesatu arte. Argazkien tonu jarraituen kasuan, metodoa 2:1 ulermen ratioetara hurbiltzen da.

Algoritmoaren irismena iturri edo izaera desberdineko irudi estandar multzo bat (argazkiak, satelitekoak, medikoak, testu eskaneatuak, etab.) erabiliz ebaluatu da. Esperimentuen emaitzak metodo unibertsalenetakoekin erkatu dira, bereziki estandar gisa proposatzen den JPEG.LS metodoarekin.

Metodoa PC plataforma batean ezartzea programazioko maila logikoetan egiten da osorik, bestelako hardwareko gailurik gabe. Horrekin batera, proposatutako algoritmoa ezar liteke hardware plataformetan eta sistema txertatuetan ere, funtzionalitaterik aldatu beharrik izan gabe.

Agradecimientos

En primer lugar, deseo expresar mi agradecimiento a mi director de tesis Dr. Javier García Zubía y al Dr. Evaristo Kahoraho Bukubiye, quienes con su valioso apoyo, colaboración y dirección técnica han hecho posible esta tesis.

Así mismo, agradezco el respaldo recibido por parte de la Facultad de Ingeniería de la Universidad de Deusto durante estos años y especialmente por la oportunidad que me ofrece para desarrollar las actividades que siempre he deseado realizar: aprendizaje y enseñanza. Además, quisiera añadir mi agradecimiento por todo el apoyo prestado y por los medios y los recursos necesarios recibidos durante estos años que han hecho posible el desarrollo de esta disertación.

También deseo expresar mi más sincero agradecimiento a la Universidad de Portsmouth en UK y especialmente al Dr. David L. Ndzi por las facilidades y asesoramiento recibidos durante los tres meses de mi estancia en esta Universidad.

Finalmente, deseo expresar mi gratitud a mi esposa, Ana e hijos: Aitor y Jon, por su sacrificio y comprensión durante estos años, y especialmente a mis padres por su inestimable ayuda e impulso recibido durante toda mi formación personal y académica.

Índice general

Índice de figuras

Índice de tablas

Índice de listados

Motivación y punto de vista personal del doctorando

Recuerdo con nostalgia cuando me gradué en Informática en la Facultad de Informática de la Universidad de Deusto en 1984. En aquellos momentos, los fabricantes de ordenadores comenzaban a incluir las primeras tarjetas gráficas en los ordenadores personales: Hércules, CGA y EGA. La imagen no tenía una importancia relevante en la representación de la información en los ordenadores. Los resultados eran visualizados en la pantalla del ordenador en forma textual o mediante gráficos generados con códigos ASCII. Las unidades de almacenamiento externa disponían de 360 kB o 1,2 MB de capacidad en formato de discos flexibles de $5^{1/4}$" y la capacidad en soporte de disco duro era de 10 o 20 MB.

Posteriormente, con la aparición de las tarjetas gráficas VGA y SVGA comenzaron a surgir las primeras aplicaciones de procesamiento de imágenes digitales. Las unidades de almacenamiento en disco duro más frecuentes contenían desde 40 a 200 Mb. La necesidad de disponer de mayor capacidad de almacenamiento tuvo como consecuencia el origen de las primeras aplicaciones en Compresión de Datos. Estas aplicaciones estaban dirigidas exclusivamente a comprimir unidades de memoria externa.

Los continuos avances tecnológicos en la fabricación microelectrónica de dispositivos, *CPUs*, memorias (*CDROM, DVD*, dispositivos externos), escáneres analógicos y digitales, impresoras láser, redes de comunicaciones, etc., han invertido el proceso de representación de la información. Actualmente, la ***imagen digital*** es el resultado final de la mayoría de las aplicaciones por ordenador y en muchos casos los resultados en forma textual son transformados y representados como imágenes.

El término "Compresión de Datos" suena como una temática propia de investigadores, de difícil comprensión y fuera de un contexto real. Sin embargo, esta idea nada tiene que ver con la realidad. La compresión de datos está presente en muchos aspectos de nuestra vida laboral y personal. Todos los días miles de imágenes, correos electrónicos y faxes son distribuidos por la red internet o por redes de telefonía, y que gracias a la compresión de datos producen un ahorro en el coste de transmisión superior a 10 veces su coste.

Antes de iniciar esta tesis doctoral, mi anterior director, Dr. D. Evarsito Kahoraho, me entregó el libro titulado "Data Compression Book", escrito por Mark R. Nelson y cuyo contenido me cautivó e introdujo de forma activa en la compresión de datos. Desde entonces, para mí, la compresión de datos significa algo más que la aplicación de un conjunto de técnicas para conseguir reducir el espacio del almacenamiento de datos y los costes de transmisión. La compresión de datos consiste en analizar y descubrir nuevas propiedades y estructuras de datos ocultas en una imagen.

Capítulo

1

Objetivo y desarrollo de la presente tesis

Actualmente la imagen es un modo de presentación de la información que se combina con texto y otras fuentes. Buena parte de la información que nosotros recibimos nos llega de forma visual, y el modo de representarla, almacenarla y transmitirla constituye parte fundamental de nuestro entorno personal y profesional.

Vivimos en un mundo analógico donde cada día el mundo digital adquiere una mayor importancia. Nuevas formas de representación visual están emergiendo del uso y aplicación de las computadoras, desarrollando nuevas tecnologías multimedia, video-conferencias, aplicaciones educativas, publicidad, televisión etc., cuyo resultado final es una imagen.

Las necesidades de almacenamiento de información crecen de forma exponencial con respecto a los avances tecnológicos en la fabricación de dispositivos de almacenamiento. En el caso particular del procesamiento de imágenes, el problema aumenta porque la cantidad de memoria necesaria para almacenar una imagen siempre es superior al texto.

La solución al problema anterior pasa por almacenar comprimidas las imágenes. Así el cometido de un algoritmo de compresión es reducir el tamaño de la imagen a almacenar (Sedgewick y Wayne 2011). La imagen comprimida puede volver a su estado original para ser presentada al usuario mediante el

correspondiente algoritmo de descompresión. Este proceso de compresión-descompresión puede no modificar en absoluto la imagen (algoritmo sin pérdida de datos) o puede hacerlo de una forma mínima (algoritmo con pérdida de datos). La desventaja del segundo tipo de algoritmo se compensa con una mayor compresión en la imagen.

La forma de afrontar la compresión de imágenes es muy diversa, dando lugar a diferentes tipos de algoritmos con diferentes características. No existe el algoritmo perfecto, sino una colección de ellos con sus ventajas y desventajas. Esta situación deriva en una investigación muy activa.

Los algoritmos de compresión son utilizados por la industria en ordenadores, smartphones, equipo médico, etc., dando lugar a diversos estándares cuyo objetivo es explotar los resultados de investigación en aplicaciones industriales, así se dispone de JPEG (ISO/IEC 14495, 2000), etc. El paso de los algoritmos de investigación a algoritmos estándares no es siempre perfecto, y no pocas veces la industria plantea y utiliza estándares de compresión que están lejos de su mejor rendimiento.

La presente tesis incide en la situación anterior definiendo un nuevo algoritmo denominado INA que partiendo de un enfoque distinto aporta numerosas ventajas a la compresión digital de imágenes sin pérdida de datos.

Este capítulo describe la hipótesis de trabajo, los objetivos, la metodología y la estructura de esta tesis para facilitar la lectura y comprensión de los resultados presentados.

1.1 Hipótesis

La hipótesis inicial quedo establecida como sigue:

> Es posible diseñar e implementar un nuevo algoritmo en el área de la compresión de imágenes digitales estáticas de continuos tonos o color en dos dimensiones que mejore los ratios de compresión de los estándares actuales para acercarlos al estado-del-arte. Dicha mejora en los ratios de compresión no debe comprometer su rendimiento en términos de velocidad y consumo de recursos de computación

1.2 Objetivos específicos

La validación de la anterior hipótesis se despliega en los siguientes objetivos específicos:

- Estudio y análisis detallado y exhaustivo de los métodos de compresión de imágenes reversibles que actualmente suponen el "estado-del-arte" en compresión de imágenes estáticas de dos dimensiones. Esta tesis incluye también aquellos métodos que, aunque no garanticen factores de compresión máximos, han tenido una influencia o han supuesto un avance científico relevante en el campo de la compresión de datos.
- Estudio comparativo y evaluación de los métodos clasificados por su modelo de compresión. Inicialmente los métodos son clasificados según su modelo de compresión (estrategia de compresión) para luego ser analizados mediante los resultados experimentales obtenidos al comprimir y descomprimir un conjunto de imágenes estándares. Los ratios de compresión máximo, mínimo y medio son calculados para cada uno de los métodos.
- Implementación del algoritmo propuesto a nivel lógico (software). En este apartado se recogen las características del algoritmo en términos de portabilidad, eficiencia, tiempos de ejecución, complejidad, etc. Los resultados experimentales son contrastados con los métodos presentados y especialmente con el compresor reversible propuesto como estándar, JPEG-LS (ISO/IEC 14495,2000).

1.3 Metodología

Una vez definidos los objetivos específicos la metodología de trabajo surge de una forma casi espontánea. En primer lugar hay que describir qué criterios o índices esenciales en el comportamiento de los algoritmos de compresión, es decir, qué es comprimir y cómo se mide la capacidad de compresión o ratio de compresión.

Un elemento crítico en el proceso de compresión son las propias imágenes a procesar. La tesis utiliza como banco de pruebas un conjunto de 32 imágenes obtenidas de diversas fuentes: University of Southern California (USA), University of Waterloo (Canada), University of Canterbury, Canterbury image compression corpus (Nueva Zelanda) y URL de los métodos estándares y convencionales**.** Las 32 imágenes seleccionads se agrupan en cinco grandes grupos: Fotográficas (10 imágenes), Aéreas y de Satélite (8 imágenes), Creadas por Ordenador (4 imágenes), Médicas (6 imágenes) y Textos y Gráficos Escaneados (4 imágenes). Cada una de ellas, como su nombre indica, tiene unas características distintas de las otras y

pueden que sean más adecuadas para un método de compresión o para otro. En cualquier caso, el grupo de imágenes Fotográficas es el más problemático a la hora de comprimirlas ya que presenta continuos cambios de intensidad de los píxels (continuos tonos) , o dicho de otra forma, el valor de sus píxeles cambia de forma continua, complicando su compresión, hasta el punto de que algunos métodos no solo no comprimen las imágenes, sino que los expanden. Buena parte del análisis detallado de los métodos de compresión recaerá sobre este tipo de imágenes.

Seguidamente cada algoritmo será implementado, para luego procesar cada una de las imágenes del banco anterior de pruebas y obtener su ratio de compresión. Este proceso nos dará idea de la eficacia del algoritmo.

Además, es necesario medir cuánto consume el algoritmo implementado en términos de tiempo y de recursos de computadora, ya que esto nos dará una imagen de la eficiencia del algoritmo.

Cada método se describe en su totalidad: principio de funcionamiento, proceso de compresión y descompresión incorporando la descripción formal del algoritmo, flujogramas, codificación del algoritmo, ejemplos prácticos y resultados experimentales de cada uno de ellos, aplicando en todos ellos la misma metodología de evaluación.

El anterior proceso se seguirá también para el nuevo algoritmo implementado, INA, de forma que este pueda ser comparado con el estado del arte, especialmente con el estándar industrial, para determinar su valor.

Todos los algoritmos son implementados en la tesis, utilizando para todos ellos el mismo ordenador, sistema operativo y lenguaje de programación, y en ningún caso se usan versiones disponibles en la comunidad investigadora. Solo de esta forma se pueden extraer conclusiones de los resultados analizados.

1.4 Estructura de la tesis

La tesis está estructurada en cinco capítulos. En este primer capítulo se han descrito los objetivos y la metodología aplicada al desarrollo de esta tesis.

El Capítulo 2, titulado “Compresión de datos e imágenes”, aborda en sus primeras secciones una introducción que tiene por objeto situar la importancia de la compresión de datos en el presente y futuro del problema del almacenamiento de información. Seguidamente se presenta el marco teórico y un número importante de términos, conceptos y definiciones con respecto a la compresión de imágenes. Se exponen los principios en los cuales se fundamenta la compresión reversible de imágenes digitales, con especial hincapié en los modelos y arquitectura de los procesos de compresión y descompresión, y las medidas de compresión en las

cuales se expresan los resultados y con las cuales se comparan los algoritmos revisados en esta tesis.

El Capítulo 3, titulado "Estado del arte de la compresión de imágenes sin pérdida de datos", se centra en la exposición y evaluación de la compresión de imágenes sin pérdida de datos aplicada a imágenes fijas de continuos tonos en dos dimensiones. Este capítulo describe cada uno de los métodos mediante un estudio exhaustivo, reflejando los principales beneficios y los puntos críticos de cada uno de los métodos. La metodología de exposición y evaluación es común para todos los métodos analizados.

El Capítulo 4, titulado "Un nuevo algoritmo de compresión de imágenes sin pérdida de datos – INA", constituye el núcleo y principal aportación de esta investigación científica. En este capítulo se describe exhaustivamente el diseño, principio de funcionamiento e implementación del nuevo algoritmo. Los resultados experimentales son comparados con los métodos presentados en el capitulo anterior.

El Capítulo 5, titulado "Conclusiones y principales aportaciones", incluye un sumario con las principales conclusiones y contribuciones recogidas a lo largo de la tesis, señalándose las vías de futuras investigaciones más interesantes.

La memoria incluye un anexo que contiene la información sobre las imágenes que han servido para evaluar los algoritmos.

Capítulo

2

Compresión de datos y de imágenes

La compresión de datos por computador consiste en reducir el número de bits necesarios para almacenar o transmitir información. En esta definición están implícitos los términos *datos* e *información*, que con frecuencia son empleados indistintamente. Los datos son valores o símbolos concretos que unidos lógicamente (ordenadamente) componen la información. Por tanto, el proceso de compresión consiste en reducir los datos manteniendo la fidelidad a la información original.

Los métodos de compresión de datos se componen de dos ciclos: compresión y descompresión. En el primer ciclo, el tamaño de los datos de entrada es reducido para almacenarlos o transmitirlos posteriormente. En el segundo ciclo, los datos comprimidos se expanden para recuperar la información de entrada. El término método o técnica de compresión de datos implícitamente hace referencia a la ejecución de ambos ciclos (Nelson y Gailly, 1995) (Sayood, 2012), tal y como muestra la Fig. 2.1.

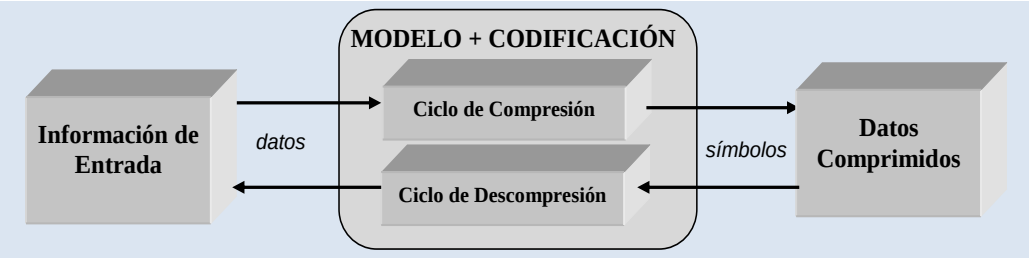


Figura 2.1. Diagrama de bloques simplificado de un método de compresión

Desde la aparición de los primeros computadores personales, en la década de los 80, hasta la actualidad han proliferado numerosas técnicas de compresión aplicadas en todas las áreas del conocimiento y aunque la compresión de datos está asociada a los computadores, las primeras técnicas de compresión surgen con anterioridad.

Este capítulo se divide en dos partes, según se aborde la compresión de datos o la de imágenes. En primer lugar, secciones 2.1 a 2.3, se introduce históricamente la compresión de datos, para luego formular el problema y abordar la necesidad de comprimir datos aportando los principales beneficios y desventajas, la clasificación de los métodos de compresión y una tabla resumen de los estándares actuales de compresión. Las secciones 2.4 a 2.7 abordan específicamente la problemática de la compresión de imágenes, describiendo las diferentes estrategias de compresión y los criterios para medir la eficacia de cada uno de ellos. La sección 2.7 incluye la clasificación de los métodos de compresión que seguirá la tesis para su análisis en el Capítulo 3.

2.1 Breve descripción histórica de la compresión de datos

A lo largo de la historia de la humanidad la compresión de datos siempre ha estado presente en el modo de representar la información. Las primeras técnicas de compresión conocidas en la *escritura* datan de la época de la antigua Grecia, siglo III a. de C. (Salomon y Motta, 2010). Los antiguos griegos para ahorrar en el coste del precio del manuscrito de papiro comprimían el texto mediante la omisión de los símbolos de puntuación y de los espacios entre palabras (Fig. 2.2), codificando la finalización de los párrafos mediante marcas fin de párrafo o el subrayado de la última línea del párrafo.

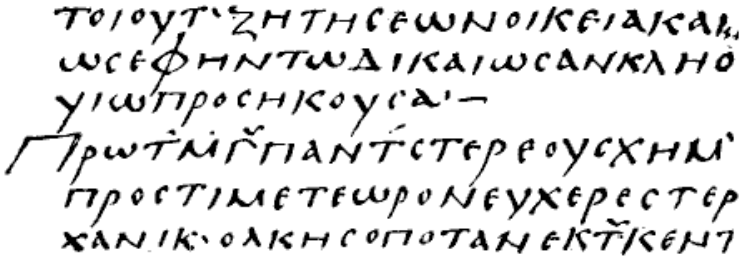


Figura 2.2. Acrónimos en papiro griego

En el *lenguaje natural*, los primeros acrónimos conocidos surgieron en la época del Imperio Romano. El uso de acrónimos reducía el coste de grabación de las tumbas romanas al comprimir o abreviar los nombres y las leyendas impresas en piedra. A esta época pertenece el acrónimo INRI (*Iesus Nazarenus Rex Iudaoerum*) cuya traducción del latín es "Jesús el Nazareno, rey de los judíos" y que la Religión

Católica ha utilizado a lo largo de los tiempos en la inscripción de los crucifijos. Estas mismas técnicas fueron utilizadas en firmas oficiales como SPQR (*Senatus Populusque Romanus*) referido al gobierno de la antigua republica romana y también en el diseño de monedas, reduciendo de esta forma considerablemente su tamaño. Estos acrónimos rápidamente fueron incorporados como términos en el lenguaje popular.

Figura 2.3. Acrónimos en monedas y tumbas romanas

En 1694, el científico Gottfried Wilhelm Leibniz (1646-1716) presenta un primer trabajo sobre *cálculo binario*. Leibniz construyó una máquina conocida como "calculadora universal" para resolver multiplicaciones y realizar teóricamente operaciones matemáticas. El sistema binario empleado usaba únicamente dos dígitos, 0 y 1, para representar cualquier número o carácter. Leibniz es reconocido como el padre del sistema binario. Este sistema se convertiría más tarde en la base de la ciencia de la computación y de la industria de los computadoras.

Figura 2.4. Gottfried Wilhelm Leibniz

En 1800, Joseph Marie Jacquard (1752-1834) inventó un telar automático basado en un sistema de codificación de tarjetas perforadas para reproducir copias automáticamente (Feldman y Ford, 1989). Las tarjetas perforadas se fundamentan en el uso de un código binario generado por la perforación de la ficha en alguna de sus campos.

Figura 2.5. Joseph Marie Jacquard y ficha perforada

En 1820, Louis Braille (1809-1852) esarrolló un sistema de codificación de lectura y escritura para personas ciegas que lleva su propio nombre. Actualmente sigue siendo utilizado aunque en versiones corregidas (Bryant, 1993). El método consiste en agrupar grupos o celdas en formato de 3 filas x 2 columnas mediante puntos en relieve. Cada uno de los seis puntos en un grupo puede ser plano o con relieve. La codificación Braille es un sistema de numeración.

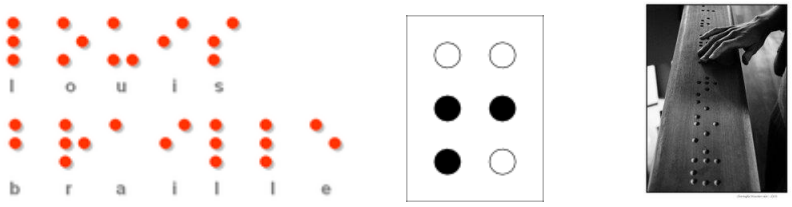


Figura 2.6 Muestra del alfabeto Braille y la máquina táctil Braille

En 1837, Samuel F.B. Morse aplicó la patente que lleva su nombre al nuevo telégrafo. Este sistema de *comunicación* es universalmente conocido como Código Morse . El sistema de codificación es utilizado para la transmisión de información y se fundamenta en la sustitución de las letras más utilizadas del alfabeto por símbolos más reducidos (Stephen, 2002). El código Morse utiliza secuencias estándares de su propio alfabeto formadas por pulsos largos y cortos para codificar datos alfanuméricos y simbolos de puntuación y especiales.

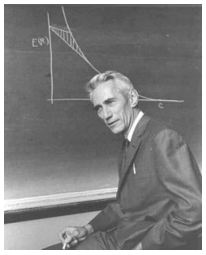
El código Morse utiliza un sistema binario (on/off – 0/1) para la transmisión de mensajes. El código Morse Internacional está formado por seis elementos:

- Marca corta (·) denominada punto
- Marca larga denominada (-)
- Caracteres internos
- Espacio corto (entre letras)
- Espacio medio (entre palabras)
- Espacio largo (entre frases)

A diferencia de los métodos anteriores y como característica especial, este sistema de codificación puede ser interpretado no solo de forma escríta sino también por el sonido y duración de los pulsos.

Figura 2.7. Samuel Morse y el emisor Morse

A finales de los años 1940, la Teoría Moderna de la Información impulsada principalmente por los trabajos de Claude E. Shannon describe las primeras técnicas de *codificación* de símbolos: La técnica de codificación es conocida como Shannon Coding (Shannon, 1948).



$$H(S) = -\sum_{i=1}^{n} P_i \log_2 P_i$$

Figura 2.8. Claude E. Shannon y la expresión del cálculo de la entropía

Estas técnicas reducen de forma eficaz el tamaño y el tiempo de transmisión de mensajes eliminando la redundancia existente en los datos basándose en la probabilidad de aparición de cada símbolo en el mensaje. Una optimización de Shannon Coding realizada por Fano R.M. y conocida como Shannon-Fano Coding (Fano, 1949) es considerada como la primera técnica de codificación de la Teoría de la Información Moderna. Posteriormente, el Codificador Huffman denominado Huffman Coding (Huffman, 1952) consiguió obtener mejores resultados con un sistema de codificación más eficiente. Estas técnicas de codificación han sido la base de los primeros métodos de compresión y de la mayoría de los métodos actuales, incluidos algunos estándares de compresión, como son: JPEG (*Joint Photographic Experts Group*) (ISO/IEC 14495, 2000). y JBIG (*Joint Bi-level Image Experts Group*) (ISO/IEC 11544, 1993).

En la actualidad, numerosos ejemplos de compresión de datos están presentes en nuestro entorno cotidiano. Estos sistemas no tienen una base científica sino que surgen de la necesidad de ahorrar tiempo y costes. La popularidad de estos métodos se ha extendido y distribuido rápidamente entre los jóvenes en diferentes contextos y pueden ser considerados como nuevos sistemas de codificación.

En el contexto de la escritura, es sorprendente observar como los estudiantes utilizan técnicas de compresión de forma eficaz para realizar sus propios apuntes y exámenes. En el lenguaje natural, los acrónimos forman parte de nuestro lenguaje, términos como, PC (*Personal Computer*), CPU (*Central Processing Unit*), IP (*Internet Protocol*), ISO (*International Organization for Standardisation*), etc. son habituales en el lenguaje científico y coloquial. En las comunicaciones, análogamente al código Morse, nuevas formas personales de reducción de tiempo y costes surgen espontáneamente en nuestras vidas, por ejemplo, una escritura abreviada de transmisión de mensajes por telefonía móvil en los cuales se suprimen letras que componen una palabra y son sobreentendidas en la expresión oral. La Tabla 2.1

refleja un resumen de los hitos más relevantes en la evolución histórica de la compresión de datos.

Tabla 2-1. Técnicas de compresión utilizadas a lo largo del tiempo

CONTEXTO	ÉPOCA	TÉCNICA
Escritura	Antigua Grecia	Eliminación de espacios
Lectura	Civilización Romana	Acrónimos
Cálculo	Siglo XVII	Código Binario
Comunicaciones y sonido	Siglo XIX	Código Morse
Ordenadores – entrada datos	Siglo XIX	Ficha perforada
Ordenadores	Siglo XX	Shannon-Fano Coding
Telefonía móvil	Siglo XXI	Mensajes móviles

El contenido de la tabla refleja como la compresión de datos a lo largo de los tiempos ha sido utilizada para reducir costes y tiempos. Desde la aparición de las computadoras y sistemas de telecomunicaciones la compresión de datos está presente en la representación de la información persiguiendo los mismos objetivos de siempre. Esta es una de las principales razones por las cuales la compresión de datos sigue vigente en las investigaciones científicas.

En la siguiente sección se describe la problemática actual que presenta la compresión de datos y la respuesta aportada por la comunidad científica e industrial.

2.2 Formulación del problema. La necesidad de comprimir datos

Los continuos avances tecnológicos proporcionan a los dispositivos de almacenamiento CD (*Compact Disc*), DVD (*Digital Video Disc*), USB (*Universal Serial Bus*) grandes cantidades de memoria y altas velocidades de procesamiento. Inicialmente, se puede pensar que la cantidad de memoria que suministran estas unidades es suficiente para resolver el problema del almacenamiento de la información. Sin embargo, esta idea está muy lejos de la realidad, la necesidad de almacenamiento de información crece de forma exponencial con respecto a los avances tecnológicos en la fabricación de dispositivos de almacenamiento.

La compresión de datos es una cuestión de tiempo y dinero. El coste del almacenamiento y transmisión digital son generalmente proporcionales a la cantidad de datos digitales que pueden ser almacenados o transmitidos. Por esta razón, analizaremos los principales beneficios de la compresión de datos en los distintos campos de aplicación.

- **Almacenamiento**: en el pasado, los documentos eran guardados en soporte papel en grandes superficies destinadas exclusivamente para almacenamiento. Estos sistemas de almacenamiento eran ineficientes en términos de almacenamiento y especialmente en los tiempos de recuperación de la información. La compresión de datos aporta una solución digital a este problema, optimizando el almacenamiento en memoria y la recuperación inmediata de la información.
- **Comunicaciones**: el ancho de banda de una comunicación digital puede ser incrementado mediante la compresión de datos en el emisor y la descompresión en el receptor. Cada vez que un fax o correo electrónico es enviado, al menos una técnica de compresión de datos se utiliza para comprimir los datos. Es decir, el tiempo de conexión telefónica es reducido y en consecuencia se disminuye considerable del coste de conexión.
- **Seguridad y protección de datos**: las técnicas de compresión de imágenes combinadas con técnicas de marca de agua (*Watermarking*) o con técnicas de encriptación garantizan una mayor seguridad de la información y de la protección de los derechos de autor respectivamente.

En la Tabla 2.2, se muestran los resultados que justifican la necesidad de comprimir datos en las principales aplicaciones.

Tabla 2-2. Aplicaciones típicas de compresión de datos

APLICACIÓN	RESOLUCIÓN	BITS/PÍXEL	TAMAÑO ARCHIVO COMPRIMIDO	ANCHO DE BANDA (KB/S)	TIEMPO DE TRANSMISIÓN (MODEM - 28.8 KB)
Una página de texto	29,69 cm x 21 cm	Resol. variable	4-8 kB	32-64 kB/página	1,1 – 2,2 s
Transmisión telefónica -Voz	10 s	8 bps	80 kB	64 kB /s	22,2 s
Imagen escala grises	512 x 512	8 bpp	256 kB	2,1 MB/imagen	1 m 13 s
Imagen color	512 x 512	24 bpp	786 kB	6,29 MB/imagen	3 m 39 s
Imágenes médicas	2048 x 1680	12 bpp	5,16 MB	41,3 MB/imagen	23 m 54 s
Imagen	2048 x 2048	24 bpp	12,58 MB	100 MB/imagen	58 m 15 s
Vídeo	640 x 480 (30 frames/s)	24 bpp	1,66 GB	221 MB/s	5 días y 8 h

La compresión de datos forma parte esencial de cualquier manifestación o presentación donde el computador esté presente. Internet no sería posible sin

técnicas de compresión de datos. La compresión de datos ya no es solo cuestión de tiempo y dinero sino también de operatividad.

2.3 Métodos de compresión de datos: *lossless* y *lossy*

En la compresión de datos no existe un método universal válido para alcanzar los mejores resultados de compresión en todas las diferentes clases de aplicaciones: texto, imagen, audio, vídeo y comunicaciones (Sayood, 2012). Los resultados de compresión están en función de las características particulares de cada aplicación. Un método de compresión que obtiene buenos resultados en las aplicaciones de compresión de textos o bases de datos generalmente no garantiza los mismos resultados en imagen o audio. La respuesta científica a lo largo del tiempo se ha direccionado hacia la idea de ofrecer soluciones particulares a cada clase de imágenes.

En este contexto, los comités de normalización han aportado numerosos estándares de compresión (ISO/IEC, 1991, 1993, 1999 y 2000). La gran cantidad de estándares obliga a pensar de nuevo en la idea inicial de normalización de la compresión de datos.

En la actualidad, científicos, tecnólogos, desarrolladores y grandes compañías dedican grandes esfuerzos y recursos al desarrollo de un método universal (Amaru y cols, 2014) (Yaguchi, Kobayashi y Shinohara, 2014). Sin embargo, la solución teórica a esta problemática está todavía muy lejana en el tiempo. En cualquier caso, todos los métodos y técnicas de compresión de datos son clasificados en dos grandes categorías: *lossless* (sin pérdida de datos) y *lossy* (con pérdida voluntaria de datos).

- ***Lossless compression***: significa compresión sin ninguna pérdida de datos o método de recuperación totalmente reversible de los datos. Los datos originales pueden ser reconstruidos a partir del fichero comprimido. *Lossless compression* se utiliza en aquellas aplicaciones en las cuales la pérdida de un solo bit puede ser crucial, ejemplo: base de datos, hojas de cálculo, imágenes médicas, científicas etc.
- ***Lossy compression***: significa compresión con pérdida voluntaria de datos. La reconstrucción de la imagen original contiene pérdida de datos los cuales pueden ser tolerables o no. *Lossy compression* es usado en aplicaciones de imágenes, audio y vídeo en las cuales sensibles cambios de datos no son percibidos debido a las limitaciones del ojo y oído humano. Estas técnicas generalmente se implementan con un hardware especial para cada aplicación. El usuario puede seleccionar el ratio de compresión sacrificando de esta forma la fidelidad a la imagen original.

En la Tabla 2.3 se muestran algunos métodos de compresión de datos correspondientes a las dos familias de compresores. En cada una de ellas se citan unos pocos métodos estándares propuestos por las organizaciones internacionales dedicadas a la normalización de la compresión de datos.

Tabla 2-3. Métodos estándares de compresión de datos

MÉTODO	APLICACIÓN	ESTÁNDAR	OTRAS TÉCNICAS
Compresión con pérdida de datos (*Lossy*)	Imágenes Fijas de Continuos Tonos	JPEG	DPC, TCB, FRACTAL, WAVELET
	Vídeo	MPEG	DIFERENCE CODING, FRACTAL Y WAVELET
	Audio	DPCM	DIFERENTIAL PULSE CODE MODULATION
Compresión sin pérdida de datos (*Lossless*)	Textos	---	Técnicas basadas modelos de sustitución, diccionario, estadísticos, ...
	Imágenes Fijas de 2 niveles de gris	JBIG2	Bitmap, RLE.
	Comunicaciones	CCITT	Técnicas basadas modelos de sustitución, diccionario, estadísticos, ...
	Imágenes Fijas de Continuos Tonos	JPEG-LS	JPEG 2000 fue propuesto como estándar sin éxito

Como se refleja en la tabla anterior, los organismos internacionales han establecido un conjunto de estándares con el fin de normalizar la compresión de datos. En los métodos clasificados como *lossy* están cubiertas todas las aplicaciones con sus respectivos métodos estándares de amplia difusión. En la compresión de datos sin pérdida de información las comunicaciones están ya normalizadas. Sin embargo, en la compresión de imágenes de continuos tonos sin pérdida de información existe un vacío sobre el posible estándar. Todo parece indicar que JPEG-LS (ISO/IEC 14495, 2000) puede ser el método propuesto por estar cerca del estado del arte en la compresión de imágenes de continuos tonos en niveles de grises y color.

Este vacío en la compresión de imágenes de continuos tonos *lossless* contrasta con las necesidades reales actuales. El formato de representación de la información está cambiando en los últimos años. La representación gráfica está sustituyendo a la representación textual y en este cambio la compresión de imágenes juega un factor importante. En imágenes médicas, 3-D, geoespaciales, etc. la pérdida de un solo bit puede ser catastrófico, en este caso la compresión de imágenes sin pérdida de datos

es la clave para garantizar menor consumo de almacenamiento y tiempo de transmisión de datos.

Una vez decidido el carácter *lossy* o *losless* del método de compresión de datos, son también importantes el tiempo de procesado y el coste económico. Cualquier software de compresión puede ser implementado completamente en un hardware específico. Un algoritmo desarrollado en una plataforma hardware o placa no puede convertirse a un algoritmo exclusivamente software. La compresión por medio de hardware tiene un componente especial, precisa una fase previa de diseño y de fabricación que incrementa el tiempo y los costes del producto. Este componente es denominado CODEC (*Coder-Decoder*).

Figura 2.9. Codifcador lossless fabricado por la NASA

A continuación, se apuntan los pros y contras de la utilización la implementación exclusivamente software.

Principales ventajas de la compresión vía software:

- El usuario final no necesita un hardware adicional.
- Disponibilidad en cualquier plataforma PC o MAC etc.
- Adaptable a cualquier innovación tecnológica en ordenadores.
- Cualquier software puede ser implementado en hardware.
- La mayoría de las aplicaciones multimedia emplean técnicas de compresión por software.

Desventajas más significativas:

- Las aplicaciones actuales y futuras realizadas en software pueden no ser soportadas por un hardware anterior
- Los recursos de CPU están dedicados completamente al software de compresión, es decir, no liberan a la CPU de este proceso

El nuevo algoritmo INA, objeto de esta tesis, será implementado por software. De esta forma garantizamos que el método propuesto cumple la propiedad de ser universal y de propósito general.

2.4 Compresión de imágenes digitales sin pérdida de datos

La compresión de imágenes digitales sin pérdida de datos consiste en eliminar o reducir la redundancia existente entre los valores de los píxeles sin pérdida de información. Un método de compresión de imágenes es *lossless* cuando la imagen original puede ser recuperada bit a bit desde los datos comprimidos. Por lo tanto, la compresión de datos *lossless* es un proceso reversible.

El proceso de digitalización de una imagen analógica implica siempre una pérdida de datos al cuantificar una señal analógica o continúa en el tiempo en valores discretos. La compresión de imágenes sin pérdida de datos hace referencia exclusivamente a la compresión de datos aplicada a imágenes digitales.

Una imagen fuente, en términos de almacenamiento, es un fichero o conjunto de valores ordenados lógicamente que configuran una información de entrada. En el ciclo de compresión, los datos de la imagen se cargan en la memoria para suprimir o reducir la redundancia existente en los datos. El conjunto de datos resultante, en términos de almacenamiento, no contiene información. Este conjunto de datos no suministra directamente información legible y el almacenamiento de estos datos genera un archivo de datos comprimido.

El objetivo de la compresión de imágenes digitales sin pérdida de datos consiste en obtener un archivo de datos de menor tamaño que el archivo original y que represente la misma información.

La Fig. 2.10 muestra el proceso típico de compresión-descompresión de una imagen, pero antes de pasar a describirlo en sus diferentes versiones, es necesario establecer cómo se mide la efectividad de cada uno de ellos ¿qué método es mejor que otros? ¿por qué?.

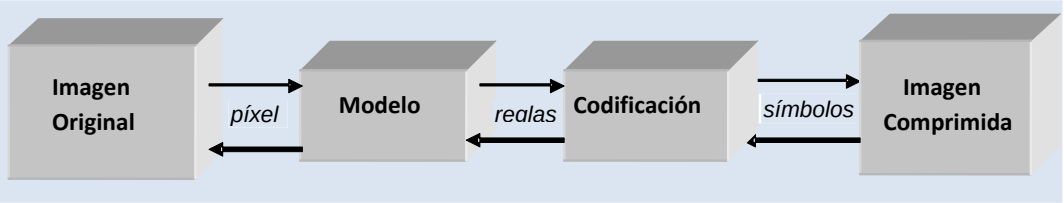


Figura 2.10. Diagrama de bloques simplificado de compresión de imágenes

En la siguiente sección se describen los criterios de medida de la compresión de datos y en concreto el seleccionado en esta tesis.

2.5 Medidas de compresión

La eficacia de un método de compresión de imágenes es determinada por el grado de reducción o de eliminación de la redundancia existente en los píxeles. Una característica común a todas las imágenes reside en los pequeños o inexistentes cambios de valor entre píxeles adyacentes. Estos píxeles están sujetos a la propiedad de correlación que existe entre sus valores y esto significa que contienen información redundante. La tarea de un método de compresión consiste en identificar la correlación y eliminar la mayor cantidad posible de redundancia en la imagen. En la compresión de imágenes sin pérdida existen tres formas de identificar y procesar la redundancia existente en los datos:

- Redundancia espacial: analiza la relación de correlación entre los valores de píxeles vecinos o adyacentes.
- Redundancia espectral: la redundancia es detectada entre los diferentes planos de color en sus bandas espectrales.
- Redundancia temporal: analiza la correlación entre frames adyacentes en secuencias de vídeo.

El propósito de la compresión de imágenes consiste en reducir el número de bits necesarios para representar una imagen fuente mediante la eliminación de la redundancia espacial o espectral. La redundancia temporal es usada únicamente en aplicaciones de vídeo y por tanto no es objeto de estudio de esta investigación.

La Teoría Moderna de la Información (Pierce, 1880) establece las bases de las primeras técnicas de codificación para eliminar la redundancia de los datos en una fuente. La influencia de estas técnicas y su uso se reflejan en los primeros compresores utilizados por ordenador. La importancia de la teoría de la información y la gran contribución a la compresión de datos reside en la propiedad de cuantificar la información de un símbolo o un mensaje, o lo que es lo mismo, cuantificar el número de bits necesarios para codificar un símbolo o un mensaje.

La medida de redundancia de datos en una imagen es determinada por la entropía fuente. La entropía, en términos de compresión de datos, es definida como la cantidad de información contenida en una fuente (S) o como el número mínimo de bits necesarios para codificar un mensaje. Por ejemplo, si la entropía de un mensaje o fuente es de 1024 bits, esto quiere decir que un fichero que comprima dicho mensaje nunca podrá tener un tamaño menor que 1024 bits. Así pues, la entropía indica el límite teórico de la compresión de datos y su unidad de medida es el bit en el caso de la compresión de imágenes.

Cada mensaje está compuesto por símbolos, y cada uno de ellos tiene una probabilidad de aparecer en el mensaje, formando parejas de datos (símbolo, probabilidad del símbolo).

La entropía de una fuente S es definida por Claude E. Shannon (Shannon, 1948) y R.M. Fano (Fanno, 1949) en sus trabajos en el MIT (*Massachusetts Institute of Technology*) como:

$$H(S) = -\sum_{i=1}^{n} P_i \log_2 P_i \qquad (2.1)$$

Donde, Pi: probabilidad de que ocurra el símbolo i en la fuente,

Pi·log2Pi: cantidad de información contenida en un símbolo, o número de bits necesarios para codificar el símbolo, o entropía del símbolo.

Por ejemplo, sea la fuente (S) una imagen de 4x4 píxeles con los valores AVVRVVRRARNRRRRN, en el que cada píxel solo puede tomar uno de cuatro valores: Rojo(R), Verde(V), Azul(A) y Negro(N). En total el mensaje tiene una longitud de 16 y contiene 8 símbolos Rojos, 4 Verdes, 2 Azules y 2 Negros, y por tanto se pueden formar las duplas de símbolo y probabilidad: (R, 1/2), (V, 1/4), (A, 1/8) y (N, 1/8). Así aplicando la expresión (2.1), resulta que la entropía de cada símbolo es H(R)=0,5, H(V)=0,5 y H(A)=H(N)=0,375 y la del mensaje o fuente es H(S)=1,375 bits.

O también, dado un mensaje S de longitud 256 con 256 símbolos distintos apareciendo cada uno de ellos una vez su entropía es 8 bits. Es decir, hacen falta 8 bits como mínimo para codificar ese mensaje: $H(Si) = -1/8 \cdot \log_2(1/8) = 0{,}03125$ bits y H(S) = 8 bits.

A continuación se aporta la el código fuente utilizado para el cálculo de la entropía aplicado a imágenes de 8 bits de profundidad (8 bits por píxel).

```
for(x=0; x<alt*anch; x++)
{
   i= array[x];
   píxel[i]++;
}
for(j=0; j<256; j++)
{
   e_píxel=píxel[x]/(anch*alt)*(log(píxel[x]/(anch*alt))/log(2.0));
   e_imagen= e_imagen + e_píxel;
}
```

Listado 2.1. Programación del cálculo de la entropía

En el anexo A se muestran los resultados de la entropía obtenidos de un conjunto de imágenes estándares ISO que serán utilizadas a lo largo de la tesis.

Un método de compresión de imágenes es más eficaz cuanto más se acerque al valor de la entropía fuente de la imagen. Sin embargo, la entropía no es una unidad de medida determinística del grado de compresión de un método, es decir, no debe de ser usado como un indicador o medida de la eficacia, ya que por ejemplo no existen las fracciones de bit (H(N)=0,375 bits) y por tanto no es una medida útil en la práctica, aunque sí lo es en el plano teórico. Frente a la entropía, un método o

técnica de compresión puede ser evaluado por los resultados obtenidos en diferentes medidas relacionadas con la disminución de las necesidades de almacenamiento, o con los tiempos de ejecución, la complejidad de cálculos, y el consumo de memoria. A continuación se describen algunos índices de compresión para distintos métodos (Nelson, 1995) (Salomon, 2012).

Fente a la anterior aproximación teórica al problema, es pertinente un enfoque más práctico, y así aparecen nuevos criterios e índices de compresión.

Ratio de compresión: es la unidad de medida de la eficacia en términos de almacenamiento de un método de compresión. El ratio de compresión es definido como el cociente entre el tamaño de la imagen fuente entre el tamaño de la imagen comprimida.

$$\text{Ratio Compresión} = \frac{\text{tamaño fichero fuente}}{\text{tamaño fichero original}} \qquad (2.2)$$

Supongamos una imagen de 640*480 píxeles con 8 bits/píxel de profundidad. El tamaño de la imagen original es 307.200 B (300 kB). Si la imagen comprimida ocupa 102.400 bytes (100 Kb), entonces el ratio de compresión resultante obtenido es:

$$\text{Ratio Compresión} = \frac{307.200\ \text{B}}{102.400\ \text{B}} = 3$$

Cuando el ratio de compresión es mayor que uno significa que el fichero original ha sido comprimido. En caso contrario, se obtiene un fichero mayor que el original y es denominado fichero expandido indicando que la compresión ha sido ineficaz.

Existen tres formas **adicionales para representar** el valor del ratio de compresión:

- ***Bit ratio:*** representa el ratio obtenido entre el número de bits de una imagen de entrada y la imagen comprimida. En el ejemplo anterior, el ratio de compresión obtenido es representado por "3:1". Esto es, se precisa un bit en la imagen comprimida para representar 3 bits en la imagen fuente. Esta representación es la más convencional y utilizada para expresar el ratio de compresión de imágenes.
- Ratio expresado en unidades de almacenamiento o ***bits por byte*** (bpb). Esta representación del ratio de compresión se expresa en número de bits por byte o píxel, En el ejemplo propuesto, bpb = 3. Esta representación se utiliza especialmente en compresión de imágenes en términos de almacenamiento y transmisión de datos vía comunicaciones.
- ***Factor de compresión o ratio expresado en valores de porcentaje***: representa la reducción del fichero original en un porcentaje de "X%". Esta forma de representar el ratio corresponde a una representación más popular que científica. Este tipo de representación es usado generalmente

en aplicaciones de archivado. En el ejemplo anterior el factor de compresión ha sido del 70%.

$$\text{Factor Compresión} = \left(1 - \frac{\text{tamaño fichero comprimido}}{\text{tamaño fichero original}}\right) * 100 \qquad (2.3)$$

Para caracterizar los métodos implementados en esta tesis se usará el ratio de compresión.

Ademas de la eficacia del compresor medida con alguno de los índices anteriores, es necesario medir su eficiencia, es decir, cuánto recursos se consumen para conseguir determinado ratio de compresión. Estos recursos pueden ser muy variados y dependen del tipo de aplicación e incluso del usuario, pero hay varios de ellos que son determinantes y aceptados por la comunidad de desarrolladores.

Tiempo de ejecución del ciclo de compresión y descompresión: el tiempo de ejecución de un ciclo de compresión/descompresión afecta a la eficacia del método. Una primera aproximación a este índice es medir el tiempo que tarda un método en comprimir y/o descomprimir una imagen. Una medida práctica del tiempo de ejecución es medir el número total de ciclos necesarios para procesar una imagen, ya sea para comprimirla o descomprimirla, ya que de esta forma se evita el efecto en la medida de la frecuencia del procesador. También se puede usar el índice de ciclos por byte CPB (*Cycles per Byte*), que representa el número de ciclos de CPU que se necesita para comprimir un byte.

Consumo de memoria: Un alto consumo de memoria (en porcentaje del total) puede ralentizar la ejecución de un algoritmo, ya sea de compresión o no. Y esto exige que los algoritmos no consuman memoria de una forma desordenada, que sean óptimos en consumo de memoria.

La restricción anterior ya no es tan exigente porque no solo los algoritmos están en general bien optimizados, sino que además en la actualidad las CPUs y los ordenadores cuentan con una suficiente cantidad de memoria RAM (*Random Access Memory*) para evitar su bloqueo durante la ejecución de un algoritmo de compresión o descompresión.

Además, el exceso de consumo de memoria provoca ralentización en el proceso de compresión, y por tanto y de forma indirecta ese efecto se encuentra incluido en el anterior índice. Es decir, no se considera crítico para el análisis de la eficiencia de los métodos de compresión la medidad de consumo de memoria.

Complejidad: La complejidad de un algoritmo es algo subjetivo y atiende a lo fácil o difícil que es implementarlo en un determinado lenguaje de programación por una determinada persona. Cuanto más complejo sea un método más fácil será que su programación e implementación por parte de un programador no sean eficientes. Por ejemplo, el uso de árboles binarios, de listas entrelazadas, de métodos

adaptativos y/o recursivos, etc. son un reto de programación. Es decir, puede que la complejidad intrínseca de un método sea un freno para su aplicación, más allá de su eficacia.

Este criterio es secundario y en ningún caso incide a priori en la eficiencia de un método de compresión.

Generalmente, los resultados de cada una de las tres medidas de eficacia de compresión descritas anteriormente están relacionados entre sí. Esto es, para mejorar el ratio de compresión es requerido un aumento de complejidad de cálculo y esto supone también un aumento de tiempo de ejecución. En caso contrario, para mejorar la velocidad y tiempo de procesamiento generalmente es necesario sacrificar el ratio de compresión.

La medida de eficiencia de un método de compresión está en función del objetivo de la aplicación del método, es decir, si se pretende que el método incida más en el almacenamiento, en la velocidad o en el cálculo. Un método de compresión puede alcanzar altos ratios de compresión, y sin embargo puede que sus tiempos de cálculo y ejecución no sean operativos en la aplicación que los contiene.

La compresión de imágenes incide más en el almacenamiento y en los tiempos de ejecución que en otros factores, y estos serán los índices utilizados en el análisis de los métodos de compresión actuales y del propuesto en la tesis, INA.

Resumiendo, no existe un método perfecto y por tanto está muy abierto el campo de investigación dando lugar a nuevos métodos que habrá que cuantificar por su eficacia y eficiencia. Además, quizá un método es perfecto para un tipo de imágenes, pero no para otro.

En la tesis se implementarán y cuantificarán varios métodos representativos para comparar sus resultados con el nuevo método propuesto, INA. En todo caso, todos ellos se implementarán en un mismo entorno.

Hardware:

- ✓ Sistema operativo: Microsoft Windows 7 Profesional 64 bits
- ✓ Procesador: Intel Core i5 2,27 GHz
- ✓ Número de núcleos: 4
- ✓ Memoría física instalada (RAM): 4,0 GB
- ✓ Memoria física total: 3,68 GB
- ✓ Memoria física disponible: 2,24 MB
- ✓ % Memoria física en uso:48%
- ✓ Memoria virtual total: 7,35 GB
- ✓ Memoria virtual disponible: 5,37 GB
- ✓ Espacio de archivo de paginación: 3,68 GB

Software:

- ✓ Microsoft Visual Studio 2010
- ✓ LabWindows/CVI 2009 © National Instruments
- ✓ Algoritmo INA

2.6 Métodos de compresión de imágenes *lossless*

Actualmente, un método de compresión de imágenes está constituido por un modelo y un sistema de codificación. El primer proceso consiste en un conjunto de reglas que preparan el camino de la codificación mediante técnicas de asignación de probabilidades, segmentación, interpolación, predicción, transformadas, etc. El sistema de codificación asigna una secuencia de bits a los datos generados por el modelo. En consecuencia, el éxito y la eficiencia del método dependen del modelo. La mayoría de los métodos aportan su propio sistema de codificación. En otros casos utilizan sistemas de codificación tradicionales como: *Run Length Encoding* (*RLE*) (Rubin, 1976) (Hemnath y Prabhu, 2013), Codificador Huffman (Huffman, 1952), Codificador Aritmético (Rissanen, 1976), etc.

En los capítulos sucesivos se describirán y analizarán en detalle todos los métodos de compresión y los conceptos que los describen, pero antes de ello se van a categorizar de forma genérica.

Si no se es un experto en compresión de imágenes, una buena metáfora de partida es pensar en un armario que se debe ordenar en baldas, anotando cada una de ellas. Primero debemos pensar cómo vamos a ordenar el armario: ¿por colores? ¿por estaciones? ¿por tipo de eventos? ¿por tipos de ropa?.... Una vez decidido esto, colocaremos cada prenda en su balda y después pasaremos a nombrar cada balda intentando hacerlo con el menor número de letras o palabras, ¿usamos el mismo número de letras para todas las baldas? ¿menos letras para las baldas más cercanas? ¿menos letras para las baldas más parecidas?... La estrategia o el cómo ordenar el armario es el modelo del método de compresión, mientras que la forma de llamar a cada balda es el sistema de codificación.

Los métodos de compresión según el tipo de modelo pueden ser: fijos o dinámicos (Nelson, 1991).

Métodos basados en modelo fijos: un modelo fijo o estático codifica los datos de entrada mediante un código fijo durante todo el proceso de compresión (Fig. 2.11). Esto significa que el valor de un píxel en la entrada siempre es codificado con un mismo código.

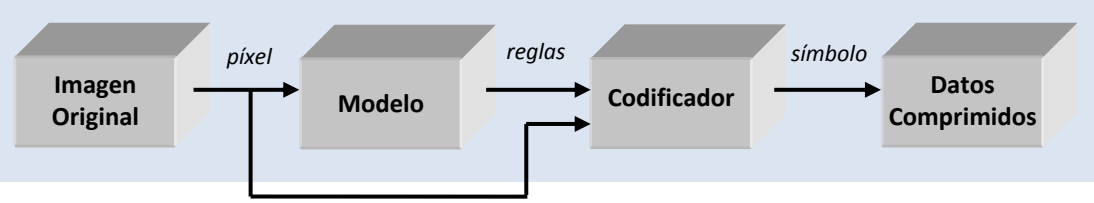


Figura 2.11. Modelo de compresión estático

Métodos basados en modelos dinámicos o adaptativos: un modelo adaptativo es un modelo que genera las probabilidades de los símbolos durante el proceso de compresión en orden a adaptar los cambios del contexto durante el proceso. Inicialmente el proceso comienza con un modelo fijo y durante el proceso el modelo se adapta a los cambios producidos por los símbolos de entrada. Por esta razón, en el ciclo de compresión, el modelo no es transmitido o almacenado junto con los datos comprimidos. En el proceso de descompresión, el decodificador se adapta al modelo de la misma forma que el codificador (Fig. 2.12). Los métodos adaptativos se adaptan mejor a las condiciones locales de cada aplicación que los modelos estáticos.

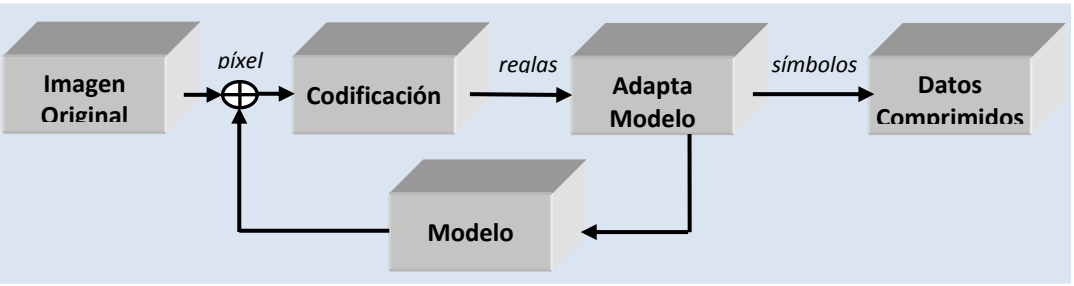


Figura 2.12. Modelo de compresión adaptativo

La diferencia más significativa entre un modelo fijo y un modelo dinámico reside en que los modelos dinámicos solo necesitan procesar los datos una vez, son métodos de un solo paso. Sin embargo, los modelos estáticos son métodos de dos pasos porque requieren procesar los datos dos veces. En un primer paso realizan una lectura previa para construir la tabla de símbolos con sus frecuencias y en un segundo paso realizan una nueva lectura para comprimir los datos. Algunos compresores combinan ambos métodos y son conocidos como compresores híbridos.

Los modelos fijos se emplean normalmente en aplicaciones de almacenamiento mientras que los modelos adaptativos se utilizan en la transmisión de datos. En estas aplicaciones el emisor y receptor utilizan el mismo modelo. En el inicio de la transmisión de datos el receptor no tiene información previa de los datos y esta es la principal razón que hace aconsejable la utilización de modelos adaptativos en las comunicaciones.

2.7 Clasificación de los métodos de compresión *lossless*

En la literatura de compresión de datos se puede encontrar la descripción de numerosos métodos y versiones de optimización. La evaluación de estos métodos es presentada en grandes e interminables listas que contienen ratios y tiempos de compresión en distintos formatos y con diferentes colecciones de imágenes.

La principal aportación en esta sección reside en establecer un nuevo enfoque de evaluación de los métodos de compresión mediante una clasificación de los métodos basándose en el criterio del modelo utilizado. Cualquier método queda automáticamente puede quedar englobado en una o más de las categorías propuestas quedando reflejada la asociación que existe entre los resultados y la categoría a la que pertenece. Una característica reseñable e importante de esta clasificación reside en la evolución de los métodos a través del tiempo y como las tecnologías de su tiempo y desarrollos científicos han marcado sus orígenes.

En la Fig. 2.13 se muestra una nueva clasificación de los métodos basada en el criterio del modelo utilizado. En cada una de las categorías se citan almenos tres métodos representativos.

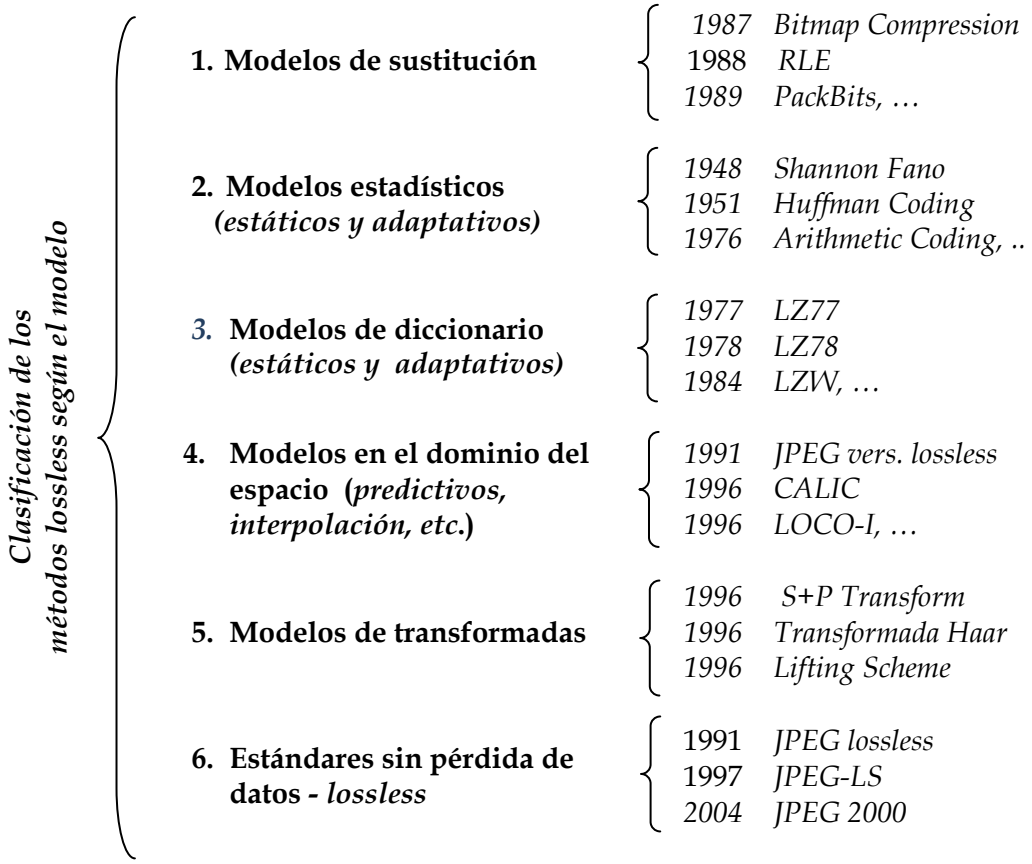


Figura 2.13. Clasificación de los métodos de compresión según el modelo

Cualquier método de compresión de imágenes reversible pertenece a una de estas categorías. Obviamente, existen métodos que combinan varios modelos o se componen de partes de algunos de ellos. Evidentemente, estos métodos pertenecen a la categoría de métodos híbridos (Amaru y cols., 2014).

Atendiendo a la clasificación basada en el modelo se realiza el estudio y la revisión del estado del arte de los métodos más relevantes de la compresión de imágenes sin pérdida de datos (Capítulo 3) relacionada con el algoritmo propuesto (Capítulo 4). En las siguientes sub-secciones se introduce cada categoría de modelo y se determinan los algoritmos más significativos que serán objeto de análisis y revisión en el capítulo que describe el estado del arte de la compresión de imágenes estáticas sin pérdida de datos.

2.7.1 Métodos basados en modelos de codificación

Los primeros métodos de compresión utilizados por computador estaban basados más en las técnicas de codificación que en un modelo de compresión (Nelson, 1991). Estas primeras técnicas utilizaban técnicas de codificación lineal muy simples y de gran rapidez de procesamiento para eliminar o reducir la redundancia en la imagen fuente. Estos métodos se caracterizaban por su sencillez ya que fueron diseñados según las características técnicas y prestaciones de los computadores de su tiempo.

Los modelos basados en técnicas de sustitución explotan la redundancia existente en una imagen, a través de la relación de un píxel con sus adyacentes. Los métodos más significativos que emplean estos modelos son *Bitmap Compression* (Held y Marchall, 1987) y RLE (*Run Length Encoding*) (Rubin, 1976).

El método *Bitmap compression* es considerado el precursor de las primeras técnicas de codificación y su importancia radica en la cantidad de versiones de optimización que surgieron con posterioridad y su influencia en muchos de los posteriores métodos de compresión. Este método surge con la aparición de los primeros ordenadores personales y su principal característica reside en su adaptación a los pocos recursos de CPU necesarios para su ejecución. Obviamente, enmarcándolo en sus inicios, su campo de aplicación corresponde a las imágenes monocromáticas e imágenes gráficas diseñadas por computador las cuales contienen representaciones de superficies y sólidos.

RLE es la técnica de codificación más popular y ampliamente utilizada en compresión de imágenes reversible. La importancia de RLE, como se podrá comprobar a lo largo de la tesis, estriba en que este sistema de codificación es recomendado y usado en posteriores métodos de compresión de imágenes entre los que destacan algunos estándares de compresión con o sin pérdida de datos además de algunos formatos de ficheros gráficos estándares.

2.7.2 Métodos basados en modelos estadísticos

Los primeros métodos de compresión basados en este modelo surgen de la aplicación de los trabajos de Shannon Claude E. (Shannon, 1948), Fanno Robert M. (Fanno, 1949) y del alumno de este último, Huffman David A. (Huffman, 1952) quien resolvío algunos de los errores del algoritmo de Shanon. Estos trabajos supusieron un gran avance en la compresión de datos con respecto a los métodos anteriores. El método de compresión está formado por un modelo y una técnica de codificación. El modelo estadístico puede ser fijo o adaptativo. En ambos casos utilizan reglas específicas para el cálculo de las probabidades de los datos, los cuales son acondicionados antes de ser codificados.

Los modelos estadísticos explotan la redundancia en la información codificando cada píxel según la probabilidad de ocurrencia en la imagen. Esto implica que el compresor debe realizar una lectura previa de todos los píxeles antes de calcular la probabilidad de cada uno de ellos. El resultado de la codificación es almacenado con la tabla de probabilidades. Los métodos más representativos son *Shanon-Fano Coding* (Fanno, 1949), *Huffman Coding* (Huffman, 1952) y *Arithmetic Coding* (Rissanen, 1976).

Los ratios de compresión son considerados como aceptables por la comunidad científica. Sin embargo, la principal aportación de los modelos estadísticos consistió en establecer las bases formales de muchos de los métodos actuales y especialmente su aplicación en el primer estándar JPEG en su versión con perdida de datos así como en otros estándares de compresión, métodos basados en transformadas wavelet, etc.

En el estudio del estado del arte se revisará el Codificador Huffman y el Codificador Aritmético en sus versiones con modelo fijo y adaptativo.

2.7.3 Métodos basados en modelos de diccionario

Los modelos basados en técnicas de diccionario utilizan un código para sustituir una cadena de símbolos o píxeles de entrada. El codificador procesa los píxeles de entrada y recorre el diccionario para encontrar su correspondencia dentro de este. Si la cadena es encontrada, utiliza un puntero al diccionario como código de salida. En caso contrario, la cadena procesada es añadida al diccionario. Los métodos más relevantes son el resultado de los trabajos desarrolados por Lempel Abraham, Ziv Jacob y Welch Terry en 1977, 1978 y1984 respectivamente denominados LZ77 (Ziv y Lempel, 1977), LZ78 (Ziv y Lempel, 1978), y LZW (Welch, 1984).

Los modelos estadísticos y de diccionario en sus versiones de modelo estáticos requieren almacenar la tabla de probabilidades y el diccionario junto con los datos. Esto incrementa el tamaño del fichero comprimido considerablemente. Los modelos

adaptativos evitan el almacenamiento de diccionarios y de tablas de probabilidades. Sin embargo, al comienzo del proceso no consiguen buenos resultados porque carecen de información previa de los datos.

En el estado del arte se analizarán el método LZ77 como método representativo de modelo de diccionario estático y el método LZW basado en un modelo de diccionario adaptativo; y este es sin ninguna duda el método más importante de la familia de modelos de diccionario.

2.7.4 Métodos basados en modelos en el dominio del espacio

Estos métodos combinan modelos en el dominio del espacio para eliminar la redundancia espacial existente en una imagen, de forma local o global. En una segunda etapa aplican modelos basados en técnicas de codificación mencionadas anteriormente. El método más representativo es el estándar JPEG *lossless* (ISO/IEC JTC1/SC29/WG, 1991). El modelo está basado en un algoritmo de predicción de un píxel a partir de los píxeles procesados con anterioridad. El resultado es denominado predicción residual y este valor es codificado mediante uno de los dos métodos recomendados por el comité de estandarización: *Huffman Coding* (Codificador Huffman) o *Arithmetic Coding* (Codificador Aritmético). Existen otros métodos en el dominio del espacio que explotan la redundancia espacial de forma local o global aplicando modelos basados en estrategias de segmentación, interpolación etc.

En el estado del arte se revisará el primer método estándar desarrollado por el comité de estandarización JPEG *lossless* basado en técnicas de predicción. Debido a las limitaciones de los ratios de compresión, este estándar fue reconsiderado y en pocos años fue sustituido por un otro método con un nuevo enfoque y un modelo diferente. JPEG-LS (ISO/IEC 14495, 2000) es el actual estándar recomendado por estar más cerca de las necesidades de la compresión de imágenes reversibles. Este método mantiene un sistema de predicción espacial del píxel basada en el contexto del píxel. La diferencia fundamental reside en que JPEG *lossless* utiliza un modelo fijo de predicción mientras que JPEG-LS utiliza un sistema adaptativo del píxel.

2.7.5 Métodos basados en modelos de transformadas

Estos métodos tienen sus antecedentes en los métodos aplicados a la compresión con pérdida de datos. Aunque actualmente su principal campo de aplicación continua siendo en este campo es reseñable destacar la importancia que estos modelos tienen en técnicas sin pérdida de información. A diferencia de los métodos anteriores que explotan la información de forma espacial, estos nuevos métodos aplican algoritmos en el dominio de transformadas para explotar la información espacial de forma frecuencial. Los métodos basados en la transformada *Wavelet* más

relevantes *S-Transform (*Stockwell y Lowe,1996), Transformada de Haar (Haar, 1910), *Lifting Scheme* (Sweldens, 1996), etc. Otros métodos optimizados como *S+P Transform* han conseguido un espacio relevante en el campo de la compresión de imágenes médicas.

En el estado del arte se revisará en el apartado de modelo estático la Transformada de Haar por su importancia e influencia en otras transformadas y *Lifting Scheme* como la versión adaptativa más relevante en el campo de compresión de imágenes reversible utilizando la versión que incluye conceptos de la transformada de Haar.

2.7.6 JPG-LS propuesto como estándar

JPEG-LS es el nuevo método de compresión *lossless/cerca-lossless* propuesto como estándar para imágenes fijas de continuos cambios de tonos. Este nuevo estándar está basado en el algoritmo LOCO-I *(LOw COmplexity Lossless)*. El término "*cerca-lossless*" es referido a una forma de ejecutar el estándar con una pérdida insignificativa de datos no visible por el ojo humano. Un valor o límite de error es definido con anterioridad a la compresión para determinar el valor máximo de error aceptado sin sacrificar la calidad de la imagen original.

En el estado del arte se revisará en detalle el estándar propuesto por el comité de estandarización y sus resultados experimentales serán analizados y comparados de forma exhaustiva con el método propuesto en este trabajo.

2.8 Conclusiones

La compresión de imágenes es una tarea fundamental en el mundo actual de la informática y las comunicaciones. Los métodos de compresión buscan reducir al máximo el tamaño de los ficheros que almacenan o transmiten las imágenes originales. Las dos características fundamentales de un método de compresión son el ratio de compresión y la velocidad de ejecución o procesado.

En la actualidad no existe un método que tenga el mejor ratio de compresión y sea el más veloz, a esto hay que sumar que el rendimiento de cada método cambia con los distintos tipos de imágenes habituales. Es decir, el diseño de métodos de compresión es un campo muy activo de investigación.

De dicha actividad y de su importancia dan fe los distintos estándares ya desarrollados: JPEG, JBIG, etc.

La motivación de esta tesis surge de la necesidad de mejorar los resultados de los ratios de compresión proporcionados por los estándares de compresión de imágenes de continuos tonos JPEG *lossless* y JPEG-LS. Estos resultados no son muy

aceptables en términos de almacenamiento por la comunidad científica, aunque el comité de estandarización aconseja su despliegue debido a su sencillez, rapidez de ejecución y escasa complejidad.

En el capítulo siguiente se elabora un estado del arte mediante la implementación y análisis de los métodos más representativos. En este estudio se analiza el principio de funcionamiento de cada método, el algoritmo utilizado, un ejemplo ilustrativo y la programación básica del algoritmo. Adicionalmente se aportan los resultados experimentales obtenidos del test de compresión aplicado a un conjunto de imágenes digitales normalizadas por ISO (International Organization for Standardisation) y a otras imágenes significativas en diferentes campos de aplicación. Finalmente, para cada método se aportan las conclusiones basadas en los resultados obtenidos en cada método.

Capítulo

3

Estado-del-arte de la compresión de imágenes digitales sin pérdida de datos

En este capítulo se analizan los métodos que constituyen el estado del arte de la compresión de imágenes estáticas bidimensionales sin pérdida de datos. El análisis y revisión de estos métodos se realiza atendiendo a una clasificación basada en el criterio del modelo de compresión expuesto en el capitulo anterior.

Todos los métodos son revisados exhaustivamente mediante una estructura común que se compone de una descripción detallada del ciclo de compresión y descompresión, incluyendo la descripción formal del algoritmo, la programación del núcleo del algoritmo, el flujograma y un ejemplo práctico ilustrativo del principio de funcionamiento de cada uno de los ciclos. Adicionalmente, y como medida del rendimiento, se presentan los resultados experimentales en términos de ratio de compresión y de consumo de recursos para cada método usando para ello el mismo test de evaluación compuesto por un conjunto de imágenes estándares ISO. Por último se aportan las conclusiones más relevantes a partir de los resultados obtenidos prestando especial énfasis a las ventajas y desvantajas de cada método.

En las secciones 3.1 y 3.2 se revisan los dos métodos más relevantes basados en técnicas de sustitución conocidos como *Bitmap Compression* y RLE respectivamente. En las secciones 3.3 y 3.4 se revisan los métodos más significativos basados en modelos estadísticos en sus versiones estáticas y adaptativas: el

Codificador Huffman y el Codificador Aritmético. En la sección 3.5 se revisa el método basado en un modelo de diccionario estático denominado LZ77 y en la siguiente sección el método de diccionario LZW basado en un modelo adaptativo. En la sección 3.7 se revisan los modelos predictivos en el dominio espacial revisando la versión sin pérdida de datos del primer estándar de compresión de imágenes estáticas denominado JPEG *lossless*. En las secciones 3.8 y 3.9 se revisan los modelos basados en transformadas Wavelet, en la versión estática la Transformada de Haar y *Lifting Scheme* en la versión adaptativa. Finalmente, en la sección 3.10 se revisa el actual estándar de compresión de imágenes sin pérdida de datos: JPEG-LS (ISO/IEC 14495, 2000).

El modelo y los resultados obtenidos en cada método son objeto de estudio y comparación con el algoritmo INA propuesto en esta tesis y que se presenta en el capítulo 4.

3.1 Método de compresión *Bitmap*

Bitmap compresssion (Held y Marchall, 1987) es un método de compresión de imágenes fijas sin pérdida de datos basado en un modelo espacial de sustitución de datos. Este método consiste en segmentar una imagen estática en bloques de píxeles de tamaño fijo. El tamaño del bloque permanece fijo e invariable durante el procesamiento completo de una imagen. El codificador genera un mapa de bits cuyos bits iguales a uno corresponden a la sustitución del píxel más repetido en el bloque y los bits colocados a cero corresponden a la presencia de píxeles con valores diferentes del más repetido.

El nombre del método es por tanto muy significativo y su aplicación está asociada con la aparición de los primeros ordenadores personales. *Bitmap Compression* es un método de compresión de imágenes digitales sencillo y muy simple en términos de consumos de tiempos, recursos de CPU y memoria. La sencillez de implementación del método facilita su aplicación en compresión de imágenes estáticas representadas en color o en escala de grises.

El método *Bitmap* establece las bases formales de los primeros programas de compresión por ordenador. La importancia de este método y principal contribución a la compresión de datos reside en la influencia y repercusión que tuvo sobre los posteriores métodos de la familia de compresión *Bitmap*, entre los cuales destacan los métodos RLE (*Run Length Encoding)* (Rubin, 1976) y *Packbits* (Macintosh, 1988).

3.1.1 Descripción del método

Este método obtiene resultados de compresión aceptables y se considera eficiente cuando el valor de un píxel en un bloque de píxeles aparece con alta probabilidad.

La estructura de un *Bitmap Compression* está formada por tres elementos: valor de sustitución, mapa de bits y campo de almacenamiento con los píxeles diferentes al píxel de sustitución.

- **Píxel de sustitución**: este valor corresponde al valor del píxel más repetido en el bloque. El tamaño de almacenamiento es un byte.
- **Mapa de bits**: consiste en una sucesión de bits del tamaño del bloque procesado. Los bits igual a uno sustituyen al píxel de repetición y los valores igual a cero identifican a los píxeles diferentes al píxel de repetición.
- **Campo de almacenamiento secuencial**: está formado por todos los píxeles diferentes al píxel de repetición y son almacenados de forma consecutiva en orden de aparición en el bloque.

El método utiliza diferentes técnicas para determinar el tamaño de bloque o mapa de bits. El mapa de bits presenta tres posibles alternativas de tamaño dependiendo de las características o naturaleza de la imagen fuente a comprimir.

En imágenes estáticas de continuos cambios de intensidad de color es conveniente utilizar tamaños de bloque pequeños porque los valores de los píxeles a sustituir varían sensiblemente en bloques consecutivos. Esta técnica de compresión suele utilizar bloques de ocho píxeles. Las aplicaciones más convencionales de esta alternativa son las fotográficas o de continuos cambio de intensidad de color.

En imágenes en las cuales los cambios afectan uniformemente a las filas de la imagen es favorable utilizar como bloque el tamaño de la fila. Las típicas aplicaciones que se ajustan a este perfil corresponden a imágenes médicas y a gráficos creados por ordenador, los cuales contienen estructuras de sólidos.

En casos muy particulares y especiales de compresión de gráficos generados por ordenador es conveniente usar como tamaño del bloque el tamaño completo de la imagen.

En la mayoría de las aplicaciones de compresión, la procedencia de la imagen fuente digital es conocida previamente. Por tanto, es posible seleccionar el tamaño del bloque adecuado antes de aplicar un método de compresión. La Fig. 3.1 ilustra la estructura del método mediante un diagrama de bloques.

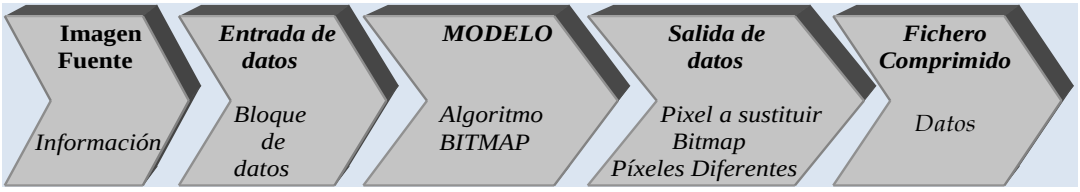


Figura 3.1. Diagrama de bloques del método de compresión Bitmap

Este método realiza una lectura previa de la imagen fuente para almacenar los valores de los píxeles en memoria y a continuación calcula el píxel de mayor probabilidad de aparición o píxel de sustitución. El modelo utiliza el mismo tipo de segmentación de bloque durante el procesamiento completo de la imagen. Esto significa, que no es posible una lectura previa de una fila mediante segmentación en bloques de ocho píxeles y aplicar posteriormente una segmentación lineal de una fila, aunque las condiciones sean favorables para ello.

El procesamiento de los datos se realiza íntegramente en memoria RAM donde almacena los valores de los píxeles leídos en el proceso inicial. La unidad de tratamiento es el valor del píxel más repetido del bloque.

La salida del modelo puede generar compresión o expansión de datos. La compresión se produce cuando el píxel procesado es igual al píxel de sustitución. En este caso, el codificador sustituye el valor del píxel por un bit con valor lógico uno en el mapa de bits. En caso contrario, el codificador coloca un cero en su correspondiente bit del mapa de bits para indicar que no existe compresión. El valor de este píxel es almacenado en el campo de almacenamiento secuencial de los valores diferentes.

El tamaño de la imagen fuente T_o expresado en bytes es calculado aplicando la siguiente expresión:

$$T_0 = \frac{f * c * p}{8} \tag{3.1}$$

Donde, f: representa el número de filas de la imagen,
c: número de columnas y
p: el número de bits utilizado para representar el valor del píxel,

En el caso de una profundidad de ocho bits por píxel, obtendremos:

$$T_0 = \frac{f * c}{8} \tag{3.2}$$

En términos probabilísticos, el tamaño del fichero comprimido en bytes, es calculado mediante la siguiente expresión:

$$Tc = To\,(1- P) + \frac{To}{8} = To\,(1{,}125 - P) \tag{3.3}$$

Donde P: probabilidad del píxel más repetido Vr

Sustituyendo la expresión (3.3) en la ecuación del cálculo del ratio de compresión, obtenemos:

$$Rc = \frac{To}{Tc} = \frac{To}{To(1{,}125 - P)} = \frac{1}{(1{,}125 - P)}$$

En consecuencia, y como el ratio de compresión es igual a uno, no existe compresión, la probabilidad de aparición del píxel de sustitución es calculada por las siguientes expresiónes:

$$1 = \frac{1}{(1{,}125 - P)} \rightarrow 1{,}125 - P = 1 \rightarrow P = 0{,}125$$

Del resultado de la expresión anetrior se deduce que si la probabilidad de aparición de un píxel en un bloque supera el 12,5%, entonces se consigue la compresión de datos. En caso contrario, el resultado implica una expansión de datos que indica que el tamaño de fichero comprimido es mayor que el tamaño de la imagen original. A continuación, se analizan las diferentes alternaticas atendiendo al tamaño del bloque y se aportan nuevas expresiones de cálculo del tamaño del fichero comprimido para cada una de ellas.

3.1.1.1 Bitmap clásico: bloque del tamaño de ocho píxeles

El cálculo del tamaño del fichero comprimido se obtiene aplicando la siguiente expresión:

$$\text{Tamaño Fichero Comprimido} = \sum_{i=1}^{f} \left(10 - N_r(i)\frac{c^*}{8}\right) \tag{3.4}$$

donde, Nr: número total de apariciones del píxel más repetido en el bloque,
c: número de columnas,
f: número total de filas de la imagen,
c: Si la imagen es múltiplo de 8, c*=c , en caso contario c*= c+1.*

El valor decimal diez es el número de bytes de almacenamiento resultante de la suma de un *byte* que representa el valor de sustitución, más un byte correspondiente al tamaño del mapa de bits, y más ocho bytes que ocupa el bloque.

3.1.1.2 Bitmap fila: bloque del tamaño fila

El número de columnas en una imagen computerizada suele ser múltiplo de ocho: 256, 512, 1024, etc. Por tanto, es posible generar mapas de bits completos para cada fila. En caso contrario, los últimos *bits* menos significativos del mapa de bits son recolocados con valor cero para completar el último byte. El tamaño del fichero comprimido aplicando un único valor de repetición correspondiente al píxel de mayor frecuencia en la imagen se obtiene de la expresión (3.5).

$$T_c = \sum_{i=1}^{f} 1 + \frac{c^*}{8} + (c^* - N_r(i)) \tag{3.5}$$

Donde, N_r :número total de apariciones del píxel más repetido en la fila.

3.1.1.3 Bitmap imagen: bloque del tamaño imagen

La técnica de utilizar como bloque la imagen completa es un caso especial de imágenes gráficas generadas por ordenador. El cálculo del tamaño del fichero comprimido se obtiene aplicando la siguiente expresión:

$$T_c = 1 + \frac{f * c}{8} + \sum_{i=1}^{f} (c^* - N_r(i)) \tag{3.6}$$

3.1.2 Proceso de compresión

El ciclo de compresión procesa la imagen fuente aplicando una de las tres técnicas de segmentación descritas en la sección 3.1.1. El procesamiento de los píxeles de la imagen se realiza fila a fila de izquierda a derecha y de arriba hacia abajo hasta completar la imagen. El valor del píxel procesado se almacena en memoria en un *array* unidireccional. El algoritmo de compresión recorre el *array* y obtiene el valor del píxel más repetido en el bloque. Este valor representado por *Vr* es el valor que se procede a sustituir en el bloque.

Para obtener el mapa de bits, el algoritmo recorre nuevamente el *array*. Cada vez que el píxel a sustituir es encontrado, coloca un uno en su posición correspondiente en el mapa de bits *Mb* desapareciendo del bloque procesado. En caso contrario, coloca un cero y registra el valor del píxel diferente *Pd* con un byte de longitud. El proceso se muestra en la Tabla 3.1.

Tabla 3-1. Estructura de almacenamiento de datos en un *Bitmap*

REGISTRO 1			REGISTRO 2				REGISTRO n		
Vr	**Bitmap**	**Pd**	**Vr**	**Bitmap**	**Pd**	**……..**	**Vr**	**Bitmap**	**Pd**
150	11001111	151,152	152	111111110	151	…….	151	111111110	152

Los datos son almacenados en un fichero comprimido mediante una estructura fija de registros formada por los campos de longitud variable *Vr*, *Bitmap* y *Pd.* A continuación, se describe el algoritmo de compresión que genera esta estructura.

3.1.2.1 Algoritmo de compresión

El algoritmo de compresión genera siempre un mapa de bits de ocho bits de longitud permitiendo almacenar su valor en un byte. Esto implica que el tratamiento del mapa de bits se realiza a nivel de byte eliminando cualquier tipo de operación más compleja a nivel de bit. Los pasos del algoritmo de compresión son los siguientes:

Paso 1: Lectura del bloque y almacenamiento de los datos en memoria
Paso 2: Registra el valor del píxel más repetido - Vr

Paso 3: Procesa los píxeles almacenados en memoria uno a uno
Paso 4: ¿Es igual al valor del píxel registrado - Vr?
¿Si?
Coloca un uno en el mapa de bits
El píxel es borrado de la memoria
¿No?
Coloca un cero en el mapa de bits
El valor del píxel es guardado en memoria- Pr
Paso 5: Incrementa en uno el contador del mapa de bits
Paso 6: Repite el ciclo desde el paso tercero hasta la marca final de fichero fuente

El algoritmo de compresión genera por cada ocho píxeles un registro variable de bytes constituido por un byte que contiene el píxel más repetido, un segundo byte que contiene el mapa de bits y a continuación un número variable de bytes que contienen los valores de los píxeles diferentes. En consecuencia, este método no permite recolocar los valores de los píxeles en la imagen original y por tanto debe reservar memoria adicional para los datos comprimidos.

3.1.2.2 Programación del algoritmo

La ejecución de la programación del algoritmo de compresión se realiza íntegramente en memoria *RAM*, procesando los valores, píxel a píxel en cada bloque, hasta finalizar la imagen. Los datos comprimidos se almacenan en un vector unidimensional. La escritura del fichero de salida se realiza volcando el vector completo al fichero de salida.

```
1  Algoritmo Bitmap - bloque de ocho píxeles (8bpp)-
2  // Ciclo de columnas para cada fila
3  // Inicializa ocho bytes
4  for(x=0;x<anchura*anchura;x++) {
5       j=array[x];
6     píxel[j]++;
7     // registra el píxel más repetido, vrep
8       if (píxel[j]>vrep)  {
9            vrep=píxel[j];
10           repetido=j;    }
11      y++;
12      if (y>7) {
13           compri_ocho=0;
14          compri_ocho= 10 - vrep;
15          compri_imag= compri_imag + compri_ocho;
16          y=0;
17       vrep=0;
18           repetido=0;
19           for (z=0;z<256;z++)     {
20                píxel[z]=0; }
21      }
22 }
```

Listado 3.1. Programación simplificada del algoritmo de compresión Bitmap

3.1.2.3 Flujograma del algoritmo de compresión Bitmap

La Fig. 3.2 muestra el flujograma del algoritmo para bloques de ocho píxeles.

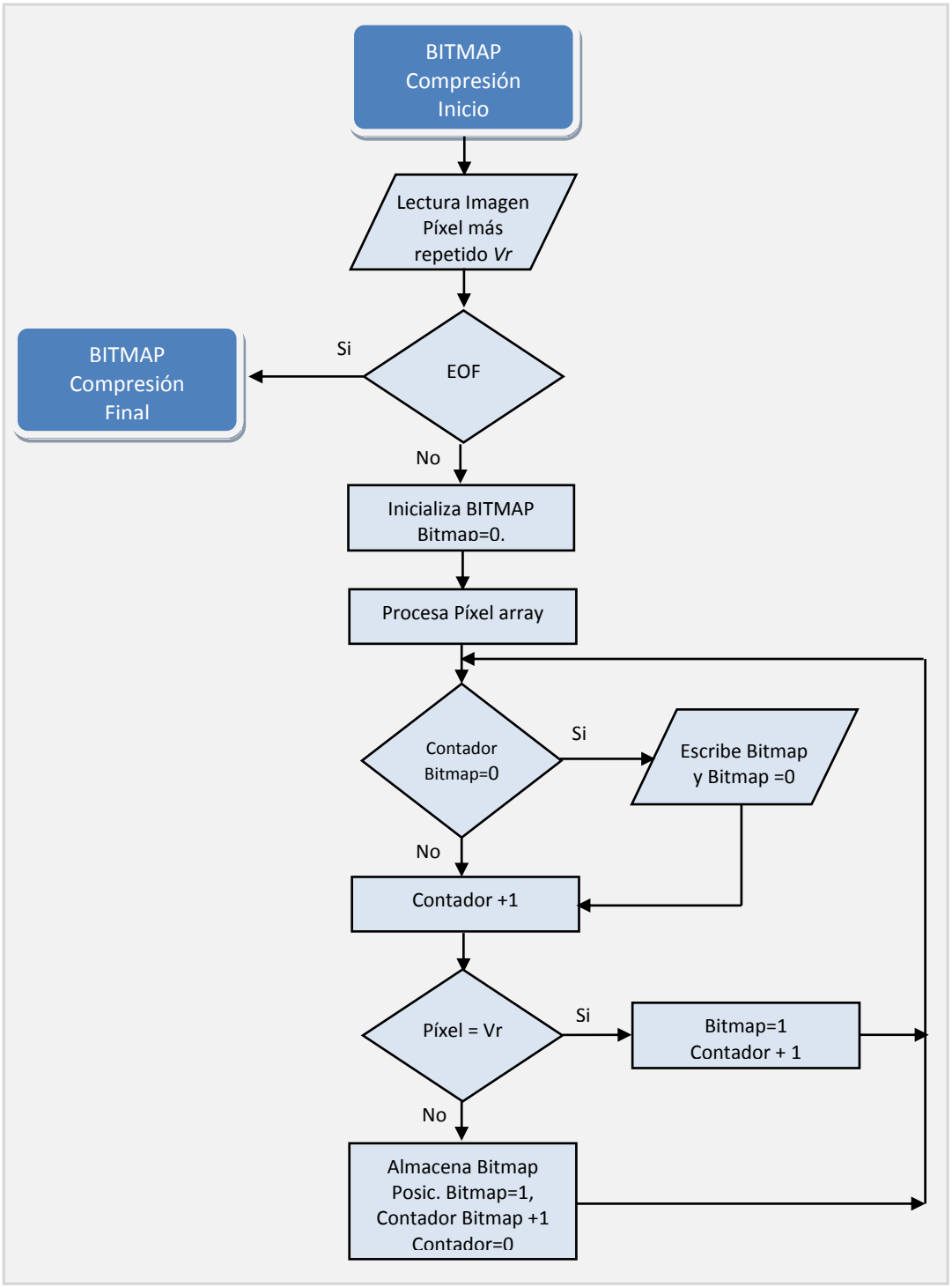


Figura 3.2. Flujograma del algoritmo de compresión Bitmap

3.1.2.4 Ejemplo práctico del proceso de compresión

En este ejemplo se ilustra el principio de funcionamiento del método para un bloque de tamaño de ocho píxeles. Dada una imagen cualquiera que contiene los siguientes ocho primeros píxeles en sus dos primeras, tal y como muestra la Fig. 3.3.

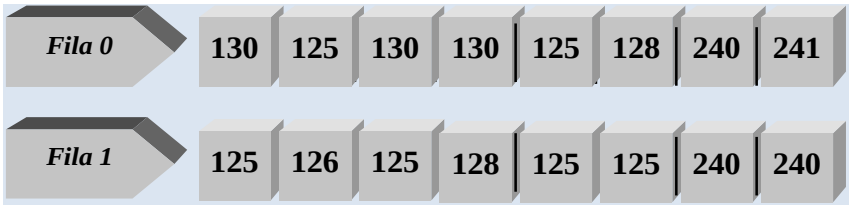


Figura 3.3. Valores decimales de los píxeles de una imagen cualquiera

En un primer paso se realiza una lectura previa de la fila para obtener el valor del píxel más repetido, en el ejemplo propuesto, el valor del píxel de repetición *Vr*=130. A continuación, se procede a construir el mapa de bits y para ello requiere realizar una nueva lectura en memoria de los datos almacenados de la fila comparando cada píxel con el píxel de sustitución. En caso de igualdad, coloca un uno en el mapa de bits en la posición del píxel procesado. En caso contario coloca un cero.

Los resultados se almacenan en memoria para ser volcados al fichero comprimido cuando finalice el ciclo de compresión. Tres tipos de campos son utilizados: un primer campo de 8 bits para registrar el píxel de repetición *Vr*, un segundo campo de un byte para almacenar el mapa de bits *Bitmap* y por último, un tercer campo *Pd* que contendrá tantos bytes como píxeles diferentes sean identificados (Fig. 3.4).

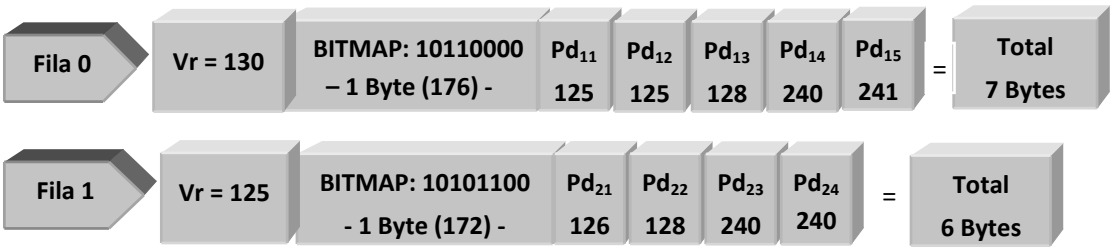


Figura 3.4. Contenido de la estructura del Bitmap

En este ejemplo simplificado, el resultado es una reducción de 3 bytes con respecto al tamaño de los datos originales reduciendo el tamaño de 16 bytes del bloque inicial a 13 bytes en el bloque comprimido. El ratio de compresión es directamente proporcional al porcentaje de aparición del símbolo a sustituir, sin tener en cuenta otros posibles porcentajes altos del resto de los píxeles. El tamaño del fichero comprimido para bloques de ocho píxeles es calculado mediante la expresión (3.4) y el resultado de compresión es mostrado en la Tabla 3.2.

Tabla 3-2. Cálculo del tamaño de los datos comprimidos

FILAS	VALOR DE REPETICIÓN Y NÚMERO DE REPETICIONES	TAMAÑO DATOS ORIGINALES	TAMAÑO DATOS COMPRIMIDOS
Fila 0	V_r =130 N_r=3	8	7
Fila 1	V_r =130 N_r=4	8	6

El resultado de almacenamiento de los datos comprimidos en representación decimal y binaria es mostrado en la Tabla 3.3

Tabla 3-3. Resultado y estructura de los datos comprimidos

FORMATO	V. REP.	BITMAP	PÍXELES DIFERENTES DEL VALOR DE REPETICIÓN				
Decimal	V_r=130	Bitmap=176	$Pd_{1,1}$=125	$Pd_{1,2}$=125	$Pd_{1,3}$=128	$Pd_{1,4}$=240	...
Binario	10000010	10110000	01111101	01111101	10000000	1111 0000	...

Los datos de los bloques de ocho píxeles almacenan consecutivamente el campo del valor de repetición, el campo del mapa de bits y a continuación el campo con los valores diferentes del píxel de repetición.

En la siguiente subsección se describe el ciclo de descompresión para recuperar de forma reversible los datos comprimidos. El proceso no es simétrico porque no tiene que realizar lecturas previas de los datos..

3.1.3 Proceso de descompresión

El proceso de descompresión es más rápido que el proceso de compresión porque no tiene que calcular el valor del píxel con mayor probabilidad, ni tampoco requiere comparaciones de píxeles. La expansión de los datos no precisa de operaciones de direccionamiento con punteros ni de operaciones aritméticas complejas.

Desde el punto de vista de la programación, el contenido del mapa de bits es un conjunto de ocho bits que contiene bits iguales a uno (valor a sustituir por el valor del píxel de repetición) y bits con valor cero (valores de los píxeles diferentes al píxel de sustitución). El procesamiento del mapa de bits implica el uso de operaciones de lógica binaria y de comparaciones a nivel de bit mediante máscaras. Si el valor de la máscara es igual a uno en representación binaria significa que el bit de menos peso es uno y el resto de los bits son cero, "00000001" representado en binario. El incremento del contador genera un desplazamiento de *bits* hacia la izquierda. El bit menos significativo LSB (*Least Significant Bit*) que se incorpora por la derecha es recolocado con valor cero y el bit más significativo MSB (*Most Significant Bit*) desplazado por la izquierda desaparece del byte. Una vez efectuado el desplazamiento se realiza una operación *OR* entre el mapa de bits y la máscara. El resultado de la comparación determinará el valor del bit en el mapa de *bits*.

El algoritmo de descompresión opera en el mapa de bits mediante operaciones binarias expandiendo directamente los datos registro a registro.

3.1.3.1 Algoritmo de descompresión

El algoritmo de descompresión procesa el mapa de bits mediante operacione a nivel de bit. Cada bit del mapa de bits es sustitudo por su correspondiente valor de píxel en la salida. El algoritmo recupera los ocho píxeles y tiene dos alternativas de almacenamiento: aprovechar la misma memoria de la entrada de datos recolocando los resultados o generar una nueva zona de memoria para los píxeles recuperados.

Paso 1: lectura de un registro. Almacena en memoria los campos correspondientes al píxel de sustitución, mapa de bits y píxeles de repetición.
Paso 2: procesa uno a uno los ocho bits del mapa de bits.
Paso 3: ¿el bit actual es igual a uno?
¿Si?
coloca el valor del píxel de repetición en el array unidimensional.
¿No?
recupera de memoria el valor de píxel diferente y lo almacena secuencialmente en el array unidimensional.
Paso 4: el proceso se repite desde el paso 3 hasta finalizar el mapa de bits.
Paso 5: repite el ciclo desde el paso uno hasta detectar final del fichero comprimido.

3.1.3.2 Algoritmo Bitmap-1 de descompresión para bloques de 8 bpp

El algoritmo de descompresión obtiene directamente el píxel de repetición *Vr* en la lectura del fichero comprimido. El resto de operaciones se realizan a nivel de bit y byte de forma similar al proceso de compresión.

```
// Lectura del píxel más repetido- Vr
Vr = fgetc(filecompri);
bitmap=byte;   // convierte caracter del fichero a INT para analizar BITS
aux2=128;
// Recupera  bitmap
for (y=0; y<8; y++){
// Detecta uno o cero con función AND de AUX2=1 -> desplaza de 1 a la izda
   if (bitmap & aux2) {
      array[x]= repetido;
      printf("Entra por igual a 1 - PÍXEL Recuperado= %d\n", array[x]);
      x++; }
   else {
      x++;
      byte= fgetc(filecompri);}
   if (aux2!=7)
      aux2=aux2>>1;
 }
```

Listado 3.2. Programación simplificada del algoritmo de descompresión *Bitmap* (8 bpp)

3.1.3.3 Flujograma del algoritmo de descompresión

En la figura 3.5 se muestra la rutina de descompresión de un bloque de tamaño de ocho bytes.

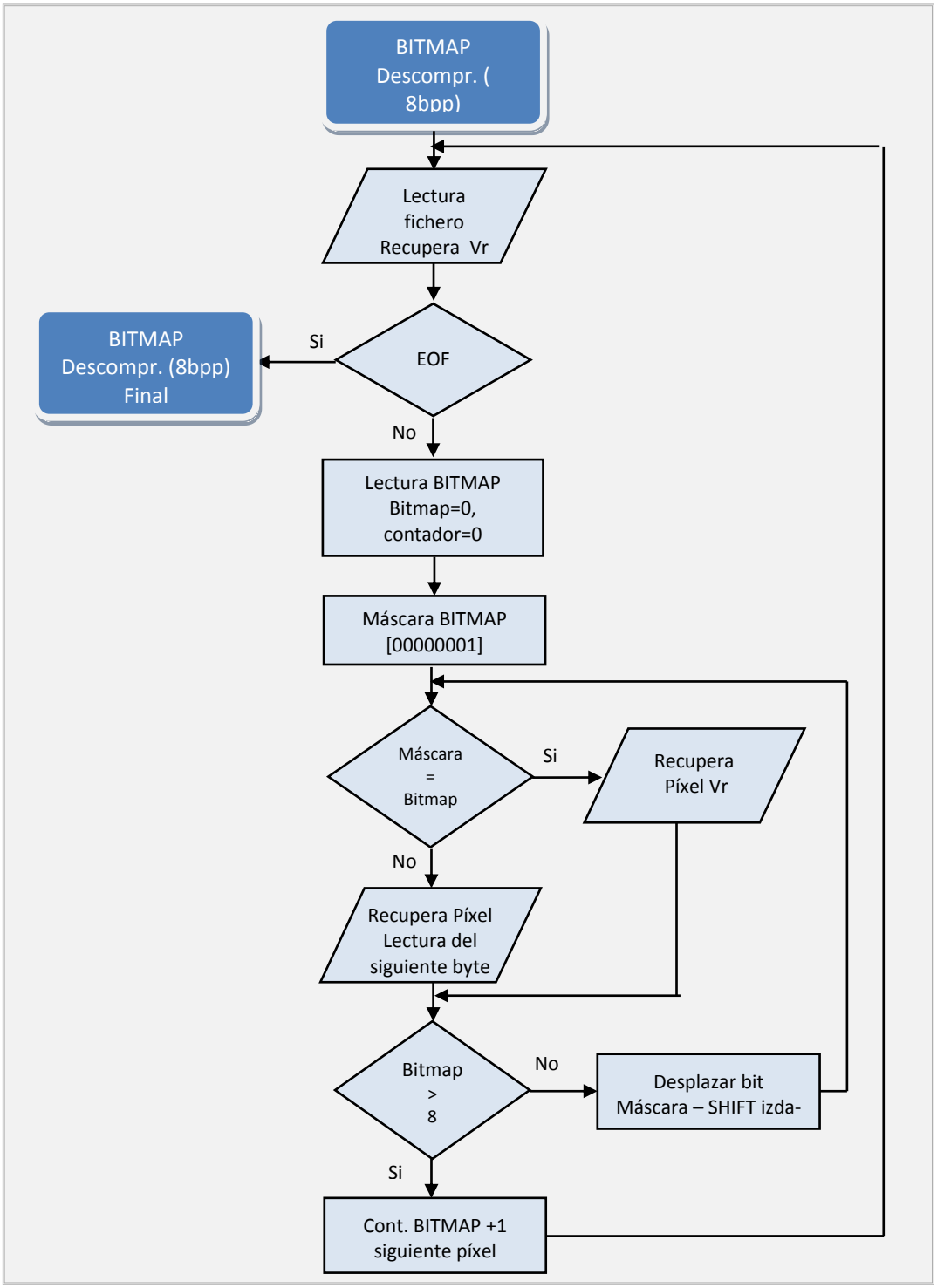


Figura 3.5. Flujograma del algoritmo de descompresión Bitmap (8 bpp)

3.1.3.4 Ejemplo práctico del proceso de descompresión

El ciclo de descompresión realiza la lectura secuencial de los datos del fichero comprimido obtenidos en el proceso de compresión y mostrados en la Tabla 3.3.

FORMATO	V. REP.	BITMAP	PÍXELES DIFERENTES DEL VALOR DE REPETICIÓN				
Decimal	V_r=130	Bitmap=176	$Pd_{1,1}$=125	$Pd_{1,2}$=125	$Pd_{1,3}$=128	$Pd_{1,4}$=240	...
Binario	10000010	10110000	01111101	01111101	10000000	1111 0000	...

La estructura del fichero comprimido corresponde a un formato de registros de longitud variable que se compone de los siguientes campos:

- Campo del valor de repetición *Vr* (1 byte).
- Campo Bitmap (el tamaño es fijo dependiendo de ta técnica de segmentación utilizada).
- Secuencia de píxeles diferentes del valor de repetición *Pd.*

El proceso de decodificación consiste en un bucle de procesamiento de sustitución de los bits del mapa de bits iguales a uno por el valor de repetición y los valores iguales a cero por su correspondiente valor en el campo de píxeles diferentes, tal y como se muestra en la Tabla 3.4.

Tabla 3-4. Recuperación de los datos desde el fichero comprimido

FICHERO COMPRIMIDO	VALOR DECIMAL RECUPERADO	
Píxel Rep - (V_{r1})	130	
BitMap 1	176	
Píxel diferente 1 - ($Pd_{1,1}$)	125	Bitmap comenzando por el bit de mayor nivel
Píxel diferente 2 - ($Pd_{1,2}$)	125	Se almacena a continuación.(Dato5 y Dato7)
Píxel diferente 3 - ($Pd_{1,3}$)	128	
Píxel diferente 4 - ($Pd_{1,4}$)	240	
Píxel diferente 5 - ($Pd_{1,5}$)	241	
Píxel Rep - (V_{r2})	125	
BitMap 2	10101100	4 bytes a remplazar y 4 bytes almacenados
	Dato2	consecutivamente (Dato2-Dato4-Dato6-Dato8)
	Dato4	
	Dato6	
	Dato 8	
Píxel Rep. (Vr3)	...	Hasta finalizar el fichero comprimido

El resultado de la recuperación total de los píxeles originales es el siguiente:

Fila 0 → 130, 125, 130, 130, 125, 128, 240, 241
Fila 1 → 125, 126, 125, 128, 125, 125, 240, 240

3.1.4 Resultados experimentales

La Tabla 3.5 contiene los resultados de los ratios de compresión obtenidos aplicando el método *Bitmap* según el modelo y las técnicas de codificación descritas anteriormente.

Tabla 3-5. Ratios de compresión del método *Bitmap* para imágenes con un byte de profundidad

IMAGEN FUENTE	NOMBRE IMAGEN	BLOQUES DE TAMAÑO: 8 PÍXELES	BLOQUES DE TAMAÑO: FILA	BLOQUES DE TAMAÑO: IMAGEN
		RATIO COMPRESIÓN	RATIO COMPRESIÓN	RATIO COMPRESIÓN
Fotográficaas	Airplane	1.058	0,944	0,913
	Baboon	0,938	0,906	0,897
	Barbara	0,971	0,908	0,896
	Boats	0,982	0,921	0,907
	Cameraman	1,064	0,938	0,910
	Goldhill	0,987	0,917	0,897
	Lena	0,995	0,909	0,897
	Man	1,073	0,956	0,931
	Peppers	0,998	0,911	0,898
	Zelda	0,992	0,909	0,897
	Media	**1,006**	**0,921**	**0,904**
Aéreas y Satélite	Aerial	0,983	0,920	0,905
	Airfield	0,987	0,922	0,914
	Earth	1,032	0,974	0,962
	Meteosat	0,992	0,922	0,911
	Moon	1,241	1,125	1,121
	Moonsurface	1,064	0,922	0,905
	SanDiego	0,984	0,907	0,929
	WashingtonIR	0,932	0,902	0,895
	Media	**1,027**	**0,949**	**0,942**
Creadas por Ordenador	Circles	3,659	1,824	1,494
	Gray	3,849	1,261	0,929
	Slope	1,437	1,211	0,941
	Squares	4,000	2,510	2,265
	Media	**3,236**	**1,701**	**1,407**
Médicas	Elbowx	1,343	1,068	0,985
	Finger	0,915	0,909	0,898
	MRI_Brain1	1,283	1,046	1,032
	MRI_Brain2	1,278	1,153	1,134
	MRI_Brain3	1,001	0,950	0,946
	Shoulder	0,954	1,120	1,120
	Media	**1,129**	**1,041**	**1,019**
Textos y Gráficos escaneados	Mercados1	1,797	1,324	1,119
	Mercados2	1,677	1,718	1.095
	Mercados3	1,625	1,625	1,278
	Mercados4	2,106	2,176	1,519
	Media	**1,801**	**1,711**	**1,253**

En la Fig. 3.6 se muestra la representación gráfica de los de los ratios de compresión expresados en términos de de valores medios para cada clase de imagen fuente y técnica *Bitmap*.

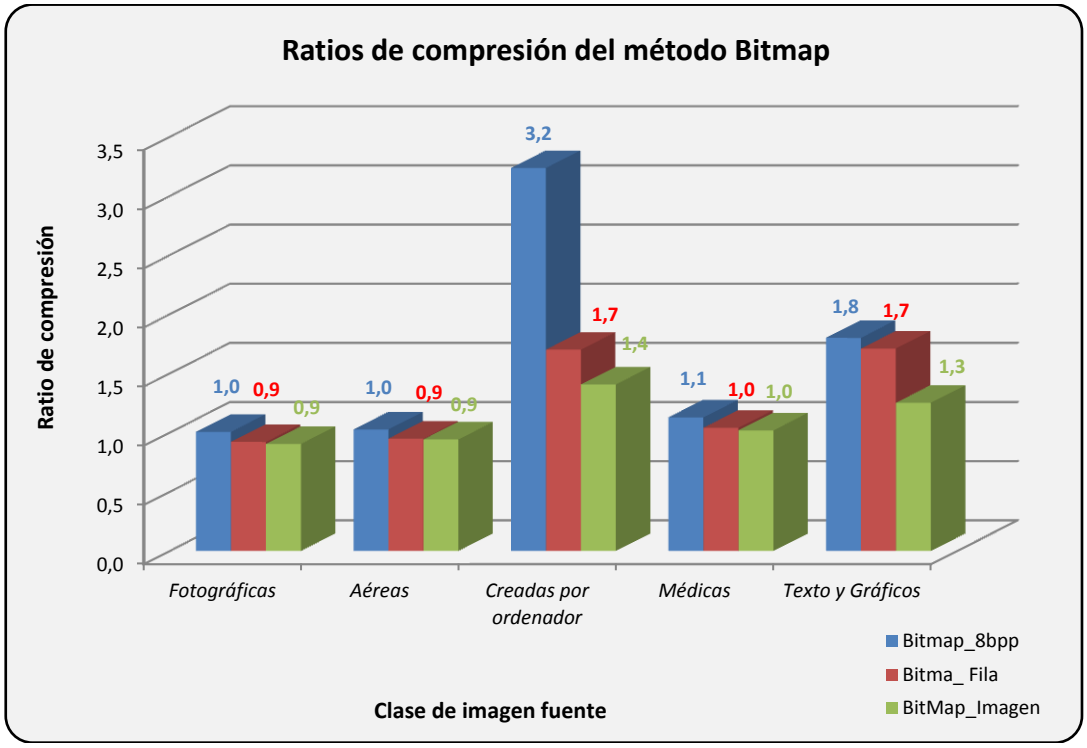


Figura 3.6. Representación gráfica de los ratios de compresión

Los resultados experimentales demuestran que este método consigue sus mejores resultados de compresión en imágenes generadas por ordenador. En el caso particular de imágenes médicas y de imágenes escaneadas con contenidos en forma textual y gráficas, el método garantiza siempre compresión de datos aunque sus resultados están lejos del estado del arte. En la representación gráfica se aprecia que la técnica *Bitmap* de bloques de ocho píxeles obtiene los mejores ratios de compresión para cada clase de imagen fuente, mientras que la técnica de la imagen como bloque alcanza los peores resultados.

En la compresión de imágenes de continuos cambios de tonos y aéreas, los resultados reflejan una expansión de los datos. Esto significa que el fichero comprimido es de mayor tamaño que el fichero original. La justificación de estos resultados debe ser enmarcada teniendo en cuenta tres características que denotan su ámbito de aplicación. Primera, este método surge con la aparición de los primeros ordenadores personales en la década de los 70 y 80. Segunda, la mayoría de las imágenes gráficas visualizadas en estos ordenadores estaban soportadas en imágenes mono-cromáticas, incluso muchas de ellas realizadas mediante caracteres *ASCII*. Inicialmente, las versiones de *CPU* estaban basadas en microprocesadores Intel 8088,

8086 y 80286. En este contexto, con graficos generados por ordenador las técnicas *Bitmap* por fila e imagen obtenían ratios de compresión aceptables.

Por último, los resultados de compresión de la colección de imágenes usada en la evaluación del método están comprendidos en el rango [0,896 - 4,000]. El límite inferior significa expansión de datos mientras que el límite superior es mayor que el ratio de compresión sin pérdida de datos de 2:1, considerado como un valor aceptable para el estado del arte.

La eficiencia de un método de compresión no es únicamente una medida en términos de almacenamiento y unidades de ratio de compresión. La complejidad y los recursos de CPU y memoria son factores importantes para determinar la eficacia del método.

En este contexto, las necesidades de memoria para la aplicación del método son escasas debido a que los ordenadores personales disponían de 256 kB de memoria principal y el programa se ejecutaba íntegramente en memoria.

La versión *Bitmap* clásica reserva un byte para el valor de repetición, un segundo byte para el campo bitmap y como máximo ocho bytes para el bloque. En total, un máximo de 10 bytes para codificar cualquier bloque independientemente del número de repeticiones. Esto significa que el método no requiere memoria adicional externa y que los consumos de memoria se acercan al mínimo del estado del arte. Esta versión *Bitmap* es la que menos consumo de memoria de memoria principal requiere para la ejecución del método.

La versión *Bitmap* fila requiere reservar más espacio de memoria que la versión clásica. Esta versión precisa reservar más espacio en bytes que el numero de columnas de la imagen porque puede gernerar expansión en lugar de compresión. En cualquier caso, los consumos de memoria siguen siendo insignificantes y por tanto la ejecución se realiza íntegramente en memoria principal.

La versión *Bitmap* imagen es la versión que utiliza más recursos de memoria porque requiere una reserva mínima del tamaño de la imagen y al igual que la versión *Bitmap* fila la reserva de espacio de memoria depende del número de repeticiones del píxel. La ejecución de esta versión también se realiza íntegramente en memoria principal y los consumos son más altos que en las dos versiones anteriores.

Para evaluar el método *Bitmap* se han realizado las pruebas experimentales y se han determinado los recursos de CPU basados en el tiempo de compresión y ciclos de CPU de ambos procesos: compresión y descompresión. Los resultados son ofrecidos en las Tabla 3.6, Tabla 3.7 y Tabla 3.8.

Tabla 3-6. *Bitmap* clásico. Tiempos de ejecución del ciclo de compresión y descompresión

IMAGEN FUENTE	NOMBRE IMAGEN	CICLO DE COMPRESIÓN		CICLO DE DESCOMPRESIÓN	
		CICLOS CPU	TIEMPO (ms)	CICLOS CPU	TIEMPO (ms)
Fotográficaas	Airplane	962.540	435.84	372674	168.75
	Baboon	1.122.329	508,20	479984	217,34
	Barbara	1.147.201	519,46	483094	218,75
	Boats	2.538.949	1.149,65	800627	362,53
	Cameraman	706.495	319,90	103696	46,95
	Goldhill	1.099.854	498,02	427834	193,73
	Lena	1.117.263	505,90	434994	196,97
	Man	248.387	112.47	981.41	44.44
	Peppers	1.158.164	524,42	414104	187,51
	Zelda	1.166.597	528,24	411209	186,20
	Media	**1.126.778**	**455,38**	**392822**	**161,00**
Aéreas y Satélite	Aerial	1.112.784	503,87	430555	194,96
	Airfield	1.103.751	499,78	419497	189,95
	Earth	172.004	77,88	73438	33,25
	Meteosat	2.784.175	1.260,69	1092762	494,81
	Moon	960.338	434,85	318898	144,40
	Moonsurface	282.586	127,96	122109	55,29
	SanDiego	4.468.220	2.023,23	1758181	796,11
	WashingtonIR	21.964.733	9.945,74	8404905	3.805,78
	Media	**4.106.074**	**1.859,25**	**1.577.543**	**714,32**
Creadas por Ordenador	Circles	172.459	78,09	91854	41,59
	Gray	685.096	310,21	226473	102,55
	Slope	231.638	104,89	93723	42,44
	Squares	159.824	72.37	94592	42.83
	Media	**312.254**	**123,30**	**126.661**	**46,65**
Médicas	Elbowx	1.052.191	476,44	380596	172,34
	Finger	324.144	146,77	123486	55,92
	MRI_Brain1	4.184.722	1.894,86	107158	48,52
	MRI_Brain2	323.426	146,45	82202	37,22
	MRI_Brain3	526.178	238,26	169327	76,67
	Shoulder	713.077	322,88	243684	110,34
	Media	**1.187.290**	**537,61**	**184.409**	**83,50**
Textos y Gráficos escaneados	Mercados1	3.178.773	1.439,36	1142300	517,24
	Mercados2	3.367.716	1.524,92	1245268	563,86
	Mercados3	2.661.097	1.204,96	981096	444,25
	Mercados4	1.235.976	559,66	466993	211,46
	Media	**1.740.594**	**788,15**	**639.276**	**289,47**

Tabla 3-7. *Bitmap* fila. Tiempos de ejecución del ciclo de compresión y descompresión

IMAGEN FUENTE	NOMBRE IMAGEN	CICLO DE COMPRESIÓN		CICLO DE DESCOMPRESIÓN	
		CICLOS CPU	TIEMPO (ms)	CICLOS CPU	TIEMPO (ms)
Fotográficaas	Airplane	1.055.704	478,03	400.689	181,43
	Baboon	1.396.529	632,36	460.575	208,55
	Barbara	1.400.525	634,16	494.225	223,79
	Boats	1.266.913	573,66	480.545	217,59
	Cameraman	419.425	189,92	112.149	50,78
	Goldhill	1.359.910	615,77	600.583	271,95
	Lena	1.211.367	548,51	454.375	205,74
	Man	310.237	140,48	105.398	47,72
	Peppers	1.715.264	776,68	594.598	269,24
	Zelda	1.308.218	592,37	503.764	228,11
	Media	**1.144.409**	**518,19**	**420.690**	**190,49**
Aéreas y Satélite	Aerial	1.149.824	520,65	434.462	196,73
	Airfield	1.221.737	553,21	506.194	229,21
	Earth	197.003	89,20	123.248	55,81
	Meteosat	2.988.426	1.353,17	1.093.325	495,06
	Moon	1.006.118	455,58	379.489	171,83
	Moonsurface	423.639	191,83	121.588	55,06
	SanDiego	4.696.089	2.126,41	1.716.522	777,25
	WashingtonIR	23.721.740	10.741,32	10.573.874	4.787,90
	Media	**4.425.572**	**2.003,92**	**1.868.588**	**846,11**
Creadas por Ordenador	Circles	302.247	136,86	79.547	36,02
	Gray	1.085.502	491,52	450.698	204,08
	Slope	465.067	210,58	104.414	47,28
	Squares	224.067	101,46	100.707	45,60
	Media	**519.221**	**235,11**	**183.842**	**83,25**
Médicas	Elbowx	1.568.277	710,12	437.071	197,91
	Finger	301.284	136,42	112.441	50,91
	MRI_Brain1	344.639	156,05	106.213	48,09
	MRI_Brain2	388.960	176,12	123.632	55,98
	MRI_Brain3	714.489	323,52	237.884	107,72
	Shoulder	877.255	397,23	284.369	128,76
	Media	**699.151**	**316,58**	**216.935**	**98,23**
Textos y Gráficos escaneados	Mercados1	4.169.951	1.888,17	2.171.652	983,33
	Mercados2	4.239.737	1.919,77	2.089.432	946,10
	Mercados3	2.693.624	1.219,69	1.155.080	523,03
	Mercados4	1.306.177	591,44	725.472	328,50
	Media	**3.102.372**	**1.404,77**	**1.535.409**	**695,24**

Tabla 3-8. *Bitmap* imagen. Tiempos de ejecución del ciclo de compresión y descompresión

IMAGEN FUENTE	NOMBRE IMAGEN	CICLO DE COMPRESIÓN		CICLO DE DESCOMPRESIÓN	
		CICLOS CPU	TIEMPO (ms)	CICLOS CPU	TIEMPO (ms)
Fotográficaas	Airplane	1.022.487	462,99	546.440	247,43
	Baboon	1.310.588	593,44	693.606	314,07
	Barbara	7.696.223	3.484,89	591.232	267,71
	Boats	1.367.366	619,15	719.199	325,66
	Cameraman	385.522	174,57	195.272	88,42
	Goldhill	1.431.528	648,2	753.044	340,98
	Lena	1.327.584	601,14	729.856	330,48
	Man	259.025	117,29	139.170	63,02
	Peppers	1.285.497	582,08	583.580	264,25
	Zelda	1.228.150	556,11	592.748	268,4
	Media	**1.731.397**	**783,986**	**554.415**	**251,042**
Aéreas y Satélite	Aerial	1.394.882	631,61	755.331	342,02
	Airfield	1.283.935	581,37	791.523	358,41
	Earth	256.170	116	125.235	56,71
	Meteosat	2.671.028	1.209,45	178.548	808,47
	Moon	887.482	401,86	458.026	207,4
	Moonsurface	301.690	136,61	150.162	67,99
	SanDiego	4.095.704	1.854,55	2.011.984	911,04
	WashingtonIR	19.730.457	8.934,05	8.254.864	3.737,84
	Media	**3.827.669**	**1.733,19**	**1.590.709**	**811,24**
Creadas por Ordenador	Circles	283.052	128,17	138.682	62,8
	Gray	1.288.116	583,27	715.095	323,8
	Slope	509.522	230,71	143.773	65,1
	Squares	177.269	80,27	113.329	51,32
	Media	**564.490**	**255,605**	**277.720**	**125,755**
Médicas	Elbowx	1.083.657	490,68	535.110	242,3
	Finger	366.479	165,94	197.069	89,23
	MRI_Brain1	305.834	138,48	247.048	111,86
	MRI_Brain2	284.818	128,97	162.953	73,79
	MRI_Brain3	642.382	290,87	321.648	145,64
	Shoulder	652.717	295,55	392.607	177,77
	Media	**555.981**	**251,75**	**309.406**	**140,10**
Textos y Gráficos escaneados	Mercados1	3.554.136	1.609,33	2.372.180	1.256,34
	Mercados2	3.529.696	1.598,26	1.854.394	982,11
	Mercados3	2.509.850	1.136,47	1.201.446	636,3
	Mercados4	1.208.522	547,22	669.328	354,49
	Media	**1.800.367**	**815,21**	**1.016.225**	**538,21**

La evaluación de los tiempos y ciclos de CPU empleados en cada proceso refleja que los tiempos de descompresión siempre son muy inferiores a los tiempos de compresión, especialmente en el método *Bitmap* clásico de 8 píxeles. Por otra parte, en

el proceso de compresión se observa que el método *Bitmap* de 8 píxeles alcanza los menores tiempos de compresión y de descompresión y que el método *Bitmap* imagen es el que peores resultados obtiene en términos de ratios y tiempos de compresión. En la disertación de esta tesis la prioridad está establecida en términos de ratios de compresión. Por tanto, estos resultados son antepuestos a los recursos de CPU utilizados.

3.1.5 Conclusiones del método *Bitmap*

BitMap Compression es un método de compresión de imágenes digitales simple y sencillo de implementar en un ordenador. Este método no requiere punteros ni estructuras complejas de cálculo en los ciclos de compresión y descompresión. La ejecución de este método en un ordenador apenas requiere recursos de CPU y de memoria. Adicionalmente, la sencillez del método facilita su implementación en cualquier plataforma hardware basada en microprocesador. Este método obtenía ratios de compresión aceptables y sobre todo tiempos de ejecución de los ciclos de compresión y descompresión más rapidos que los proporcionados por los métodos alternativos de su epóca.

El método se basa más en la codificación espacial que en el modelo de compresión y es eficiente únicamente cuando un píxel se repite con mucha frecuencia. El método presenta como puntos críticos más desfavorables dos grandes inconvenientes: primero, necesita información previa, esto es, requiere una lectura previa del bloque de segmentación antes de identificar el valor a sustituir, y segundo, no tiene en cuenta la redundancia de los píxeles restantes.

El valor de repetición es único para cada bloque sin tener en cuenta la probabilidad del resto de los valores. En los casos en los cuales las probabilidades de los posibles valores de repetición de un bloque tengan el mismo o similar factor de probabilidad, entonces, el método no tiene en cuenta estos valores significativos. Supongamos un caso especial tengamos dos valores, el primer valor de repetición de un bloque tiene un factor del 55% y otro píxel tenga un 45%. En este caso, el tratamiento de un único valor de repetición provoca despreciar el porcentaje del resto de los píxeles.

Esta problemática es resuelta por otros métodos pertenecientes a la misma familia de compresión y especialmente por el método *Run Length Encoding*. Las variantes más conocidas de RLE son *Windows Bitmap Compression (Microsoft)* y *PackBits (Apple Computer Inc.)*. La primera opción está soportada por los formatos de fichero *BitMap* de *Microsoft* (Microsoft, 1995) (Charlap, 1995) y la segunda por *Apple Macintosh* en los formatos PICT (*Macintosh Picture*) y TIFF (*Tagged Image File Format*). El método RLE es el más popular y significativo de esta familia de compresores y el

método más utilizado en estándares de compresión. Este método es descrito en la siguiente sección.

3.2 Run Length Encoding (RLE)

Run Length Encoding es uno de los métodos de compresión sin pérdida de datos más eficiente, simple y popular de la familia de compresores de sustitución. Este método, al igual que *Bitmap Compression*, aplica un sistema de codificación espacial para eliminar la redundancia de los píxeles. La diferencia entre ambos métodos reside en que *Bitmap* codifica únicamente el píxel de mayor probabilidad mientras que RLE elimina la redundancia de píxeles consecutivos.

El principio en en cual se basa este método consiste en la sustitución de una secuencia consecutiva de píxeles repetidos por una secuencia más corta formada por el valor del píxel y un contador de repeticiones (Rubin, 1976). La secuencia formada por el número de repeticiones consecutivas es reemplazada por el valor del píxel más el número de veces que se repite en la secuencia.

Los ratios de compresión alcanzados por este método dependen del tipo de imagen fuente. Los mayores ratios de compresión son conseguidos en imágenes gráficas por computador, especialmente en imágenes que contienen superficies de sólidos. Por el contrario, este método como los métodos basados en técnicas de sustitución no es operativo cuando la compresión se aplica a imágenes de continuos cambios de intensidad de color. Incluso, puede suceder que el tamaño de la imagen comprimida resultante pueda ser mayor que el tamaño original.

RLE es un método de compresión soportado por formatos de ficheros *Bitmap*, como son BMP (Marv, 1994), TIFF (*Tagged Image File Format*), PNG (*Portable Network Graphics*), TGA (*Text Graphic Adapter*) y PCX (*Zsoft Paintbrush image format*) y por estándares de compresión como JPEG, JBIG, etc. Tsukiyama tiene dos patentes de *Run Length Encoding* (Tsukiyama, 1986 y 1989). La primera cubre la versión inicial, un byte de longitud por píxel repetido. La segunda patente cubre la invención de limitar el contador a un máximo de 16 repeticiones y por tanto la codificación de la longitud se realiza con 4 bits.

3.2.1 Descripción del método

La idea básica de los métodos de compresión de imágenes digitales basados en técnicas de sustitución consiste en tomar ventaja de la propiedad de vecindad entre píxeles consecutivos. Esta propiedad reside en la característica que presentan los píxeles adyacentes de tener el mismo valor de intensidad de color o variaciones mínimas entre píxeles vecinos. Los métodos de compresión basados en técnicas de

sustitución y especialmente los métodos predictivos explotan esta propiedad para realizar la compresión de forma más eficiente.

El principio de funcionamiento consiste en representar una secuencia de valores de píxeles en forma pares ordenados. Cada par ordenado contiene un primer elemento que representa el número de veces que se repite el valor del píxel o *run length* y un segundo elemento que representa el valor del píxel. El resultado del procesamiento de imagen fuente es una sucesión de pares ordenados:

$$(r_1, v_1)(r_2, v_2)(r_3, v_3), \dots, (r_n, v_n) \tag{3.7}$$

donde r_i representa el número de veces que se repite el valor del píxel y v_i representa el valor del píxel.

En la Fig. 3.7 se representa el diagrama de bloques del proceso de compresión del método RLE.

Figura 3.7. Proceso de compresión del método RLE

La codificación forma pares de elementos fijos limitados a valores de repetición comprendidos en un rango [0-255] atendiendo a la máxima representación de un byte. Esta limitación implica que si una fila contiene más de 256 valores de repetición consecutivos es necesario crear un nuevo par con el mismo valor del píxel y su nuevo valor de repetición. Las versiones posteriores solucionaron esta desventaja utilizando un valor de escape cada vez que cambia el valor del píxel.

La unidad de tratamiento en la entrada de datos es el píxel. El método procesa secuencialmente todos los píxeles y por cada cambio de valor registra dos bytes. El primer byte contiene el número de repeticiones consecutivas el segundo el valor del píxel. En el caso desfavorable de no existir repetición, el primer byte contendrá un cero. La salida del modelo genera bloques de **longitud fija** de dos bytes de tamaño por cada píxel diferente que encuentra en la entrada.

La salida de datos puede generar compresión o expansión de los datos. En el caso más favorable, cuando todos los píxeles de la imagen son iguales y el número de filas "*f*" es múltiplo de 256, el tamaño del fichero comprimido en bytes, con una representación de ocho bits/píxel, es calculado mediante la siguiente expresión:

$$T_c = 2 * \left(\frac{f}{256}\right) * c \tag{3.8}$$

Cuando el número de filas no sea múltiplo de 256, el tamaño del fichero comprimido en bytes es calculado mediante la siguiente expresión:

$$T_c = 2 * \left(\frac{f}{256} + 1\right) * c \tag{3.9}$$

En el caso más desfavorable, cuando todos los píxeles consecutivos son diferentes, el tamaño de fichero comprimido ocupará el doble de almacenamiento.

$$T_c = 2 * T_o \tag{3.10}$$

En el caso de imágenes de continuos cambios de tonos es recomendable añadir un mapa de bits de un byte de longitud para indicar las posiciones de repetición de los píxeles. Los bits iguales a uno indican que el píxel es repetitivo mientras que los bits a cero indican que no existe repetición del píxel y por tanto no debe utilizarse un byte de repetición. Esta ídea se basa en la utilización del campo de bits del método *Bitmap* revisado anteriormente. La salida del codificador sigue manteniendo la longitud fija en la salida del codificador.

3.2.2 Proceso de compresión

La imagen es analizada píxel a píxel mediante técnicas *scan line*. En el caso de imágenes en color, cada plano de la imagen R-G-B (*Red-Green-Blue*) es separado previamente y codificado independientemente. Existen diferentes técnicas de rastreo de los píxeles de una imagen: filas, columnas o zigzag (Fig. 3.8).

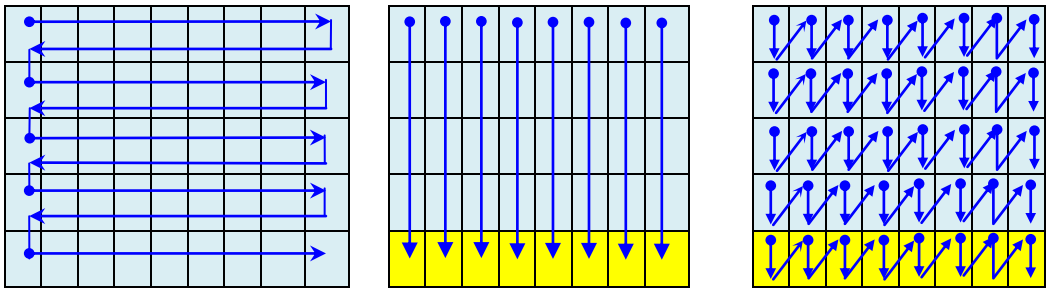


Figura 3.8. Técnicas de rastreo de imágenes en RLE

Todos los píxeles de la fila son tratados de forma secuencial comparando cada píxel con su adyacente. En caso de igualdad, el contador se incrementa en uno, en caso contrario, se escribe en el fichero comprimido el número de veces que el valor del píxel anterior ha ocurrido y su valor. El contador es reiniciado con valor uno y el ciclo es repetido hasta completar la fila. Este proceso continua hasta que imagen es procesada completamente y convertida a códigos en formato de pares ordenados en el fichero de compresión.

3.2.2.1 Algoritmo de compresión

Cada secuencia de píxeles iguales y consecutivos genera un par ordenado (r_i, v_i), donde r_i es el número de repeticiones y v_i es el valor del píxel. El algoritmo procesa la imagen píxel a píxel y almacena en un array los pares ordenados.

Paso 1. Inicializa valor del píxel a cero y valor de repetición a uno
Paso 2. Lectura del primer píxel
Paso 3. Comprueba si el siguiente valor es igual al anterior
¿Si?
Paso 4. Incrementa en uno el valor de repetición
¿No?
Paso 5. Escribe valor de repetición en la salida
Paso 6. Escribe valor de píxel en la salida
Paso 7. Coloca el valor de repetición a uno y registra el valor del píxel
Paso 8. Repite el ciclo desde el Paso 2 hasta marca final de fichero

3.2.2.2 Programación del algoritmo

Este algoritmo utiliza operaciones arítmeticas básicas, un campo contador de repeticiones y una zona de memoria adicional para almacenar las duplas de salida. La complejidad del método es inexistente y los recursos de memoria son mínimos. Además, la velocidad de procesamiento es elevada debido al tipo de operaciones de cálculo realizadas.

```
1  // Algoritmo RLE
2  // Bucle de procesamiento de la imagen original
3  for(x=0;x<anchura*anchura;x++)
4  {     // Inico - guarda el valor del primer píxel
5       if (x==0) {
6         v_ píxel=array[x];
7         runl=0; }
8       else
9         {
10             if (array[x]==v_píxel) {
11                 runl++; }
12            else
13            {
14                 y++;
15                 salida[y]=runl
16                 y++;
17                 salida[y]= v_píxel;
18                 v_píxel=array[x];
19                runl=1;
20             }
21       }
22 }
```

Listado 3.3. Programación del algoritmo de compresión RLE

3.2.2.3 Flujograma del algoritmo de compresión RLE

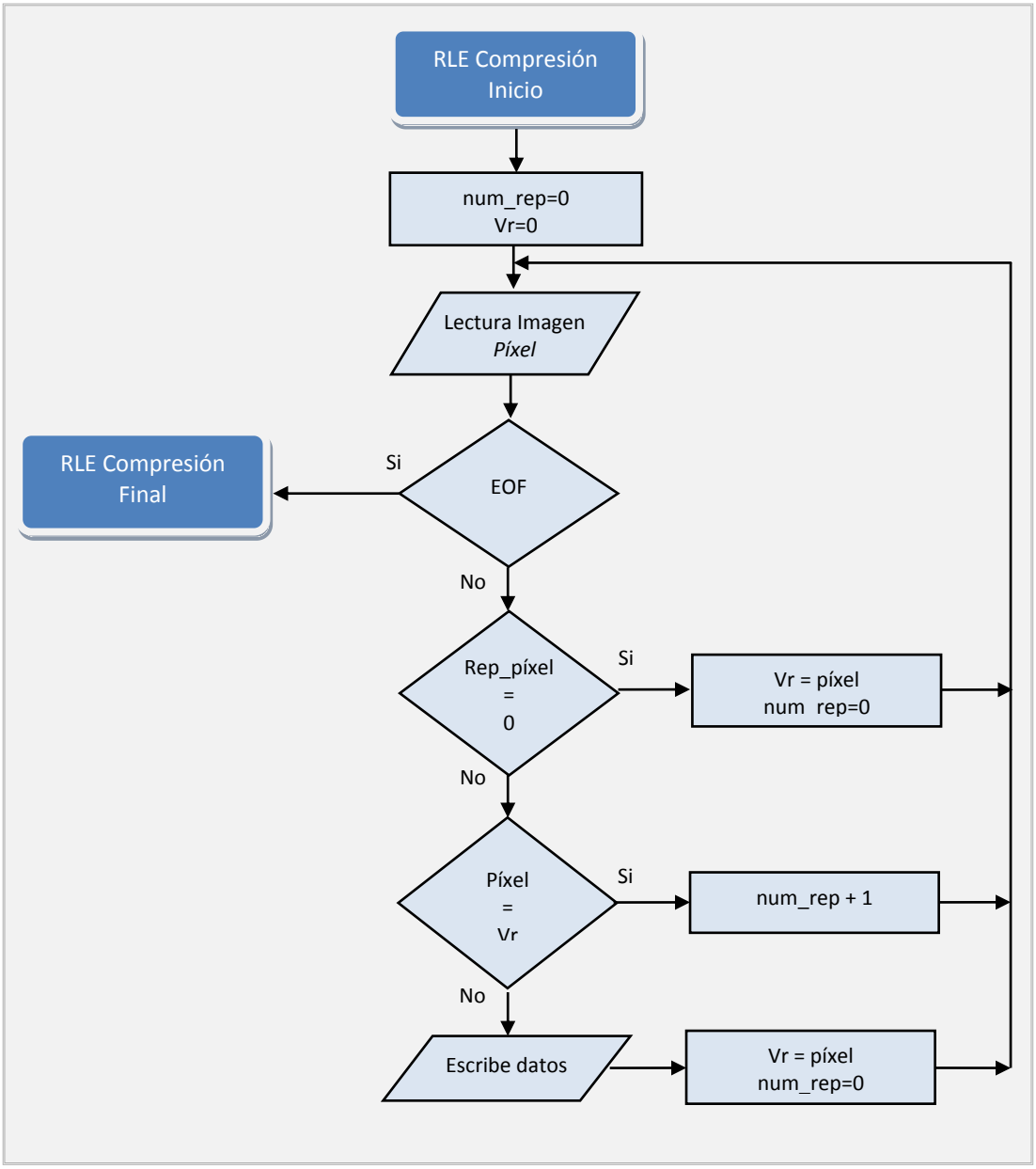


Figura 3.9. Flujograma del algoritmo de compresión RLE

3.2.2.4 Ejemplo práctico del proceso de compresión

Dada la siguiente secuencia de valores de píxeles en formato decimal de una imagen cualquiera se pretende aplicar el método de compresión RLE para reducir el espacio de almacenamiento.

Ejecutando la rutina del algoritmo de codificación se obtiene una secuencia reducida formada por once pares ordenados con sus correspondientes valores del píxel y longitud de la carrera, tal y como se muestra en la Tabla 3-9.

Tabla 3-9. Resultado de la compresión del ejemplo práctico del método RLE

DATOS COMPRIMIDOS																				
r_1	c_1	r_2	c_2	r_3	c_3	r_4	c_4	r_5	c_5	r_6	c_6	r_7	c_7	r_8	c_8	r_9	c_9	r_{10}	c_{10}	r_{11}	c_{11}
0	1	0	11	1	1	0	4	6	6	0	7	0	8	0	10	4	12	2	11	0	10

La salida del codificador genera datos de longitud fija formados por una sucesión de pares ordenados. En los casos de píxeles sin repeticiones se consume un byte innecesario con valor cero. En el caso especial de usar un código de escape el tamaño de la salida se reduciría en siete bytes.

3.2.3 Proceso de descompresión

El proceso de descompresión es el más simple y más rápido de todos los métodos universales y por supuesto más rápido que el proceso de compresión. Los pares ordenados de dos bytes de longitud en el archivo comprimido son procesados secuencialmente recuperando el valor del píxel y expandiéndolo con la longitud de la carrera de repetición definida en el primer byte.

3.2.3.1 Algoritmo de descompresión

Paso 1 Inicializa contador de repetición a cero.
Paso 2. Lee dos bytes del fichero comprimido
Paso 3. Escribe el píxel de salida
Paso 4. Incrementa en 1 el contador
Paso 5 Si el contador es menor que el número de repeticiones, escribe píxel de salida e incrementa en uno el contador
Paso 6. Si contador es mayor que el número de repeticiones, repite desde el paso 2 hasta final de fichero
Paso 7. Repite desde el paso 5.

3.2.3.2 Programación del algoritmo

```
1  // Algoritmo RLE descompresión
2  // Bucle de procesamiento de la imagen original
3  z=0;
4  for(x=0;x<anchura*anchura;x+2) {
5      // Inicio –lee dos bytes: píxel y valor de repetición
6      vpíxel=array[x+1];
7      for (contador=0; contador<array[x]; x++) {
8          z++;
9          entrada[z]=vpíxel; }
10 }
```

Listado 3.4. Programación del algoritmo de descompresión RLE

3.2.3.3 Flujograma del algoritmo de descompresión

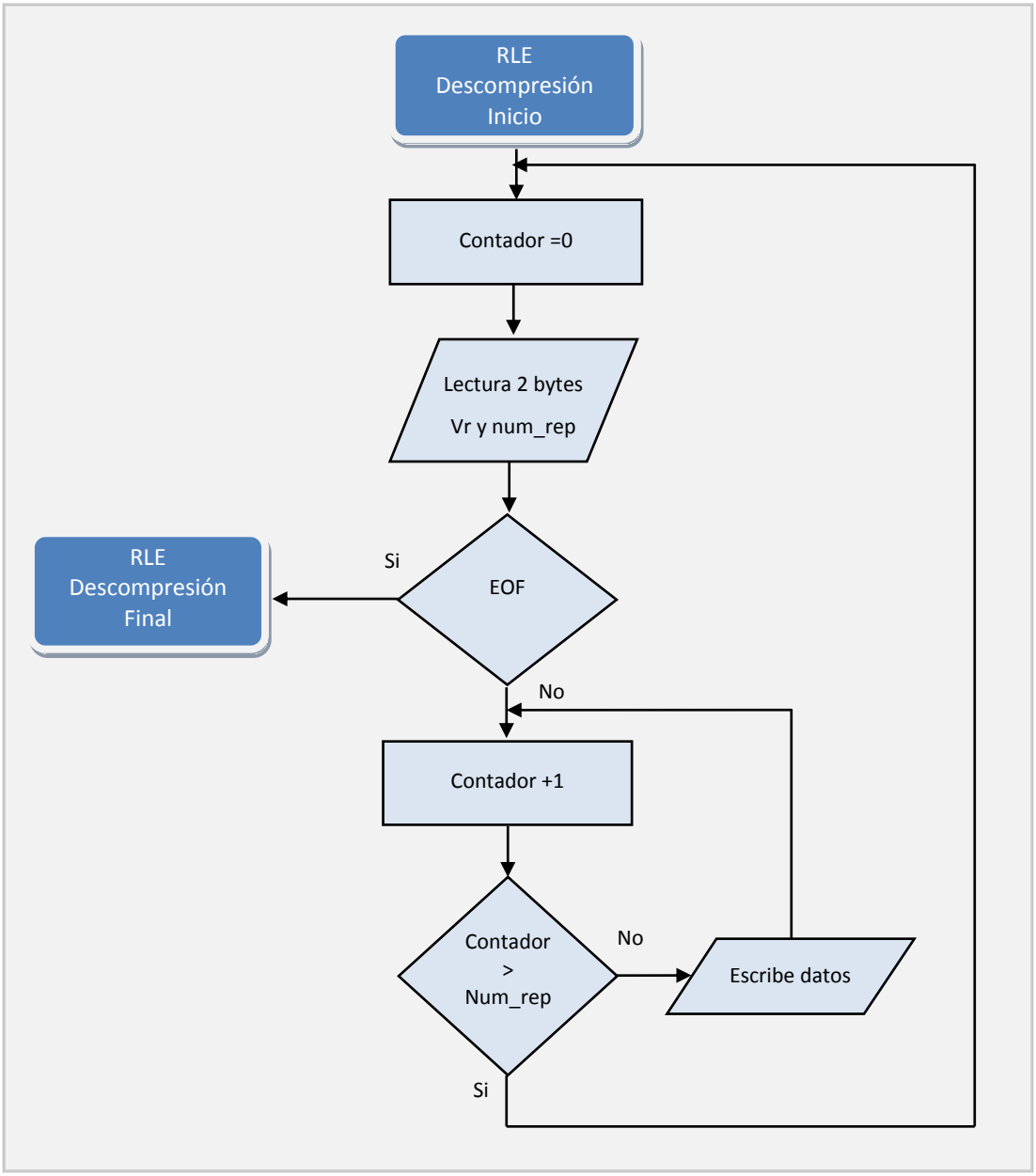


Figura 3.10. Flujograma del algoritmo de descompresión RLE

3.2.3.4 Ejemplo práctico del proceso de descompresión

Considerando los resultados obtenidos en el ejemplo del proceso de compresión (Tabla 3.10), el algoritmo de descompresión procede a expandir los datos para la recuperación reversible de estos. La lectura del fichero comprimido es de dos en 2 bytes y la expansión de datos es directa.

Tabla 3-10. Muestra de los once primeros datos del fichero comprimido del método RLE

DATOS COMPRIMIDOS																					
r_1	c_1	r_2	c_2	r_3	c_3	r_4	c_4	r_5	c_5	r_6	c_6	r_7	c_7	r_8	c_8	r_9	c_9	r_{10}	c_{10}	r_{11}	c_{11}
0	1	0	11	1	1	0	4	6	6	0	7	0	8	0	10	4	12	2	11	0	10

El proceso de descompresión realiza una lectura secuencial de los datos del fichero comprimido. Los pares ordenados (0,1); (0,11); (1,1); (0,4); (6,6); (0,7); (0,8); (0,10); (4,12); (2,11); (0,10) se expanden generando los valores iniciales según se muestra en la Tabla 3.11.

Tabla 3-11. Resultado del ciclo de descompresión del ejemplo RLE

EXPANSIÓN DE LOS DATOS COMPRIMIDOS																							
p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}	p_{11}	p_{12}	p_{13}	p_{14}	p_{15}	p_{15}	p_{17}	p_{18}	p_{19}	p_{20}	p_{21}	p_{22}	p_{23}	p_{24}
1	11	1	1	4	6	6	6	6	6	6	6	7	8	10	12	12	12	12	12	11	11	11	10

En el caso especial de secuencia de escape es necesario utilizar operaciones a nivel de bit para analizar mapa de bits y en función del valor de cada bit proceder a expandir o sustituir el valor del píxel.

Para evaluar los resultados del método se ha aplicado el métodos RLE clásico y una versión optimizada que utiliza un campo bitmap para cada bloque de ocho píxeles diferentes con una longitud de cuatro bits para representar la carrera de avance. En la siguiente sección se exponen los resultados y la evaluación del método para ambas versiones.

3.2.4 Resultados experimentales

En la Tabla 3.12 se reflejan los resultados de la evaluación del método RLE en su versión clásica y con símbolo de escape.

Los resultados de la Tabla 3.12 muestran que el ámbito de aplicación óptimo de este método es la compresión de imágenes generadas por ordenador y la compresión de imágenes que incluyen textos y gráficos. La tabla de resultados refleja que el método de compresión con secuencias de escape y cuatro bits de longitud de carrera de avance obtiene mejores resultados en todas las imágenes evaluadas que el la versión clásica y optimizada. Sin embargo, el método RLE produce expansión en lugar de compresión en nueve de las diez imágenes de continuos cambios de intensidad y en seis de las ocho imágenes aéreas.

Tabla 3-12. Resultados experimentales del método RLE aplicado a imágenes con un byte de profundidad

IMAGEN FUENTE	NOMBRE IMAGEN	RLE CLÁSICO OPTIMIZADO	RLE 4BITS
		RATIO COMPRESIÓN	RATIO COMPRESIÓN
Fotográficaas	Airplane	0,937	0,990
	Baboon	0,894	0,906
	Barbara	0,901	0,924
	Boats	0,899	0,920
	Cameraman	0,944	0,995
	Goldhill	0,902	0,928
	Lena	0,906	0,938
	Man	0,970	1,037
	Peppers	0,904	0,932
	Zelda	0,902	0,932
	Media	**0,916**	**0,950**
Aéreas y Satélite	Aerial	0,913	1,057
	Airfield	0,913	0,999
	Earth	0,934	0,972
	Meteosat	0,914	0,947
	Moon	1,347	1,429
	Moonsurface	0,988	1,293
	SanDiego	0,909	0,933
	WashingtonIR	1,003	1,015
	Media	**0,990**	**1,081**
Creadas por Ordenador	Circles	38,102	52,241
	Gray	80,388	(*) 146,408
	Slope	1,597	1,874
	Squares	78,769	105,026
	Media	**49,714**	**76,387**
Médicas	Elbowx	1,172	1,460
	Finger	0,890	0,896
	MRI_Brain1	1,216	1,418
	MRI_Brain2	1,136	1,381
	MRI_Brain3	0,923	0,959
	Shoulder	1,158	1,353
	Media	**1,083**	**1,245**
Textos y Gráficos escaneados	Mercados1	1,783	2,099
	Mercados2	1,546	1,821
	Mercados3	1,706	1,937
	Mercados4	2,268	2,711
	Media	**1,826**	**2,142**

La Fig. 3.11 muestra gráficamente los resultados de la Tabla 3.12 en una escala logarítmica para obtener una representación global de los datos. En esta gráfica se aprecia la eficiencia del método RLE en las imágenes creadas por ordenador, en las cuales alcanza ratios de media de 76,3. Este valor confirma que este método sigue siendo válido para la compresión de esta clase de imágenes.

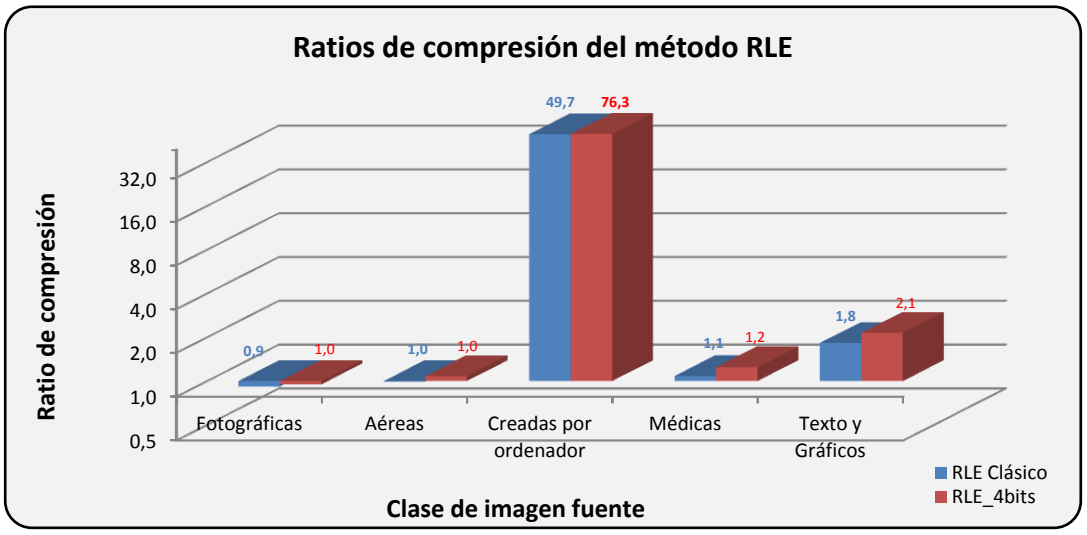


Figura 3.11. Representación de los ratios de compresión RLE(1)

Con el objetivo de ilustrar gráficamente el resto de las ratios medios correspondientes a las otras clases de imágenes, permítannos extraer del gráfico los datos de las imágenes creadas por ordenador para mejorar la representación del resto de los datos, tal y como se muestra en la Fig. 3.12.

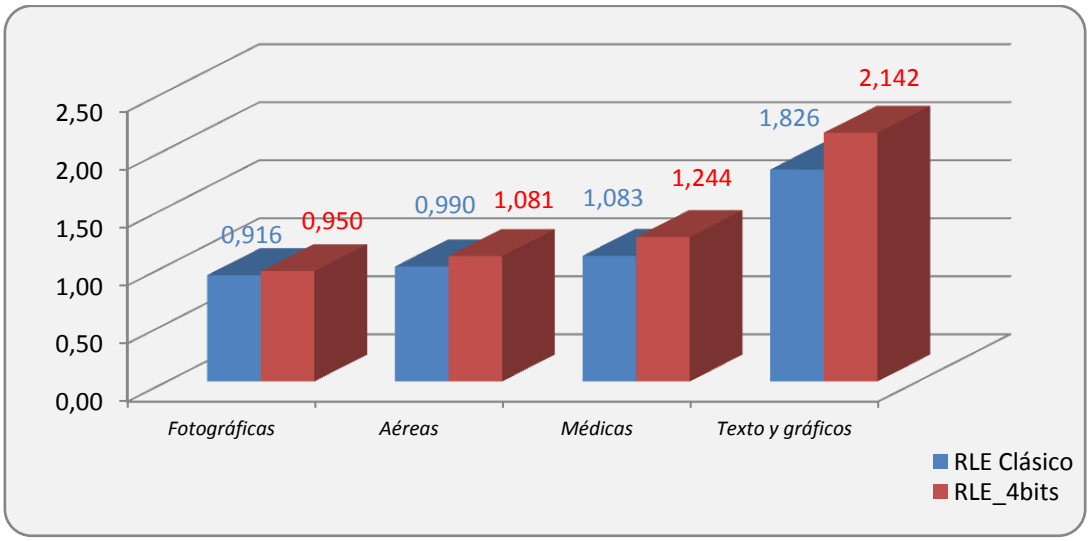


Figura 3.12. Representación de los ratios de compresión RLE(2)

En términos de consumos de memoria, el método RLE utiliza únicamente dos bytes, uno para el valor de repetición y otro para la longitud de la carrera. Por tanto, los requisitos del método son mínimos y por supuesto, la ejecución se realiza completamente en la memoria principal.

En las Tablas 3.13 y 3.14 se recogen los resultados de los tiempos y ciclos de CPU para ambos procesos.

Tabla 3-13. RLE clásico optimizado. Tiempos de ejecución de los ciclos de compresión y descompresión

IMAGEN FUENTE	NOMBRE IMAGEN	CICLO DE COMPRESIÓN		CICLO DE DESCOMPRESIÓN	
		CICLOS CPU	TIEMPO (ms)	CICLOS CPU	TIEMPO (ms)
Fotográficaas	Airplane	969.692	439,08	425.729	192,77
	Baboon	973.628	440,87	452.592	204,94
	Barbara	982.934	445,08	439.286	198,91
	Boats	1.156.923	523,86	585.200	264,98
	Cameraman	361.347	163,62	107.724	48,78
	Goldhill	1.053.618	477,09	442.735	200,47
	Lena	1.089.923	493,52	451.921	204,63
	Man	302.749	137,09	111.340	50,42
	Peppers	991.441	448,93	441.675	199,99
	Zelda	961.188	435,23	440.722	199,56
	Media	**884.344**	**400,44**	**389.892**	**176,55**
Aéreas y Satélite	Aerial	972.501	440,36	439.926	199,20
	Airfield	980.649	444,04	440.294	199,37
	Earth	156.727	70,97	76.622	34,69
	Meteosat	2.355.142	1.066,42	1.047.625	474,37
	Moon	379.733	171,95	224.122	101,48
	Moonsurface	11.413.830	5.168,26	5.456.628	2.470,80
	SanDiego	4.033.947	1.826,60	1.742.019	788,80
	WashingtonIR	19.501.026	8.830,20	8.480.190	3.839,89
	Media	**4.974.194**	**2.252,35**	**2.238.428**	**1.013,58**
Creadas por Ordenador	Circles	52.367	23,71	57.777	26,16
	Gray	157.475	71,31	108.714	49,23
	Slope	254.075	115,05	94.787	42,92
	Squares	48.055	21,76	23.705	10,73
	Media	**127.993**	**57,96**	**71.246**	**32,26**
Médicas	Elbowx	848.276	384,11	353.689	160,15
	Finger	256.677	116,23	115.440	52,27
	MRI_Brain1	251.954	114,09	116.300	52,66
	MRI_Brain2	138.898	62,89	76.950	34,84
	MRI_Brain3	390.015	176,60	185.574	84,03
	Shoulder	480.916	217,76	222.945	100,95
	Media	**394.456**	**178,61**	**178.483**	**80,82**
Textos y Gráficos escaneados	Mercados1	2.308.570	1.045,34	1.036.556	469,36
	Mercados2	2.527.692	1.144,56	1.141.171	516,73
	Mercados3	1.760.208	797,03	819.162	370,92
	Mercados4	779.541	352,98	420.596	190,45
	Media	**1.844.003**	**834,98**	**854.371**	**386,87**

Tabla 3-14. RLE 4 bits. Tiempos de ejecución del ciclo de compresión y descompresión

IMAGEN FUENTE	NOMBRE IMAGEN	CICLO DE COMPRESIÓN		CICLO DE DESCOMPRESIÓN	
		CICLOS CPU	TIEMPO (ms)	CICLOS CPU	TIEMPO (ms)
Fotográficaas	Airplane	1.138.599	515,57	535.159	242,32
	Baboon	1.242.250	562,50	598.692	271,09
	Barbara	1.157.663	524,20	579.159	262,25
	Boats	1.249.959	565,99	585.945	265,32
	Cameraman	285.260	129,17	137.259	62,15
	Goldhill	1.276.304	577,92	577.774	261,62
	Lena	1.250.967	566,45	568.861	257,58
	Man	260.919	118,15	133.098	60,27
	Peppers	1.347.240	610,04	578.246	261,83
	Zelda	1.250.163	566,08	571.244	258,66
	Media	**1.045.932**	**473,61**	**486.544**	**220,31**
Aéreas y Satélite	Aerial	1.307.486	592,04	565.762	256,18
	Airfield	1.256.673	569,03	574.676	260,22
	Earth	238.011	107,77	88.937	40,27
	Meteosat	2.919.877	1.322,14	1.379.692	624,73
	Moon	14.141.928	6.403,56	6.518.749	2.951,73
	Moonsurface	343.474	155,53	183.979	83,31
	SanDiego	4.878.831	2.209,17	2.314.384	1.047,97
	WashingtonIR	24.122.356	10.922,77	11.400.127	5.162,06
	Media	**6.151.080**	**2.785,25**	**2.878.288**	**1.303,31**
Creadas por Ordenador	Circles	54.557	24,70	37.423	16,95
	Gray	183.686	83,17	130.393	59,04
	Slope	187.448	84,88	118.470	53,64
	Squares	94.956	43,00	38.541	17,45
	Media	**130.162**	**58,94**	**81.207**	**36,77**
Médicas	Elbowx	928.653	420,50	418.817	189,64
	Finger	314.405	142,36	160.993	72,90
	MRI_Brain1	299.980	135,83	145.047	65,68
	MRI_Brain2	123.454	55,90	70.143	31,76
	MRI_Brain3	512.066	231,87	212.285	96,12
	Shoulder	546.106	247,28	343.287	155,44
	Media	**454.111**	**205,62**	**225.095**	**101,92**
Textos y Gráficos escaneados	Mercados1	2.427.710	1.099,28	1.166.053	528,00
	Mercados2	2.640.712	1.195,73	1.273.649	576,72
	Mercados3	1.986.614	899,55	961.036	435,16
	Mercados4	839.959	380,34	431.626	195,44
	Media	**1.973.749**	**893,73**	**958.091**	**433,83**

Los resultados experimentales muestran que el método RLE es eficiente en imágenes aplicaciones de imágenes médicas, texto escaneados que continen gran cantidad de blancos y en imágenes gráficas generadas por ordenador que incluyen

superficies con sólidos. En el caso de imágenes de continuos tonos, como son las imágenes fotográficas, los resultados no garantizan compresión e incluso producen menor compresión que el método *Bitmap*. Los resultados obtenidos visualizan el mismo comportamiento que el método *Bitmap* revisado anteriormente.

Obviamente, los tiempos de descompresión siempre son inferiores a los de compresión. El método clásico optimizado reduce los tiempos de compresión y descompresión en todas las imágenes procesadas. Por último, es importante tener en cuenta que los resultados del método RLE convencional sin optimizar son significativamente inferiores a los dos métodos revisados.

Claramente el método RLE alcanza ratios de compresión muy elevados en la compresión de gráficos generados por ordenador, incluso muy superiores a los métodos más representativos de la compresión de imágenes *lossless* que serán revisados en las siguientes secciones.

3.2.5 Conclusiones del método RLE

El método RLE es muy eficiente cuando los valores de los píxeles adyacentes se repiten de forma consecutiva. En este contexto, esta técnica obtiene los mejores resultados en aplicaciones de compresión de imágenes cuando estas contienen objetos con superficies de sólidos y gráficos diseñados por ordenador. En la compresión de imágenes que incluyen de texto y gráficos escaneados explotan su principal característica en las líneas con espacios en blanco producidas por los textos. Por el contrario, se observa que para codificar los dos últimos bytes precisa de cuatro bytes. En imágenes fijas de continuos tonos con cambios bruscos de valores intensidad de los píxeles puede ocasionar que el archivo comprimido sea mayor que el original. Las versiones posteriores *PackBits* y *SunRaster* utilizan secuencias de escape para mejorar los ratios de compresión.

Una de las características más sobresalientes de este método es la velocidad de procesamiento lineal de los píxeles. El algoritmo únicamente utiliza operaciones de sustitución, esto es, mover y reemplazar directamente sin necesidad de punteros ni bucles repetitivos. Estas características han contribuido a que RLE esté presente en numerosos sistemas embebidos (Hemnath y Prabhu, 2013) (Long y Xiang, 2012) y en estándares de compresión JPEG, JBIG, CCITT (*Comite Consultatif International Telegraphique et Telephonique*) etc., en los cuales el modelo genera valores consecutivos iguales a cero o cercanos a cero antes de aplicar la codificación RLE.

Las limitaciones de este método quedan claramente expuestas en las imágenes de continuos cambios de intensidad. Con respecto al modelo, la salida de datos hacia el codificador siempre está limitada a al menos un byte de identificación del píxel más cuatro bits de la longitiud de la carrera de repetición.

En las secciones 3.3 y 3.4 se presentan los métodos basados en modelos estadísticos que surgieron como alternativa a los modelos basados en técnicas de sustitución. Los métodos basados en modelos estadísticos se basan en la probabilidad de los píxeles de las imágenes sin tener en cuenta la propiedad de vecindad ni la repetición de los valores de los píxeles consecutivos.

La implementación de los métodos de compresión estadísticos por ordenador fue posterior a la implementación de los métodos de sustitución aunque las técnicas de codificación fueron formuladas en la Teoría Moderna de la Información antes de la aparición de los ordenadores.

3.3 El Codificador Huffman (Huffman Coding)

Huffman Coding (Huffman, 1952) es un método de compresión sin pérdida de datos basado en un modelo de codificación estadístico. Este método es una adaptación y optimización de *Shanon-Fano Coding*. El algoritmo de Huffman es recomendado y utilizado como método de codificación en estándares de compresión con pérdida de datos, JPEG y MPEG, en métodos archivadores de compresión, PKZIP, así como también en la compresión de datos del estándar de transmisión de datos mediante fax CCITT-1, actualmente ITU-T T.4 (*International Telecommunication Union - Telecommunications section*).

El codificador en los métodos de métodos de compresión basados en modelos de sustitución genera códigos de longitud fija. Por el contrario, el Codificador Huffman genera símbolos de longitud variable en función de la probabilidad del píxel.

El Codificador Huffman, como cualquier método basado en un modelo estadístico, puede presentar dos tipos de modelos: estático o adaptativo. En un modelo estático la tabla de probabilidades de todos los píxeles es construida antes de iniciarse el proceso de compresión, mientras que en un modelo adaptativo, la codificación de los píxeles es realizada progresivamente actualizadando el modelo y los datos simultáneamente.

En este capítulo se describe el principio de funcionamiento de ambos modelos, se aportan y exponen los resultados experimentales, y por último se analizan las conclusiones basadas en los resultados obtenidos.

3.3.1 Descripción del modelo estático de Huffman

El principio de funcionamiento del codificador estático de Huffman consiste en calcular la probabilidad de los píxeles de una imagen y codificar los píxeles de mayor probabilidad con menor número de bits que los píxeles con menor

probabilidad. La estructura de un modelo estático de Huffman es ilustrada en la Fig. 3.13.

Figura 3.13. Estructura del ciclo de compresión

En el ciclo de compresión, el píxel es la unidad de tratamiento de entrada de datos al modelo. En un modelo estático es necesario procesar previamente la imagen completa para calcular las probabilidades de los píxeles (Nelson, 1991). Las probabilidades son almacenadas en una tabla formada por una columna que registra los píxeles con valores diferentes a_i y una segunda columna que contiene sus correspondientes probabilidades $P(a_i)$. La tabla de probabilidades debe ser guardada junto con los códigos de salida para garantizar que el proceso de descompresión sea reversible y permita la recuperación exacta de la imagen original.

El tratamiento de los datos almacenados de la tabla de probabilidades se realiza mediante estructuras de árboles binarios (Nelson y Gaylly, 1995). Los píxeles con mayor probabilidad son colocados en la parte superior de la estructura del árbol binario y los píxeles con menor probabilidad son colocados en la base. El proceso de codificación binaria comienza desde el nodo raíz o nodo único situado en la parte superior del árbol. Cada nodo padre contiene dos nodos hijos los cuales son codificados atendiendo a su posición en el árbol. El nodo situado en la rama izquierda es codificado con valor "1" y el nodo de la rama derecha con "0".

El método de codificación de un nodo cualquiera consiste en recorrer el árbol binario desde el nodo raíz hasta el nodo a codificar, asociando un bit con valor "0" o "1" por cada nodo que encuentra en su recorrido. La sucesión de *bits* es almacenada en una cadena que determina el código del píxel tratado.

La codificación de cada nodo es única y el tamaño del código expresado en número de bits es variable. Por tanto, la codificación de la imagen original consiste en una sucesión de códigos de longitud variable almacenados en un fichero comprimido. La longitud L o número de bits necesarios para codificar todos los píxeles es calculada aplicando la siguiente expresión:

$$L = \sum_{i=1}^{n} r(a_i) * l(a_i) \tag{3.11}$$

donde, n: el número total de píxeles,
$r(a_i)$ es el número de apariciones del píxel,
y $l(a_i)$ es la longitud del código en bits.

En el ciclo de descompresión, la unidad de tratamiento de datos en la entrada del modelo es el bit. La secuencia de *bits* almacenada en el fichero comprimido es procesada uno a uno hasta encontrar la correspondencia biunívoca con el código de píxel. En los métodos anteriores, la decodificación era inmediata a partir del código almacenado en el fichero comprimido. En el método estático no es posible porque Huffman debe reconstruir la estructura de árbol binario a partir de la tabla de probabilidades. Obviamente, el fichero generado en el ciclo de descompresión debe ser el mismo que el obtenido en el proceso de compresión. La Fig. 3.14 ilustra el proceso de descompresión.

Figura 3.14. Estructura del ciclo de descompresión

A continuación se describen detalladamente los procesos de compresión y descompresión de este método.

3.3.2 Proceso de compresión

Inicialmente, la imagen completa es procesada píxel a píxel recorriendo las filas de arriba hacia abajo y las columnas de izquierda a derecha. En algunas versiones de este método, la imagen es segmentada en bloques de 8x8 píxeles para facilitar los cálculos y reducir los tiempos de ejecución (Sayood, 2012).

El resultado de este procesamiento es una tabla de probabilidades que agrupa los píxeles por su valor asignándoles sus correspondientes probabilidades. Esta tabla es clasificada en orden decreciente de probabilidades.

$$P(a_1) \geq P(a_2) \geq P(a_3) \geq \ldots \geq P(a_n) \qquad (3.12)$$

El proceso de codificación genera los códigos óptimos de Huffman en función de las probabilidades de los píxeles. El codificador asigna códigos más cortos a los píxeles con mayor probabilidad y códigos con mayor longitud de bit a los píxeles con menor probabilidad. La longitud *L* de los códigos satisface la siguiente expresión:

$$l(a_1) \geq l(a_2) \geq l(a_3) \geq \ldots \geq l(a_n) \qquad (3.13)$$

El proceso de codificación construye una estructura de árbol binario a partir de la tabla de probabilidades. El codificador puede presentar dos aspectos constructivos: árbol binario completo o incompleto. Un árbol binario completo tiene la propiedad de asociar a cada nodo padre dos nodos hijos y esto significa que todos los píxeles están representados con su correspondiente nodo en la base del árbol binario (Nelson, 1991). Por el contrario, un árbol binario incompleto es aquel que le falta algún nodo hijo, y por tanto, los nodos de los píxeles están repartidos en nodos a lo largo del árbol. En la Fig. 3.15 se muestran los dos posibles aspectos constructivos de los árboles binarios a partir de la tabla de probabilidades: Tabla 3.15. En ambas estructuras, el número de nodos *N* resultante es el mismo y es calculado mediante la siguiente expresión:

$$N = 2 * p - 1 \tag{3.14}$$

Donde, p: número de píxeles diferentes contenidos en la tabla de probabilidades

Tabla 3-15. Tabla de probabilidades de Huffman

VALOR DEL PÍXEL	PROBABILIDAD
15	0,05
18	0,05
21	015,
23	0,05
24	0,05
25	0,65

La representación del arbol binario completo o incompleto se obtiene a partir de la tabla de probabilidades, según se muestra en la Fig. 3.15.

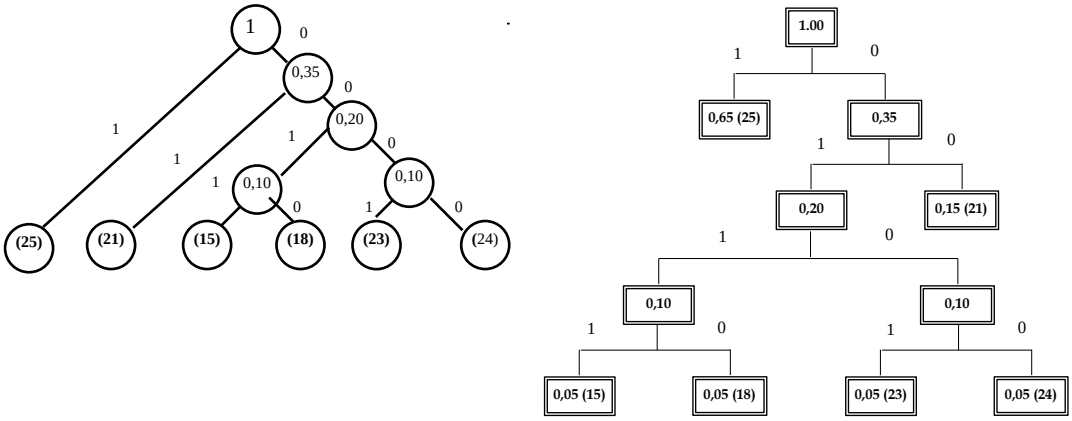


Figura 3.15. Árbol binario completo e incompleto

Todos los píxeles están representados por su correspondiente nodo en el árbol binario. El código de Huffman de cada nodo consiste en recorrer la estructura de

árbol binario desde el nodo raíz hasta el nodo a codificar. Por cada nodo atravesado añade un bit. El valor de este bit es uno si pertenece a la rama izquierda y cero si está posicionado en la otra rama del nodo. La concatenación secuencial de estos bits produce el código del nodo.

Aplicando la codificación Huffman a las dos estructuras de árbol binario, se demuestra que aunque los códigos generados son diferentes mantienen la propiedad de ser únicos para cada píxel, y por tanto, garantizan el mismo ratio de compresión en ambas estrategias de construcción del árbol binario. La Tabla 3.16 refleja la codificación para las estructuras de árboles binarios completo e incompleto.

Tabla 3-16. Codificación del árbol binario completo e incompleto

VALOR DEL PÍXEL	PROBABILIDAD	CODIFICACIÓN DEL ÁRBOL BINARIO COMPLETO	CODIFICACIÓN DEL ÁRBOL BINARIO INCOMPLETO
25	0,65	1	1
21	0,15	01	00
18	0,05	0010	0110
23	0,05	0001	0001
24	0,05	0000	0000
15	0,05	0011	0111

La codificación del árbol binario de Huffman debe cumplir dos propiedades: la primera, generar códigos de longitud variable, y la segunda, asignar símbolos únicos a cada nodo del árbol. Estas dos propiedades garantizan una decodificación reversible.

El modelo estático produce un código óptimo porque el tamaño medio de cada código expresado en bits se aproxima a la entropía de cada píxel. La longitud de cada código contenido en la tabla de probabilidades es calculada mediante la siguiente expresión:

$$l_i = -\log_2 P(a_i) \tag{3.15}$$

La longitud media del código depende de número de píxeles y de sus probabilidades. Esta longitud es expresada en términos de entropía por:

$$H(S) \leq 1 \leq H(S) + \frac{1}{N} \tag{3.16}$$

donde N es el número de elementos del mensaje o fuente

Un código de Huffman no puede alcanzar el valor de la entropía H(S) porque no permite "bits fraccionales", necesita al menos un bit por píxel. Esto significa, que dada una fuente binaria de probabilidades p_1= *0,99* y p_2= *0,01* la longitud óptima

para el primer símbolo es *–log(0,99)= 0,0145*. El código Huffman asignará un código de un bit al píxel, "desperdiciando" 0,9815 bits, casi el 100%.

El fichero comprimido resultante contendrá la tabla de probabilidades y la sucesión de códigos de longitud variable para cada uno de los píxeles. A continuación se describe el algoritmo de Huffman en el proceso de compresión.

3.3.2.1 Algoritmo del ciclo de compresión

El algoritmo estático construye la tabla de probabilidades para cada píxel diferente. La tabla es ordenada en función de sus probabilidades para generar una estructura de árbol binario. Los códigos de salida se obtienen aplicando una codificación basada en una estructura de árbol binario completo. A continuación se detallan estos pasos.

Paso 1. Calcula la probabilidad de aparición de cada valor o píxel "p_i" y genera una tabla ordenada en orden decreciente de probabilidades asignándoles un peso que corresponda a su probabilidad.

Paso 2. Coloca los dos primeros valores de menor probabilidad y crea un nodo padre para esos dos nodos.

Paso 3. Asigna al nodo padre un peso igual a la suma de las probabilidades de los nodos hijos

Paso 4. Elimina los dos nodos de la lista y añade el nodo padre.

Paso 5. Busca los siguientes valores con menor probabilidad y crea un nodo padre calculando la suma de sus probabilidades y generando un nivel más alto hasta que todas las líneas emerjan en el nodo raíz que tendrá una probabilidad igual a uno.

Paso 6. Una vez el árbol es construido, el procedimiento de codificación de cada nodo píxel consiste en recorrer el árbol desde el nodo raíz a los nodos de menor probabilidad, asignando un "1" a un lado del nodo y un "0" al otro y aplicando el mismo criterio en todos los nodos del árbol. Por tanto, los códigos y su longitud dependen del camino seleccionado.

3.3.2.2 Programación del algoritmo

La codificación de un píxel es la secuencia completa de bits generados al recorrer el árbol binario desde el nodo raíz al píxel a codificar o nodo que lo representa en el árbol binario. En la práctica, la implementación de este algoritmo se realiza desde el nodo inicial a codificar hasta el nodo raíz. Por cada nodo atravesado se añade un bit en la posición MSB y después efectúa una operación de desplazamiento de un bit de desplazamiento a la derecha.

Para facilitar la programación se incorpora una estructura de nodo compuesta por cuatro valores. Estos valores correspondes al número de nodo, a un peso o probabilidad del nodo, y a dos valores que representan los valores y posiciones de los nodos hijos del nodo actual. En el caso del nodo inicial los valores de los nodos hijos son cero. La estructura de cada nodo es la siguiente:

[número de nodo; peso; nodo hijo rama izquierda; nodo hijo rama derecha]

```
// Huffman Coding
/*******  1. Contruir la tabla de probabilidades ******/
for(i=0; i<256; i++) {
   value = array_píxel[i];
   tabla_prob[value]++ ;
}
nmax=i;
/*****  2. Clasifica la tabla de probabilidades ******/
contador=0;
for (i=0;f*c; i++) { // f*c número de píxeles en la imagen
   for ( j=1;  f*c-1; j++) {
       if (tabla_prob[i]< tabla_prob[j])and(tabla_prob[i]<>0){
          cambia_prob= tabla_prob[i];
          cambia_ind =  tabla_ind[i];
          tabla_prob[i] = tabla_prob[j];
          tabla_ind[i] = tabla_ind[j];
          tabla_prob[j] = cambia_prob;
          tabla_ind[j] = cambia_ind;
          contador ++1;
         }
    }
}
/**************  3. Construir árbol binario ******/
counter=0;
//Inicializa_codigos con valores iniciales
//Estructuraparanodosiniciales(nodo, eso=probabilidad,0)
for (i=0;nmax; i++){
    crea_estru_nodo(i, tabla_prob[i], 0) }
    // Siguientes nodos, operaciones recursivas
    // Actualiza codigo y función recursiva
    // Primero lee nodo izquierda. Desplazamiento a la izda .
    // Colocar bit "0"
    // Desplazar código una posición a la izdqa
    // Incrementar peso del nodo
    shift "codigo"
    //  Recursión para el nodo izquierdo
    crea_codigo(actual_nodo-> (nodo_izqda, peso, codigo);
    // Recursión nodo derecha, colocar "1" en LSB
    code = code OR 1; //In c: "code|=1;"
    // recursiónparanodododerecha crea_codigo(nodo_actual-
hijo_dcha,peso,codigo);  } }
// Insertar nuevo nodo con dos elementos de menor probab.
// Volver al inicio del bucle hasta finalizar la tabla
//Vuelve a la raíz del árbol
```

Listado 3.5. Programación del algoritmo de compresión de Huffman

En los últimos años, una versión modificada de la codificación de Huffman ha surgido como alternativa práctica al tradicional sistema de codificación. La principal ventaja de este nuevo método reside en la reducción de operaciones recursivas mediante la compactación de códigos. Este método conocido como Código de Huffman Canónico (*Canonical Huffman Codes*) y se caracteriza por utilizar la longitud del código obtenida mediante Huffman como criterio de codificación para obtener los nuevos códigos canónicos (Hu y Tucker, 1971).

3.3.2.3 Flujograma del algoritmo de compresión

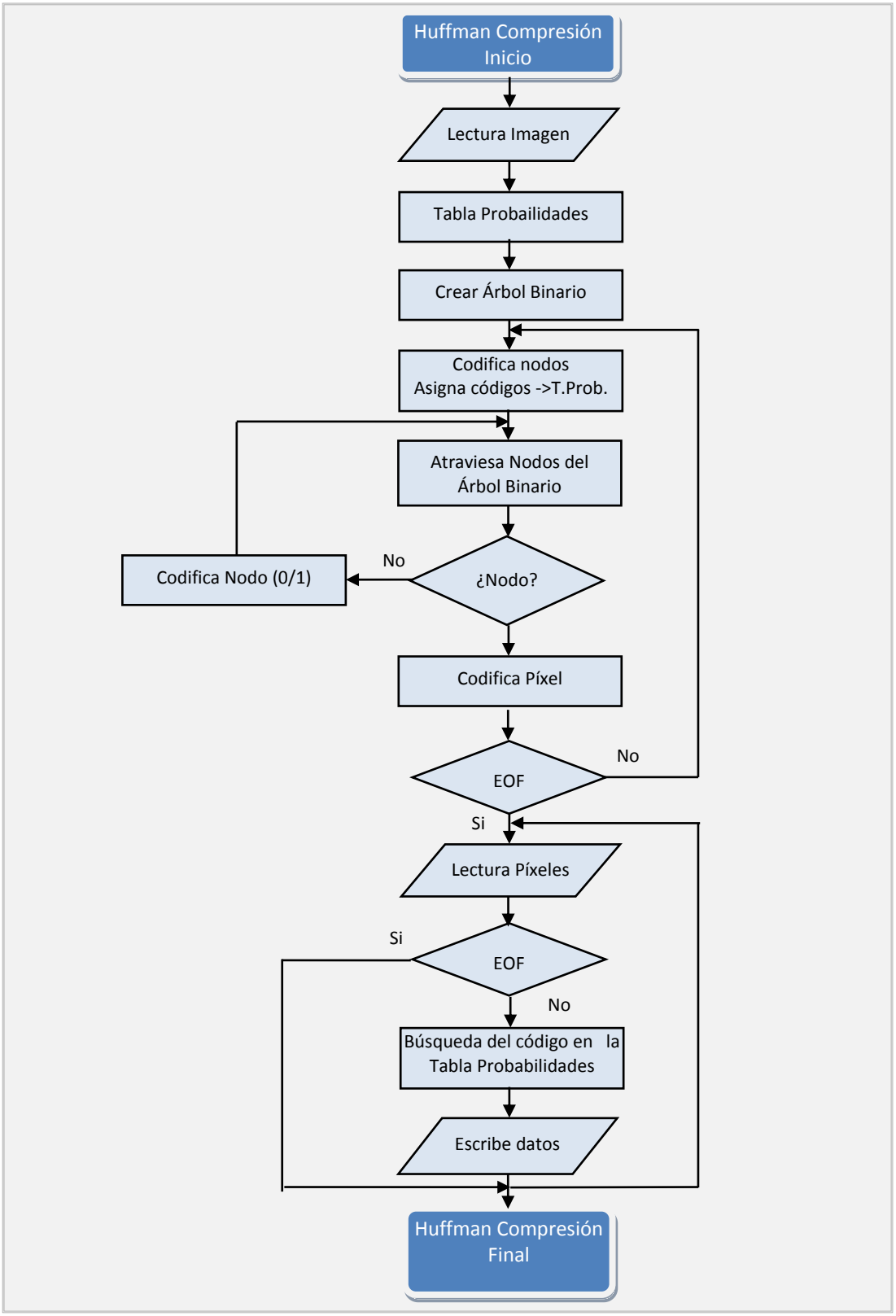


Figura 3.16. Flujograma del proceso de compresión

3.3.2.4 Ejemplo práctico del proceso de compresión

Dado un bloque de 100 píxeles, la Tabla 3.17 describe de forma práctica el procedimiento para obtener los códigos de salida del proceso de compresión.

Tabla 3-17. Valores de los píxeles de un bloque cualquiera de una imagen

25	25	25	25	25	25	25	25	25	25
25	25	25	25	25	25	25	25	25	25
25	25	25	25	25	25	25	25	25	25
25	25	25	21	21	21	23	24	18	15
25	25	25	21	21	21	23	24	18	15
25	25	25	21	21	21	23	24	18	15
25	25	25	21	21	21	23	24	18	15
25	25	25	21	21	21	23	24	18	15
25	25	25	25	25	25	25	25	25	25
25	25	25	25	25	25	25	25	25	25

El algoritmo Huffman comienza la compresión realizando una lectura previa del bloque para calcular las probabilidades de los píxeles.

Paso 1) Construir la tabla de probabilidades en orden descendente de probabilidades

Tabla 3-18. Tabla de probabilidades de Huffman

VALOR DEL PÍXEL	PROBABILIDAD
24	0,05
23	0,05
18	0,05
15	0,05
21	0,15
25	0,65

Paso 2) Generar una estructura de árbol binario a partir de la tabla de probabilidades. Este árbol puede diseñarse horizontalmente o verticalmente. En ambos formatos, los píxeles con menor probabilidad son colocados en la base de la estructura y progresa horizontalmente o verticalmente creando nodos padres en función de las probabilidades, hasta llegar al nodo raíz.

Paso 3) Procesar el árbol binario para determinar los códigos binarios correspondientes a cada nodo. Desde el nodo raíz, progresando hacia niveles inferiores y de izquierda a derecha trazando el camino hasta el nodo. A cada nodo atravesado se le asigna un uno si el camino se dirige por la rama izquierda y un cero si es por la derecha.

Paso 4) Aplicando el algoritmo de tratamiento del árbol binario se obtienen los siguientes nodos y codificación.

Tabla 3-19. Generación y codificación de los nodos

PÍXEL	NODO	ESTRUCTURA DEL NODO
24	1	[1; 0,05; 0]
23	2	[2; 0,05; 0]
18	3	[3; 0,05; 0]
15	4	[4; 0,05; 0]
21	5	[5; 0,15; 0]
25	6	[6; 0,65; 0]

A partir de la tabla de numeración y peso de los nodos se genera el árbol binario tal y como se muestra en la Fig. 3.17.

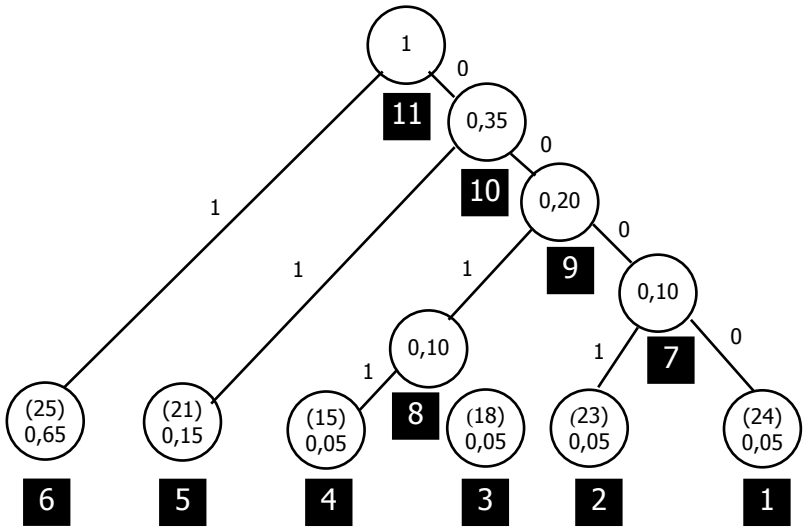


Figura 3.17. Árbol binario de Huffman

A partir de los nodos iniciales, los códigos de los siguientes nodos se obtienen mediante operaciones recursivas de búsqueda de nodos usando operaciones de punteros y desplazamientos. La Fig. 3.18 muestra las operaciones de sustitución en un de árbol horizontal.

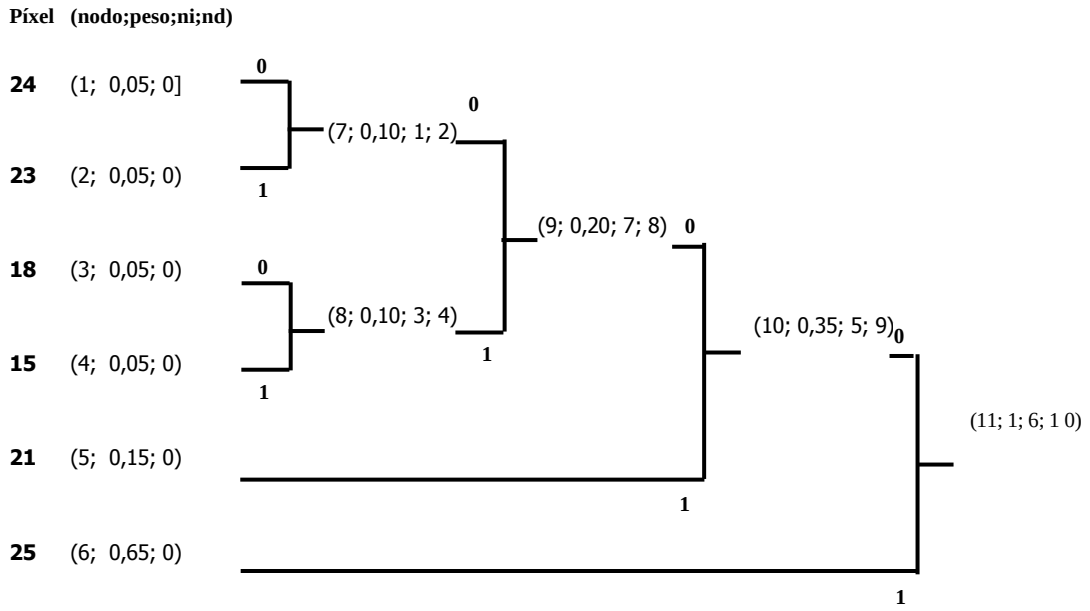


Figura 3.18. Codificación horizontal de Huffman

La Tabla 3.20 ilustra el proceso de codificación paso a paso para facilitar la comprensión de la programación y de las operaciones recursivas necesarias para obtener el código.

Tabla 3-20. Codificación horizontal de Huffman

NÚMERO NODO	ESTRUCTURA DEL NODO	CODIFICACIÓN	
		RAMA IZQUIERDA	RAMA DERECHA
7	[7; 0,10; 1; 2]	Nodo 1-> **0**	Nodo 2-> **1**
8	[8; 0,10; 3; 4]	Nodo 3-> **0**	Nodo 4-> **1**
9	[8; 0,20; 7; 8]	Nodo 7 (Nodo 1, Nodo 2) Nodo 1-> **00** - Nodo 2-> **01**	Nodo 8 (Nodo 3, Nodo 4) Nodo 3-> **10** – Nodo 4-> **11**
10	[10; 0,35; 5; 9]	Nodo 5-> **1**	Nodo 9 (Nodo 7 , Nodo 8) Nodo 7(Nodo 1, Nodo 2) Nodo 1-> **000 -** Nodo 2->**001** Nodo 8 (Nodo 3, Nodo 4)
11	[11; 1; 6; 10]	Nodo 6-> **1**	Nodo 10(Nodo 5, Nodo 9) Nodo 5->**01** Nodo 9(Nodo 7,Nodo 8) Nodo7 (Nodo 1, Nodo 2) Nodo 1->**0000 –** Nodo 2->**0001** Nodo 8 (Nodo3, Nodo4) Nodo 3 ->**0010** – Nodo 4->**0011**

Paso 5) El código asignado a cada píxel es incluido en la tabla de probabilidades de forma temporal para el acceso indexado a cada píxel según se ilustra en la Tabla 3.21.

Tabla 3-21. Tabla indexada por nodos

VALOR PÍXEL	NÚMERO DE NODO	PROBABILIDAD SÍMBOLO	CODIFICACIÓN
24	1	0,05	0000
23	2	0,05	0001
18	3	0,05	0010
15	4	0,05	0011
21	5	0,15	01
25	6	0,65	1

Paso 6) Escribir el fichero comprimido mediante la sustitución de cada píxel en la imagen original por su correspondiente codificación binaria contenida registrada en la tabla de probabilidades. El resultado de la codificación es una sucesión de bits formada por códigos únicos de longitud variable. La longitud total resultante de la codificación del ejemplo propuesto es igual a 175 bits.

"1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;01;01;01;0001;0000;0010;0011;1;1;1;01;01;01;0001;0000;0010;0011;1;1;1;01;01;01;0001;0000;0010;0011;1;1;1;01;01;01;0001;0000;0010;0011;1;1;1;01;01;01;0001;0000;0010;0011;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;"

El modelo estático produce un código óptimo porque el tamaño medio de cada código expresado en bits se aproxima a la entropía de cada píxel. La entropía de este bloque se obtiene aplicando la expresión (3.1).

$$H(S) = \sum_{i=1}^{n} P_i \log_2 P_i = 167{,}889 \text{ bits} \qquad (3.17)$$

La longitud L obtenida en el ejemplo se aproxima a la entropía y corresponde con el resultado de la expresión (3.11).

$$L = \sum_{i=1}^{n} r(a_i) * c(a_i) = (5*4) + (5*4) + (5*4) + (5*4) + (15*2) + (65*1) = 175 \text{ bits} \qquad (3.18)$$

La longitud del código de salida está muy cerca del valor de la entropía, por lo tanto la compresión es aceptable.

Paso 7) El fichero comprimido contiene la tabla de probabilidades. El tamaño de la tabla de probabilidades *Tp* expresado en bytes para una imagen de un byte de profundidad, se calcula mediante la siguiente expresión:

$$T_p = 2 * n \tag{3.19}$$

*siendo, **n**, número de píxeles.*

Aplicando la expresión (3.19) obtenemos que el tamaño de la tabla de probabilidades tiene una longitud de 12 bytes. El tamaño total del bloque comprimido será igual a la suma del tamaño del fichero comprimido más el tamaño de la tabla de probabilidades. Esto es:

$$T_t = \frac{175 \text{ bits}}{8} + 12 \text{ bytes} = 34 \text{ bytes} \tag{3.20}$$

Por tanto, el ratio de compresión supone un 34% de reducción de almacenamiento.

$$R_c = \frac{\text{Tamaño Imagen Fuente}}{\text{Tamaño Imagen Comprimida}} = \frac{100 \text{ bytes}}{34 \text{ bytes}} = 2{,}95 \text{ bytes} \tag{3.21}$$

El método de compresión *Huffman* consigue excelentes ratios de compresión. Sin embargo, los tiempos de ejecución son elevados porque requiere procesar la tabla dos veces: la primera para calcular las probabilidades de los píxeles y la segunda para codificarlos. El proceso de codificación es muy eficiente pero para recuperar los datos en el proceso de descompresión es necesario almacenar previamente la tabla de probabilidades, y esto supone un considerable incremento del tamaño del fichero comprimido.

En la siguiente sección, se describe el proceso de descompresión del método Huffman aplicando la misma estructura de revisión y ejemplo práctico que en el proceso de compresión.

3.3.3 Proceso de descompresión

En términos generales, un proceso de descompresión siempre es más rápido que su correspondiente proceso de compresión. En el caso del Codificador Huffman esta premisa también se cumple aunque en el deCodificador Huffman sea necesario reconstruir el árbol binario a partir de la tabla de probabilidades para generar el código de cada símbolo. El proceso completo se simplifica porque no es necesario calcular la tabla de probabilidades.

El proceso de descompresión utiliza la tabla de probabilidades para construir la estructura de árbol binario que corresponde a esa tabla. Los códigos de salida para cada píxel se obtienen aplicando el mismo procedimiento de codificación y construcción del árbol binario que en el proceso de compresión.

Una vez obtenidos los códigos, el decodificador procede a la lectura secuencial del fichero comprimido. La unidad de tratamiento en este proceso es el píxel. Cada vez que un píxel es leído en la entrada, comprueba si existe su correspondiente código en la tabla de probabilidades. En caso afirmativo, coloca el valor del píxel en el fichero comprimido y a continuación desplaza el puntero una posición. En caso

contrario, concatena el valor del bit con el anterior. El proceso se repite hasta la finalización del fichero comprimido. Todos los valores de los píxeles son recuperados de forma reversible reconstruyendo la imagen original.

3.3.3.1 Algoritmo del ciclo descompresión

El algoritmo aplica los mismos primeros pasos (Paso-1 hasta Paso-6) exactamente igual que en el algoritmo de compresión y a continuación procede a la lectura de los códigos.

Paso 1. Genera la estructura de árbol binario a partir de la tabla de probabilidades
Paso 2. Coloca los dos primeros valores de menor probabilidad y crea un nodo padre para esos dos nodos.
Paso 3. Asigna al nodo padre un peso igual a la suma de las probabilidades de los nodos hijos
Paso 4. Elimina los dos nodos de la lista y añade el nodo padre.
Paso 5. Identifica los siguientes valores con menor probabilidad y crea un nodo padre calculando la suma de sus probabilidades y generando un nivel más alto hasta que todas las líneas emerjan cuya probabilidad será igual a uno.
Paso 6. Una vez el árbol binario es completado, el procedimiento de codificación consiste en recorrer el árbol desde el nodo raíz a los nodos de menor probabilidad, asignando un "1" a un lado del nodo y un "0" al otro aplicando el mismo criterio en todos los nodos del árbol. Por tanto, los códigos y su longitud en bits dependen del camino seleccionado y son almacenados temporalmente en la tabla de probabilidades.
Paso 7. Lectura secuencial de los bits del fichero comprimido. Concatena el valor del bit en una cadena de texto.
¿Existe la cadena como código en la tabla de probabilidades?
Sí.
Paso 7.1. Escribe el valor del píxel en el fichero descomprimido.
No.
Paso 7.2. Concatena el bit en la cadena.
Paso 8. Repite el paso anterior hasta detectar final del fichero comprimido

3.3.3.2 Programación del algoritmo del ciclo de descompresión

El proceso de descompresión comienza con la lectura de la tabla de probabilidades que contiene cada símbolo con su correspondiente probabilidad y a continuación la lectura del ficharo comprimido. A partir de este punto, Huffman construye el árbol binario para generar los códigos. Por tanto, el núcleo de la programación del algoritmo es el mismo que en el compresor excepto el proceso inicial de la apertura y lectura de la tabla de probabilidades y del fichero de datos comprimidos.

3.3.3.3 Flujograma del proceso de descompresión

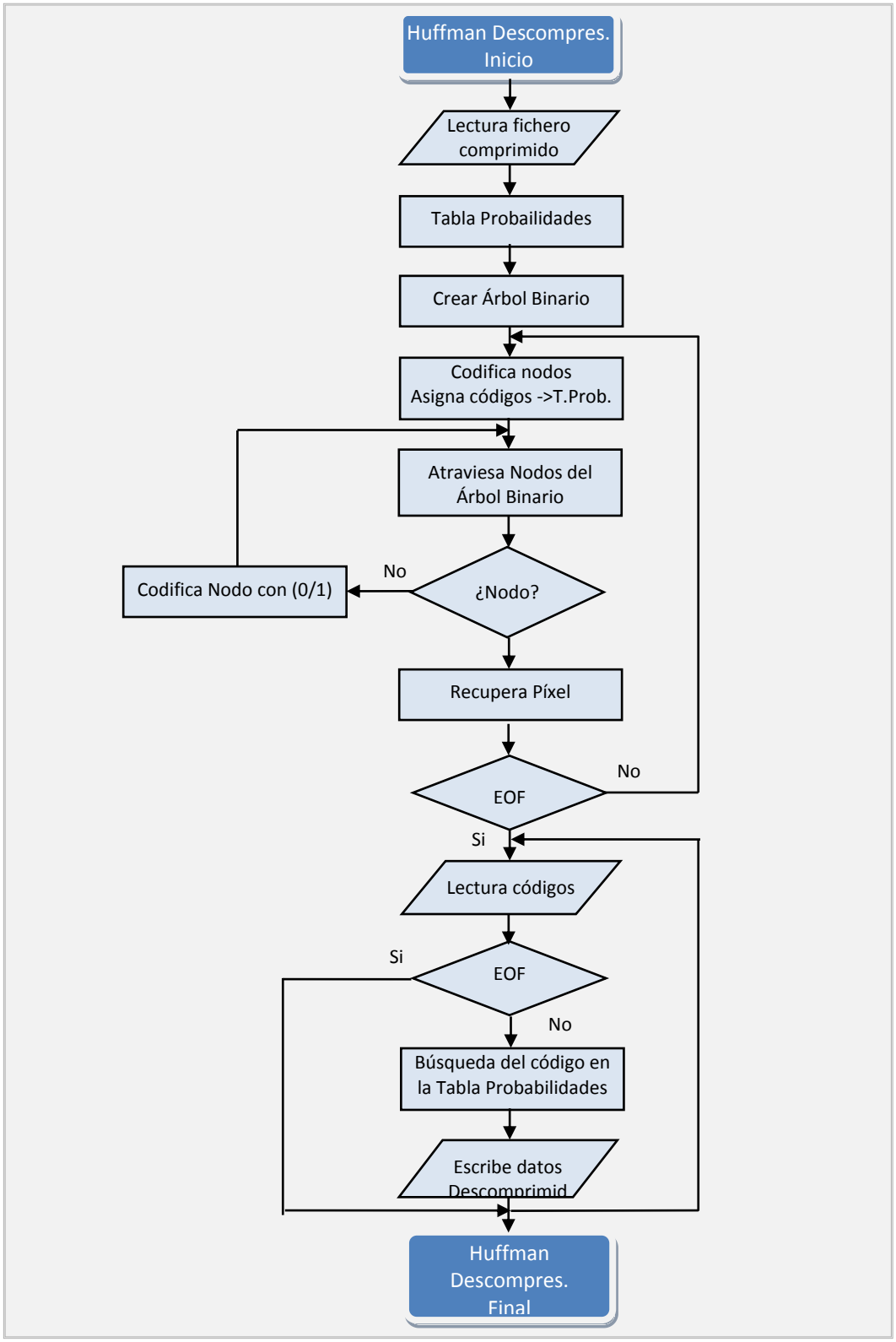


Figura 3.19. Proceso de descompresión de Huffman

3.3.3.4 Ejemplo práctico del proceso de descompresión

A partir de los resultados obtenidos en el ejemplo propuesto en el proceso de compresión se describe un ejemplo práctico de expansión de los datos comprimidos.

Paso 1) Genera la misma estructura de árbol binario que el algoritmo de compresión obteniendo los mismos códigos (Fig. 3.20).

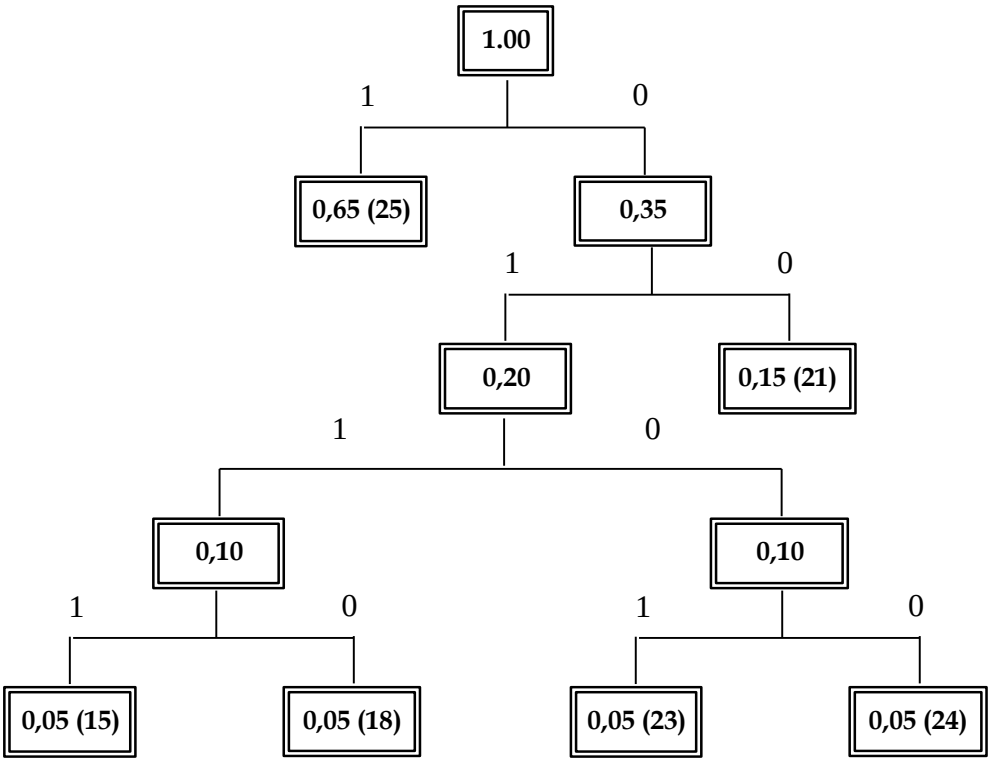


Figura 3.20. Árbol binario de Huffman

Paso 2) Registra los códigos temporales en la tabla de probabilidades, Tabla 3.22.

Tabla 3-22. Tabla de códigos temporal de nodos

PÍXEL	PROBABILIDAD	CODIFICACIÓN
15	0,05	0111
18	0,05	0110
21	0,15	00
23	0,05	0101
24	0,05	0100
25	0,65	1

Paso 3) Lectura del fichero comprimido a nivel de bit. Inicialmente procesa el primer bit y el puntero se posiciona en el primer símbolo que comience por el valor del bit en la tabla de probabilidades (Fig. 3.21).

Fichero Comprimido
"1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1, 00,00,00,0101,0100,0110, 0111, 1, 1,1,00, 00,00,0101,0100, 0110, 0111,1,1,1,00,00,00,0101,0100,0110,0111 1,1,1, 0,00,00, 0101 ,0100,0110, 0111,1,1,1,00,00,00,0101,0100,0110 ,0111,1,1,1,1 ,1,1, 1,1,1,1,1,1,1,1,1,1,1,1, 1,1,1"

Píxel	Probabilidad	Códigos
15	0,05	0111
18	0,05	0110
21	0,15	00
23	0,05	0101
24	0,05	0100
25	0,65	1

Figura 3.21. Acceso indexado a la codificación de los nodos

Si existe el código entonces escribe el valor del píxel en el fichero descomprimido. En caso contrario, añade el valor del bit a una cadena inicialmente vacía y procede a leer el siguiente bit del fichero comprimido. Esta operación se repite bit a bit hasta finalizar el fichero. Una vez detectado fin de fichero el fichero comprimido, la imagen original es recuperada sin pérdida de datos (Tabla 3.23).

Tabla 3-23. Recuperación de los datos originales

25	25	25	25	25	25	25	25	25	25
25	25	25	25	25	25	25	25	25	25
25	25	25	25	25	25	25	25	25	25
25	25	25	21	21	21	23	24	18	15
25	25	25	21	21	21	23	24	18	15
25	25	25	21	21	21	23	24	18	15
25	25	25	21	21	21	23	24	18	15
25	25	25	21	21	21	23	24	18	15
25	25	25	25	25	25	25	25	25	25
25	25	25	25	25	25	25	25	25	25

El proceso de descompresión es más rápido que el proceso de compresión porque no tiene que realizar una lectura previa para calcular la tabla de probabilidades.

Los modelos adaptativos o dinámicos obtienen mejores resultados en términos de ratio de compresión. Por el contrario, los tiempos de ejecución se incrementan considerablemente al tener que actualizar el árbol binario por cada entrada de datos.

Para optimizar la construcción del árbol binario, el modelo adaptativo en algunas variaciones de este método construye un árbol inicializando todos los valores de los píxeles con todos los valores posibles, desde 0 a 255, y con un peso igual a uno. En algunas aplicaciones, esto provoca que el modelo pueda generar nodos que no son utilizados posteriormente. Para corregir la generación de nodos inexistentes, se inicializa una tabla vacía y los nuevos valores se incorporan a medida que son leídos con una secuencia de escape.

En la siguiente sección se describe el modelo de Huffman adaptativo y se analizan las ventajas y desventajas de este modelos en comparación con el modelo estático.

3.3.4 Codificador Huffman adaptativo

Los codificadores de Huffman adaptativos (Knuth, 1985) (Vitter, 1987) optimizan el método de compresión estático de *Huffman*. Estos métodos permiten realizar el proceso de codificación en un solo paso, basándose en las estadísticas de los datos procesados con anterioridad. El proceso de compresión no tiene conocimiento previo de los datos. El árbol de Huffman se actualiza por cada entrada de datos, teniendo en cuenta los datos procesados anteriormente y sin disponer de información sobre los futuros datos a procesar (Vitter, 1988).

La unidad de tratamiento en la entrada de datos es el píxel. Cuando un píxel es procesado, comprueba si el píxel ha sido leído anteriormente o no. En caso afirmativo, aumenta en uno el peso del nodo y regenera el árbol desde el nodo actual hasta el nodo raíz. En caso contrario, genera un nuevo nodo identificado con un número de nodo y le asigna un peso inicial igual a uno. La codificación del árbol se realiza desde la raíz hasta el nivel más bajo codificando con "0" o "1" las ramas del nodo (Fig. 3.22).

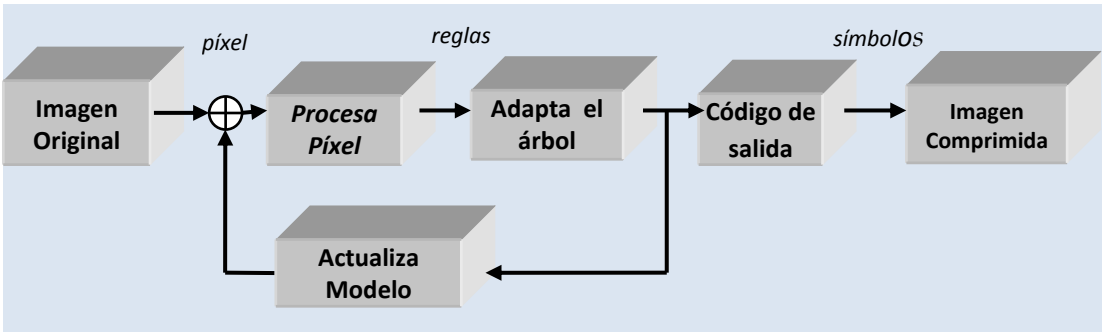


Figura 3.22. Modelo del algoritmo Huffman adaptativo

Un modelo de compresión de datos adaptativo en un primer paso inicializa el modelo. En un segundo paso, por cada símbolo de entrada de datos en el modelo codifica el símbolo actualizando el código y el modelo. La compresión Huffman

adaptativa se diferencia de la versión estática en que los datos estadísticos se generan a la vez que se obtienen los códigos de salida.

La estructura del árbol binario de *Huffman e*s completa y cumple la propiedad de hermandad (Nelson, 1991). Esto significa que al organizar los nodos del árbol en orden creciente atendiendo al peso de los nodos, cada nodo aparece junto a su hermano.

La problemática del algoritmo adaptativo reside en cómo se actualiza el modelo. Esto significa reconstruir o modificar íntegramente el árbol binario. Este procedimiento provoca que los tiempos de ejecución sean extremadamente lentos. Una solución óptima consiste en actualizar únicamente la parte del árbol binario que es afectada (Knuth, 1971) y (Knuth, 1984). La clave para resolver esta problemática es la introducción de un nodo vacío o nodo especial denominado NYT (*Not Yet Transmitted*).

La manipulación del árbol binario debe cumplir las siguientes propiedades:

- Cada nodo tiene un asignado un peso y un número de nodo único
- Los nodos con más peso tienen un número de nodo de orden más alto
- Un nodo padre siempre tiene mayor peso, y por tanto, mayor número de orden. Obviamente el nodo raíz es el nodo de mayor peso y más alto número de orden.
- En cada nivel, el nodo situado más a la derecha tiene mayor orden aunque tengan el mismo peso
- Cada nodo hijo contiene un valor, excepto el nodo NYT o nodo donde todos los nuevos caracteres son añadidos
- Los nodos internos contienen el mismo peso que la suma de los pesos de los nodos hijos
- Todos los nodos del mismo peso pertenecen al mismo bloque y son ordenados consecutivamente.

El árbol construido que cumple estas propiedades es denominado árbol binario de Huffman. Esto es, cumple con la propiedad de numerar los nodos de izquierda a derecha desde el nivel inferior al nodo raíz, en orden creciente de incremento de peso.

El método aplica diferente procedimiento de codificación para los píxeles nuevos que para los píxeles codificado anteriormente. En caso de un píxel nuevo, la codificación identifica este nodo mediante un código NYT con valor nulo (Cormack y Horspool, 1984). En el caso de los píxeles codificados anteriormente simplemente se actualiza el código del píxel.

3.3.4.1 Algoritmo de compresión de Huffman adaptativo

Paso 1. Calcula el número total de nodos del árbol binario.
Paso 2. Inicializa el árbol binario con un nodo, conocido como código de escape o NYT (Not-Yet-Transmitted).
Paso 3. Lee píxel en la entrada del modelo
Paso 4. NO EXISTE
Paso 4.1 Genera dos nuevos nodos hijos: el izquierdo con peso igual a uno y el derecho con peso igual a cero
Paso 4.2 La suma de los dos pesos actualiza el nodo de nivel superior y el número de nodo EXISTE
Paso 4.3. Incrementa en uno el peso del nodo
Paso 4.4 Actualiza el árbol binario de forma recursiva desde el nodo raíz
Paso 5. Volver al paso 2 hasta final de fichero
Paso 6. Escribir fichero comprimido

El algoritmo de actualización del árbol binario (propiedad *sibling*) es el siguiente:

Paso 4.4.1. Sea N el píxel asignado al nodo P con peso X.
Paso 4.4.2 ¿P es el nodo raíz?
- Si -
Paso 4.4.3. Incremeta el peso X en uno y finalizar.
- No –
Paso 4.4.4. Intercambiar nodo P con el nodo superior de igual peso excepto que sea su nodo padre.
Paso 4.4.5. Aumentar el peso de X en uno.
Paso 4.4.6. Saltar al paso 4.4.2 hasta llegar al nodo raíz y de esta forma actualizar los nodos superiores y regenerar el árbol.

3.3.4.2 Programación del algoritmo de Huffman adaptativo

En el siguiente código se aporta una programación simplificada del algoritmo de Huffman adaptatitvo que soporta la propiedad *sibling*.

```
1  // Algoritmo Huffman adaptativo
2  // Propiedad Sibling o de hermandad
3  nodo[I]++;
4  if (node[i] > nodo[i+1])
5    {
6    val = nodo[i+1];
7    j = i+1;
8    while (nodo[j] == símbolo)
9    {
10       j++;
11   }
12   cambia (nodo[i], nodo[j-1]);
13 }
```

Listado 3.6. Programación del algoritmo de descompresión de Huffman adaptativo

3.3.4.3 Flujograma del algoritmo de compresión adaptativo

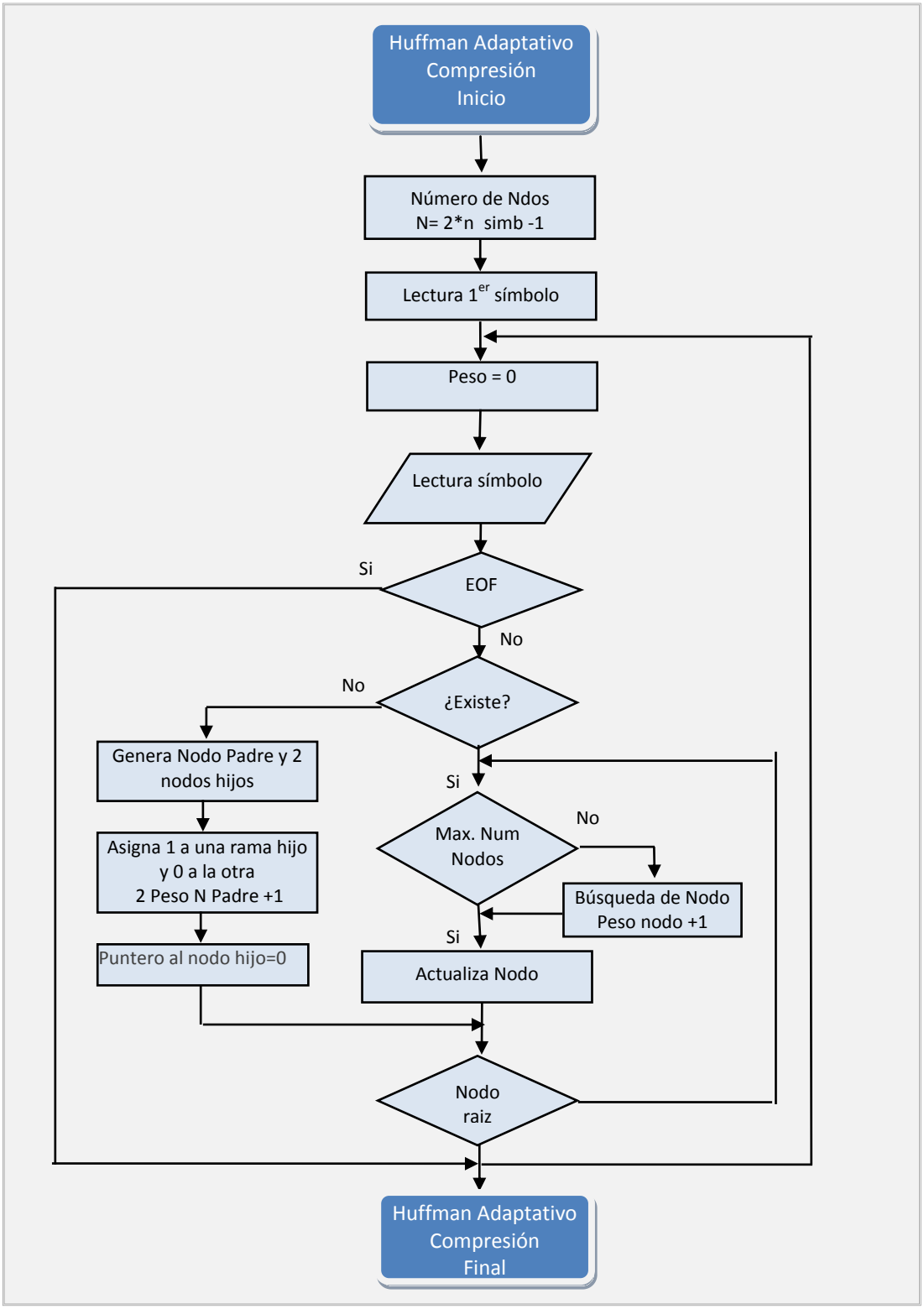


Figura 3.23. Flujograma del algoritmo Huffman adaptativo

3.3.4.4 Ejemplo práctico

Consideremos una sucesión de los siguientes siete valores de píxeles: 25, 25, 21, 23, 24, 25, 24. El alfabeto está formado por cuatro valores diferentes de píxeles. Esto significa que podríamos codificar cada uno de ellos con tres bits de longitud. El algoritmo Huffman adaptativo opera ejecutando los siguientes pasos:

Paso 1) El número total de nodos posibles existentes se obtiene de la expresión 3.22.

$$\text{Num\_Nodos} = 2*\text{Num\_Symbolos}+1=9 \qquad (3.22)$$

El nodo raíz es identificado con el índice "9" y su peso inicial es cero. La codifcación de los símbolos del alfabeto fuente es la siguiente:

25	21	23	24
000	001	010	011

Paso 2) Lectura del primer píxel de la imagen "25". Como no existe en el árbol binario genera dos nuevos nodos identificados como "8" con peso inicial igual a uno y un nodo "7" identificado como o NYT con peso inicial nulo. La suma de los pesos de los dos nodos actualiza el nodo de nivel superior (Fig. 3.24).

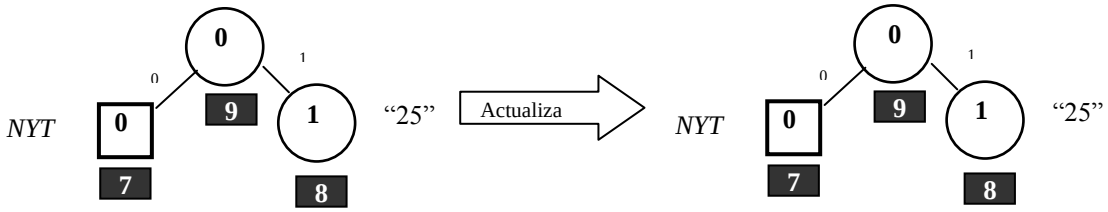


Figura 3.24. Codificación del primer píxel

El primer valor a codificar o transmitir siempre es el valor del primer píxel de la imagen. Por tanto, codifica o transmite el valor "25" mediante una representación binaria de un byte de longitud que corresponde al código "*000*"

Paso 3) Procesa el siguiente píxel de la secuencia "25". El píxel ya existe en el árbol binario, por tanto, incrementa en uno el peso del nodo y actualiza el árbol de forma recursiva desde el nodo actual hasta el nodo raíz (Fig. 3.25).

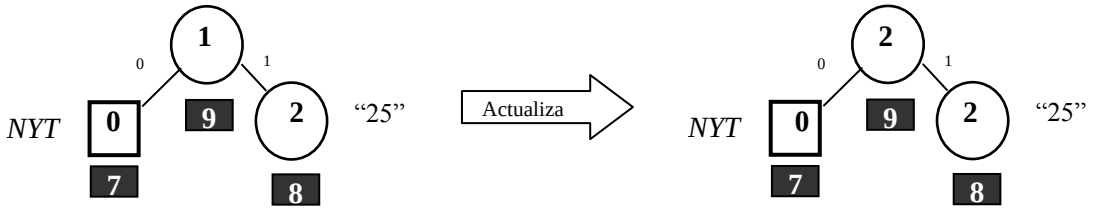
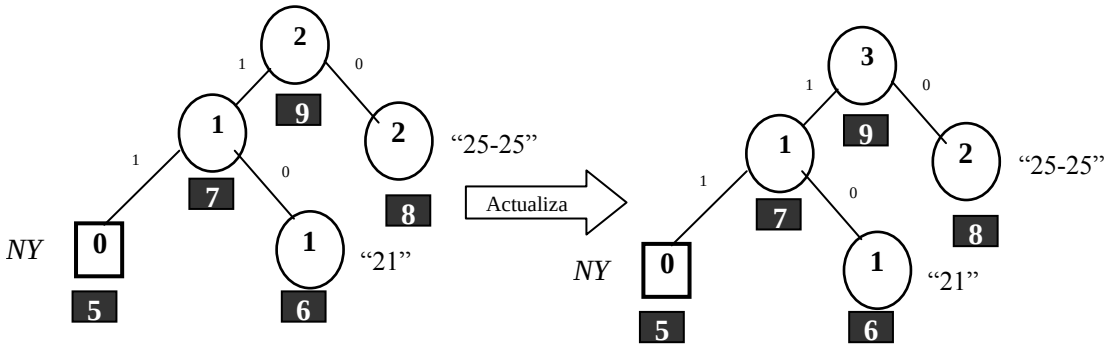


Figura 3.25. Codificación del segundo píxel

El píxel a codificar ya existe en el árbol, entonces se añade a la cadena la codificación del símbolo en el árbol binario de Huffman. La codificación del píxel con valor "25" es codificada con "1" en el árbol binario. A la cadena anterior, le añade la entrada de un nuevo dato o NYT "0" y el código del píxel en el árbol binario "1". El código resultante será "*0001*" .

Paso 4) El siguiente píxel "21" no existe en el árbol, por consiguiente, crea dos nuevos nodos: un nodo identificado como "6" de peso inicial uno y el nodo "7" de peso inicial nulo y actualiza el árbol binario desde este nodo hasta el nodo raíz (Fig. 3.26).

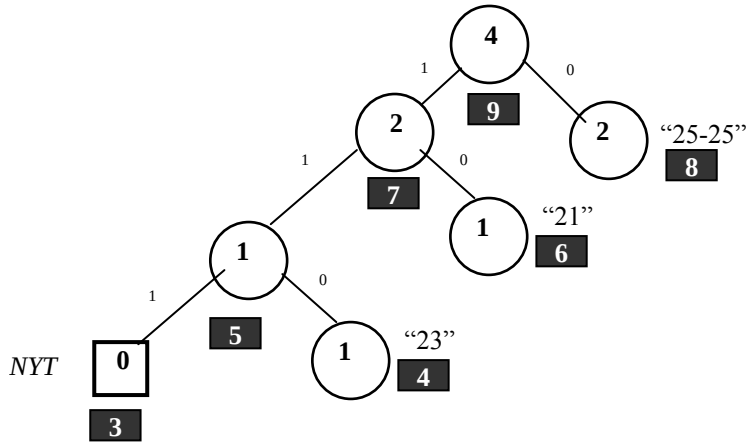


NYT 21

(Código de salida = 0001 0 001)

Figura 3.26. Codificación del tercer píxel

Paso 5) El siguiente píxel de la secuencia de entrada es "23". Este valor de píxel es nuevo, entonces repite el procedimiento anterior y después actualiza el árbol.



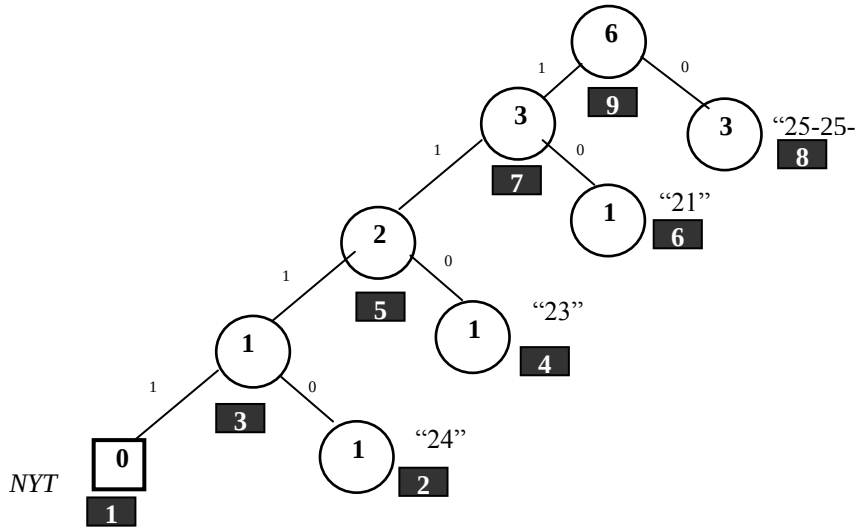
NYT 23

(Código de salida = 00010001 00 010)

Figura 3.27. Codificación del cuarto píxel

Paso 6 y 7) Procesa los siguientes píxeles "24" y "25" respectivamente. El píxel "24" no existe en el árbol, entones crea un nuevo nodo padre con sus respectivos nodos

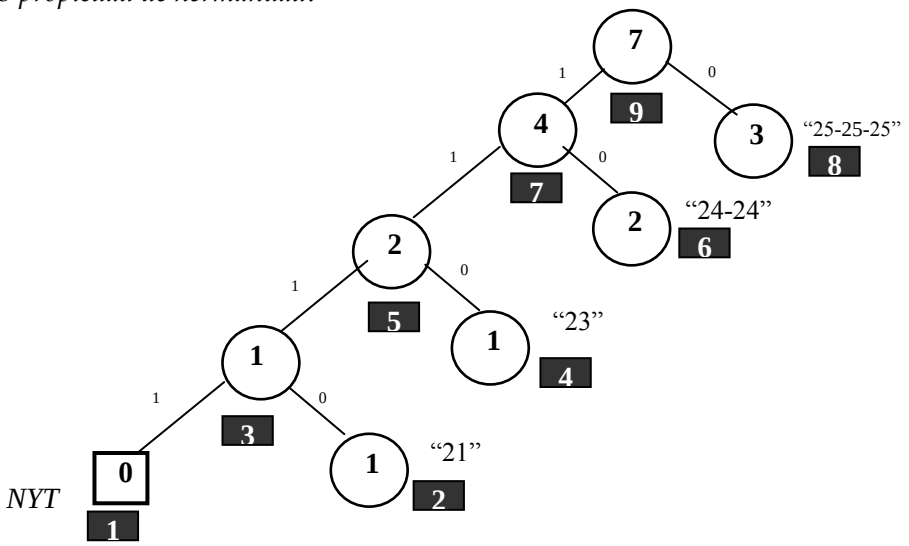
hijos "2" y "1" y de nuevo actualiza el arbol. El píxel con valor "25" ya existe en el árbol. Por tanto, incrementa en uno su peso y actualiza árbol.



(Código de salida = 0001000100010 000 011 0)

Figura 3.28. Codificación del quinto y sexto píxel

Paso 8) En este último paso se ilustra un caso especial en el cual el píxel "24" se repite alterando el orden del peso de los nodos. En este caso, el árbol binario debe regenerarse cumpliendo las propiedades de un árbol *Huffman*, esto significa que debe numerar los nodos de izquierda a derecha y desde el nivel inferior hasta el nodo raíz, en orden creciente de peso. Esta propiedad es denominada con el término *sibling* o *propiedad de hermandad*.



(Código de salida = 000100010001000001l0 00 011)

Figura 3.29. Codificación del último píxel

El proceso de codificación genera la siguiente secuencia de bits de salida:

Código de salida: 000100010001000001100001l = 25 bits

Evidentemente, aunque la codificación supone una importante reducción de bits, es importante destacar que los modelos adaptativos requieren guardar los valores del alfabeto (píxeles diferentes) y sus códigos para recuperar los datos de modo reversible.

A continuación se describe el ciclo de descompresión del modelo adaptativo de Huffman.

3.3.5 Proceso de descompresión

En los ciclos de compresión y descompresión, el transmisor y el compresor no disponen de información previa de los datos. El ciclo de descompresión genera un nodo raíz etiquetado con el símbolo <Nodo=2* *Num_Symbolos* + *1*> y construye un árbol binario con las mismas reglas que las utilizadas en el proceso de compresión (Sayood, 2012). En un primer paso decodifica el primer código incrementando en uno el peso y posteriormente actualiza el árbol. Este proceso continua hasta finalizar la lectura de todos los símbolos del fichero comprimido. En la Fig. 3.30 se ilustra el modelo adaptativo de Huffman.

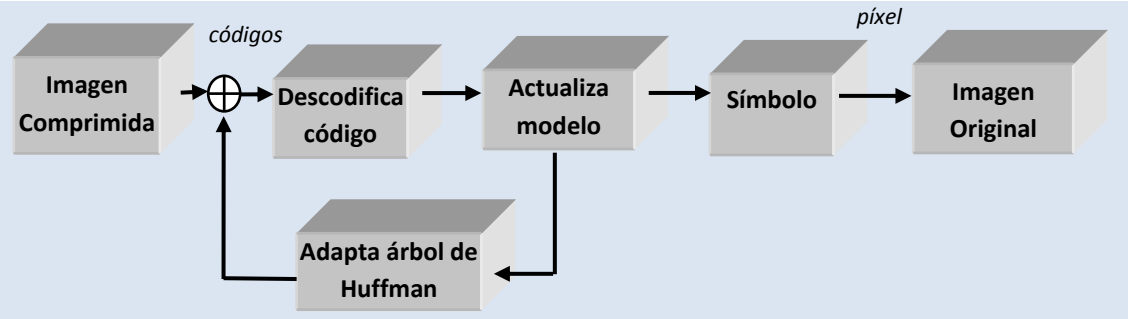


Figura 3.30. Modelo de Huffman adaptativo de descompresión

Para recuperar los datos desde el fichero comprimido el decodificador debe realizar operaciones a nivel de bit atravesando el árbol desde el nodo raíz hasta el nodo del píxel a recuperar. Si consigue valor de símbolo lo escribe en la salida en un byte de longitud y actualiza el árbol.

El árbol construido en el proceso de compresión debe ser exactamente igual al generado en el proceso de descompresión, aunque los datos se procesen en diferente orden.

3.3.5.1 Algoritmo de descompresión de Huffman adaptativo

El algoritmo adaptativo no genera la tabla de probabilidades pero requiere actualizar en cada entrada los nodos del árbol binario. Este proceso ralentiza significativamente los tiempos de ejecución.

Paso 1. Generar el árbol binario con la probabilidad de cada nodo igual a 1
Paso 2. Comenzar desde el nodo raíz
Paso 3. Analizar el siguiente símbolo
Paso 4. ¿Es igual a 1?
Paso 4.1 . SI, Intercambiar posición de hijos y colocarle en rama izquierda
Paso 5. ¿Es igual a 0?
Paso 5.1 .colocar el hijo en rama derecha
Paso 6. ¿Es nodo?
Paso 6.2 . Salida del código binario del símbolo
Paso 6.3 . Incrementa el contador en uno
Paso 6.3 Actualiza el árbol Huffman
Paso 6.4 Vuelve al paso 2
Paso 7. Vuelve al paso 3 hasta finalizar lectura de símbolos

3.3.5.2 Programación del algoritmo adaptativo de descompresión

La programación del algoritmo requiere operaciones recursivas para generar los pesos de cada nodo en cada entrada de datos. El tipo de operaciónes usadas son de tipo puntero y de ciclos.

Las operaciones de ciclos del tifpo *for* o *while* producen la ralentización de la generación de los códigos. En el caso de la telecomunicaciones, los códigos son enviados por las comunicaciones y es el receptor el que se encarga de decodificar los datos sin ralentizar las comunicaiones. Esta es la principal aplicación de los modelos adaptativos en el campo de la compresión de datos vía telecomunicaciones.

```
// Algoritmo Huffman adaptativo
// Bucle de procesamiento de la imagen original
nodo[I]++;
if (node[i] > nodo[i+1])
{
  val = nodo[i+1];
  j = i+1;
  while (nodo[j] == símbolo)
     {
      j++;
     }
  cambia (nodo[i], nodo[j-1]);
}
```

Listado 3.7. Programación del algoritmo de descompresión de Huffman adaptativo

3.3.5.3 Flujograma del algoritmo de descompresión

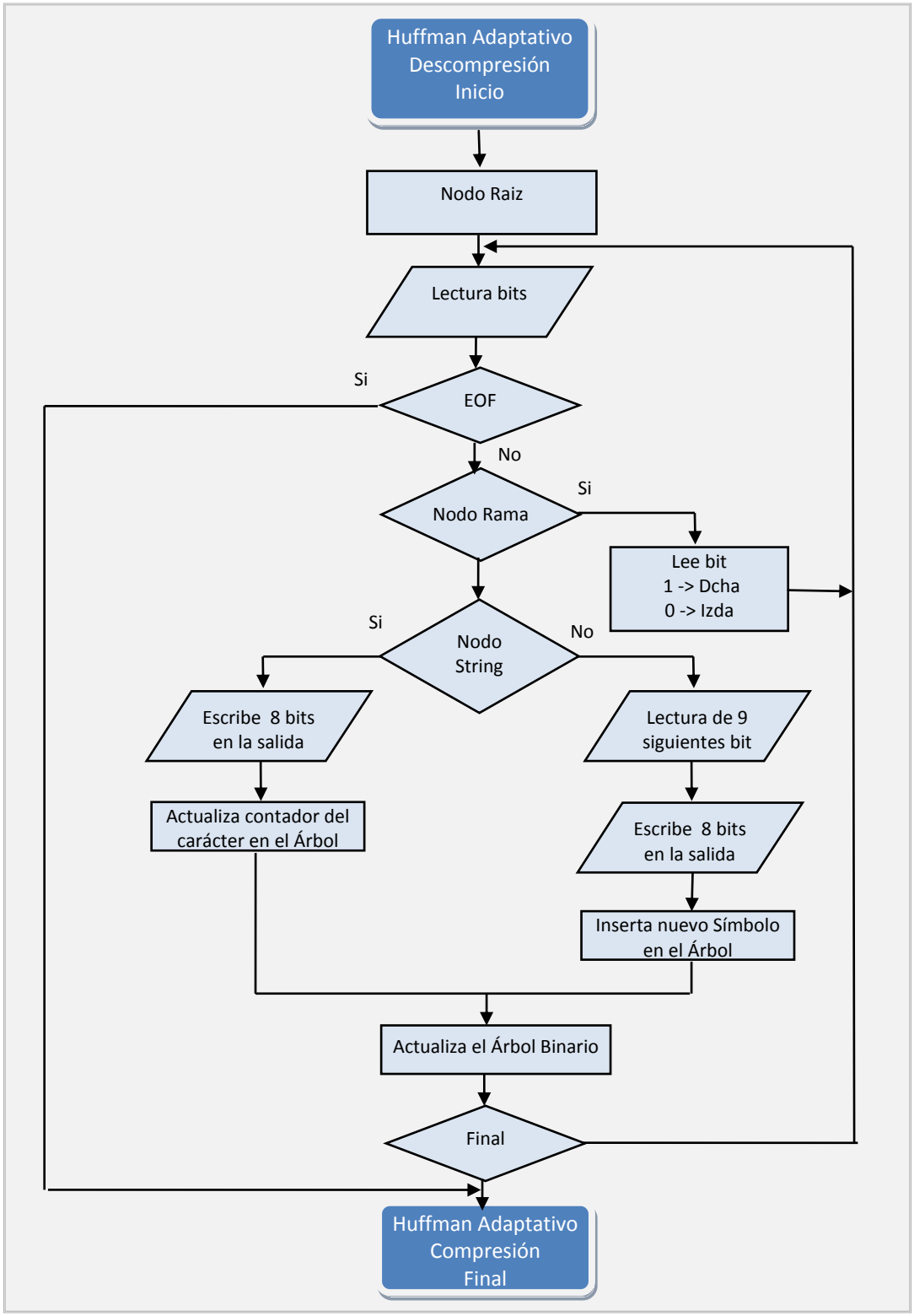


Figura 3.31. Flujograma del proceso de descompresión de Huffman adaptativo

3.3.5.4 Ejemplo del proceso de descompresión

A partir del código de salida del codificador "000100010001000001100001l" el ciclo de descompresión aplica las mismas reglas que en el ciclo de compresión para recuperar los valores originales.

Paso 1) calcula el número de nodos del árbol aplicando la expresión 4.19

$$Num\_Nodos = 2 * Num\_Symbolos + 1 = 9$$

Paso 2) Inicializa el árbol binario con el nodo raíz y sus nodos hijos recuperando el valor del primer píxel.

(Código = 000)

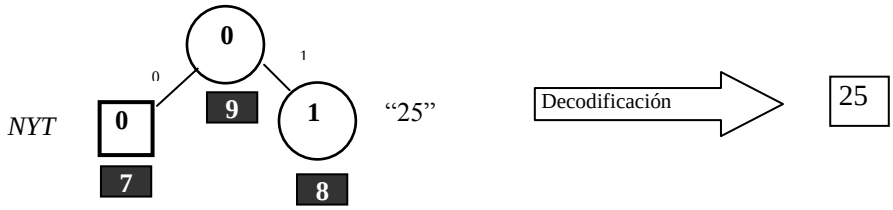


Figura 3.32. Recuperación del primer píxel

Paso 3) El siguiente valor es un código de salida "1". Esto significa que no puede ser un nuevo código sino que es un valor de repetición del píxel anterior.

(Código = 000 1)

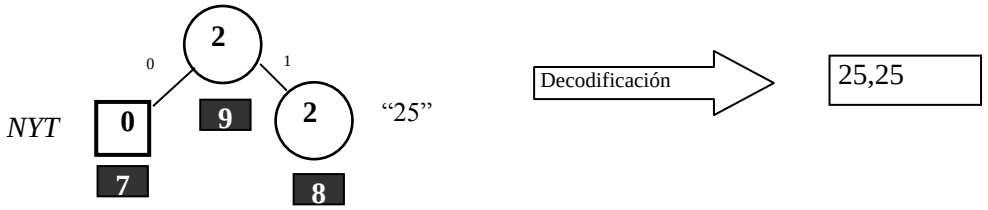


Figura 3.33. Recuperación del segundo píxel

Paso 4) Aplicando las mismas reglas genera el árbol y lo actualiza en cada entrada de datos. En el siguiente grafico se representa el árbol actualizado hasta la última entrada en la cual es necesario realizar la operación *sibling*.

(Código 0001 0 001 00 010 000 011 0)

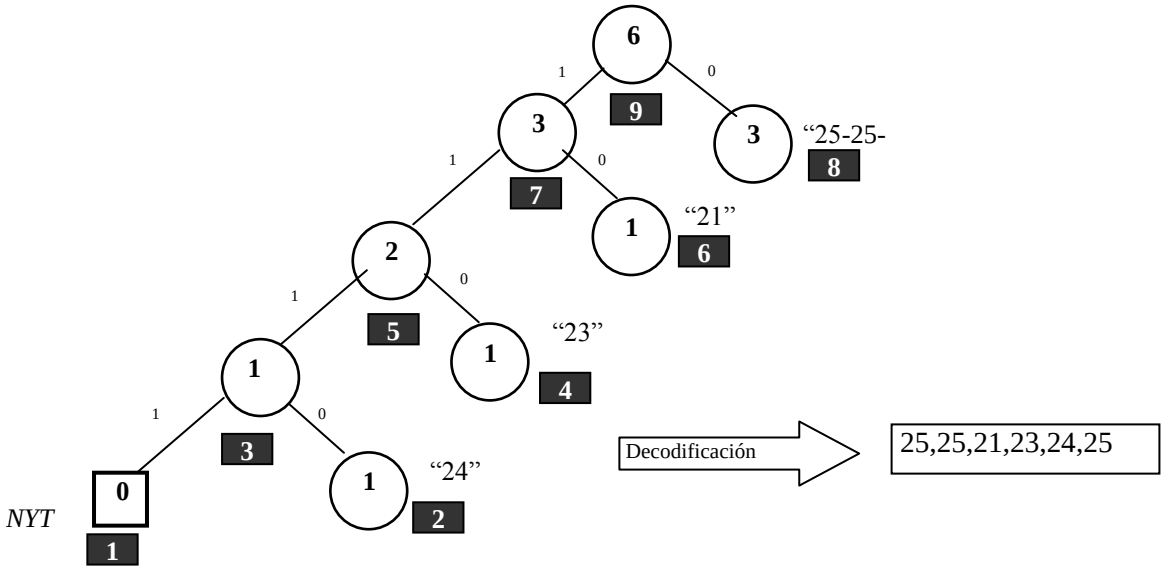


Figura 3.34. Recuperación de los seis primeros píxeles

Paso 5) En un último paso, el código "011" se repite e implica el aumento de peso de su nodo en uno y por tanto la necesidad de regeneración del árbol (*sibling*).

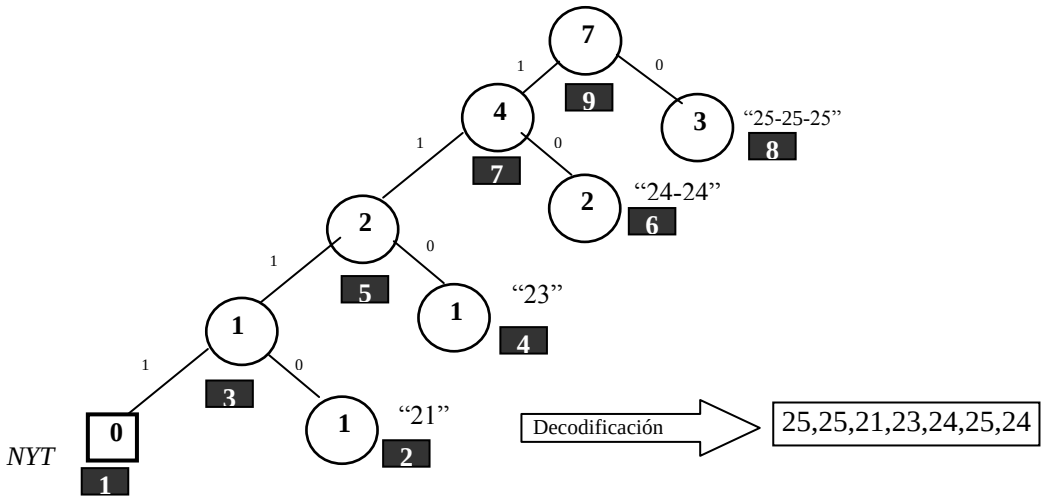


Figura 3.35. Recuperación del último píxel

En definitiva, cada píxel excepto el primero es recuperado de la secuencia de salida a partir del código NYT y del propio código del píxel. El árbol se reconstruye y actualiza aplicando las mismas reglas que en el codificador.

3.3.6 Resultados experimentales

En la Tabla 3.24 se muestran los resultados de la evaluación del Codificador Huffman estático y adaptativo.

Tabla 3-24. Resultados experimentales del Codificador Huffman

IMAGEN FUENTE	NOMBRE IMAGEN	HUFFMAN ESTATICO	HUFFMAN ADAPTATIVO
		RATIO COMPRESIÓN	RATIO COMPRESIÓN
Fotográficaas	Airplane	1,192	1,188
	Baboon	1,083	1,081
	Barbara	1,043	1,041
	Boats	1,105	1,105
	Cameraman	1,115	1,124
	Goldhill	1,067	1,064
	Lena	1,071	1,068
	Man	1,101	1,096
	Peppers	1,048	1,051
	Zelda	1,095	1,000
	Media	**1,092**	**1,082**
Aéreas y Satélite	Aerial	1,138	1,136
	Airfield	1,120	1,117
	Earth	1,201	1,169
	Meteosat	1,079	1,085
	Moon	1,223	1,223
	Moonsurface	1,181	1,188
	SanDiego	1,069	1,069
	WashingtonIR	1,062	1,062
	Media	**1,134**	**1,131**
Creadas por Ordenador	Circles	4,218	3,831
	Gray	1,775	1,758
	Slope	1,055	1,057
	Squares	5,985	5,258
	Media	**3,258**	**2,976**
Médicas	Elbowx	1,300	1,346
	Finger	1,115	1,107
	MRI_Brain1	1,253	1,251
	MRI_Brain2	1,572	1,575
	MRI_Brain3	1,148	1,000
	Shoulder	1,219	1,222
	Media	**1,268**	**1,250**
Textos y Gráficos escaneados	Mercados1	1,674	1,476
	Mercados2	1,557	1,562
	Mercados3	1,595	1,589
	Mercados4	1,804	1,794
	Media	**1,657**	**1,605**

La Tabla 3.24 refleja que el modelo adaptativo consigue mejores ratios de compresión que el modelo convencional en la mayoría de las imágenes procesadas, aunque las diferencias son poco significativas.

Los resultados obtenidos muestran que siempre existe compresión para todos los tipos de imágenes (Fig. 3.36). Los ratios de compresión oscilan en un rango de [1,0 – 1,657] excepto en las imágenes generadas por ordenador cuyo ratio superior es alcanzado en la imagen "squares" con un valor de 5,985. En este tipo de imágenes es importante destacar que RLE alacanzaba ratios medios de 76,487 y en la imagen "squares" obtiene el mayor ratio de compresión 105,026, que supera ampliamente al Codificador Huffman

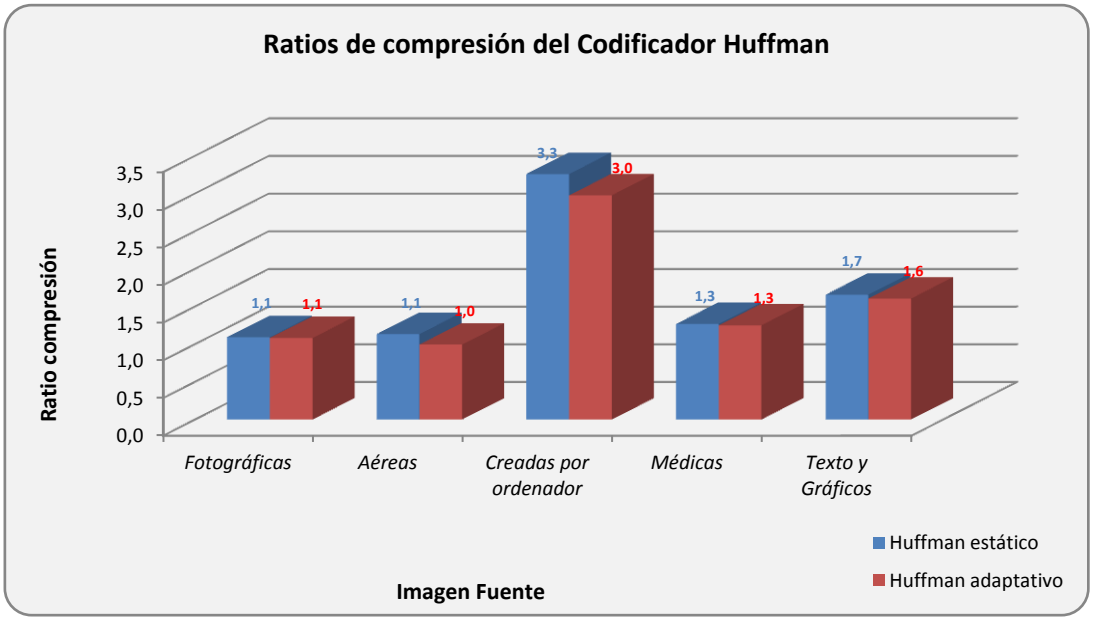


Figura 3.36. Resultados comparativos de los ratios de compresión Huffman

La Fig. 3.36 muestra que el modelo estático siempre alacanza mejores resultados de compresión que el modelo adaptativo en términos de almaceneamiento. Por el contrario, como los ratios de compresión son muy similares, el modelo adaptativo es más óptimo en aplicaciones de transmisión de datos en las cuales se realiza la compresión y descompresión de datos en una sola etapa.

En las Tablas 3.25 y 3.26 se muestran los resultados de tiempos y ciclos de CPU de ambos modelos. Como característica a resaltar, cabe destacar que los tiempos de descompresión para ambos métodos suelen ser mayores que los de compresión (Nelson y Gailly, 1995). Esta característica es particular del método Huffman ya que tiene construir de nuevo el árbol binario. A continuación se adjuntan las tablas de resultado para ambos modelos.

Tabla 3-25. Huffman estático. Tiempos de ejecución del ciclo de compresión y descompresión

IMAGEN FUENTE	NOMBRE IMAGEN	CICLO DE COMPRESIÓN		CICLO DE DESCOMPRESIÓN	
		CICLOS CPU	TIEMPO (ms)	CICLOS CPU	TIEMPO (ms)
Fotográficaas	Airplane	2.670.182	1.209,09	2.530.500	1.145,84
	Baboon	2.845.217	1.288,35	2.683.290	1.215,03
	Barbara	2.523.447	1.142,65	2.933.855	1.328,49
	Boats	2.719.946	1.231,63	2.764.841	1.251,96
	Cameraman	1.005.467	455,29	844.447	382,38
	Goldhill	2.895.274	1.311,02	2.867.483	1.298,43
	Lena	2.515.270	1.138,95	2.878.440	1.303,40
	Man	917.871	415,62	944.317	427,60
	Peppers	2.746.503	1.243,65	2.954.538	1.337,85
	Zelda	3.039.828	1.376,47	2.568.285	1.162,95
	Media	**2.387.901**	**1.081,27**	**2.397.000**	**1.085,39**
Aéreas y Satélite	Aerial	3.018.192	1.366,68	2.733.111	1.237,59
	Airfield	2.828.698	1.280,87	2.768.852	1.253,77
	Earth	574.294	260,05	678.994	307,46
	Meteosat	5.750.554	2.603,93	6.273.483	2.840,71
	Moon	23.849.818	10.799,51	30.921.520	14.001,67
	Moonsurface	1.649.364	746,85	1.084.855	491,24
	SanDiego	8.271.151	3.745,28	9.746.876	4.413,51
	WashingtonIR	41.745.876	18.903,08	44.853.623	20.310,31
	Media	**10.960.993**	**4.963,28**	**12.382.664**	**5.607,03**
Creadas por Ordenador	Circles	615.940	278,91	367.553	166,43
	Gray	2.540.751	1.150,49	1.668.483	755,51
	Slope	865.322	391,83	1.042.823	472,20
	Squares	778.240	352,39	318.047	144,01
	Media	**1.200.063**	**543,41**	**849.227**	**384,54**
Médicas	Elbowx	2.802.502	1.269,01	2.280.672	1.032,72
	Finger	885.009	400,74	907.198	410,79
	MRI_Brain1	867.337	392,74	850.569	385,15
	MRI_Brain2	617.466	279,60	461.225	208,85
	MRI_Brain3	1.215.601	550,44	1.252.397	567,10
	Shoulder	1.443.647	653,70	1.824.977	826,37
	Media	**1.305.260**	**591,04**	**1.262.840**	**571,83**
Textos y Gráficos escaneados	Mercados1	7.570.484	3.428,01	7.195.342	3.258,14
	Mercados2	7.571.939	3.428,67	6.729.527	3.047,22
	Mercados3	5.736.201	2.597,43	5.124.701	2.320,53
	Mercados4	3.335.592	1.510,40	2.662.387	1.205,56
	Media	**6.053.554**	**2.741,13**	**5.427.989**	**2.457,86**

En términos de consumo de memoria, el Codificador Huffman requiere almacenar la tabla de probabilidad y el árbol binario. Los recursos de memoria son significativos y claramente muy superiores a los requerimiento de memoria de los métodos revisados. La ejecución del método se realiza en memoria principal.

Tabla 3-26. Huffman Canonical. Tiempos de ejecución del ciclo de compresión y descompresión

IMAGEN FUENTE	NOMBRE IMAGEN	CICLO DE COMPRESIÓN		CICLO DE DESCOMPRESIÓN	
		CICLOS CPU	TIEMPO (ms)	CICLOS CPU	TIEMPO (ms)
Fotográficaas	Airplane	2.290.688	1.037,25	24.574.013	11.127,43
	Baboon	2.546.699	1.153,18	37.116.679	16.806,92
	Barbara	1.965.733	890,11	44.405.567	20.107,42
	Boats	2.366.129	1.071,41	31.808.863	14.403,47
	Cameraman	778.346	352,45	7.312.447	3.311,17
	Goldhill	2.478.647	1.122,36	39.127.491	17.717,44
	Lena	2.353.998	1.065,92	39.433.006	17.855,78
	Man	738.038	334,19	8.563.864	3.877,83
	Peppers	2.310.727	1.046,33	40.894.403	18.517,52
	Zelda	2.555.514	1.157,17	37.423.540	16.945,87
	Media	**2.038.452**	**923,04**	**31.065.987**	**14.067,09**
Aéreas y Satélite	Aerial	2.523.120	1.142,50	28.161.227	12.751,77
	Airfield	2.504.143	1.133,91	31.845.709	14.420,15
	Earth	529.597	239,81	4.160.861	1.884,09
	Meteosat	5.226.002	2.366,40	83.418.017	37.772,77
	Moon	21.975.370	9.950,73	471.455.754	213.481,31
	Moonsurface	1.460.526	661,35	5.486.087	2.484,17
	SanDiego	7.512.343	3.401,69	151.106.538	68.423,01
	WashingtonIR	40.039.726	18.130,51	759.427.441	343.878,64
	Media	**10.221.353**	**4.628,36**	**191.882.704**	**86.886,99**
Creadas por Ordenador	Circles	490.875	222,27	1.579.795	715,35
	Gray	2.244.050	1.016,14	10.117.165	4.581,18
	Slope	593.847	268,90	11.747.323	5.319,34
	Squares	559.980	253,56	1.748.544	791,75
	Media	**972.188**	**440,22**	**6.298.207**	**2.851,91**
Médicas	Elbowx	2.350.469	1.064,32	21.227.030	9.611,88
	Finger	856.467	387,82	7.973.469	3.610,49
	MRI_Brain1	754.438	341,62	7.230.939	3.274,26
	MRI_Brain2	578.222	261,83	1.690.893	765,66
	MRI_Brain3	1.022.172	462,85	7.515.347	3.403,05
	Shoulder	1.187.192	537,58	20.807.241	9.421,79
	Media	**1.124.827**	**509,34**	**11.074.153**	**5.014,52**
Textos y Gráficos escaneados	Mercados1	6.912.361	3.130,01	69.923.252	31.662,16
	Mercados2	6.779.780	3.069,97	63.791.986	28.885,84
	Mercados3	5.126.231	2.321,22	48.491.699	21.957,67
	Mercados4	2.817.360	1.275,74	23.128.949	10.473,09
	Media	**3.605.955**	**1.632,82**	**34.222.648**	**15.496,46**

Los resultados expuestos en las Tablas 3.25 y 3.26 muestran que los tiempos de descompresión son muy elevados, especialmente para el modelo adaptativo. Estos tiempos de ejecución no son operativos teniendo en cuenta que la diferencia en

ratios de compresión son insignificativas. Únicamente, en la transmisión telemática de datos se podría justificar su uso.

3.3.7 Conclusiones del codificador Huffman

El Codificador Huffman obtiene códigos cerca de la entropía para cada píxel y este método es imposible de superar cuando las probabilidades son potencias de dos. Sin embargo, presenta tres inconvenientes relacionados con el tamaño del árbol binario, los tiempos de ejecución y la imposibilidad de aplicación en imágenes monocromáticas.

El tamaño del árbol debe ser verificado en cada actualización para detectar un posible desbordamiento (*overflow*). El desbordamiento se produce cuando el nodo raíz excede del límite de capacidad del contador en el árbol. En este caso, es necesario regenerar el árbol asignando a los nodos un nuevo peso que se obtienen dividiendo el peso de cada nodo por un factor fijo. Este proceso puede provocar perdida de la característica de hermandad al truncar los pesos de los nodos. Para mantener la propiedad de hermandad es preciso reescalar todos los valores de los nodos. Los tiempos de ejecución se incrementan considerablemente en la construcción o actualización del árbol binario. Esto implica numerosas operaciones con bucles y recursivas que ralentizan los tiempos de ejecución en ambos ciclos.

En imágenes monocromáticas, no es posible aplicar el codificador en ninguno de sus modelos porque no es posible producir compresión sobre un alfabeto binario. El algoritmo de Huffman aplicado sobre un alfabeto de dos símbolos asignará un código uno o un cero, independientemente de la probabilidad de dichos píxeles. En este contexto, no es posible realizar la compresión de los datos.

El Codificador Huffman asigna un número entero de *bits* de longitud variable para cada símbolo de entrada. Aplicando la Teoría de la Información se obtiene que a un símbolo con probabilidad de 0,4, le debería ser asignado un código de longitud igual a 1,32 bits ($\log_2 0{,}4 = 1{,}32$). Sin embargo, el algoritmo asigna códigos enteros en el proceso de codificación y esto significa que este valor únicamente puede ser codificado con uno o dos bits. Este problema es resuelto por el Codificador Aritmético que permite generar un código de salida que puede ser representado por un número fraccional de *bits* en lugar de un número entero. En la siguiente sección se revisa el Codificador Aritmético.

3.4 El Codificador Aritmético (Arithmetic Coding)

Arithmetic Coding (Rissanen , 1976) es un método de compresión de datos *lossless* que utiliza un modelo estadístico para procesar los datos de entrada en el codificador. El primer caso práctico de aplicación del algoritmo del cual se tiene constancia es el

realizado por Pasco (Pasco, 1976.) y Rissanen. Sin embargo, la idea original es atribuida a Peter Elias, en 1960.

Arithmetic Coding es un método recomendado por Joint Bi-Level Image Processing Group-JBIG para su uso en el estándar de codificación de imágenes binarias. Este método es utilizado también en el estándar JPEG. Numerosas versiones de este método están sujetas a protección por patentes, especialmente es conocida la patente propiedad de IBM.

Arithmetic Coding y *Huffman Coding* son métodos de compresión basados en modelos aritméticos. La probabilidad del símbolo de entrada es utilizada para generar un código de salida. La diferencia entre ambos métodos reside en el camino utilizado en el proceso de compresión y en la asignación del valor de la probabilidad. En *Arithmetic Coding* la probabilidad de los símbolos de entrada es representada siempre con valores reales comprendidos entre cero y uno y el código de salida resultante para toda la entrada de datos corresponde a un número real comprendido entre los valores cero y uno (Jyun-Ying H., Yin-Chen L y Yuh-Ming H., 2013). Este método obtiene mejores ratios de compresión que *Huffman Coding* cuando la probabilidad de los símbolo no es múltiplo de dos.

3.4.1 Descripción del método

La idea básica del Codificador Aritmético consiste en representar todos los símbolos de entrada por un número real menor que uno en la salida del codificador (Fig. 3.37). Este número real está comprendido en un intervalo entre los valores cero y uno (Sayood, 2012).

El principio de funcionamiento de este método se basa en dos conceptos fundamentales para cada entrada de datos en el modelo: la probabilidad del símbolo/píxel de entrada y el intervalo en cual se codifica este símbolo.

La probabilidad del píxel de entrada determina el número y el rango de los subintervalos. La notación $[x,y)$ denota un intervalo de números reales cuyo rango comprende desde x a y, dónde x está incluido en el intervalo e y está excluido. Cada intervalo corresponde a un píxel de entrada y el tamaño es proporcional a la probabilidad del símbolo (Witten, Neal y Cleary,1987). Los intervalos necesarios para representar los datos son menores a medida que se incrementan el número de datos de entrada, y en consecuencia, el número de bits necesarios para almacenarlos aumenta. El límite inferior del último intervalo generado corresponde al valor de salida del codificador y este representa la salida del codificador.

El diagrama de bloques de la estructura del proceso de compresión se muestra en la Fig. 3.37.

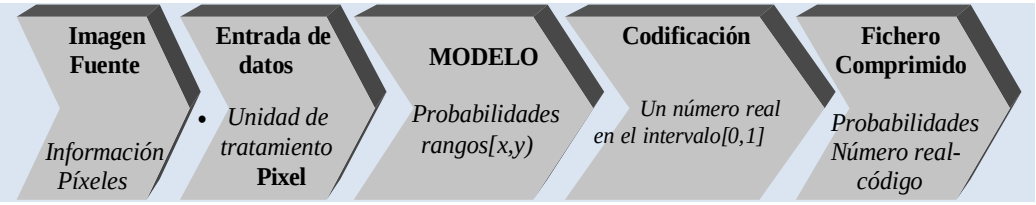


Figura 3.37. Estructura del ciclo de compresión de Arithmetic Coding

Arithmetic Coding separa claramente el modelo del proceso de codificación. Esto significa que pueden utilizarse diferentes modelos manteniendo la misma técnica de codificación (Langdon, 1984). Esta característica garantiza que en los modelos adaptativos, la ejecución de los algoritmos de compresión y descompresión reduzcan los tiempos de ejecución.

La salida del modelo genera un número real entre cero y uno para toda la entrada de datos (Huang y Liang, 2011). Esto es, en el proceso de compresión se obtiene un número fraccional para representar la secuencia completa de símbolos o píxeles de entrada. Por el contrario, en el proceso de descompresión se recuperan los símbolos de entrada a partir del número fraccionario.

3.4.2 Proceso de compresión

El proceso de compresión procesa los píxeles fila a fila comenzando por la izquierda y de arriba hacia abajo hasta completar la imagen. Para optimizar la ejecución la imagen es segmentada en bloques de píxeles.

Inicialmente calcula las probabilidades de cada píxel y los registra directamente en el fichero comprimido. Atendiendo a las probabilidades de cada símbolo, escala todos los símbolos de entrada en un intervalo completo [0,1). Esta representación denota que el intervalo está mitad-abierto, $0 \leq x < 1$.

El píxel es la unidad de tratamiento de entrada. En un modelo fijo a cada píxel se le asigna la probabilidad y obtiene los rangos escalados con respecto a sus probabilidades entre cero y uno. El valor inferior *lim_inf* y superior *lim_sup* de cada subintervalo generado para símbolo de entrada es calculado mediante las siguientes expresiones:

$$\begin{aligned} intervalo_{compresión} &= [lim_{inf}, lim_{sup}] \\ lim_{inf} &= lim_{\text{inf}\_ant} + (rango_{ant} * rango_{\text{inf}\_valor\%}) \\ lim_{sup} &= lim_{\text{inf}\_ant} + (rango_{ant} * rango_{\text{sup}\_valor\%}) \end{aligned} \tag{3.23}$$

Supongamos el siguiente alfabeto {r,a,n,g,o,s} cuyas probabilidades son .0,2; 0,3; 0,1; 0,2; 0,1; y 0,1. El codificador recibe el siguiente mensaje "arnns". Entonces el codificador genera los subintervalos de escalado de acuerdo a sus probabilidades. Aplicando la expresión 3.23, se obtiene los intervalos de la Fig. 3.38:

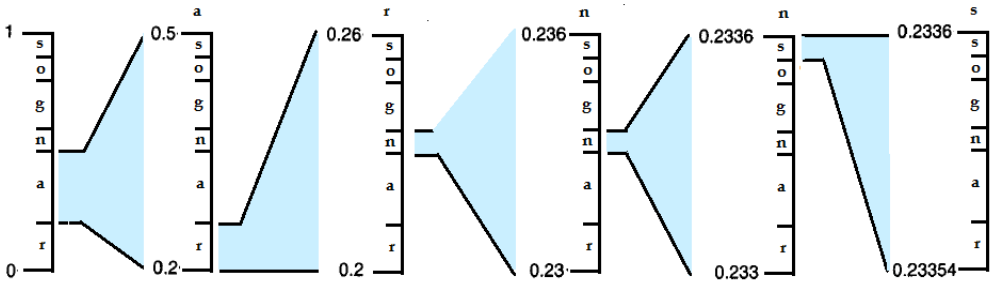


Figura 3.38. Representación del escalado de intervalos

El tamaño de cada subintervalo es proporcional a las probabilidades de los símbolos de entrada. Por cada píxel procesado, el subintervalo generado es más pequeño que el anterior y requiere más bits de codificación. La importancia de este método reside en que no necesita almacenar los códigos en el fichero comprimido. El resultado es un número real que corresponde al valor inferior del rango del último subintervalo.

La salida del codificador corresponde a un número en coma flotante. En la práctica, el codificador utiliza valores enteros fijando de esta forma el número de bits del rango inferior y superior del intervalo.

3.4.2.1 Algoritmo de compresión

Paso 1. Obtener las probabilidades de los símbolos de entrada (píxeles)
Paso 2. Escalar los valores probabilísticos en el intervalo [0, 1]
Paso 3. Escribir en el fichero de salida las probabilidades
Paso 4. lim_inf = 0 (16bits)
Paso 5. lim_sup = 1 (16bits)
Paso 6. rango = 1 (16bits)
Paso 7. Bucle para cada símbolo de entradas.
Paso 7.1 rango = lim_sup – lim_inf
*Paso 7.2 lim_sup = lim_inf + rango * lim_sup(símbolo)*
*Paso 7.3 lim_inf = lim_inf+ rango * lim_inf(símbolo)*
Paso 7.4 Retornar al Paso 7.
Paso 8. Final de entrada
Paso 9. Escribe el número o código en el fichero de salida

3.4.2.2 Programación del algoritmo del Codificador Aritmético

```
// Arithmethic Coding – algoritmo de compresión simplificado
im_inf = 0x0000 ;
lim_sup = 0xFFFF ;
 rango = 1.0 ;
Do while (not. EOF!)   {
    lim_sup = lim_inf + rango * lim_sup( simbolo ) ;
    lim_inf = lim_inf + rango * lim_inf( simbolo ) ;
    rango =lim_sup – lim_inf ;    }
}
codigo_salida= lim_inf;
```

Listado 3.8. Programación del algoritmo de compresión del Codificador Aritmético

3.4.2.3 Flujograma del algoritmo de compresión

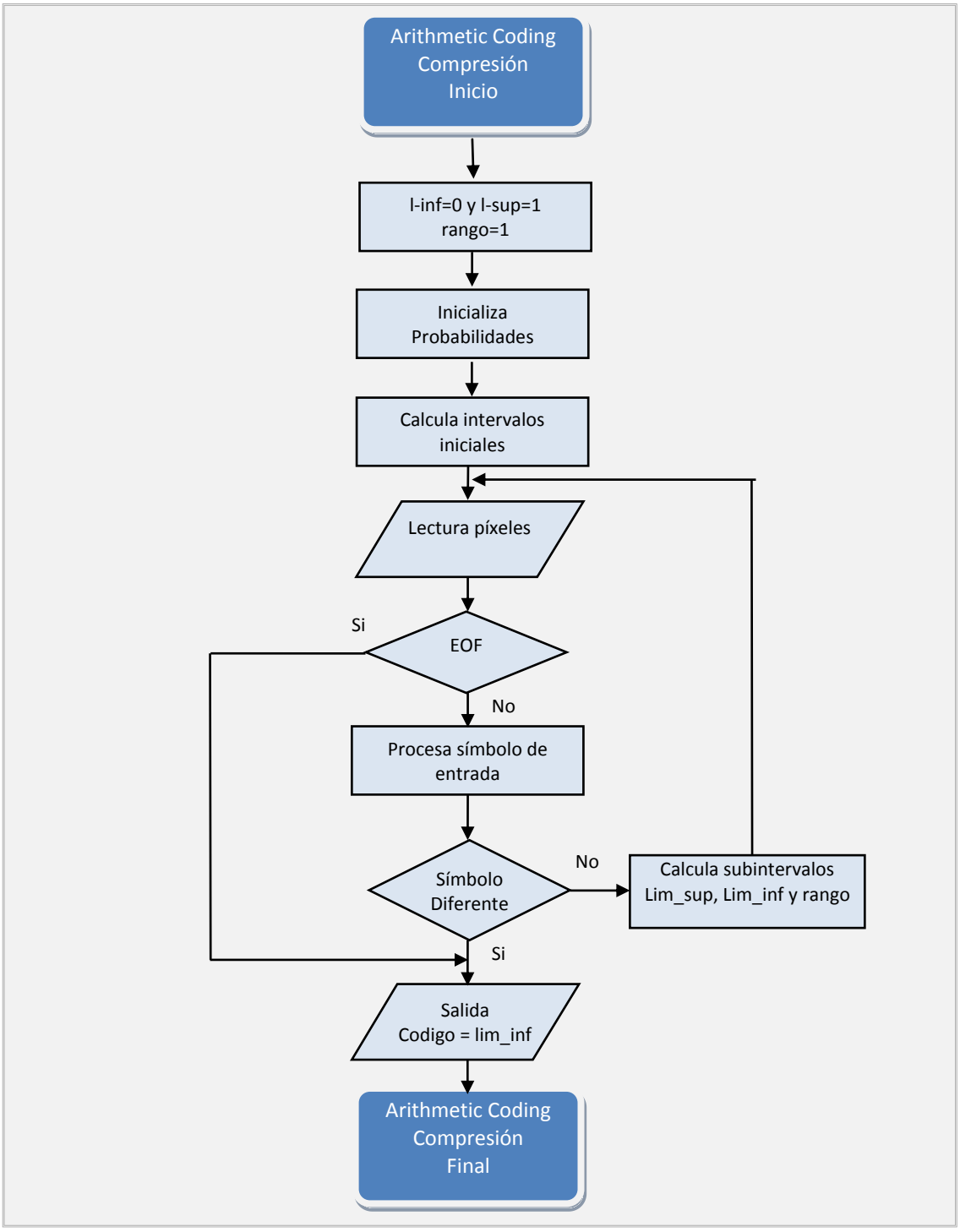


Figura 3.39. Flujograma del algoritmo de compresión de Arithmetic Coding

3.4.2.4 Ejemplo práctico

Dada una cadena de entrada cualquiera con los valores decimales siguientes: "100,65,6,22,11,201,73,5,65,7" se desea comprimir los datos mediante el Codificador Aritmético. Para ilustrar el funcionamiento del algoritmo a los valores se les asocia una cadena alfabética "COMPRESION".

VALOR ASCII DEL TEXTO (PÍXEL)	100	65	6	22	11	201	73	5	65	7
Texto	C	O	M	P	R	E	S	I	O	N

Paso 1. Calcula las probabilidades individuales de cada símbolo de entrada.

P(C)	P(O)	P(M)	P(P)	P(R)	P(E)	P(S)	P(I)	P(N)
1/10	2/10	1/10	1/10	1/10	1/10	1/10	1/10	1/10

Paso 2. Obtiene los rangos de probabilidades entre cero y uno.

VALOR PÍXEL	CADENA ORDENADA	PROBABILIDAD DEL PÍXEL	RANGO INFERIOR/SUPERIOR
100	C	1/10	$0{,}00 \geq r < 0{,}10$
201	E	1/10	$0{,}10 \geq r < 0{,}20$
5	I	1/10	$0{,}20 \geq r < 0{,}30$
6	M	1/10	$0{,}30 \geq r < 0{,}40$
7	N	1/10	$0{,}40 \geq r < 0{,}50$
65	O	2/10	$0{,}50 \geq r < 0{,}70$
22	P	1/10	$0{,}70 \geq r < 0{,}80$
11	R	1/10	$0{,}80 \geq r < 0{,}90$
73	S	1/10	$0{,}90 \geq r < 0{,}10$

Paso 3. Los datos de entrada comienzan con el símbolo "C", entonces el compresor asigna como intervalo de entrada el correspondiente a este código [0,00, 0,10]. Los siguientes intervalos se calcularán mediante la expresión formulada anteriormente (3.23).

$$intervalo_{compresión} = [lim_{inf}, lim_{sup}]$$
$$lim_{inf} = lim_{\text{inf}\_ant} + (rango_{ant} * rango_{\text{inf}\_valor\%})$$
$$lim_{sup} = lim_{\text{inf}\_ant} + (rango_{ant} * rango_{\text{sup}\_valor\%})$$

El resultado de los intervalos generados por el codificador se muestran en la tabla (3.27):

Tabla 3-27. Generación de subintervalos

PÍXEL	SÍMBOLO	INTERVALO	DESCRIPCIÓN
100	C	[0,00 , 0,10]	Lee el primer dato comprendido en el intervalo [0,00 , 0,10]
65	O	[0,05 , 0,07}	Límite inferior = 0,00 + (0,1 * 50%) = 0,05 Límite superior = 0,00 + (0,1 * 70%) = 0,07
6	M	[0,056 , 0,058}	Límite inferior = 0,05 + (0,2 * 30%) = 0,056 Límite superior = 0,05 + (0,2 * 40%) = 0,058
22	**P**	[0,05756 , 0,05758}	...
11	**R**	[0,0574 , 0,0576}	...
201	**E**	[0,057562 , 0,057564}	...
73	**S**	[0,0575638 , 0,0575640}	...
5	**I**	[0,05756384 , 0,05756386}	...
65	**O**	[0,057563850 0,0557563854}	...
7	**N**	[0,0575638516 0,05756638520]	...

La salida del codificador corresponde al límite inferior del último intervalo procesado, en el ejemplo propuesto <0,0575638516>. El valor codificado en binario corresponde a la siguiente secuencia de bits almacenada en el fichero comprimido: **00001110101111001000000010010.**

3.4.3 Proceso de descompresión

El proceso de descompresión obtiene en un primer paso la tabla de símbolos y probabilidades. En un segundo paso, a cada símbolo de esta tabla se le asigna un subintervalo comprendido en el rango [0, 1) en función de su probabilidad. Tercero, lee del fichero comprimido el código binario que corresponde al rango de inferior del último subintervalo calculado en el proceso de compresión. Este valor es significativo por su pertenencia al intervalo de probabilidades de un símbolo. Este corresponde al primer símbolo de entrada codificado.

$$\text{lim\_inf}(\text{símbolo}) \leq \text{código} < \text{lim\_sup}\ (\textit{símbolo}) \qquad (3.24)$$

A partir de este valor aplica el mismo procedimiento que en el proceso de codificación para obtener los subintervalos y e esta forma reconstruir los datos de entrada.

$$\sum_{i=1}^{n-1} P(i) \leq \frac{valor(i-1) - lim\_inf\,(i-1)}{lim\ \_sup(i-1) - lim\_inf\,(i-1)} < \sum_{i=1}^{n} P(i) \qquad (3.25)$$

donde, n representa el número de símbolos de entrada.

3.4.3.1 Algoritmo de descompresión

El algoritmo de descompresión comienza con la lectura del valor codificado que corresponde al valor inicial del último subintervalo codificado. A partir de este valor el codificador genera los subintervalos aplicando el mismo procedimiento que en el proceso de compresión. Los pasos del algoritmo de descompresión son los siguientes

Paso 1. Leer las probabilidades de los símbolos en el fichero comprimido
Paso 2. Reconstruir los rangos iniciales a partir de las probabilidades
Paso 3. Lee el código o valor del codificador y obtiene primer símbolo
Paso 4. Calcula el valor o resultado para construir los subintervalos.
Paso 4.1 Identifica cada símbolo por su rango
rango = (lim_sup –lim_inf) del simbolo anterior
Valor= (código – lim_inf) / rango
Paso 5. Retorna al paso 4 hasta marca de final de fichero comprimido
Paso 6. Escribe fichero descomprimido

3.4.3.2 Programación del algoritmo de descompresión

```
1  // Arithmethic Coding
2  // Bucle de procesamiento de la imagen codificada
3  // Primer valor
4  simbolo = '';
5  código=valor;
6  for( i=0; i<n;i++) {
7     if (valor>prob(i))
8        lim_inf= inf(i);
9        lim_sup=sup (i);      }
10    // ciclo asta final de fichero
11    do {
12       rango= lim_sup - lim_inf;
13       valor=(valor - lim_inf) / rango;
14       lim_inf  =inf(i);
15       lim_sup = sup(i);
16       for( i=0; i<n;i++) {
17          if (valor>prob(i))
18              lim_inf= inf(i);
19              lim_sup=sup (i);  }
20 // Escribe fichero expandido
```

Listado 3.9. Programación del algoritmo de descompresión del Codificador Aritmético

3.4.3.3 Flujograma del proceso de descompresión

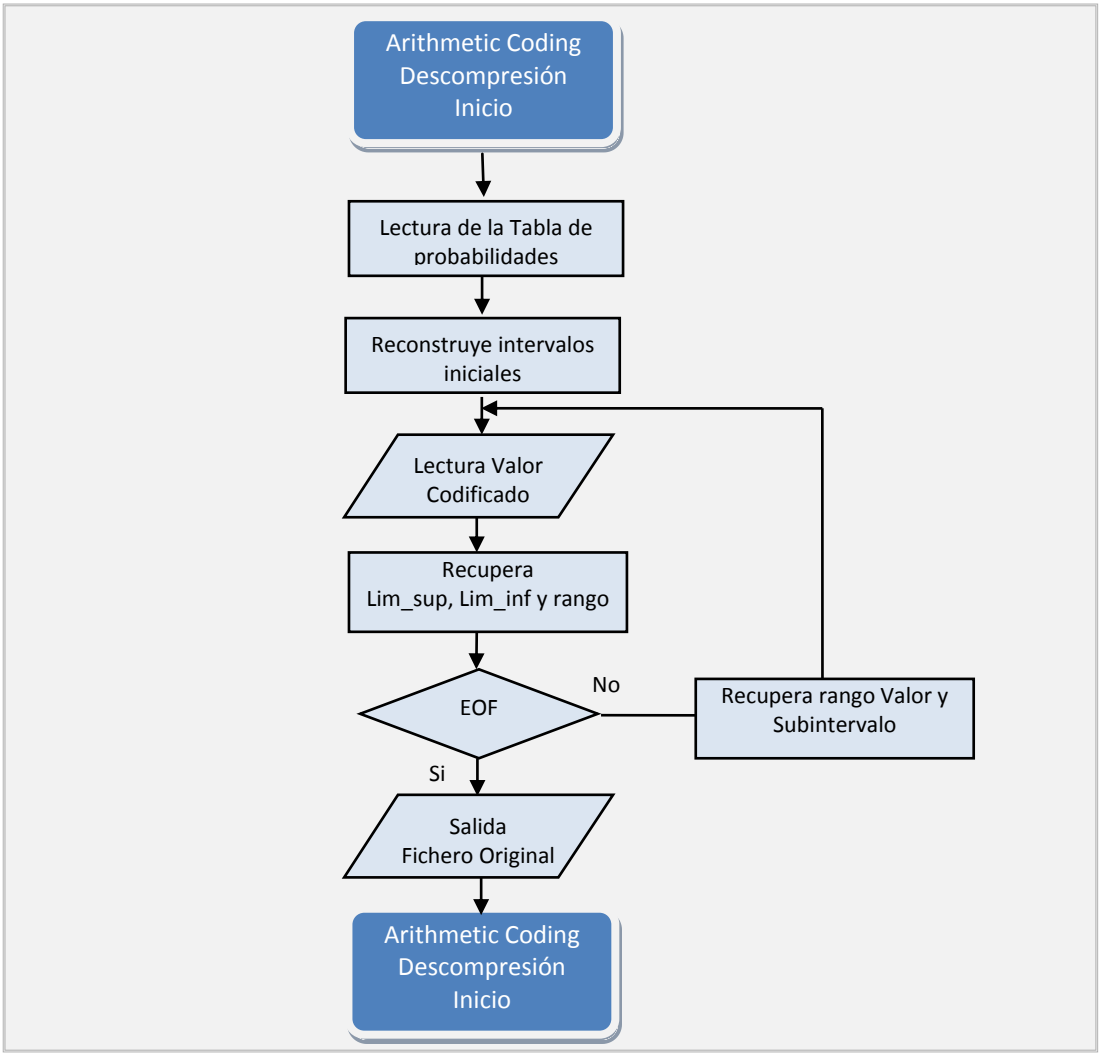


Figura 3.40. Flujograma del algoritmo de descompresión Aritmético

3.4.3.4 Ejemplo práctico

Aplicando el ejemplo propuesto en el proceso de compresión, el valor de salida pertenece al intervalo representado por el símbolo "C" o en el caso del píxel el definido por el valor 100. Luego el primer dato a descomprimir es "C".

$$\begin{matrix} 0 \leq \text{valor comprimido} < 0{,}10 \\ 0{,}0575638516 \end{matrix} \tag{3.26}$$

Los siguientes valores se obtienen de la expresión:

$$\text{Valor} = \frac{(\text{Valor anterior} - \text{Límite inferior anterior})}{(\text{Limite superior anterior} - \text{Límite inferior anterior})} \tag{3.27}$$

Tabla 3-28. Reconstrucción de la tabla

VALOR CALCULADO	RESULTADO	RANGO	SÍMBOLO	PÍXEL
	0,0575638516	[0,00 , 0,10]	C	100
$\frac{(0,0575638516 - 0,00)}{(0,0 - 0,00)}$	0,575638516	[0,5 , 0,7]	O	65
$\frac{(0,575638516 - 0,5)}{(0,7 - 0,5)}$	0,37819258	[0,3 , 0,4]	M	6
$\frac{(0,7819258 - 0,3)}{(0,4 - 0,3)}$	0,7819258	[0,7 , 0,8]	P	22
$\frac{(0,7819258 - 0,7)}{(0,8 - 0,7)}$	0,819258	[0,8 , 0,9]	R	11
$\frac{(0.819258 - 0.8)}{(0.9 - 0.8)}$	0,19258	[0,1 , 0,2]	E	201
$\frac{(0,19258 - 0,1)}{(0,2 - 0,1)}$	0,9258	[0,9 , 1.0]	S	73
$\frac{(0,9258 - 0,9)}{(1,0 - 0,9)}$	0,258	[0,2 , 0,3]	I	5
$\frac{(0,258 - 0,2)}{(0,3 - 0,2)}$	0,58	[0,5 , 0,7]	O	65
$\frac{(0,58 - 0,5)}{(0,7 - 0,5)}$	0,4	[0,4 , 0,5]	N	7

3.4.4 El Codificador Aritmético: modelo adaptativo

El Codificador Aritmético y el Codificador Huffman son métodos basados en modelos estadísticos y en consecuencia es posible aplicar versiones basadas en modelos adaptativos (Langdon y Risannen, 1983). En los modelos estáticos las probabilidades de los símbolos al inicio del proceso de compresión son fijas y conocidas. Por el contrario, en los modelos adaptativos son dinámicas (Liang, Demin y Dong, 2012). Esto significa que las probabilidades al inicio del proceso son desconocidas y se van estimando a medida que avanza el proceso de codificación. El principio básico de funcionamiento del algoritmo se mantiene inalterable en ambos modelos. En las Fig. 3.41 y 3.42 se ilustran los flujogramas de los modelos adaptativos de ambos ciclos.

3.4.4.1 Codificador Aritmético adaptativo

La Fig. 3.41 muestra el diagrama de bloques del proceso de compresión del modelo Aritmético adaptativo.

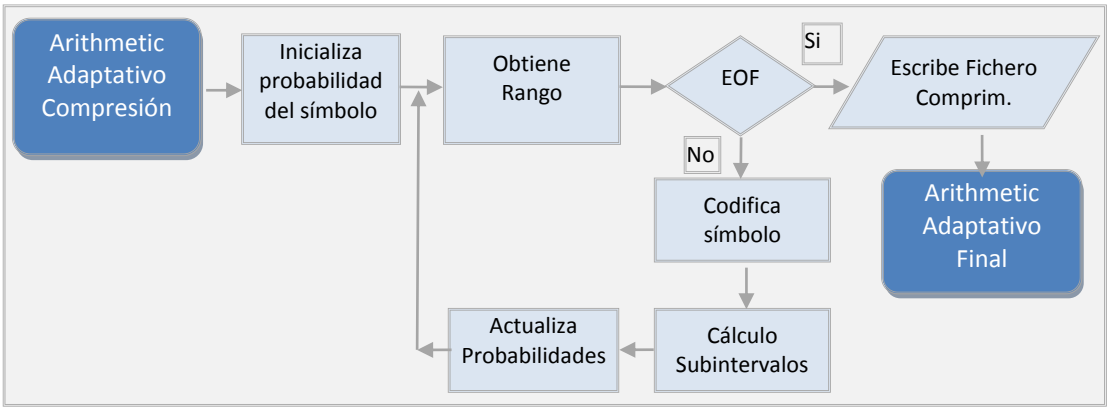


Figura 3.41. Diagrama de bloques de un Codificador Aritmético adaptativo

El fichero comprimido contiene la tabla de probabilidades de los píxeles y el código binario correspondiente al valor de rango inferior del último subintervalo generado en el codificador.

3.4.4.2 Decodificador Aritmético adaptativo

En la figura 3.42 se muestra el diagrama de bloques del proceso de descompresión del modelo aritmético adaptativo.

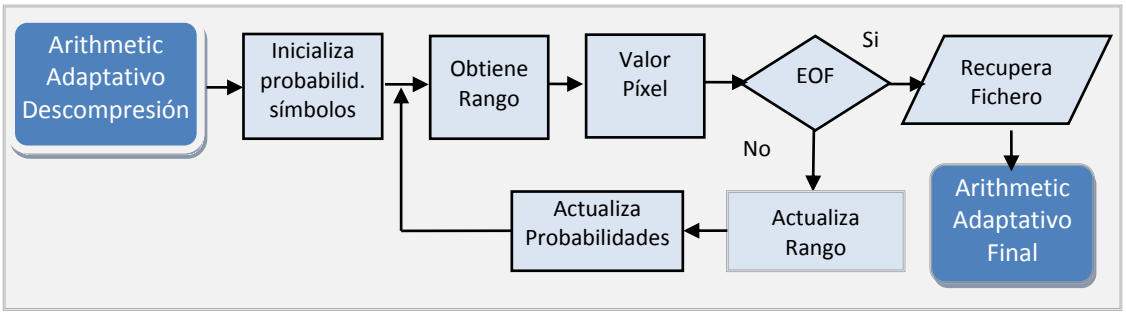


Figura 3.42. Diagrama de bloques de un Decodificador Aritmético adaptativo

La versión adapativa del Codificador Aritmético mantiene la característica de ralentización de los tiempos de ejecución que la versión adaptativa de Huffman. En ambos casos, requiere la actualización de los códigos por cada entrada de datos. La velocidad de procesamiento del Codificador Aritmético adaptativo es superior a Huffman porque no tiene que actualizar el árbol binario.

3.4.5 Resultados experimentales

La tabla 3.29 contiene los resultados de compresión de ambos modelos: estáticos y adaptativos.

Tabla 3-29. Resultados experimentales del Codificador Aritmético

IMAGEN FUENTE	NOMBRE IMAGEN	CODIFICADOR ARITMÉTICO ESTATICO	CODIFICADOR ARTITMÉTICO ADAPTATIVO
		RATIO DE COMPRESIÓN	RATIO DE COMPRESIÓN
Fotográficaas	Airplane	1,200	1,286
	Baboon	1,089	1,107
	Barbara	1,050	1,080
	Boats	1,112	1,184
	Cameraman	1,131	1,211
	Goldhill	1,072	1,164
	Lena	1,076	1,104
	Man	1,115	1,182
	Peppers	1,054	1,088
	Zelda	1,101	1,136
	Media	**1,100**	**1,154**
Aéreas y Satélite	Aerial	1,145	1,163
	Airfield	1,126	1,152
	Earth	1,218	1,251
	Meteosat	1,085	1,159
	Moon	1,227	1,340
	Moonsurface	3,564	3,707
	SanDiego	1,096	1,094
	WashingtonIR	1,065	1,089
	Media	**1,441**	**1,494**
Creadas por Ordenador	Circles	4,561	5,391
	Gray	1,828	3,585
	Slope	1,068	1,115
	Squares	7,533	8,142
	Media	**3,748**	**4,558**
Médicas	Elbowx	1,311	1,383
	Finger	1,131	1,141
	MRI_Brain1	1,273	1,314
	MRI_Brain2	1,588	1,648
	MRI_Brain3	1,162	1,182
	Shoulder	1,229	1,284
	Media	**1,282**	**1,325**
Textos y Gráficos escaneados	Mercados1	1,485	1,837
	Mercados2	1,570	2,109
	Mercados3	1,611	1,960
	Mercados4	1,816	2,572
	Media	**1,621**	**2,120**

Como se muestra en la Tabla 3.29, los resultados experimentales reflejan que los ratios de compresión son mejores que los ratios del método Huffman en todas las imágenes evaluadas. Al igual que el Codificador Huffman, el Codificador Aritmético alcanza los mejores ratios de compresión en las imágenes generadas por ordenador, aunque los ratios de compresión siguen estando lejos de los alcanzados por el método RLE.

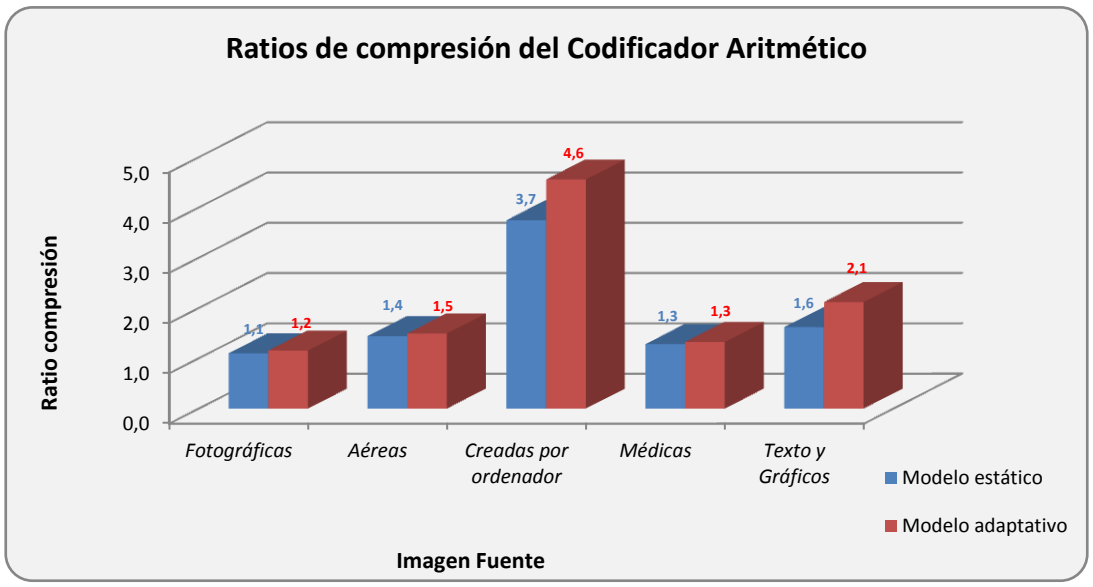


Figura 3.43. Ratios de compresión del Codificador Aritmético

La Fig. 3.43 muestra claramente que los modelos adaptativos consiguen mejores ratios de compresión que los modelos estáticos en todas las clases de imagen fuente.

En términos de consumos de memoría, el Codificador Aritmético utiliza operaciones en coma flotante y esto significa que la precisión de los ordenadores debería ser siempre igual o mayor en el ordenador del proceso de descompresión. Estas operaciones no requieren tanta memoria como precisa el Codificador Huffman porque no utiliza estructura de árbol binario. Sin embargo, los consumos son significativamente muy superiores a los requeridos *Bitmap Compression* y RLE.

En las Tablas 3.30 y 3.31 se reflejan los resultados experimentales de las mediadas de tiempo y ciclos de CPU para ambos modelos. Los ratios de compresión del modelos adaptativos son sensiblemente superiores a los correspondientes al módelo estático pero los tiempos y ciclos de CPU de los ciclos de compresión y descompresión siempre son mayores, como ocurre en todos los modelos adaptativos. Sin embargo, estas diferencias no son tan notables como ocurre en el Codificador Huffman, y por tanto, el modelo aritmético adaptativo es operativo. Las Tablas 3.30 y 3.31 muestran los resultados para los modelos estático y adaptativo.

Tabla 3-30. Codificador Aritmético estático. Tiempos de ejecución del ciclo de compresión y descompresión

IMAGEN FUENTE	NOMBRE IMAGEN	CICLO DE COMPRESIÓN		CICLO DE DESCOMPRESIÓN	
		CICLOS CPU	TIEMPO (ms)	CICLOS CPU	TIEMPO (ms)
Fotográficaas	Airplane	455.139	206,09	466.174	211,09
	Baboon	483.168	218,78	511.353	231,55
	Barbara	499.692	226,27	530.677	240,30
	Boats	483.348	218,87	499.986	226,40
	Cameraman	174.838	79,17	188.626	85,41
	Goldhill	481.311	217,94	497.849	225,43
	Lena	474.311	214,77	495.724	224,47
	Man	173.165	78,41	169.105	76,57
	Peppers	487.003	220,52	529.788	239,89
	Zelda	473.056	214,21	475.976	215,53
	Media	**418.503**	**189,50**	**436.526**	**197,66**
Aéreas y Satélite	Aerial	490.951	222,31	481.290	217,93
	Airfield	463.600	209,92	487.377	220,69
	Earth	139.236	63,05	145.754	66,00
	Meteosat	1.088.645	492,95	1.140.735	516,54
	Moon	5.713.702	2.587,24	5.819.779	2.635,27
	Moonsurface	289.433	131,06	301.118	136,35
	SanDiego	1.762.644	798,15	1.751.406	793,06
	WashingtonIR	8.153.828	3.692,16	8.371.805	3.790,86
	Media	**2.262.755**	**1.024,61**	**2.312.408**	**837,67**
Creadas por Ordenador	Circles	111.473	50,48	121.692	55,10
	Gray	371.348	168,15	377.856	171,10
	Slope	160.997	72,90	167.624	75,90
	Squares	699.234	316,62	117.693	53,29
	Media	**335.763**	**152,04**	**196.216**	**88,85**
Médicas	Elbowx	442.550	200,39	475.899	215,49
	Finger	169.080	76,56	177.442	80,35
	MRI_Brain1	157.509	71,32	171.322	77,58
	MRI_Brain2	121.710	55,11	170.910	77,39
	MRI_Brain3	208.308	94,32	270.437	122,46
	Shoulder	299.219	135,49	291.718	132,09
	Media	**233.063**	**105,53**	**259.621**	**117,56**
Textos y Gráficos escaneados	Mercados1	1.445.444	654,52	1.388.297	628,64
	Mercados2	1.410.222	638,57	1.345.790	609,39
	Mercados3	1.047.759	474,44	1.058.110	479,13
	Mercados4	549.278	248,72	556.125	251,82
	Media	**1.113.176**	**504,06**	**1.087.081**	**492,25**

Tabla 3-31. Codificador Aritmético adaptativo. Tiempos de ejecución del ciclo de compresión y descompresión

IMAGEN FUENTE	NOMBRE IMAGEN	CICLO DE COMPRESIÓN		CICLO DE DESCOMPRESIÓN	
		CICLOS CPU	TIEMPO (ms)	CICLOS CPU	TIEMPO (ms)
Fotográficaas	Airplane	627.613	284,19	706.322	319,83
	Baboon	825.695	373,89	901.056	408,01
	Barbara	866.246	392,25	916.883	415,18
	Boats	811.065	367,26	861.924	390,29
	Cameraman	256.927	116,34	325.299	147,30
	Goldhill	857.953	388,49	937.963	424,72
	Lena	853.870	386,64	948.147	429,33
	Man	302.142	136,81	296.175	134,11
	Peppers	895.057	405,29	995.854	450,94
	Zelda	944.063	427,48	1.007.023	455,99
	Media	**724.063**	**327,86**	**789.665**	**357,57**
Aéreas y Satélite	Aerial	671.193	303,92	767.614	347,59
	Airfield	838.758	379,80	860.322	389,57
	Earth	197.548	89,45	242.431	109,78
	Meteosat	2.337.137	1.058,29	2.464.430	1.115,93
	Moon	13.641.503	6.177,05	14.392.061	6.516,00
	Moonsurface	474.317	214,78	567.140	256,81
	SanDiego	3.688.215	1.670,07	3.950.184	1.788,69
	WashingtonIR	16.527.577	7.483,90	17.759.000	8.041,51
	Media	**4.797.031**	**2.172,16**	**5.125.398**	**2.320,74**
Creadas por Ordenador	Circles	225.976	102,32	300.464	136,05
	Gray	647.935	293,39	710.420	321,69
	Slope	243.939	110,46	280.442	126,99
	Squares	238.454	107,97	260.742	118,07
	Media	**339.076**	**153,54**	**388.017,00**	**175,70**
Médicas	Elbowx	1.079.688	488,90	1.150.736	521,07
	Finger	229.935	104,12	294.274	133,25
	MRI_Brain1	296.314	134,17	321.515	145,59
	MRI_Brain2	242.953	110,01	275.998	124,98
	MRI_Brain3	543.429	246,07	609.160	275,84
	Shoulder	529.809	239,90	635.509	287,77
	Media	**487.021**	**220,53**	**547.865**	**248,08**
Textos y Gráficos escaneados	Mercados1	1.675.207	758,56	1.836.006	831,37
	Mercados2	1.762.346	798,01	1.939.867	878,40
	Mercados3	1.183.969	536,12	1.284.262	581,53
	Mercados4	624.244	282,67	730.087	330,59
	Media	**1.311.442**	**593,84**	**1.447.556**	**655,47**

3.4.6 Conclusiones del Codificador Aritmético

El Codificador Aritmético consigue mejores resultados de compresión que los métodos *RLE* y Codificador Huffman en todas las clases de imagen fuente excepto en las imágenes gráficas creadas por ordenador. A pesar de esto, su uso no es tan popular como los anteriores por los siguientes inconvenientes:

- Consume muchos recursos en términos de CPU y memoria. El uso ideal sería en un computador que tuviera infinita precisión.
- Posibles errores de *overflow* en equipos con diferentes precisiones. La precisión normal de los computadores PC oscilan entre 16, 32 o 64 bits (Sodagar, Bing-Bing y Wus, 2000).
- Incompatibilidad entre los computadores por la precisión utilizada. Algunos modelos de ordenadores utilizan diferente longitud de bits en coma flotante.
- El modelo es extremadamente preciso, un simple error de un *bit* en la entrada de datos puede generar una salida completamente diferente a la imagen original.
- El proceso de compresión se realiza en bloques reducidos de píxeles, teniéndose que repetir el proceso continuamente.

Una posible solución a la mayoría de estos inconvenientes consistiría en utilizar enteros sin signo para representar los límites inferior y superior de cada rango. Los valores límites de los rangos se reducen progresivamente y cuando el rango del intervalo es muy pequeño, los dígitos más significativos del valor inferior y superior pueden coincidir. En este caso, se concatena un cero al valor del límite inferior y un nueve al valor superior.

Una alternativa a los métodos basados en modelos estadísticos son los métodos de compresión basados en modelos de diccionario, los cuales se describen en la siguiente sección. Estos métodos utilizan operaciones con punteros como alternativa a los cálculos probabilísticos y al uso de operaciones aritméticas propias de los modelos estadísticos y que afectan a los recursos de CPU.

3.5 Métodos de compresión basados en modelos de diccionario

Los métodos de compresión basados en modelos de diccionario pertenecen exclusivamente a la categoría de compresores sin pérdida de datos. Los métodos revisados anteriormente *Bitmap*, *RLE*, Codificador Huffman y Codificador Aritmético procesan en cada ciclo de entrada de datos un único símbolo de longitud fija como unidad de entrada al modelo para obtener un código de salida basándose en la redundancia o probabilidad de aparición del símbolo.

Los métodos basados en modelos de diccionario procesan datos de longitud variable formados por cadenas de datos en la entrada para generar un código de longitud fija en la salida que contiene una dirección o puntero al diccionario en el caso que haya sido procesado anteriormente. El diccionario crece progresivamente a medida que se procesan los datos de entrada.

Las técnicas de diccionario no son aplicaciones exclusivas de compresión de datos (Renugadevi y Nithya, 2013). Estos modelos son utilizados en los más importantes archivadores de compresión: ZIP (Katz, P., 1989a), PKZIP (Katz, P., 1989b), ARJ (*Archived RobertJung*) (ARJ, 2014) y en formato de fichero gráficos estándares GIF (*Graphics Interchange Format*) (Compuserve, 1990) y TIFF (*Tagged Image File Format*) y en estándares de compresión da datos y de telecomunicaciones.

3.5.1 Introducción

Los métodos basados en modelos de diccionario pueden utilizar los dos tipos de modelos: estáticos y adaptativos, tal y como muestra la Fig. 3.44.

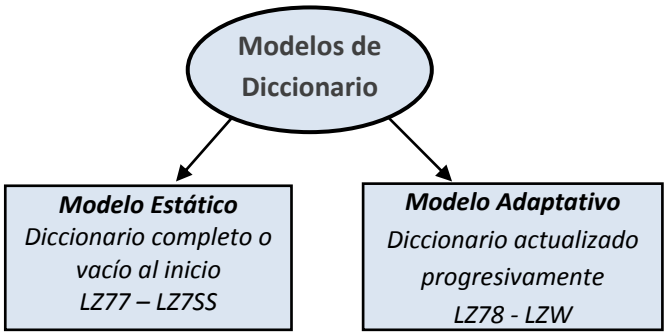


Figura 3.44. Clasificación de los métodos basados en diccionario

Los modelos estáticos cargan en memoria el diccionario antes de comenzar el ciclo de compresión y permanece invariable durante todo el proceso. Estos modelos suelen utilizar diccionarios universales y en algunas aplicaciones específicas implica un conocimiento previo del tipo de la imagen fuente. Los ratios de compresión que se obtienen son muy aceptables. Sin embargo, necesitan almacenar el diccionario con el fichero comprimido para recuperar los datos. La Fig. 3.45 muestra el diagrama de bloques del modelo de diccionario estático.

Figura 3.45. Modelo de diccionario estático

El diccionario adaptativo se crea y actualizan durante el proceso de entrada de datos sin disponer de información previa de los datos de entrada (Fig. 3.46).

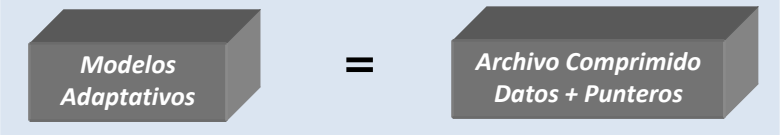


Figura 3.46. Modelo de diccionario adaptativo

Los modelos estáticos establecen una correspondencia entre los datos de entrada y la cadena más larga del los datos procesados anteriormente. Si la cadena de entrada existe en el diccionario, entonces genera un puntero a la cadena del diccionario. Los métodos que aplican este modelo se basan en el algoritmo LZ77 creado por Abraham Lempel y Jakov Ziv en 1.977 (Ziv y Lempel,1977).

Los modelos adaptativos no incluyen un diccionario inicial. En el proceso de compresión actualiza el diccionario con los datos de entrada y si la cadena de entrada ya existe en el diccionario entonces genera un puntero de salida al diccionario. Estos métodos están basados en el Algoritmo LZ78 de Lempel y (Ziv y Lempel, 1977). Terry Welch, basándose en este último algoritmo, desarrolló una implementación por hardware para controladores de discos denominado *LZW* (Welch, 1984).

En esta sección, los modelos estáticos y adaptativos son revisados mediante el estudio y análisis de los dos métodos de diccionario más representativos de la familia de los modelos de diccionario: LZ77 y LZW.

3.5.2 Método de diccionario basado en un modelo estático: LZ77

Las primeras técnicas de diccionario utilizadas como método de codificación para eliminar la redundancia en los datos se remonta al trabajo de Schwartz E.S. (Schwartz y Kallick, 1964) en los cuales describió una técnica de diccionario estático para codificar texto con códigos fijos. La idea teórica de esta técnica se basa en seleccionar un diccionario que contenga entre 500 y 1000 palabras. Estos tipos de diccionarios contienen un 75% de la mayoría de palabras usadas en los textos convencionales en lengua inglesa.

Posteriormente, nuevas técnicas de diccionario fueron utilizadas para codificar texto. Todas estas técnicas construyen diferentes clases de diccionarios estáticos, incluso combinan esta técnica con RLE (Tamakoshi y cols., 2013).

El gran avance y principal aportación al desarrollo de los métodos basados en modelos de diccionario es atribuida a Lempel-Ziv quien resolvió la problemática de las anteriores técnicas aplicando simultáneamente el mismo método de diccionario en transmisor y en el receptor.

El algoritmo *LZ77* es un método de compresión basado en un modelo de diccionario estático. Este método es considerado como el precursor de los métodos actuales basados en modelos de diccionario. Las siguientes modelos de diccionario adaptativos LZ78 y LZW son optimizaciones y versiones de este método.

3.5.3 Proceso de compresión

La idea básica consiste en procesar los datos de entrada y buscar una correspondencia exacta en el diccionario. La cadena de entrada de mayor longitud que es encontrada en el diccionario produce un código de salida en el compresor.

Este método a diferencia de los métodos de diccionario convencionales que usan modelos estáticos no utiliza un diccionario con información previa antes de la compresión. El diccionario es una parte de la secuencia de datos o píxeles de entrada procesados. La unidad de tratamiento de los datos de entrada datos al modelo es el píxel. El modelo procesa píxel a píxel formando cadenas de datos. Esto es, la entrada en el codificador forma una cadena de longitud variable. En aplicaciones gráficas y de imágenes, este diccionario contiene los valores posibles de los píxeles desde 0 a 256. El diccionario es temporal, no es transmitido ni almacenado, y es utilizado como zona de memoria temporal para contener la secuencia de códigos codificados previamente. La fig. 3.47 ilustra el diagrama de bloques del mproceso de compresión del método LZ77.

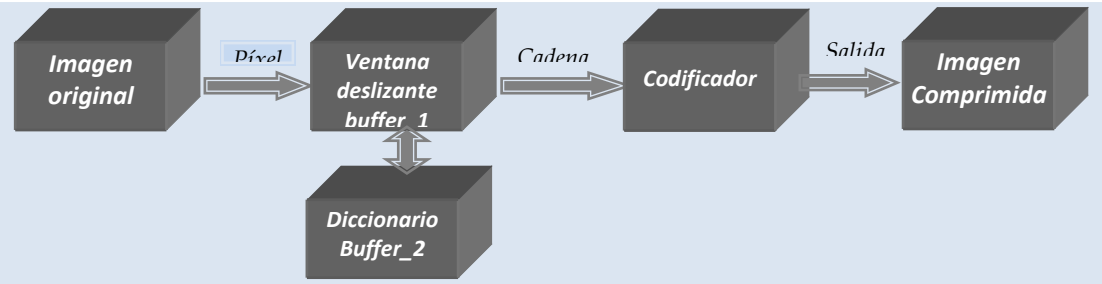


Figura 3.47. Diagrama de bloques del ciclo de compresión LZ77

LZ77 utiliza una ventana dividida en dos partes independientes destinadas a contener en una de ellas datos del diccionario y en la otra los valores a comprimir. El tamaño de la ventana utilizado suele ser de un tamaño reducido y fijo. Esta ventana suele utilizar una zona de memoria comprendida entre 4K y 16K. El diccionario contiene los datos procesados previamente y que pueden ser almacenados en esa zona limitada. La zona de datos contiene la siguiente cadena de datos de entrada a ser comprimidos. A medida que los datos son procesados en la entrada ellos son eliminados de esta zona y la siguiente cadena de datos es almacenada desde el fichero fuente. En la práctica, el tamaño del diccionario crece progresivamente a medida que son procesados los datos de entrada, mientras que el

tamaño de la zona de datos se va reduciendo progresivamente (Fig. 3.48). Por esta razón, este método de compresión es conocido popularmente como método de diccionario de *ventana deslizante*.

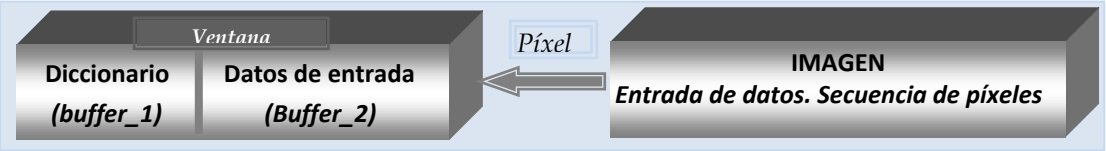


Figura 3.48. Ventana deslizante, buffer_1 y buffer_2

El algoritmo utiliza dos zonas de memoria o *buffers* independientes. Una primera zona temporal, *buffer_1* contiene el diccionario o secuencia de datos codificados a partir de los datos procesados. Una segunda zona, *buffer_2* contiene la siguiente cadena de datos pendientes de codificar.

El codificador analiza el *buffer_2* para determinar la cadena de entrada a codificar y localiza en el *buffer_1* el primer símbolo que coincide con el primer valor de la cadena de entrada. Una vez encontrada una correspondencia en el diccionario, usa un puntero de desplazamiento (*offset)*. Este puntero indica la distancia entre el inicio de la cadena actual en el *buffer_2* y su correspondencia en el diccionario.

El codificador aprovecha la dirección actual para examinar el siguiente símbolo del diccionario y analizar si existe correspondencia con el siguiente símbolo de entrada que está apuntado por el *offset + 1*. El número de símbolos consecutivos del *buffer_2* que se corresponden con los mismos símbolos en el *buffer_1* se denomina *longitud de correspondencia*.

Cuando el codificador consigue la máxima longitud de correspondencia entre la cadena de entrada y el diccionario, entonces codifica esta secuencia en la salida como una terna de enteros o tupla <*o,l,s*>, donde "*o*" representa el *offset* , "*l*" la longitud de correspondencia y "*s*" es el siguiente carácter de la cadena codificada.

En el caso de que no exista una correspondencia entre el símbolo de entrada y el diccionario, entonces, el valor del *offset* y la longitud de correspondencia es igual a cero, y el tercer elemento es reemplazado por el mismo símbolo de entrada <*0,0,s*>. El código es almacenado en la salida y la subcadena es eliminada del *buffer_2*.

La cantidad de compresión depende del tamaño del diccionario o *buffer_1*, del *buffer_2* y de la entropía de la fuente con respecto al modelo LZ77. Para facilitar el cálculo de número de bits necesario para codificar la entrada de datos "*N*", el tamaño del *buffer_1* es representado como "*D*", el tamaño en el *buffer_2* como "B", y la longitud total de la entrada de datos como "*S*".

$$\mathrm{N} = \log_2 \mathrm{D} + \log_2 \mathrm{B} + \log_2 \mathrm{S} \qquad (3.28)$$

3.5.3.1 Algoritmo de compresión

Los pasos del algoritmo son los siguientes

Paso 1. Inicializa punteros en el inicio del buffer_1(diccionario) y buffer_2 (datos)
Paso 2.offset=0, longitud =0, string= “”
Paso 3. Ciclo de lectura de buffer_2 (datos) hasta final de fichero.
Paso 4. Lee píxel del buffer_2
Paso 5. Comprueba si existe correspondencia en el diccionario
Paso 5.1 ¿Si?
Paso 5.1.1 offset=puntero (posición en buffer_1)
Paso 5.1.2 string = píxel
Paso 5.1.3 longitud=longitud +1
Paso 5.1.4 Incrementa en 1 el puntero del buffer_1
Paso 5.1.5 Repite ciclo desde paso 4
Paso 5.2 ¿No?
Paso 5.2.1 Salida codificada (o, l, s)
Paso 5.2.2 string = píxel
Paso 5.2.3 longitud=0
Paso 5.2.4 offset=0
Paso 6. Fin de buffer_2 (datos)

3.5.3.2 Programación del algoritmo

```
// Algoritmo LZ77
píxel= offset_;
cadena="";
for(x=0; x<end_buffer_2; x++){
    get píxel(x);
    cadena=str(cadena +píxel(x));
    for(y=0 ; y<end_buffer_1; y++) {
       if (píxel (x)<>datos(y)) then
           marca=0;
       else
           offset= y;
           marca=1;
       end if }
    int longitud =0, sal=0;
    i= x;
    j=offset;
    if (marca==1) then
       while (sal==0)
         if  (píxel(i)==datos(j)) then
               longitud ++;
               j++;
               i++;
        else
               salida(offet, longitud, píxel(x+1));
        end if
    endif
    if (marca==0) then
       offset=0;
       longitud=0;
       salida(offet, longitud, píxel(x+1));
end if
}
```

Listado 3.10. Programación del algoritmo de compresión LZ77

3.5.3.3 Flujograma del algoritmo de compresión LZ77

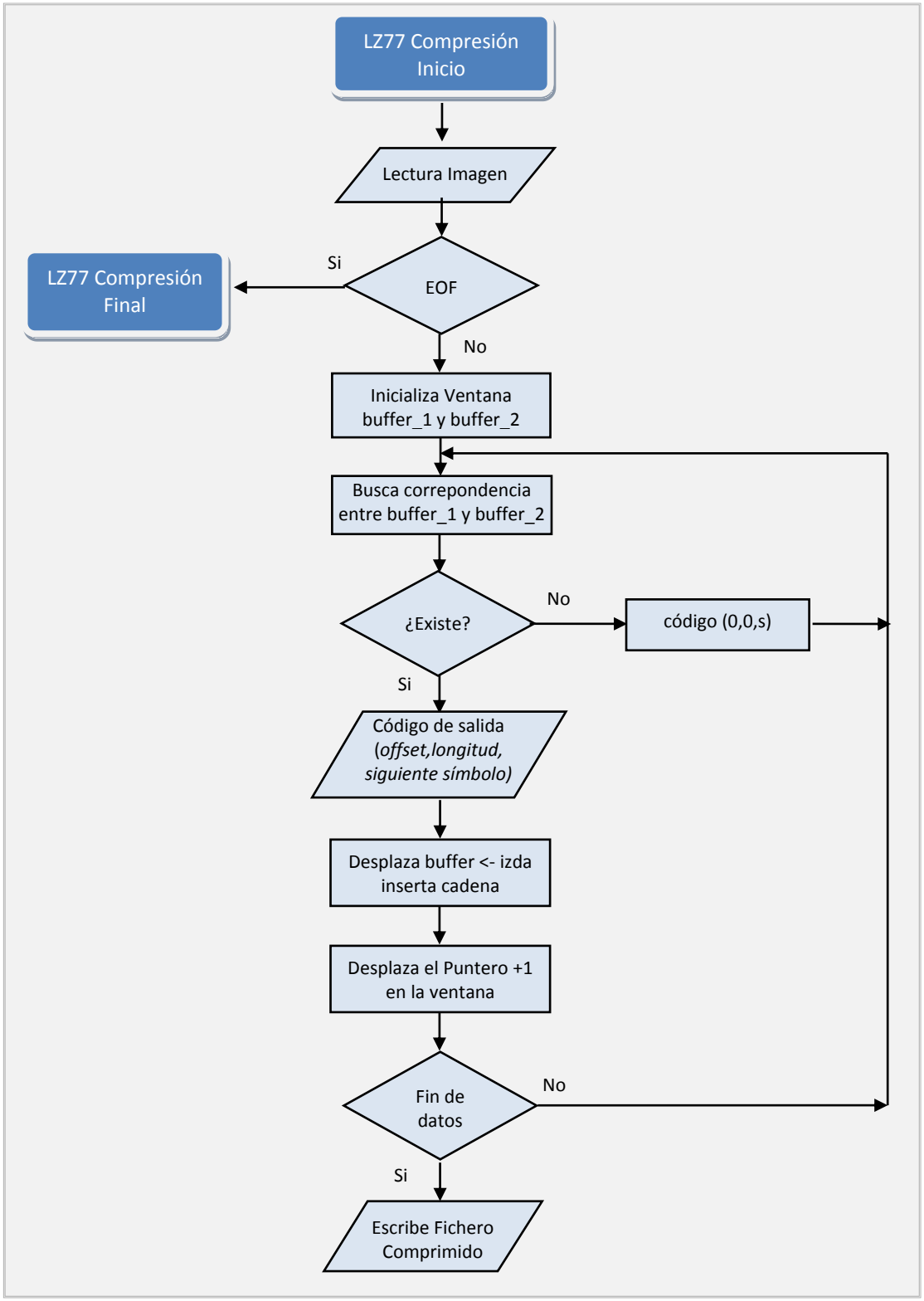


Figura 3.49. Flujograma del algoritmo de compresión LZ77

3.5.3.4 Ejemplo práctico

Dada la siguiente secuencia de píxeles de entrada: "121-118-121-118-123-118-121-118-121-118-121-121-121-121-121-121" el algoritmo genera los códigos de salida con las ternas de datos, paso a paso, siguiendo la secuencia del algoritmo LZ77, como se refleja en la Tabla 3.32.

Tabla 3-32. Secuencia de codificación del ciclo de compresión LZ77

	DICCIONARIO BUFFER_1	CADENA DE DATOS BUFFER_2	FICHERO DE ENTRADA	CÓDIGO SALIDA	COMENTARIOS
			121-118-121-118-123-118-121-118-121 118-121-121-121-121-121-121		Diccionario Vacío
		121-118-121-118	123-118-121-118-121-118-121-121-121-121-121-121	*0 0 121*	Carga datos en Buffer_2
	121	118-121-118-123	118-121-118-121-118-121-121-121-121-121-121	*0 0 118*	118 no encontrado
	121-118	121-118-123-118	121-118-121-118-121-121-121-121-121-121	*2 2 123*	121-118 encontrado
121	118-121-118-123	118-121-118-121	118-121-121-121-121-121-121	*0 3 121*	118-121-118 encontrado
121-118-121-118-123	118-121-118-121	118-121-121-121	121-121-121	*0 2 121*	118-121 encontrado
121-118-121-118-123-118-	121-118-121-121	121-121-121-121		*2 3 121*	121-121-121 encontrado

3.5.4 Proceso de descompresión

El proceso de descompresión como en la mayoría de los métodos es más sencillo y rápido que el proceso de compresión. En el proceso de descompresión no es necesario realizar comparaciones *ni* sustituciones en la ventana. Únicamente requiere una ventana de las mismas características que el compresor. Los códigos de entrada son reconstruidos a partir la expansión de la tuplas que forman los códigos de salida. El procedimiento consiste en expandir los "*l*" símbolos a partir de la posición del offset "*o*" del diccionario y concatenarlos con el siguiente símbolo "*s*". A continuación, el puntero de la ventana deslizante se desplaza *l + 1* posiciones en la cadena de datos.

3.5.4.1 Algoritmo de descompresión

El algoritmo de descompresión realiza la lectura de tres en tres bytes expandiendo directamente las tuplas de datos. El primer byte contiene el puntero de desplazamiento a partir del cual expande el valor del píxel con la longitud de carrera de avance. El algoritmo genera el contenido de las ventanas siguiendo el

mismo procedimiento que en el proceso de compresión. Los pasos del algoritmo son los siguientes:

Paso 1. Inicializa punteros en el inicio del buffer_1(diccionario) y buffer_2 (datos)
Paso 2.offset=0, longitud =0, string= ""
Paso 3. Ciclo de lectura de los datos comprimidos hasta final de fichero.
Paso 3.1. Por cada tupla (o, l, s)
Paso 3.2. Sustituye en buffer 1 y buffer_2
Paso 3.3 Desplaza puntero en la ventana (longitud +1)
Paso 3.4. Repite ciclo desde paso 4
Paso 4. Reconstruye datos de entrada.
Paso 6. Fin de buffer_2 (datos)

3.5.4.2 Programación del algoritmo de descompresión LZ77

```
// Algoritmo LZ77 ciclo de descompresión
// Ciclo procesamiento de la imagen original
      píxel= offset;
dato=' ';
for(i=0; x<fin_buffer_2; i++)  {
   get píxel[i];
   string=str(string + píxel [i]);
   for(j=0 ; j<fin_buffer_1; j++) {
       if (píxel [i]<>string[j]) then
           marca=0;
       else
           offset= j;
           marca=1;
       end if
   }
   int longitud =0, salida=0;
   y= i;
   z=offset;
   if (marca==1) then
       while (sal==0)
            if (píxel[y]==datos(z)) then
                  longitud ++;
                  z++;
                  y++;
             else
                  salida(offset, longitud, píxel[y+1]);
             end if
    endif
    if (marca==0) then
         offset=0;
         longitud=0;
         salida(offset, longitud, píxel(z+1));
    end if
}
```

Listado 3.11. Programación del algoritmo de descompresión LZ77

3.5.4.3 Flujograma del ciclo de decompression de LZ77

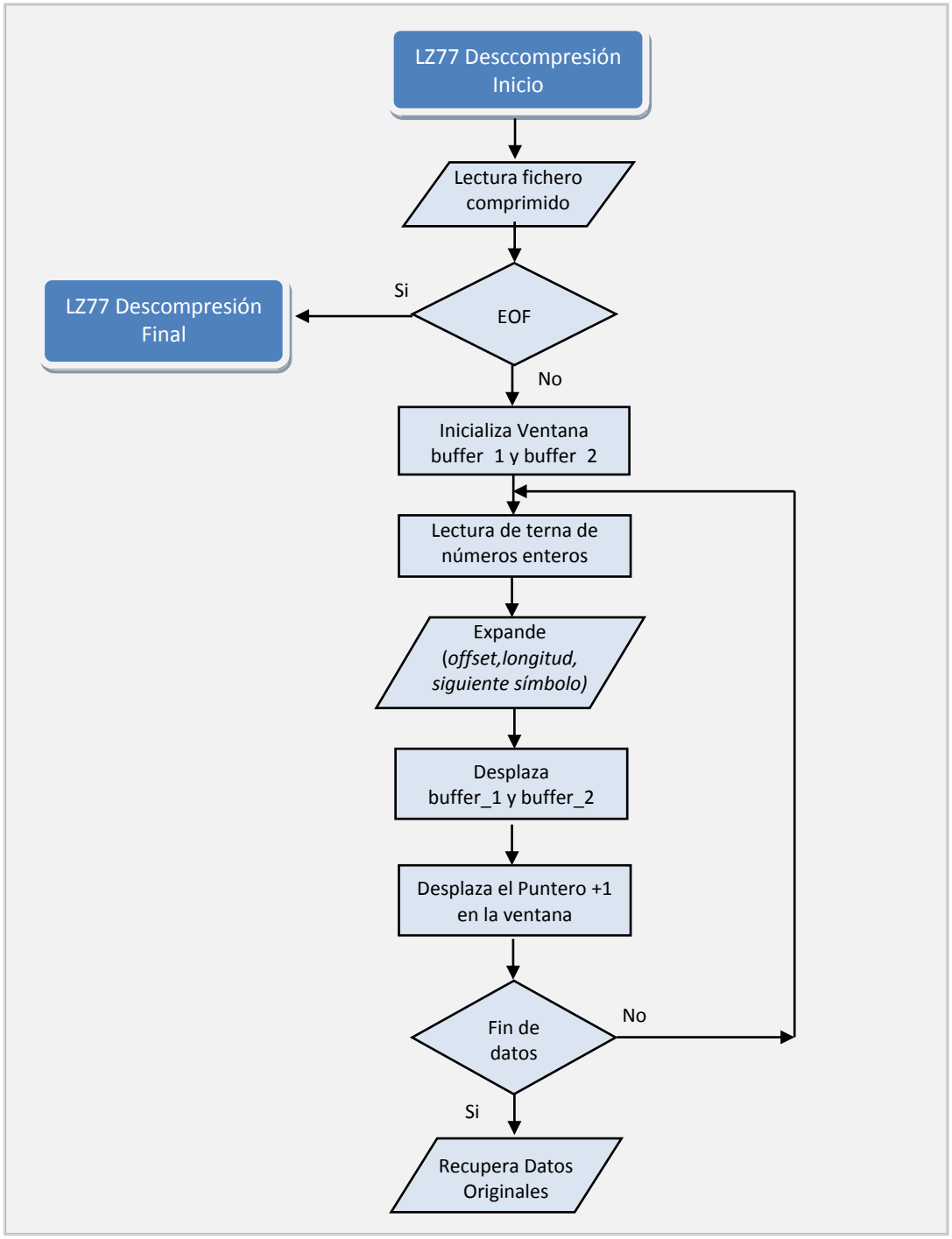


Figura 3.50. Flujograma del algoritmo de descompresión LZ77

3.5.4.4 Ejemplo práctico del ciclo de descompresión

A partir del resultado del ejercicio práctico propuesto en el ciclo de compresión aplicando el algoritmo de descompresión se obtiene la expansión y recuperación de los datos, tal y como se muestra en la Tabla 3.33.

Tabla 3-33. Secuencia del ciclo de descompresión LZ77

CÓDIGO ENTRADA	SALIDA	FICHERO DESCOMPRIMIDO	DICCIONARIO BUFFER_1	CADENA DE DATOS BUFFER_2
0 0 121	121			121
0 0 118	118		121	118
2 2 123	121-118-123		121-118	121-118-123
0 3 121	118-121-118-121	121	118-121-128-123	118-121-118-121
0 2 121	118-121-121	121-118-121-118-123	118-121-118-121	118-121-121
2 3 121	121-121-121-121	121-118-121-118-123-118-121-118-121-118	121-118-121-121	121-121-121-121

3.5.5 Diccionario basado en un modelo adaptativo: LZW

El algoritmo LZW es una variación de LZ78 y es el método más conocido y popular de todos los métodos de diccionario. Como su precursor, recibe el nombre de sus autores, Lempel, Ziv y Welch, quienes aportaron una modificación y optimización a la versión LZ78 y consecuentemente mejores resultados en ratios de compresión. Este método se emplea en los formatos gráficos GIF.

Al igual que todos los métodos de compresión basados en técnicas de diccionario, la entrada de datos al modelo son cadenas de longitud variable. El codido de salida en LZ77 es una terna de enteros <*offset, longitud, símbolo*> que representa la máxima correspondencia entre el diccionario y la cadena en la entrada de datos. El algoritmo adaptativo LZ78, reduce el signo de salida a una representación formada por un par, <*índice, símbolo*>. El índice representa un apuntador al diccionario y el símbolo coincide con el código del string o cadena de la entrada.

Terry Welch propusó una nueva técnica basada en la eliminación del segundo elemento de representación de la correspondencia. El codificador en este caso genera un código de salida único (puntero) para representar el índice de la cadena en el diccionario. Por tanto, el fichero comprimido de salida contendrá una sucesión de punteros o direcciones.

Inicialmente, el algoritmo genera un diccionario con un tamaño de 4 kB tamaño que contiene de 0 a 255 referencias principales de cadenas y de 256 a 4096 referencias o subcadenas.

La entrada al codificador es un conjunto concatenado de símbolos de entrada. La entrada es de tamaño variable y tan grande como la longitud de esa cadena contenida en el diccionario. Cada vez que un símbolo es procesado, si existe en el diccionario genera un código de salida que representa el índice en el diccionario y el

símbolo es añadido al diccionario como nueva cadena. LZW construye el mismo diccionario dinámicamente en el codificador y decodificador.

La Fig. 3.41 muestra el diagrama de bloques del modelo adapatativo LZW.

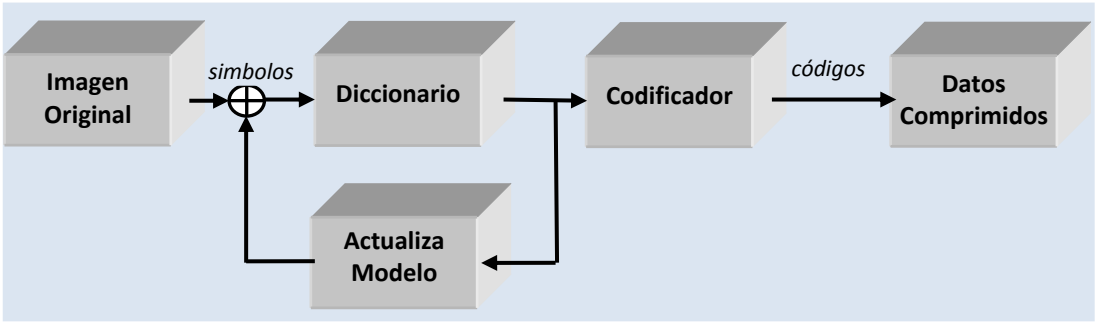


Figura 3.51. Diagrama de bloques de un modelo LZW

El proceso de compresión realiza los siguientes pasos:

(1) Crea la tabla de códigos con un campo índice de 0 a 255 referencias y un campo para las subcadenas.

(2) Inicializa la tabla.

(3) El primer dato o valor del píxel del fichero de entrada es leído. El campo índice es incrementado en uno, el valor del dato es almacenado en el campo de cadenas y el índice es almacenado en el fichero comprimido.

(4) El segundo dato es leído incrementando el índice, el campo de cadenas contendrá el valor anterior más el actual y a continuación escribe en el fichero comprimido.

(5) Continua leyendo los datos de entrada de forma variable o cadenas de datos y comprueba si los datos leídos ya existen como una cadena en la tabla.

(6) En el caso que los datos existan como una cadena entonces escribe el índice correspondiente a esa cadena en el fichero comprimido, genera un nueva cadena en la tabla con el nuevo dato e incrementa el índice. El proceso se repite desde el paso (5).

(7) En el caso que la cadena no exista en la tabla de símbolos entonces almacena el valor del dato leído en la tabla e incrementa el campo índice preparándole para el siguiente dato. El proceso se repite desde el paso (5).

(8) El proceso de compresión finaliza cuando todos los datos de entrada son leídos.

La salida del algoritmo LZW produce códigos de longitud fija que son almacenados en el fichero comprimido.

3.5.5.1 Algoritmo de compresión LZW

El algoritmo LZW opera de forma recursiva para actualizar los índices y generar las cadenas de datos. El índice es un apuntador a una dirección del diccionario o tabla y corresponderá al código de salida.

Paso 1. Inicializa diccionario (w) con entradas de 0-255. Un byte de profundidad
Paso 2. Bucle hasta finalizar fichero
Paso 2.1. procesa un símbolo - k
Paso 2.2 . ¿existe concatenación "w+k" en el diccionario?
Paso 2.3 Si. w=wk
Paso 2.4 No existe correspondencia. Recoloca el indice
Paso 2.4.1 Codifica salida para w
Paso 2.4.2 Concatena "w+k" -> wk
Paso 2.4.3 Añade wk al diccionario
Paso 2.4.4 asigna indice w=k
Paso 3. Vuelve a paso 2
Paso 4. El resultado es una sucesión de bits que forman todos los símbolos de entrada

3.5.5.2 Programación del algoritmo de compresión

Inicialmente el algoritmo almacena una tabla con los 256 índices, de 0 a 255 (líneas 4-6). A continuación procede a leer la imagen de entrada y almacena los valores de los píxeles. Cuando una secuencia se repite es localizada en la tabla y se ctualiza el índice. Las operaciones básicas usadas en este algoritmo son operaciones con punteros y con cadenas.

```
1  // Algoritmo LZW
2  //Algoritmo de Compresión
3  // Inicializa tabla
4  indice=255;
5  for(i=0;i<255, i++)
6      tabla[i]=i;
7  // ciclo del proceso de compresión
8  for(x=0;x<anchura*anchura;x++)
9  {
10   k= array[x];
11   for( i=x;i<indice, i++)
12   {
13     if ((str(w)+str(k))=tabla(i));
14         w=str(w)+str(k);
15     else
16     {
17         y++;
18         salida[y]=w;
19         indice++;
20         tabla[indice]=w;
21         w=k;
22     }
23   }
24 }
```

Listado 3.12. Programación del algoritmo de compresión LZW

3.5.5.3 Flujograma del algoritmo de compresión LZW

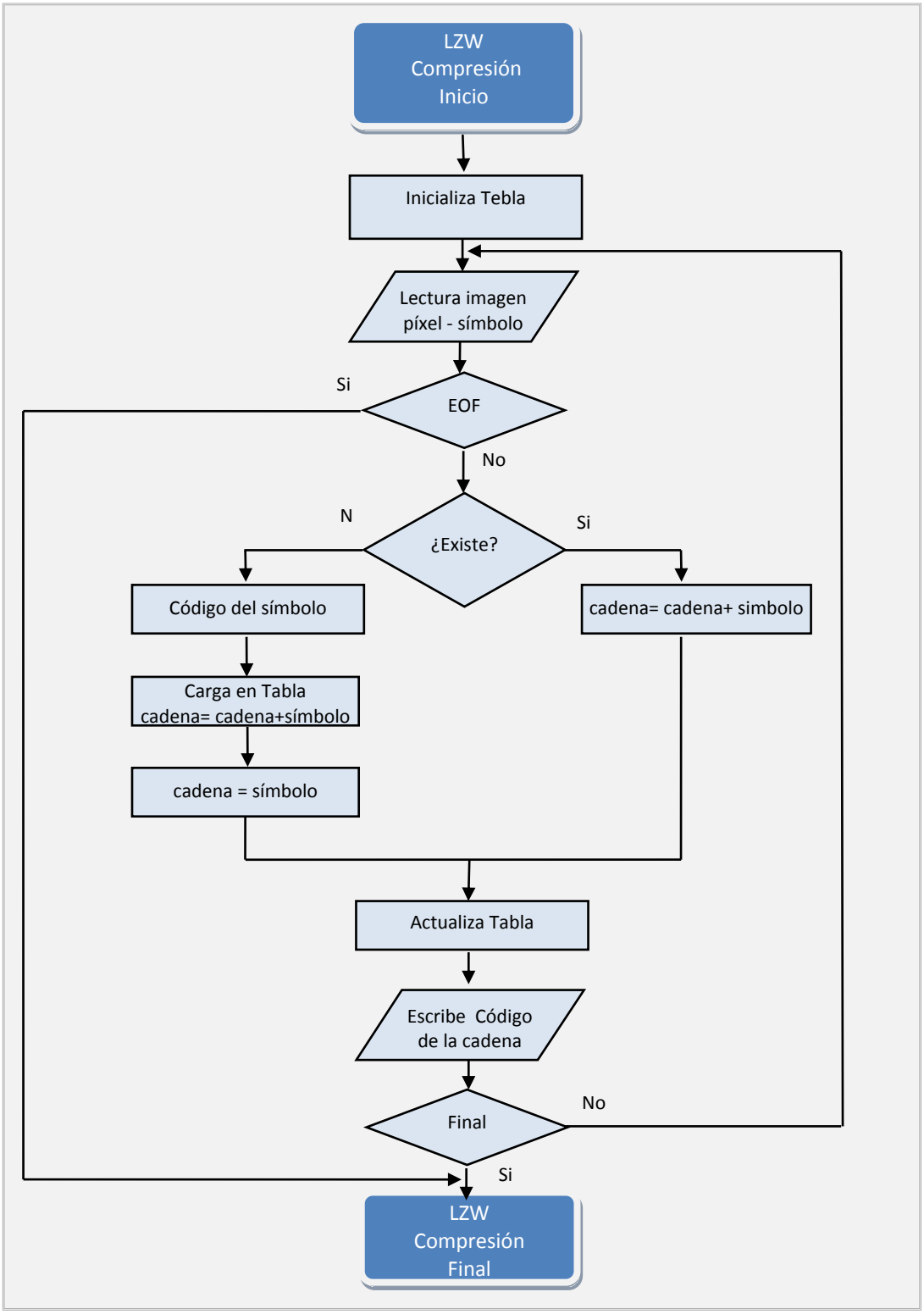


Figura 3.52. Flujograma del algoritmo de compresión LZW

3.5.5.4 Ejemplo práctico de compresión LZW

Dados los valores de los 20 píxeles (8 bits/píxel) de una imagen cualquiera, se aplica el algoritmo LZW para comprimir los datos de entrada. El alfabeto fuente contiene valores de 0 hasta 15.

Datos de entrada:

VALORES																			
p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}	p_{11}	p_{12}	p_{13}	p_{14}	p_{15}	p_{16}	p_{17}	p_{18}	p_{19}	p_{20}
10	11	11	10	09	11	10	11	09	02	06	11	10	11	04	09	11	09	11	09

Tabla 3-34. Creación deuna tabla LZW

ÍNDICE	CÓDIGO	DESCRIPCIÓN
00	00	
01	01	
02	02	
03	03	
...	...	
14	14	
15	15	
16	10-11	Cadena con los dos primeros datos leídos
17	11-11	Incrementa índice y baja ultimo dato leído
18	11-10	Fichero comprimido "10-11-11"
19	10-09	
20	09-11	Fichero comprimido "10-11-11-09-"
21	11-10	11 Cadena existente, escribe su índice 18
22	11-09	Fichero Comprimido "10-11-11-09-18"
23	09-02	Fichero Comprimido "10-11-11-09-18-11"
24	02-06	...
25	06-11	
26	11-10-11	04 Fichero Comprimido :"10-11-11-09-18-11- 09-02-06-21"
27	04-09	
28	09-11	09 Cadena existente. Índice 20
29	09-11-09	Cadena existente. Índice 28

Fichero Comprimido:

DATOS COMPRIMIDOS												
d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8	d_9	d_{10}	d_{11}	d_{12}	d_{13}
10	11	11	09	18	11	09	02	06	21	04	20	28

El tamaño original de 20 bytes es comprimido a 13 bytes.

3.5.5.5 Ejemplo práctico de descompresión

La salida del codificador del ejemplo del modelo estático LZ77 es la secuencia "121-118-121-118-123-118-121-118-121-118-121-121-121-121-121-121" (Tabla 3.35).

Tabla 3-35. Datos de la tabla de compresión LZW

ENTRADA	W	K	SALIDA	COMENTARIO
121-118	121	118	121	D[257] ← 121-118
121	118	121	118	D[258] ←118-121
118	121	118		w = 257
123	257	123	257	D[259] ← 121-118-123
118	123	118	123	D[260] ← 123-118
121	118	121		w = 258
118	258	118	258	D[261] ←118-121-118
121	118	121		w = 258
118	258	118		w = 261
121	261	121	261	D[262] ← 118-121-118-121
121	97	121	121	D[263] ← 121-121
121	97	121		w = 263
121	263	121	263	D[264] ← 121-121-121
121	97	121		w = 263

Tabla 3-36. Reconstrucción de la tabla en el proceso de descompresión LZW

INDICE	CADENA	W	K
0		0	NULL
⋮		⋮	⋮
118		0	118
119		0	119
120		0	120
121			121
122		0	122
...	...	...	...
255		0	255
256	Secuencia de escape - ESC		ESC
257	121-118	121	118
258	118-121	118	121
259	121-118-123	257	123
260	123-118	123	118
261	118-121-118	258	118
262	118-121-118-121	261	121
263	121-121	121	121
264	121-121-121	263	121

3.5.5.6 Proceso de descompresión del LZW

La descompresión es simple y rápida. La idea básica consiste en no transmitir el diccionario. Esto implica que el diccionario reconstruido en el ciclo de descompresión debe ser organizado igual que en el ciclo de compresión. El proceso de descompresión reconstruye la tabla de símbolos durante la lectura de los datos comprimidos.

La Fig. 3.53 muestra el diagrama de bloques del modelo LZW.

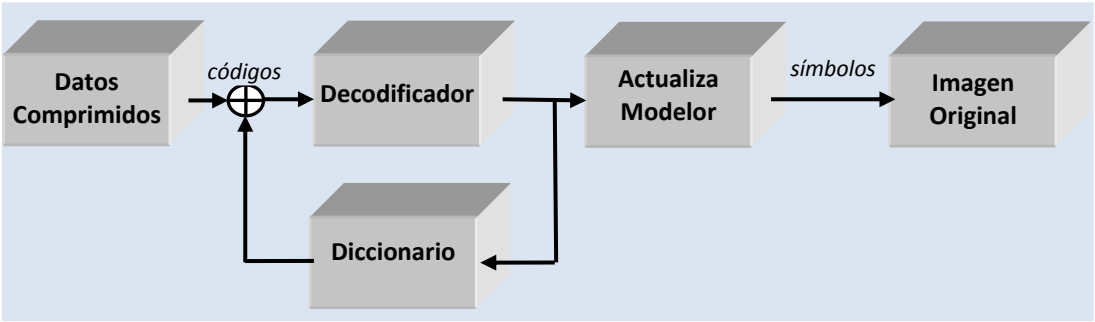


Figura 3.53. Diagrama de bloques del modelo de descompresión LZW

El proceso de descompresión en pasos es el siguiente:

1) Crea e inicializa la tabla de símbolos con sus correspondientes códigos (0 a 255).
2) Lee el primer byte que contiene el código, busca su correspondiente índice en la tabla y escribe el contenido del campo cadena en el fichero de salida. Establece el dato como el primer valor del campo cadena en la tabla.
3) Mientras no se produzca final de fichero comprimido opera del siguiente modo:
    3.1) Lee el siguiente dato y busca su correspondiente índice en la tabla.
    3.2) Escribe la cadena en el fichero de salida.
    3.3) Actualiza la tabla con el nuevo dato.
    3.4) Repite el proceso

3.5.5.7 Algoritmo de descompresión LZW

El algoritmo de descompresión realiza la lectura del fichero comprimido reconstruyendo el diccionario y actualiza el modelo para recuperar los píxeles originales.

Paso 1. Inicializa diccionario (w) con entradas de 0-255. Un byte de profundidad
Paso 2. Lee primer código - k
Paso 3. Escribe símbolo en fichero descomprimido
Paso 4. w=k
Paso 5. Bcule hasta finalizar fichero comprimido
Paso 5.1 Lee siguiente codigo – k
Paso 5.2 cadena = indice diccionario para k
Paso 5.3 Escribe salida = cadena
Paso 5.4 Concatena (w+primer símbolo de la entrada a diccionario)
Paso 5.5 w=cadena
Paso 6. Vuelve al pàso 5
Paso 7. Fichero descomprimido

3.5.5.8 Programación del algoritmo de descompresión LZW

La programación del algoritmo usa el mismo procedimiento y el mismo tipo de operaciones de punteros y de bucles que el algoritmo de compresión. La única diferencia es el fichero de lectura.En el proceso de compresión realiza la lectura de píxeles de la imagen original mientras que en el fichero comprimido procesa bytes que contienen códigos, el resto de la programación es similar.

```
// Algoritmo LZW
//Algoritmo de descompresión
// Inicializa tabla
indice=255;
for( i=0;i<255, i++)
    tabla[i]=i;
    // ciclo del proceso de descompresión
    k= array[x];
    salida[y]=k;
    y++;
    w=k
    for(x=1;x<anchura*anchura;x++)
    {
       for (i=x;i<indice, i++) {
          if ((str(w)+str(k))=tabla(i)){
              salida[y]=tabla[i];
              y++;}
          else
          {    w=str(w)+str(k);
              indice++;
              tabla[indice]=w;
              w=k;}
    }
}
```

Listado 3.13. Programación del algoritmo de descompresión LZW

3.5.5.9 Flujograma del algoritmo de descompresión LZW

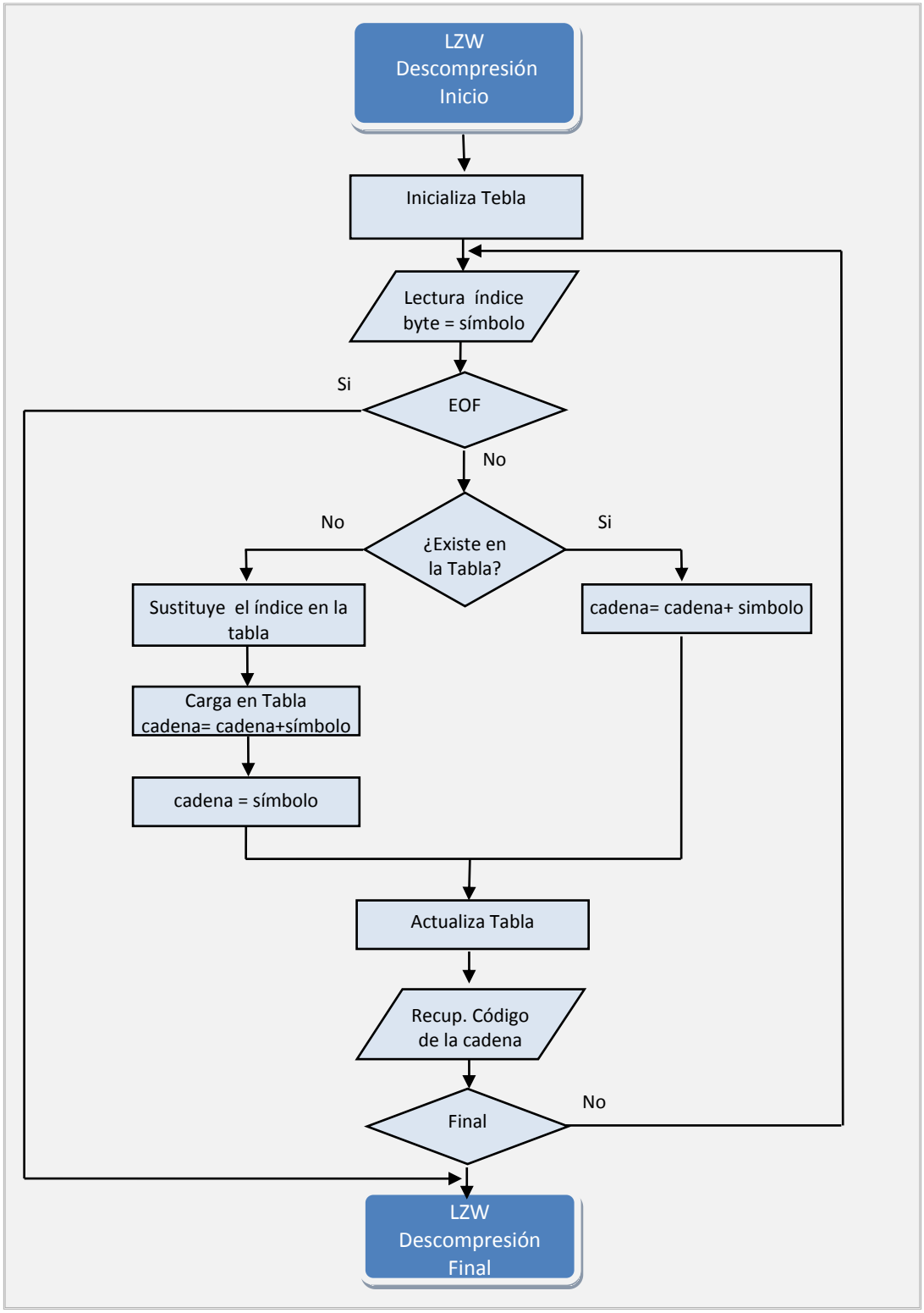


Figura 3.54. Flujograma del algoritmo de descompresión LZW

3.5.5.10 Ejemplo práctico del algoritmo de descompresión LZW

En este ejemplo práctico usamos la salida del codificador propuesto en el ejemplo del proceso de compresión de LZW. La secuencia "121-118-121-118-123-118-121-118-121-118-121-121-121-121-121-121".

El diccionario es reconstruido utilizando cuatro columnas: entrada, índice, diccionario y salida. El diccionario contiene inicialmente los 256 valores de píxeles (0-255) y a continuación un nuevo elemento de secuencia de escape.

Atendiendo a la programación del Listado 3.13 (líneas 12-23) es posible recuperar los valores del código de salida, w, k y el índice. La tabla 3-37 contiene la recuperación de estos valores.

Tabla 3-37. Reconstrucción del diccionario LZW

ÍNDICE	CODIGO	W	K	SALIDA	COMENTARIO
121	121			121	Inicio
118	118	121	118	118	índice < 257, D[257] ← 121-118
257	257	118	121	121-118	índice < 258, D[258] ← 118-121
123	123	257	123	123	índice < 259, D[259] ← 121-118-123
258	258	123	118	118-121	índice < 260, D[260] ← 123-118
261	261	258	118	118-121-118	índice = 261, D[261] ← 118-121-118
121	121	261	121	121	índice < 262, D[262] ← 118-121-118-121
263	263	97	121	121-121	índice = 263, D[263] ← 121-121

Estos métodos son muy utilizados en la práctica porque el balance entre el ratio de compresión y los tiempos de ejecución es aceptable. La principal característica del algoritmo LZW reside en los tiempos de compresión bajos, los cuales no son los tiempos característicos de un modelo adaptativo. Esto es debido a que el algoritmo procesa códigos variables cortos. Estos métodos son utilizados en el formato de archivos como GIF y TIFF y en archivadores.

3.5.6 Resultados experimentales de los métodos de diccionario

La Tabla 3.38 refleja el ratio de compresión obtenidos por los modelos LZ77 y LZW.

Tabla 3-38. Resultados experimentales del método de diccionario LZ77 y LZW

IMAGEN FUENTE	NOMBRE IMAGEN	LZ77 MODELO ESTATICO	LZW MODELO ADAPTATIVO
		RATIO DE COMPRESIÓN	RATIO DE COMPRESIÓN
Fotográficaas	Airplane	1,195	1,090
	Baboon	0,909	0,851
	Barbara	0,955	0,870
	Boats	0,984	0,961
	Cameraman	1,171	0,863
	Goldhill	0,988	0,913
	Lena	0,997	0,984
	Man	1,234	0,959
	Peppers	0,983	0,917
	Zelda	0,987	0,887
	Media	**1,040**	**0,930**
Aéreas y Satélite	Aerial	1,009	0,970
	Airfield	0,962	0,913
	Earth	1,054	0,992
	Meteosat	1,037	0,930
	Moon	1,352	1,171
	Moonsurface	4,499	2,495
	SanDiego	0,978	0,893
	WashingtonIR	0,903	0,789
	Media	**1,474**	**1,144**
Creadas por Ordenador	Circles	7,446	20,766
	Gray	8,172	4,726
	Slope	2,544	1,457
	Squares	7,448	27,653
	Media	**6,403**	**13,650**
Médicas	Elbowx	1,837	1,735
	Finger	0,897	0,819
	MRI_Brain1	1,202	1,198
	MRI_Brain2	1,689	1,501
	MRI_Brain3	1,064	0,976
	Shoulder	1,294	1,044
	Media	1,330	1,212
Textos y Gráficos escaneados	Mercados1	3,186	1,531
	Mercados2	3,366	1,306
	Mercados3	2,372	1,712
	Mercados4	3,841	1,865
	Media	3,191	1,604

Los resultados de la Tabla 3.38 muestran que los métodos basados en modelos de diccionario obtienen ratios de compresión sensiblemente mejores que los modelos de sustitución y que los estadísticos. Los ratios de compresión superan ampliamente a los resultados del Codificador Aritmético en las imágenes creadas por ordenador y en las imágenes que combinan textos y graficos. El método RLE mantiene su supremacía en las imágenes creadas por ordenador (Fig. 3.55).

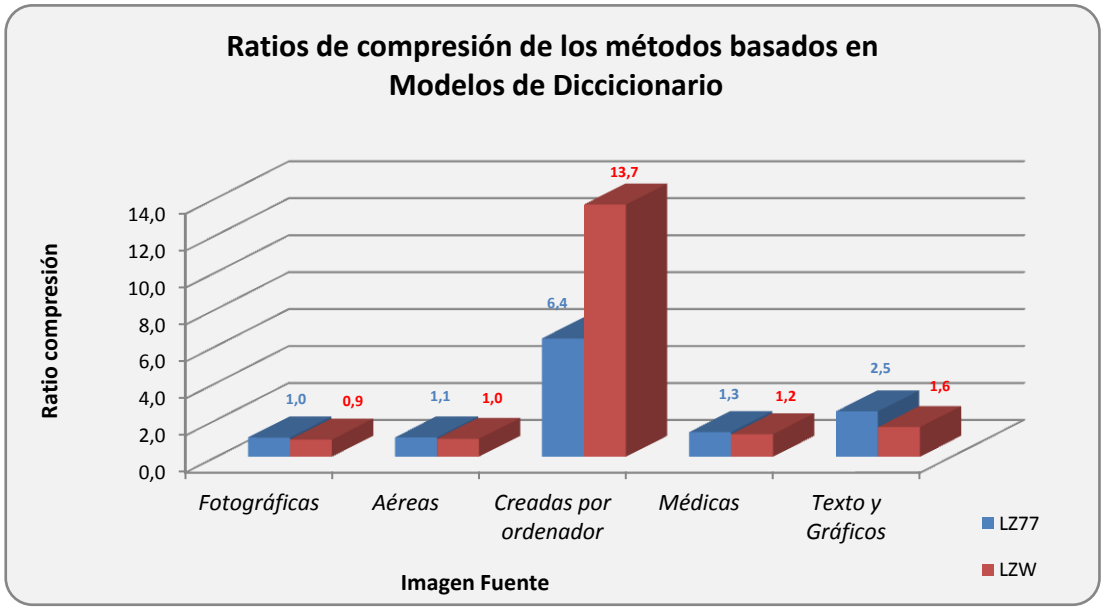


Figura 3.55. Ratios de compresión de los métodos de diccionario LZ77 y LZW

La Fig. 3.55 refleja nitidamente que el modelo adaptativo es más eficiente que el modelo estático en las imágenes creadas por ordenador, mientras que en el resto de las clases de imágenes es menos eficiente.

Con relación a los tiempos de ejecución y ciclos de CPU de los procesos de compresión y descompresión, se confirma que estos tiempos se reducen drásticamente comparados con los métodos revisados anteriormente y aunque los tiempos de ejecución del modelo estático es menor que el de los adaptativos, estos permanecen significativamente mucho menores que los tiempos de compresión de los métodos de sustitución, estadísticos y de Huffman. Las Tablas 3.39 y 3.40 muestran los resultados de ambos modelos.

En términos de consumos de memoria ambos métodos utlizan ventanas de tamaño reducido y una tabla de 256 indices. Los consumos de memoria son bajos y por tanto se ejecutan íntegramente en la memoria RAM sin necesidad de memoria adicional.

Tabla 3-39. LZ77. Tiempos de ejecución del ciclo de compresión y descompresión

IMAGEN FUENTE	NOMBRE IMAGEN	CICLO DE COMPRESIÓN		CICLO DE DESCOMPRESIÓN	
		CICLOS CPU	TIEMPO (ms)	CICLOS CPU	TIEMPO (ms)
Fotográficaas	Airplane	1.171.297	530,37	61.321	27,77
	Baboon	1.316.627	596,18	54.081	24,49
	Barbara	1.186.064	537,06	55.564	25,16
	Boats	1.338.017	605,86	48.473	21,95
	Cameraman	411.370	186,27	53.434	24,20
	Goldhill	13.358.925	6.048,99	45.392	20,55
	Lena	1.187.560	537,73	48.264	21,85
	Man	362.053	163,94	71.844	32,53
	Peppers	1.163.384	526,79	58.764	26,61
	Zelda	1.134.570	513,74	51.659	23,39
	Media	**2.262.987**	**1.024,69**	**54.880**	**24,85**
Aéreas y Satélite	Aerial	1.227.102	555,64	59.792	27,07
	Airfield	1.225.989	555,13	61.820	27,99
	Earth	228.608	103,51	59.653	27,01
	Meteosat	3.810.492	1.725,41	50.703	22,96
	Moon	15.640.969	7.082,31	52.147	23,61
	Moonsurface	1.018.165	461,03	60.905	27,58
	SanDiego	4.779.463	2.164,16	67.088	30,38
	WashingtonIR	21.682.151	9.817,78	56.431	25,55
	Media	**6.201.617**	**2.808,12**	**58.567**	**26,52**
Creadas por Ordenador	Circles	300.113	135,89	48.765	22,08
	Gray	1.290.169	584,19	47.825	21,66
	Slope	410.404	185,83	62.641	28,36
	Squares	370.288	167,67	49.238	22,30
	Media	**592.744**	**268,40**	**52.117**	**23,60**
Médicas	Elbowx	1.251.756	566,80	52.349	23,70
	Finger	345.222	156,32	66.644	30,18
	MRI_Brain1	338.341	153,20	67.753	30,68
	MRI_Brain2	316.689	143,40	66.271	30,01
	MRI_Brain3	562.717	254,80	53.510	24,23
	Shoulder	674.806	305,56	45.965	20,81
	Media	**581.589**	**263,35**	**58.749**	**26,60**
Textos y Gráficos escaneados	Mercados1	4.174.560	1.890,26	51.600	23,36
	Mercados2	4.313.232	1.953,05	65.236	29,54
	Mercados3	3.178.253	1.439,13	54.850	24,84
	Mercados4	1.753.514	794,00	61.085	27,66
	Media	**3.354.890**	**1.519,11**	**58.193**	**26,35**

Tabla 3-40. LZW. Tiempos de ejecución del ciclo de compresión y descompresión

IMAGEN FUENTE	NOMBRE IMAGEN	CICLO DE COMPRESIÓN		CICLO DE DESCOMPRESIÓN	
		CICLOS CPU	TIEMPO (ms)	CICLOS CPU	TIEMPO (ms)
Fotográficaas	Airplane	220.964	100,05	202.596	91,74
	Baboon	229.438	103,89	216.611	98,08
	Barbara	224.950	101,86	203.447	92,12
	Boats	219.964	99,60	196.029	88,76
	Cameraman	98.977	44,82	114.317	51,76
	Goldhill	243.073	110,07	223.416	101,16
	Lena	231.409	104,78	197.650	89,50
	Man	101.415	45,92	111.314	50,40
	Peppers	223.158	101,05	194.485	88,06
	Zelda	222.646	100,82	188.536	85,37
	Media	**201.599**	**91,29**	**184.840**	**83,69**
Aéreas y Satélite	Aerial	225.224	101,98	191.277	86,61
	Airfield	225.068	101,91	199.295	90,24
	Earth	84.613	38,31	84.994	38,49
	Meteosat	492.684	223,09	399.017	180,68
	Moon	2.334.581	1.057,12	2.059.024	932,34
	Moonsurface	159.732	72,33	128.085	58,00
	SanDiego	758.946	343,66	623.220	282,20
	WashingtonIR	3.634.939	1.645,93	2.979.937	1.349,34
	Media	**989.473**	**448,04**	**833.106**	**377,24**
Creadas por Ordenador	Circles	77.085	34,90	77.350	35,02
	Gray	137.126	62,09	115.298	52,21
	Slope	99.887	45,23	93.638	42,40
	Squares	83.229	37,69	68.985	31,24
	Media	**99.332**	**44,98**	**88.818**	**40,22**
Médicas	Elbowx	208.527	94,42	174.488	79,01
	Finger	99.811	45,20	112.294	50,85
	MRI_Brain1	107.444	48,65	94.573	42,82
	MRI_Brain2	79.243	35,88	88.981	40,29
	MRI_Brain3	118.263	53,55	112.570	50,97
	Shoulder	147.073	66,60	131.384	59,49
	Media	**126.727**	**57,38**	**119.048**	**53,91**
Textos y Gráficos escaneados	Mercados1	670.461	303,59	606.890	274,81
	Mercados2	718.899	325,52	542.531	245,66
	Mercados3	538.431	243,81	456.240	206,59
	Mercados4	270.444	122,46	244.193	110,57
	Media	**549.559**	**248,85**	**462.464**	**209,41**

Los modelos de diccionario estáticos y adaptativos se caracterizan por sus bajos tiempos de procesamiento. Sin embargo, el proceso de compresión puede generar expansión de datos en imágenes de continuos tonos. La principal aplicación de este

método se focaliza en imágenes creadas por ordenador e imágenes escaneadas donde la sucesión de blancos es potencialmente compresible por este método.

3.5.7 Conclusiones de los métodos de diccionario

Los ratios de compresión obtenidos por LZ77 son muy aceptables para muchos tipos de imágenes. Los requisitos de memoria son muy bajos para ambos ciclos: compresión y expansión, porque únicamente mantiene un tamaño de ventana en memoria que ocupa un zona de 4 kB a 16 kB. Sin embargo, el proceso de compresión consume demasiado tiempo de procesamiento por las continuas búsquedas y comparaciones entre el diccionario y la cadena de entrada localizada en la zona de datos de entrada.

En términos de recursos de CPU, la búsqueda de la cadena de datos más larga en la entrada de datos y su correspondencia en el diccionario puede ralentizar considerablemente el proceso de compresión. Por otra parte, el tamaño de la ventana es muy limitado y solo contiene los datos procesados recientemente y, por tanto, los datos de las ventanas previas no quedan referenciados. Para evitar este inconveniente, la solución consiste en aumentar el tamaño de la ventana. Como consecuencia, el proceso de búsqueda de correspondencia y los tiempos de CPU se incrementan drásticamente en el proceso de compresión

En términos de tamaño de memoria, en el caso de entrada no compresible o aquellas cadenas sin correspondencia en el diccionario puede suponer expansión en lugar de reducción del fichero de salida.

En términos de transmisión de datos, el principal inconveniente reside en que los métodos basados en modelos estáticos requieren almacenar o transmitir el diccionario junto a los códigos de salida. Esta problemática es simplificada y solucionada utilizando modelos adaptativos. A continuación se analiza el método más relevante de los métodos de diccionario basados en modelos adaptativos: *LZW*

Los modelos adaptativos como LZW son idóneos para aplicaciones de transmisión de datos ya que permiten una visualización sucesiva de los datos en el receptor. Con relación a los inconvenientes, puede suceder que el espacio de diccionario reservado inicialmente no sea suficiente. En este caso disponemos de dos opciones: guardar las entradas más antiguas y usar una tabla *LRU* o adaptar el tamaño de diccionario.

Como alternativa a los métodos tradicionales de compresión revisados en las secciones anteriores de este capítulo, el comité estándar denominado JPEG desarrolló un estándar de compresión de imágenes *lossy* que contenía la primera versión estándar *lossless*. La relevancia e importancia de este método reside en la característica de intento de normalización de la compresión de datos.

Adicionalmente, presentó una importante novedad referida al modelo de compresión. El primer estándar JPEG versión *lossless*, a diferencia de los métodos analizados anteriormente, desarrolló un nuevo modelo basado en la predicción del píxel explotando la propiedad de vecindad de los píxeles. La predicción de cualquier píxel se determina mediante un sistema predictivo en el cual el valor del píxel a procesar es calculado mediante técnicas predictivas en las que intervienen tres píxeles vecinos que han sido procesados anteriormente y por tanto sus valores son conocidos.

Los métodos de compresión basados en modelos predictivos supusieron un gran avance en la compresión de imágnes ya que obtenían mejores ratios de compresión en menores tiempos de ejecución que los analizados anteriormente. En la siguiente sección se analiza el método más representativo de estos modelos y primer estándar de compresión sin pérdida de datos, denominado JPEG *lossless*.

3.6 *JPEG lossless:* primer estándar de compresión reversible de imágenes digitales

JPEG (*Joint Photographic Experts Group*) fue un grupo creado por las organizaciones ISO (*International Organization for Standardization)* y CCITT (*International Telephone y Telegraph Consultative Commitee*) para desarrollar el primer estándar internacional de compresión de imágenes fijas de continuos tonos, en color y escala de grises. El resultado final es un método normalizado de compresión de imágenes fijas denominado con el nombre del propio comité, JPEG *(*ISO/IEC JTC1/SC29/WG, 1991). El comité recomendó el uso de un nuevo formato de fichero de imágenes, *JPG*, para normalizar el almacenamiento de imágenes comprimidas mediante la aplicación de estándar *JPEG* para facilitar el intercambio de imágenes entre diferentes formatos de ficheros gráficos.

Este estándar es un método de compresión de imágenes *lossy* que combina técnicas sin pérdida de datos (*lossless*) con técnicas de pérdida de datos voluntarias sin sacrificar la fidelidad a la imagen original. Este algoritmo obtiene unos ratios de compresión con pérdida de datos de 10:1 a 50:1 sin afectar visiblemente a la calidad de la imagen y de 2:1 en modo *lossless*.

El propósito del comité consistió en establecer un estándar de compresión de imágenes fijas de continuos tonos, color y escala de grises con pérdida de datos. Para conseguir este objetivo el estándar debía cumplir las siguientes condiciones:

- Estar cerca del estado-del-arte, entendido como un ratio 2:1, de la compresión de imágenes sin afectar a la calidad de la imagen.
- Ser aplicable a prácticamente cualquier clase de imagen fuente digital de continuos tonos sin restricción de tamaño, color, píxeles etc.

- Ser independiente del hardware, facilitando el desarrollo del software para diferentes CPUs y componentes de hardware con bajos requisitos.
- Establecer directrices encaminadas a futuros posibles cambios o variaciones en función de los avances tecnológicos.

Una vez cumplido los requisitos imprescindibles, JPEG fue designado como método *lossy* estándar internacional en 1991. Sin embargo, la versión extendida sin pérdida de datos que estába cerca del estado del arte no ha tenido la misma aceptación y repercusión que la versión *lossy*. El estándar JPEG actualmente dispone de al menos 29 diferentes sistemas de compresión de datos (Hudson, Yasuda y Sebestyen, 1988).

3.6.1 Descripción del método

El estándar JPEG en su versión sin pérdida de datos consiste en un método basado en un modelo de predicción del píxel, y para la codificación del valor predicho recomienda el uso de una de la dos posibles alternativas: codificador estático Huffman o el Codificador Aritmético. La complejidad de cálculo de la codificación aritmética en términos de recursos de *CPU* es más costosa que la del Codificador Huffman. Sin embargo, en términos de almacenamiento, la codificación aritmética reduce el tamaño entre un 5% y un 10% más que el Aritmético.

La idea básica consiste en reducir los valores de los píxeles antes de ser comprimidos. Para conseguir este propósito incorpora un nuevo concepto en el modelo que difiere de forma notable de los modelos usados en los métodos revisados anteriormente, esto es conocido como predicción del píxel. Explotando la relación de vecindad de los píxeles y los pequeños cambios de intensidad de color que presentan los píxeles adyacentes, es posible predecir el píxel actual y reducir su valor teniendo en cuenta el contexto de este píxel. El algoritmo calcula el error residual de predicción, diferencia entre el valor predecido y su valor real. El resultado es un conjunto de valores reducidos para cada contexto. Estos valores residuales son codificados mediante la codificación aritmética o codificación *Huffman* atendiendo a las recomendaciones del comité.

En la Fig. 3.56 se ilustra el diagrama de bloques simplifado del estándar JPEG en la versión reversible (Leger, Omachi y Wallace, 1991).

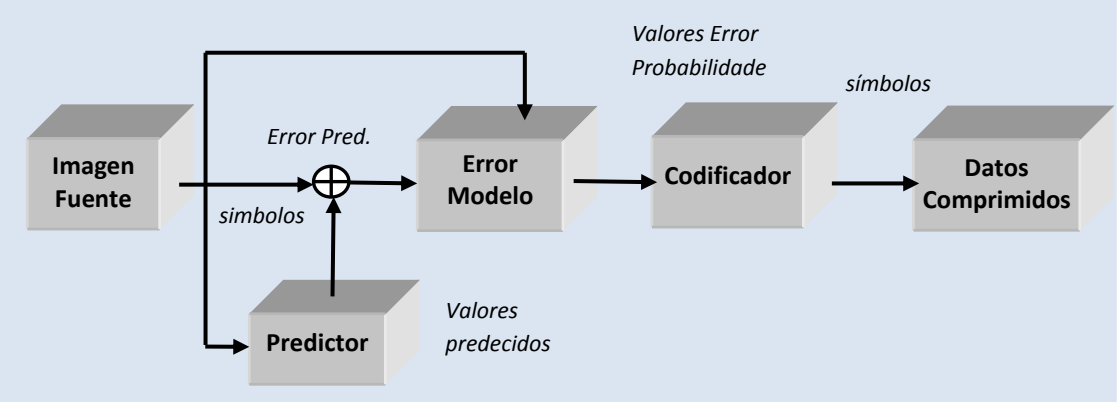


Figura 3.56. Diagrama de bloques del estándar JPEG lossless

El modo de operación sin pérdida de datos es una versión extendida e independiente del popular modo JPEG con pérdida de datos. Una de las características más relevante reside en la simplicidad del método lineal predictivo, el cual es independiente de la DCT (*Discrete Cosine Transform*) que constuituye el núcleo de la versión irreversible.

El ratio de compresión conseguido por el algoritmo JPEG sin pérdida de datos en imágenes de continuos cambios es cercano a la relación 2:1. Este ratio de compresión no supone un gran avance en el estado-del-arte de compresión de imágenes estáticas aunque obtiene mejores ratios que los métodos revisados anteriormente en este capítulo.

En las siguientes secciones se aborda la revisión del estándar en su versión sin pérdida de datos mediante la misma estructura de análisis y evaluación aplicada en los métodos anteriores.

3.6.2 Proceso de compresión

El ciclo de compresión define en un primer paso el contexto del píxel a predecir. Este contexto está formado por los tres píxeles vecinos al píxel a predecir. En un segundo paso selecciona un predictor de los ochos posibles preestablecidos por el estándar para obtener una primera predicción del píxel. En un tercer paso, calcula el error residual o diferencia entre el valor del píxel y el valor de predicción obtenido. Por último, los valores de errores residuales son codificados mediante una codificación aritmética (Howard y Vitter, 1992) o mediante la codificación estática de Huffman.

En la Fig. 3.57 se ilustra el proceso de codificación mediante diagrama de bloques simplifcado.

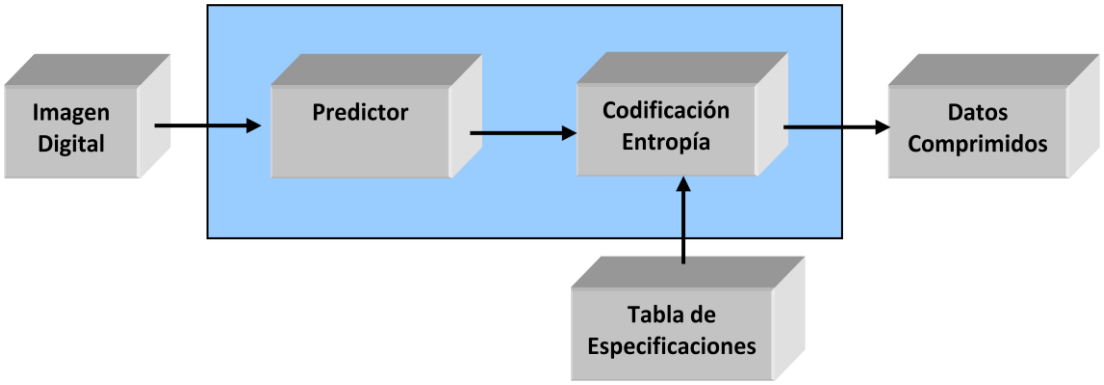


Figura 3.57. Diagrama de bloques del proceso de compresión

La imagen es procesada píxel a píxel desde la fila superior a la inferior de la imagen y de izquierda a derecha hasta completar el tratamiento de todos los píxeles. La unidad de tratamiento es el píxel en relación con su contexto.

Inicialmente, el algoritmo de compresión define el contexto de píxel x a predecir (Fig. 3.58). Este contexto está formado por tres píxeles (a,b,c) adyacentes al píxel.

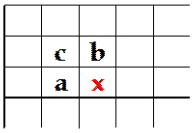


Figura 3.58. Contexto del píxel a predecir

Una imagen cualquiera presenta dos casos especiales en la formación del contexto. Un primer caso se presenta en la inexistencia del contexto de la fila superior de la imagen. Para resolver esta problemática, el modelo asigna asigna valor cero a los píxeles del contexto "c" y "b". Un segundo caso se produce en todos los primeros píxeles de cada fila, los valores del contexto "a" y "c" no son conocidos. En este caso, el modelo asignarles valor cero o el valor de "b".

Una vez definido el contexto para cada píxel a predecir, el estándar define ocho posibles predictores. La técnica de predicción es variable con respecto al predictor seleccionado. El usuario no puede seleccionar el tipo de predictor. El algoritmo selecciona el predictor que mejor se ajusta al contexto de cada píxel. En la Tabla 3.41 se reflejan las ocho técnicas de predicción disponibles en el estándar (Wallace, 1991).

Tabla 3-41. Técnica de predicción del estándar JPEG *lossless*

PREDICTOR	PREDICCIÓN
0	Sin predicción
1	Px=a
2	Px=b
3	Px=c
4	Px=a+b+c
5	Px=a+((b-c)/2)
6	Px=b+((a-c)/2)
7	Px= (a+b)/2

Los modos de predicción "1", "2" y "3" son predictores unidimensionales mientras que los siguientes, "4", "5", "6", "7" son predictores bidimensionales. El predictor "0" indica que no existe predicción, por tanto, el código de salida será el propio valor del píxel actual.

Una vez obtenida la predicción del símbolo *Px*, el algoritmo calcula el error de predicción residual. El error residual en el codificador es calculado utilizando la expresión.

$$\epsilon = P_x - x \tag{3.29}$$

El error residual es la diferencia de los valores de los píxeles de la imagen fuente y sus respectivos valores de predicción. El resultado es una serie de números positivos y negativos cercanos a cero que representan el error de predicción. La probabilidad de los valores cercanos a cero o de pequeños cambios es grande mientras que los valores mayores tienen probabilidades menores.

En este punto, los datos están preparados para ser codificados con métodos de compresión universales. El estándar especifica dos posibles métodos de codificación: Codificador Aritmético o Codificador Huffman. Obviamente, solo uno de los dos métodos es utilizado para el procesamiento de una imagen. El método de codificación en el proceso de compresión y descompresión debe ser el mismo y se especifica en un campo de la cabecera del fichero comprimido. En las siguientes secciones se describen ambos métodos aplicados en el estándar.

3.6.2.1 Codificador Huffman

La codificación mediante el Codificador Huffman se realiza de forma similar al propuesto en la versión irreversible del estándar. Esto es, aplica el mismo método de codificación tanto en la versión *lossless* como en *lossy*. La versión irreversible codifica los coeficientes obtenidos por la Transformada Discreta del Coseno. La función de esta transformada es generar coeficientes con valores cero o cercanos al cero.

Entonces estos valores se codifican utilizando uno de los dos modelos probabilísticos: Codificador Huffman o Codificador Aritmético.

El Codificador Huffman incorpora una tabla normalizada (Tabla 3.42). Esta tabla agrupa los valores residuales en clases de *categorías* a las que asigna una longitud de código denominada *amplitud*. La tabla está organizada para soportar una precisión de 2^2 hasta 2^{16} dependiendo del número de bits utilizados por píxel.

Tabla 3-42. Tabla de códigos de Huffman DC (Tabla K.3 del estándar).

CATEGORÍA	LONGITUD	CÓDIGO
0	2	00
1	3	010
2	3	011
3	3	100
4	3	101
5	3	110
6	4	1110
7	5	11110
8	6	111110
9	7	1111110
10	8	11111110
11	9	111111110
12	10	1111111110
13	11	11111111110
14	12	111111111110
15	13	1111111111110
16	14	11111111111110

La columna *categoría* indica el número de bits usados para representar la *amplitud* del valor residual y es expresada mediante una codificación binaria.

Tabla 3-43. Tabla de categorías y diferencias usada por el Codificador Huffman

CATEGORÍA	DIFERENCIAS	AMPLITUD BITS - EXTRA
0	0	-
1	-1, 1	0, 1
2	-3..-2 , 2, 3	00, 01, 10, 11
3	-7..-4, 4..7	000,...011, 100...111
4	-15..-8, 8..15	0000,...0111, 1000...1111
5	-31..-16, 16..31	00000,...,01111,10000,...,11111
6	-63..-32, 32..63	000000,...,011111,100000,...,111111

Tabla 3.43. Tabla de categorías y diferencias usada por el Codificador Huffman (cont)

CATEGORÍA	DIFERENCIAS	AMPLITUD BITS- EXTRA
7	-127..-64, 64..127	0000000,...,0111111,1000000,...,1111111
8	-255..-128, 128..255	:
9	-511..256, 256..511	:
10	-1023..-512, 512..1023	:
11	-2047..-1024, 1024..2047	:
12	-4095..-2048, 2048..4095	:
13	-8191..-4096, 4096..8191	:
14	-16383..-8192, 8192..16383	:
15	-32767..-16384, 16384..32767	:
16	32768	1000000000000000

Esto es, si el error residual es 64, el tamaño de la categoría es "7" y la secuencia binaria para una amplitud de "7" es *1000000*. En consecuencia, el símbolo del tamaño de la categoría es codificado.

3.6.2.2 Algoritmo de compresión

El algoritmo de codificación es el siguiente:

Paso 1. Encontrar el valor de ε *en la tercera columna de tabla de diferencias*
Paso 2. Buscar en la tabla la categoría que le corresponde a ε
Paso 3. Codificar la magnitud del error residual.
Paso 4. Si $\varepsilon>0$*, entonces:*
  Paso 4.1 Codificar en binario ε *con la longitud de bits que le corresponde a la categoría*
  Paso 4.2 El bit más significativo siempre debe ser 1.
Paso 5. Si $\varepsilon<=0$*, entonces:*
  Paso 5.1 Codificar en binario $\varepsilon-1$ *con la longitud de bits que le corresponde a la categoría*
  Paso 5.2 El bit más significativo siempre debe ser 0.
Paso 6. Resultado: secuencia de bits

Por ejemplo, supongamos un primer caso en el cual el error residual es positivo $\varepsilon=4$. La categoría que le corresponde es igual a 3 (Tabla 3.43), el código que le corresponde a esa categoría en la Tabla 3.42 es "*100*", y a continuación a este código base se le añaden los bits más significativos de la codificación binaria del valor de ε con tres bits de longitud, esto es 4 en binario, "100". El resultado es la secuencia de bits "100100". En caso de valores negativos, por ejemplo, si $\varepsilon=-4$, entonces la categoría en sigue siendo la misma 3, y su código binario "100", a estos bits hay que añadir los tres bits menos significativos de [–4-1=-5) que en binario aplicando

complemento a uno es igual a "010". El código resultante es la secuencia de bits "100010".

3.6.2.3 Codificador Aritmético

El modelo de Codificador Aritmético utiliza una tabla estática de dos dimensiones en la cual el valor de error a codificar está condicionado por las diferencias de los valores de la izquierda y por los de la fila superior. Las diferencias están clasificadas en cinco categorías (Tabla 3.44).

Tabla 3-44. Categorías del Codificador Aritmético

CATEGORÍA	DIFERENCIAS
1	(0) - cero
2	(+S) - diferencias pequeñas positivas
3	(-S) - diferencias pequeñas negativas
4	(+L) - diferencias grandes positivas
5	(-L) - diferencias grandes negativas

La combinación de ambas categorías proporciona 25 diferentes combinaciones. En la Fig. 3.59 se muestra el primer array bidimensional de 5x5 para los 100 primeros bits. Este primer grupo es etiquetado como S0. Los valores de S0 se obtienen del contexto, Contexto (Da, Db). que proporciona una valor de 0, 4, ..., 92 o 96, dependiendo de las clasificaciones de diferencias del contexto (Da, Db).

Diferencia superior – posición b

Diferencia izquierda posicición - a

	0	**(+S)**	**(-S)**	**(+L)**	**(-L)**
0	0	4	8	12	16
(+S)	20	24	28	32	36
(-S)	40	44	48	52	56
(+L)	60	64	68	72	76
(-L)	80	84	88	92	96

Figura 3.59. Array de dos dimensiones del modelo arirtmético

El número total de índices de contexto contenidos en la tabla estándar del Codificador Aritmético (Tabla 3.45) es 158. Los primeros 100 indices apuntan a los 25 conjuntos de cuatro contenedores seleccionados por un S0 contexto-índice como se ha indicado anteriormente. Los 58 indices de contexto restantes consisten en dos

grupos de 29 contenedores etiquetados como x1, ..., x15, M2, ..., M15 para codificar la categoría y la magnitud.

Tabla 3-45. Codificación de las categorías y magnitudes

INDICE DEL CONTEXTO	VALOR	DECISIÓN DE CODIFICACIÓN
S0	L_Contexto(Da,Db)	V = 0
SS	S0 + 1	Signo
SP	S0 + 2	Sz < 1 if V > 0
SN	S0 + 3	Sz < 1 if V < 0
X1	X1_Contexto(Db)	Sz < 2
X2	X1 + 1	Sz < 4
X3	X1 + 2	Sz < 8
.	.	.
.	.	.
X15	X1 + 14	Sz < 215
M2	X2 + 14	Magnitud de bits, si Sz < 4
M3	X3 + 14	Magnitud de bits, si Sz < 8
.	.	.
.	.	.
M15	X15 + 14	Magnitud de bits, si Sz < 215

Estas diferencias se codifican mediante el paso de la asignación estadística de los valores a la codificación de árbol binario mostrado en la Fig. 3.60.

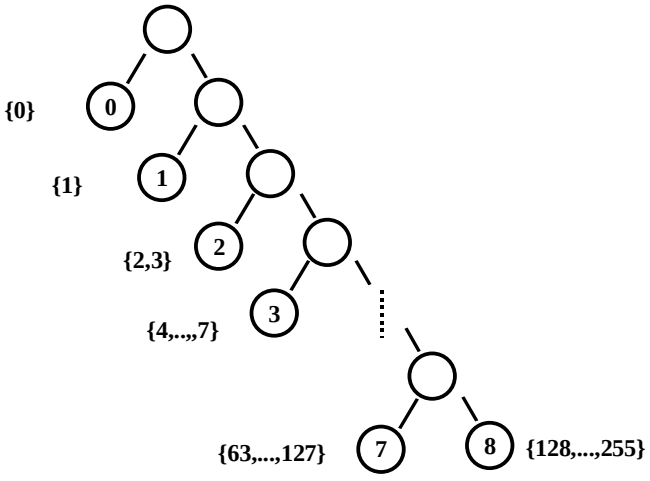


Figura 3.60. Árbol binario del Codificador Aritmético

El Codificador Aritmético utiliza menos bits para la codificación que el Codificador Huffman, aproximadamente un 10%. Sin embargo, este codificador es más lento porque emplea más recursos de cálculo. Por este motivo y por usos de patentes la opción más utilizada en implementaciones software es la de Huffman. Por el

contrario, cuando la implementación se hace a nivel hardware, la versión del Codificador Aritmético es la más usada por su sencillez de implementación y porque los tiempos de ejecución se reducen considerablemente.

3.6.2.4 Ejemplo práctico

Dada una secuencia de píxeles de una imagen cualquiera se realiza la compresión sin pérdida de datos aplicando el estándar JPEG en su versión sin pérdida de datos usando como técnica de codificación el codificador estático de Huffman.

Píxeles: {15, 11, 8, 10, 6, 6, 2}

El estándar utiliza un predictor para el contexto de cada píxel y calcula el error residual. En este ejemplo el predictor utlizado es el "1" porque Px=a.

Tabla 3-46. Predictor y errores residuales de los píxeles

PREDICTORES							
Píxel	15	11	8	10	6	6	2
Predictor	1	1	1	1	1	1	1
ε	15	-4	-3	2	-4	0	4

A continuación, extrae la categoría de cada error residual accediendo a la tabla de categorías, Tabla 3.47, y coloca su correspondiente símbolo binario.

Tabla 3-47. Categoría de los errors residuales

ERRORES RESIDUALES							
Píxel	15	11	8	10	6	6	2
Predictor	1	1	1	1	1	1	1
ε	15	-4	-3	2	-4	0	4
Categoría	4	3	2	2	3	0	3
Código	101	100	011	011	100	0	100

Por último, añade el código de bits extras de cada valor residual accediendo a la tabla de categorías y diferencias Tabla 3.48 generando de esta forma los códigos de salida.

Tabla 3-48. Bits extras de los errores residuales

BIT EXTRAS							
Píxel	15	11	8	10	6	6	2
Predictor	1	1	1	1	1	1	1
ε	15	-4	-3	2	-4	0	4
Categoría	4	3	2	2	3	0	3
Código	101 1111	100 011	011 00	011 10	100 011	0	100 1001

El proceso es simple y rápido debido a la ejecución de operaciones de búsqueda mediante punteros y operaciones de desplazamiento de bits. A continuación se describe el proceso de descompresión que consiste en realizar las operaciones en orden a como las realiza el codificador.

3.6.3 Proceso de descompresión

El ciclo de descompresión es totalmente reversible. JPEG *lossless* utiliza un predictor estático y esto significa que el modelo de predicción es definido antes de comenzar la compresión.

La Fig. 3.61 muestra el diagrama de bloques del proceso de descompresión

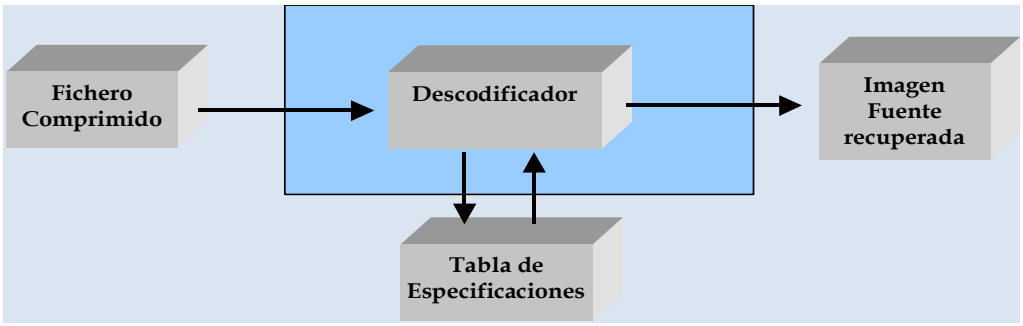


Figura 3.61. Proceso de descompresión del estándar JPEG lossless

El proceso consiste en calcular el error residual ε utilizando el mismo predictor que el compresor, decodificar ε aplicando la misma técnica que el compresor para obtener la predicción del píxel a descomprimir *Px*. El valor del píxel es recuperado aplicando la operación inversa del compresor.

$$x = Px + \varepsilon \tag{3.30}$$

El fichero comprimido contiene un campo que identifica el tipo de codificador utilizado en el proceso de compresión: Codificador Aritmético o Codificador Huffman. Una vez conocido el tipo de codificador utilizado en el proceso de compresión carga la tabla de especificaciones correspondiente aplicando las mismas reglas que el algoritmo de compresión.

3.6.3.1 Algoritmo de descompresión

El ciclo de descompresión utiliza las mismas tablas que el proceso de codificación. El código de salida está compuesto por el código binario de la categoría y el código binario de diferencias.

El algoritmo de descompresión se basa en los siguientes pasos:

Paso 1. Decodifica la categoría comparando con la tabla y obtiene su código
Paso 2. Analiza el siguiente bit de entrada – S_{bit}

Paso 3. Si S_{bit}*<>0*
Paso 3.1 nbits= S_{bit} *x* $2^{categoria-1}$ *+ (siguientes – 1 bits)*
Paso 3.2 Operación SHIFT de nbits a la izquierda
Paso 4. Si S_{bit}*=0*
Paso 4.1 ε*=* ε*-1; codigo*S_{bit} *x* $2^{categoria-1}$ *+(siguientes – 1 bits +1)*
Paso 5. Resultado: secuencia de bits

3.6.3.2 Ejemplo práctico

Aplicando el mismo ejemplo propuesto en el proceso de codificación, el código del fichero comprimido es el siguiente:

Código	**101** 1111	**100** 011	**011** 00	**011** 10	**100** 011	**0**	**100**

El primer código de entrada corresponde a la secuencia de bits 1011111. Los tres bits más significativos indican la clase de *categoría*, y los siguientes el valor de las diferencias o *amplitud* codificada según los bits extras en cada categoría. El bit que aparece después de la categoría es un uno. Este bit es desplazado a la izquierda y se incorpora el nuevo bit por la derecha repitiéndose el tratamiento de desplazamiento hasta finalizar el código que determina el valor del error residual.

Tabla 3-49. Cálculo del error residual

ERROR RESIDUAL							
Código	**101** 1111	**100** 011	**011** 00	**011** 10	**100** 011	0	**100** 1001
Categoría	4	3	2	2	3	0	3
ε	15	-4	-3	2	-4	0	4

Una vez calculado el error residual se recupera el valor de cada píxel original mediante la operación inversa aplicada en el codificador y con el mismo predictor, $x = Px + \varepsilon$. El resultado se muestra en la Tabla 3.50.

Tabla 3-50. Recuperación del valor original de cada píxel

VALOR DELOS PÍXELES							
Código	**101** 1111	**100** 011	**011** 00	**011** 10	**100** 011	0	**100** 1001
Categoría	4	3	2	2	3	0	3
ε	15	-4	-3	2	-4	0	4
Predictor	1	1	1	1	1	1	1
Píxel	15	11	8	10	6	6	2

3.6.4 Resultados experimentales

Los resultados experimentales del primer estándar en sus versiones del Codificador Aritmético y de Huffman se muestran en la Tabla 3.51.

Tabla 3-51. Resultados experimentales de JPEG modo *lossless*

IMAGEN FUENTE	NOMBRE IMAGEN	JPEG MODO LOSSLESS CODIFICADOR HUFFMAN	JPEG MODO LOSSLESS CODIFICADOR ARITMÉTICO
		RATIO DE COMPRESIÓN	RATIO DE COMPRESIÓN
Fotográficaas	Airplane	1,741	1,829
	Baboon	1,238	1,257
	Barbara	1,519	1,572
	Boats	1,511	1,550
	Cameraman	1,532	1,592
	Goldhill	1,512	1,567
	Lena	1,686	1,743
	Man	2,149	2,332
	Peppers	1,609	1,644
	Zelda	1,814	1,904
	Media	**1,631**	**1,699**
Aéreas y Satélite	Aerial	1,355	1,387
	Airfield	1,303	1,309
	Earth	1,385	1,424
	Meteosat	2,107	2,194
	Moon	2,381	2,602
	Moonsurface	1,472	1,475
	SanDiego	1,284	1,330
	WashingtonIR	1,153	1,155
	Media	**1,555**	**1,609**
Creadas por Ordenador	Circles	6,276	6,620
	Gray	35,607	51,467
	Slope	3,894	4,192
	Squares	93,529	269,811
	Media	**34,826**	**83,022**
Médicas	Elbowx	3,350	3,657
	Finger	1,147	1,124
	MRI_Brain1	1,787	1,835
	MRI_Brain2	2,825	2,927
	MRI_Brain3	1,576	1,638
	Shoulder	2,310	2,429
	Media	**2,166**	**2,268**
Textos y Gráficos escaneados	Mercados1	1,974	1,919
	Mercados2	1,672	1,605
	Mercados3	2,233	2,253
	Mercados4	2,035	1,975
	Media	**1,978**	**1,938**

Los resultados de la Tabla 3.51 reflejan que los modelos predictivos, en este caso, el estándar JPEG modo *lossless* consiguen mejores ratios de compresión en todos los tipos de imágenes que los modelos revisados anteriormente. En el caso de imágenes creadas por ordenador es especialmente significativo el ratio de compresión, especialmente la imagen "squares" que es reducidad de 65536 bytes a 243 bytes. El estándar en modo *lossless* explota la propiedad de vecindad de los píxeles de una imagen. En este caso, la mayoría de los píxeles son iguales al anterior por lo que utiliza el predictor 0, generando de esta forma una compresión máxima. Por oto lado, se demuestra un salto significativo en los ratios de compresión del resto de imágenes fuente cuyos ratios oscilan en un rango de valores de [1,5 – 3,7].

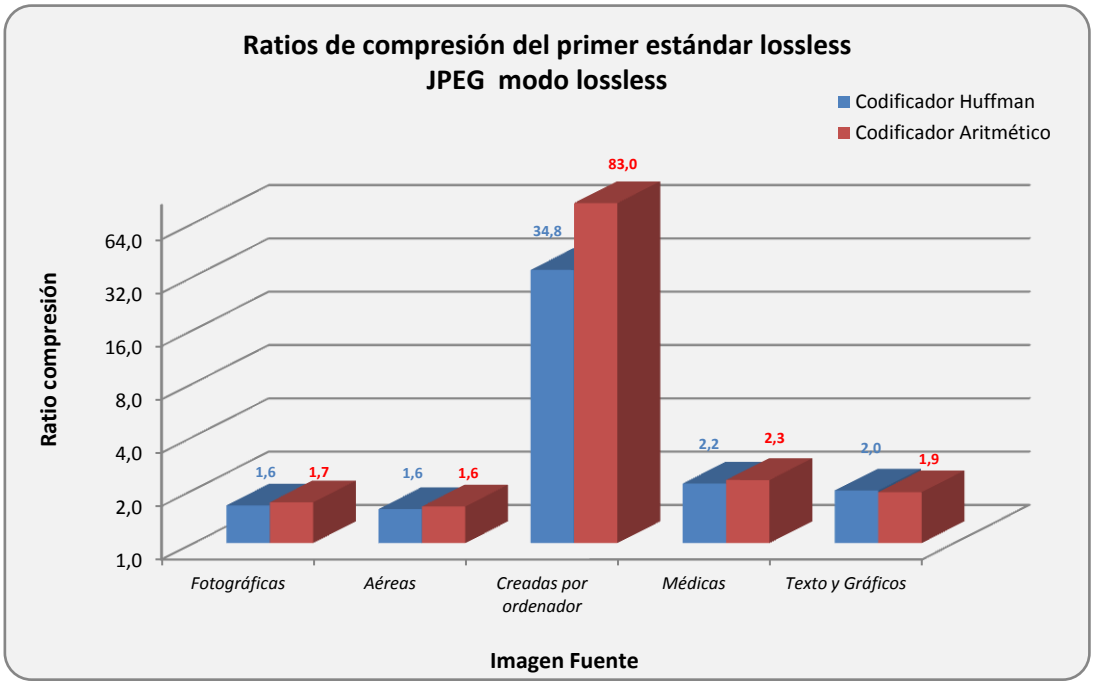


Figura 3.62. Gráfico comparativo de los ratios de compresión (rango 0-80)

En la Fig. 3.62 se observa que el estándar en la versión que utiliza el Codificador Aritmético obtiene mejores resultados que el Codificador Huffman. Sin embargo, el Codificador Huffman es más ampliamente utilizado por las razones que se apuntaron en los inconvenientes del uso del Codificador Aritmético.

En términos de consumos de memoria, el algoritmo almacena en memoria el contexto de los tres píxeles adyacentes al píxel a predecir. Esto significa que los consumos de memoria no son significativos. Sin embargo, el método estándar aconseja la utilización de un Codificador Aritmético o Huffman. Esto implica que el método requiere almacenar las tabla estándares de codificación más las tablas de probabilidades, rangos o árbol binario en base al codificador seleccionado.

La Fig. 3.63 muestra el excelente resultado de la compresión de imágenes creadas por ordenador que alcanza el máximo ratio entre todos los métodos lossless evaluados en esta tesis.

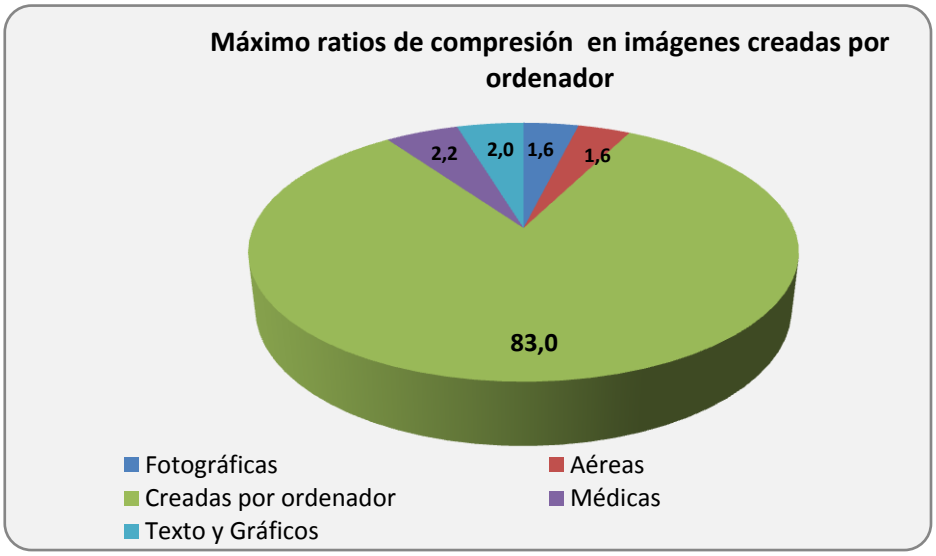


Figura 3.63. Máxima compresión de JPEG lossless para imágenes creadas por ordenador

El ratio de compresión obtenido por JPEG *lossless* en imágenes de continuos cambios se aproxima a la relación 2:1. Este ratio de compresión es superado por algunos métodos convencionales mencionados en este capitulo. Por este motivo, el comité JPEG desarrolló otro estándar para compresión de imágenes de continuos tonos denominado JPEG-LS. A pesar de este nuevo estándar, sigue existiendo un vació en estándares *lossless* que no ha sido cubierto por JPEG-LS (ISO/IEC 14495, 2000) ni por JPEG 2000, ya que ambos estándares no han tenido la aceptación esperada por los usuarios.

Las Tablas 3.52 y 3.53 muestras los tiempos de ejecución de los ciclos de compresión y descompresión del JPEG. El contenido de estas tablas demuestran que los resultados experimentales no siguen una misma tendencia de resultados en cada tipo de codififcador. Esto es:

- El método JPEG *lossless* usando el Codificador Huffman obtiene los mejores tiempos de ejecución en el proceso de compresión y descompresión.
- El método JPEG *lossless* usando el Codificador Aritmético obtiene mejores tiempos de ejecución en el proceso de compresión que en el de descompresión.
- Los tiempos de ejcución son muy aceptables y solamente superados por el método LZ77 aunque el balance entre ratios y tiempos es muy superior en el estándar.

Tabla 3-52. JPEG *lossless* Aritmético. Tiempos de ejecución del ciclo de compresión y descompresión

IMAGEN FUENTE	NOMBRE IMAGEN	CICLO DE COMPRESIÓN		CICLO DE DESCOMPRESIÓN	
		CICLOS CPU	TIEMPO (ms)	CICLOS CPU	TIEMPO (ms)
Fotográficaas	Airplane	186.805	84,59	291.108	131,82
	Baboon	216.650	98,10	248.917	112,71
	Barbara	203.376	92,09	307.783	139,37
	Boats	190.834	86,41	263.737	119,42
	Cameraman	108.018	48,91	116.115	52,58
	Goldhill	198.766	90,00	217.939	98,68
	Lena	181.363	82,12	249.288	112,88
	Man	91.811	41,57	106.437	48,20
	Peppers	186.503	84,45	230.442	104,35
	Zelda	222.740	100,86	234.381	106,13
	Media	**178.687**	**80,91**	**226.615**	**102,61**
Aéreas y Satélite	Aerial	214.281	97,03	228.622	103,52
	Airfield	217.045	98,28	239.160	108,29
	Earth	83.862	37,97	86.293	39,07
	Meteosat	334.503	151,46	400.538	181,37
	Moon	1.567.686	709,86	1.759.837	796,86
	Moonsurface	172.165	77,96	220.923	100,04
	SanDiego	724.300	327,97	743.104	336,48
	WashingtonIR	3.284.995	1.487,46	3.731.365	1.689,58
	Media	**824.855**	**373,50**	**926.230**	**419,40**
Creadas por Ordenador	Circles	75.640	34,25	81.885	37,08
	Gray	119.949	54,31	122.161	55,32
	Slope	77.861	35,26	155.327	70,33
	Squares	82.338	37,28	106.418	48,19
	Media	**88.947**	**40,28**	**116.448**	**52,73**
Médicas	Elbowx	133.400	60,40	178.931	81,02
	Finger	99.035	44,84	122.981	55,69
	MRI_Brain1	130.390	59,04	92.067	41,69
	MRI_Brain2	91.724	41,53	97.292	44,05
	MRI_Brain3	166.732	75,50	111.461	50,47
	Shoulder	122.172	55,32	133.943	60,65
	Media	**123.909**	**56,11**	**122.779**	**55,60**
Textos y Gráficos escaneados	Mercados1	492.934	223,20	565.279	255,96
	Mercados2	532.939	241,32	646.829	292,89
	Mercados3	358.991	162,55	455.299	206,16
	Mercados4	235.343	106,56	299.216	135,49
	Media	**405.052**	**183,41**	**491.656**	**222,63**

Tabla 3-53. JPEG *lossless* Huffman. Tiempos de ejecución del ciclo de compresión y descompresión

IMAGEN FUENTE	NOMBRE IMAGEN	CICLO DE COMPRESIÓN		CICLO DE DESCOMPRESIÓN	
		CICLOS CPU	TIEMPO (ms)	CICLOS CPU	TIEMPO (ms)
Fotográficaas	Airplane	176.648	79,99	111.885	50,66
	Baboon	194.633	88,13	122.471	55,46
	Barbara	165.010	74,72	114.337	51,77
	Boats	163.033	73,82	115.857	52,46
	Cameraman	120.184	54,42	84.281	38,16
	Goldhill	264.391	119,72	145.900	66,07
	Lena	226.732	102,67	103.025	46,65
	Man	114.167	51,70	75.877	34,36
	Peppers	176.056	79,72	127.105	57,55
	Zelda	174.501	79,02	122.293	55,38
	Media	**177.536**	**80,39**	**112.303**	**50,85**
Aéreas y Satélite	Aerial	294.321	133,27	263.434	119,29
	Airfield	222.882	100,92	146.931	66,53
	Earth	96.522	43,71	69.969	31,68
	Meteosat	291.797	132,13	203.001	91,92
	Moon	1.163.718	526,95	707.105	320,19
	Moonsurface	169.653	76,82	124.034	56,16
	SanDiego	422.309	191,23	275.314	124,67
	WashingtonIR	1.737.041	786,55	886.966	401,63
	Media	**549.780**	**248,95**	**334.594**	**151,51**
Creadas por Ordenador	Circles	107.255	48,57	103.386	46,81
	Gray	159.966	72,43	94.058	42,59
	Slope	142.319	64,44	85.928	38,91
	Squares	76.256	34,53	111.308	50,40
	Media	**121.449**	**54,99**	**98.670**	**44,68**
Médicas	Elbowx	206.293	93,41	91.007	41,21
	Finger	116.625	52,81	77.726	35,20
	MRI_Brain1	114.864	52,01	102.255	46,30
	MRI_Brain2	116.924	52,94	86.886	39,34
	MRI_Brain3	142.951	64,73	98.394	44,55
	Shoulder	132.997	60,22	108.627	49,19
	Media	**138.442**	**62,69**	**94.149**	**42,63**
Textos y Gráficos escaneados	Mercados1	340.204	154,05	253.531	114,80
	Mercados2	375.935	170,23	241.767	109,48
	Mercados3	302.660	137,05	203.639	92,21
	Mercados4	204.200	92,46	131.639	59,61
	Media	**203.833**	**92,30**	**138.429**	**62,68**

3.6.5 Conclusiones del primer estándar: JPEG *lossless*

El estándar JPEG *lossless* de compresión de imágenes de continuos tonos en color o escala de grises no es la panacea en compresión de imágenes pero sí facilita que muchas aplicaciones integren este algoritmo en el campo del procesamiento de imágenes digitales. Sin embargo, la normalización también mantiene la propiedad biunívoca entre el proceso de compresión y descompresión de un método. Esto significa que dado un archivo comprimido la recuperación de la imagen únicamente se conseguirá aplicando el proceso de descompresión del propio método aunque las propiedades de la imagen: color, dimensiones, tipo de imagen, etc. sean las mismas..

Las principales ventajas del estándar JPEG *lossless* expuestas en este analisis respecto a los metodos convencionales revisados en las secciones anteriores son las siguientes:

- Compatibilidad software y hardware
- No requiere utilizar tablas adicionales
- Baja complejidad de cálculos
- Consumo de memoria eficiente
- Facilita el intercambio de información entre ficheros de formato gráfico

Este método su versión sin pérdida de datos nunca ha tenido mucho éxito porque sus ratios de compresión típicos están comprendidos entre 1,4:1 y 2:1. Este ratio de compresion es inferior a muchos compresores *lossless* tradicionales. Como consecuencia la organización ISO decidió desarrollar un nuevo estándar sin pérdida de datos que esté cerca del estado del arte para la compresión de imágenes de continuos tonos. Este nuevo estándar es conocido como JPEG-LS que será revisado en la sección 4.9.

Antes de analizar el estándar actual, en la siguiente sección se revisan los algoritmos basados en transformadas *wavelets* precursores del estándar JPEG-LS y JPEG 2000.

3.7 Métodos de compresión de datos en el dominio de transformadas

Los métodos de compresión basados en modelos en el dominio de transformadas consisten en la aplicación de transformaciones matemáticas a los datos de entrada para representarlos en un nuevo dominio, tiempo o frecuencia. A partir de estos resultados, el proceso de codificación elimina o reduce la redundancia de los datos.

Los dos métodos a analizar, Transformada de Haar (Haar, 1910) (Lepik y Hein, 2014) y *Lifting Scheme* (Sweldens, 1996), se basan en *wavelets* y serán descritas en esta sección. El primer método se basa en un modelo estático, mientras que el

segundo lo hace en uno adaptativo, siguiendo la misma estrategia que otros modelos ya analizados en este capítulo.

Históricamente se han usado las transformadas *wavelets* para implementar métodos *lossy* (Goswami y Chan, 2011) ya que el uso de wavelets supone utilizar los conceptos de Transformada de Fourier Discreta (DFT) y de Transformada de Coseno Discreta (DCT), y en ambos hay pérdida de datos al discretizar la función original.

Joseph Fourier demostró que una función en el tiempo también podía expresarse como la suma de varias funciones exponenciales en el dominio de la frecuencia. Es decir, que del plano del tiempo se podía pasar al plano de la frecuencia aplicando la Transformada de Fourier. La DCT es similar a la DFT pero en este caso las funciones exponenciales se convierten en cosenos.

Basándonos en este enfoque y aplicando este método en la compresión de datos, ámbito distinto del original, es posible transformar los datos en un plano espacial expresado por los píxeles de una imagen mediante la DCT. Esta transformación por ser carácter discreto supone una pérdida de datos.

La transformada *wavelet* (traducida al español como ondita, ondícula u ondoleta, pero no muy usual) no es una idea reciente, la primera referencia data de 1909, donde aparece en un apéndice de la tesis doctoral de Alfréd Haar. El enfoque es similar al de Fourier, pero la Transformada Wavelet Discreta integra datos temporales y frecuenciales, frente a la DFT con solo datos frecuenciales. Este enfoque favorece su uso en diferentes campos, como por ejemplo la compresión de datos.

No es objeto de esta tesis el profundizar en aspectos matemáticos, sino su aplicación algorítmica a la compresión de datos. Desde este punto de vista, la transformada wavelet supone desplazar y escalar una función original que se regenera de forma continua.

Los algoritmos en el domino de las transformadas *wavelets* se basan en explotar la frecuencia espacial de los datos para conseguir la compresión de datos (Goswami y Chan, 2011). En el campo de la compresión de imágenes con pérdida de datos las transformadas *wavelet* han dado lugar a los algoritmos: EZW (*Embedded Zero-tree Wavelet*) (Shapiro, 1993), SPIHT (S*et Partitioning In Hierarchical Trees*) (Said y Pearlman, 1996), ASWDR (*AdaptivelyScanned Wavelet Difference Reduction*) (Raja y Suruliandi, 2011) y WDR (*Wavelet Difference Reduction*) (Tian y Wells, 1996). En la compresión sin pérdida de datos el método precursor es la transformada secuencial *S_transform*. En los últimos años, las *wavelets* han generado grandes expectativas en modo *lossless*, y estas juegan un factor muy destacable en el modo de operación *lossless* en el estándar JPEG 2000.

3.7.1 Introducción

La idea fundamental consiste en analizar una función en diferentes escalas. Una *wavelet* madre es usada para construir *wavelets* en diferentes escalas y trasladarlas a cada función que está siendo analizada. Los resultados de trasladar una *wavelet* dependen de cómo se relaciona esta con la función que está siendo analizada. Este desarrollo se basa en el concepto de multirresolución y será más comprensible al describir los tres pasos mediante sus algoritmos y ejemplos.

En el contexto de la compresión de imágenes, el concepto de multirresolución (Mallat, 1989) es la clave y una de las principales aportaciones de los métodos de transformada *wavelet* discretas a la compresión de imágenes. La idea básica consiste en que dada una imagen digital I_0 de nxm píxeles, ésta puede descomponerse en una nueva imagen I_1 de (n/2)x(m/2) píxeles. Para ilustrar esta idea, supongamos que dos píxeles adyacentes son reemplazados por un único píxel con el valor medio de sus valores. Obviamente, esta transformación implica una pérdida de datos aunque la información se mantiene con una insignificante perdida visual al escalar la imagen. En este punto de transformación, la imagen original es una superposición de la imagen promediada y para que esta sea reversible deben guardarse los valores promediados junto con los valores de detalles o distancias, esto es, $I_0 = I_1 + d_1$. Esto significa que la imagen original puede reconstruirse con los valores de la imagen escalada y sus coeficientes de detalle.

Aplicando el mismo procedimiento a la imagen escalada I_1 obtenemos una nueva imagen I_2 de (n/4)x(m/4) píxeles y los coeficientes de nivel 2. Para reconstruir la imagen original I_0 no es necesario guardar las imágenes escaladas anteriormente, con la última imagen escalada y los coeficientes de las anteriores es posible reconstruir la imagen sin pérdida de datos. Por tanto, la imagen original es reversible y se obtiene aplicando la siguiente expresión:

$$I_0 = I_2 + (d_1 + d_2) \tag{3.31}$$

Aplicando el mismo procedimiento después de *n* pasos, la imagen original es representada mediante la imagen final promediada más una superposición de los coeficientes obtenidos en todos los niveles.

$$I_0 = I_n + (d_1 + d_2 + d_3 +, \dots, +d_n) \tag{3.32}$$

La imagen final escalada *In* puede ser transmitida o almacenada en lugar de la imagen original. Para recuperar la imagen original es necesario guardar todos los coeficientes (Said y Pearlman, 1993).

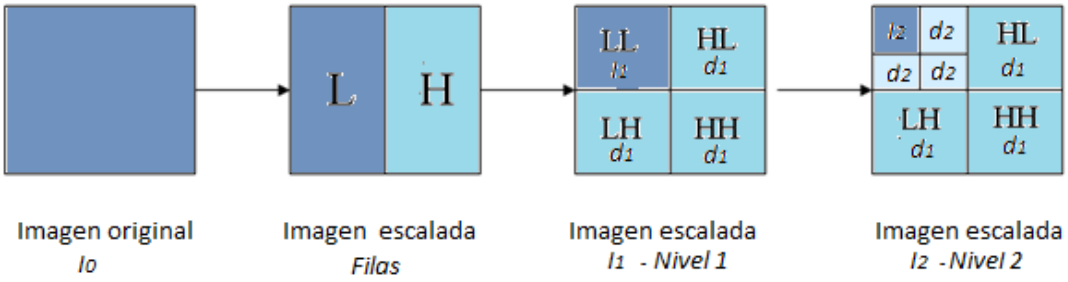


Figura 3.64. Diagrama de bloques de una estructura de multirresolución

La estructura resultante de la multiresolución presenta las siguientes transformaciones en una escala de nivel 1:

- LL representa los contenidos de aproximación de la imagen resultante de un filtrado de paso bajo en ambas direcciones: horizontal y vertical.
- LH representa los contenidos de detalle horizontal resultante de un filtrado de paso alto vertical y un filtrado de paso bajo horizontal.
- HL representa los detalles verticales resultantes de un filtrado de paso bajo vertical y filtrado de paso alto horizontal.
- HH representa detalles imagen diagonal resultante de un filtrado de paso alto vertical y horizontal.

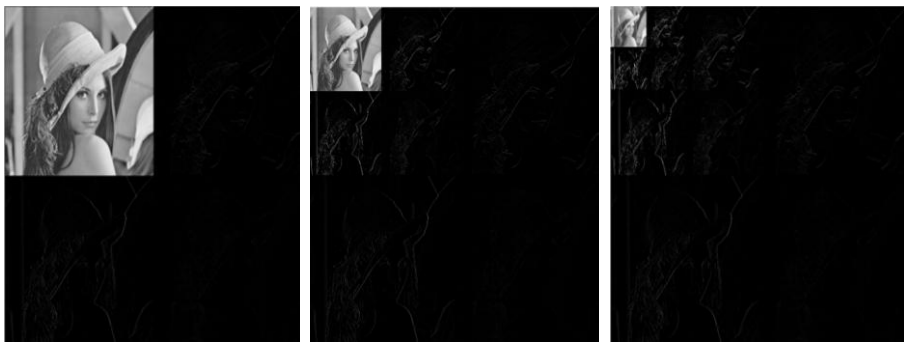


Figura 3.65. Imagen escalada en nivel 1,2 y 3

Las sucesivas transformaciones mantienen siempre el mismo número de filas y columnas. La ventaja que aportan a la compresión de imágenes radica en que los valores de los coeficientes se aproximan a cero o son cero y en consecuencia estos valores pueden ser eficientemente codificados con menos bits por métodos de compresión convencionales. En las versiones *lossy* se establece un valor umbral (media, mediana, moda, etc.) y los valores por debajo del umbral son sustituidos por cero alcanzándose grandes ratios de compresión (Menaka, Kumar y Bharathi, 2013). Por el contrario, en las versiones *lossless* de la transformadas *wavelet* es necesario guardar los coeficientes para ser codificados por el Codificador Huffman, RLE o el Codificador Aritmético. La estructura general de un método de compresión *wavelet* consiste en una transformada *wavelet* y un codificador (Fig. 3.66).

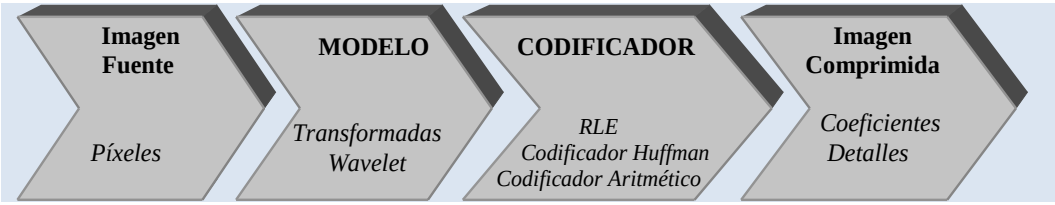


Figura 3.66. Diagrama de bloques de una estructura de compresión wavelet

En las siguientes secciones se analiza y revisa la transformada W*avelet Integer to Integer* (Calderbank, Daubechies, Sweldens y Yeo, 1996) (Calderbank, Daubechies, Sweldens y Yeo, 1998) como la versión estática más representativa aplicando la transformada de Haar como modelo *lossless* precursor, y la versión adaptativa vía *Lifting* que actualmente supone la principal contribución de las transformadas *wavelets* digitales sin pérdida de datos a la compresión de datos.

3.7.2 Transformada wavelet de Haar

La idea básica es transformar la imagen original como una superposición de wavelets que se obtienen recursivamente partiendo de la denominada *wavelet* madre (n=0) mediante escalado y traslación (Wang, 2012).

Inicialmente, una señal es dividida en dos funciones cuyos rangos son de 0 a ½ y ½ a 1 respectivamente. A continuación, la señal original es dividida en cuatro funciones de 0 a ¼ , ¼ a ½, ½ a ¾ y ¾ a 1, y así sucesivamente. Los códigos resultantes de cada señal y los coeficientes representan la señal original en una determinada escala o resolución.

Como se ve, la sucesivas *wavelets* se obtienen a partir de la *wavelet* madre (n=0) reduciendo progresivamente la escala en potencias de dos. El escalado y la traslación *wavelet* se implementan con matrices que son ortogonales, lo que supone una gran ventaja algorítmica.

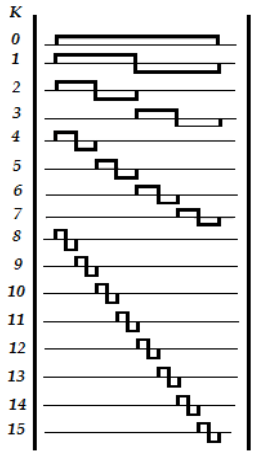


Figura 3.67. Funciones base y escala básica de la transformada de Haar

La transformada *wavelet* de Haar se basa en explotar la gran correlación que existe entre los píxeles adyacentes de una imagen. Considerando la imagen como una secuencia de pares de píxeles (*a,b*) es posible aplicar una simple transformación lineal que sustituya estos valores por su media (*s*) y sus diferencias (*d*).

$$s = \left(\frac{a + b}{2}\right) \tag{3.33}$$

$$d = b - s \quad d = a - s \tag{3.34}$$

La transformada *wavelet* de Haar se puede aplicar de forma completa a la señal s_n mediante una matriz de $N \times N$ siendo $N=2^n$. El coste computacional es proporcional a N. La imagen se procesa en bloques reducidos, generalmente de 8x8 píxeles.

3.7.2.1 Proceso de compresión

El proceso de compresión consiste en procesar la imagen en bloques reducidos de 8x8 píxeles para reducir espacio de memoria y tiempos de ejecución. En un primer paso, el algoritmo de compresión convierte los datos de la imagen 2-D en un vector lineal de las filas de la imagen antes de aplicar la transformada de Haar. A continuación, el mismo procedimiento se aplica a a las columnas obteniendo una una imagen escalada de primer nivel que contiene los datos formados por los promedios y diferencias manteniendo la información visual de la imagen. Este proceso de multirresolución mediante sucesivos escalados se repite hasta completar todos los bloques de la imagen. El resultado es una imagen o matriz de la misma dimensión que la imagen original. La ventaja es que el contenido de esta imagen contendrá una gran cantidad de ceros debido a la correlación existente entre los píxeles y valores próximos a cero. Por tanto, se necesitan menos bits para su representación que los datos originales. A partir de estos datos es posible codificar de forma eficiente mediante cualquiera de los tres métodos propuesto: *RLE o* Huffman, o Arithmetic Coding. En la Fig. 3.68 se representa la estructura del ciclo de compresión.

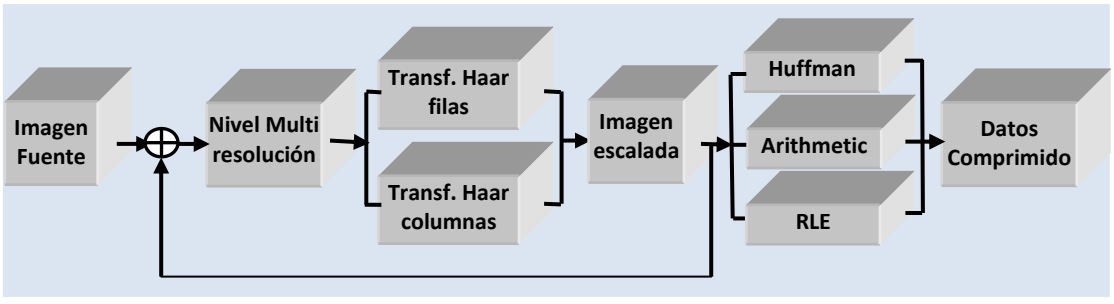


Figura 3.68. Diagrama de bloques del ciclo de compresión Haar

Una imagen discreta puede ser representada como una matriz de *NxM* que contiene *n* funciones de longitud *m.* Esta representación en forma matricial nos permite aplicar la transformada *wavelet* de Haar.

$$f = \begin{Bmatrix} f_{0,1} & f_{1,1} & \cdots & f_{m,1} \\ f_{0,2} & f_{1,2} & \cdots & f_{m,2} \\ \cdots & \cdots & \cdots & \cdots \\ f_{0,n} & f_{1,n} & \cdots & f_{m,n} \end{Bmatrix} \Longrightarrow \begin{matrix} f_1 = \{f_{0,1}, f_{1,1}, f_{2,1}, \ldots, f_{m,1}\} \\ f_2 = \{f_{0,2}, f_{1,2}, f_{2,2}, \ldots, f_{m,2}\} \\ \\ f_n = \{f_{0,n}, f_{1,n}, f_{2,n}, \ldots, f_{m,n}\} \end{matrix}$$

Inicialmente, la imagen original es segmentada en bloques de 8 x 8 píxeles. El primer paso, consiste en generar un vector con los ocho píxeles de la primera fila. Para ilustrar su funcionamiento asignamos unos valores cualquiera.

$$\begin{aligned} \mathrm{f}_1 &= \{\mathrm{f}_{0,1}, \mathrm{f}_{1,1}, \mathrm{f}_{2,1}, \mathrm{f}_{3,1}, \mathrm{f}_{4,1}, \mathrm{f}_{5,1}, \mathrm{f}_{6,1}, \mathrm{f}_{7,1}\} \\ y &= \{448, 768, 704, 640, 1280, 1408, 1600, 1600\} \end{aligned} \tag{3.35}$$

El número de píxeles representados es $8=2^3$, por tanto, el número máximo de niveles de transformación para este vector es 3. Si el número de elementos del vector fuera 2^k, entonces el número de transformaciones sería k. Aplicando la transformada de Haar obtenemos el siguiente resultado:

Cálculo de las medias: $\underbrace{448 \quad 768}_{608} \quad \underbrace{704 \quad 640}_{672} \quad \underbrace{1280 \quad 1408}_{1344} \quad \underbrace{1600 \quad 1600}_{1600}$

$$\left[\underbrace{448 \quad 768}_{608} \quad \underbrace{704 \quad 640}_{672} \quad \underbrace{1280 \quad 1408}_{1344} \quad \underbrace{1600 \quad 1600}_{1600} \right] \tag{3.36}$$

Cálculo de las distancias:

$$\left[\underbrace{608\text{-}768}_{-160} \quad \underbrace{672\text{-}640}_{32} \quad \underbrace{1344\text{-}1408}_{-64} \quad \underbrace{1600\text{-}1600}_{0} \right] \tag{3.37}$$

Los valores promediados se convierten en los cuatro primeros elementos de la transformada y las distancias se colocan a continuación de éstos. A las medias se les conoce como *coeficientes de aproximación,* y a las distancias se les denomina *coeficientes de detalle*. El resultado de la primera transformación es el siguiente:

$$y_1 = \{608, 672, 1344, 1600, -160, 32, -64, 0\}$$

En un siguiente paso se aplica la transformada de Haar a y_1 para obtener y_2. El resultado es el siguiente:

$$y_2 = \{640, 1472, -32, -128, -160, 32, -64, \ 0\}$$

Los coeficientes de aproximación actuales se representan junto a sus coeficientes de detalle, mientras que los coeficientes de detalle calculados anteriormente se mantienen en la misma posición.

$$y_3 = \{1056, -416, -32, -128, -160, 32, -64, \ 0\} \tag{3.38}$$

El primer coeficiente de aproximación es la media de todos los elementos. El resto de los elementos son los coeficientes de detalle. El número de elementos permanece siendo el mismo. En estos pasos de transformaciones no se obtiene compresión, el objetivo es preparar los datos para la codificación.

Todas estas transformaciones sucesivas aplicadas a un vector pueden representarse y resolverse eficientemente de forma matricial (Klein, 2013). En el ejemplo propuesto, la transformación de y a y_1 se obtiene aplicando la expresión [3.39].

$$\mathrm{y}_1 = \mathrm{y} * \mathrm{M}_1 \tag{3.39}$$

$$M_1 = \begin{pmatrix} 1/2 & 0 & 0 & 0 & 1/2 & 0 & 0 & 0 \\ 1/2 & 0 & 0 & 0 & -1/2 & 0 & 0 & 0 \\ 0 & 1/2 & 0 & 0 & 0 & 1/2 & 0 & 0 \\ 0 & 1/2 & 0 & 0 & 0 & -1/2 & 0 & 0 \\ 0 & 0 & 1/2 & 0 & 0 & 0 & 1/2 & 0 \\ 0 & 0 & 1/2 & 0 & 0 & 0 & -1/2 & 0 \\ 0 & 0 & 0 & 1/2 & 0 & 0 & 0 & 1/2 \\ 0 & 0 & 0 & 1/2 & 0 & 0 & 0 & -1/2 \end{pmatrix}$$

La transformación de y_1 a y_2 se obtiene aplicando la siguiente expresión:

$$\mathrm{y}_2 = \mathrm{y}_1 * \mathrm{M}_2 \tag{3.40}$$

$$M_2 = \begin{pmatrix} 1/2 & 0 & 1/2 & 0 & 0 & 0 & 0 & 0 \\ 1/2 & 0 & -1/2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1/2 & 0 & 1/2 & 0 & 0 & 0 & 0 \\ 0 & 1/2 & 0 & -1/2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

La tercera transformación se obtiene:

$$\mathrm{y}_3 = \mathrm{y}_2 * \mathrm{M}_3 \tag{3.41}$$

$$M_3 = \begin{pmatrix} 1/2 & 1/2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1/2 & -1/2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Por tanto, la transformación puede realizarse en un solo paso mediante la siguiente ecuación:

$$\mathrm{y}_3 = \mathrm{y} * \mathrm{W} \tag{3.42}$$

donde W es la matriz de transformación conocida como la matriz de Haar:

$$\mathrm{W} = \mathrm{M}_1 * \mathrm{M}_2 * \mathrm{M}_3 \tag{3.43}$$

$$W = \begin{pmatrix} 1/8 & 1/8 & 1/4 & 0 & 1/2 & 0 & 0 & 0 \\ 1/8 & 1/8 & 1/4 & 0 & -1/2 & 0 & 0 & 0 \\ 1/8 & 1/8 & -1/4 & 0 & 0 & 1/2 & 0 & 0 \\ 1/8 & 1/8 & -1/4 & 0 & 0 & -1/2 & 0 & 0 \\ 1/8 & -1/8 & 0 & 1/4 & 0 & 0 & 1/2 & 0 \\ 1/8 & -1/8 & 0 & 1/4 & 0 & 0 & -1/2 & 0 \\ 1/8 & -1/8 & 0 & -1/4 & 0 & 0 & 0 & 1/2 \\ 1/8 & -1/8 & 0 & -1/4 & 0 & 0 & 0 & -1/2 \end{pmatrix}$$

La matriz resultante de transformar las filas se obtiene mediante la siguiente ecuación:

$$\mathrm{Q} = \mathrm{I} * \mathrm{W} \tag{3.44}$$

siendo I la matriz original.

La transformada de Haar aplica las mismas transformaciones a cada columna de la matriz *Q*. Esto es lo mismo que multiplicar la matriz original *I* por la transpuesta de *W*. El resultado de la ecuación final para la matriz transformada de las filas y columnas es la matriz *T*.

$$\mathrm{T} = \mathrm{W}^{\mathrm{T}} * \mathrm{I} * \mathrm{W} \tag{3.45}$$

Esta transformada es conocida como la transformada rápida de Haar la cual es reversible. La imagen *T* no supone compresión sino que su objetivo es transformar la imagen original en una imagen transformada que contenga una gran cantidad de valores igual a cero y próximos a cero para favorecer la codificación mediante un método de compresión tradicional.

3.7.2.2 Algoritmo de compresión

El algoritmo de compresión no puede aplicarse a toda la imagen en un solo paso porque los cálculos matriciales serían muy costosos. La solución consiste en procesar los datos en bloques más reducidos, por ejemplo bloques de 8x8 píxeles. Esto supone que es necesario operaciones complementarias para organizar los coeficientes de aproximación y los de detalles según la estructura de multirresolución *wavelet*.

Los pasos para calcular la transformada de Haar de una array de *n* elementos (potencia de dos) son los siguientes: (transformada unididimensional)

Paso 1. Obtiene la media de cada par de píxeles (n/2 coeficientes de aproximación)
Paso 2. Coloca la primera la primera mitad del array con los coeficientes de aproximación
Paso 3. Obtiene las diferencias entre la media y un píxel (n/2 coeficientes de detalle)
Paso 4. Completa la segunda mitad del array con los coeficientes de detalle
Paso 5. Repetir el proceso en la primera parte del array desde el paso 1.

En operaciones de matrices el algoritmo se reduce a un solo paso (transformada bidimensional)

Paso 1. . Transformar la imagen original en una matriz de un nivel de resolución (T) aplicando la ecuación
Paso 2. Organiza la matriz en aproximaciones (resolución) y coeficientes de detalle
Paso 3. Repite los pasos 1 y hasta finalizar la imagen

3.7.2.3 Programación del algoritmo de compresión de Haar wavelet

La programación del algoritmo se basa en operaciones con matrices y bucles. Las operaciones con matrices se ejecutan rápidamente en los ordenadores actuales mientras que los bucles ralentizan los tiempos de ejecución.

En el Listado 3.14 se muetra la programación del proceso de compresión para una iteración por filas. El resultado es aplicado a una nueva iteración pero con columnas.

```
// Transformada de Haar – Proceso de compresion
int main()
{
double H1[8][8],N[8][8];
// Operación por filas
MatrixMul(M1,W,8,8,8,H1);  //  Transformada de Haar
for(i=0;i<8;i++)    // ciclo para imprimir datos de H1
{
   For (j=0;j<8;j++)
       printf("%3.1lf ",H1[i][j]); // ecsribe en pantalla
}
printf("\n"); }
// Operación por columnas
MatrixMul(Wt,H1,8,8,8,N);
for(i=0;i<8;i++)
{
```

```
16      for(j=0;j<8;j++)
17        printf("%3.1lf  ",N[i][j]);  // escrbe en pantalla
18  }
19  }
```

Listado 3.14. Programación del algoritmo de compresión de Haar

3.7.2.4 Ejemplo práctico

Para simplificar el procedimiento tomemos una matriz de 8x8 píxeles correspondientes al primer bloque de 8x8 píxeles de la imagen Lena.

$$I = \begin{Bmatrix} 162,162,162,161,162,157,163,171 \\ 162,162,162,161,162,157,163,171 \\ 162,162,162,161,162,157,163,171 \\ 162,162,162,161,162,157,163,171, \\ 162,162,162,161,162,157,163,171, \\ 164,164,158,155,161,159,159,160, \\ 160,160,163,158,160,162,159,156 \\ 159,159,155,157,158,159,156,157; \end{Bmatrix}$$

Aplicando la transformada de Haar al bloque por filas obtenemos nueve valores igual a cero.

$$M1 = \begin{Bmatrix} 161.2, & 0.5, & 0.2, & -1.2, & 0.0, & 0.5, & 2.5, & 1.0 \\ 161.2, & 0.5, & 0.2, & -1.2, & 0.0, & 0.5, & 2.5, & 1.0 \\ 161.2, & 0.5, & 0.2, & -1.2, & 0.0, & 0.5, & 2.5, & 1.0 \\ 161.2, & 0.5, & 0.2, & -1.2, & 0.0, & 0.5, & 2.5, & 1.0 \\ 161.2, & 0.5, & 0.2, & -1.2, & 0.0, & 0.5, & 2.5, & 1.0 \\ 160.0, & 0.2, & 3.8, & 0.2, & 0.0, & 1.5, & 1.0, & -0.5, \\ 159.8, & 0.5, & -0.2, & 1.8, & 0.0, & 2.5, & -1.0, & 1.5, \\ 157.5, & 0.0, & 1.5, & 1.0, & 0.0, & -1.0, & -0.5, & -0.5 \end{Bmatrix}$$

Aplicando la transformada a las columnas obtenemos 29 ceros.

$$M2 = \begin{Bmatrix} 160.4, & 0.4, & 0.8, & -0.4, & 0.0, & 0.7, & 1.5, & 0.7 \\ 0.8, & 0.1, & -0.5, & -0.8, & 0.0, & -0.2, & 1.0, & 0.3 \\ 0.0, & 0.0, & 0.0, & 0.0, & 0.0, & 0.0, & 0.0, & 0.0 \\ 1.0, & 0.1, & 0.7, & -0.9, & 0.0, & 0.1, & 1.2, & -0.1 \\ 0.0, & 0.0, & 0.0, & 0.0, & 0.0, & 0.0, & 0.0, & 0.0 \\ 0.0, & 0.0, & 0.0, & 0.0, & 0.0, & 0.0, & 0.0, & 0.0 \\ 0.6, & 0.1, & -1.8, & -0.8, & 0.0, & -0.5, & 0.8, & 0.8 \\ 1.1, & 0.2, & -0.9, & 0.4, & 0.0, & 1.8, & -0.2, & 1.0 \end{Bmatrix}$$

El resultado es una matriz con el mismo número de componentes que la original en la que contiene 29 valores igual a cero. Este resultado facilita la codificación por métodos convencionales: RLE o Codificador Aritmético o Codificador Huffman. El

principal problema aparece con los resultados negativos, ya que exige añadir un nuevo bit lo que compromete la capacidad de compresión del método.

El proceso de descompresion aplica la transformada de Haar inversa para recuperar los datos.

3.7.2.5 Proceso de descompresión

El ciclo de descompresión se realiza desde la imagen transformada que presenta la estructura de datos de la Fig. 3.69.

Nivel de resolución Aproximaciones Coeficiente de las medias Píxeles más significativos	Coeficientes de detalles Diferencia entre medias y valores Píxeles menos significativos
Coeficientes de detalles Diferencia entre medias y valores	Coeficientes de detalles Diferencia entre medias y valores

Figura 3.69. Estructura de la matriz de descompresión

A partir de estos datos se recuperan los datos en una primera fase de decodificación aplicando el mismo método de codificación que en el ciclo de compresión (Fig. 3.70). La salida del decodificador es el resultado de la transformada inversa de Haar.

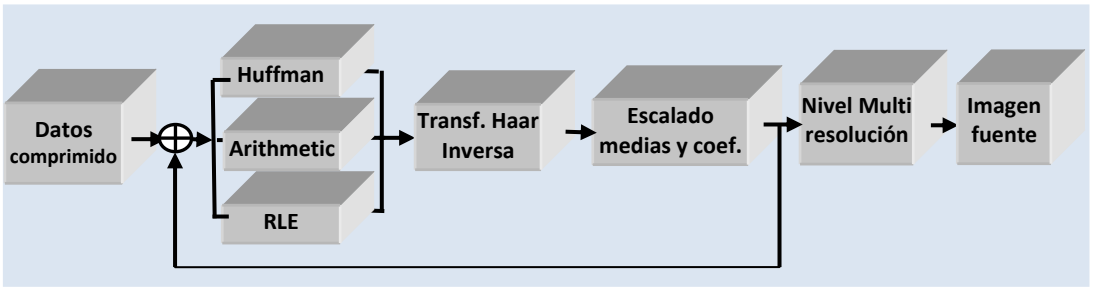


Figura 3.70. Diagrama de bloques de una estructura de compresión wavelet

La recuperación de los datos *(a,b)* es totalmente reversible a partir de *s* y *d* de la ecuación (3.46).

$$a = s - \left(\frac{d}{2}\right) \quad b = s + \left(\frac{d}{2}\right) \tag{3.46}$$

La matriz transformada de Haar es invertible mediante transformación de matrices en el modo bidimensional.

$$W^{-1} = M_3^{-1} * M_2^{-1} * M_1^{-1} \tag{3.47}$$

Esto significa que es posible reconstruir la imagen original a partir de la imagen comprimida. De la expresión (3.42) obtenemos la siguiente ecuación:

$$y = y_3 * W^{-1} \tag{3.48}$$

Para reconstruir la imagen original *I* desde la imagen transformada *T*, usamos la ecuación (3,44) porque la inversa de una matriz ortogonal es igual a su traspuesta.

$$I = (W^{-1})^T * T * W^{-1} \tag{3.49}$$

3.7.2.6 Programación del algoritmo inverso de Haar

```
1  // Transformada de Haar – algoritmo de descompresión
2  InvMatrix(W,8,invW);
3  InvMatrix(Wt,8,invWt);
3  /*Reordenar y descomprimir columnas*/
4  j=0;
5  h=0;
6  k=anchura*altura/8-anchura;
7  for(i=0;i<anchura*altura;i++){
8      arrayaux[0]=arrayt2[j];
9      arrayaux[1]=arrayt2[k];
10     arrayaux[2]=arrayt2[k+anchura];
11     arrayaux[3]=arrayt2[k+anchura*2];
12     arrayaux[4]=arrayt2[k+anchura*3];
13     arrayaux[5]=arrayt2[k+anchura*4];
14     arrayaux[6]=arrayt2[k+anchura*5];
15     arrayaux[7]=arrayt2[k+anchura*6];
16
17 MatrixMul(arrayaux,invW,1,8,8,arrayresult);
18 InvMatrix(W,8,invW);
19 InvMatrix(Wt,8,invWt);
20 arrayt4[i]=arrayresult[0];
21     arrayt4[i+anchura]=arrayresult[1];
22     arrayt4[i+anchura*2]=arrayresult[2];
23     arrayt4[i+anchura*3]=arrayresult[3];
24     arrayt4[i+anchura*4]=arrayresult[4];
25     arrayt4[i+anchura*5]=arrayresult[5];
26     arrayt4[i+anchura*6]=arrayresult[6];
27     arrayt4[i+anchura*7]=arrayresult[7];
28
29     i=i+(anchura*8)-1;
30     j=j+anchura;
31     k=k+(anchura*7);
32     if(k%((anchura*(altura-1))+h)==0) {
33            k=k-(7*anchura*altura/8)+1;
34            i=i-(anchura*altura)+1;
35            j=j-(anchura*altura/8)+1;
36            h++; }
37 }
38 /*Reordenar y comprimir por filas*/
39 j=0;
40 k=anchura/8;
41 for(i=0;i<(anchura*altura);i++){
42       arrayaux[0]=arrayt4[j];
43       arrayaux[1]=arrayt4[k];
44       arrayaux[2]=arrayt4[k+1];
45       arrayaux[3]=arrayt4[k+2];
46       arrayaux[4]=arrayt4[k+3];
47       arrayaux[5]=arrayt4[k+4];
48       arrayaux[6]=arrayt4[k+5];
49       arrayaux[7]=arrayt4[k+6];
50
51       MatrixMul(arrayaux,invW,1,8,8,arrayresult);
```

```
52
53        arrayt3[i]=arrayresult[0];
54        arrayt3[i+1]=arrayresult[1];
55        arrayt3[i+2]=arrayresult[2];
56        arrayt3[i+3]=arrayresult[3];
57        arrayt3[i+4]=arrayresult[4];
58        arrayt3[i+5]=arrayresult[5];
59        arrayt3[i+6]=arrayresult[6];
60        arrayt3[i+7]=arrayresult[7];
61
62        k=k+7;
63        j++;
64        If ((k%anchura)==0) {
65              j=j+(7*anchura/8);
66              k=k+(anchura/8); }
67              i=i+7;
68        }
69  }
```

Listado 3.15. Programación del algoritmo de descompresión de Haar

3.7.2.7 Ejemplo práctico

A partir de la matriz obtenida en el proceso de compresión

$$M2 = \begin{Bmatrix} 160.4, & 0.4, & 0.8, & -0.4, & 0.0, & 0.7, & 1.5, & 0.7 \\ 0.8, & 0.1, & -0.5, & -0.8, & 0.0, & -0.2, & 1.0, & 0.3 \\ 0.0, & 0.0, & 0.0, & 0.0, & 0.0, & 0.0, & 0.0, & 0.0 \\ 1.0, & 0.1, & 0.7, & -0.9, & 0.0, & 0.1, & 1.2, & -0.1 \\ 0.0, & 0.0, & 0.0, & 0.0, & 0.0, & 0.0, & 0.0, & 0.0 \\ 0.0, & 0.0, & 0.0, & 0.0, & 0.0, & 0.0, & 0.0, & 0.0 \\ 0.6, & 0.1, & -1.8, & -0.8, & 0.0, & -0.5, & 0.8, & 0.8 \\ 1.1, & 0.2, & -0.9, & 0.4, & 0.0, & 1.8, & -0.2, & 1.0 \end{Bmatrix}$$

Aplicando las transformadas inversas de la expresión (3.40) y (3.41) se recupera la imagen original.

$$I = \begin{Bmatrix} 162,162,162,161,162,157,163,171 \\ 162,162,162,161,162,157,163,171 \\ 162,162,162,161,162,157,163,171 \\ 162,162,162,161,162,157,163,171, \\ 162,162,162,161,162,157,163,171, \\ 164,164,158,155,161,159,159,160, \\ 160,160,163,158,160,162,159,156 \\ 159,159,155,157,158,159,156,157; \end{Bmatrix}$$

Los resultados de este método se verán conjuntamente con el de Lifting Scheme, como corresponde a los dos métodos, estático y adaptativo, respectivamente.

3.7.3 Lifting Scheme

El método *Lifting Scheme* también se basa, como la Transformada de Haar, en la aplicación de transformadas *wavelets*, y por tanto ambos métodos están soportados por el mismo esquema matemático-algorítmico. El método de Haar usa un modelo estático mientras que el *Lifting Scheme* es dinámico (Yang, Zhu y Yang, 2008), siguiendo la misma estrategia que las versiones adaptativas de los modelos estadísticos y de diccionario.

Las técnicas *lifting* fueron propuestas por Sweldens (Sweldens, 1966) para convertir transformadas no reversibles en reversibles. La idea básica consiste en desarrollar todas las operaciones en el dominio espacial y sin usar espacio de memoria extra. La imagen transformada es procesada *in-place* reemplazando los píxeles de la imagen original.

Estas técnicas son una herramienta muy flexible para construir nuevas y sucesivas descomposiciones de una señal existente (Said y Pearlman, 1996). Las transformadas de elevación son usadas como alternativa a la implementación con transformadas *wavelet* clásicas en la compresión de imágenes (Han, Liu y Xu., 2013). Adicionalmente, *Lifting Scheme* ofrece la posibilidad de recolocar y sustituir filtros lineales por no lineales construyendo de esta forma descomposiciones *wavelet* adaptativas.

La estructura general del método se compone de tres operaciones básicas: *split* o division, predicción y *update* o actualización (Daubechies y Sweldens, 1996). La Fig. 3.71 ilustra las tres operaciones.

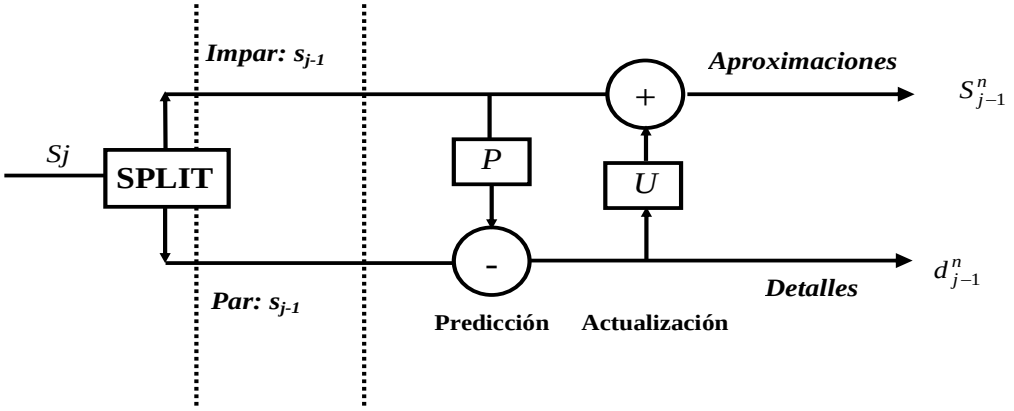


Figura 3.71. Tipos de operaciones de Lifting Scheme

En la primera etapa, la operación *split* divide una señal de entrada unidimensional S_j, formada por la secuencia de píxeles de una imagen, en dos nuevas subseñales más pequeñas, y que en adelante las denominaremos, $Impar_{j-1}$ *y* Par_{j-1}. La subseñal *Impar* es una *señal de aproximación* y la subseñal *Par* es una *señal de detalle* obtenida del proceso de transformada *wavelet* conocida como *Lazy Wavelet Transform.* Este

proceso *Lazy Wavelet* puede implementarse con un gran número de transformadas, entre las cuales destaca la transformada de Haar, que será utilizada en este caso.

$$\textit{Split o división}: S_j \text{ en } Par_{j-1} \text{ e } Impar_{j-1} \tag{3.50}$$

En la segunda etapa, el operador de predicción $\boldsymbol{P}$ actúa sobre la subseñal $Impar_{j-1}$ para modificar la subseñal Par_{j-1}, generando una nueva *señal de detalle* d_{j-1} explotando la propiedad de correlacion de los píxeles. Los detalles son reemplazados como la diferencia entre los datos y la predición.

$$\textit{Predicción:}\ d_{j-1} = Par_{j-1} - P(Impar_{j-1}) \tag{3.51}$$

En la práctica, el operador de predicción $\boldsymbol{P}$ se elige de tal manera que P ($Impar_{j-1}$) es una estimación de Par_{j-1}, por tanto, la nueva señal d_{j-1} debería ser menor que $Impar_{j-1.}$

En la última etapa, el operador de actualización $\boldsymbol{U}$ (*update*) actúa sobre la subseñal d_{j-1} para modificar la subseñal $Impar_{j-1}$, generando una *señal de aproximación*, $S_{j-1} = Impar_{j-1} + U(d_{j-1})$. La misión del operador de actualización consiste en el mantenimiento de las propiedades globales de la imagen en las sucesivas descomposiciones.

$$\textit{Update o actualización}: S_{j-1} = Impar_{j-1} + \boldsymbol{U}\,(d_{j-1}) \tag{3.52}$$

El operador de actualización es seleccionado de tal manera que la señal S_{j-1} resultante cumpla una cierta restricción como puede ser la preservación de la media de la señal de entrada S_j.

En la práctica, *Lifting Scheme* consiste en una secuencia de pasos sucesivos de las operaciones de *split*, predicción y actualización (Hattay, Belaid y Naanaa, 2013). El objetivo es obtener una descomposición resultante formada por detalles y aproximaciones y que representan la señal original manteniendo la propiedad de recursividad.

Esta transformada proporciona una correlación espacial entre los píxeles de una imagen, pero no significa que se produzca una *compresión* directa de los datos. En este contexto, a la aplicación de las sucesivas operaciones de elevación la denominaremos en adelante proceso de avance y a la recuperación la denominaremos proceso inverso.

Lifting Scheme requiere de un proceso adicional de codificación de la señal resultante que supuestamente contendrá componentes con valores cercanos a cero o ceros para una compresión elevada o máxima en la etapa de codificación. Al igual que los métodos estándares JPEG se recomienda el uso del Codificador Huffman o del Codificador Aritmético para realizar la codificación de los datos resultantes. La Fig. 3.72 muestra el diagrama de bloques del proceso de compresión de *Lifting Scheme.*

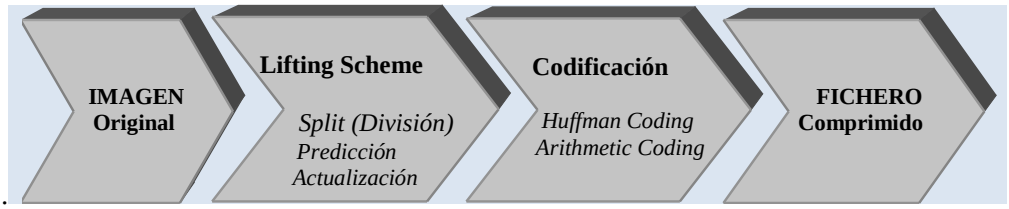


Figura 3.72. Diagrama de bloques del proceso de avance de Lifting Scheme

La eficiencia de este método reside más en el modelo adaptativo que en la codificación. Todas las transformadas de elevación no consiguen el mismo ratio de compresión sino que la diferencia entre ellas reside principalmente en el filtro utilizado en la operación de *split* (transformada *wavelet*) y en los filtros de predicción. *Lifting Scheme* puede utilizar muchos bancos de filtros de los diferentes filtros existentes: transformada de Haar, Daubechies (Daubechies I., 1988), polinomial (Jansen y Oonincx, 2010), etc.

La transformación inversa es reversible totalmente además de rápida y sencilla (Fig. 3.73). El proceso reversible consiste en invertir el orden de las operaciones y alternar los signos de los operadores + y -.

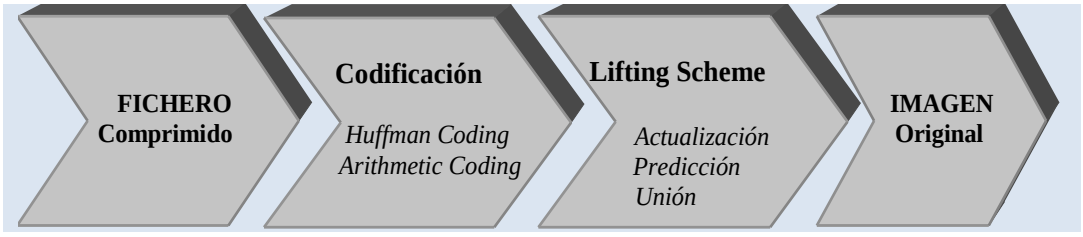


Figura 3.73. Diagrama de bloques del proceso inverso de Lifting Scheme

Una vez revisada la transformada de Haar en la anterior subsección nos permitirá comprender y descubrir el principio de funcionamiento de *Lifting Scheme* aplicando el filtro de Haar en los ciclos de compresión y descompresión.

La transformada de Haar se basa en cálculos de las medias y sus diferencias (aproximaciones). Esto es, puede ser interpretada como un filtro de paso bajo y un filtro de paso alto. Esta transformada suele ser usada frecuentemente como filtro de descomposición, *split,* por su facilidad de implementación.

$$\text{Proceso de avance} \Rightarrow \text{Impar}_j = \frac{\text{Impar}_{j-1}+\text{Par}_{j-1}}{2} \tag{3.53}$$

$$\text{Proceso inverso} \Rightarrow \text{Impar}_{j-1} = \text{Impar}_j - \frac{\text{Par}_{j-1}}{2} \tag{3.54}$$

A continuación se procede a la revisión del proceso de compresión del método *Lifting Scheme* aplicando la transformada de Haar como filtro *Split y predicci*. En este método se utiliza la misma metodología de revisión que en los anteriores.

3.7.3.1 Proceso de compresión

El objetivo del proceso de compresión mediante *Lifting Scheme* (Said y Pearlman, 1996) consiste en explotar la correlación entre píxeles adyacentes para que las diferencias en las sucesivas descomposiciones de las *señales de detalles* contengan valores iguales a cero y de esta forma preparar los datos para una codificación mediante el Codificador Aritmético o Huffman (Kabir y cols., 2013). La salida del proceso de compresión *lifting* estará formada por una secuencia de aproximaciones de los datos en el último nivel y de otra secuencia formada por los valores residuales obtenidos en cada nivel de detalles.

En la Fig. 3.74 se ilustra el ciclo de compresión usando la transformada Haar como transformada *wavelet* en la operación Split.

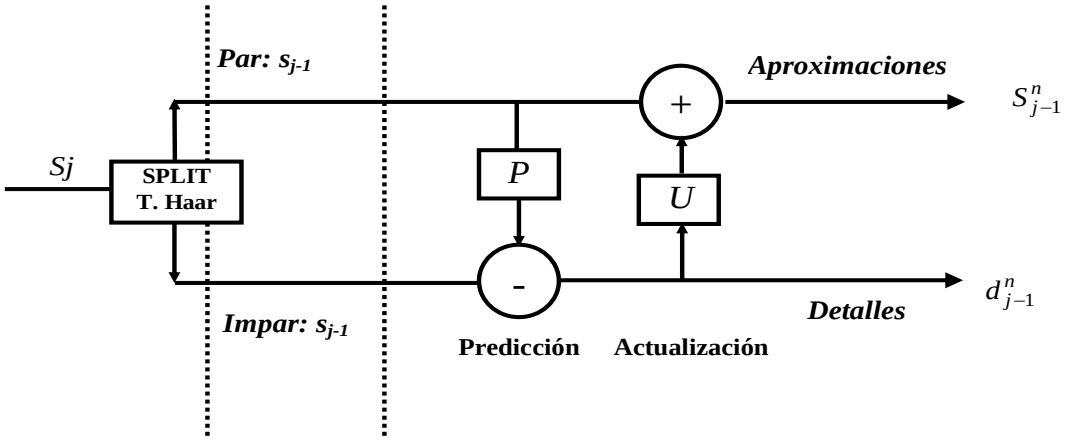


Figura 3.74. Estructura de la transformada wavelet de Haar

El proceso de compresión descompone la imagen en bloques de tamaño 2^n. El tamaño convencional suelen ser bloques de 8 píxeles, 2^3. La idea básica en la que se basa y se formula la transformada *wavelet* de Haar consiste en asociar a *Impar* y *Par* los píxeles *a* y *b*, aplicando la transformada de medias y diferencias a cada par de píxeles (*a,b*), obteniéndose de esta forma las ecuaciones de la transformada *wavelet* directa de Haar para dos píxeles :

$$d = b - a \tag{3.55}$$

$$s = \frac{(a + b)}{2} \tag{3.56}$$

La Fig. 3.75 muestra una gráfica de implementación de las sucesivas descomposiones de la señal y las superposiciones.

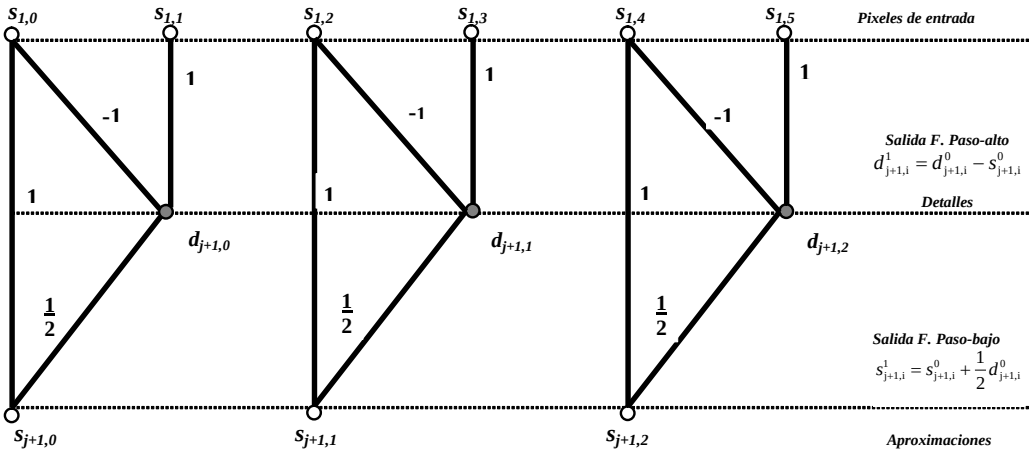


Figura 3.75. Implementación de las iteraciones de la transformada de Haar

Una vez expuesto el modelo de avance, un grafico ilustrativo de implementación y definida la transformada de Haar como filtro *wavelet* de *Lifting Scheme* se procede a describir las tres operaciones: *Split*, predicción y actualización.

Operación *split*

La operación *split* consiste en dividir la´secuencia de píxeles S_j en dos subconjuntos de componentes separados y denotados por los elementos con posiciones impares y pares, siendo *j* el índice de diferencias y *n* el número de píxeles.

$$S_j = (Par_{j-1}, Impar_{j-1})$$

$$S_j = \{S_{j,0}, S_{j,1}, S_{j,2}, \dots, S_{j,n-1}, S_{j,n}\} \tag{3.57}$$

Dada una señal de entrada S_j está es separada en dos señales. Una primera señal S_{j-1} está formada por 2^{n-1} medias S_{j-1} y una segunda señal d_{n-1} formada por 2^{n-1} diferencias d_{j-1}.

$$S_j = \{S_{j-1,0}, d_{j-1,0}, S_{j-1,1}, d_{j-1,1}, \dots, S_{j-1,n}, d_{j-1,n,}\} \tag{3.58}$$

En la Fig. 3.76 se refleja la estructura de división de los datos

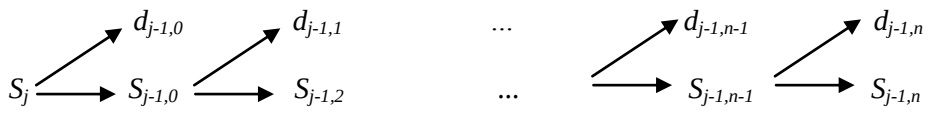


Figura 3.76. Estructura de división de los datos en el proceso split

El uso de la transformada *wavelet* de Haar en la operación *Split* divide la señal en dos subseñales *split* => s_j = (s_{j-1}, d_{j-1}); siendo s_{j-1} la subseñal de aproximación *Impar* $_{j-1}$ y d_{j-1} la subseñal de detalles *Par* $_{j-1}$. De aquí en adelante, denotaremos los píxeles impar y par como *a* y *b*.

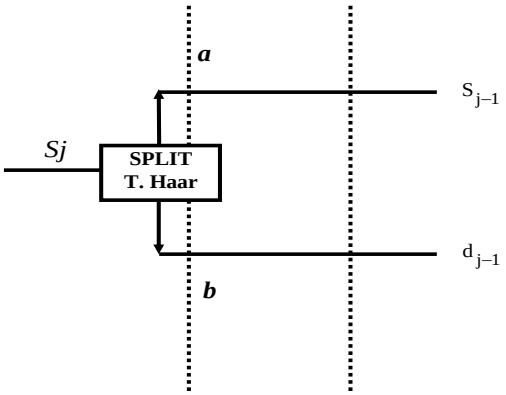


Figura 3.77. Operación Split

Operación de predicción

Cada componente en la señal $Impar_{j-1}$ o *a* es consecutivo a su correspondiente valor en la señal *Par* $_{j-1}$ o *b*. Los dos valores son correlativos y por tanto cada uno puede ser usado para predecir el otro, la operación de predicción tiene por objetivo encontrar un valor residual o diferencia d_{j-1} entre el componente impar y su predicción desde el componente par:

$$\mathrm{d_{j-1}} = \mathrm{Impar_{j-1}} - P\left(\mathrm{Par_{j-1}}\right) \tag{3.59}$$

siendo ***P*** un operador de predicción.

$$\mathrm{S_{j-1}} = \mathrm{Impar_{j-1}} - P\left(\mathrm{Par_{j-1}}\right) \tag{3.60}$$

La Fig. 3.78 ilustra el funcionamiento del operador de predicción,

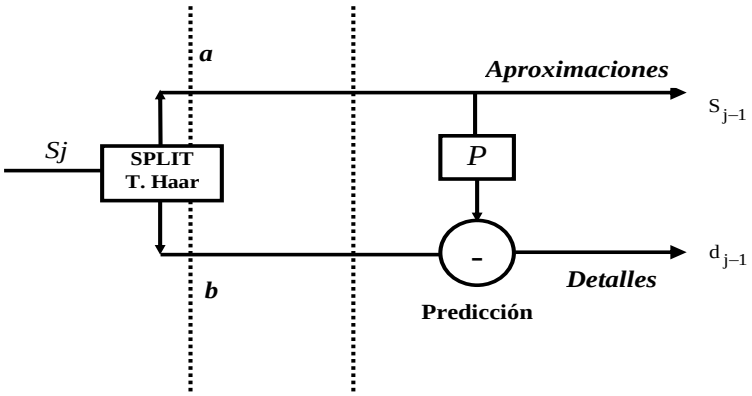


Figura 3.78. Operación de predicción

Aplicando la Transformada *wavelet* de *Haar* en *Lifting Scheme*, obtenemos:

$$d = b - a \tag{3.61}$$

$$s = \frac{a + b}{2} \tag{3.62}$$

Reescribiendo las expresiones anteriores,

$$s = a + \frac{d}{2} \tag{3.63}$$

Las variables *s* y *d* no son necesarias, *a* reemplaza a *s y b* reemplaza a *d* obteniéndose las siguientes ecuaciones para la operación de predicción.

$$b = b - a \tag{3.64}$$

$$a = a + \frac{b}{2} \tag{3.65}$$

Aplicando la Transformada clásica de Haar, el operador de predicción ***P*** puede definirse como un simple desplazamiento desde la muestra más cercana, esto es, utiliza el conjunto de componentes pares para predecir el componente impar, usa par_{j-1} para predecir $impar_{j-1}$.

Operación de Actualización

La operación de actualización consiste en encontrar una aproximación S_{j-1} para actualizar el componente Par_{j-1}.

$$s_{j-1} = s_{j-1} + U(d_{j-1}) \tag{3.66}$$

siendo U: un operardor de actualización.

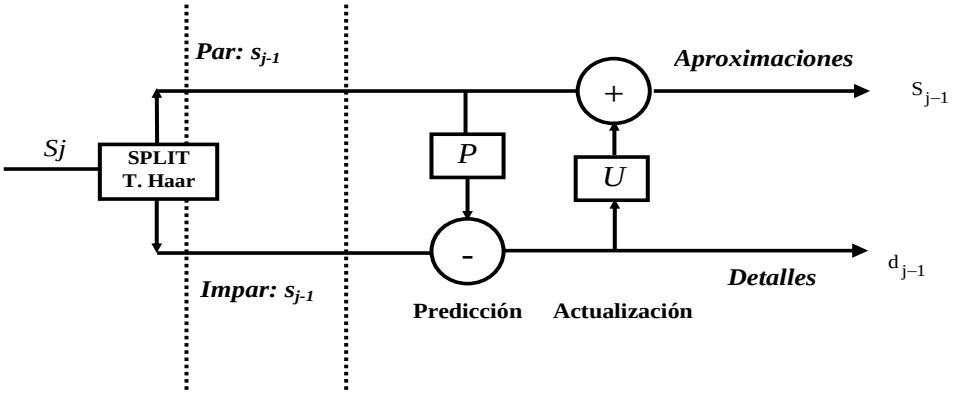


Figura 3.79. Operación de actualización

La aproximación S_{j-1}^{n} se convierte en el nuevo dato y la secuencia de pasos o descomposiciónes de la señal se guarda en d_{j-1}^{n}, siendo *n* el nivel de descomposición o iteración. Una de las principales ventajas de esta operación reside

en que es realizada *in-place*. Esto es, S_{j-1}^{n} y d_{j-1}^{n} reemplazan *in-place* a los píxeles originales sin necesidad de memoria adicional. Los componentes S_j del conjunto *Par* son reemplazados con las medias S_{j-1} y los componentes impares son sobreescritos con las diferencias d_{j-1}.

La clave del éxito de una transformada basada en *Lifting Scheme* radica en que el operador de predicción *P* obtenga valores residuales muy pequeños mientras el operador de actualización *U* preserva los datos originales y los prepara para el siguiente nivel.

Alternativamente, a la Transformada de Haar, otras transformadas como son las transformadas biortogonalaes construyen un operador de predicción basado en una interpolación lineal entre dos muestras adyacentes. Otras transformadas *wavelet* aplican *lifting* con filtros lineales más sofisticados que explotan la predicción entre varios píxeles adyacentes para grantizar una mejor predicción y actualización (Mukherjee, Swamy y Bhavani, 2014). La clave del éxito en el proceso de ompresión mediante técncias *lifting* reside en los filtros utizados.

3.7.3.2 Algoritmo de compresión del Lifting Scheme

El algoritmo de la transformada *Lifting digital* (Sweldens, 1995) es el siguiente:

Paso 1. Organiza los datos de una imagen (píxeles) en una secuencia de registros
Paso 2. Separa los datos en componentes impares (o) y pares (d)
Paso 3. Define y calcula un valor residual (r) como la diferencia entre el componente impar y su predicción desde el componente par

$$r = o - P[e]$$

siendo P un operador de predicción
Paso 4. Busca una aproximación (c) del dato impar actualizando el componente par.

$$c = e + U[r]$$

donde U es un operador de actualización

Paso 5. El valor (c) se convierte en el nuevo dato y la secuencia se repite en el siguiente nivel desde el primer paso

3.7.3.2.1 Programación del algoritmo

```
1  // Lifting Scheme proceso de avance
2  for (i=0; i=n/2; i+1){
3  for (a = 0; a<n; a+2){
4      a= array[a];
5      b= array[a+1];
6      b = b-a;         // valor de detalle
7      a = a+(b/2);     // valor de aproximación
8      array(inda)= a;
9      array(indb)= b;}
10 }
```

Listado 3.16. Programación del algoritmo de compresión Lifting Scheme

3.7.3.2.2 Ejemplo práctico (Lifting Scheme)

Supongamos cuatro píxeles cualquiera con los siguientes valores: 1, 3, 8, 6. El esquema de elevación utiliza la Transformada *wavelet* de Haar y almacenamiento *in-place*.

Inicialmente los cuatros píxeles de la imagen original son almacenados en un array unidemensional con índices consecutivos: pixel[0]=1, pixel[1]=3, pixel[2]=8 y pixel[3]=6.

PIXEL[0]	PIXEL[1]	PIXEL[2]	PIXEL[3]
1	3	8	6

La operación Split separa los píxeles impares y pares según el índice (Fig. 3.80).

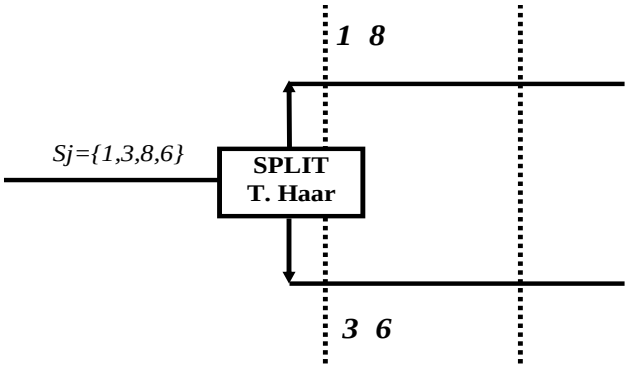


Figura 3.80. Ejemplo prácticode la operación de Split

Aplicando las ecuaciones (3.64) y (3,65) se obtiene el primer nivel de descomposición de la señal:

1ª Iteración: $a = 1$ y $b = 3 \rightarrow \begin{cases} b = b - a = 3 - 1 = 2 \\ a = a + \frac{b}{2} = 1 + \frac{2}{2} = 2 \end{cases}$

1ª Iteración: $a = 8$ y $b = 6 \rightarrow \begin{cases} b = b - a = 6 - 8 = -2 \\ a = a + \frac{b}{2} = 8 + \frac{(-2)}{2} = 7 \end{cases}$

Píxel[0]	Píxel[1]	Píxel[2]	Píxel[3]
a	b	a	b
1	3	8	6
2	2	7	-2
$S^1_{j-1,0}$	$d^1_{j-1,0}$	$S^1_{j-1,1}$	$d^1_{j-1,1}$

La señal resultante es la suma de la señal de aproximación y la señal de detalle $S_{j\text{-}1}$= {2, 2, 7, -2}. En la Fig. 3.81 se muestran los resultados de primera descomposición de la señal S_j.

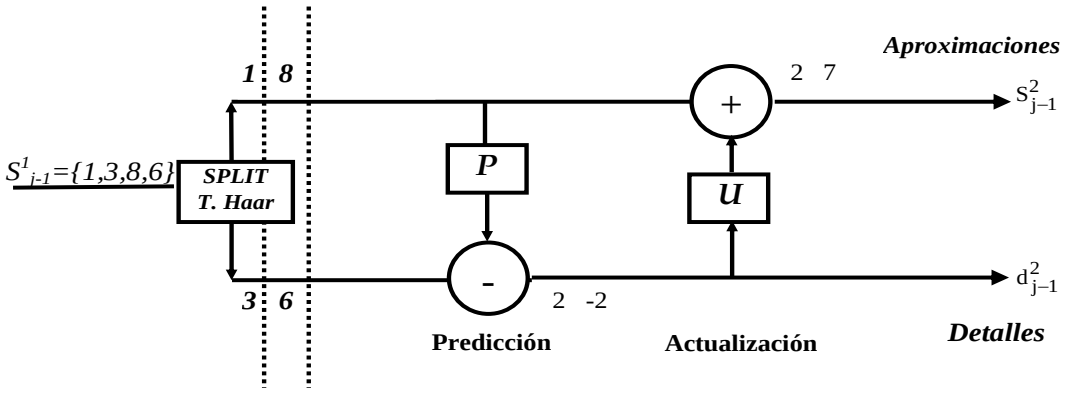


Figura 3.81. Señal resultante del primer nivel de descomposición

Los valores de detalles y aproximaciones son guardados *in-place* reemplazando en el array a los valores originales de la imagen pixel[0]=2, pixel[1]=2, pixel[2]=7 y pixel[3]=-2**.** En la segunda iteración *a* pasa a ser el valor de la aproximación $s^1_{j-1,0}$ y el *b* pasa a ser $s^1_{j-1,1}$. De esta forma se itera el algoritmo con los nuevos valores de *a* y *b*, manteniéndose los valores de detalles. A continuación se procede al segundo y último nivel ya que son 4 píxeles (n/2 niveles).

2ª Iteración: $a = 2$ y $b = 7 \rightarrow \begin{cases} b = b - a = 7 - 2 = 5 \\ a = a + \frac{b}{2} = 2 + \frac{5}{2} = 4{,}5 \end{cases}$

Pixel[0]	Pixel[1]	Pixel[2]	Pixel[3]
a	b	a	b
~~1~~	~~3~~	~~8~~	~~6~~
$S^1_{j-1,0}$	$d^1_{j-1,0}$	$S^1_{j-1,1}$	$d^1_{j-1,1}$
2	2	7	-2
4,5	2	5	-2
$S^2_{j-1,0}$	$d^2_{j-1,0}$	$d^2_{j-1,0}$	$d^2_{j-1,1}$

La señal resultante está formada por los elementos de la nueva señal de aproximación y los de detalle conservando su lugar *in-place*. S^1_{j-1}= {4,5, 2, 5, -2} y los valores reemplzan a los píxeles en sus respectivos índices pixel[0]=4,5, pixel[1]=2, pixel[2]=5 y pixel[3]=-2. La Fig. 3.82 muestra los resultados del segundo nivel de descomposición de la señal original.

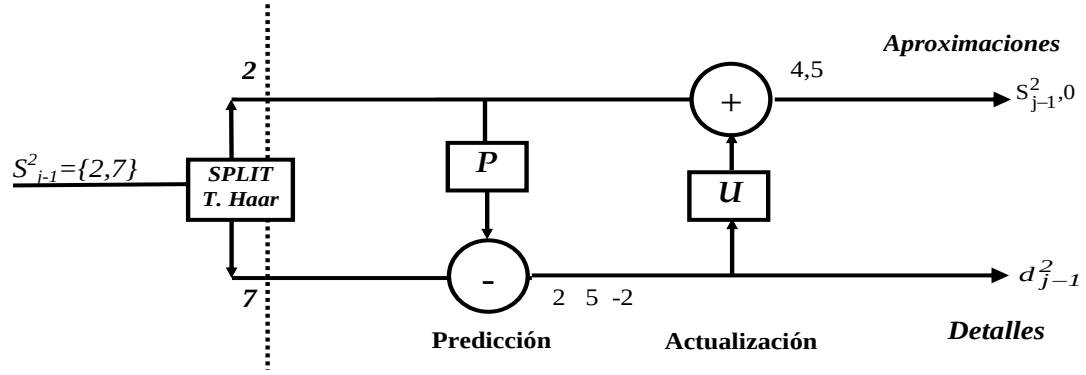


Figura 3.82. Señal resultante del último nivel de descomposición

La actualización en Haar preseva la recuperación de la señal mediante la media de todos los componentes de la imagen original. Obsérvese que píxel[0]= 4,5 y este valor se corresponde con la media de todos píxeles originales, m= (1+3+8+6)/4= 4,5, y marca el final de los niveles de descomposición de la señal n=4/2=2.

En la Fig. 3.83 se refleja el orden y colocación *in-place* de los valores.

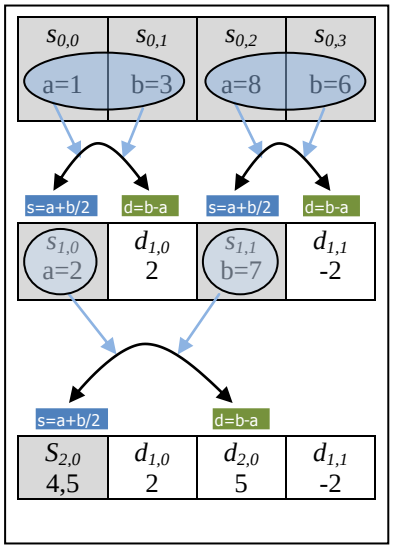


Figura 3.83. Predicción S_i para n iteraciones

La problemática que surge al aplicar la Transformada de Haar es que necesita más información para que el proceso sea totalmente reversible. Esto es, necesita un bit de signo y una representación con decimales.

Durante todo el proceso de compresión se utiliza un único conjunto de arrays porque los datos son recolocados durante la ejecución *in-place* en el array original de la imagen.

Por último, los valores de la señal s_{j-1}^{n} son codificados mediante el Codificador Huffman o por el Codificador Aritmético. Obviamente, el fichero comprimido debe incorporar esta información para que el decodificador aplique el mismo método que en el proceso de descompresión.

3.7.3.3 Proceso de descompresión

El proceso de descompresión comienza recuperando los valores de las señales de aproximación y detalles mediante el proceso de descompresión del método utilizado en el proceso de avance de la transformada de elevación.

El siguiente paso consiste en invertir los signos de operadores aritméticos usados en el proceso de avance y aplicar la transformada inversa de Haar. La secuencia de sucesivas transformaciones y suma de las señales de aproximación y

detalles finaliza cuando el valor de la señal es igual a la media de todos los elementos o bien igual a $n/2$ (Said y Pearlman, 1996). La Fig. 3.84 ilustra el proceso de reconstrucción reversible de la señal.

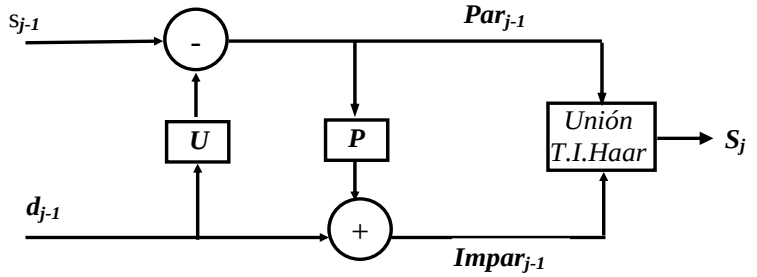


Figura 3.84. Diagrama de bloques del proceso de reconstrucción

La imagen puede recuperarse a partir de las S_{n-1} medias y d_{n-1} diferencias de forma recursiva uniendo las medias y sus diferencias mediante la transformada inversa (Dogiwal, Shishodia y Upadhyaya, 2014). La Fig. 3.85 ilustra el uso de la transformada inversa mediante la estructura de medias y diferencias.

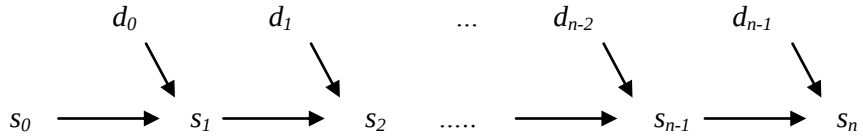


Figura 3.85. Estructura de la transformada wavelet inversa de Haar

El proceso reversible consiste en la secuencia inversa del proceso de avance: actualizar, predicción y suma de las señales de aproximación y detalles para recuperar los datos datos.

Operación de Actualización

La operación de actualización reconstruye los valores promediados para garantizar la recuperación de la señal. La Fig. 3.86 muestra el proceso inverso de la actualización.

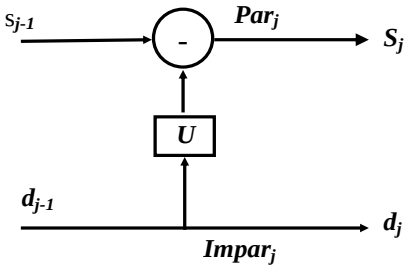


Figura 3.86. Estructura de la transformada wavelet inversa de Haar

Las operación de actualización inversa finaliza cuando el primer componente de la señal de aproximación es igual a la media de todos los componentes recuperados de la señal, o sea, el número de iteraciones es igual a *n*/2.

Operación de Predicción

Una vez verificada y procesada la operación de actualización realiza la operación de predicción inversa. La Fig. 3.87 muestra el diagrama del proceso de predicción inverso.

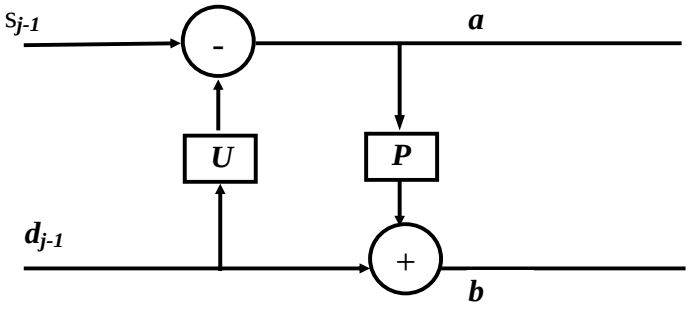


Figura 3.87. Operación inversa de predicción

La ecuación general de predicción del proceso reversible de *Lifting Scheme* es definida con las siguientes expresiones:

$$s_j = s_{j-1} - \frac{d_{j-1}}{2} \tag{3.67}$$

$$d_j = d_{j-1} + s_{j-1} \tag{3.68}$$

La transformada *wavelet* inversa de Haar ejecuta las operaciones de avance en orden inverso facilitando de nuevo los cálculos y la recolocación de los datos *in-place.*

$$\mathrm{a} = \mathrm{a} - \frac{\mathrm{b}}{2} \tag{3.69}$$

$$\mathrm{b} = \mathrm{b} + \mathrm{a} \tag{3.70}$$

Operación de Unión:

Una vez que los dos componentes de los conjuntos pares e impares han sido reconstruidos, los valores de la fila son recuperados mediante la unión de ambos componentes. Esta operación mueve los valores promediados y sus diferencias dentro de las localizaciones de los conjuntos *Impar* y *Par* y son unidos en la localización exacta S_j sin añadir espacio extra de memoria.

3.7.3.3.1 Algoritmo de descompresión lifting

El algoritmo reversible *lifting* contiene los siguientes pasos:

Paso 1. Comienza con los valores residuales del último nivel y los valores residuales
Paso 2. Reconstruye el componente par (e) mediante la operación inversa siguiente:

$$e = c + U[r]$$

Paso3. Reconstruye el componente impar mediante la siguiente operación

$$o = r + P[e]$$

Paso 4. Combina los componentes pares e impares para generar el dato en la escala o nivel previo y repite la secuencia de pasos

3.7.3.3.2 Programación del algoritmo inverso Lifting

La reconstruccion de los datos consiste en las mismas operaciones del proceso de avance invirtiendo los signos de los operadores.

```
// Lifting Scheme proceso iverso
for (i=0; i=n; i++)
{
    array(i)= datoc;   // carga valor del fichero comprimido
}
for (i=0; i=n/2; i+1){
for (a=0; a=n; a+2)
{
    a=array(a);
    b= array(a+1);
    a = a-(b/2);
    b = b+a;
    array(inda)= a;
    array(indb)= b;
}
}
```

Listado 3.17. Programación del algoritmo inverso Lifting Scheme

3.7.3.3.3 Ejemplo práctico

Considerando los resultados del ejercicio propuesto en el proceso de avance, los datos originales son recuperados aplicando el algoritmo inverso. Los datos están almacenados en el array píxel. Dados los cuatro valores comprimidos, el numero de iteraciones serán n/2.

ixel[0]	Pixel[1]	Pixel[2]	Pixel[3]
a	b	a	b
4,5	2	5	-2
$S^2_{j-1,0}$	$d^2_{j-1,0}$	$d^2_{j-1,0}$	$d^2_{j-1,1}$

La Fig. 3.88 muestra la operación de actualización inversa.

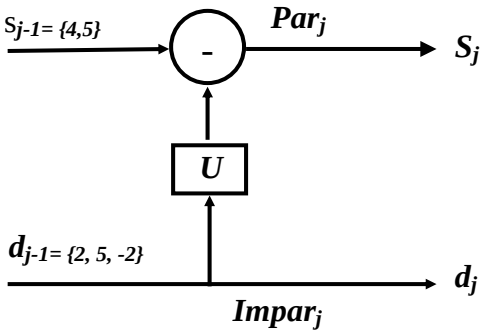


Figura 3.88. Operación inversa de predicción nivel j-1

Aplicando las ecuaciones (3.69) y (3,70) se reconstruye el primer nivel de descomposición de la señal:

1ª Iteración: $a = 4{,}5$ y $b = 5 \rightarrow \left\{ \begin{matrix} a = a - \frac{b}{2} = 4{,}5 - \frac{5}{2} = 2 \\ b = b + a = 5 + 2 = 7 \end{matrix} \right\}$

Píxel[0]	Píxel[1]	Píxel[2]	Píxel[3]
a	b	a	b
2	2	7	-2
$S^1_{j-1,0}$	$d^1_{j-1,0}$	$S^1_{j-1,1}$	$d^1_{j-1,1}$
~~4.5~~	~~2~~	~~5~~	~~-2~~
$S^2_{j-1,0}$	$d^2_{j-1,0}$	$d^2_{j-1,0}$	$d^2_{j-1,1}$

La señal resultante es la suma de la señal de aproximación y la señal de detalle S_{j-1}= {2, 2, 7, -2}. En la Fig. 3.89 se muestran los resultados de primera descomposición de la señal S_j.

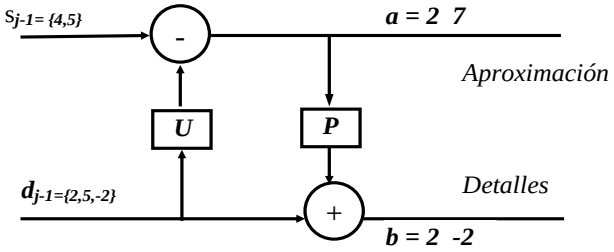


Figura 3.89. Operación inversa de predicción nivel 1

Siguiente nivel de descomposición de la señal:

2ª Iteración: $a = 2$ y $b = 2 \rightarrow \left\{ \begin{matrix} a = a - \frac{b}{2} = 2 - \frac{2}{2} = 1 \\ b = b + a = 2 + 1 = 3 \end{matrix} \right\}$

2ª Iteración: $a = 7$ y $b = -2 \rightarrow \left\{ \begin{matrix} a = a - \frac{b}{2} = 7 - \frac{(-2)}{2} = 8 \\ b = b + a = -2 + 8 = 6 \end{matrix} \right\}$

Píxel[0]	Píxel[1]	Píxel[2]	Píxel[3]
a	b	a	b
1	3	8	6
~~2~~	~~2~~	~~7~~	~~-2~~
~~4.5~~	~~2~~	~~5~~	~~-2~~

La señal resultante es la unión de la señal de aproximación y la señal de detalle Sj= {1,3,8,6}. En la Fig. 3.90 se muestran los resultados de la recuperación de la señal original Sj.

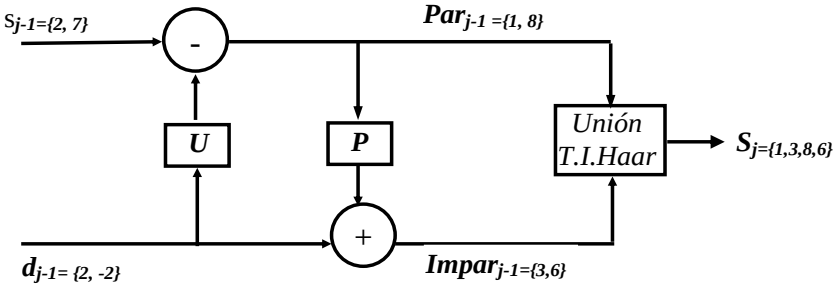


Figura 3.90. Operación inversa de predicción nivel 2

El proceso inverso recupera los datos de forma reversible. Aplicando la transformada de Haar debemos tener en cuenta que el promedio con decimales debe tenerse en cuenta al igual que los valores negativos. Por tanto, requiere usar dos bits adicionales: un bit de signo (bit más significativo del byte - *MSB*) y un bit de marca de decimales.

3.7.4 Resultados experimentales

La Tabla 3.54 contiene los resultados correspondientes a la Transformada de Haar (modelo *wavelet* estático) y a *Lifting Scheme* (modelo *wavelet* adaptativo).

Tabla 3-54. Resultados experimentales de la transformada Haar y Lifting Scheme

IMAGEN FUENTE	NOMBRE IMAGEN	TRANSFORMADA DE HAAR (DWT)	LIFTING SCHEME
		RATIOS DE COMPRESIÓN	RATIOS DE COMPRESIÓN
Fotográficaas	Airplane	1,363	1,888
	Baboon	1,079	1,249
	Barbara	1,172	1,428
	Boats	1,184	1,467
	Cameraman	1,236	1,693
	Goldhill	1,320	1,503
	Lena	1,326	1,717
	Man	1,261	1,920
	Peppers	1,204	1,656
	Zelda	1,283	1,687
	Media	**1,243**	**1,621**
Aéreas y Satélite	Aerial	1,223	1,408
	Airfield	1,093	1,329
	Earth	1,019	1,319
	Meteosat	1,221	1,469
	Moon	1,667	2,480
	Moonsurface	1,173	1,524
	SanDiego	1,098	1,370
	WashingtonIR	1,007	1,147
	Media	**1,188**	**1,506**
Creadas por Ordenador	Circles	10,331	16,093
	Gray	97,881	146,629
	Slope	2,304	5,415
	Squares	79,572	141,302
	Media	**47,522**	**77,360**
Médicas	Elbowx	2,769	3,848
	Finger	1,065	1,108
	MRI_Brain1	1,126	1,640
	MRI_Brain2	1,652	2,653
	MRI_Brain3	1,189	1,457
	Shoulder	1,389	1,928
	Media	**1,532**	**2,106**
Textos y Gráficos escaneados	Mercados1	2,403	3,284
	Mercados2	2,529	3,183
	Mercados3	2,178	2,821
	Mercados 4	2,652	3,135
	Media	**2,441**	**3,106**

En la Fig. 3.91 se refleja significativamente que la transformada de *Lifting Scheme* consigue mejores resultados que la Transformada Haar, como es típico para un método adaptativo frente a uno estático. La Transformada Haar debe almacenar bits adicionales para el signo y para determinar si la posición es par o impar, lo que degrada su ratio de compresión.

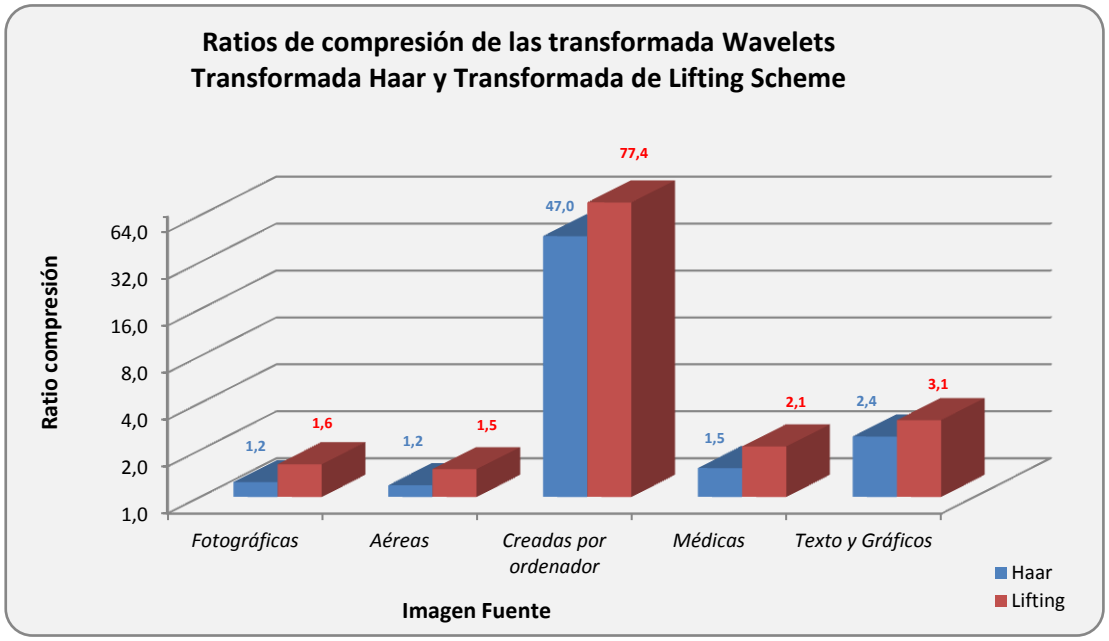


Figura 3.91. Gráfica comparativa de los ratios obtenidos por la transformada Haar y la transformada Lifting Scheme en una escala vertical de (0-3)

En términos de recursos de memoria, el algoritmo *Lifting Scheme* recoloca los datos en la imagen original, por tanto los consumos de memoria son insignificativos y su ejecución se realiza íntegramente en la memoria RAM. Al igual que los métodos estándares, este método aconseja la utilización de uno de los dos codificadores estadísticos: Aritmético o Huffman. En esta etapa de codificación es donde se producen los consumos más elevados.

En cuanto al consumo de recursos mostrado en la Tablas 3.55 y 3.56 se puede ver que el tiempo de ejecución es significativamente menor en el método *Lifting Scheme* que en el de Transformada de Haar. Esta situación sitúa al *Lifting Scheme* como el mejor método de compresión basado en *wavelets*.

Tabla 3-55. Transformada de Haar. Tiempos de ejecución del ciclo de compresión y descompresión

IMAGEN FUENTE	NOMBRE IMAGEN	CICLO DE COMPRESIÓN		CICLO DE DESCOMPRESIÓN	
		CICLOS CPU	TIEMPO (ms)	CICLOS CPU	TIEMPO (ms)
Fotográficaas	Airplane	1.458.023	660,20	1.409.084	638,04
	Baboon	1.304.624	590,74	1.243.825	563,21
	Barbara	1.487.108	673,37	1.407.096	637,14
	Boats	1.178.543	533,65	1.129.383	511,39
	Cameraman	852.508	386,02	793.388	359,25
	Goldhill	1.210.080	547,93	1.152.086	521,67
	Lena	1.152.903	522,04	1.119.820	507,06
	Man	286.260	129,62	246.618	111,67
	Peppers	1.142.611	517,38	1.110.279	502,74
	Zelda	1.572.443	712,01	1.543.446	698,88
	Media	**1.164.510**	**527,30**	**1.115.502**	**505,105**
Aéreas y Satélite	Aerial	1.285.587	582,12	1.236.603	559,94
	Airfield	1.205.287	545,76	1.159.837	525,18
	Earth	200.638	90,85	163.183	73,89
	Meteosat	4.741.313	2146,89	4.581.708	2.074,62
	Moon	17.854.090	8084,42	17.620.347	7.978,58
	Moonsurface	1.196.012	541,56	1.153.631	522,37
	SanDiego	5.642.363	2554,89	5.498.505	2.489,75
	WashingtonIR	28.554.481	12929,61	28.466.916	12.889,96
	Media	**7.584.971**	**3.434,51**	**7.485.091**	**3.389,29**
Creadas por Ordenador	Circles	313.005	141,73	291.715	132,09
	Gray	1.433.730	649,20	1.390.908	629,81
	Slope	360.354	163,17	340.919	154,37
	Squares	339.793	153,86	311.414	141,01
	Media	**611.721**	**276,99**	**583.739**	**264,32**
Médicas	Elbowx	1.496.428	677,59	1.353.430	612,84
	Finger	322.722	146,13	302.382	136,92
	MRI_Brain1	305.628	138,39	282.638	127,98
	MRI_Brain2	2.788.662	1262,72	2.660.240	1.204,57
	MRI_Brain3	1.655.216	749,49	1.582.624	716,62
	Shoulder	3.065.934	1388,27	2.892.703	1.309,83
	Media	**1.605.765**	**727,10**	**1.512.336**	**684,79**
Textos y Gráficos escaneados	Mercados1	8.638.753	3911,67	8.344.189	3.778,29
	Mercados2	9.039.279	4093,03	8.765.894	3.969,24
	Mercados3	4.730.271	2141,89	4.497.875	2.036,66
	Mercados4	4.722.873	2138,54	4.521.108	2.047,18
	Media	**6.782.794**	**3.071,28**	**6.532.267**	**2.957,84**

Tabla 3-56. *Lifting Scheme*. Tiempos de ejecución del ciclo de compresión y descompresión

IMAGEN FUENTE	NOMBRE IMAGEN	CICLO DE COMPRESIÓN		CICLO DE DESCOMPRESIÓN	
		CICLOS CPU	TIEMPO (ms)	CICLOS CPU	TIEMPO (ms)
Fotográficaas	Airplane	871.811	394,76	813.706	368,45
	Baboon	728.018	329,65	691.821	313,26
	Barbara	909.597	411,87	865.141	391,74
	Boats	786.674	356,21	732.876	331,85
	Cameraman	331.048	149,90	322.214	145,90
	Goldhill	636.985	288,43	605.780	274,30
	Lena	732.943	331,88	690.540	312,68
	Man	302.338	136,90	294.763	133,47
	Peppers	675.324	305,79	640.011	289,80
	Zelda	689.922	312,40	644.030	291,62
	Media	**666.466**	**301,78**	**630.088**	**285,31**
Aéreas y Satélite	Aerial	683.076	309,30	636.014	287,99
	Airfield	710.659	321,79	667.219	302,12
	Earth	256.667	116,22	234.971	106,40
	Meteosat	1.198.861	542,85	1.112.700	503,84
	Moon	5.257.453	2.380,60	4.901.026	2.219,21
	Moonsurface	660.395	299,03	605.365	274,11
	SanDiego	2.213.956	1.002,49	2.069.642	937,14
	WashingtonIR	11.129.851	5.039,65	10.421.320	4.718,82
	Media	**2.763.865**	**1.251,49**	**2.581.032**	**1.168,70**
Creadas por Ordenador	Circles	236.835	107,24	220.457	99,82
	Gray	357.682	161,96	322.205	145,90
	Slope	454.920	205,99	423.856	191,92
	Squares	308.234	139,57	291.366	131,93
	Media	**339.418**	**153,69**	**314.471**	**142,39**
Médicas	Elbowx	523.537	237,06	478.917	216,86
	Finger	340.257	154,07	322.285	145,93
	MRI_Brain1	262.961	119,07	242.338	109,73
	MRI_Brain2	271.971	123,15	259.140	117,34
	MRI_Brain3	327.757	148,41	296.631	134,32
	Shoulder	379.744	171,95	350.747	158,82
	Media	**351.038**	**158,95**	**325.010**	**147,17**
Textos y Gráficos escaneados	Mercados1	1.671.537	756,88	1.571.732	711,69
	Mercados2	1.920.629	869,67	1.800.096	815,09
	Mercados3	1.354.844	613,48	1.257.160	569,25
	Mercados4	884.421	400,47	798.075	361,37
	Media	**1.457.858**	**660,13**	**1.356.766**	**614,35**

3.7.5 Conclusiones

En esta sección se han presentado los métodos de compresión basados en las *wavelets* e implementados con la Transformada de Haar y con su modelo adpatativo mediante *elevación* (método *Lifting Scheme*). Los ratios de compresión de ambos métodos son comparables, pero en todos los tipos de imagen el método adaptativo *Lifting Scheme* obtiene un rendimiento superior al de Haar. Esta situación es idéntica a la observada en todos los métodos anteriores que contaban con modelo estático y modelo adaptativo.

Ambos métodos se caracterizan por pasar del plano espacial al plano de frecuencia mediante las *wavelet* y su implementación en Transformada de Haar. Este método se fundamenta matemáticamente en la Transformada de Fourier y en la DCT, pero algorítmicamente se implementa mediante operaciones matriciales básicas. La clave de los métodos –de *Haar* y *Lifting Scheme*- reside en que se basa en la utilización de matrices que cumplen la propiedad de ser ortogonales (transpuesta=inversa) y sus valores son cero o potencia de dos. Su transformada, por tanto, se ejecuta principalmente en operaciones con enteros y operaciones de desplazamiento, todas ellas de bajo consumo de memoria y tiempo o ciclos CPU.

Los métodos Transformada de Haar y *Lifting Scheme* tienen rendimientos superiores a los métodos predictivos vistos en la sección anterior. Desde el punto de vista de los estándares, si la estrategia predictiva soporta el JPEG *lossless*, el JPEG 2000 no es más que la actualización del anterior integrando el uso de *wavelets*. Sin embargo ninguno de los dos estándares, JPEG *lossless* y JPEG 2000, han acabado por constituir un estándar apreciado por su rendimiento excepto en algunos ámbitos de aplicación.

En la siguiente sección se revisa el método JPEG-LS que es el único estándar de compresión de imágenes digitales sin pérdida de datos desarrollado por el comité de estandarización utilizando exclusivamente *wavelets* y sus transformadas.

3.8 JPEG-LS propuesto como estándar de compresión reversible de imágenes

El estándar anterior JPEG del año 1991 en su versión sin perdida de datos es ampliamente reconocido y popular en técnicas de compresión *lossy*. JPEG *lossless* es una versión extendida del estándar y aunque es sencillo de implementar los resultados no están cerca del estado-del-arte de la compresión sin pérdida de datos. En 1995, el comité JPEG decidió desarrollar un nuevo estándar internacional para cubrir este vacío. El candidato propuesto desde 1997 es un nuevo estándar internacional de compresión en modo *lossless* y cerca-*lossless* para imágenes de continuos tonos conocido como JPEG-LS (ISO/IEC 14495, 2000). Los datos

comprimidos se almacenan en un nuevo formato grafico estándar denominado JLS creado especialmente para este propósito. El núcleo del nuevo estándar está basado en el algoritmo LOCO-I (*Low Complexity Lossless Algorithm*) desarrollado en los laboratorios de Hewlett-Packard y presentado en IEEE Data Compression Conference(Weinberger, Seroussi y Sapiro, 1996).

JPEG-LS es un método de predicción adaptativa basado en un modelo estadístico de contexto del píxel que aplica la codificación de Golomb-Rice (Golomb, 1966) (Wang y cols., 2012). Este método incorpora una importante novedad que consiste en un analizador de textura de regiones que permite codificar los píxeles en modo longitud de carrera o *Run-Length*. Otra característica sobresaliente y que heredera del algoritmo LOCO-I es la relación de correspondencia entre la compresión y su complejidad.

El más reciente estándar de compresión de imágenes es conocido como JPEG-2000 creado por el *Joint Photographic Experts Group* en el año 2000 (ISO/IEC 15444-1, 2000) (Skodras, Schelkens y Ebrahimi, 2009). En principio se pensó que podría arrebatar a JPEG-LS su reinado en la compresión de imágenes reversible. Sin embargo, aunque JPEG-LS es anterior, es un método más simple y garantiza un máximo error por muestra que JPEG 2000 no puede garantizar (Jin y cols., 2013) (Qi, Li, Guichun y Nam, 2012). Por lo tanto, actualmente JPEG-LS sigue siendo la propuesta de estándar en compresión de imágenes de continuos tonos en color y escala de grises sin pérdida de datos (Savakis, 2002).

JPEG-LS representa el estado del arte en compresión de imágenes de continuos tonos por su baja complejidad y por necesistar mínimos requisitos de memoria (Wang y cols., 2012). Sin embargo, existen otros métodos de compresión como son los populares CALIC (Wu y Memon, 1996) y FELICS (Howard y Vitter,1993) que obtienen similares ratios de compresión. La organización internacional ISO recomienda la utilización de JPEG-LS por la relación entre la compresión y su complejidad.

El modelo del estándar JPEG-LS se compone de tres componentes fundamentales: bloque de predicción del píxel, bloque de determinación del contexto y error de predicción. El bloque de predicción se compone de un modelo estático de predicción del píxel y un componente adaptativo basado en su contexto. Una vez conocida la predicción del píxel, el codificador computa mediante un modelo probabilístico la predicción residual como la diferencia entre el valor del píxel y su predicción condicionada por el contexto. Por ultimo, el codificador opera en dos modos: modo regular (*Golomb Code*) o en modo ejecución o longitud de carrera (*Run Length*).

La Fig. 3.92 muestra el modelo y los principales componentes del estándar JPEG-LS.

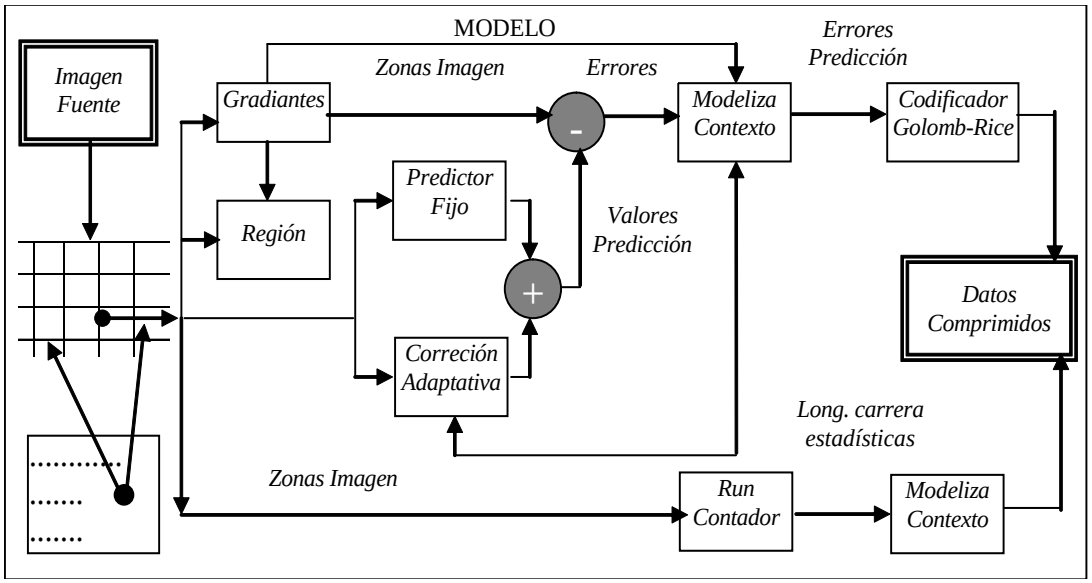


Figura 3.92. Diagrama de bloques del modelo del estándar JPEG-LS

El bloque de predicción del píxel utiliza un modelo dinámico de predicción lineal. El anterior estándar, JPEG *lossless*, utiliza un modelo estático de predicción seleccionando uno de los ocho predictores propuestos antes de comenzar el ciclo de compresión o descompresión. Por el contrario, JPEG-LS utiliza un predictor dinámico aplicando un algoritmo de detección de bordes horizontales o verticales sobre un contexto fijo de píxeles procesados anteriormente.

El boque del contexto utiliza un modelo adaptativo seleccionando un modelo probabilístico para calcular la precisión residual condicionada en el contexto del píxel.

3.8.1 Proceso de compresión

El ciclo de compresión puede realizar la compresión en uno de los dos modos propuestos: modo carrera o en modo regular. En un primer modo, el estándar, basándose en el método RLE, codifica los píxeles en modo carrera cuando identifica una secuencia de píxeles repetidos y consecutivos. En un segundo modo, el estándar toma ventaja de la propiedad de vecindad de los píxeles y aplica el modo regular que consiste en un algoritmo de predicción del píxel en base al valor de los píxeles adyacentes conocidos. Al conjunto de estos píxeles se le denomina contexto. El algoritmo obtiene un valor residual o diferencia entre el valor predecido y el valor del píxel. El resultado de esta diferencia es un valor cercano a cero y aplicando la codificación de Golomb-Rice obtiene los códigos de salida.

La Fig. 3.93 ilustra el funcionamiento del proceso de compresión mediante diagrama de bloques.

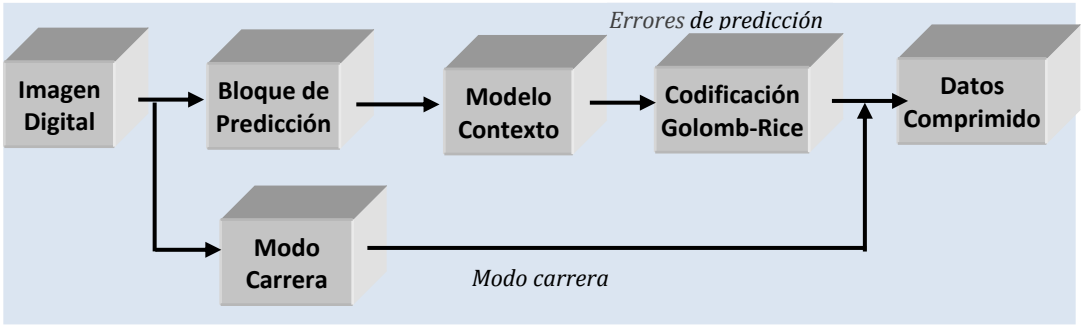


Figura 3.93. Diagrama de bloques simplificado del ciclo de compresión

A continuacións se describen las principales operaciones del estándar aquí descrito.

BLOQUE DE PREDICCIÓN

JPEG-LS analiza los valores de los píxeles adyacentes procesados previamente para determinar el valor inicial de predicción del píxel actual. Estos valores determinan el contexto del píxel a predecir. El modelo explota la propiedad de vecindad de cuatro píxeles adyacentes para definir un contexto.

Según se ilustra en la Fig. 3.94, la predicción de un símbolo x se basa en el análisis de los píxeles adyacentes a, *b, c y d* que determinan el contexto del píxel actual x. Los píxeles *a*, *b*, *c,* y *d* han sido procesados anteriormente y por tanto el codificador conoce sus valores.

	c	**b**	**d**		
	a	*x*	y	z	v

Figura 3.94. Contexto de predección del píxel x

El contexto del píxel se utiliza para predecir el píxel actual y establecer el modo de codificarlo, en modo ejecución o en modo regular.

- Modo ejecución: los píxeles adyacentes son iguales al píxel actual.
- Modo regular: los píxeles adyacentes son diferentes del actual.

Si el contexto indica que los siguientes píxeles *y, z, v* son iguales al píxel x entonces el codificador selecciona el modo ejecución. En este modo, el codificador se inicia en el actual píxel x y registra la carrera más larga de los píxeles iguales al actual. El límite máximo de la carrera es el último píxel de la fila procesada. En este modo, el modelo salta el procedimiento de cálculo de error de predicción y de codificación.

La salida del codificador será un código que registra la longitud de la carrera o *run length*.

Cuando los píxeles del contexto son diferentes, el contexto determina que debe utilizarse el modo regular y entonces usa *a, b,* y *c* para obtener la predicción inicial del píxel *x* mediante un predictor y garantizando la recuperación de los datos.

El predictor es determinado dinámicamente mediante un algoritmo simple de detección de bordes. La predicción es diseñada para explotar un análisis de textura de los píxeles del contexto. Cuando un borde es detectado entre los tres píxeles, el píxel que no está en el borde es asignado como el valor predictivo. Esto es, si existe un eje vertical a la izquierda del píxel a predecir, toma como valor el píxel *b*, en caso de que exista un eje horizontal sobre el píxel actual, toma el valor del píxel *a*, y en el caso de no detectar ningún borde, es calculado mediante la expresión (*a*+*b*-*c*).

$$y = \begin{Bmatrix} \min(a,b), & \text{Si } c \geq \max(a,b) \\ \max(a,b), & \text{Si } c \geq \min(a,b) \\ a + b - c & \end{Bmatrix} \tag{3.71}$$

En el siguiente ejemplo se ilustra el funcionamiento y resultado de predicción del píxel en modo regular (Fig. 3.95).

	18	**15**	*d*
	17	*x*	

Borde vertical detectado

C=18 > max(17,15)

Predictor=min(17,15)

P_x=15

	222	**223**	*d*
	225	*x*	

Borde horizontal detectado

c=222 <max(225,223)

Predictor=max(225,223)

P_x=225

	80	120	d
	60	*x*	

Sin bordes

60<80<120

Pred.= 60+80-120=20

P_x=20

Figura 3.95. Predicción del píxel x por detección de bordes

En el caso especial de la primera fila, el codificador no dispone de información del contexto del píxel, entonces los píxeles *b*, *c* y *d* deben ser considerados con valor cero. Del mismo modo, si el píxel actual esta localizado al inicio o al final de cualquiera de las filas, el codificador sustituye el valor de *a* o *d* por el valor reconstruido de *b* (*Rb*) siendo cero si está en la primera fila y para *c* reconstruye el valor que fue asignado al píxel *a* cuando el primer píxel de la línea anterior fue codificado.

3.8.1.1 Algoritmo de predicción

Paso 1. ¿ c ≥ max(a,b) ?
Paso 2. Si -> Px= min(a,b)
Paso 2.1 Salta nuevo píxel
Paso 3. ¿ c ≤ min(a,b)?

Paso 3.1. Si -> Px= min(a,b)
Paso 3.2 Salta nuevo píxel
Paso 4. Px=a+b-c
Paso 5. Predecir nuevo píxel

3.8.1.2 Programación del algoritmo de predicción

El algoritmo de predicción es simple y rápido. La programación contiene únicamente operaciones de comparación y asignación minimizando de esta forma los recursos de memoria.

El Listado 3.18 muestra la programación del algoritmo de predicción del método JPEG-LS.

```
// Algoritmo JPEG-LS Predictor
if(c>=max(a,b))
      Px=min(a,b);
else
     if(c<=min(a,b))
            Px=max(a,b)
     else
      Px=a+bRc;
end if;
end if;
```

Listado 3.18. Programación del algoritmo de predicción

3.8.1.3 Modelo del contexto

El contexto de un píxel debe ser determinado previamente con los píxeles reconstruidos *Ra, Rb, Rc* y *Rd* correspondientes a los valores actuales de *a, b, c* y *d*. El modelo contexto presenta dos modelos probabilisticos posibles: por áreas de regiones y por áreas de bordes. En ambos casos, el primer paso consiste en determinar los valores de los gradiantes locales del contexto D_1, D_2 y D_3. Véase la expresión (3.59).

$$\begin{aligned} D_1 &= R_d - R_b \\ D_2 &= R_b - R_c \\ D_3 &= R_c - R_a \end{aligned} \tag{3.72}$$

En un segundo paso, cuantifica D_1, D_2 y D_3 comparando los tres gradiantes locales D_i con unos umbrales o parámetros de coeficientes positivos T_1, T_2 y T_3, los cuales pueden ser definidos por el usuario para calcular una región Q_i que a su vez está formada por tres regiones (Q_1, Q_2 y Q_3 respectivamente). Estas conforman un vector [Q_1, Q_2, Q_3] que representa el contexto del píxel actual. En consecuencia, existen nueve regiones de cuantificación conectadas por cada uno de los gradiantes Q_1, Q_2 y Q_3. Cada región Q_i puede tomar hasta nueve posibles valores enteros pertenecientes al intervalo [-4, +4] que supone $9^3 = 729$ diferentes números de regiones.

$$D_i \leq -T_3 \qquad Q_i = -4$$

$$-T_3 < D_i \leq -T_2 \qquad Q_i = -3$$
$$-T_2 < D_i \leq -T_1 \qquad Q_i = -2$$
$$-T_1 < D_i \leq 0 \qquad Q_i = -1$$
$$T_2 < D_i \leq T_3 \qquad Q_i = 3$$
$$T_1 < D_i \leq T_2 \qquad Q_i = 2$$
$$0 < D_i \leq T_1 \qquad Q_i = 1$$
$$T_3 < Q_i \qquad Q_i = 0$$

En caso de no cumplir ninguno de los casos anteriores

$$Q_i = 0$$

Figura 3.96. Cuantificación de los gradiantes

En imágenes de ocho bits de profundidad, las regiones de cuantificación por defecto son:

$$0\}, \pm\{1,2\}, \pm\{3,4,5,6\}, \pm\{7,8,9,..20\}, \pm\{>=21\}, \{...$$

En un tercer paso, mapea los números de las regiones Q_i a un entero Q en el intervalo [0, 364] usando la propiedad de simetría. La tupla (0,0,0) es mapeada a cero, y los restantes 728 tuplas son mapeados a [1, 728/2=364]. Las tuplas *(-a, -b,-c)* y *(+a,+b,+c)* son mapeadas con el mismo valor. El entero Q se convierte en el contexto para el píxel actual a predecir.

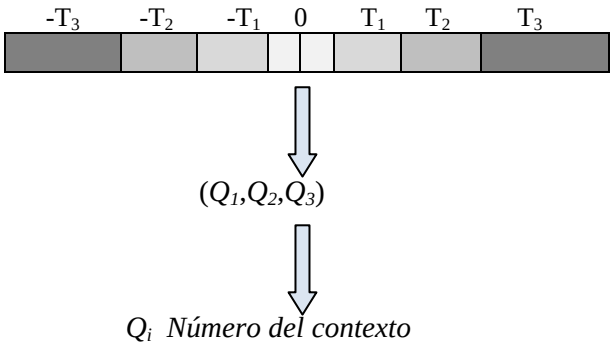


Figura 3.97. Número de contexto Q

Después de determinar el número del contexto Q, el codificador predice el píxel x en dos pasos. En un primer paso calcula la predicción del píxel actual, P_x, basándose en la regla de los bordes (Fig. 3.97) y en un segundo paso corrige y actualiza el valor de predicción.

```
if (sign=+1) then
     Px=Px+C[Q];
else
     Px=Px-C[Q];
end if;
if (Px>MaxVal) then
    Px=MaxVal;
else
    if (Px<0) then
        Px=0;

    end if;

end if;

```

Listado 3.19. Programación del contexto *Q* y actualización

La importancia del contexto reside en que la prediction residual de los píxeles que pertenecen a un mismo contexto tienen la misma distribución estadística.

PREDICCIÓN RESIDUAL (error de predicción)

Una vez predecido P_x en cada píxel x, el codificador obtiene la predicción residual. Este valor es calculado como la diferencia entre el valor real del píxel x y su valor predecido.

$$\varepsilon = x - P_x \tag{3.73}$$

En caso de resultar un error con valor negativo el signo es invertido. La inversión del signo se realiza reasignando valores positivos a los valores negativos usando una secuencia de superposición única y reversible.

El inicio de la secuencia es el siguiente: 0, -1, 1, -2, 2, -3, 3, -4, 4 ... . La expressión matemática de esta superposición se muestra a continuación:

$$M(\varepsilon) = \begin{cases} 2\varepsilon & \epsilon \geq 0, \\ 2|\varepsilon| - 1 & \varepsilon < 0 \end{cases} \tag{3.74}$$

CODIFICACIÓN

El estándar recomienda utilizar uno de los dos posibles modos de codificación en función del contexto: modo regular (códigos de Golomb) y modo ejecución (*run length*).

3.8.1.4 Codificación en modo ejecución

Cuando los gradiantes son iguales a cero significa que el píxel actual es igual a los adyacentes, entonces el codificador salta el proceso de predicciones y directamente codifica la salida con un código que identifica el número de repeticiones del píxel o carrera.

3.8.1.4.1 Algoritmo del codificador en modo ejecución

El codificador busca a partir del píxel actual *x*, una secuencia de píxeles consecutivos con valores idénticos a los valores reconstruidos.

Paso 1. Si los gradieantes son cero, el codificador introduce run-mode
Paso 2. $|x-b| <= \varepsilon$
Si
Paso 2.1. Incrementa contador en uno y registra contador
Paso 2.2.¿Fin de línea?
Si
Paso 2.2.1 Asigna a c el valor de a de la fila anterior
Paso 2.3. Retorna el run-length al paso 2
Paso 3. Salta a la predicción del siguiente píxel

3.8.1.5 Codificación en modo regular

El error de predicción una vez mapeado, se codifica mediante un procedimiento derivado de la codificación Golomb. Este es un sistema de codificación óptimo para codificar una distribución geométrica de números enteros no negativos sin utilizar tablas adicionales usando para ello únicamente un parámetro "*K*". La familia de códigos de los parámetros de *Golomb* siempre son potencia de dos. El procedimiento de codificación por Golomb dependerá del contexto que es determinado por los valores de las muestras en las posiciones *a*, *b, c,* y *d*, así como los errores de predicción previamente codificados por el mismo contexto.

Dado un parámetro entero positivo *n*, el método de codificación Golomb G_m codifica un entero en dos partes: una representación unitaria de *[n/m],* y una representación binaria modificada de *y mod m* (usando [*log m*] bits si $y<2^{[\log m]} - m$ y en caso contrario [*log m*] bits).

$$q = \left[\frac{n}{m}\right], \qquad r = n - q_m, \qquad c = [\log_2 m] \tag{3.75}$$

En el caso especial de códigos Golomb con $m = 2^k$ el codificador introduce un simple procedimiento de codificación y decodificación: el código de *y* es construido añadiendo *k* bits menos significativos de *y* a la representación unitaria del número formado por los restantes bits mas significativos de *y.* La longitud del código será $k + 1 + [y/2^k]$.

MSB	LSB
8-k bits(Unary)	*K bits (binario)*

Figura 3.98. Estuctura de representación del código Golomb

El siguiente paso consiste en determinar el parametro *k* del código Golomb adaptativamente. El parámetro *k* depende del contexto y el valor de *k* para un contexto es actualizado cada vez que un píxel es encontrado dentro de este contexto.

El parámetro *K* de la codificación Golomb-Rice es obtenido mediante la expresión (3.76).

$$k = \left\lceil \log_2 \frac{A(i)}{N(i)} \right\rceil \tag{3.76}$$

Donde, A[i]: suma acumulada de errores de predicción,
N[i]: número de predicciones residuales vistas en el contexto i.

Supongamos un valor de predicción residual del píxel igual a *14* y *k=2,* aplicando el codificador *Golomb Code* tenemos:

$14_{10} = 00001110_2$ -> k=2, entonces

MSB	LSB
8-k bits(Unitario) *000011*	*K=2, últimos bits* *10*

La representación unitaria resultante será: $000011_2 = 3_{10} = 000_{unitario}$

El código de Golomb para sistemas binarios representa el valor binario que incluye un bit de separación igual a uno, por tanto, la secuencia binaria corresponde a 000110.

3.8.1.5.1 Algoritmo de codificación en modo regular

El algoritmo del ciclo compresión en modo regular es el siguiente:

Paso 1. Inicialización
Paso 2. Computa los gradiantes locales D_1, D_2 y D_3
Paso 3. ¿ $D_1=D_2=D_3$?
Si
Paso 3.1 Tratamiento Run Length (algoritmo de codificación)
No
Paso 3.2 Tratamiento Modo Regular
Paso 4. Construye vector contexto Q, cambia a signo positivo si es negativo y mapea en rango [1,364]
Paso 5. Calcula la predicción residual y actualiza con el contexto del paso 4.
Paso 6. Computa el parámetro K de Golomb y mapea ε.
Paso 7. Codifica las predicciones residuales mediante Golomb usando el parámetro k.
Paso 8. Actualiza el contexto
Paso 9. Repite los pasos para los siguientes píxeles hasta finalizar la imagen.

3.8.1.6 Ejemplo práctico

El siguiente ejemplo está basado en una muestra de cuatro píxeles de la imagen del estándar y que se muestra en la Fig. 3.99. En este ejemplo se ilustra el principio de funcionamiento de los dos modos posibles de codificación: modo ejecución y modo regular.

Índices 0 1 2 3 4 5

0	0	0	0	0	0	0
1	0	0	0	90	74	74

Figura 3.99. Ejemplo de una zona de la imagen propuesta en el estándar

Los valores reconstruidos están reflejados por las zonas sombreadas R_b, R_c y R_d son los valores reconstruidos de la primera fila (índice fila=0) y R_a y R_c corresponden a la primera columna (índice columna=0) para el píxel x a predecir.

Inicialmente, el codificador procede a codificar los elemento de la fila [0,0,90].

$R_c=0$	$R_b=0$	$R_d=0$
$R_a=0$	$x=0$	

$$D_1=R_d-R_b=0, \quad D_2=R_b-R_c=0, \quad D_3=R_c-R_a=0$$

A partir del error residual se actualiza el contexto y remapea el error utilizando los códigos de Golomb dentro de un rango [0, M-1] y aplicando la expresión (3.77) se obtiene el parámetro k.

$$\begin{gathered} \mathrm{M}(\varepsilon) = (\mathrm{ErrVal}\ 2) \quad \mathrm{Signo} = 179 \\ \mathrm{k} = \left\lceil \log_2 \frac{\mathrm{A(i)}}{\mathrm{N(i)}} \right\rceil = 4 \\ \mathrm{Q} = \frac{(2\mathrm{T}+1)^3}{2} = 366, \quad \mathrm{T} = 4 \end{gathered} \tag{3.77}$$

ErrVal (mod) indica el valor de *ErrVal* después módulo división. Los parámetros antes de codificar se muestran en la siguiente tabla.

Run Val	Run Contador	Run Índice	ErrVal (mod)	TEMP	k	mapeo	ErrVal $M(\varepsilon)$	Q	A[Q]	N[Q]	Nn[Q]
0	2	0	90	4	4	0	179	366	4	1	0

ErrVal (ε) = 90_{10} = 1011010_2 => K=2

Valor binario: los 2 bits menos significativos, 10.
Valor Unitario: resto de píxeles más significativos $10110_2 = 22_{10}$. El valor unitario esta formado por 22 ceros consecutivos.
Salida de bits: 11 + (22 ceros de la codificación unitaria) + 1 + 10

El cálculo y actualización de la suma de errores $A[Q]$ y número de predicciones $N[Q]$ es obtenido aplicando las expresiones mencionadas anteriormente.

$$\begin{gathered} \mathrm{A[Q]} = \mathrm{A[Q]} + \mathrm{abs[ErrVal]} = 93 \\ N[Q] = N[Q] + 1 = 2 \end{gathered} \tag{3.78}$$

RUN ÍNDICE	A[Q]	N[Q]	Nn[Q]
1	93	2	0

El siguiente tratamiento corresponde al píxel con valor 74. El codificador utiliza el modo regular porque *Ra* es cero.

Rc=0	*Rb=0*	*Rd=0*
Ra=90	*x=74*	

$$D_1=R_d-R_b=0,\quad D_2=R_b-R_c=0,\quad D_3=R_c-R_a=90$$

A continuación, el algoritmo calcula la cuantificación de los gradiantes:

$$Q_1=0$$
$$Q_2=0$$
$$T_3 < Q_3 \rightarrow Q_i= 4$$

Los parámetros obtenidos antes de la codificación se muestran en la siguiente tabla.

Q1	Q2	Q3	Px	Signo	ε ErrVal	k	ErrVal $M(\varepsilon)$	A[Q]	A[Q]	N[Q]	Nn[Q]
0	0	4	90	-1	16	2	32	4	0	0	1

$$ErrVal\ (\varepsilon) = 16_{10} = 0001\ 0000_2 => K=2$$

Valor binario: los 2 bits menos significativos, 00.

Valor Unitario: resto de píxeles más significativos $100_2 = 4_{10}$.

El valor unitario esta formado por 4 ceros consecutivos.

Salida en bits: (4 ceros de la codificación unitaria) + 1(diferenciador) + 00 → 0000100

De nuevo, cálculo y actualización de la suma de errores $A[Q]$ y número de predicciones $N[Q]$ para el siguiente píxel:

$$A[Q] = A[Q] + abs[ErrVal] = 4 + 16 = 20$$
$$N[Q] = N[Q] + 1 = 2 \qquad (3.79)$$

A[Q]	B[Q]	C[Q]	N[Q]
20	0	1	2

El proceso continuaría con los siguientes píxeles. La codificación definitiva de estos cuatro píxeles es la siguiente:

Codificación Run Length (0,0,90) *M. Regular (74)*
11000000000000000000000000110 0000100
---------- 27 bits ---------- --7 bits-- = 34bits

El número total de bits resultante es 34, de los cuales cinco son unos y los veintinueve restantes son ceros. Luego la probabilidad de que sea un cero es 29/(29+5) ≈ 0,85. La mediana del *run length* de una codificación binaria es calculada aplicando la expresión (3.80).

$$\mathrm{m} = \left[-\frac{\log_2(1+\mathrm{p})}{\log_2 \mathrm{p}} \right] \tag{3.80}$$

Cuando la probabilidad *p* es muy alta y varía en función de *m* entonces garantiza unos muy buenos resultados mediante una codificación binaria por *Golomb Code*. Básicamente, la codificación de una cadena binaria puede ser comprimida con RLE atendiendo a los siguientes pasos:

(1) Contar el número de unos y ceros de la secuencia binaria.

(2) Calcular la probabilidad *p* de un bit cero

(3) Calcular el parámetro *m* aplicando la ecuación (3.80)

(4) Construir la familia de código Golomb para *m*.

(5) Para cada longitud de carrera de *n* ceros escribir el código Golomb de *n* en la salida comprimida.

En caso de probabilidades altas de un cero produce compresión. Cuando la probabilidad es baja pero la de un uno es muy alta se procede a codificar el *run length* en lugar de los ceros. Sin embargo, cuando la probabilidad de un uno y de un cero es similar no se recomienda este método porque no produce compresión significativa.

3.8.2 Proceso de descompresión

El ciclo de compresión y descompresión son prácticamente simétricos. El codificador y decodificador emplean las mismas reglas en orden inverso. El decodificador procesa el fichero comprimido *JLS* leyendo la cabecera del fichero para extraer la información necesaria para el tratamiento de los datos comprimidos almacenados secuencialmente en el fichero. El codificador realiza los siguientes pasos para el tratamiento de los datos codificados:

(1) Decodifica los códigos de predicciones residuales usando el decodificador *Golomb-Rice* o *Run Length*.

(2) Remapea los valores residuales

(3) Obtiene la predicción de error residual y actualize el contexto

(4) Refina la predicción considerando el context.

(5) Obtiene la predicción del píxel *x*.

(6) En base al contexto *Ra*, *Rb*, *Rc* y *Rd* recupera el valor del píxel *x*

(7) Repite el proceso desde el paso 1 hasta fin de fichero del fichero descomprimido

El proceso de descompresión se muestra en el siguiente diagrama de bloques simplificado (Fog. 3.100).

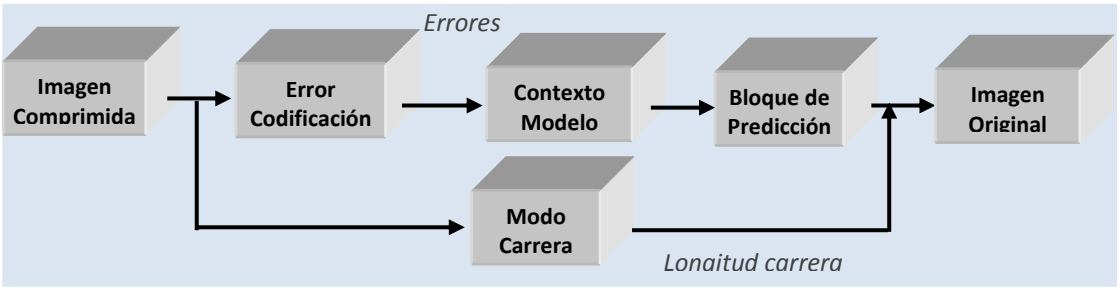


Figura 3.100. Diagrama de bloques del ciclo de descompresión JPEG-LS

Una vez calculada la predicción residual, el valor del píxel en el codificador se calcula con la siguiente expresión:

$$x = \varepsilon + P_x \tag{3.81}$$

El píxel actual *x* es recuperado y pasa a formar parte del contexto actual que será necesario utilizar para calcular los píxel adyacentes siguientes y pendientes de decodificar.

3.8.2.1 Algoritmo de descompresión

Paso 1. Lectura del código de salida
Paso 2. Actualiza los valores A[Q], B[Q] y C[Q] del contexto- modelo
Paso 3. Actualiza el contador del contexto añadiendo ε a B y |ε| a A reduciendo a la mitad A, B y C si N=reset e incrementar N,
Paso 4. Golomb Code decodifica la predicción residual usando el parámetro k y mapea M(ε).
Paso 5. Calcula la predicción residual y actualiza con el contexto.
Paso 6. Calcula la predicción fija
Paso7. Genera los gradiante cuantificados y construye el vector contexto Q, cambia a signo positivo si es negativo y mapea en rango [1,364]- El contexto nuimero cero está reservado para mode run)
Paso 8. Determina los gradiantes locales
Paso 9. Si D1=D2=D3=0, modo de tratamiento run. En caso contrario, tratamiento en modo regular
Paso 10. Calcula los gradiantes locales D1, D2 y D3.
Paso 11. Recupera el valor del píxel X y pasa a formar parte del contexto.
Paso 12. Inicializa variables, parámetros, contadores,
Paso 12. Vuelve al paso 1 hasta finalizar la imagen

3.8.3 Resultados Experimentales

La Tabla 3.57 muestra los ratios de compresión obtenidos por el estándar JPEG-LS sobre una muestra de imágenes estándar de ISO procedente de diferentes fuentes.

Tabla 3-57. Ratios de compresión del estándar JPEG-LS

IMAGEN FUENTE	NOMBRE IMAGEN	JPEG-LS RATIO DE COMPRESIÓN
Fotográficaas	Airplane	2,115
	Baboon	1,325
	Barbara	1,645
	Boats	1,668
	Cameraman	1,855
	Goldhill	1,698
	Lena	1,885
	Man	2,709
	Peppers	1,782
	Zelda	1,997
	Media	**1,868**
Aéreas y Satélite	Aerial	1,621
	Airfield	1,437
	Earth	1,630
	Meteosat	1,640
	Moon	2,916
	Moonsurface	1,574
	SanDiego	1,418
	WashingtonIR	1,229
	Media	**1,683**
Creadas por Ordenador	Circles	51,829
	Gray	132,077
	Slope	5,088
	Squares	101,335
	Media	**72,582**
Médicas	Elbowx	3,810
	Finger	1,170
	MRI_Brain1	2,097
	MRI_Brain2	3,027
	MRI_Brain3	1,826
	Shoulder	2,360
	Media	**2,381**
Textos y Gráficos escaneados	Mercados1	3,255
	Mercados2	2,825
	Mercados3	3,202
	Mercados4	3,529
	Media	**3,203**

Los resultados expuestos en la Tabla 3.57 demuestran que el método estándar JPEG-LS mejora los resultados de los métodos revisados anteriormente aproximándose a ratios de 2:1 en imágenes fotográficas y superándolo en el resto de tipo de imágenes excepto en las imágenes creadas por ordenador, en las cuales es superado por su estándar antecesor.

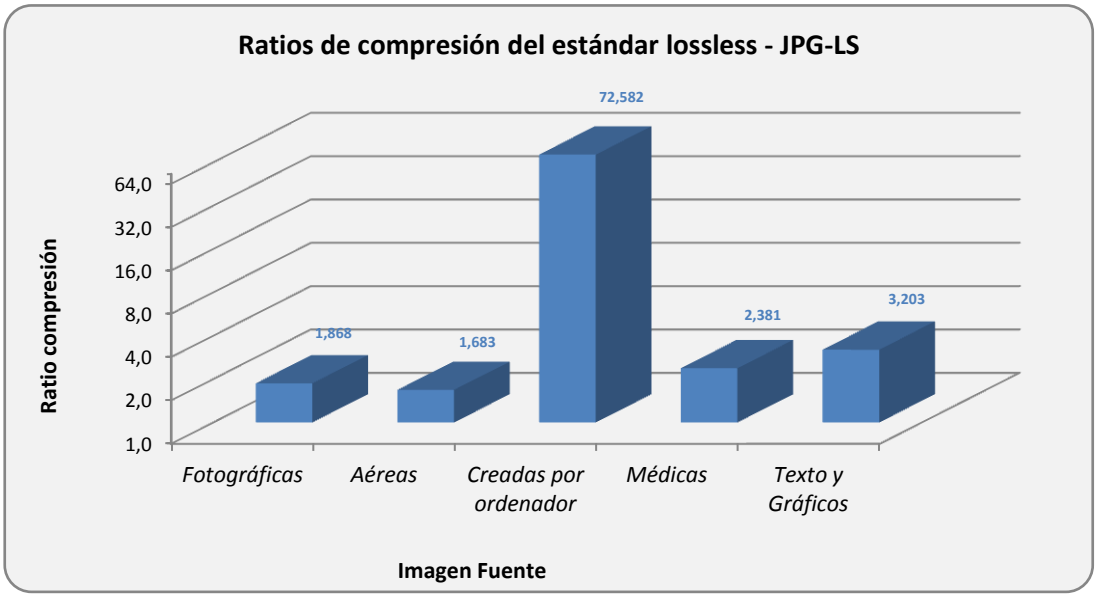


Figura 3.101. Ratios de compresión del estándar JPEG-LS

La Fig. 3.101 refleja la superación de las medias de compresión del factor 2:1 que supera al resto de los métodos revisados anteriormente. Aunque este método no garantiza los mejores resultados en todas las imágenes podemos concluir que es considerado como el estado del arte en la compresión de imágenes digitales sin pérdida de datos.

En términos de recursos de memoria, los consumos de memoria del estándar JPEG-LS son insignificantes y por supuesto, se ejecuta íntegramente en memoria principal RAM.

La Tabla 3.58 muestra los tiempos de ejecución de los ciclos de compresión y descompresión del estándar JPEG-LS. El contenido de esta tabla demuestra que los tiempos de ejecución alcanzados están cerca del estado del arte y mejora los tiempos del estándar anterior y del resto de los métodos revisados. Por otra parte, los tiempos de descompresión son infereiores a los tiempos de compresión aunque esta diferencia no es tan significativa como sucede en el resto de los métodos. Por último, desctacar que los tiempos de ejecución se mantienen casi constantes en función del tamaño de la imagen original.

Tabla 3-58. JPEG-LS. Tiempos de ejecución del ciclo de compresión y descompresión

IMAGEN FUENTE	NOMBRE IMAGEN	CICLO DE COMPRESIÓN		CICLO DE DESCOMPRESIÓN	
		CICLOS CPU	TIEMPO (ms)	CICLOS CPU	TIEMPO (ms)
Fotográficaas	Airplane	126.839	57,43	115.022	52,08
	Baboon	109.420	49,55	115.850	52,46
	Barbara	175.150	79,31	125.961	57,04
	Boats	110.641	50,10	135.613	61,41
	Cameraman	139.090	62,98	84.238	38,14
	Goldhill	174.699	79,11	163.319	73,95
	Lena	114.084	51,66	111.155	50,33
	Man	125.397	56,78	112.843	51,10
	Peppers	174.326	78,94	117.863	53,37
	Zelda	213.380	96,62	155.506	70,41
	Media	**146.303**	**66,25**	**123.737**	**56,03**
Aéreas y Satélite	Aerial	160.197	72,54	146.938	66,53
	Airfield	185.136	83,83	150.372	68,09
	Earth	146.415	66,30	79.325	35,92
	Meteosat	228.427	103,43	206.089	93,32
	Moon	717.444	324,87	595.293	269,55
	Moonsurface	154.664	70,03	84.427	38,23
	SanDiego	308.661	139,76	288.621	130,69
	WashingtonIR	979.358	443,46	1.027.736	465,37
	Media	**360.038**	**163,03**	**322.350**	**145,96**
Creadas por Ordenador	Circles	121.672	55,09	82.040	37,15
	Gray	120.910	54,75	98.694	44,69
	Slope	149.683	67,78	81.755	37,02
	Squares	130.444	59,07	77.288	35,00
	Media	**130.677**	**59,17**	**84.944**	**38,46**
Médicas	Elbowx	168.588	76,34	114.173	51,70
	Finger	136.560	61,84	90.018	40,76
	MRI_Brain1	143.047	64,77	90.394	40,93
	MRI_Brain2	126.276	57,18	84.059	38,06
	MRI_Brain3	135.128	61,19	89.187	40,38
	Shoulder	135.538	61,37	94.642	42,85
	Media	**140.856**	**63,78**	**93.746**	**42,45**
Textos y Gráficos escaneados	Mercados1	223.502	101,20	180.143	81,57
	Mercados2	242.019	109,59	211.945	95,97
	Mercados3	190.891	86,44	177.190	80,23
	Mercados4	150.734	68,25	111.498	50,49
	Media	**201.787**	**91,37**	**170.194**	**77,07**

3.8.4 Conclusiones del estándar JPEG-LS

El estándar JPEG-LS consigue ratios de compresión cercanos al estado-del-arte, entendido este como 2:1, con muy baja complejidad de ejecución, pocos requisitos de memoria y tiempo de ejecución insignificante comparado con otros métodos. Adicionalmente, la implementación del método es sencilla.

Estas ventajas aconsejan su utilización como método estándar de compresión sin pérdida de datos aplicado a imágenes digitales de continuos tonos dejando obsoleta a la versión *lossless* anterior y a la versión *lossless* incluida en el último estándar JPEG 2000.

3.9 Análisis comparativo de métodos de compresión de imágenes

Una vez implementados y analizados todos los métodos de compresión de imágenes reversibles sin pérdida de datos frente a un banco de imágenes representativo, es momento de comparar los resultados en términos de eficacia y eficiencia.

El rendimiento de un método se mide primariamente por el ratio de compresión, y secundariamente por el tiempo de ejecución. La compeljidad del algoritmo y de su implementación es una característica adicional.

La Tabla 3.59 resume mediante los valores medios los resultados obtenidos en este capítulo. Para cada método y tipo de imagen se anotan el valor medio del ratio de compresión y el tiempo en *ms* de la ejecución del algoritmo de compresión, prescindiendo de la velocidad de descompresión y de los ciclos CPU.

Los resultados detallados están descritos en la revisión de cada método, esta tabla busca un análisis global del rendimiento de los diferentes métodos con el fin de establecer el estado-del-arte con el que comparar los resultados del método INA desarrollado en la tesis.

Adicionalmente, la clasificación aportada en esta tesis (Sección 2.7) de los métodos de compresión sin pérdida de datos por el criterio del modelo utilizado nos ha permitido comprobar experimentalmente como el resultado y el comportamiento de cada método son extensibles a su categoría con repecto al tipo de imagen fuente.

La tabla 3.59 muestra los ratios de compresión y tiempos de ejecución de cada método en cada una de los tipos de imagen fuente: fotográfica, aéreas y satélite, creadas por ordenador, médicas y textos con gráficos contenidos en imágenes escaneadas. Las celdas sombreadas señalan los ratios y tiempos más relevantes y principales aportaciones de los métodos analizados en términos de resultados.

Tabla 3-59. Ratios de compresión y tiempos de ejecución de los métodos del estado del arte

	FOTOGRÁFICAS	AÉREAS Y SATÉLITE	CREADAS POR ORDENADOR	MÉDICAS	TEXTOS Y GRÁFICOS ESCANEADOS
Bitmap clásico 1 byte/píxel	1,006 455,38 ms	1,027 1.859,25 ms	3,236 123,30 ms	1,129 537,61 ms	1,801 788,15 ms
Bitmap fila	0,921 518,19 ms	0,949 2.003,92 ms	1,701 235,11 ms	1,041 316,58 ms	1,711 1.404,77 ms
Bitmap imagen	0,904 783,986 ms	0,942 1.733,19 ms	1,407 255,605 ms	1,019 251,75 ms	1,253 815,21 ms
RLE clásico	0,916 400,44 ms	0,990 2.252,35 ms	49,714 57,96 ms	1,083 178,61 ms	1,826 834,98 ms
RLE 4 bits	0,950 473,61 ms	1,081 2.785,25 ms	76,387 58,94 ms	1,245 205,62 ms	2,142 893,73 ms
Huffman estático	1,092 1.081,27 ms	1,134 4.963,28 ms	3,258 543,41 ms	1,268 591,04 ms	1,657 2.741,13 ms
Huffman adaptativo	1,082 923,04 ms	1,131 4.628,36 ms	2,976 440,42 ms	1,250 509,34 ms	1,605 1.632,82 ms
Codificador Aritmético Estático	1,100 189,50 ms	1,441 1.024,61 ms	3,748 152,04 ms	1,282 105,53 ms	1,621 504,06 ms
Codificador Aritmético Adaptativo	1,154 327,86 ms	1,494 2.172,16 ms	4,558 153,54 ms	1,325 220,53 ms	2,120 593,84 ms
LZ77 Estático	1,040 1.024,69 ms	1,474 2.808,12 ms	6,403 268,40 ms	1,330 263,35 ms	3,191 1.519,11 ms
LZW Adaptativo	0,930 91,20 ms	1,144 448,04 ms	13,650 44,98 ms	1,212 57,38 ms	1,604 248,85 ms
JPEG lossless + Huffman	1,631 80,91 ms	1,555 373,50 ms	34,826 40,28 ms	2,166 56,11 ms	1,978 183,41 ms
JPEG lossless + Aritmético	1,699 80,39 ms	1,609 248,95 ms	83,022 54,99 ms	2,268 62,69 ms	1,938 92,30 ms
Transformada Haar	1,243 527,30 ms	1,188 3.434,51 ms	47,522 276,99 ms	1,532 727,10 ms	2,441 3.071,28 ms
Lifting Scheme	1,621 301,78 ms	1,506 1.251,49 ms	77,360 153,69 ms	2,106 158,95 ms	3,106 660,13 ms
JPEG-LS	1,868 66,25 ms	1,683 163,03 ms	72,582 59,17 ms	2,381 63,78 ms	3,203 91,37 ms

La Tabla 3.59 permite un análisis detallado por método, por tipo estático y adaptativo y por tipo de imagen, pero sin embargo este no es el objetivo específico de este capítulo, que simplemente es establecer el estado-del-arte de la compresión de imágenes sin pérdida de datos en un marco homogéneo (mismo ordenador, mismo lenguaje, etc.) para posibilitar el análisis del nuevo método propuesto, INA.

Una primera conclusión es que las imágenes denominadas fotográficas son las que presentan un menor ratio de compresión, es decir, son las más "difíciles" de comprimir y por tanto son las elegidas para establecer la comparativa entre los métodos actuales y el método INA aportado en la tesis.

La segunda conclusión importante y evidente es que en general el ratio de compresión 2:1 puede ser visto como un valor global útil; aunque para ello haya que prescindir de los ratios asociados a imágenes creadas por ordenador ya que son fácilmente comprimibles y su valor es muy elevado. Si se toma como referencia a las imágenes de continuos tonos, denominadas Fotográficas en la tabla, entonces se ve que el valor del ratio de compresión tiende a 2:1 pero no lo alcanza, y por tanto se convierte en un objetivo cuantitativo evidente para contrastar la calidad del nuevo método INA implementado en la tesis.

Las casillas sombreadas remarcan qué métodos son mejores para cada tipo de imagen. Del análisis de estos datos se obtiene como tercera conclusión que el método que se convierte en el estado-del-arte es el JPEG-LS. Este resultado y el anterior son coherentes con la decisión de la industria de adoptarlo como estándar.

La anterior conclusión no queda comprometida por el consumo de recursos, ya que el tiempo de ejecución del método JPEG-LS es en general menor que el de los otros métodos, y en ningún caso es marcadamente superior. Además, el método JPEG-LS es considerado sencillo de implementar.

El único método comparable al JPEG-LS es el *Lifting Scheme*, pero este obtiene menores ratios de compresión en todos los casos, excepto para imágenes creadas por ordenador, y además su tiempo de ejecución es significativamente superior.

Concluyendo, para analizar el método INA propuesto en el siguiente capítulo el análisis se centrará fundamentalmente en el bloque de imágenes Fotográficas, en el método JPEG-LS y en el ratio de compresión 2:1. Todo lo anterior conforma el estado-del-arte de la compresión de imágenes sin pérdida de datos.

Capítulo

4

Algoritmo de compresión INA aplicado a imágenes digitales de continuos tonos

En esta disertación un nuevo método de compresión de imágenes digitales sin pérdida de datos basado en un modelo diferente a los métodos tradicionales es presentado. La principal novedad y aportación científica de este método y núcleo de esta tesis radica en el modelo y en la descripción formal del nuevo algoritmo de compresión sin pérdida de datos aplicado a imágenes de continuos tonos, denominado Algoritmo INA (Larrauri, 2012). La implementación del método de compresión se ha realizado completamente a nivel lógico mediante software convencional aunque sus máximas prestaciones se alcanzarían en una plataforma hardware.

La mayoría de los métodos tradicionales *lossless* utilizan técnicas de eliminación o reducción de la redundancia existente en los datos (píxeles) mediante modelos estadísticos (*Huffman Coding*, *Arithmetic Coding*), modelos de diccionario, modelos de sustitución (LZW, LZ77), modelos predictivos (JPEG *lossless*), modelos basados en transformadas *wavelets* (Transformada Haar y *Lifting Scheme*) o mediante estándares de compresión sin pérdida de datos (JPEG-LS y JPEG 2000). Los ratios alcanzados oscilan desde 1,5:1 y 2:1 considerándose estos últimos como resultados muy aceptables (Larrauri y Kahoraho, 2003) (Khobragade y Thakare, 2014a).

Alternativamente a los métodos tradicionales, proponemos un método de compresión basado en un nuevo modelo de compresión compuesto por tres procesos: segmentación, estructura de árbol binario y codificación.

Este método obtiene un ratio de compresión superior a los métodos convencionales con muy alta probabilidad, como es ilustrado en los resultados experimentales de este método. Incluso, la combinación de este método con el Codificador Aritmético o el Codificador Huffman mejora los ratios de compresión de estos codificadores en más de un 20% en la mayoría de los casos (véase sección 4.4).

Adicionalmente, es importante significar que el proceso de segmentación añade una nueva dimensión al método propuesto porque permite la compresión de datos en paralelo. Esta característica reduce significativamente los tiempos de ejecución del algoritmo en base al número de microprocesadores y núcleos del computador en el cual se realiza la implementación del algoritmo (Shuai, 2008). Esto es, en un computador con un microprocesador de un solo núcleo están disponibles dos versiones del algoritmo: la primera consiste en la ejecución secuencial del algoritmo con un hilo de programación y la segunda permite la ejecución con dos hilos de programación en paralelo.

En plataformas con múltiples microprocesadores y núcleos aprovecha esta configuración para la ejecución del algoritmo en paralelo sobre *n* procesadores simultáneamente (Gilchrist, 2004). Por consiguiente, los tiempos de ejecución se reducen considerablemente en función de la versión del algoritmo y del número procesadores de los que disponga el computador.

En este capítulo el método propuesto es presentado aplicando la misma estructura organizativa usada en todos los métodos revisados en el capítulo anterior.

4.1 Introducción

Los ratios de compresión obtenidos en compresión de imágenes *lossless* dependen directamente de la estructura interna de los datos o variación de los valores de los píxeles con respecto a sus adyacentes. En la compresión de imágenes no existe un método universal que garantice los mismos ratios de compresión para diferentes tipos de fuentes (científicas, fotográficas, médicas, escaneadas, etc.) (Khobragade . y Thakare, 2014b). Un método de compresión puede conseguir grandes ratios en imágenes fotográficas y resultados poco eficientes en otro tipo de imagen. Incluso puede producirse la contradicción de obtener un fichero de salida "comprimido" de mayor tamaño que el fichero original de entrada (véanse los resultados

experimentales de los métodos basados en modelos de sustitución mostrados en el capítulo anterior).

La estructura interna de los píxeles de cualquier imagen está influenciada por la naturaleza de la imagen fuente. La distribución de los valores de los píxeles es diferente para cada clase de imagen fuente. Las imágenes fotográficas contienen cambios significativos entre los valores de los píxeles adyacentes mientras que en imágenes gráficas diseñadas por ordenador existe una gran correlación entre ellos. A nivel teórico, la solución consistiría en seleccionar a priori el método que mejor se ajuste a esta fuente. Sin embargo, en la práctica, dos imágenes provenientes de la misma fuente origen no siempre garantizan una misma distribución de datos.

La solución teórica consiste en analizar la imagen con anterioridad y por medio de un histograma o cálculos probabilísticos extraer las características de la imagen (Fig. 4.1). En base a estos resultados se determinan el método, el modelo y la técnica de codificación que mejor se ajustan a la imagen procesada. Este método sería el que más se aproxime a la entropía fuente. Sin embargo, en la práctica este procedimiento no es factible porque supone una lectura previa de la imagen, un análisis de los modelos y técnicas de codificación óptimas y una toma de decisiones para inferir el método ideal para esa imagen particular. En consecuencia, los tiempos de respuesta y recursos de CPU necesarios para comprimir una imagen se incrementan considerablemente y su utilización es inoperativa.

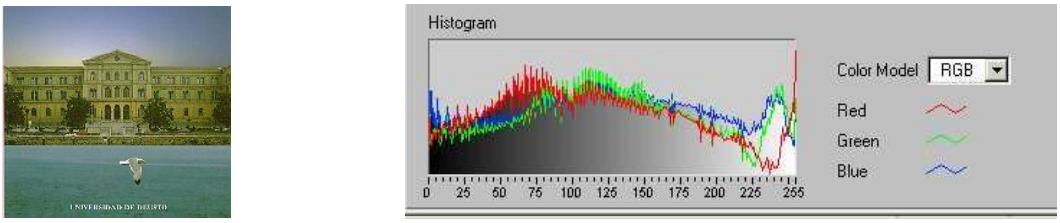


Figura 4.1. Histograma de colores de una imagen RGB

Alternativamente, en esta tesis es presentado un método de compresión con un nuevo enfoque que obtiene resultados cercanos a un ratio de compresión 2:1, sin pérdida de datos e independiente de la procedencia de la imagen original. A partir de este ratio, cuanto mayor sea la redundancia existente en la imagen, mayores ratios de compresión serán alcanzados.

El método propuesto ha sido evaluado aplicando un conjunto de imágenes estándares digitales de continuos tonos de diferentes tipòs de imágenes (fotográficas, satélite, médicas, creadas por ordenador y escaneada). Los resultados experimentales son comparados con los resultados del estándar de compresión de imágenes de continuos tonos sin pérdida de datos denominado *JPEG*-LS que como se demostró en el capítulo anterior es el método que alcanza los mayores ratios de compresión de todos los métodos revisados en el estado del arte.

Adicionalmente, se ha evaluado el algoritmo propuesto bajo ejecución en paralelo y con múltiples núcleos. En esta evaluación no se recogen estudios comparativos con los métodos convencionales ya que la mayoría de ellos no han sido diseñados para la ejecución en paralelo porque para codificar un píxel requieren información de los anteriores.

La estructura de presentación de la descripción del método propuesto que se aborda a continuación es la siguiente: la sección 4.2 presenta una descripción general del método y las posibles configuraciones de compresión en paralelo con sus respectivos modelos; la sección 4.3 describe detalladamente el proceso de compresión, prestando especial enfásis en la descripción formal y principio de funcionamiento del Algoritmo INA incluyendo la programación y flujograma del algoritmo junto con un ejemplo práctico ilustrativo. La sección 4.4 describe el proceso de descompresión dedicando especial interés en el Algoritmo INA inverso que incluye programación, flujograma y un ejemplo ilustrativo de la recuperación de los datos comprimidos en el proceso de compresión; la sección 4.5 presenta los resultados experimentales del método propuesto: por último, la sección 4.6 aporta las conclusiones obtenidas por los resultados experimentales y un estudio comparativo con el estándar JPEG-LS.

4.2 Descripción del método

Esta sección contiene una descripción general del método propuesto que sirve de introducción de los conceptos, estructuras, componentes y terminología del método que son usados en la descripción exhaustiva y detallada de las siguientes subsecciones. La descripción es ilustrada incorporando como ejemplo datos de la imagen estándar Lena.

El método de compresión de imágenes propuesto está basado en un nuevo enfoque de modelo espacial de datos que realiza la compresión de datos en tres procesos secuenciales: segmentación, estructura de árbol binario y codificación. La Fig. 4.2 muestra el diagrama general del método propuesto (Larrauri y Kahoraho, 2002a) (Larrauri, 2012).

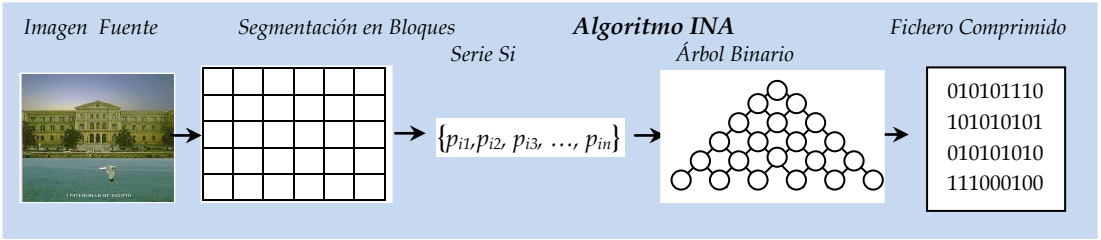


Figura 4.2. Diagrama general del método propuesto

El *proceso de segmentación* divide una imagen a comprimir en bloques de longitud fija o variable según la versión del método seleccionada. El bloque B_i es la unidad de tratamiento de la imagen. Dados los tres primeros bloques de 4x4 píxeles de la imagen estándar Lena, el proceso de segmentación en un nivel general es ilustrado en la Fig. 4.3. La descripción detallada del proceso de segmentación se presenta en la sección 4.3.1.1.

Bloque 1				Bloque 2				Bloque 3			
162	162	162	161	162	157	163	161	166	162	162	160
162	162	162	161	162	157	163	161	166	162	162	160
162	162	162	161	162	157	163	161	166	162	162	160
162	162	162	161	162	157	163	161	166	162	162	160

Figura 4.3. Proceso de segmentación

La aplicación del algoritmo a cada bloque genera dos tipos de estructuras de datos (Larrauri y Kahoraho, 2003). Una primera estructura consiste en una serie de valores numéricos organizados en orden ascendente S_i que contiene los valores diferentes de los píxeles pertenecientes al bloque procesado y una segunda estructura en forma de árbol binario completo cuyos nodos activos están formados por duplas de píxeles A_i consecutivos. La descripción formal del Algoritmo INA se presenta en la sección 4.3.1.1 (Larrauri, 2012b).

La Fig. 4.4 muestra los resultados de la de las dos estructuras generadas por Algoritmo INA para el bloque nº 1. El algoritmo permanece estático durante la ejecución de todos los bloques de la imagen.

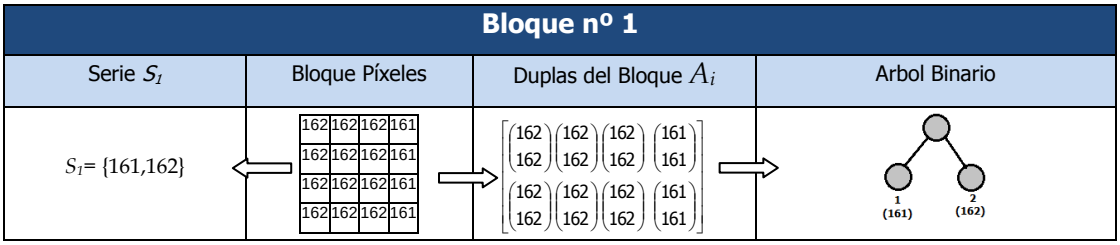


Figura 4.4. Algoritmo INA: serie de píxeles del bloque y árbol binario

El codificador del Algoritmo INA realiza la *codificación* de ambas estructuras de datos S_i y A_i de forma independiente (Larrauri, 2012b). La codificación de la serie *Si* se lleva a cabo mediante una nueva técnica de predicción lineal *adaptativa* propuesta exclusivamente para este algoritmo, mientras que la codificación de la estructura de datos en forma de árbol binario completo A_i es realizada mediante una técnica de codificación directa, que no requiere recorrer el árbol binario ni tampoco de la utilización de tablas como las utilizadas en las versiones reversibles de los

estándares de compresión de imágenes: JPEG *lossless*, JPEG-LS y JPEG 2000. Esta técnica es una versión optimizada de la técnica de codificación transversal clásica.

En el ejemplo propuesto, la serie S_i = {161,162] es comprimida con la técnica de predicción diseñada para este método y que utiliza el siguiente procedimiento en nivel superior de funcionamiento:

- El codificador almacena directamente el primer píxel p_0 de la serie S_i en 1 byte. Valor en binario p_0= "10100001".
- El codificador calcula el valor residual de predicción para el último píxel de la serie que en este caso concide con el segundo. El código binario resultante de p_1 es "0001" (el procedimiento es analizado y explicado en la subsección dedicada al algoritmo 4.3.2).

Por tanto, la serie S_i ocupa 12 bits y el código resultante es la concatenación de los códigos binarios de p_0 y p_1 "101000010001".

Una vez codificada la serie, el algoritmo procede a codificar el árbol binario A_i. El algoritmo sustituye los píxeles por sus correspondientes índices y de esta forma la codificación es directa y rápida. El código de salida de cada dupla consta de una sucesión de unos seguida de ceros. El código de cada dupla se construye sustituyendo el índice del primer elemento de la dupla (posición superior) por unos y a continuación tantos ceros como el índice del segundo elemento de la dupla (posición inferior). La Fig. 4.5 ilustra una codificación simplificada de las duplas de píxeles (Larrauri y Kahoraho, 2003), como ejemplo de una codificación con dos píxeles diferentes.

$$\binom{1}{1} => 10, \quad \binom{1}{2} => 100, \quad \binom{2}{2} => 110, \quad \binom{2}{2} => 1100$$

Figura 4.5. Codificación simplificada de duplas de píxeles

Este innovador método de codificación se aproxima a las técnicas de codificación transversales. La importancia del diseño del codificador reside en que no tiene que construir ni recorrer el árbol binario en ninguno de los dos ciclos. Los métodos tradicionales, como por ejemplo el Codificador Huffman, construyen el árbol y recorren sus ramas para llegar al nodo a codificar.

La Fig. 4.6 muestra la técnica directa de obtención del código binario tomando el valor de los índices para el primer bloque de la imagen Lena (el índice 1 corresponde al píxel 161 y el índice 2 al 162).

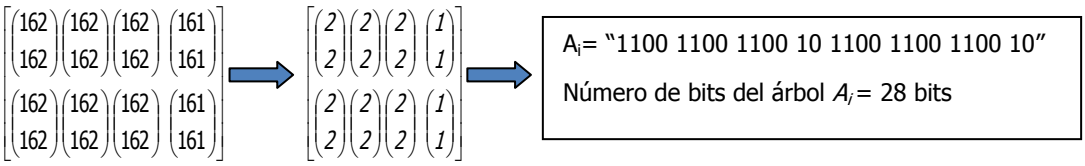


Figura 4.6. Codificación simplificada del árbol binario

Una vez calculado el código binario de la serie y del árbol binario, la codificación del bloque nº 1 es la concatenación de los bits del código de la serie S_i con los bits del árbol binario B_i, resultando la siguiente sucesión de bits: "1010000100011100110011001011001100110010".

El tamaño de la codificación del bloque es igual a 40 bits (12bits + 28 bits = 40 bits). El ratio de compresión del bloque obtenido es el siguiente:

$$\text{Ratio Compresión Bloque } B_1 = \frac{\text{Número bits bloque original}}{\text{Número bits codificados } (S_i + A_i)} = \frac{16 \text{ píxeles } \times 8 \left(\frac{\text{bits}}{\text{píxel}}\right)}{40} = 3{,}20$$

El ratio de compresión en este caso bloque es muy significativo. Obviamente es un caso muy favorable de compresión puesto que el bloque contiene dos píxeles diferentes. Este caso concreto es un recurso especial recogido por el Algoritmo INA.

Al predecir el último píxel detecta que el bloque está formado por dos píxeles considerándose como un caso especial de codificación. El codificador asigna un "1" al índice uno (161) y un "0" al índice 2 (162). En consecuencia, la longitud de bits del código de salida de las duplas es de 16 bits, dos bits por cada dupla y la longitud total del código del bloque nº 1 es de 28 bits (12 bits de la serie S_i más 16 bits de A_i). El ratio de compresión del bloque aplicando este recurso especial es el siguiente:

$$\text{Ratio Compresión} = \frac{\text{Número bits bloque original}}{\text{Número bits codificados } (S_i + A_i)} = \frac{16 \text{ píxeles } \times 8 \left(\frac{\text{bits}}{\text{píxel}}\right)}{28} = 4{,}57$$

Por tanto, el bloque original ha sido reducido a más de una cuarta parte.

En el caso de utilizar una codificación transversal clásica del árbol binario los elementos de la base del árbol se introducen según su orden de aparición en el bloque. Aunque la estructura de árbol binario coincide con la estructura del Algoritmo INA en número de nodos y niveles, los nodos activos en cada árbol son diferentes y por tanto los códigos generados serán también diferentes. Esto implica que la longitud de los códigos de salida sea diferente. La Fig. 4.7 muestra la codificación transversal clásica para dos nodos.

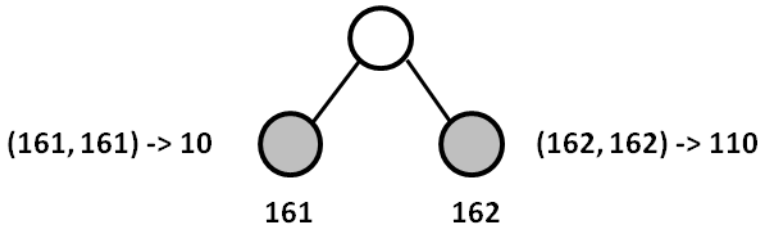


Figura 4.7. Codificación transversal clásica de los nodos

La codificación transversal genera los elementos de la base del arból binario en el orden de aparición de los píxeles en el bloque. Para codificar un nodo cualquiera, el codificador recorre la base del árbol para localizar el primer píxel de la dupla. Una vez localizado asigna tantos unos como nodos recorridos y a continuación atraviesa la rama transversalmente asignado tanto ceros como nodos atravesados hasta llegar al segundo píxel de la dupla.

En el ejemplo propuesto, el codificador genera la base del árbol con los valores de los píxeles 162 y 161 que corresponden a los índices 1 y 2 respectivamente atendiendo al orden de aparición en el bloque. Al índice 1 le asigna un bit igual a uno y al 162 dos unos. Como ambos nodos pertenecen a la base del árbol entonces el codificador añade un cero al final de cada uno de ellos porque cada uno de ellos constituyen un nodo cuya dupla contiene dos píxeles iguales (161,161) y (162,162). El nodo (162,162) es codificado con los bits "10" y el (161,161) con los bits "110". La Fig. 4.8 ilustra la codificación del bloque:

$$\left[\begin{pmatrix}162\\162\end{pmatrix}\begin{pmatrix}162\\162\end{pmatrix}\begin{pmatrix}162\\162\end{pmatrix}\begin{pmatrix}161\\161\end{pmatrix}\begin{pmatrix}162\\162\end{pmatrix}\begin{pmatrix}162\\162\end{pmatrix}\begin{pmatrix}162\\162\end{pmatrix}\begin{pmatrix}161\\161\end{pmatrix}\right]$$

BLOQUE Nº 1
Codif. Transversal de nodos activos : (162,162) -> "**10**" (161,161) -> "**100**"
Cod. Transversal del árbol binario= "**1010100101010100**" -> 18 bits.
Longitud total del bloque codificado = 12 bits + 18 bits = 40 bits
Ratio Compresión = 3,20

Figura 4.8. Codificación transversal propuesta en el algoritmo INA

El método propuesto no ha sido diseñado exclusivamente para alcanzar mayores ratios que los métodos estándares y convencionales, sino que además el Algoritmo INA incorpora un diseño que mejora los tiempos de ejecución mediante *la compresión de datos en paralelo* (Larrauri, 2012). La ejecución en paralelo permite reducir los tiempos de ejecución significativamente. A continuación se presentan las configuraciones generales de la compresión en paralelo para el método propuesto, como se muestra en las Fig. 4.9, Fig. 4.10 y Fig. 4.11.

Este método permite una adaptación automática a la ejecución del algoritmo para diferentes configuraciones de hardware sin cambios de programación. La diferencia entre ellas reside en el número de procesadores y núcleos disponibles. En todas ellas el algoritmo genera las dos estructuras mencionadas anteriormente: estructura de datos de árbol binario completo y una serie de valores diferentes ordenada ascendentemente. La Fig. 4.9 muestra el modelo para cada una de las configuraciones y que a continuación describiremos más detalladamente.

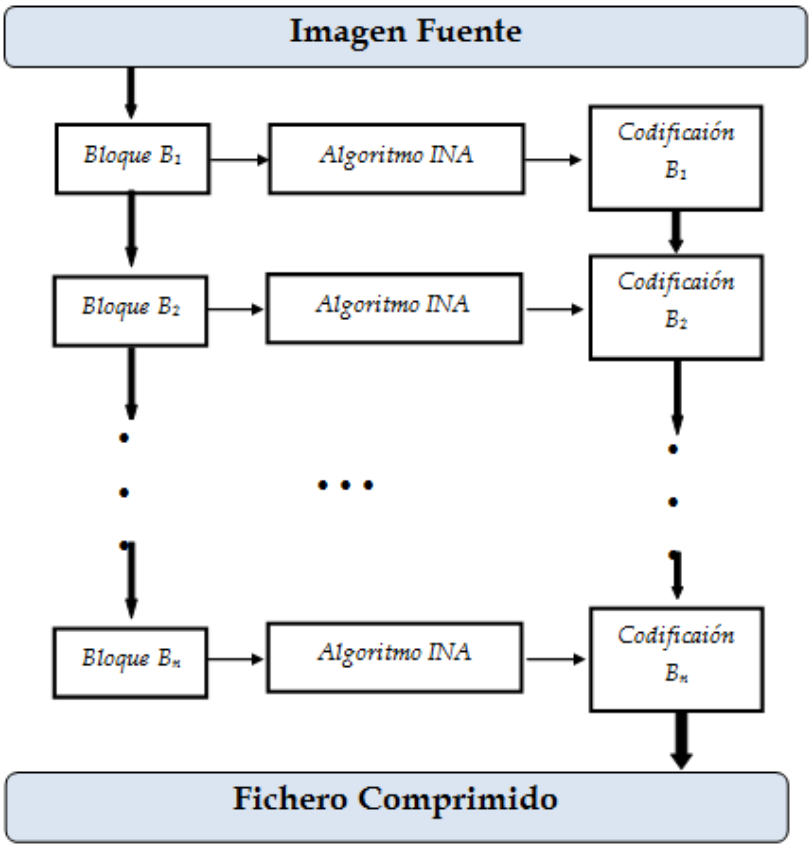


Figura 4.9. Configuración simplificada del método secuencial

La versión básica del método corresponde a la ejecución secuencial del algoritmo en un microprocesador con un solo núcleo. El Algoritmo INA ejecuta los procesos en modo secuencial, cada bloque procesado genera su correspondiente codificación binaria en la salida. Los bloques son procesados uno a uno hasta completar la codificación de la imagen completa.

Una versión simplificada en paralelo mediante programación del algoritmo con dos hilos en paralelo es presentada y desarrollada para equipos con un solo microprocesador y doble núcleo. La Fig. 4.10 muestra la estructura simplificada de la versión simplificada del Algoritmo INA en paralelo.

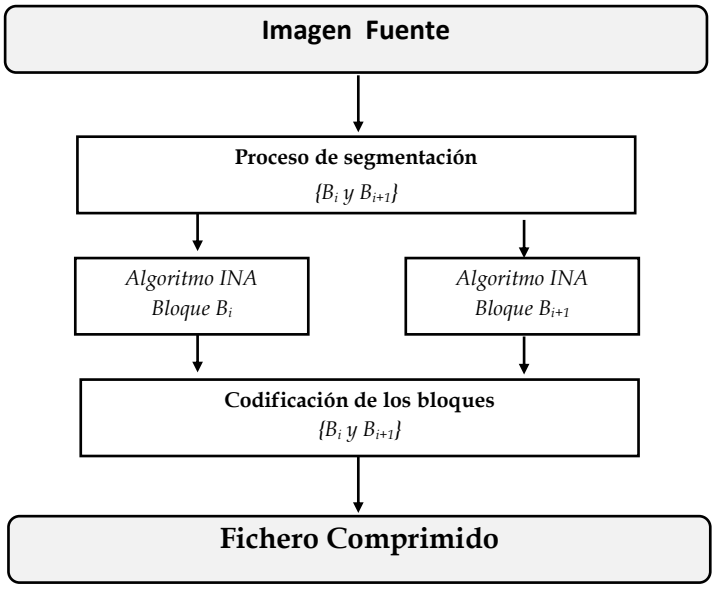


Figura 4.10. Configuración del método en paralelo con dos hilos

Una tercera versión ha sido diseñada para equipos con múltiples procesadores y núcleos (Gilchrist y Cuhadar, 2008). En esta versión, el algoritmo procesa independientemente cada bloque en paralelo y los datos codificados son almacenados en la salida mediante una estructura de gestión de colas FIFO (*First In*, *First Out*).

El tamaño de la cola se establece en base al número de procesadores consiguiendo un buen balance entre la velocidad y la cantidad de memoria necesaria para ejecutar en paralelo.

El tamaño de bloque predeterminado es fijo e igual a 16 bytes. Si el sistema utiliza dos procesadores entonces coloca los dos punteros a los búferes de la cola FIFO para que con sólo un hilo por procesador pueda leer o modificar a la vez. Tan pronto como la cola se ha completado, los hilos de programación ejecutan el Algoritmo INA (Larrauri, 2012). Una vez procesados los datos de las colas por el algoritmo, la memoria del bloque original se libera y un puntero al bloque comprimido junto con el número de bloque se almacena en la cola. Los datos comprimidos de cada bloque son almacenados en el array de salida atendiendo al orden de bloque. Una vez que el espacio de la cola queda libre, los siguientes bloques están disponibles para ser procesados. Este proceso continúa hasta que todos los bloques de la imagen son tratados y todos los hilos de programación han finalizado los últimos bloques. Los datos comprimidos están divididos en zonas de memoria independientes de las comunicaciones entre procesadores. En la Fig. 4.11 se muestra la estructura simplificada del algoritmo con múltiples microprocesadores.

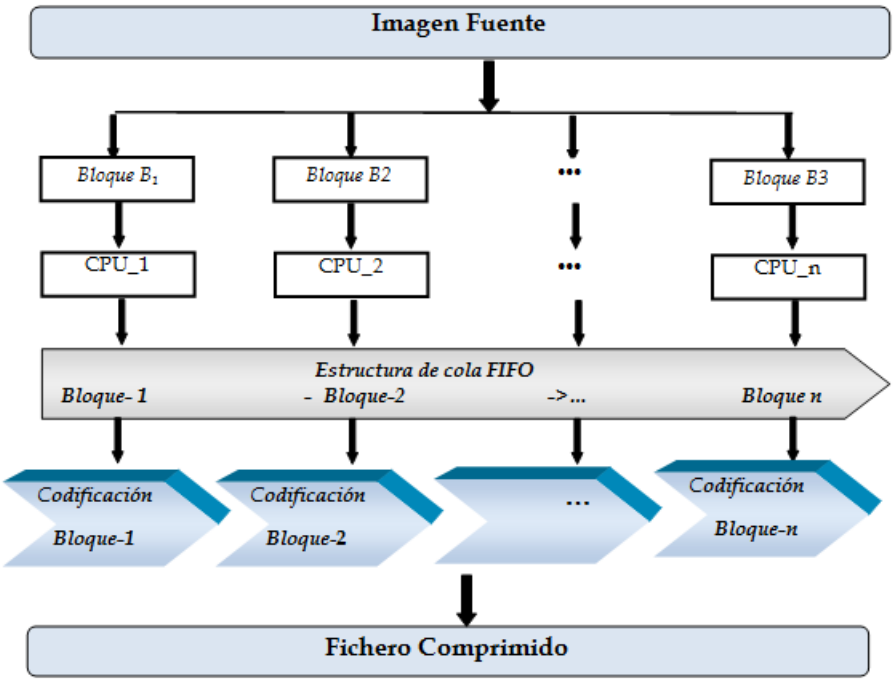


Figura 4.11. Configuración en paralelo con múltiples procesadores

En el proceso de compresión, la imagen original es divida en bloques de longitud fija o variable según el proceso de segmentación seleccionado. La segmentación es el contexto de tratamiento de los píxeles en el proceso de compresión y descompresión. Cada bloque es comprimido aplicando el Algoritmo INA y el resultado es una codificación binaria que es almacenada en memoria consecutivamente hasta completar todos los bloques de la imagen. El proceso de la segmentación se realiza secuencialmente de arriba hacia abajo y de izquierda a derecha hasta finalizar la imagen. Los beneficios de la segmentación son la reducción de cálculos, tiempos de procesamiento y recursos de *CPU* al operar con bloques de tamaño reducido. En la Figura 4.12 se ilustra la estructura del método en diagrama de bloques simplificado (Larrauri, 2012).

Figura 4.12. Diagrama de bloques simplificado del ciclo de compresión

El proceso de descompresión recupera los píxeles de cada bloque en el mismo orden en el que fueron comprimidos. El algoritmo de decodificación extrae del fichero comprimido los píxeles de la serie S_i formada por los valores diferentes. Los valores de los píxeles del bloque se expanden directamente desde el fichero

comprimido sin tener que construir la estructura de árbol binario. En consecuencia, el proceso de descompresión es más rápido que el proceso de compresión.

Figura 4.13. Diagrama general del ciclo de descompresión

La unidad de tratamiento en el proceso de descompresión es el bit. La descompresión se realiza progresivamente de forma inversa recuperando bloque a bloque hasta finalizar el tratamiento completo de del fichero comprimido. El ciclo de descompresión decodifica un bloque aplicando el método de predicción inversa de los píxeles que forman el bloque.

El proceso de segmentación añade una nueva dimensión y un nuevo enfoque al modelo de compresión. Esto es, la segmentación posibilita la *compresión en paralelo* con dos hilos así como la posibilidad de utilizar más procesadores y múltiples núcleos. Esta característica añade potencia al método y reduce notablemente los tiempos de ejecución. Incluso, facilita la integración de la compresión de datos con técnicas de encriptación para cada bloque ofreciendo una mayor seguridad y protección de datos. Cada bloque puede ser comprimido y encriptado con distintos niveles de seguridad. Esta característica garantiza la seguridad en la transmisión y almacenamiento de datos aunque no son objeto del desarrollo de esta tesis.

En la siguiente sección se aborda el proceso de compresión y la descripción detallada del Algoritmo INA.

4.3 Proceso de compresión

En la sección anterior dedicada a la descripción general del método se han introducido los modelos y configuraciones desde el punto de vista de la estrategia del método para ambos ciclos: compresión y descompresión. La finalidad de la sección anterior ha consistido en presentar los conceptos y los componentes fundamentales del método que posteriormente serán analizados en detalle en las siguientes secciones.

En esta sección se aborará el proceso de compresión y especialmente el Algoritmo INA a un nivel de detalle que incluso permita facilitar la implementación del método. A continuación se procede a la descripción detallada del proceso.

El proceso de compresión comienza diviendo la imagen original en bloques de píxeles o segmentación. El principio de funcionamiento del ciclo de compresión consiste en la aplicación secuencial del Algoritmo INA a cada bloque (Larrauri, 2012a). El algoritmo genera dos estructuras de datos: una primera estructura S_i contiene los valores diferentes del bloque ordenados ascendentemente y una segunda estructura de datos en árbol binario completo A_i formada por nodos resultantes de la relación de duplas de píxeles. El codificador genera una secuencia binaria para cada estructura. La concatenación de los códigos binarios de A_i con S_i produce el código de salida del bloque. El ciclo de compresión finaliza cuando todos los bloques de la imagen original han sido procesados y codificados generando el fichero comprimido. La Fig. 4.14 ilustrael proceso de compresión.

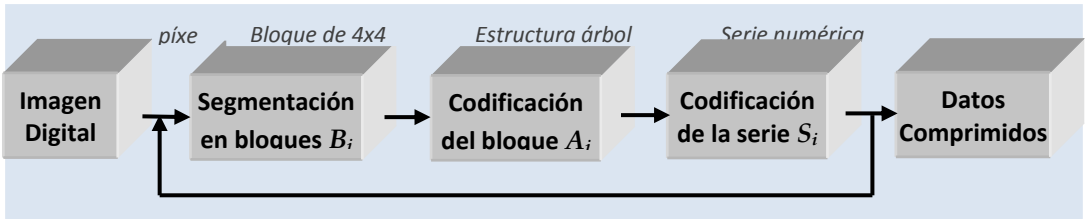


Figura 4.14. Diagrama de bloques simplificado del proceso de compresión

Inicialmente los valores de los píxeles de la imagen fuente son almacenados en memoria en un *array*. El método de compresión ofrece la posibilidad de seleccionar dos tipos de segmentación: fija o variable. El tipo de segmentación permanece invariable durante todo el proceso de compresión y descompresión y no es posible combinar ambas clases de segmentación ni tampoco es posible comprimir usando bloques fijos y descomprimir mediante bloques variables y viceversa.

Por cada bloque tratado es generado un nuevo *array* que contendrá los valores diferentes de los píxeles del bloque en orden ascendente mientras que los datos del árbol binario permanecen en el *array* principal. El algoritmo aplicado a cada bloque obtendrá los códigos binarios de salida que serán almacenados secuencialmente o en dos partes dependiendo del tipo de ejecución seleccionada: secuencial o paralelo. En la Fig. 4.15 se ilustra su principio de funcionamiento.

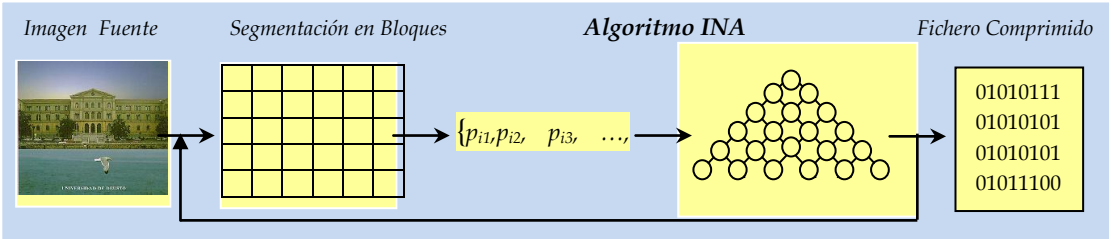


Figura 4.15. Principio de funcionamiento del proceso de compresión

El proceso de segmentación es la clave del éxito para el tratamiento de los datos mediante una estructura de árbol binario completo(Larrauri, 2012b). El algoritmo

de compresión INA es el corazón del método y principal novedad y aportación científica del método propuesto. Los píxeles de cada bloque son tratados como duplas de píxeles generando una estructura de árbol binario. Cada dupla de píxeles genera un nodo activo en el árbol binario. El proceso de codificación lineal permite reducir el *codeword* (número de bits utilizados para representar un nodo desde la raíz hasta el propio nodo).

En las siguientes subsecciones, se describe detalladamente los tres procesos que constituyen el método de compresión propuesto: segmentación, estructura de árbol binario y codificación.

4.3.1 Segmentación en bloques

El proceso de segmentación de una imagen consiste en dividir una imagen en bloques de píxeles de tamaño fijo o variable. La segmentación de una imagen depende del número de bits empleado para representar un píxel. El valor numérico del píxel define la intensidad del color. En caso de imágenes monocromáticas, cada píxel es representado por un bit ("0" negro y "1" blanco). Este tipo de imágenes no son objeto de esta tesis.

Las imágenes representadas en escala de grises utilizan diferentes números de bits para representar un píxel. Cuando el número de bits es igual a 4 bits/píxel significa que cada píxel es representado por un valor entre 0 y 15 que define la intensidad del color del píxel de oscuro a claro. En caso de utilizar un byte de profundidad (8 bits/píxel), la gama de intensidades se establece en un rango entre 0 y 255. Los valores de los píxeles son almacenados en un *array* unidimensional o bidimensional. Estas imágenes son el objeto de compresión de esta tesis.

Las imágenes en color verdadero *(RGB True)* están constituidas por 3 planos independientes que definen cada uno de ellos un plano de color rojo, verde y azul (*Red, Green, Blue*). Esta transformación permite aplicar el Algoritmo INA del mismo modo que a una imagen en escala de grises con una profundidad de 8 bits/píxel. Cada plano es procesado independientemente para tomar ventaja de la correlación entre píxeles adyacentes. Por otra parte, una imagen RGB puede ser almacenada en un único fichero registrando tres bytes consecutivos (RGB, RGB, RGB, …). En este caso, cada gama de color puede ser separada en diferentes *arrays* para ser procesados cada color como una imagen en escala de grises.

Figura 4.16. Planos de colores R-G-B

En el método propuesto en esta tesis están disponibles dos versiones de segmentación según la técnica de *scan*, segmentación fija o variable. En la versión de segmentación fija la imagen es dividida en bloques fijos de 8 o 16 píxeles y en la segunda versión los bloques son de longitud variable pero con un límite de 8 píxeles por cada serie. Esta versión es más apropiada para la compresión de datos en el campo de la visión artificial porque la segmentación es lineal, fila por fila.

Los bloques definen el orden secuencial del procesamiento de la imagen desde el inicio hasta su finalización. En la segmentación fija el número de elementos del bloque B_i y el número de duplas de píxeles es invariable mientras que el número de elementos de la serie S_i es variable. Por el contrario, en la segmentación variable, el número de elementos del bloque B_i y el número de duplas de píxeles es variable mientras que el número de elementos de la serie S_i es fijo.

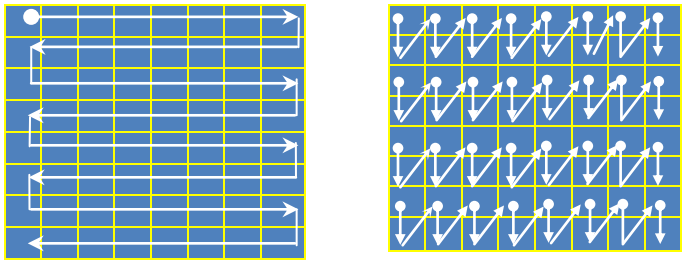
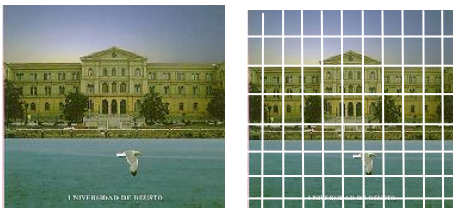


Figura 4.17. Técnicas de scan usadas en el proceso de segmentación

El bloque define el contexto del tratamiento de los datos y prepara los datos para la aplicación del Algoritmo INA que generará una estructura de datos en forma de árbol binario A_i y una serie numérica S_i (Larrauri, 2002a). Los píxeles diferentes p_{in} de cada bloque B_i son colocados en orden ascendente y estos son sutituidos por sus correspondientes índices en la base del árbol binario completo.



Bloque B_i

$p_{1,1}$	$p_{1,2}$	$p_{1,3}$	$p_{1,4}$
$p_{2,1}$	$p_{2,2}$	$p_{2,3}$	$p_{2,4}$
$p_{3,1}$	$p_{3,2}$	$p_{3,3}$	$p_{3,4}$
$p_{4,1}$	$p_{4,2}$	$p_{4,3}$	$p_{4,4}$

$Bi = \{ p_1, p_2, p_3, p_4 .... p_{16} \}$

Figura 4.18. Segmentación de la imagen en bloques

La segmentación de la imagen no genera compresión directa de datos. Sin embargo, aporta los siguientes beneficios (Larrauri y Kahoraho, 2003):

- Representa el orden secuencial de procesamiento de la imagen.
- Incrementa la velocidad de procesamiento al operar con bloques de tamaño reducido.

- Reduce el espacio de almacenamiento o memoria.
- Incrementa la rapidez de visualización de los datos en el ciclo de compresión y descompresión de la imagen.
- Prepara los datos para generar estructuras de datos.
- Facilita la ejecución en paralelo con uno o más procesadores.
- En la versión *in-place* prepara los datos para ser usados por el Codificador Huffman o por el Codificador Aritmético mejorando sus propios resultados.

La rutina de segmentación es simple y rápida. La programación del proceso de segmentación depende de la forma en la cual se hayan guardado los píxeles: *array* unidimensional o bidimensional. En un *array* unidimensional, la segmentación en bloques variables es más simple porque los índices de los píxeles son consecutivos y el bloque se cierra al completar los ocho píxeles diferentes. La segmentación en bloques fijos requiere refinar los índices. Sin embargo, el tamaño del bloque solo requiere un contador. El Listado 4.1 muestra la rutina de programación para un *array* unidimensional.

```
// Segmentación Variable
for (i=0;i<altura*anchura;i++)
{
  x=array[i];
  pixel[x]++;
  if (pixel[x]==1)
    {
      bloque[cont]=x;
      cont++;
      if (cont>7) //
        {
          cont=0;
           // algoritmo ina;
        }
    }
}
```

```
// Segementación fija- Bloques 4x4 píxeles
for (i=0; i<altura; i+4) {
 for (j=0; j<anchura; j++) {
   bloque[cont]=array[i*anchura+j];
   pixel [array[i*anchura+j]]++;
   if (pixel [array[i*anchura+j]]==1) {
     serie[x2]=array[i*anchura+j],x2++; }
     bloque[cont+1]=array[(i+1)*anchura+j]
     pixel [array[(i+1)*anchura+j]]++;
   if (pixel[array[(i+1)*anchura+j]]==1) {
    serie[x2]=array[(i+1)*anchura+j],x2++; }
    bloque[cont+8]=array[(i+2)*anchura+j];
    pixel [array[(i+2)*anchura+j]]++;
  if (pixel [array[(i+2)*anchura+j]]==1) {
    serie[x2]=array[(i+2)*anchura+j],x2++; }
     bloque[cont+9]=array[(i+3)*anchura+j];
     pixel [array[(i+3)*anchura+j]]++;
  if (pixel [array[(i+3)*anchura+j]]==1) {
    serie[x2]=array[(i+3)*anchura+j],x2++; }
   cont=cont+2;
   if (cont>6)
    }
 }
```

Listado 4.1. Programación del proceso de segmentación variable y fija

En la Tabla 4.1 se muestran los resultados obtenidos al aplicar cada tipo de segmentación a los tres primeros bloques de la imagen Lena.

Tabla 4-1. Segmentación en bloques variables y fijos

SEGMENTACIÓN VARIABLE	SEGMENTACIÓN FIJA
********** Bloque n°: 1 ********** Bi= {162,162,162,161,162,157,163,161,166, 162,162,160,155,163,160,155,157,156} Número de píxeles del bloque = 18 [0-17] Píxeles diferentes = 8 S_i = {155,156,157,160,161,162,163,166} $A_i = \left[\binom{162}{162}\binom{162}{161}\binom{162}{157}\binom{163}{161}\binom{166}{162}\binom{162}{160}\binom{155}{163}\binom{160}{155}\binom{157}{156}\right]$	********** Bloque n°: 1 ********** $\left[\binom{162}{162}\binom{162}{162}\binom{162}{162}\binom{161}{161}\binom{162}{162}\binom{162}{162}\binom{162}{162}\binom{161}{161}\right]$ B_i = {162,162,162,161,162,162,162,161, 162,162,162,161,162,162,162,161} S_i= {161,162} Índice | 1 | 2 valor | 161 | 162
******** Bloque n°: 2 ********** B_i={161,161,154,156,154,157,153,157,154, 152,156,154,154,156,154,158,155,155} Número de píxeles del bloque = 18 [18-35] Píxeles diferentes = 8 Si={152,153,154,155,156,157,158,161} $A_i = \left[\binom{161}{161}\binom{154}{156}\binom{154}{157}\binom{153}{157}\binom{154}{152}\binom{156}{154}\binom{154}{156}\binom{154}{158}\binom{155}{155}\right]$	********** Bloque n°: 2 ********** $\left[\binom{162}{162}\binom{157}{157}\binom{163}{163}\binom{161}{161}\binom{162}{162}\binom{157}{157}\binom{163}{163}\binom{161}{161}\right]$ B_i={162,162,157,157,163,163,161,161, 162,162,157,157,163,163,161,161} S_i = {157, 161, 162, 163} ÍndiceE | 1 | 2 | 3 | 4 valor | 157 | 161 | 162 | 163
******** Bloque n°: 3 ********** Bi={0,158,167,160,166,166,165,166,172,171} Numero de píxeles del bloque = 10 [36-45] Píxeles diferentes = 8 Serie S_i= {0,158,160,165,166,167,171,172} $A_i = \left[\binom{0}{158}\binom{160}{165}\binom{154}{157}\binom{166}{167}\binom{171}{172}\right]$	********** Bloque n°: 3 ********** $\left[\binom{166}{166}\binom{162}{162}\binom{162}{162}\binom{160}{160}\binom{166}{166}\binom{162}{162}\binom{162}{162}\binom{160}{160}\right]$ B_i={166,166,162,162,162,162,160,160, 166,166,162,162,162,162,160,160} S_i = {160, 162, 166} ÍndiceE | 1 | 2 | 3 valor | 160 | 163 | 166

Una vez segmentada la imagen en bloques de datos tamaño muy reducido, estos ya están preparados para aplicar directamente el Algoritmo INA.

4.3.1.1 Algoritmo de compresión INA

El núcleo y principal aportación de esta tesis es el diseño y desarrollo de un nuevo algoritmo de compresión de imágenes digitales de continuos tonos conocido como Algoritmo INA. Este algoritmo presenta un enfoque diferente de los algoritmos tradicionales. El algoritmo se compone de tres pasos que pueden ser ejecutados

secuencialmente o en paralelo según la versión del algoritmo (Larrauri, 2012). La descripción formal del algoritmo es la siguiente:

Paso 1. Generar una estructura de datos numérica S_i con los píxeles diferentes del bloque B_i clasificados en orden ascendente.

Paso 2. Generar una estructura de datos en forma de árbol binario completo A_i formada por duplas de píxeles del bloque B_i.

Paso 3. Codificar la serie S_i aplicando una nueva técnica de predicción lineal y el árbol binario B_i mediante un sistema directo de codificación transversal que no requiere recorrer el arbol.

Una vez finalizado el tercer paso retorna al primer paso repitiendo los mismos pasos hasta finalizar el procesamiento del último bloque. En la Fig. 4.19 se muestra el diagrama de bloques de la estructura del algoritmo.

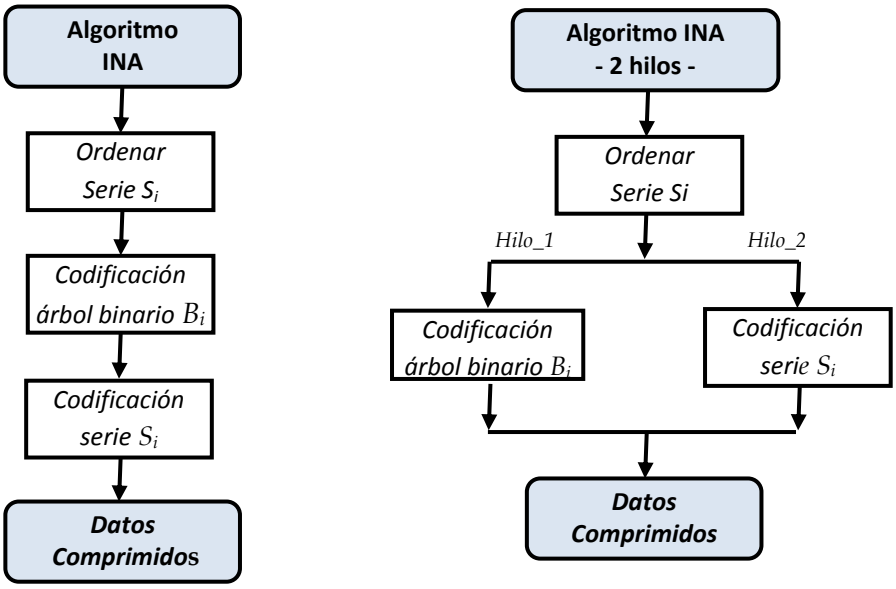


Figura 4.19. Diagrama de bloques del Algoritmo INA con uno y dos hilos

La unidad de entrada de datos al algoritmo es un bloque de píxeles que define el *contexto* de aplicación del algoritmo durante el procesamiento completo de la imagen. La dupla de píxeles es la unidad de tratamiento del árbol binario y el píxel es la unidad de tramiento de la serie númerica. La salida del codificador de ambas estructuras es un código binario por bloque.

En las siguientes subsecciónes se procede a la descripción formal del algoritmo paso a paso. El principio de funcionamiento del algoritmo es acompañado de un ejemplo simplificado (bloque nº 2 de la imagen Lena) aplicando la misma metodología usada en los métodos revisados en el estado del arte.

4.3.1.1.1 Paso 1. Generar una estructura de datos S_i

Una vez que obtenido el bloque de píxeles en el proceso de segmentación el algoritmo ordena ascendentemente los valores numéricos de los píxeles diferentes de cada **bloque** B_i generando una **serie** S_i de números enteros positivos que satisfaga la siguiente expresión:

$$S_i = \{p_1 < p_2 < p_3 < \cdots < p_n\} \quad (4.1)$$

siendo $n \leq 16$ en la versión de segmentación en bloques fijos y $n=8$ en la segmentación variable.

La ordenación de los píxeles de la serie en un array unidimensional se ha implementado mediante un *método de inserción* seleccionado entre los principales métodos de clasificación existentes. La selección de este método se ha realizado después de un estudio y test comparativo de tiempos de ejecución entre los principales métodos de ordenación de valores numéricos (véase la sección de implementación del algoritmo donde se aportan los resultados experimentales). La programación del algoritmo de inserción es simple. El Listado 4.2 muestra la codificación del algoritmo de inserción.

```
*** Algoritmo de inserción *****
for(y=0;y<x2;y++)  // x2=número de píxeles diferentes
{
  index = serie[y];
  for (a=y-1;a>= 0 && serie[a]>index;a--)
  {
    serie[a + 1] = serie[a];
    indblo[serie[a+1]]= a+1; // almacena valor índice
  }
  serie[a+1] = index;
  indblo[serie[a+1]]= a+1;
}  // Fin de ordenación del bloque
```

Listado 4.2. Programación del algoritmo de inserción

Tomemos como ejemplo los valores de los píxeles de la figura siguiente, el resultado es el bloque B_i y la serie S_i que se muestra a continuación:

$$\begin{bmatrix} p_{0,0} & p_{0,1} & p_{0,2} & p_{0,3} \\ p_{1,1} & p_{1,1} & p_{1,2} & p_{1,3} \\ p_{2,1} & p_{2,1} & p_{2,2} & p_{2,3} \\ p_{3,1} & p_{3,1} & p_{3,2} & p_{3,3} \end{bmatrix} \Longrightarrow \begin{bmatrix} 162 & 157 & 163 & 161 \\ 162 & 157 & 163 & 161 \\ 162 & 157 & 163 & 161 \\ 162 & 157 & 163 & 161 \end{bmatrix} \Longrightarrow \quad Si=\{157<161<162<163\}$$

Figura 4.20. Obtención de la Serie S_i

La disposición del almacenamiento de los píxeles de la serie S_i en el array unidimensional es la mostrada en la Tabla 4.2.

Tabla 4-2. Disposición de los píxeles de la serie S_i

PÍXELES SERIE S_I				
Índice	1	2	3	4
S_i	157	161	162	163

El índice de cada dato juega un factor esencial en la codificación del bloque B_i que se describe a continuación.

4.3.1.1.2 Paso 2. Generar una estructura de datos en forma de árbol binario A_i

El algoritmo genera una estructura de datos que representa un árbol binario completo formado por los nodos activos resultantes del procesamiento de las duplas de píxeles del bloque B_i. La Fig. 4.21 muestra el orden de procesamiento de duplas y bloques (Larrauri, 2002a).

BLOQUE	DUPLAS (índices)	ORDEN DE DUPLAS
$\begin{bmatrix} \binom{162}{162} \binom{157}{157} \binom{163}{163} \binom{161}{161} \\ \binom{162}{162} \binom{157}{157} \binom{163}{163} \binom{161}{161} \end{bmatrix}$	$\begin{bmatrix} \binom{p_h}{p_l} \binom{p_h}{p_l} \binom{p_h}{p_l} \binom{p_h}{p_l} \\ \binom{p_h}{p_l} \binom{p_h}{p_l} \binom{p_h}{p_l} \binom{p_h}{p_l} \end{bmatrix} \Rightarrow \begin{bmatrix} \binom{3}{3} \binom{1}{1} \binom{4}{4} \binom{2}{2} \\ \binom{3}{3} \binom{1}{1} \binom{4}{4} \binom{2}{2} \end{bmatrix}$	$\begin{bmatrix} 1 \Rightarrow & 2 \Rightarrow & 3 \Rightarrow 4 \\ 5 \Rightarrow & 6 \Rightarrow & 7 \Rightarrow 8 \end{bmatrix}$

Figura 4.21. Estructura de datos del bloque

La construcción del árbol binario es realizada siguiendo los siguientes pasos:

1. El nivel base o inferior del árbol binario está formado n nodos, siendo n el número de elementos de la serie S_i. La codificación de cada nodo contendrá una secuencia de uno(s) seguida de un cero que representa la base del árbol. La secuencia de uno(s) consiste en la sustitución del valor del índice por tantos unos como unidades contenga el índice. La Fig. 4.22 ilustra la formación del nivel inferior del árbol binario.

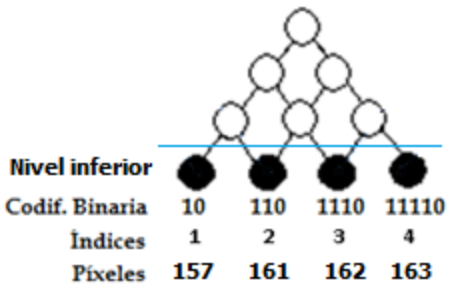


Figura 4.22. Representación de los datos del bloque en una estructura de árbol binario

2. Cada par de píxeles adyacentes de la base del árbol genera un nodo inactivo padre. La generación de todos los nodos padres desde un nivel cualquiera crea un nivel superior en el árbol formado por (*n-1*) nodos.
3. El proceso se repite en cada nivel de nodos padres hasta llegar al nodo raíz añadiendo un cero en la secuencia de ceros por cada nivel superior atravesado.
4. Finalmente, cada dupla de píxeles del bloque B_i genera un nodo activo en la estructura de árbol binario.

Las propiedades del árbol binario generado son las siguientes (Larrauri, 2012):

a) El árbol binario es completo, es decir cada nodo hijo pertenece a un nodo padre.

b) Todos los píxeles del bloque están representados al menos una vez en un nodo activo.

c) Siempre existe un nodo activo representado en la rama izquierda y otro en la rama derecha del árbol. Esta propiedad permite reducir todavía aun más el número de bits de la codificación.

d) Posibilidad de compactación del árbol binario representando solamente los niveles o distancias existentes.

e) Cada nodo está representado por un código único e inferior a n+1 bits, siendo n el número de píxeles diferentes.

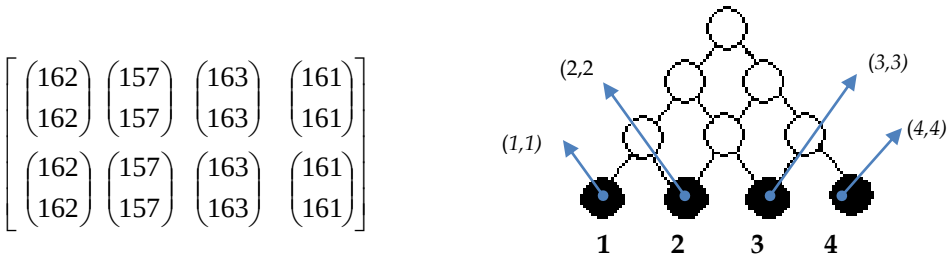


Figura 4.23. Árbol binario, estructura de nodos y codificación

Cada dupla de píxeles genera un nodo en el árbol binario. El orden de procesamiento de los pares se realiza de izquierda a derecha y de arriba hacia abajo. En la versión de segmentación en bloques fijos, el número de nodos del árbol binario siempre es fijo y contendrá cuatro o bien ocho nodos mientras que el número de elementos será variable y oscilará entre 1 y 15 elementos. En la segmentación variable, el número de nodos es variable mientras que el tamaño del árbol binario permanece fijo y estará constituido por ocho nodos.

4.3.1.1.3 Paso 3. Codificar las estructuras de datos S_i y A_i

EL algoritmo propone dos tipos de técnicas de codificación, una por cada tipo de estructura de datos creada. El árbol binario es codificado mediante una codificación transversal directa y la serie mediante una nueva técnica de predicción diseñada exclusivamente para este algoritmo. A continuación se procede a la descripción de ambas técnicas, la técnica transversal y la técnica de predicción.

4.3.1.1.4 Paso 3.1. Codificación transversal directa del árbol binario

La codificación transversal genera un código binario por cada nodo del árbol binario formado cuyo código binario es una secuencia de unos y ceros. La codificación binaria de un nodo cualquiera está influenciada por la posición de cada píxel del nodo en la serie *Si*. El índice del primer elemento del nodo define el número de unos y el valor absoluto de la diferencia entre el primer y segundo índice más uno define el numero de ceros.

$$\mathrm{Cn_i} = \mathrm{unos(p_h)} + \mathrm{ceros(|b(p_h) - b(p_l)| + 1)}$$

siendo, Cn_i: el código resultante,
n_i : el nodo a codificar $unos(p_h)$.

El código resultante es la concatenación de unos y ceros, tal y como se muestra en la Tabla 4.3.

Tabla 4-3. Codificación transversal del árbol binario

Nº NODOS	DUPLA NODO	ÍNDICE PRIMER PÍXEL	ÍNDICE SEGUNDO PÍXEL	CODIFICACIÓN
1	(162,162)	3 – "111"	3 - "0"	1110
2	(157,157)	1- "1"	1 – "0"	10
3	(163,163)	4- "1111"	4 –"0"	11110
4	(161,161)	2- "11"	2 - "0"	110
5	(162,162)	3 – "111"	3- "0"	1110
6	(157,157)	1- "1"	1- "0"	10
7	(163,163)	4- "1111"	4- "0"	11110
8	(161,161)	2- "11"	2- "0"	110
Código de salida del árbol binario A_i = "1110101111011011101011110110"				T (B_1) = 28bits

El código de salida es único, no contiene incertidumbre, y por tanto la recuperación de los píxeles es reversible. El tamaño del bloque original tiene una longitud de 128 bits (16 píxeles x 8 bits/píxel). Si la codificación de la serie mediante el sistema de predicción fuera menor de 36 bits, el ratio de compresión sería de 2:1.

El número total de bits necesarios $T(B_i)$ para codificar un árbol binario aplicando el algoritmo propuesto a una estructura de datos en forma de árbol binario compacto , compuesta por ocho nodos resultantes de las duplas de píxeles

de cada bloque B_i , construido a partir de una matriz de 4x4 píxeles, es calculado mediante la siguiente expresión (Larrauri, 2012):

$$T(B_i) = \sum_{i=1}^{n/2} b(p_h) + |b(p_h) - b(p_l)| + 1 \quad (4.2)$$

siendo, n el número total de elementos de la serie S_i, $b(p_l)$ el número de bits que corresponden al índice del píxel de menor orden de la dupla y $b(p_h)$ el número de bits del índice del píxel de mayor orden.

En la implementación de la codificación transversal directa, el algoritmo utiliza una nueva técnica basada en la codificación de los nodos en función del valor de los índices en la dupla de píxeles. Si el primer índice del nodo es menor o igual al segundo se aplica la codificación transversal clásica, en caso contrario, el algoritmo incorpora un nuevo recurso que optimiza el código del nodo. Esta técnica consiste en representar cada nodo como una sucesión de unos, p_i, resultante de la diferencia del índice mayor menos el menor manteniendo el mismo número de ceros del segundo indice. La concatenación de los bits genera el código del nodo.

$$b(p_h) = |b(p_h) - b(p_l)| \quad (4.3)$$

Un ejemplo de optimización se muestra en la Tabla 4.4.

Tabla 4-4. Optimización de la codificación transversal mediante el Algoritmo INA

DUPLAS DE PÍXELES	CODIFICACIÓN TRANSVERSAL	OPTIMIZACIÓN ALGORITMO INA
(1,1)	10	No se optimiza--
(1,6)	100000	No se optimiza--
(6,4)	111111000 ⇒	110000
(6,6)	1111110	No se optimiza--
(5,3)	1111100 ⇒	11000

Obviamente, el algoritmo informa al decodificador que esa dupla ha sido codificada con el recurso de optimización añadiendo un bit de control. Los bits de control se almacenan siempre al inicio de la codificación del bloque.

En términos de almacenamiento y expresado en ecuaciones algorítmicas tendremos que el número de bits para una secuencia de unos es definido por la siguiente expresión:

$$b(p_h) = \log_2(S_n - S_a) \quad (4.4)$$

siendo, *Sn* el índice del último elemento de la serie *Si* y *Sa* el índice del píxel actual.

El número de ceros se obtiene mediante $\log_2 p_l$. El código binario de la dupla o nodo es la concatenación de los bits resultantes de codificar p_h y p_l. El Listado 4.3 contiene el código fuente para la codificación de un bloque fijo cualquiera.

```
*** Programación de los nodos del Bloque Bi *****
for (y=0; y<16; y++)
{
  numbits=0;
{
  if (y%2)  // si tiene resto el píxel -IMPAR
  {
    if ( bloque[y-1]<bloque[y])
    {
       for(z=0; z<(indblo[bloque[y-1]])+1; z++)
       codeword= indblo[bloque[y-1]];
      }
      else
      {
         for(z=0; z<(indblo[bloque[y]])+1; z++)
         codeword= indblo[bloque[y]];
      }
num2= abs(indblo[bloque[y-1]]-indblo[bloque[y]])+1;
aux3= num2;
numbits= log(x2-codeword+1)/log(2.0); // numbit es entero sin decimales
loga= log(x2-codeword+1)/log(2.0);    // contiene  decimales
codifica_nodo(bit1, bit2);
}
```

Listado 4.3. Programación de los nodos del bloque B_i

Una vez codificado el bloque de píxeles *Bi*, el algoritmo ejecutará el paso tercero para codificar la serie *Si* generada por el bloque. En la siguiente subsección se describe exhaustivamente la técnica de predicción lineal propuesta en el Algoritmo INA.

4.3.1.1.5 Paso 3.2 . Codificación predictiva de S_i

La serie S_i está formada por los píxeles diferentes del bloque clasificados en orden ascendente. La codificación se realiza mediante una nueva y original técnica de predicción lineal aplicada a series de números enteros positivos ordenados ascendentemente.

Dada una serie S_i cualquiera, ordenada en forma ascendente, es posible codificar los valores de los píxeles diferentes explotando las pequeñas diferencias de valores existentes entre los píxeles adyacentes.

La técnica de codificación predictiva propuesta por el Algoritmo INA codifica los píxeles de la serie mediante el cálculo de diferencias entre ellos (Larrauri, 2012). Esas diferencias la denotamos como v_r o *valor residual* y siempre serán valores menores que los píxeles y por tanto la codificación produce compresión de datos reversible. Adicionalmente, cada valor residual es asociado a unos bits de control o *predictores* que facilitan la codificación y establecen el número de bits necesarios para codificar el valor residual.

Esta técnica puede considerarse en algun aspecto similar a la utilizada en la versión *lossless* del primer estándar JPEG, o a otras técnicas de codificación predictiva. Sin embargo, la metodología de la técnica propuesta por el Algoritmo

INA es completamente distinta a las convencionales porque explota la propiedades de estructura de datos S_i. Esto es, la propiedad de *diferente* (todos los elementos de la serie son diferentes) y la propiedad *mayor que* (cada elemento de la serie es mayor que el anterior). Las técnicas convencionales utilizan un contexto o grupo de píxeles ya codificados o conocidos para predecir un píxel explotando la propiedad de igualdad (máxima compresión) o pequeñas diferencias entre ellos para codificarlos satisfactoriamente.

La técnica predictiva desarrollada en este algoritmo consiste en tres pasos.

Paso 1. Codificar en binario el primer píxel de la serie

Paso 2. Codificar el último píxel en función del primero para determinar el número máximo de bits necesarios para codificar cualquier elemento de la serie.

Paso 3. Codificar el resto de los píxeles de la serie en función de su diferencia con el anterior teniendo en cuenta el número máximo y mínimo de bits.

En la Fig. 4-24 se ilustra el principio de funcionamiento de la técnica de predicción propuesta para codificar la serie S_i.

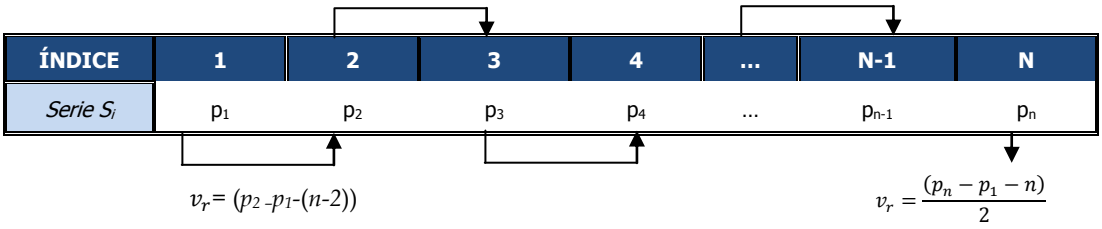


Figura 4.24. Técnica de predicción lineal

Paso 1. Codificar el primer píxel p_1 de la serie S_i.

El primer píxel es almacenado directamente en la salida del codificador ocupando un byte (8 bits). Tomando ventaja de la propiedad de correlación existente entre los píxeles es posible reducir el número de bits del primer píxel de cada serie usando un bit de control que indicará si el primer píxel de la siguiente serie es igual al último píxel de la anterior.

Mediante un bit de control se reduce drásticamente el número de bits para representar el primer píxel de la serie quedando reducido a un solo bit en caso de igualdad o bien a la mitad (4 píxeles) cuando es cercano al anterior. Continuando con el ejemplo propuesto, el primer píxel, p_1=157 y es codificado y guardado en la salida binaria.

ÍNDICE	1	2	3	4
Serie S_i	157	161	162	163

Salida binaria del primer píxel p_1=157 => 10011101

El Listado 4.4 contiene la programación para obtener la codificación binaria de un valor decimal:

```
aux=128;  // MSB aux=1; SHIFT a la derecha y compara
for (y=0; y<8; y++) {
  if (píxel_0 & aux2)
     salida[cont1]= '1', cont1++;
  else
     salida[cont1]= '0', cont1++;
aux2=aux2>>1; }
```

Listado 4.4. Programación del algoritmo de transformación de decimal a binario

La codificación se guarda en un array de memoria. Cuando todos los bloques de la imagen se hayan codificado se almacena en un fichero binario. La salida binaria contiene los siguientes bits: "10011101".

Paso 2. Predicción del último píxel p_n de la serie S_i.

Una vez obtenido el valor del primer píxel, es conocido si su valor es par o impar y obviamente es menor que el siguiente píxel de la serie. Para predecir el último píxel aportamos una nueva expresión para el cálculo del valor residual de predicción para este píxel.

$$V_r = \frac{p_n - p_1 - n}{2} \tag{4.5}$$

El número de bits necesarios *Nb* para codificar y almacenar v_r es igual al $\log_2 v_r$. Si el último píxel de la serie requiere más de tres bits para su codificación entonces toma como referencia el mayor valor posible "255" y en consecuencia el número de bits necesarios será calculado aplicando la expresión 4.6:

$$Nb = \log_2(255 - p_1) \tag{4.6}$$

Los predictores propuestos por el algoritmo son definidos con dos bits de estado (bit_1 y bit_2) asociados al tipo de paridad, longitud del código en bits y cálculo de valor residual. La Tabla 4.5 contiene los predictores usados por el algoritmo.

Tabla 4-5. Predictores, bits de estado, paridad y bits usados en la técnica de predicción

Nº PREDICTOR	BITS DE ESTADO BIT_1	BIT_2	PARIDAD Y NÚMERO DE BITS REQUERIDOS PARA REPRESENTAR EL PÍXEL	NB
1	0	0	Los píxeles p_1 y p_n son pares o impares Paridad = SI y Nb < 3 bits	4 bits= 2 bits control + 2 de v_r
2	0	1	Los píxeles p_1 y p_n son pares o impares Paridad= SI y Nb >2bits	v_r=(255-p_1-(n-2))/2 Nb= log2 v_r
3	1	0	Los píxeles p_1 y p_n tienen diferente paridad Paridad= NO y Nb < 3 bits	4 bits= 2 bits control + 2 de v_r
4	1	1	Los píxeles p_1 y p_n tienen diferente paridad Paridad= NO y Nb > 2 bits	v_r=(255-p0-(n-2))/2 Nb= log2 v_r

Premisa 1: la diferencia de dos números naturales pares en los cuales el primer operando es mayor que el segundo será siempre un número par y por tanto si lo dividimos entre dos, el resultado será un numero natural par.

En base a esta primera premisa, la recuperación del píxel p_n es reversible a partir del píxel p_1, siempre que tengan la misma paridad.

$$v_r = \frac{(p_n - p_1 - n)}{2} \Rightarrow \quad p_n = 2v_r + p_1 + n \tag{4.7}$$

Premisa 2: la diferencia de dos números naturales con paridad opuesta siempre es un número impar y el resultado dividido entre dos siempre es un número decimal.

En base a esta segunda premisa, la recuperación del píxel p_n no es reversible a partir del píxel p_1, porque existe ambigüedad en los datos al extraer la parte entera. Para resolver esta problemática se añade un bit de estado que garantiza la eliminación de la ambigüedad en los datos.

A partir de estas dos premisas formuladas, el algoritmo de predicción añadirá dos bits adicionales denominados bits de estado. El primer bit (bit_1) indicará la paridad, asignando "0" si tienen la misma paridad y "1" si la paridad es distinta. Para reducir más el tamaño del código se añade un segundo bit de estado (bit_2) que indica si el valor residual se representa con dos bits o con más bits, mediante el bit "0" y "1" respectivamente. El Listado 4.5 contiene la programación de la función de paridad.

```
*** Programación del bit_1 de paridad *****
int par(int num1,int num2)  // función
{
  if (num1%2)  // si tiene resto el píxel1
     tipo1 = 'I';   // impar
else
     tipo1 = 'P';   // par
```

```
if (num2%2)   // si tiene resto
     tipo2 = 'I';
else
     tipo2 = 'P';
if (tipo1==tipo2)
    iguales= 'S';
else
    iguales= 'N';
return (0);
}
```

Listado 4.5. Programación de la función de paridad

Aplicando las dos premisas enunciadas para obtener la codificación binaria del último píxel se obtiene:

$$\text{Si}=\{157<161<162<163\}$$

$$v_r = \frac{p_n - p_1 - n}{2} = \frac{163 - 157 - 4}{2} = 1$$

Tabla 4-6. Bits de estado

BIT DE ESTADO					
Índice 1	**Índice n**	**Bit Paridad Diferente Bit_1**	**Bit estado <2bits Bit_2**	**v_r En binario**	**Código último píxel**
$p_1 = 157$	$p_4 = 163$	0	0	01	0001

$V_r = (p_n - p_0 - (n-2))/2$

La salida binaria del codificador para el último elemento de la serie p_4 es "0001" que se obtiene de la secuencia de los bits de estado, bit_1 =>"0" y bit_2=>"0", más el valor residual en binario, v_r"=>"01". El Listado 4.6 contiene la programación de la codificación del último píxel.

```
// * Codificación último píxel de la serie * //
// * Bits de estado: 00 -> Iguales y se codifica con 2 bits **** //
// * Bits de estado: 01 -> Iguales y Numero de bits: Log2((255-p1)/2 * //
// * Bits de estado: 10 -> Diferentes y se codifica con 2 bits *** //
// * Bits de estado: 11 -> Diferentes. Numero de bits: Log2((255-p1)/2) * //
// * Al finalizar del proceso actualizar el numbit como Log2((p7-p0)/2) * //
 iguales=' ', bit1= ' ', bit2= ' ', fin=  ' ';
 dif=0, marca=0, numbits=0, vr=0;
 num1=serie[0];
 num2= serie[x2-1];
 par (num1,num2);  //   llamada a la función de paridad
 vr= (num2 - num1) - (x2-2);
 media= vr/2.0;
 píxel_7=vr;
 aux3=media;
codifica_ultimo(bit1, bit2);  // Llamada a la función  de codificar píxel
```

Listado 4.6. Programación del último píxel de la serie

La salida binaria es actualizada con la codificación del último píxel y se almacena en secuencia: "10011101**0001**". El resto de los elementos de la serie son codificados utilizando una misma técnica de predicción para todos ellos.

Paso 3. Codificar el resto de elementos de la Serie S_i.

El algoritmo utiliza una misma técnica de predicción para el resto de los elementos de la serie S_i, desde p_2 a p_{n-1}. En el caso particular de que el píxel a predecir corresponda al píxel siguiente ($p_i = p_{i-1} + 1$), entonces es codificado directamente con dos bits de estado "00". Cuando el píxel a predecir es igual al actual más dos ($p_i = p_{i-1} + 2$) se codifica con el código "01". En los casos en los cuales la diferencia entre el píxel actual y el píxel a predecir sea mayor que dos, entonces es necesario añadir la codificación del valor residual. El valor residual v_r es calculado con la expresión 4.8:

$$v_r = \frac{(p_n - p_1 - 2)}{2} \tag{4.8}$$

Los bits de estado en este caso indican si existe paridad o no, bits de estado "10" indica paridad y "11" indica no paridad. Para determinar el número de bits N_b necesarios para codificar el valor residual, el número de bits es actualizado en cada cálculo del valor residual con la expresión 4.9:

$$N_b = \log_2\left(\frac{p_n - p_{i-1} - 2}{2}\right) \tag{4.9}$$

Si el número de bits en el paso anterior es menor de tres bits, entonces la diferencia entre el primer píxel y el último es considerada como muy pequeña y en consecuencia el bit de estado uno es codificado con un bit cero. En caso contrario, cuando la diferencia es grande debe ser codificada con el número de bits actualizado en el paso anterior. En la Tabla 4.7 se representan los predictores disponibles.

Tabla 4-7. Predictores disponibles para el resto de píxeles de la serie

Nº PREDICTOR	BITS DE ESTADO BIT_1	BIT_2	PREDICCIÓN	N_b
1	0	0	$p_i = p_{i-1} + 1$	2
2	0	1	$p_i = p_{i-1} + 2$	2
3	1	0	Paridad $v_r = (p_i - p_{i-1} - 2)/2$	$N_b = \log_2(p_n - p_{i-1} - 2)/2$
4	1	1	Paridad opuesta $v_r = (p_i - p_{i-1} - 2)/2$	Nb no es necesario porque no hay siguiente

En el primer predictor, los bits de estado "00" indican que el siguiente valor coincide con el siguiente píxel de la serie $p_i = p_{i-1} + 1$. En el segundo predictor, los bits de estado "01" indican que el siguiente píxel de la serie es igual al anterior más dos, $p_i = p_{i-1} + 2$. En el caso del tercer predictor, los bits de estado "10" significan que los píxeles tienen la misma paridad y el algoritmo debe actualizar el valor del píxel,

porque no se ha cumplido ni *"00"* ni tampoco *"01"*. El valor residual será almacenado con el cálculo del número de bits actualizado N_b. En el último predictor, los bits de estado "11" indican que la paridad de los dos píxeles es diferente y entonces el valor residual es calculado y representado igual que en el predictor anterior. El número de bits N_b no es necesario calcular porque no es actualizado para acotar el número de elementos con los que será representado el siguiente elemento de la serie.

El final de los elementos es detectado de forma ingeniosa sin tener que usar bits adicionales. Todos los elementos son obtenidos en función del último de la serie. Esto es, una vez obtenido el valor de predicción del penúltimo elemento de la serie, su valor residual es la diferencia con el último píxel de la serie, que anteriormente fue almacenado en el array con el índice uno. Si sumamos el valor residual al valor del penúltimo píxel entonces nos dará un valor igual al último píxel. En consecuencia, el final de la codificación de la serie es detectada.

La Tabla 4.8 refleja el resultado completo de los códigos binarios obtenidos por en el ejemplo propuesto mediante la técnica de predicción predictiva del Algoritmo INA.

Tabla 4-8. Tabla de predicción de valores de la serie

PÍXELES	v_r	BIT_1	BIT_2	VALOR	CÓDIGO	N_B
[157]				157	10011101	8
(157,163]	1	0	0	01	0001	4
161	1	0	1	11	1001	4
162		0	0		00	2
Código de salida de la Serie Si "100111010001100100"						18 bits

El resultado de la codificación predictiva de la serie S_i ocupa 18 bits y la codificación transversal directa del árbol binario A_i ocupa 28 bits. Entonces, el bloque codificado con el Algoritmo INA es almacenado en el fichero comprimido ocupando un total 46 bits. Por otra parte, el bloque original de píxeles ocupa de 128 bits (16 píxels x 8 bits/píxel = 128 bits).

$$\text{Ratio Compresión} = \frac{\text{Tamaño del Bloque Original}}{\text{Tamaño del Bloque Codificado}} = \frac{128}{46} = 2{,}78 \text{ bits}$$

En conclusión, el ratio de compresión obtenido por el Algoritmo INA es muy aceptable. Esto significa que supera ampliamente el ratio de 2:1 para un bloque cualquiera con 4 píxeles diferentes. El Listado 4.7 contiene la programación de la tabla 4.8

```
// * Codificación píxeles de la serie * //
// * Bits de estado: 00 -> Siguiente píxel p2=p1+1. * //
// * Bits de estado: 01 -> Siguiente píxel p2=p1+2. * //
// * Bits de estado: 10 -> Iguales ((p2-p1)/2)   Numbits > 2 bits * //
// * Bits de estado: 11 -> Iguales ((p2-p1)/2).  Numbits >2 bits  * //
// * actualizar el numbits como Log2((p7-pi)/2) * //
for (z=1; z<x2-1; z++)// ciclo para codificar todos los píxeles x2-1
{
   vr= (num2 - num1);
   media= vr/2.0;
   if (media==0)
      numbits=0;
   else
      numbits= (log(media)/log(2.0) + 1);
   final=' ';
  if (num2-num1==1)  //  píxel igual al anterior +1
     bit1='0, bit2='0', marca=1;
  if (num2-num1==2)     // (píxel anterior + 2)
      bit1='0', bit2='1', marca=1;
  if (iguales=='S' && (num2-num1)>2 && marca!=1) // Paridad y>2 píxeles
     bit1='1', bit2='0', marca=1;
  if (iguales=='N' && (num2-num1)>2 && marca!=1) // No Paridad e y>2
     bit1='1', bit2='0', marca=1;
   marca=0
   aux3=media;
   if (final !='f')
      codifica_píxel(bit1, bit2);  // función  de codificar píxel
   vr= serie[x2-1] - (num2+1);
   if (vr <2 || (serie[x2-2] - (num2+1) - (x2-2-(z+1))==0))
      z=x2+1;
 }
```

Listado 4.7. Programación de los píxeles de la serie

En algunos casos especiales, los valores de la serie podrían ser codificados con un menor número de píxeles.

4.3.1.1.6 Codificación de casos especiales

En la codificación de la serie existen varios casos especiales que surgen de la disposición de los píxeles en la serie y que permiten simplificar y reducir la codificación significativamente. Los casos más especiales son considerados como recursos de optimización. Los casos más favorables surgen cuando solo hay uno o dos elementos en la serie S_i o cuando todos los píxeles de la serie son consecutivos. Por el contrario, los casos más desfavorables surgen cuando todos los píxeles son diferentes y no consecutivos. Obviamente este caso desfavorable lo es también para el resto de los métodos convencionales debido a que no existe redundancia en los datos. A continuación se describe detalladamente el comportamiento del algoritmo en cada uno de esto casos mencionados.

Caso especial 1. Todos los valores de los píxeles del bloque B_i son iguales

La serie S_i contendrá un único elemento. En este caso especial, el valor residual v_r es igual a 1. Sustituyendo en la expresión (5-10)

$$v_r = p_n - p_1 - (n-2) = p_n - p_n - (n-2) = 1 \quad (4.10)$$

siendo *n* el número de píxeles diferentes

Supongamos la siguiente serie: S_i*={161}, entonces* v_r= (161-161-(1-2))= 1. El codificador genera la siguiente secuencia de bits "0000".

00 -> Bits de estado

00 -> Valor residual

Este caso es el más especial de todos porque la serie S_i es codificada con el menor número de bits posibles, 12 bits (4 bits + 8 bits del elemento p_1) en lugar de los 128 bits necesarios para codificar el bloque original. En consecuencia, el bloque de píxeles B_i no requiere ser codificado mediante el árbol binario.

El algoritmo usa un *flag* de un bit para indicar si los siguientes bloques son iguales al anterior. El codificador salta la codificación y genera un *Run Length* de bloques.

Caso especial 2. Todos los elementos de la serie S_i *son consecutivos*

El valor de cada píxel es igual al anterior más uno y por tanto el valor residual es igual a 1.

$$p_1 < p_2 < p_3 < \ ... < p_{n-1} < p_n \quad (4.11)$$

$$v_r = p_n - p_1 - (n-2) = p_n - p_n - (n-2) = 1 \quad (4.12)$$

En ambos casos especiales, el resultado de la ecuación anterior es uno. Sin embargo, son codificados con diferentes bits de estado porque el codificador tiene información sobre el número de píxeles del bloque.

Ejemplo : S_i*={161<162<163<164<165},* v_r*= (165-161)-(5-2))= 1*

La codificación especial es la siguiente:

00 -> Bits de estado (paridad o no)
01 -> Valor residual
1 -> Flag (exclusivo para la segmentación variable)

En este caso especial, la recuperación del píxel sigue siendo reversible porque no existe incertidumbre en el código de salida. Sin embargo, no es conocido el número de elementos consecutivos. La codificación del bloque aporta esta información sobre el número de elementos de la serie, ya que es deducido de la codificación de las ocho duplas de píxeles. En caso de segmentación variable es necesario añadir un flag de un bit para indicar el final. La tabla 4.9 muestra un caso práctico de píxeles consecutivos.

Tabla 4-9. Codificación especial de una serie S_i con píxeles consecutivos

CASO	PÍXELES SERIE S_I	VALOR RESIDUAL = 1	CODIFICACIÓN DE v_r CON 2 BITS
1	{161,162}	v_r= (162-161)-(2-2)= **1**	Bits de estado: 10 — v_r: 01
2	{161, 162, 163}	v_r=(163-161)-(3-2)= **1**	Bits de estado: 00 — v_r: 01
3	(161,162,163,164)	v_r= (164-161)- (4-2)=**1**	Bits de estado: 10 — v_r: 01
4	(161,162,163,164,165)	v_r = (165-161)-(5-2)=**1**	Bits de estado: 00 — v_r: 01
5	{161,163]	v_r = (163-161)-(2-2)= 2	Caso NO especial
6	(161,163,164,165)	v_r = (165-161)-(4-2)=**2**	Caso NO especial, falta el valor 162

En los cuatro primeros casos todos los píxeles de cada serie son consecutivos y por tanto el valor residual es igual a uno. En los dos últimos casos, y aunque la serie tenga dos elementos al no ser estos consecutivos, entonces $v_r>1$. En cada predicción, el algoritmo actualiza el número de bits necesario para codificar el píxel actual por cada píxel predecido.

$$N_b = \log_2(p_n - p_i - (n-2)) \tag{4.13}$$

Caso especial 3. Todos los píxeles de la serie S_i son diferentes

La codificación de este caso representa una de las grandes ventajas del método con respecto a los métodos de compresión convencionales y especialmente con respecto al estándar JPEG-LS. Las imágenes digitales de continuos tonos, como su nombre indica, presentan pequeñas variaciones entre los píxeles adyacentes y es frecuente encontrar zonas de la imagen en las que todos los píxeles son diferentes.

La Tabla 4.10 refleja la distribución de los 32.768 bloques obtenidos mediante una segmentación fija de 2x4 píxeles para imágenes fotográficas de 512x512 píxeles (512*512 píxeles/8 (píxeles/bloque)=32.768 bloques).

Tabla 4-10. Número de bloques con diferentes píxeles por serie/bloque

NOMBRE IMAGEN	NÚMERO DE BLOQUES FIJOS DE X2X4 PÍXELES							
	1 PÍXEL	2 PÍXELES	3 PÍXELES	4 PÍXELES	5 PÍXELES	6 PÍXELES	7 PÍXELES	8 PÍXELES
Airplane	67	1.045	2.909	5.052	6.207	6.240	5.979	5.269
Baboon	0	0	5	149	903	3.839	10.449	17.423
Barbara	0	17	322	1.751	4.160	6.151	8.760	11.697
Boats	0	12	180	1.071	3.428	7.816	10.994	9.267
Goldhill	16	35	329	1.377	3.511	7.404	11.097	8.999
Lena	2	60	598	2.588	5.876	8.345	8.541	6.758
Peppers	1	40	458	1.698	4.911	9.287	9.983	6.390
Zelda	0	8	210	1.757	5.868	9.969	9.779	5.177

Los resultados mostrados en la Tabla 4.10 reflejan que todas las imágenes fotográficas evaluadas contienen un gran número de bloques con 6, 7 y 8 píxeles diferentes.

Esta es la problemática a la que se enfrentan los métodos convencionales para tratar de incrementar los ratios de compresión en imágenes de continuos tonos. En estos casos, la respuesta del estándar JPEG-LS no es la más favorable porque el predictor del píxel *x* en la mayoría de los casos no será uno de los vértices del plano de predicción formado por los vértices *a*, *b*, c y *x*.

Alternativamente, el algoritmo propuesto toma ventaja de este caso especial porque como demostraremos a continuación reduce considerablemente el número de bits eliminando las ramas del árbol codificadas en cada nodo. Según la propiedad del árbol binario completo, en el caso de que todos los píxeles sean diferentes entonces cada uno estará representado en una rama distinta del árbol binario. Por tanto, el algoritmo de codificación del árbol binario elimina las dos ramas activas del nodo codificado y desaparecen del árbol binario reduciendo automáticamente en cada paso dos bits de la codificación del siguiente nodo. Más aún, el último nodo no es necesario codificarlo porque contendrá los dos píxeles de la serie que faltarán por codificar.

En el siguiente ejemplo simplificado a 8 píxeles se ilustra el principio de funcionamiento de la codificación de un bloque cualquiera formado por ocho píxeles con valores diferentes.

$$\begin{bmatrix} 154 & 153 & 152 & 160 \\ 157 & 158 & 155 & 151 \end{bmatrix} \Longrightarrow \qquad S_i=\{151<152<153<154<155<157<158<160\}$$

La codificación progresiva reduce en cada codificación el tamaño del árbol binario.

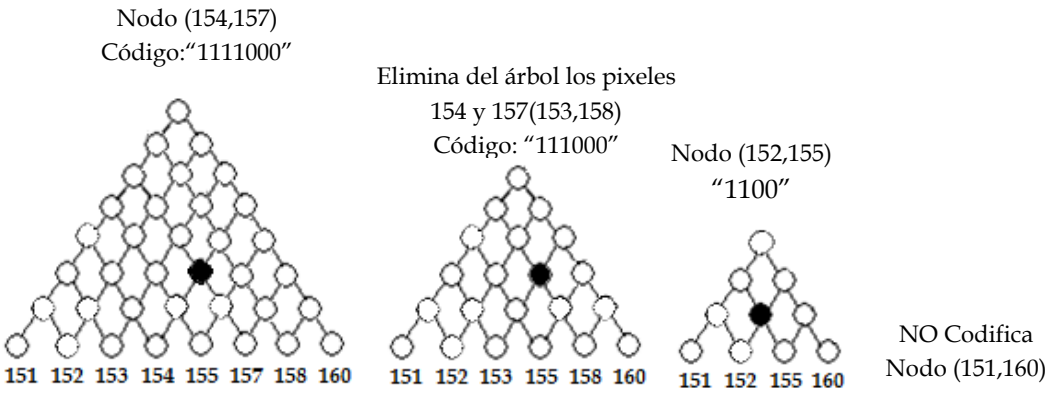


Figura 4.25. Codificación especial cuando todos los píxeles son diferentes

Esta figura pone de manifiesto los beneficios de la codificación progresiva del árbol binario con todos sus píxeles diferentes. El último nodo del árbol A_i no es necesario codificarlo porque es conocido el valor de los dos píxeles de la serie S_i que faltarían por ser codificados. Aplicando esta misma metodología, el algoritmo de codificación general podría obtener directamente el código del nodo final del arbol A_i siempre que los índices no hayan sido codificados previamente (caso especial 4).

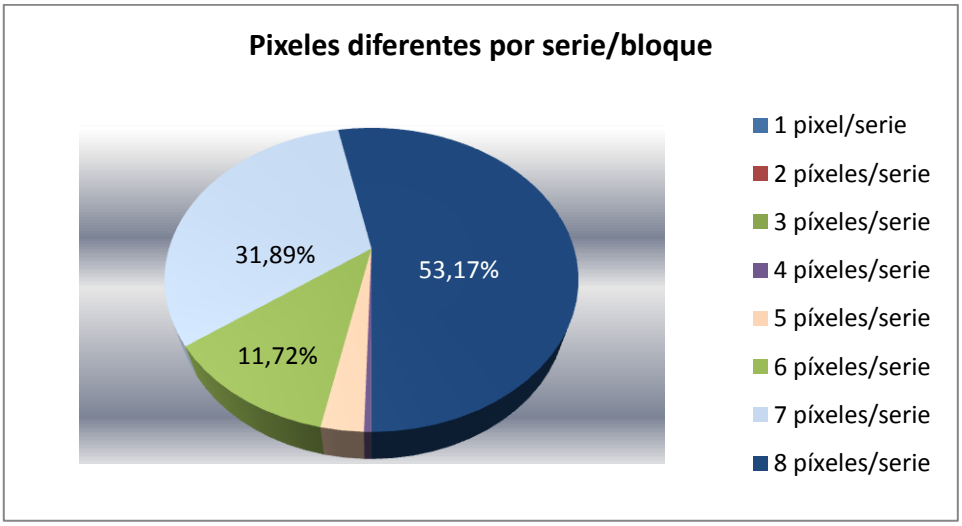


Figura 4.26. Gráfica de porcentajes significativos de píxeles por bloque en la imagen Baboon

La Tabla 4.10 muestra como las imágenes Baboon y Barbara contienen el mayor número de bloques con todos sus píxeles diferentes. La Fig. 4.26 recoge estos datos significativos en forma de porcentaje para la imagen Baboon. El Algoritmo INA toma ventaja de esta característica frente a los métodos tradicionales y en especialmente con respecto al estándar de compresión JPEG-LS. La tabla refleja que el resto de las imágenes tiene el mayor nmero de bloques con 7 y 6 píxeles diferentes por lo que el algoritmo propuesto toma ventaja sobre el resto de métodos.

Caso especial 4. El último nodo contiene uno o dos valores de píxeles no utilizados previamente en la codificación del árbol.

La propiedad de árbol binario completo implica que todos los píxeles intervienen al menos en un nodo activo del árbol binario. Esto significa que después de codificar los tres primeros nodos o bien los siete primeros nodos según la versión de segmentación, si faltaran dos píxeles por intervenir entonces se deducen automáticamente su pertenencia al último nodo y no es necesario codificarlos. Esta característica se produce con alta probabilidad en los bloques con siete y seis píxeles diferentes. El orden de los nodos p_h y p_l es desconocido, por tanto el decodificador debe ser informado y para ello el codificador añade un bit de control en el fichero comprimido.

4.3.1.2 Flujograma del proceso de compresión

El flujograma presentado en esta sección corresponde a la ejecución secuencial del proceso de compresión. Las versiones en paralelo ejecutan los tres procesos secuenciales simultáneamente en *n* procesadores.

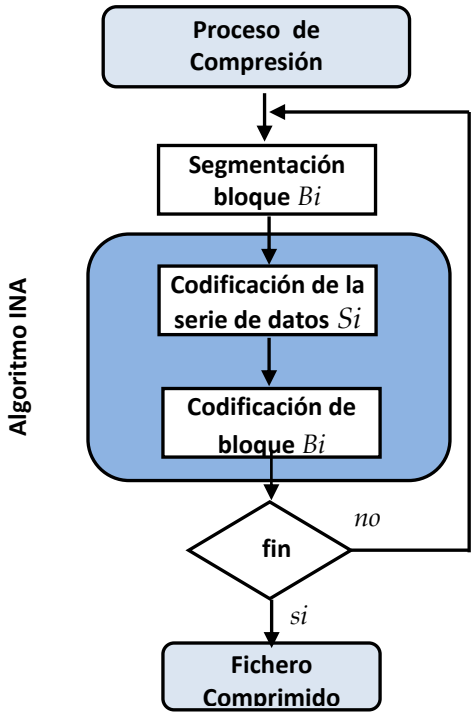


Figura 4.27. Flujograma del proceso de compresión

4.3.1.3 Ejemplo práctico

A continuación se describen los resultados obtenidos en dos ejemplos prácticos que ilustran el funcionamiento de ambas versiones del método. Los píxeles son los obtenidos desde el inicio de la imagen Lena. El primer ejemplo utiliza una

segmentación de bloques fijos de 4x4 píxeles y el segundo utiliza una segmentación variable con un máximo de 8 píxeles diferentes.

Ejemplo 1. Segmentación en bloques fijos de 4x4 píxeles

El primer bloque de tamaño de 4x4 píxeles de la imagen Lena está formado por los siguientes píxeles.

$$B_i = \begin{bmatrix} \begin{pmatrix}162\\162\end{pmatrix} & \begin{pmatrix}162\\162\end{pmatrix} & \begin{pmatrix}162\\162\end{pmatrix} & \begin{pmatrix}161\\161\end{pmatrix} \\ \begin{pmatrix}162\\162\end{pmatrix} & \begin{pmatrix}162\\162\end{pmatrix} & \begin{pmatrix}162\\162\end{pmatrix} & \begin{pmatrix}161\\161\end{pmatrix} \end{bmatrix}$$

Número de píxeles del bloque $B_i = 16$ Serie $S_1 = \{161,162\}$

Tabla 4-11. Datos probabilísticos del bloque con segmentación fija

ÍNDICE	VALOR	FRECUENCIA	PROBABILIDAD	ENTROPIA_PÍXEL
0	161	12	0,75	-0,415037
1	162	4	0,25	-2,000000
		---------	----------	----------
Suma		18	1,000000	-2,415037

Aplicando el algoritmo propuesto, obtenemos la Tabla 4.12.

Tabla 4-12. Codificación del bloque fijo de 4x4 píxeles

ÍNDICES	DUPLAS	ORDEN_P1	ORDEN_P2	BITS	CÓDIGO OPT.
1	[162,162]	2	2	2	11
2	[162,162]	2	2	2	11
3	[162,162]	2	2	2	11
4	[161,161]	1	1	2	00
5	[162,162]	2	2	2	11
6	[162,162]	2	2	2	11
7	[162,162]	2	2	2	11
8	[161,161]	1	1	2	00
Salida binaria A_1 = {1111110011111100}				16	

La codificación de la serie S_i mediante la técnica de predicción lineal INA se refleja en la Tabla 4.13.

Tabla 4-13. Predicción lineal de la serie con segmentación fija

PÍXELES		NUMBITS	BIT1	BIT2	VALOR	CÓDIGO	SUMA_BITS
[161,]		8 bits				10100001	8
[161,162]	2	2	0	0	0	000	3
Salida binaria de la serie S_1= "10100001000" …							11 bits

El resultado obtenido de la compresión de la serie mediante la técnica de predicción lineal propuesta ocupa 11 bits y el resultado de la codificación del bloque ocupa 16 bits. El ratio de compresión obtenido es 128/27= 4,74 que supone una reducción muy significativa.

Ejemplo 2. Segmentación en bloques variables con 8 píxeles diferentes por bloque

El primer bloque resultante de la segmentación variable de los primeros píxeles de la imagen Lena es el siguiente:

Serie S_1 = {155,156, 157,160, 161, 162, 163,166}
B_1 = {162,162,162,161,162,157,163,161,166,
162,162,160,155,163,160,155,157,156}
Numero de píxeles del bloque = 18 [0-17]

Tabla 4-14. Datos probabilísticos del bloque con segmentación variable

ÍNDICE	VALOR	FRECUENCIA	PROBABILIDAD	ENTROPIA_PÍXEL
0	155	2	0,111111	-0,352214
1	156	1	0,055556	-0,231662
2	157	2	0,111111	-0,352214
3	160	2	0,111111	-0,352214
4	161	2	0,111111	-0,352214
5	162	6	0,333333	-0,528321
6	163	2	0,111111	-0,352214
7	166	1	0,055556	-0,231662
		---------	----------	----------
Suma		18	1,000000	-2,752715

Aplicando el algoritmo al bloque variable se obtiene la Tabla 4.15.

Tabla 4-15. Codificación del bloque variable con 8 nodos

Nº ÍNDICE	PAR PÍXELES	ORDEN P1	ORDEN P2	CODEWORD CLÁSICO	CODEWORD MAYOR	BITS
1	[162,162]	6	6	1111110		7
2	[162,161]	6	5		111100	6
3	[162,157]	6	3		111000	6
4	[163,161]	7	5		1000000	7
5	[166,162]	8	6		11111100	8
6	[162,160]	6	4		110000	6
7	[155,163]	1	7	10000000		7
8	[160,155]	4	1		1110	4
9	[157,156]	3	2		100	3
B_1 = {111111011110011100010000001111110011000010000000111010 0}						54 bits

Los 54 bits obtenidos en la codificación requieren ocho bits adicionales de control. La compresión de los nodos del bloque son reflejados en la Tabla 4.16.

Tabla 4-16. Predicción lineal de la serie con segmentación variable

PÍXELES	NUMBITS	BIT1	BIT2	VALOR	CÓDIGO	SUMA_BITS
[155,]	8 bits				10011011	8
[155,166]	1	1	0	4	1010	4
[156]	2	0	0		00	2
[157]	2	0	0		00	2
[160]	2	1	0		10	2
[161]	2	0	0		00	2
[162]	2	0	0		00	2
[163]	2	0	0		00	2
Código de salida de la serie S_1 = "100110111010000010000000"						24 bits

El resultado de la codificación del primer bloque en la versión de segmentación variable ocupa 83 bits. El bloque original contiene 18 píxeles que ocupan 144 bits (18 píxeles -> 18 bytes -> 144 bits). Esto significa que factor de compresión es 144/86= 1,674.

El fichero binario de salida está compuesto por dos campos: cabecera y datos. La cabecera contiene la información referente a la versión del algoritmo utilizada en el proceso de compresión y ocupa el primer byte del fichero comprimido. La cabecera contiene bits de control y bits de datos, como se muestra en la Tabla 4.17.

Tabla 4-17. Formato de la cabecera del fichero comprimido

CABECERA DEL FICHERO COMPRIMIDO (1 BYTE)							
Bit 7	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit2**	**Bit 1**	**Bit 0**
Modo de ejecución	Clase de segmentación	Hilos de programación	Núcleos	Cantidad de microprocesadores			

El *nible* superior contiene los bits de control y el *nible* inferior contiene la información sobre el número de procesadores usado en el proceso de compresión. La implementación del algoritmo añade cuatro bytes de datos más en la cabecera que suministran información sobre la resolución de la imagen (filas x columnas), formato de imagen, etc. A continuación se describe la función de cada uno de los bits de control.

Bit 7: modo de ejecución. El bit más significativo, MSB, indica el tipo de ejecución del algoritmo.

$$Modo\ de\ ejecución \begin{cases} 0 \rightarrow Ejecución\ secuencial \\ 1 \rightarrow Ejecución en\ paralelo \end{cases}$$

Bit 6: clase de segmentación. Este bit informa del tipo de segmentación empleada en el proceso de compresión.

$$Segmentación \begin{cases} 0 \rightarrow Segmentación\ fija \\ 1 \rightarrow Segmentación\ variable \end{cases}$$

Bit 5: hilos de programación. Este bit informa sobre el número de hilos de programación usados en el algoritmo.

$$Hilos\ de\ programación \begin{cases} 0 \rightarrow un\ hilo \\ 1 \rightarrow dos\ hilos \end{cases}$$

Bit 4: núcleos. Indica el número de núcleos utilizados por procesador.

$$Modo\ de\ ejecución \begin{cases} 0 \rightarrow un\ núcleo \\ 1 \rightarrow dos\ núcleos \end{cases}$$

Bits 3 a Bit 0: Indican la cantidad de procesadores usados en la versión de ejecución en paralelo

La cabecera del fichero comprimido suministra información de la imagen original y de los datos comprimidos necesaria para el proceso de descompresión. El algoritmo en la versión de compresión secuencial con un solo hilo de programación

almacenará consecutivamente la codificación binaria de la serie y la del bloque respectivamente.

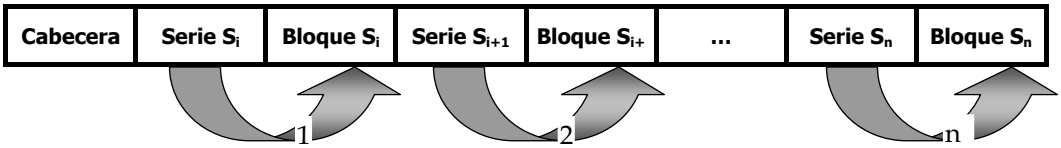


Figura 4.28. Estuctura del procesamiento con un hilo de programación

La ejecución del algoritmo en paralelo con *m* núcleos almacenará sus respectivas codificaciones binarias en grupos de *m* series seguidos por *m* bloques. La configuración del fichero de salida presenta la estructura de la Fig. 4.29.

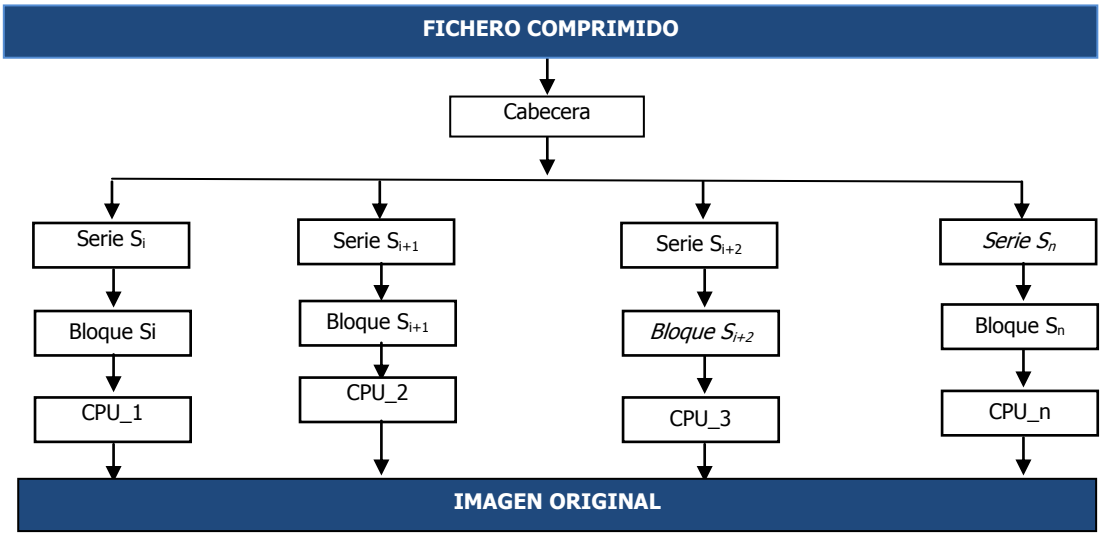


Figura 4.29. Estuctura de procesamiento en paralelo

Una vez finalizado el proceso de compresión, el Algoritmo INA en la versión *in-place* está en disposición ser utilizarlo como parte de pre-procesamiento de los datos para optimizar los resultados de compresión de algunos algoritmos convencionales.

4.3.1.4 Optimización de algoritmos convencionales

Los algoritmos de compresión de imágenes sin pérdida de datos son algoritmos finalistas porque no es posible *comprimir después de la compresión* y mucho menos aplicar un algoritmo distinto al utilizado en el ciclo de compresión.

En esta sección, se describe una ventaja del Algoritmo INA sobre el resto de algoritmos que consiste en optimizar los ratios de compresión de algunos de los métodos tradicionales, aunque los ratios conseguidos por estos siempre serán inferiores a los obtenidos por el Algoritmo INA.

La idea básica de la optimización está focalizada en la etapa de segmentación en la cual se produce el pre-procesamiento y preparación de los datos antes de aplicar el Algoritmo INA. El algoritmo realiza los mismos pasos descritos en el proceso de compresión codificando las series S_i de píxeles diferentes. Sin embargo, los bloques originales B_i no son comprimidos aplicando una estructura de árbol binario A_i, sino que cada píxel del bloque es reemplazado *in-place* por su correspondiente índice en la serie.

El alfabeto fuente de 256 píxeles (8 bits/píxel) es reducido a un máximo 16 píxeles posibles (4 bits/píxel), y por consiguiente, la longitud de los códigos será menor. Obviamente, en el proceso de optimización sacrificamos el tiempo de compresión en favor del ratio de compresión. En el siguiente ejemplo se ilustra el principio de funcionamiento del Algoritmo INA *in-place* aplicado al primer bloque de la imagen *Lena* para ambas versiones de segmentación.

$$\text{Bloque } B_1 = \{162,162,162,161,162,157,163,161,166,162,162,160,155,\\ 163,160,155,157,156\}$$

La serie S_1 resultante del proceso de segmentación estará formada por los siguientes índices que apuntan a los siguientes valores de los píxeles.

Tabla 4-18. Índices y valores de la serie S_1 en la compresión *in-place*

ÍNDICE	0	1	2	3	4	5	6	7
Píxeles de la serie S_1	155	156	157	160	161	162	163	166

Al aplicar el Algoritmo INA *in-place* los valores de los píxeles son sustituidos en el *array* de la imagen original por sus correspondientes índices.

$$\text{Serie } S_1 = \{5,5,5,4,5,2,6,4,7,5,5,3,0,\ 6,3,0,2,1\}$$

En el ejemplo de segmentación en bloques fijos, los resultados obtenidos son los siguientes:

$$\begin{bmatrix} \binom{162}{162} & \binom{162}{162} & \binom{162}{162} & \binom{161}{161} \\ \binom{162}{162} & \binom{162}{162} & \binom{162}{162} & \binom{161}{161} \end{bmatrix} \Longrightarrow \begin{bmatrix} \binom{1}{1} & \binom{1}{1} & \binom{1}{1} & \binom{0}{0} \\ \binom{1}{1} & \binom{1}{1} & \binom{1}{1} & \binom{0}{0} \end{bmatrix}$$

$$\text{Serie } S_1 = \{161,162\} \Longrightarrow \text{Serie } S_1 = \{0,1\}$$

Bloque $S_1 \approx 48$ bits (3 bits/píxel) Bloque $S_1 \approx 16$ bits (1 bit/píxel)

El bloque original B_1 formado por 16 píxeles con 8 bits de profundidad se transforma en un bloque con 1 bit/píxel. Teniendo en cuenta que la codificación de la serie S_1 usa 10 bits utilizando el Algoritmo INA y que cualquier codificador convencional emplearía 16 bits (1 bit/píxel) para codificar el bloque. El resultado de

la codificación de la serie S_i más la sustitución de los bloques por sus índices codificados con con un método tradicional, Codificador Huffman o Aritmético, mejora los ratios de compresión obtenidos por el método tradicional.

En esta sección hemos analizado y abordado el proceso de compresión de imágenes mediante el algoritmo propuesto. La descripción formal del Algoritmo INA ha sido expuesta y evaluada paso a paso. El principio de funcionamiento del algoritmo ha sido desarrollado a través de los tres primeros bloques de la imagen Lena. Los resultados experimentales del proceso de compresión son recogidos en la sección 4.5 y comparados con los resultados del estándar JPEG-LS. En la siguiente sección se analiza el proceso reversible.

4.4 Proceso de descompresión

El proceso de descompresión consiste en recuperar los datos de la imagen original de modo reversible a partir del fichero comprimido binario generado en el proceso de compresión. La imagen original y la imagen recuperada deben ser exactamente iguales, bit a bit.

El proceso de descompresión no es inversamente simétrico al proceso de compresión (Fig. 4.31). En el ciclo de descompresión no existe el proceso de segmentación y por tanto no requiere ordenar los píxeles. En consecuencia, como en la mayoría de los métodos convencionales el proceso de descompresión es mucho más rápido que el proceso de compresión (Larrauri, 2012).

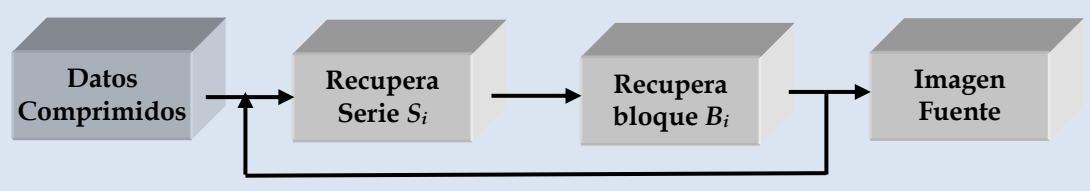


Figura 4.30. Diagrama general del proceso de descompresión

El ciclo de descompresión comienza con la lectura del primer byte que corresponde a la cabecera del proceso de compresión. La cabecera contiene los códigos de control que definen las características en las que se ha realizado el ciclo de compresión. Atendiendo a los valores de los bits de control se definen las condiciones en las cuales debe realizarse el proceso de descompresión.

El procesamiento completo del fichero comprimido es realizado secuencialmente recuperando intercaladamente cada bloque de datos con su correspondiente serie de píxeles. El decodificador inicialmente no tiene información previa de las características de los píxeles con relación al tamaño y la longitud de los códigos como ocurría en el proceso de compresión. La clave del algoritmo de

descompresión radica en la eliminación de esta incertidumbre en la recuperación de los datos. Para eliminar esta incertidumbre el ciclo de descompresión aprovecha una de las características del algoritmo que combina la decodificación del bloque de datos con la serie asociada. Los bloques codificados corresponden a una codificación fija de ocho o cuatro pares de píxeles. Por tanto, la codificación preserva el número de píxeles de la serie. Tomando ventaja de esta información es posible decodificar la serie completa asociada al bloque. La Fig. 4.31 muestra el diagrama de bloques del proceso de descompresión.

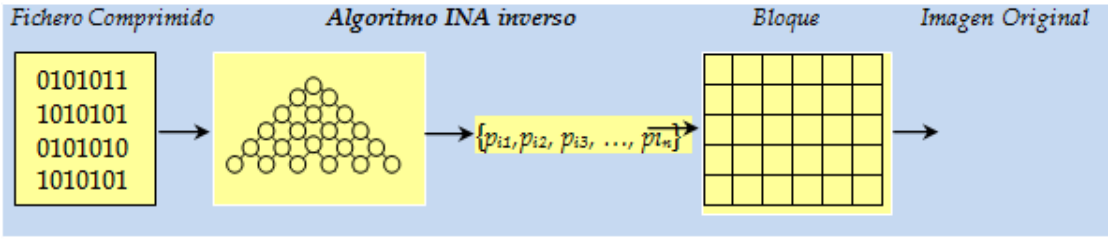


Figura 4.31. Diagrama de bloques del proceso de descompresión

El algoritmo inverso de predicción procesa datos del fichero comprimido a nivel de bit. La descompresión del archivo comprimido extrae la serie *Si* y el bloque B_i recuperando secuencialmente bloque a bloque la imagen original.

4.4.1 Algoritmo INA inverso

El algoritmo de descompresión en la versión secuencial básica recupera los valores de los píxeles de la serie S_i y a continuación los valores del bloque B_i. La expansión de los datos opera a nivel de bit y la salida del decodificador genera valores numéricos representados en formato decimal (píxeles).

Pasos del algoritmo de descompresión:

Paso 1. Recuperar los índices de cada nodo. El número de bits a uno representa el valor del índice del primer píxel y el número de ceros menos uno representa el índice del segundo píxel (codificación transversal inversa directa)..

Paso 2. Recuperar los píxeles diferentes de la serie de valores y expandirlos en el bloque

Paso 3. Sustituir en el bloque B_i los índices recuperados en el paso 1 por los valores de los píxeles de S_i recuperados en el paso 2.

El algoritmo de descompresión contiene únicamente dos pasos ya que no realiza el paso de ordenación de los píxeles. En consecuencia, el proceso de descompresión siempre será más rápido que el proceso de compresión. El algoritmo inverso en un

primer paso realiza la lectura de los datos bit a bit hasta encontrar el final de la serie S_i. Los píxeles son recuperados en orden por medio de sus respectivos índices en el *array* de la serie S_i. En el segundo paso, el algoritmo inverso continúa procesando los datos correspondientes a la codificación del bloque B_i. Cada código es una concatenación de unos y ceros, el número de unos consecutivos corresponde al índice del primer píxel y el número de ceros corresponde al índice del segundo píxel en el nodo.

Mediante punteros al *array* de la serie con sus respectivos índices recupera los píxeles del nodo. En la Fig. 4.32 se muestra la estructura del algoritmo de descompresión con uno y dos hilos de programación.

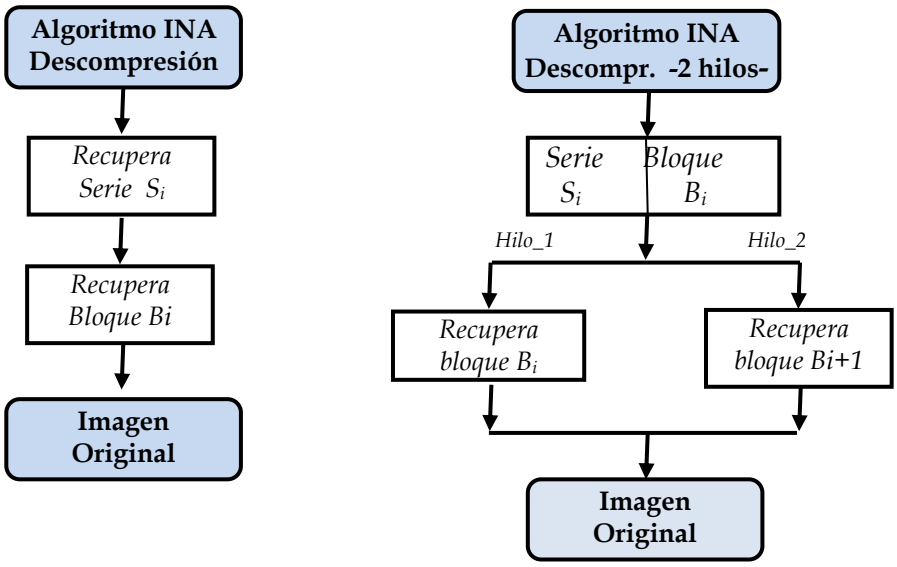


Figura 4.32. Algoritmo INA para la ejecución secuencial y con dos hilos

4.4.1.1 Descripción formal del algoritmo de descompresión

El Algoritmo INA inverso realiza dos pasos para recuperar las dos estructuras de datos generadas en el proceso de compresion: recuperar la serie y recuperar los píxeles del bloque que contiene el valor de los índices. La descripción del algoritmo inverso es acompañada del mismo ejemplo propuesto en el algoritmo de compresión. El algoritmo de predicción inverso de la serie es analizado exhaustivamente en el paso 1 y el algoritmo inverso de codificación transversal es analizado en el paso 2.

4.4.1.2 Paso 1. Recuperar los píxeles de la serie S_i

Este paso tiene por objeto recuperar los píxeles diferentes de la serie S_i. Para ello, aplica el algoritmo de predicción inverso que consiste en la lectura de los datos binarios del fichero comprimido.

La técnica predictiva de decodificación consiste en tres pasos ejecutados en el mismo orden que en el proceso de codificación de la serie.

Paso 1.1. Recuperar el primer píxel de la serie p_1
Paso 1.2. Recuperar último píxel de la serie p_n
Paso 1.3. Recuperar el resto de los píxeles de la serie (p_2 a p_{n-1}).

En la Fig. 4.33 se ilustra el principio de funcionamiento de la técnica de predicción propuesta para codificar la serie S_i.

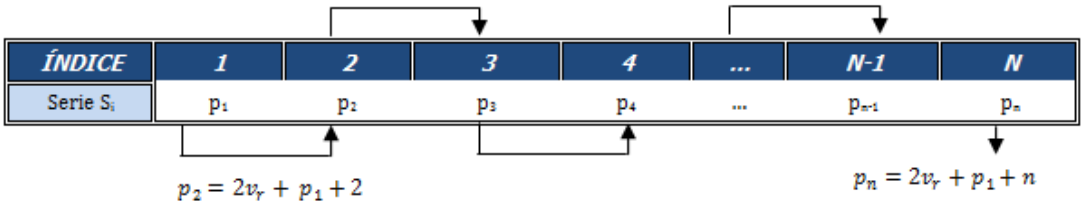


Figura 4.33. Técnica de predicción lineal

Paso 1.1. Recuperar el primer píxel p_1 de la serie S_i.

El algoritmo posiciona el puntero en el inicio de los datos comprimidos de la serie para recuperar el valor del primer píxel. Este valor es almacenado en el array de descompresión.

En el ejemplo propuesto el código binario codificado de la Serie S_i es el siguiete: "100111010001100100". Los ocho primeros bits "10011101" corresponden al primer píxel, representado en decimal p_1= 157.

Fich. Comprimido:	10011101 0001100100"
Puntero	
p_1	**p_1= 157**

Figura 4.34. Recuperación del primer píxel de la serie

La salida del decodificador recupera el primer píxel de la serie S_i.={157}.

Paso 1.2. Recuperar el último píxel p_n de la serie

En el proceso de compresión el último píxel de la serie fue almacenado a continuación del primero con objetivo de calcular el máximo número de bits necesarios para codificar el resto de los elementos de la serie. La codificación de este elemento contiene información de los bits de estado y del valor residual.

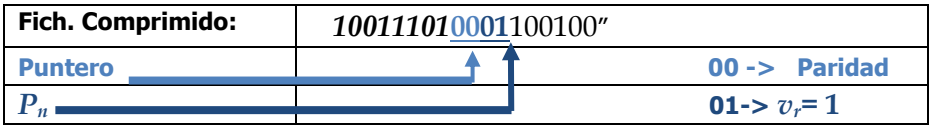


Figura 4.35. Recuperación del último píxel de la serie

El puntero examina los dos primeros bits "00" que indican paridad en los píxeles y que v_r se representará con dos bits. El algoritmo desplaza el puntero en dos posiciones hacia adelante y extrae el valor residual en binario "01", por tanto, v_r= 1. Aplicando la expresión (4.5) se obtiene el último píxel:

$$v_r = \frac{p_n - p_1 - n}{2} \quad \Rightarrow \quad p_n = 2v_r + p_1 + 4 = 163$$

Los dos elementos tienen paridad impar y en consecuencia es directamente reversible. La salida del decodificador recupera el último píxel de la serie S_i.={157, …, 163}.

Paso 1.3. Recuperar el resto de los píxeles de la serie

El resto de los elementos de la serie codificada contienen una primera parte con la información de los bits de estado y una segunda parte con los valores residuales. El proceso de descompresión aplica la misma tabla de predictores que el proceso de compresión para recuperar el resto de los píxeles. En el ejemplo propuesto se obtiene:

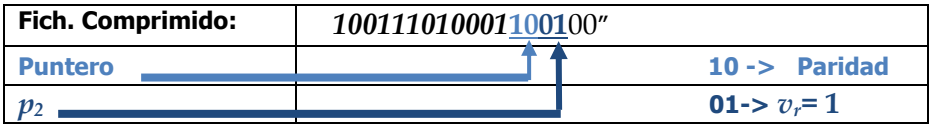
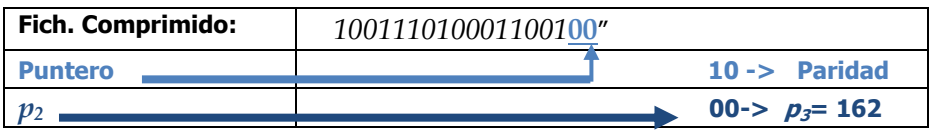


Figura 4.36. Recuperación del resto de píxeles de la serie

El predictor apuntado en binario es "10" indicando que tienen la misma paridad (157, 161) pero la diferencia entre ambos es mayor que 2. Los dos siguientes bits "01" representan el valor residual v_r=1 siendo N_b=2 bits. Aplicando la operación inversa de predicción para los píxeles comprendidos entre entre p_2 y p_{n-1}, incluidos estos mismos, obtenemos:

$$v_r = \frac{(p_i - p_{i-1} - 2)}{2} \qquad p_2 = 2v_r + p_1 + 2 = 161$$

La salida del decodificador recupera el último píxel de la serie S_i.={157, 161,…, 163}.



Por último, el algoritmo se posiciona en los siguientes bits de estado "00" que indica que el píxel a predecir es el siguiente. Este último elemento únicamente puede ser 162 porque debe ser mayor que 161 y menor que 163. Utilizando el recurso de optimización el codificador no tendría que decodificarlo. La salida del decodificador recupera el penúltimo píxel de la serie S_i.={157, 161,162, 163}. Todos los píxeles han sido recuperados satisfactoriamente sin pérdida de datos.

Una vez obtenidos todos los valores de la serie se procede a recuperar el bloque original.

4.4.1.3 Paso 2. Recuperar los índices de cada nodo del árbol binario

El algoritmo de decodificación del árbol binario recupera todos los nodos del árbol sustituyendo la secuencia de unos y ceros por sus correspondientes índices y que en el paso final del algoritmo de descompresión serán sustituidos por sus píxeles originales en el bloque.

La secuencia completa a decodificar es una sucesión de unos y ceros por cada nodo existente. Cada nodo del árbol binario contiene como mínimo de uno(s) seguido de cero(s). La recuperación de los datos originales del bloque es completada cuando todos los nodos hayan sido decodificados. Atendiendo al ejemplo propuesto, la codificación del árbol binario obtenida en el proceso de compresión es la siguiente A_i = "1110101111011011101011110110".

Tabla 4-19. Formato del bloque codificado

	1110	10	11110	110	1110	10	11110	110
Códigos	110	10	1110	110	110	10	1110	110
Nodos	N_1	N_2	N_3	N_4	N_5	N_6	N_7	N_8
Duplas	(3,3)	(1,1)	(4,4)	(2,2)	(3,3)	(1,1)	(4,4)	(2,2)

A partir de la información de los nodos es deducido automáticamente el número de píxeles de la serie, en este ejemplo son cuatro píxeles (10-110-1110–1110). Obsérvese que todos los píxeles intervienen al menos una vez en la secuencia de pares (*propiedad del árbol binario completo*).

El siguiente paso consiste en sustituir los índices de los pares por los valores de los píxeles. Este procedimiento requiere utilizar los índices de la serie S_I recuperada previamente. Los valores de los nodos son índices apuntadores a la serie que extraen el valor original del píxel en la posición exacta del bloque.

2.1) Recuperación de los indices de los nodos.

El decodificador realiza una recuperación directa de los índices de las duplas procesando los bits del fichero comprimido uno a uno. El algoritmo posiciona el puntero al inicio de los datos y procede a la lectura de bits hasta encontrar un cero. El número de bits recorridos corresponde al primer índice p_h de la primera dupla. En la implementación se usa un contador de bits. El contador es reiniciado a uno y con el puntero posicionado en el bit cero el algoritmo recorre de nuevo los datos hasta encontrar un uno. El número de bits recorridos es registrado en un campo de memoria para recuperar su índice usando la operación inversa del codificador, expresión 5-14, (Larrauri y Kahoraho, 2013) se obtiene:

$$p_l = |b(p_h) - b(p_l)| + 1 = |3 - 1| + 1 = 3 \tag{4.14}$$

El resultado de la recuperación de los índices de la primera dupla es la siguiente:

$$\left[\binom{p_h}{p_l} => \binom{3}{3}\right]$$

A continuación el algoritmo procede a la recuperación del resto de las duplas aplicando el mismo procedimiento que en la recuperación de la primera dupla. La recuperación de todas las duplas se muestra en la Fig. 4.37.

$$\left[\binom{3}{3}\right] \Rightarrow \left[\binom{1}{1}\right] \Rightarrow \left[\binom{4}{4}\right] \Rightarrow \left[\binom{2}{2}\right] \Rightarrow \left[\binom{3}{3}\right] \Rightarrow \left[\binom{1}{1}\right] \Rightarrow \left[\binom{4}{4}\right] \Rightarrow \left[\binom{2}{2}\right]$$

Figura 4.37. Recuperación de las duplas de píxeles

La recuperación de los índices de las duplas finaliza cuando todos los nodos hayan sido decodificados.

4.4.1.4 Paso 3. Sustituir los índices por sus correspondientes valores

Una vez recuperado los índices de las duplas en B_i se procede a reemplazarlos pòr sus correspondientes valores en la serie S_i.

Valores de las duplas representadas con índices:

$$\left[\binom{3}{3}\right] \Rightarrow \left[\binom{1}{1}\right] \Rightarrow \left[\binom{4}{4}\right] \Rightarrow \left[\binom{2}{2}\right] \Rightarrow \left[\binom{3}{3}\right] \Rightarrow \left[\binom{1}{1}\right] \Rightarrow \left[\binom{4}{4}\right] \Rightarrow \left[\binom{2}{2}\right]$$

Reemplazar los índices por sus correspondientes valores de píxeles:

$$\left[\binom{162}{162}\right] \Rightarrow \left[\binom{157}{157}\right] \Rightarrow \left[\binom{163}{163}\right] \Rightarrow \left[\binom{161}{161}\right] \Rightarrow \left[\binom{157}{157}\right] \Rightarrow \left[\binom{163}{163}\right] \Rightarrow \left[\binom{161}{161}\right]$$

Reordenar las duplas recuperando el bloque original:

$$\begin{bmatrix} \begin{pmatrix}162\\162\end{pmatrix} & \begin{pmatrix}157\\157\end{pmatrix} & \begin{pmatrix}163\\163\end{pmatrix} & \begin{pmatrix}161\\161\end{pmatrix} \\ \begin{pmatrix}162\\162\end{pmatrix} & \begin{pmatrix}157\\157\end{pmatrix} & \begin{pmatrix}163\\163\end{pmatrix} & \begin{pmatrix}161\\161\end{pmatrix} \end{bmatrix}$$

Al finalizar el proceso de decodificación, el bloque original se ha recuperado recuperada bit a bit y sin pérdida de datos. El Algoritmo INA inverso procede a la recuperación de la imagen original una vez procesados todos los bits.

4.4.1.5 Ejemplo práctico

Los dos ejemplos propuestos en el proceso de compresión que corresponden a los píxeles iniciales de la imagen Lena y son descomprimidos aplicando los pasos descritos en el algoritmo inverso de descompresión.

Ejemplo 1. Decodificación de la segmentación en bloques fijos

La salida binaria del bloque está formada por una sucesión de 11 bits pertenecientes a la serie S_i y una sucesión de 16 bits correspondiente a la codificación del bloque mediante la estructura de árbol binario.

Salida binaria del Bloque Completo = "**10100001**0001111110011111100"

Aplicando el Algoritmo INA inverso para recuperar la serie S_1. comienza recuperando el primer píxel almacenado en el primer byte. Los ocho primeros bits corresponden a la codificación binaria del primer valor de la serie:

10100001 -> p_1= *161*

Salida binaria del Bloque Completo = "*10100001***000**1111110011111100"

Los dos siguientes bits corresponden a los bits de estado que suministran información sobre la recuperación del último píxel de la serie. El código "00-0" implica que existe al menos otro píxel con el siguiente valor al actual, p_2=*162*.

Tabla 4-20. Recuperación de los píxeles de la serie

PÍXEL	PÍXELES	NUMBITS	BIT1	BIT2	VALOR	CÓDIGO	PÍXEL RECUPERADO
p_1	[161,]	8 bits				10100001	161
p_n	[161,162]	3	0	0	0	000	162

El siguiente paso consiste en recuperar los datos del bloque B_1 usando la información de la serie previamente recuperada. La codificación de cada píxel

ocupa un bit, el bit igual a cero representará al píxel "161" y el bit igual a uno representará al píxel con valor "162".

Salida binaria B_1 = {1111110011111100}

En el ejemplo propuesto el tipo de segmentación es fijo con un tamaño de 4x4 píxeles. Por tanto, la decodificación del bloque en este caso consiste en expandir directamente cada bit por el valor apuntado por su índice en el *array* de la serie.

Tabla 4-21. Recuperación de los índices de los píxeles de la serie

ÍNDICE	0	1
Píxeles de la serie S_1	161	162

Resultado de la decodificación:

Tabla 4-22. Recuperación de todos los píxeles del bloque

ÍNDICE	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
B1	162	162	162	162	162	162	161	161	162	162	162	162	162	162	161	161

El bloque reconstruido a partir de los índices y sus valores es exactamente igual al original:

$$B_1 = \begin{bmatrix} \begin{pmatrix}162\\162\end{pmatrix} & \begin{pmatrix}162\\162\end{pmatrix} & \begin{pmatrix}162\\162\end{pmatrix} & \begin{pmatrix}161\\161\end{pmatrix} \\ \begin{pmatrix}162\\162\end{pmatrix} & \begin{pmatrix}162\\162\end{pmatrix} & \begin{pmatrix}162\\162\end{pmatrix} & \begin{pmatrix}161\\161\end{pmatrix} \end{bmatrix}$$

El algoritmo inverso reconstruye secuencialmente todos los bloques codificados hasta procesar la imagen completa.

Ejemplo 2. Decodificación de la segmentación en bloques variables

La salida binaria del bloque estaba formada por una sucesión de 23 bits pertenecientes a la serie S_i y una sucesión de 54 bits correspondiente a la codificación del bloque mediante la estructura de árbol binario.

Código de salida de la serie S_1 = "10011011010000001000000"

Píxel_0

Los ocho primeros bits representan el valor del primer píxel de la serie "155",

$10011011_2 = 155_{10}$ -> ***índice = 0 -> p_1= 155***

El siguiente valor a recuperar corresponderá al último píxel de la serie. Los bits de estado "10" indican que tiene diferente paridad y que puede ser representado con

dos píxeles. El puntero es desplazado ocho posiciones apuntando a los dos siguientes bits "10" que representan el valor del valor residual *Vr=4* aunque marca valor igual a 2, ya que no puede ser los dos siguiente "00" ni siguiente. El valor 4 indica que realmente que corresponde a 2.

Código de salida de la serie S_1 = "*10011011*1010000010000000"

$$V_r = \frac{(p_n - p_0 - 2)}{2} \quad \Longrightarrow \quad p_n = 2 \cdot V_r + p_0 + 2 = 165$$

Para recuperar el valor exacto es necesario añadir uno al valor porque la paridad es diferente.

índice = n -> p_n = 166

El resto de los píxeles son recuperados teniendo en cuenta los bits de estado y sus correspondientes valores. El puntero avanza cuatro posiciones y apunta al siguiente dato a extraer que corresponde al segundo píxel. Los bits de estado "00" indican que el píxel a recuperar es igual al siguiente píxel.

Código de salida de la serie S_1 = "*1001101110100*0010000000"

Píxel_1índice = 1 -> p_1= 156

El puntero continúa avanzando dos posiciones y apunta a los dos siguientes bits de estado que presentan el mismo predictor "00" que el anterior.

Código de salida de la serie S_1 = "*100110111010000*10000000"

índice = 2 -> p_2= 157

El puntero avanza dos posiciones en la salida binaria y apunta a los bits de estado "10" que indica que el píxel a recuperar es igual al siguiente más dos.

Código de salida de la serie S_1 = "*1001101110100000*0000000"

índice = 3 -> p3= 160

Los siguientes valores a recuperar son consecutivos. Sin embargo. El penúltimo valor no es consecutivo con el último codificado en la segunda posición. Por tanto, es necesario decodificar todos los valores para establecer qué valor será el último de la serie.

Código de salida de la serie S_1 = "*10011011010000010*00 00 00"

índice = 4 -> p_4= 161 ; índice = 5 -> p_5= 162 ; índice = 6 -> p_6= 163;

La recuperación completa de la serie S_1 = [**155, 156,157,160,161,162,163,166**}

ÍNDICE	1	2	3	4	5	6	7	8
Píxeles de la serie S_1	155	156	157	160	161	162	163	166

Una vez obtenidos los valores de la serie S_1 se procede a decodificar el bloque B_1. La codificación de cada nodo es una sucesión binaria que comienza por un bit con valor uno y finaliza en cero. Inicialmente, el puntero se posiciona en el primer bit de la codificación binaria del bloque.

Nodo 1. Recuperación del primer nodo

El puntero recorre la sucesión de unos, la cantidad de unos corresponde al índice del píxel en la serie S_i extrayendo el valor contenido en el *array* con el índice apuntado.

B_1 = {1111110 111100 111000 1000000 11111100 110000 10000000 1110 100}

El píxel_0 y primer elemento del nodo es el valor apuntado en el array de la serie S_i con el índice "6". El valor almacenado con este índice corresponde al "162". El siguiente píxel del nodo es definido por el resultado de la expresión. El offset obtenido es cero, y por tanto el puntero apunta al mismo índice y valor, por tanto, el valor del píxel_1 es "162" y los dos primeros píxeles recuperados son 162,162.

Nodo 2. Recuperación del segundo nodo.

El puntero es desplazado y posicionado en el inicio de la codificación del siguiente nodo. Este nodo contiene un flag de par con el primer índice mayor que el segundo. Entonces, la longitud de la cadena de unos más la cadena de ceros indica el índice del primer píxel, en este caso, el valor del índice es seis. La diferencia entre el número de unos y ceros más uno indica el índice del siguiente valor que corresponde al índice cinco.

B_1 = {*1111110* 111100 111000 1000000 11111100 110000 10000000 1110 100}

Los valores apuntados por estos índices en la el *array* de la serie corresponden a los píxeles 162, 161.

El procedimiento para decodificar el resto de los nodos es el mismo que el aplicado en los dos primeros nodos atendiendo al valor del flag que indica el orden de los píxeles en el nodo. Al finalizar el procesamiento del bloque el puntero apunta al inicio de los datos del siguiente bloque variable. Los datos recuperados han sido almacenados secuencialmente. Obviamente la segmentación variable expande directamente los valores de los píxeles en la fila actual de la imagen recuperada y no

requiere dar formato de bloque como en la segmentación fija. Los píxeles recuperados son expandidos como sigue:

$$B_1 = \{162,162,162,161,162,157,163,161,166,\ 162,162,160,155,163,160,155,157,156\}$$

La segmentación variable es más apropiada utilizarla para aplicaciones de visión artificial y especialmente en aplicaciones con cámaras lineales que requieran compresión de datos.

4.4.1.6 Flujograma del proceso de descompresión

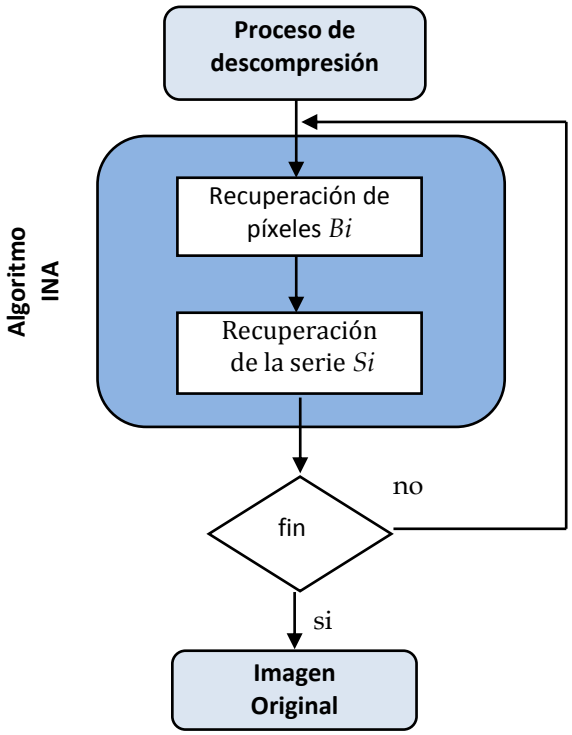


Figura 4.38. Flujograma del proceso de descompresión

4.5 Resultados experimentales

La evaluación del método propuesto se ha realizado sobre la misma muestra de imágenes estándar procedente de diferentes fuentes. En esta evaluación se recogen los resultados experimentales sobre las características de las imágenes clasificadas en sus correspondientes categorías por tipo de imagen fuente y la correlación que existe con los ratios de compresión obtenidos para cada una de las imágenes.

Una vez analizados los resultados de de la segmentación fija en bloques de tamaño 2x2 píxeles, 2x4 píxeles y 4x4 píxeles, 8x 8 píxeles, etc., la segmentación que mejores ratios de compresión obtiene en la mayoría de los casos es la segmentación

en bloques fijos de 2x4 píxeles. Los resultados que se aportan en este estudio y test comparativo se refieren a esta clase de segmentación. Por otra parte, la segmentación variable más recomendable es la segmentación en bloques variables con 4 y 8 píxeles diferentes. En el test de resultados se aportan los datos referentes a 8 píxeles por serie.

Tabla 4-23. Número de píxeles por serie-bloque mediante segmentación fija 2x4 píxeles

CLASE IMAGEN FUENTE	NOMBRE IMAGEN	NÚMERO DE BLOQUES FIJOS DE 2X4 PÍXELES							
		1 PÍXEL	2 PÍXELE	3 PÍXELES	4 PÍXELES	5 PÍXELES	6 PÍXELE	7 PÍXELE	8 PÍXELES
Fotográficas	**Airplane**	67	1.045	2.909	5.052	6.207	6.240	5.979	5.269
	Baboon	0	0	5	149	903	3.839	10.449	17.423
	Barbara	0	17	322	1.751	4.160	6.151	8.760	11.697
	Boats	0	12	180	1.071	3.428	7.816	10.994	9.267
	Goldhill	16	35	329	1.377	3.511	7.404	11.097	8.999
	Lena	2	60	598	2.588	5.876	8.345	8.541	6.758
	Peppers	1	40	458	1.698	4.911	9.287	9.983	6.390
	Zelda	0	8	210	1.757	5.868	9.969	9.779	5.177
Aéreas y Satélite	**Aerial**	5	181	975	2.108	3.761	5.773	8.932	11.033
	Airfield	345	164	215	555	2.625	7.482	11.94	9.438
	Earth	8	250	471	195	453	910	1.427	1.286
	Meteosat	51	648	1.768	4.704	9.959	16.205	22.42	24.239
	Moon	11.4	8.654	25.501	45.421	67.537	85.773	87.76	58.366
	Moonsurface	1.614	31.154	0	0	0	0	0	0
	SanDiego	25	1.921	7.108	8.274	9.373	14.253	32.876	57.242
	WashingtonIR	436	270	2169	5409	13090	48.493	177.266	385.679
Creadas por Ordenador	**Circles**	7.803	389	0	0	0	0	0	0
	Gray	31.87	884	0	6	0	0	0	0
	Slope	888	2.739	479	1.556	598	931	705	296
	Squares	8.192	0	0	0	0	0	0	0
Médicas	**Elbowx**	1.334	0	0	0	0	0	0	0
	Finger	0	0	0	2	17	265	1.996	5.912
	MRI_Brain1	910	521	711	499	676	1.420	2.052	2.148
	MRI_Brain2	537	6.456	1463	664	862	1.518	2.057	1.755
	MRI_Brain3	9	183	559	707	808	1202	2.062	2.662
	Shoulder	3.553	974	1.237	1.157	1.897	3.361	4.369	3.070
Textos y Gráficos escaneados	**Mercados1**	2853	50.740	15.168	11.688	7.441	5.141	4.155	2.695
	Mercados2	1245	64.052	14.768	12.401	6.550	6.624	6.287	2.668
	Mercados3	2761	19.712	13.196	9.804	6.147	6.078	6.265	5.501
	Mercados4	1949	15.013	7.158	3.458	2.082	1.744	1.521	884

En la Tabla 4.23 se reflejan los resultados que aportan las características de cada imagen procesada mediante segmentación fija de 2x4 píxeles (8 píxeles/bloque). Las columnas de la tabla muestran para cada bloque de 2x4 píxeles, si todos ellos son distintos (última columna), si los 8 son iguales (primera columna) o si el número de píxeles distintos está entre 1 y 8. El número total de bloques es fijo en todas las imágenes, 32.768 bloques/imagen.

La Tabla 4.23 refleja la distribución de los píxeles por bloque. Estos resultados enmarcan y definen cada una de las categorías de imagen fuente. La imágenes fotográficas contienen el mayor número de píxeles diferentes con 6, 7, y 8 píxeles. Esto indica que los cambios de intensidad entre píxeles adyacentes son continuos. Y es aquí donde el algoritmo propuesto toma ventaja sobre el resto. Sin embargo, en las imágenes generadas por ordenador la distribución de los píxeles muestra que los bloques con menor número de píxeles distintos por serie son los más repetidos. A continuación son analizados los resultados para cada categoría de imagen fuente.

4.5.1.1 Imágenes fotográficas

La distribución de los datos muestra que los continuos cambios de píxeles es la característica más frecuente en esta clase de imagen fuente. Basándose en esta característica, es fácilmente deducible por qué los métodos convencionales no ofrecen buenos resultados en términos de ratios de compresión e incluso los métodos basados en modelos de sustitución obtiene ficheros comprimidos de mayor tamaño que el original.

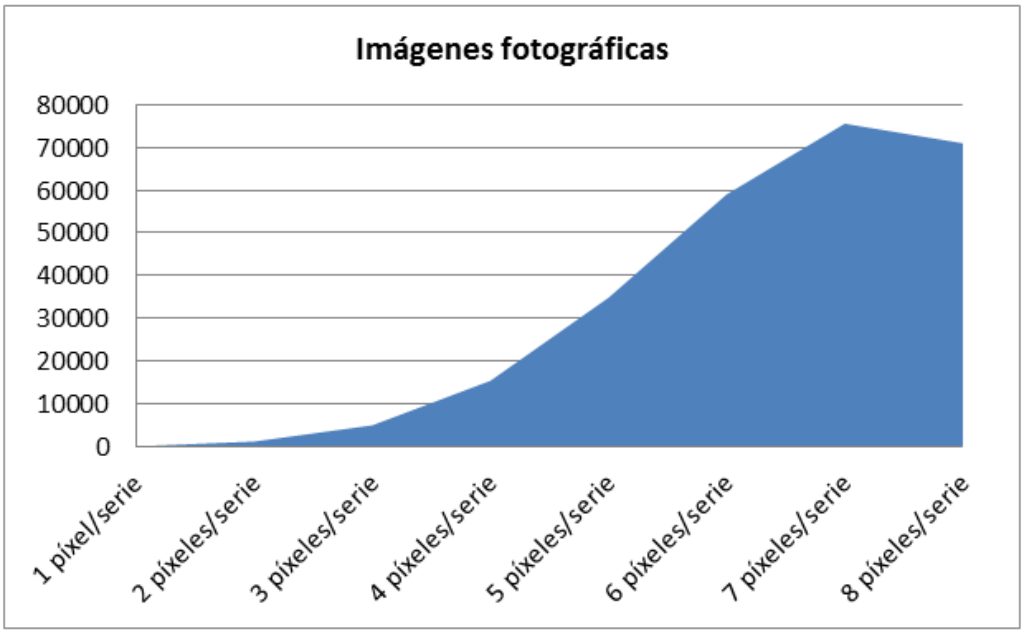


Figura 4.39. Distribución de los píxeles por bloque en imágenes fotográficas

Los datos reflejados en la tabla 4.23 muestran como los bloques con series menores de cuatro píxeles siempre siguen la misma distribución. El bloque con menor número de píxeles repetidos es el que contiene un solo píxel, luego el que contiene dos, tres y cuatro. En estos casos el método propuesto siempre garantiza ratios inferiores a 2:1. En los siguientes casos, la distribución de los píxeles indica que los bloques con 6, 7 y 8 píxeles diferentes son los más frecuentes. El algoritmo propuesto toma ventaja de los bloques de siete y ocho píxeles reduciendo considerablemente la longitud de los códigos de salida aplicando los casos de codificación especial.

4.5.1.2 Imágenes aéreas y de satélite

La distribución de los píxeles es similar a la de las imágenes fotográficas. Esto significa que el algoritmo funcionará efectivamente para este tipo de imágenes.

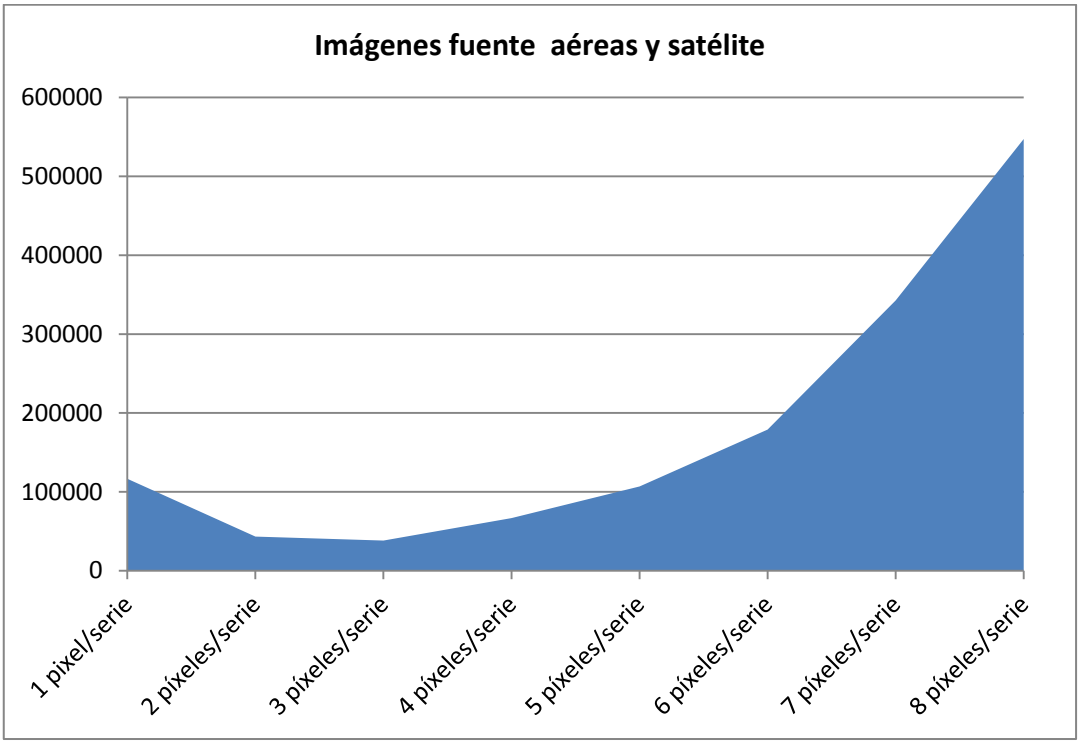


Figura 4.40. Distribución de los píxeles por bloque en imágenes aéreas y de satélite

4.5.1.3 Imágenes creadas por ordenador

La distribución de los píxeles por bloque denota que los bloques con menor número de píxeles son los más repetidos como se muestra en la Fig. 4.41.

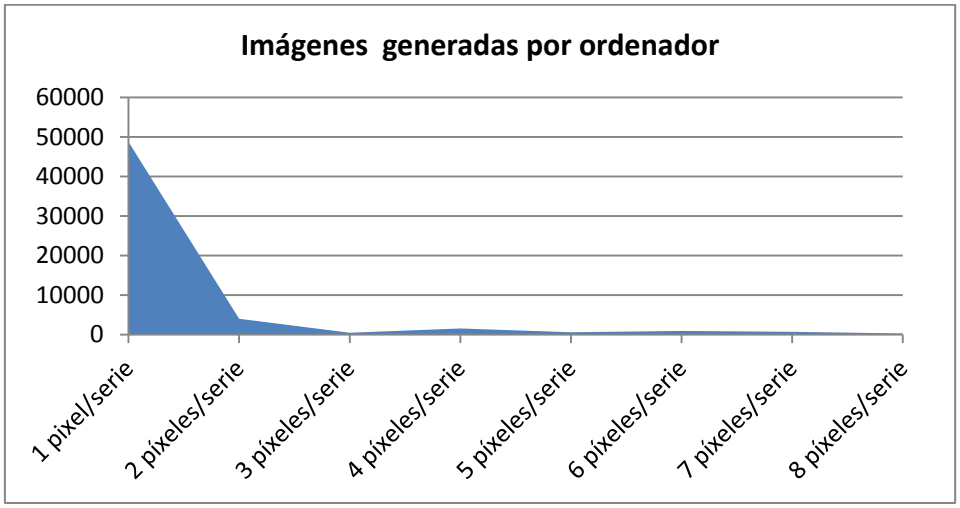


Figura 4.41. Distribución de los píxeles por bloque en imágenes creadas por ordenador

Este tipo de imágenes fuente es ideal para ser codificadas con métodos basados en modelos de sustitución y probabilísticos. El Algoritmo INA aprovecha la característica de pequeños cambios entre píxeles vecinos para aplicar la técnica *Run Length.* La versión del algoritmo mediante segmentación variable es más recomendable que la segmentación fija porque aplica la codificación de los píxeles repetidos por medio de la carrera de avance.

4.5.1.4 Imágenes médicas

La distribución de los píxeles en las imágenes fuente médicas no siguen un patrón definido como en las imágenes creadas por ordenador. En este caso los bloques más significativos son los bloques con las serie más pequeñas y las más grandes. La Fig. 4.42 muestra los resultados obtenidos.

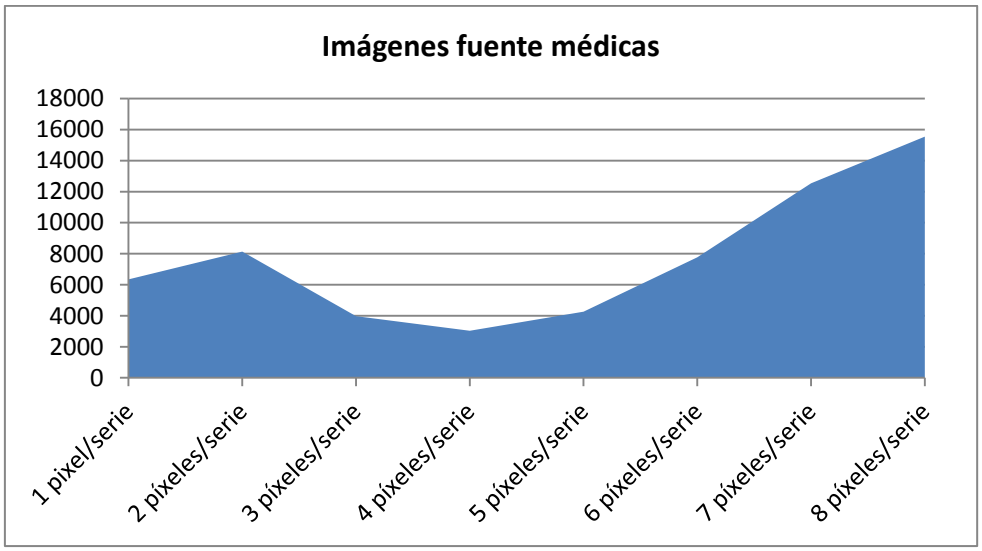


Figura 4.42. Distribución de los píxeles por bloque en imágenes creadas por ordenador

Esta distribución es muy favorable para el funcionamiento del algoritmo porque se sitúa en la codificación de los bloques que garantizan mayor reducción como son: lo bloques con series de píxeles más pequeñas y los bloques con series más grandes.

4.5.1.5 Imágenes de texto y gráficos combinados

La distribución de los píxeles en las imágenes que contiene información textual y gráfica es similar a los resultados de las imágenes creadas por ordenador. Estos resultados están influenciados por la característica que presentan estas imágenes al representar la líneas con caracteres en blanco producidas por el texto. Entre línea y línea existen espacios en blanco repetitivos. La Fig. 4.43 muestra los resultados obtenidos.

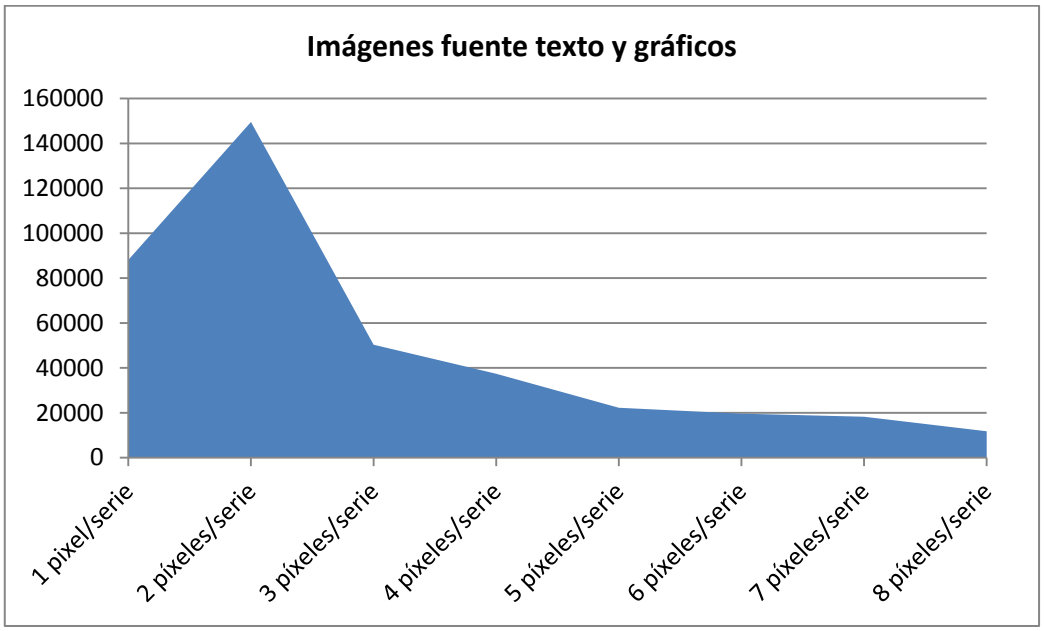


Figura 4.43. Distribución de los píxeles por bloque en imágenes creadas por ordenador

En este tipo de imágenes el algoritmo propuesto obtiene un ratio de compresión de 2:1 como mínimo.

Del análisis de los resultados de las características de la segmentación en bloques de las imágenes se concluye que cada método se comporta de diferente forma según la distribución de los píxeles. A continuación se muestran los resultados obtenidos por el método propuesto basado en la aplicación del algoritmo INA.

4.5.1.6 Resultados de compresión de imágenes para el Algoritmo INA

La Tabla 4.24 muestra estos resultados de los ratios de compresión del Algoritmo INA implementado mediante la segmentación fija en bloques de 2x4 píxeles y los compara con los correspondientes a JPEG-LS.

Tabla 4-24. INA. Ratios de compresión con segmentación fija (2x4 píxeles)

IMAGEN FUENTE	NOMBRE IMAGEN	JPEG-LS	ALGORITMO INA (2X4PÍXELES)
		RATIO DE COMPRESIÓN BITS/PÍXEL	RATIO DE COMPRESIÓN BITS/PÍXEL
Fotográficaas	Airplane	2,115	2,14
	Baboon	1,325	1,48
	Barbara	1,645	1,70
	Boats	1,668	1,79
	Cameraman	1,855	2,15
	Goldhill	1,698	1,80
	Lena	1,885	1,95
	Man	2,709	2,79
	Peppers	1,782	1,94
	Zelda	1,997	2,01
	Media ...	1,868	1,975
Aéreas y Satélite	Aerial	1,621	1,72
	Airfield	1,437	1,63
	Earth	1,630	1,86
	Meteosat	1,640	1,76
	Moon	2,916	2,37
	Moonsurface	1,574	1,20
	SanDiego	1,418	1,61
	WashingtonIR	1,229	1,38
	Media ...	1,683	1,692
Creadas por Ordenador	Circles	51,829	4,19
	Gray	132,077	4,15
	Slope	5,088	3,22
	Squares	101,335	4,19
	Media ...	72,58	3,937
Médicas	Elbowx	3,810	3,15
	Finger	1,170	1,35
	MRI_Brain1	2,097	1,66
	MRI_Brain2	3,027	2,82
	MRI_Brain3	1,826	1,86
	Shoulder	2,360	2,26
	Media ...	2,381	2,183
Textos y Gráficos escaneados	Mercados1	3,255	3,09
	Mercados2	2,825	3,00
	Mercados3	3,202	2,80
	Mercados4	3,529	3,15
	Media ...	3,202	3,01

Además de los ratios de compresión es importante medir el consumo de recursos del Algoritmo INA en cada una de las imágenes. En este caso es muy importante distinguir entre los resultados en un procesador con un solo núcleo (Tabla 4.25), con dos y con cuatro (Tabla 4.26), ya que los tiempos mejoran sustancialmente.

Tabla 4-25. INA. Tiempos de ejecución del ciclo de compresión y descompresión para 1 hilo

IMAGEN FUENTE	NOMBRE IMAGEN	CICLO DE COMPRESIÓN		CICLO DE DESCOMPRESIÓN	
		CICLOS CPU	TIEMPO (ms)	CICLOS CPU	TIEMPO (ms)
Fotográficaas	Airplane	344.739	156,09	314.856	142,56
	Baboon	344.426	155,95	315.253	142,74
	Barbara	344.547	156,00	316.468	143,29
	Boats	343.860	155,69	315.850	143,01
	Cameraman	118.049	53,45	113.698	51,48
	Goldhill	344.116	155,71	316.093	143,26
	Lena	343.816	155,81	318.147	144,05
	Man	117141	53,22	113455	51,37
	Peppers	344.739	155,67	316.689	143,39
	Zelda	344.426	155,64	316.291	143,21
	Media	**298.986**	**135,32**	**275.680**	**124,84**
Aéreas y Satélite	Aerial	329.933	149,39	303.194	137,28
	Airfield	329.293	149,10	302.510	136,97
	Earth	272.580	123,42	251.999	114,10
	Meteosat	427.049	193,36	389.484	176,35
	Moon	5.232.417	2.369,13	7.483.589	3.388,41
	Moonsurface	327.971	148,50	301.825	136,66
	SanDiego	1.321.298	598,26	1.291.602	584,81
	WashingtonIR	6.378.454	2.888,03	6.216.193	2.814,56
	Media	**1.827.374**	**827,40**	**2.067.550**	**936,14**
Creadas por Ordenador	Circles	118.049	53,45	113.698	51,48
	Gray	364.902	165,22	356.421	161,38
	Slope	95.057	43,04	92.142	41,72
	Squares	112.726	51,04	109.678	49,66
	Media	**172.684**	**78,19**	**167.985**	**76,06**
Médicas	Elbowx	13.506	6,12	13.113	5,94
	Finger	82.654	37,42	80.247	36,33
	MRI_Brain1	90.028	40,76	87.406	39,58
	MRI_Brain2	153.887	69,68	149.405	67,65
	MRI_Brain3	82.540	37,37	80.136	36,28
	Shoulder	197.606	89,47	191.850	86,87
	Media	**103.370**	**46,80**	**100.360**	**45,44**
Textos y Gráficos escaneados	Mercados1	1.264.914	572,73	1.228.072	556,04
	Mercados2	1.265.303	572,90	1.228.450	556,22
	Mercados3	951.837	430,97	924.113	418,42
	Mercados4	518.136	234,60	503.044	227,77
	Media	**1.000.048**	**452,80**	**970.920**	**439,61**

Una de las principales ventajas de este método reside en la aportación de un nuevo enfoque orientado a la compresión de imágenes en paralelo. La mayoría de los métodos convencionales y especialmente los estándares de compresión basan su potencia de compresión en la predicción de los píxeles. Estos métodos no están diseñados para la compresión en paralelo y no ofrecen posibilidades de adaptación. Por otra parte, en la actualidad, los computadores convencionales ofrecen versiones con múltiples núcleos o procesadores y facilitan la ejecución de programas en paralelo mediante hilos de programación.

En este contexto, el método INA encaja idealmente en la *compresión de imágenes en paralelo*. El proceso clave para la compresión en paralelo reside en la segmentación de la imagen en bloques fijos. El usuario dispone de una opción de configuración del método INA para seleccionar los núcleos (procesadores) y el número de hilos de programación en función de las características de su computador. En base a esta configuración, el método segmenta la imagen en bloques para el procesamiento independiente en paralelo. Por último es importante destacar que el proceso de descompresión es independiente del proceso de compresión con relación al número de procesadores e hilos utilizado porque los datos se guardan en la misma secuencia que en la versión estática secuencial.

Obviamente, el objetivo fundamental de la compresión en paralelo radica en el deseo de reducir los tiempos de ejecución. En muchas aplicaciones estos tiempos son críticos.

La Tabla 4.26 contiene los resultados de compresión obtenidos en la ejecución en paralelo del método propuesto aplicando las siguientes versiones: una primera versión con un procesador y dos hilos de programación, una segunda con dos procesadores y dos hilos por núcleo y por último usando cuatro núcleos y un hilo por núcleo.

La simple observación de las Tablas 4.25 y 4.26 indica que el paso de 1 hilo a 2 hilos no supone una reducción drástica de los tiempos de ejecución. Sin embargo el paso de 2 hilos y 1 núcleo a 4 hilos con dos núcleos, y de estos 2 núcleos a 4 hilos y 4 núcleos, supone una reducción del 50% en cada paso. Es decir, el uso del potencial de los actuales procesadores reduce sensiblemente el tiempo de ejecución.

Tabla 4-26. INA. Tiempos de ejecución en las versiones de compresión en paralelo

IMAGEN FUENTE	NOMBRE IMAGEN	MÉTODO INA. COMPRESIÓN DE IMÁGENES EN PARALELO		
		1 µP Y 2 HILOS (ms)	2 µP Y 4 HILOS (ms)	4 µP (ms)
Fotográficaas	Airplane	119,02	61,54	34,78
	Baboon	118,96	62,19	35,01
	Barbara	120,01	62,45	35,22
	Boats	119,16	61,98	34,89
	Cameraman	30,08	17,09	12,45
	Goldhill	119,66	62,89	35,19
	Lena	119,07	62,78	35,92
	Man	29,87	16,66	12,13
	Peppers	119,09	62,54	35,29
	Zelda	119,13	62,41	35,36
	Media	**101,41**	**53,25**	**30,62**
Aéreas y Satélite	Aerial	119,03	62,58	35,66
	Airfield	119,21	62,66	35,87
	Earth	19,04	11,32	9,64
	Meteosat	290,88	148,96	74,48
	Moon	1.791,93	901,33	454,62
	Moonsurface	119,14	62,89	35,55
	SanDiego	476,21	242,44	125,14
	WashingtonIR	2.299,28	1.154,02	572,93
	Media	**654,34**	**330,78**	**167,99**
Creadas por Ordenador	Circles	29,89	16,85	12,69
	Gray	119,14	62,67	35,85
	Slope	29,83	16,92	12,46
	Squares	29,07	16,59	12,67
	Media	**51,98**	**28,26**	**18,42**
Médicas	Elbowx	5,10	3,98	5,98
	Finger	29,96	16,67	12,34
	MRI_Brain1	32,79	18,99	13,59
	MRI_Brain2	55,81	29,93	18,5
	MRI_Brain3	29,65	16,57	12,68
	Shoulder	71,89	37,81	22,41
	Media	**37,53**	**20,66**	**16,80**
Textos y Gráficos escaneados	Mercados1	4.546,71	2.278,31	1145,83
	Mercados2	457,85	232,84	120,35
	Mercados3	342,99	176,50	93,01
	Mercados4	386,72	196,38	102,59
	Media	**1.433,57**	**721,01**	**365,45**

Los resultados de las Tablas 4.24, 4.25 y 4.26 pueden integrase en la tabla resumen del estado del arte, Tabla 3.59, para dar lugar a la nueva Tabla 4.27. En ella se muestran los valores medios del ratio de compresión para los 16 métodos implementados en el estado del arte más el INA y el tiempo consumido en cada proceso de compresión, como valor medio de cada tipo de imágenes.

Tabla 4-27. Resumen comparativo de los resultados de los métdos implementados

	FOTOGRÁFICAS	AÉREAS Y SATÉLITE	CREADAS POR ORDENADOR	MÉDICAS	TEXTOS Y GRÁFICOS ESCANEADOS
Bitmap 1 byte	1,006 455,38 ms	1,027 1.859,25 ms	3,236 123,30 ms	1,129 537,61 ms	1,801 788,15 ms
Bitmap fila	0,921 518,19 ms	0,949 2.003,92 ms	1,701 235,11 ms	1,041 316,58 ms	1,711 1.404,77 ms
Bitmap bloque	0,904 783,986 ms	0,942 1.733,19 ms	1,407 255,605 ms	1,019 251,75 ms	1,253 815,21 ms
RLE clásico	0,916 400,44 ms	0,990 2.252,35 ms	49,714 57,96 ms	1,083 178,61 ms	1,826 834,98 ms
RLE 4 bits	0,950 473,61 ms	1,081 2.785,25 ms	76,387 58,94 ms	1,245 205,62 ms	2,142 893,73 ms
Huffman estático	1,092 1.081,27 ms	1,134 4.963,28 ms	3,258 543,41 ms	1,268 591,04 ms	1,657 2.741,13 ms
Huffman adaptativo	1,082 923,04 ms	1,131 4.628,36 ms	2,976 440,42 ms	1,250 509,34 ms	1,605 1.632,82 ms
Codificador Aritmético Estático	1,100 189,50 ms	1,441 1.024,61 ms	3,748 152,04 ms	1,282 105,53 ms	1,621 504,06 ms
Codificador Aritmético Adaptativo	1,154 327,86 ms	1,494 2.172,16 ms	4,558 153,54 ms	1,325 220,53 ms	2,120 593,84 ms
LZ77 Estático	1,040 1.024,69 ms	1,474 2.808,12 ms	6,403 268,40 ms	1,330 263,35 ms	3,191 1.519,11 ms
LZW Adaptativo	0,930 91,20 ms	1,144 448,04 ms	13,650 44,98 ms	1,212 57,38 ms	1,604 248,85 ms
JPEG lossless + Huffman	1,631 80,91 ms	1,555 373,50 ms	34,826 40,28 ms	2,166 56,11 ms	1,978 183,41 ms
JPEG lossless + Aritmético	1,699 80,39 ms	1,609 248,95 ms	83,022 54,99 ms	2,268 62,69 ms	1,938 92,30 ms
Transformada Haar	1,243 527,30 ms	1,188 3.434,51 ms	47,522 276,99 ms	1,532 727,10 ms	2,441 3.071,28 ms
Lifting Scheme	1,621 301,78 ms	1,506 1.251,49 ms	77,360 153,69 ms	2,106 158,95 ms	3,106 660,13 ms

Tabla 4-27. Resumen comparativo de los resultados de los métdos implementados (cont)

	FOTOGRÁFICAS	AÉREAS Y SATÉLITE	CREADAS POR ORDENADOR	MÉDICAS	TEXTOS Y GRÁFICOS ESCANEADOS
JPEG-LS	1,868 66,25 ms	1,683 163,03 ms	72,582 59,17 ms	2,381 63,78 ms	3,203 91,37 ms
Algoritmo INA	1,975 1 hilo 135,32 ms 4 hilos 30,62 ms	1,692 1 hilo 827,40 ms 4 hilos 167,99 ms	3,937 1 hilo 78,19 ms 4 hilos 18,42 ms	2,183 1 hilo 46,80 ms 4 hilos 16,80 ms	3,01 1 hilo 452,80 ms 4 hilos 365,45 ms

4.6 Conclusiones del Algoritmo INA

El Algoritmo INA propuesto en la tesis comprime las imágenes en dos fases: preprocesado de datos para obtener el árbol binario y codificación del árbol binario (Larrauri, 2012). En ambas fases INA propone un método innovador que abre nuevos caminos para el desarrollo de nuevos algoritmos de compresión de imágenes sin pérdida de datos. Dichas estrategias han sido descritas e implementadas en detalle en este capítulo.

Metodológicamente destaca que INA tenga como elemento básico el aprovechamiento de las posibilidades computacionales de los procesadores actuales, sobre todo el hecho de que cuenten con varios núcleos.

El Algoritmo INA esta centrado en la compresión de imágenes fotográficas con continuos cambios de tono, es decir, aquellas que son más difíciles de comprimir ya que son pocos los valores que se repiten entre píxeles cercanos.

Por tanto, INA se acerca al problema de compresión de imágenes sin pérdida de datos desde una perspectiva novedosa enfocada primordialmente a imágenes fotográficas. Más allá de estas aportaciones cualitativas es necesario observar el rendimiento de INA principalmente en términos de ratio de compresión, y secundariamente en términos de consumo de recursos. Si bien dicha comparación ha de hacerse para todos los métodos implementados en el estado del arte, no es menos cierto que el método más perfecto en la actualidad es el JPEG-LS, que de hecho es el estándar desarrollado por la industria para la compresión de imágenes.

La Tabla 4.27 resume los ratios de compresión medios para cada tipo de imagen para los 17 algoritmos de compresión impelmentados. A la vista de las Tablas 4.24 a 4.33 se pueden establecer varias conclusiones parciales, pero la más importante es que el nuevo Algoritmo INA diseñado e implementado en la tesis tiene un rendimiento similar al del algoritmo JPEG-LS, estándar actual de la

industria. Dicho rendimiento es algo mayor –ratio de compresión de 1,975 frente a 1,868, un 6%- para imágenes fotográficas con continuos cambios de tono, aquellas que presentaban una mayor complejidad a la hora de ser comprimidas. Esta mejora conlleva un aumento en el consumo de recursos, aunque este puede ser reducido al utilizar la ejecución en distintos hilos y núcleos del procesador.

De una forma más detallada en cuanto a los ratios de compresión:

Imágenes fotográficas.

INA presenta el mayor ratio de compresión para este tipo de imágenes, 1,975. El ratio es superior en un 6% al del algoritmo JPEG-LS, pero para JPEG *lossless* la mejora es del 20%, y para el resto de algoritmos la mejora en el ratio de compresión ronda el 80%. Hay que tener en cuenta que varios métodos de compresión obtenían un fichero de mayor tamaño que el original.

Un análisis detallado de la compresión de imágenes fotográficas muestra que INA en ningún caso obtiene un ratio de compresión menor que JPEG-LS, y que en el mejor caso su ratio de compresión es un 11% superior al de JPEG-LS (1,48 frente a 1,325) para la imagen Baboon, y en el peor caso la mejora es inferior al 1% (2,01 frente a 1,997) para la imagen Zelda.

Imágenes áereas y de satélite

En este caso el análisis es similar al anterior, el Algoritmo INA mejora el rendimiento del JPEG-LS y es claramente superior al resto de ellos en general. Para este tipo de imágenes, el rendimiento de los métodos JPEG *lossless* es similar al de JPEG-LS e INA, los mismo que los métodos basados en codificadores.

Imágenes creadas por ordenador.

Para este tipo de imágenes el rendimiento del Algoritmo INA baja mucho respecto del resto de métodos, y es claramente inferior. Su ratio de compresión es varias veces inferior al de JPEG-LS. La razón para esta bajada estriba en que como ya se ha analizado en este capítulo, este tipo de imágenes presenta muchos píxeles que se repiten, es decir, no hay cambios continuos como en las imágenes fotográficas. Es decir, INA no es adecuado para comprimir imágenes creadas por ordenador, que sin embargo son bien comprimidas por otros métodos menos evolucionados.

Imágenes médicas.

El Algoritmo INA tiene un ratio de compresión de imágenes similar al obtenido por JPEG-LS, un 10% inferior a este. El valor medio de su ratio de compresión, 2,183, es mayor que el del resto de métodos de compresión, excepto del ratio de compresión de uno de los métodos JPEG *lossless*, que es un 4% superior.

Textos y gráficos escaneados.

De nuevo el Algoritmo INA tiene un rendimiento algo inferior al JPEG-LS, un 6%, pero claramente superior al resto de los métodos, excepto para el método LZ77 que es también un 6% mejor.

Desde el punto de vista del consumo de recursos debe remarcarse que el tiempo de ejecución del Algoritmo INA es más o menos fijo para cada tipo de imagen: los tiempos de ejecución de la serie y del árbol binario son fijos por tamaño del bloque. Como en los otros métodos, hay ciertas variaciones extremas y súbitas que deben ser asociadas al comportamiento a veces variable de los ordenadores y a la propia imagen. Este consumo constante de tiempo es una característica que no tienen lo otros métodos ya que su tiempo de ejecución es muy variable, a excepción del JPEG-LS cuyo tiempo de ejecución es estable, pero en menor medidad que el INA.

Esta característica permite predecir con fiabilidad el tiempo de ejecución del algoritmo lo que puede ser crítico en algunas aplicaciones de tiempo real o con transmisión de datos.

Los tiempos de ejecución anteriores están obtenidos ejecutando el algoritmo en un solo hilo, es decir, secuencialmente. Sin embargo el uso de los cuatro núcleos de un procesador actual reduce considerablemente el tiempo de ejecución. Los tiempos de ejecución de INA en un hilo vienen a ser el doble de los obtenidos para el JPEG-LS, pero comparando JPEG-LS con INA con 4 hilos y 4 núcleos, entonces el tiempo de ejecución es menor para imágenes fotográficas (50%), creadas por ordenador (60%) y médicas (75%), es igual para imágenes aéreas y es mayor para textos y gráficas.

Respecto de los otros métodos analizados en el estado del arte, cabe decir que INA no solo presenta un tiempo de ejecución estable sino que esté es considerablemente menor que los asociados a los otros métodos. Solo los tiempos de ejecución de JPEG *lossless* y de LZ adaptativo son comparables a los de INA, aunque son inferiores.

Resumiendo, una vez analizados los resultados de los 17 algoritmos de compresión de imágenes sin pérdida de datos se observa que el nuevo Algoritmo INA diseñado en la tesis tiene un rendimiento claramente superior a 11 de ellos para cualquier tipo de imagen excepto para imágenes creadas por ordenador (una mejora superior al 25%). Para otros 5 métodos el rendimiento es superior (alrededor del 25%), y para el método JPEG-LS el rendimiento es a veces superior (imágenes fotográficas y áreas) y otras veces es inferior (imágenes médicas y textos). Es resaltable que el algoritmo INA es claramente inferior a la mayoría de los otros métodos para imágenes creadas por ordenador. El tiempo de ejecución de INA

depende del número de hilos y de núcleos utilizados, y en general es mejor que los otros métodos implementados, y claramente comparable al JPEG-LS.

Capítulo

5

Conclusiones y líneas futuras de trabajo

En los capítulos previos se ha abordado de una forma ordenada la problemática de la compresión de imágenes sin pérdida de datos con el fin de diseñar e implementar un nuevo algoritmo que mejore las prestaciones de los ya existentes.

La compresión de datos es hoy en día una herramienta muy útil ya que una parte de los datos que se almacenan y/o transmiten son de esa naturaleza. Cada vez más la información no son signos, números, etc. que conforman un texto, sino píxeles que *dibujan* un gráfico. La compresión de imágenes simplemente busca almacenar la imagen de forma que ocupe menos espacio sin deteriorarla.

Como ya se ha descrito en la tesis, la compresión de imágenes tiene dos grandes enfoques: con pérdida y sin pérdida de datos. En el primer caso no importa que se pierda o modifique alguna información, ya que dicha pérdida no resulta relevante a la hora de procesar, siempre que se consiga reducir el tamaño del fichero comprimido de una forma relevante. En el segundo enfoque, sin pérdida de datos, se prima el hecho de que toda la información sea recuperada, sin admitirse la pérdida de un solo bit, aunque la contrapartida será el tamaño del fichero comprimido. Por último, y como tercer elemento de análisis, se tiene el consumo de recursos, entendido este como el tiempo necesario para comprimir la imagen. La tesis centra su esfuerzo en la estrategia de compresión de imágenes sin pérdida de datos, teniendo como objetivo el diseño de un mejor algoritmo de compresión de

datos en términos de tamaño del fichero comprimido, sin deteriorar en exceso el consumo de recursos. Estos es, asegurar la eficacia y mantener la eficiencia.

La compresión de imágenes ha evolucionado mucho en los últimos 50 años de historia. En este tiempo se han diseñado e implementado distintos algoritmos de compresión, donde la innovación y la creatividad han sido elementos cruciales. En el estado del arte de la tesis se han descrito métodos muy distintos entre sí, donde no pocas veces se han usado para comprimir imágenes elementos propios de otras áreas de conocimiento, como la Transformada de Fourier. En un principio los algoritmos implementaban procesos pensados desde la habilidad humana para "compactar" información, siendo el Bitmap su mejor ejemplo, pero poco a poco los algoritmos fueron acercándose más a la explotación de los recursos de un procesador y de la estructura de datos. En este caso, el uso de árboles binarios, poco próximos al modo de trabajar humano, es un buen ejemplo.

Lo anterior es relevante en la tesis por el hecho de que la mayor parte de los algoritmos (excepto los iniciales) son difíciles de explicar sin la ayuda de flujogramas; y solo toman su verdadera dimensión cuando son expresados como algoritmos. Así pues, en la descripción de los algoritmos se ha intentado mantener su comprensión a la vez que se describía su potencial algorítmico.

En general, los algoritmos de compresión de imágenes constan de dos fases. Los algoritmos más avanzados tienen una primera fase en la que *preparan* los datos (preprocesado): los reordenan, los cuantifican, etc. con el fin de facilitar la segunda fase de codificación. Esta segunda fase es en la que se produce la compresión real, pero en muchos casos su rendimiento depende del primero de ellos. La primera fase suele tener un carácter más innovador y creativo, mientras que la segunda es más técnica.

Además de lo anterior, la compresión de imágenes depende de la propia imagen. Es decir, algunas imágenes son más fáciles de comprimir que otras, o lo que es lo mismo, para algunas imágenes es irrelevante el algoritmo usado ya que el rendimiento siempre será bueno. Mientras que otras imágenes pueden ser consideradas difíciles de comprimir, o sea, el tamaño del ficehro comprimido, su rendimiento, depende mucho del algoritmo usado para dicha compresión. Las imágenes más difícles de comprimir son aquellas en las que los píxeles no se repiten mucho. Esta situación se da en las imágenes fotográficas, llamadas de continuos cambios de tono, ya que en este caso los píxeles van cambiando respecto de los que le rodean, aunque sea poco.

El algoritmo propuesto en la tesis, Algoritmo INA, se sustenta en una aportación para cada una de las fases. Por un lado, ordena los datos según un criterio novedoso que da como resultado un árbol binario que permite una codificación (segunda fase) más compacta por cualquiera de los métodos clásicos.

Este hecho se ve acrecentado por el nuevo algoritmo de codificación diseñado, lo que potencia el INA aún más. Además el Algoritmo INA está centrado específicamente en la compresión de imágenes fotográficas con continuos cambios de tono.

Por último, cabe remarcar que el ámbito de la tesis no tiene un interés meramente investigador. Los algoritmos descritos son usados por la industria informática para comprimir imágenes y dan lugar a estándares que a su vez son registrados como patentes industriales. Ees decir, la tesis no pertenece únicamente al ámbito de la investigación básica, sino que también tiene un claro enfoque aplicativo e industrial.

A continuación se describen los hitos de esta tesis, entendidos estos como la validación de la hipótesis, la consecución de objetivos, la elaboración de las conclusiones y la fijación de los retos pendientes y las nuevas líneas de trabajo.

5.1 Validación de la hipótesis

La hipótesis inicial quedó establecida como sigue:

> Es posible diseñar e implementar un nuevo algoritmo en el área de la compresión de imágenes digitales estáticas de continuos tonos o color en dos dimensiones que mejore los ratios de compresión de los estándares actuales para acercarlos al estado-del-arte. Dicha mejora en los ratios de compresión no debe comprometer su rendimiento en términos de velocidad y consumo de recursos de computación

El desarrollo de la tesis confirma que se ha validado la hipótesis de partida, ya que el nuevo Algoritmo INA de compresión de imágenes sin pérdida de datos ofrece un mejor rendimiento que los algoritmos descritos en el estado del arte, incluyendo los estándares de la industria.

Lo anterior solo tiene validez si el proceso se basa en la consecución de objetivos y en una metodología exhaustiva.

5.2 Consecución de objetivos

La validación de la hipótesis es la consecución de una serie de objetivos según una metodología predeterminada dando lugar a una serie de aportaciones:

O1. Estudio del estado del arte

Tras el análisis de los algoritmos de compresión de imágenes sin pérdida de datos se procedió a su implementación en un entorno fijo (LABwindows CVI), y de esta manera los rendimientos obtenidos son comparables y no dependen de sus diferentes implementaciones. Este objetivo es crítico para la tesis, ya que sobre él se sustenta el análisis de rendimientos de los algoritmos en términos de ratio de compresión y tiempo de ejecución (o número de ciclos).

Este objetivo configura un escenario que puede ser utilizado por otros investigadores como base para sus desarrollos.

O2. Diseño de un banco de imágenes

Como se ha descrito anteriormente, la compresión depende de la eficacia del algoritmo y de la complejidad de la imagen a comprimir. Un objetivo básico de la tesis era definir un banco o conjunto de imágenes que representara la variedad de ellas en la actividad informática. El banco diseñado tiene dos características fundamentales: es diverso y es estándar. Esto último supone que se puede obtener del estado del arte el rendimiento de determinados algoritmos para determinadas imágenes, y así contrastar la bondad de los métodos y sus implementaciones.

Este objetivo y el anterior definen un escenario investigador en el que poder concluir sobre los resultados del nuevo Algoritmo INA implementado.

O3. Definición del estado del arte

Del análisis de resultados del estado del arte resulta evidente que el mejor algoritmo de compresión de imágenes sin pérdida de datos es el denominado JPEG-LS. Este método constituye el estándar actual de compresión de datos, es decir, representa el esfuerzo de la industria por implementar el mejor algoritmo posible.

Por tanto, el nuevo algoritmo a diseñar deberá ser comparado con el JPEG-LS.

O4. Diseño del Algoritmo INA

Partiendo del conocimiento y de la experiencia adquirida en los dos primeros objetivos, el desarrollo del Algoritmo INA es el objetivo final. La descripción del nuevo algoritmo sigue las pautas marcadas para la elaboración del estado del arte, de esta forma la tesis tiene un desarrollo coherente y fácilmente contrastable.

El nuevo Algoritmo INA se basa en un enfoque innovador y obtiene rendimientos superiores a los obtenidos en el primer objetivo. Este resultado es revelador para las imágenes fotográficas con continuos cambios de tono, consideradas estas como las más complejas de comprimir.

Lo anterior es constatado al comparar el rendimiento y consumo de recursos del Algoritmo INA con el JPEG-LS, definido en el tercer objetivo como el algoritmo de referencia para la industria y el comité de estándares de compresión.

La implementación de INA utiliza como recurso el número de núcleos del procesador, lo que abre una nueva línea de trabajo y mejora su consumo de recursos.

5.3 Conclusiones

Tras la consecución de los objetivos es posible establecer una serie de conclusiones generales que afectan al campo de investigación de los métodos de compresión de imágenes sin pérdida de datos:

- Los algoritmos de compresión de imágenes sin pérdida de datos pueden ser ordenados y clasificados según su estrategia de compresión.
- El algoritmo JPEG-LS es el estándar industrial y sus ratios de compresión para imágenes fotográficas con continuos cambios de tono (imágenes fotográficas) van de 1,325 a 2,709, siendo su valor medio para dicho juego de imágenes de 1,868. El consumo de recursos para dicho método tiene un valor medio de 146.303 ciclos de CPU.
- El algoritmo JPEG-LS no es solo el mayor ratio de compresión presenta para cuatro de los cinco tipos de imágenes, incluyendo imágenes fotográficas, sino que también es el que menor consumo de recursos presenta en tres de los cinco tipos de imágenes. JPEG-LS es eficaz y eficiente y se convierte en el referente en el campo de compresión de imágenes sin pérdida de datos.
- El Algoritmo INA descrito en la tesis presenta un ratio de compresión para imágenes fotográficas con continuos cambios de tono que va de 2,01 a 2,89, siendo su valor medio de 1,935, lo que supone una mejora de aproximadamente el 5%. El consumo de recursos es similar al obtenido por JPEG-LS, aunque mayor. Por tanto el Algoritmo INA puede ser considerado comparable al estándar de la industria JPEG-LS.
- Frente a los otros métodos de compresión sin pérdida de datos, el INA se muestra claramente superior.
- El consumo de recursos del Algoritmo INA medido como el tiempo de ejecución para cuatro hilos y cuatro núcleos es bajo, y menor al de el JPEG-LS.

- El tiempo de ejecución del Algoritmo INA es más o menos fijo dentro de cada tipo de imagen, e incluso entre los distintos tipos de imagen. Esta situación hace que INA sea predecible en el tiempo de ejecución, lo que le confiere una característica de la que carecen los otros métodos implementados.
- El Algoritmo INA puede ser implementado en varios hilos de ejecución paralela lo que supone una ventaja clara ya que actualmente todos los procesadores disponen de al menos cuatro núcleos. El ratio de compresión no se ve afectado, como es natural, pero el tiempo de ejecución baja considerablemente.
- El Algoritmo INA es el primer algoritmo de compresión de imágenes que utiliza los diferentes hilos o núcleos de los procesadores actuales para mejorar su rendimiento. Este uso es innovador y favorece la implementación hardware del agoritmo en una plataforma CODEC.
- El Algoritmo INA presenta una nueva estrategia de preprocesado de imágenes basada en la correlación espacial de los píxeles que abre un nuevo campo de investigación dentro de la compresión de imágenes sin pérdida de datos. La comunidad investigadora tiene la oportunidad de mejorar el Algoritmo INA con el fin de obtener mejores ratios de compresión.
- El Algoritmo INA presenta un nuevo algoritmo de codificación de árboles binarios. Dicho algoritmo es parte principal del INA, pero también puede ser usado por otros métodos de compresión que utilicen la codificación como parte del algoritmo.

5.4 Líneas futuras de trabajo

Atendiendo al estado del arte y las conclusiones y resultados obtenidos, surgen nuevas líneas de trabajo:

- El Algoritmo INA puede ser tomado como base para el diseño de un nuevo estándar de compresión de imágenes sin pérdida de datos por parte de la industria. Este proceso puede derivar en la patente del algoritmo.
- El Algoritmo INA está caracterizado por tiempos de ejecución poco variables, es decir, se puede predecir el tiempo de compresión y descompresión. Esta característica puede dar lugar a un campo de trabajo en el mundo de aplicaciones en tiempo real para compresión de imágenes y en el mundo de transmisión de imágenes.
- El Algoritmo INA admite un desarrollo posterior que explote de forma más explícita el número de núcleos de los procesadores actuales. El esfuerzo debe centrarse en la adaptación automática del Algoritmo INA al número de

núcleos del procesador. Este nuevo escenario demanda un nuevo marco de análisis del rendimiento incluyendo el número de núcleos.

- El Algoritmo INA puede ser implementado en hardware dando lugar a una nueva plataforma CODEC. Este enfoque exige de nuevo el análisis del estado del arte y su comparativa con los estándares industriales. Esta línea futura tiene un claro enfoque industrial.
- El uso de dispositivos de procesamiento masivo paralelo, tipo FPGA, es una oportunidad para aumentar la eficacia del algoritmo.
- El Algoritmo INA puede ser refinado tanto en su fase de preprocesado de píxeles como en su fase de codificación. Las innovaciones planteadas son susceptibles de ser analizadas en detalle y mejoradas o sustituidas.
- Los algoritmos de compresión de imágenes son dependientes de las propias imágenes, en este caso se pueden diseñar nuevos algoritmos que se adapten a la estructura de las imágenes que procesan.

APÉNDICE

A

Colección de imágenes digitales estándares usadas en la evaluación de los métodos

La compresión de imágenes sin pérdida de datos utiliza una colección estándar de imágenes de diferente fuente para realizar estudios comparativos de los métodos de compresión. En esta disertación se han añdido otras imágenes estándares para cada tipo de fuente con el objetivo de completar los ratios y tiempos der cada método.

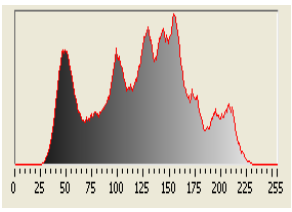
Las direcciones han sido actualizadas a fecha de Julio 2014.

- *Imágenes fotográficas: http://sipi.usc.edu/database/*
- *Imágenes aéreas y por satélite: http://cdb.paradice-insight.us/?class=1*
- *Imágenes creadas por ordenador: http://links.uwaterloo.ca/Repository.html*
- *Imágenes médicas: http://cdb.paradice-insight.us/*
- *Imágenes de textos y gráficos escaneados: http://www.imageprocessingplace.com/root_files_V3/image_databases.htm*

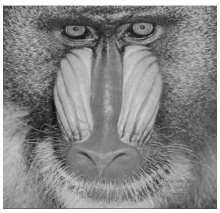
A continuación se describen cada imagen identificada por su nombre convencional , resolución y bits de profundidad y para cada una de ellas se ha calculado la entropía, los valores mínimos y máximos, y el valor medio. Un histograma para cada imagen está disponible para un rápido análisis de la luminosidad de la imagen.

IMÁGENES FOTOGRÁFICAS DE CONTINUOS TONOS

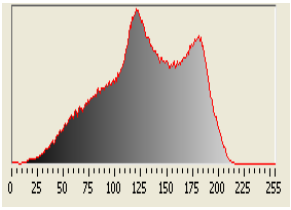
Lena 512x512
Área: 262144 píxeles
8 bits/píxel



Entropía:-7,44
Valor mínimo: 25
Valor Máximo: 245
Valor medio : 124,05

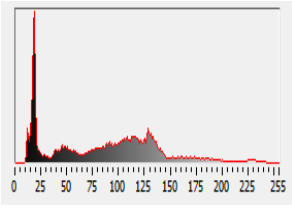


Baboon:512x512
Área: 262144 píxeles
8 bits/píxel



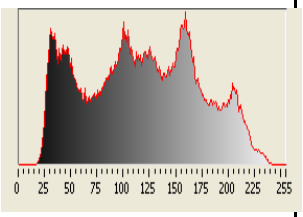
Entropía:-7,36
Valor mínimo: 0
Valor máximo: 230
Valor medio: 129,15

Man: 256x256
Área: 65536 píxeles
8 bits/píxel



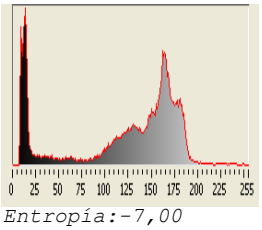
Entropía:-7,20
Valor mínimo: 6
Valor máximo: 253
Valor medio: 87,27

Bárbara: 512x512
Área: 262144 píxeles
8 bits/píxel



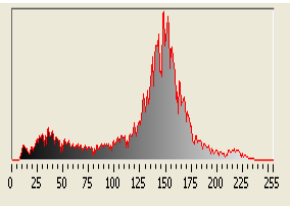
Entropía: -7,63
Valor mínimo: 12
Valor máximo: 246
Valor medio: 117,39

Cameraman:256x256
Área: 65536 píxeles
8 bits/píxel
Fuente:1



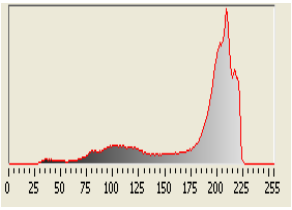
Entropía:-7,00
Valor mínimo: 7
Valor máximo: 253
Valor medio: 118,72

Boats: 512x512
Área: 262144 píxeles
8 bits/píxel
Fuente: *1*



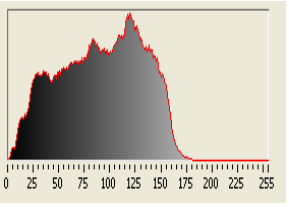
Entropía:-7,19
Valor mínimo: 0
Valor máximo: 255
Valor medio: 129,71

Airplane: 512x512
Área: 262144 píxeles
8 bits/píxel
Fuente: 1



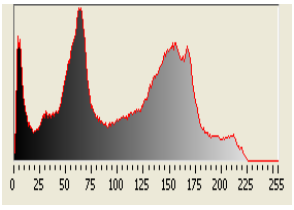
Entropía: -6,68
Valor mínimo: 25
Valor máximo: 232
Valor medio : 178,41

Zelda: 512x512
Área: 262144 píxeles
8 bits/píxel
Fuente: 1



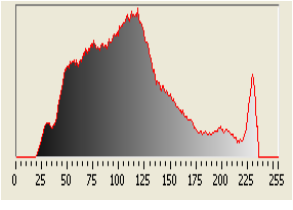
Entropía: -7,63
Valor mínimo: 0
Valor máximo: 187
Valor medio: 91,17

Peppers: 512x512
Área: 262144 píxeles
8 bits/píxel



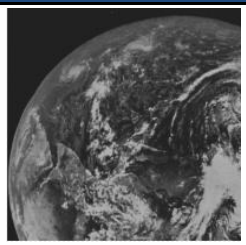
Entropía: -7,57
Valor mínimo: 0
Valor máximo: 229
Valor medio:104,20

Goldhill: 512x5112
Área 262144píxeles
8 bits/píxel

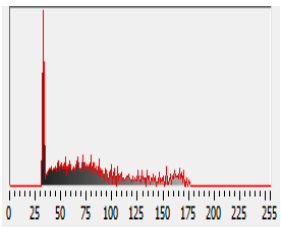


Entropía:-7,48
Valor mínimo: 16
Valor máximo:235
Valor medio:112,20

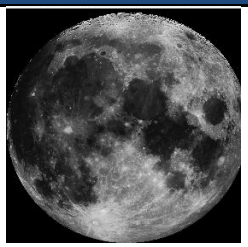
IMÁGENES AÉREAS Y SATÉLITE



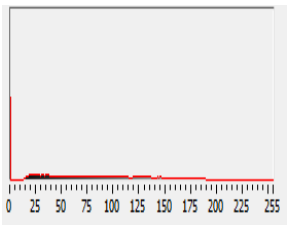
Earth: 200x200
Área: 40000 píxeles
8 bits/píxel



Entropía:-6,66
Valor mínimo: 31
Valor máximo: 176
Valor medio: 81,52

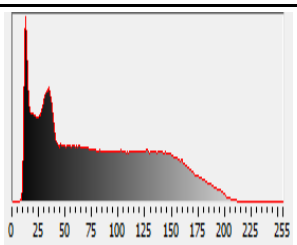


Moon: 1986x1986
Área: 3944196 píxeles
8 bits/píxel

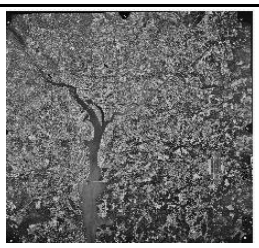


Entropía:-6,52
Valor mínimo: 0
Valor máximo: 254
Valor medio: 67,96

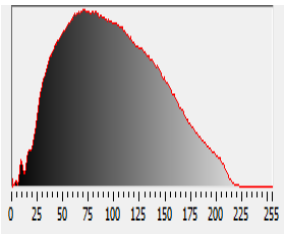
SanDiego: 1024x1024
Área: 1048576 píxeles
8 bits/píxel
Fuente: SIPI-USC



Entropía:-7,43
Valor mínimo: 4
Valor máximo: 226
Valor medio: 83,12

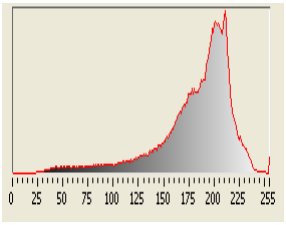


Washington: 2250x2250
Área: 5062500 píxeles
8 bits/píxel
Fuente: SIPI-USC



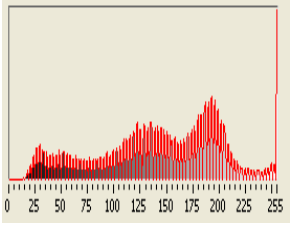
Entropía: -7,50
Valor mínimo: 0
Valor máximo: 233
Valor medio: 98,72

Aerial: 512x512
Área: 262144 píxeles
8 bits/píxel

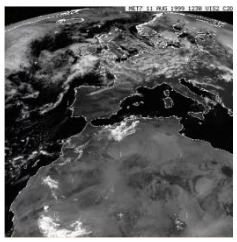


Entropía: -6,99
Valor mínimo: 12
Valor máximo: 255
Valor medio: 180,57

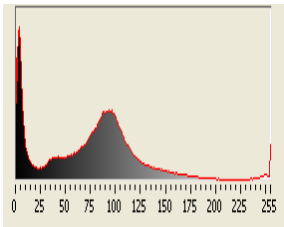
Airfield: 512x512
Área: 262144 píxeles
8 bits/píxel



Entropía: -7,12
Valor mínimo: 0
Valor máximo: 255
Valor medio: 143,46

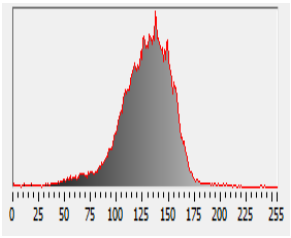


Meteosat: 800x800
Área: 640000 píxeles
8 bits/píxel



Entropía:-7,33
Valor mínimo: 0
Valor máximo: 255
Valor medio: 78,88

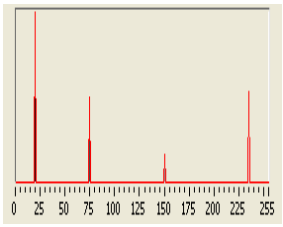
Moonsurface: 256x256
Área: 65536 píxeles
8 bits/píxel



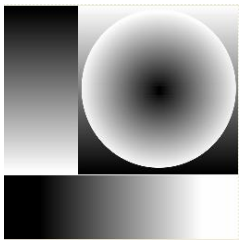
Entropía: -6,70
Valor mínimo: 0
Valor máximo: 249
Valor medio: 127,76

IMÁGENES GRÁFICAS CREADAS POR COMPUTADOR

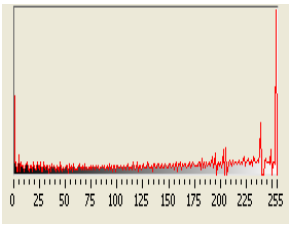
Circles: 256x256
Área: 65536 píxeles
8 bits/píxel



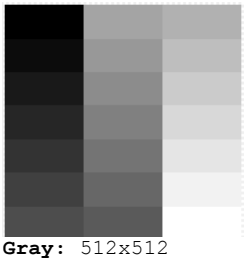
Entropía: -1,78
Valor mínimo: 20
Valor máximo: 235
Valor medio: 94,17



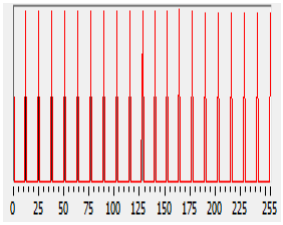
Slope: 256x256
Área:65536 píxeles
8 bits/píxel



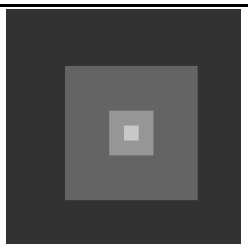
Entropía: -7,51
Valor mínimo: 0
Valor máximo: 254
Valor medio: 140,60



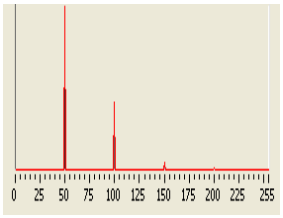
Gray: 512x512
Área: 262144 píxeles
8 bits/píxel



Entropía: -4,39
Valor mínimo: 0
Valor_máximo: 255
Valor medio: 27,03

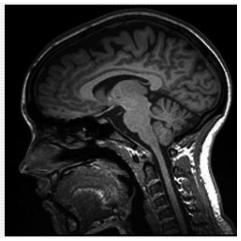


Squares: 256x256
Área: 65536 píxeles
8 bits/píxel

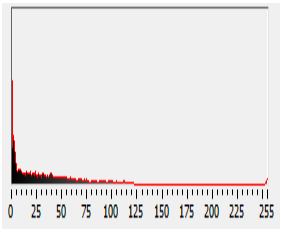


Entropía: -1,07
Valor mínimo: 50
Valor máximo: 200
Valor medio: 67,77

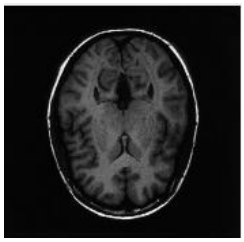
IMÁGENES MÉDICAS



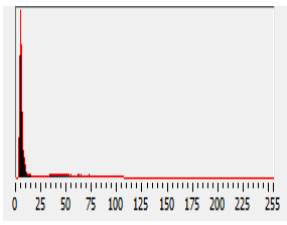
MRI_Brain_1: 250x286
Área:71500 píxeles
8 bits/píxel



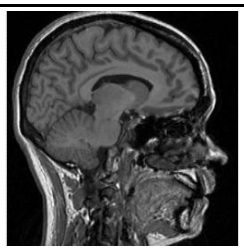
Entropía: -6,28
Valor mínimo: 0
Valor máximo: 255
Valor medio: 36,24



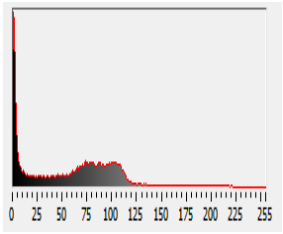
MRI_Brain_2: 350x350
Área:122500 píxeles
8 bits/píxel



Entropía: -4,97
Valor mínimo: 0
Valor máximo: 250
Valor medio: 26,45

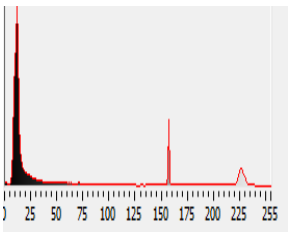


MRI_Brain_3: 512x512
Área: 262144 píxeles
8 bits/píxel



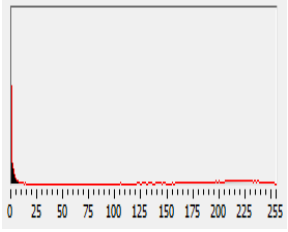
Entropía: -6,86
Valor mínimo: 0
Valor máximo:255
Valor medio: 59,87

Elbowx: 512x512
Área: 262144 píxeles
8 bits/píxel



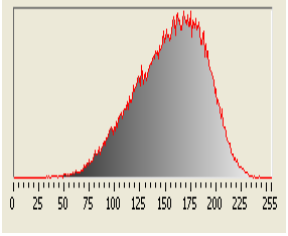
Entropia: -5,87
Valor mínimo: 0
Valor máximo: 245
Valor medio: 56,93

Shoulder: 451x347
Área: 156497 píxeles
8 bits/píxel



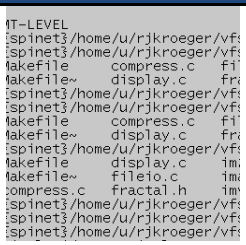
Entropía:
Valor mínimo: 0
Valor máximo: 255
Valor medio: 107,12

Finger: 256x256
Área:65536 píxeles
8 bits/píxel

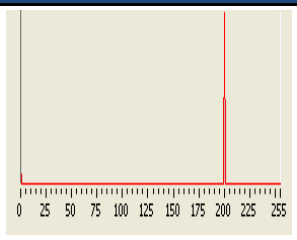


Entropia: -7,10
Valor mínimo: 0
Valor máximo: 254
Valor medio: 154,74

IMÁGENES DE TEXTOS Y GRÁFICOS ESCANEADOS

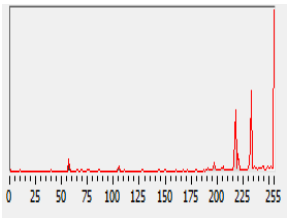


Texto: 256x256
Área: 65536 píxeles
8 bits/píxel



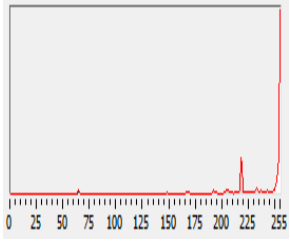
Entropía:
Valor mínimo: 0
Valor máximo: 200
Valor medio: 178,58

Mercados_1: 1025x979
Área: 1003475 píxeles
8 bits/píxel



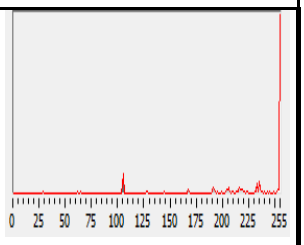
Entropía: -5,38
Valor mínimo: 0
Valor máximo: 255
Valor Medio: 208,59

Mercados_2:894x844
Área: 754536 píxeles
8 bits/píxel

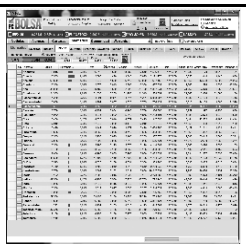


Entropía: -4,40
Valor mínimo: 0
Valor máximo: 255
Valor medio: 220,07

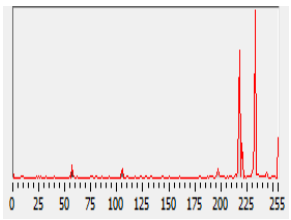
Mercados_3: 790x520
Área: 410800 píxeles
8 bits/píxel



Entropía: -4,97
Valor mínimo: 0
Valor máximo: 255
Valor medio: 212,45



Mercados_4:1027x981
Área: 1007487 píxeles
8 bits/píxel



Entropía: -5,07
Valor mínimo: 0
Valor máximo: 255
Valor medio: 196,96

Bibliografía

Amaru L, Gaillardon P.E., Burg A. y De Micheli G.(2014). Data compression via logic synthesis. En *Proceedings of the 19th Asia and South Pacific on Design Automation Conference (ASP-DAC)*, pp. 628-633.

Apple Computer (1984). Macintosh Picture. Inside Macintosh: operating system utilities (Mathematical and Logical utilities).

Bryant, J. (1993). *Louis Braille: Inventor*. ISBN 13: 9780791020777

Calderbank A. R., Daubechies I., Sweldens W. y Yeo B. L. (1996a). Wavelet transforms that map integers to integers. *Technical Report Department of Mathematics*, Princeton University.

Calderbank R. C., Daubechies I., Sweldens, W. y Yeo B. L. (1998b). Wavelet transforms that map integers to integers. *Applied and Computational Harmonic Analysis*, 5(3), pp. 332-369.

CCSDS Reports (2013). *Lossless data compression - Green Book Informational Report* CCSDS 120.0-G-3. Report concerning space data system standards.

Charlap D. (1994). The BMP file format. *Dr. Dobb's Journal*, 9(219), pp. 18-22.

Charlap D. (1995). Unlike other image-file formats like GIF (CompuServe's Graphic Interchange File format). *Dr. Dobbs's Journal*. 20(4), pp. 27-32.

Compuserve. (1990). Graphics Interchange Format programming reference. Descargado el 25/01/2012, de *http://www.w3.org/Graphics/GIF/spec-gif89a.txt*

Cormack G. V. y Horspool, R. N. (1984). Algorithms for adaptive Huffman codes. *Information Processing Letters Journal*, 18(3), pp. 159-165.

Daubechies I. y Sweldens W. (1998). Factoring wavelet transforms into lifting steps. *Journal of Fourier Analysis and Applications*. 4(3), pp 247-269.

Daubechies I. (1988). Orthonormal bases of compactly supported Wavelets. *Communications on Pure and Applied Mathematics,* 41(7), pp. 909-996.

Dogiwal S. R., Shishodia Y. S. y Upadhyaya A. (2014). Efficient *Lifting Scheme* based super resolution image reconstruction using low resolution images. En *Proceedings of International Conference on Advanced Computing, Networking and Informatics,* 27, pp. 259-266.

Fano R. M. (1949). Transmission of information. *M.I.T. Research of Electronics Technical Report*, 65.

Feldman A. y Ford P. (1989). *Scientists & inventors*. Ed.London Bloomsbury Books, ISBN-13: 978-0490004535.

Gilchrist J. (2004). Parallel compression with BZIP2. En *Proceedings of the 16th IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS)*, pp. 559-564.

Gilchrist J. y Cuhadar A. (2008). Parallel lossless data compression using on the Burrows-Wheeler transform. *International Journal of Web and Grid Services*, 4(1), pp. 117-135.

Golomb S. W. (1966). Run-length encodings. *IEEE Transactions on Information Theory*, 12(3), pp. 399-401.

Goswami J. C. y Chan A.K. (2011). *Fundamentals of Wavelets: theory, algorithms, and applications, Second Edition*. Ed. Kai Chang. ISBN: 9780470484135.

Haar A. (1910). Zur theorie der orthogonalen funktionensysteme. *Mathematische Annalen*, 69(3), pp. 331-371.

Han Z.-Z:, Liu Z.-B. y Xu Y.-S. (2011). A new adaptive wavelet transform using lifting scheme. En *Proceedings of Wavelet Analysis and Pattern Recognition (ICWAPR)*, pp. 224- 229.

Hattay J., Belaid S. y Naanaa W. (2013). Multiresolution convolutive blind source separation using adaptive lifting scheme. *IEEE Journal on Electronics, Circuits, and Systems*, 60(3), pp. 273-276.

Hemnath, P. y Prabhu, V. (2013). Compression of FPGA bitstreams using improved RLE algorithm. En *Proceedings of Information Communication and Embedded Systems (ICICES)*, pp. 834-839.

Howard P. G. y Vitter and J. S. (1993). Fast and eficient lossless image compression. En *Proceedings of the IEEE Data Compression Conference (DCC)*, pp. 351-360.

Howard P. G. y Vitter J. S. (1992). New methods for lossless image compression using Arithmetic Coding. *Journal of Information Processing and Management*, 28(6), pp. 765-779.

Hu, T. C. y Tucker, A. C. (1971). Optimal computer search trees and variable-length alphabetical codes. *SIAM Journal on Applied Mathematics*. 21(4), 514-532.

Huang J.-Y., Liang Y.-C. y Huang Y.-M. (2013). Secure integer arithmetic coding with adjustable interval size. En *Proceedings of the 19th Asia-Pacific Conference on Communications* (APCCC*)*, pp. 683- 687.

Huang Y.-M. y Liang Y.-C. (2011). A secure arithmetic coding algorithm based on integer implementation. En *Proceedings of Communications and Information Technologies (ISCIT)*, pp. 518- 521.

Hudson G., Yasuda H. y Sebestyen I. (1988). The international standardization of a still picture compression technique. En *Proceedings of IEEE Global Telecommunications Conference*, pp. 1016-1021.

Huffman, D. A. (1952). A method for the construction of minimum redundancy Codes. En *Proceedings of the I.R.E.*, 40(9), pp. 1098-1101.

ISO/IEC JTC1/SC29/WG (1991). *Information technology JPEG image compression for Lossless and near-lossless. compression of continuous-tone still images*.

ISO/IEC 11544 (1993). *Information technology - Coded representation of picture and audio information -- Progressive bi-level image compression*. ITU-T Rec. T.82

ISO/IEC 15444-1 (1999). *Information technology - JPEG 2000 image coding system*. ITU-T Rec. T800-- Part 1: Core coding system.

ISO/IEC 14495 (2000). *Information technology - Lossless and near-lossless. compression of continuous-tone still images – baseline.*

Jansen, M. H. y Oonincx, P. J. (2010). *Second Generation Wavelets and Applications*. Ed. Springer. ISBN 978-1-85233-916-6.

Jin Z., Au O.C, Wei D., Yue K., Luheng J. y Wenjing Z. (2013). A tutorial on image/video coding standards. En *Proceedings of Signal and Information Processing Association Annual Summit and Conference (APSIPA)*, pp. 1- 7.

Jingqi A., Mitra S. y Nutter B. (2014). Fast and efficient lossless image compression based on CUDA parallel wavelet tree encoding. En *Proceedings Image Analysis and Interpretation (SSIAI)*, pp. 21-24.

Kabir, M.A., Khan A.M., Islam M.T., Hossain M.L. y Mitul, A.F. (2013). Image compression using lifting based wavelet transform coupled with SPIHT algorithm. En *Proceedings of the 2013 International Conference on Informatics, Electronics and Vision (ICIEV)*, pp. 1-4.

Katz, P. (1989a). PKWARE, Inc. ("PKWARE"). Descargado el 20/11/2012 de, *http://www.pkware.com/documents/casestudies/appnote.txt.*

Katz, P. (1989b). PKWARE. Descargado el 20/11/2012 de, *http://www.pkware.com/software/pkzip.*

Klein P. N. (2013). *Coding the Matrix: Linear Algebra through Applications to Computer Science*. ISBN-13:978-0615880990.

Kaur S. M., Singh M. R. y Singh G. (2014). *An efficient lossless medical image compression*. Ed. LAP Lambert Academic. ISBN-13: 978-3659190278.

Khobragade P. B. y Thakare S. S. (2014a). Design and analysis of near lossless image compression techniques. Progress In Science and Engineering Research Journal, 2(3), pp. 193-200.

Khobragade P. B., Thakare S. S. y cols. (2014b). Image compression techniques - A review. International Journal of Computer Science and Information Technologies (IJCSIT), 5 (1), pp. 272-275.

Knuth D. E. (1985). Dynamic Huffman coding. *Journal of Algorithms*, 6(2), pp. 163-180.

Knuth, D. E. (1971). Optimum binary search trees. *Acta Informatica*. 1(1), pp. 14-25.

Langdon G. (1984). An introduction to Arithmetic coding. *IBM Journal of Research and Development*, 28(2), pp. 135-149.

Langdon, G. G. y Rissanen, J. J. (1983). A double-adaptive file compression algorithm. *IEEE Transactions on Communications Journal*, 31(11), pp. 1253-1255.

Larrauri J. I. (2002). A lossless compression algorithm for static colour images–INA. En *Proceedings of the First European Conference on Colour in Graphics, Imaging, and Vision (CGIV)*, pp. 420-423.

Larrauri J.I. (2002). A new lossless compression algorithm for static colour images. En *Proceedings of the IASTED International Conference on Signal Processing, Pattern Recognition and Applications*. (SPPRA), pp. 420-423.

Larrauri J.I. (2012). A new algorithm for lossless compression applied to two-dimensional static images. En *Proceedings of the 6th International Conference on Communications (CIT '12*), pp. 56-60.

Larrauri J.I. y Kahoraho E. (2003). A new method for lossless image compression in the field of artificial vision technology. En *Proceedings of the International Conference on Visual Information Engineering(VIE)*, pp. 242–245.

Larrauri J.I. y Kahoraho E. (2003). A new method for real time lossless image compression applied to artificial vision. En *Proceedings of Picture Coding Symposium (PCS)*, pp. 335-338.

Leger A., Omachi T. y Wallace G.K. (1991). JPEG still picture compression algorithm. *Optical Engineering Journal*, 30(7), pp. 947-954.

Lepik, Ü. y Hein H. (2014). *Haar Wavelets: with Applications (Mathematical Engineering)*. Ed. Springer. ISBN 13: 9783319042947.

Liang Z., Demin W. y Dong Z. (2012). Segmentation of source symbols for adaptive Arithmetic coding. *IEEE Transactions on Broadcasting*, 58(2), pp. 228-235.

Long S.y Xiang P. (2012). Lossless data compression for wireless sensor networks based on modified bit-level RLE. En *Proceedings of Wireless Communications, Networking and Mobile Computing (WiCOM)*, pp. 1- 4.

Macintosh (1988). Operating System Utilities Mathematical and Logical Utilities. Technical Note TN1023, *http://developer.apple.com/technotes/tn/tn1023.html*

Mallat, S. G. (1989). A theory for multiresolution signal decomposition: the wavelet representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 11(7), pp. 674-693.

Marv L. (1994). The BMP file format, *Dr. Dobb's Journal*, 9(10), 18-22.

Menaka E., Kumar S.S. y Bharathi M. (2013). Change detection in deforestation using high resolution satellite image with Haar wavelet transforms. En *Proceedings of the 2013 International Conference on Green High Performance*

Computing (ICGHPC'13), pp. 1-7.

Microsoft (2011). Bitmap compression. Internet resources for Windows developers. Descargado el 25/01/2012 de, *http://msdn2.microsoft.com/en-us/library/ s532328 (VS.85).aspx*

Mukherjee T., Swamy B.Y. y Bhavani M.V. (2014). Robust image compression using integer wavelet transform exploiting Lifting Scheme. *International Journal of Engineering Trends and Technology* (*IJETT*), 7(5), pp. 217-220.

Nelson, M. (1991). *The Data Compression Book*. Ed. M&T Books. ISBN:1558512160.

Nelson M. y Gailly J.-L. (1995). *The Data Compression Book 2nd edition*. Ed. M&T Books. ISBN 1-55851-434-1.

Pasco, R. (1976). Source coding algorithms for fast data compression. *Ph. D. dissertation*, Dept. of Electrical Engineering, Stanford University.

Pierce J.R. (1880).An Introduction to Information Theory: Symbols, Signals and Noise. ISBN-13: 978-0486417097.

Qi C., Li S., Guichun L. y Nam L. (2012). Lossy and lossless intra coding performance evaluation: HEVC, H.264/AVC, JPEG 2000 and JPEGLS. En *Proceedings of Signal & Information Processing*, pp. 1-9.

Raja S.P. y Suruliandi A. (2011). Techniques with different Wavelet Codecs Image Compression using WDR & ASWDR. ACEEE International Journal on Information Technology, 1(2).

Renugadevi S. yNithya Darisini, P.S. (2013). Huffman and Lempel-Ziv based data compression algorithms for wireless sensor networks. En *Proceedings of Pattern Recognition, Informatics and Mobile Engineering (PRIME)*, pp. 461-463.

Richter T. (2013). On the standardization of the JPEG XT image compression. En *Proceedings of Picture Coding Symposium (PCS)*, pp. 37-40.

Rissanen J.J. (1976). Generalized kraft inequality and Arithmetic Coding. *IBM Journal of Research and Development*. 20(3), pp. 198-203.

Rissanen, J. J. (1983). A Universal Data Compression System. *IEEE Transactions on Information,* 29(5), pp. 656-664.

Rubin, F. (1976). Experiments in Text File Compression. *Communications of ACM*, 19(11), pp. 617-623.

Sahni S., Vemuri B. C., Chen F., Kapoor C., Leonard C. y Fitzsimmons, J. (1998). State of the art lossless image compression algorithms. En *Proceedings of the International Conference on Image Processing*, pp. 948-952.

Said A. y Pearlman W. (1996). A new fast y efficient image codec based on set partitioning in hierarchical trees. *IEEE Transactions on Circuits and Systems for Video Technology*, 6(3), pp. 243-250.

Said A. y Pearlman W. A. (1993). Reversible image compression via multiresolution

representation and predictive coding. En *Proceedings of the SPIE 2094 on Visual Communications and Image Processing*, pp. 664-674.

Said A. y Pearlman W. A. (1996). An image multiresolution representation for lossless and lossy compression. *IEEE Transactions on Image Processing*, 5(9), pp. 1303-1310.

Schwartz E.S. y Kallick B., Generating an economical prefix code. *Communications of ACM,* 7, pp. 166-169, 1964.

Salomon D. y Motta G. (2010). *Handbook of Data Compression.* ISBN-13: 978-1848829022.

Savakis A. E. (2002). Evaluation of algorithms for lossless compression of continuous-tone images. *Journal of Electronic Imaging*, 11(1), pp. 75–86.

Sayood K. (2012). *Introduction to Data Compression* (The Morgan Kaufmann Series in Multimedia Information and Systems). ISBN-13: 978-0124157965.

Sedgewick R. y Wayne K. (2011) Algorithms. ISBN-13: 978-0321573513

Shannon C. E. (1948). A mathematical theory of communication. *The Bell System Technical Journal*, 27(3), pp. 379–423.

Shapiro J. M. (1993). Embedded image coding using ZeroTrees of Wavelet coefficients. *IEEE Transactions on Signal Processing*, 41(2), pp. 3445-3462.

Shuai D. (2008). Parallel lossless data compression: a particle dynamic approach. En *Proceeding of the 4th International Conference on Intelligent Computing (ICIC)*, pp. 266-274.

Skodras A., Schelkens P. y Ebrahimi T. (2009). *The JPEG 2000 Suite*. Ed. Wiley-IS&T Series in Imaging Science and Technology. ISBN: 9780470721476.

Sodagar I., Bing-Bing C. y Wus J. (2000). A new error resilience technique for image compression using arithmetic coding. En *Proceedings of Acoustics, Speech, and Signal Processing*, pp. 2127-2130.

Stephen W. (2002). *A New Kind of Science*. ISBN-13: 978-1579550080

Stockwell R. G., Mansinha L y Lowe R P. (1996). Localization of the complex spectrum: The S Transform. *IEEE Transactions on Signal Processing*, 44(4), pp. 998-1001.

Sweldens W. (1996). The Lifting Scheme: a custom design construction of biorthogonal wavelets. *Journal of Applied and Computational Harmonic Analysis*, 3(2), pp.186-200.

Tamakoshi Y., Tomohiro I., Inenaga S., Bannai H. y Takeda M. (2013). From Run Length Encoding to LZ78 and back again. En *Proceedings of the IEEE Data Compression Conference (DCC)*, pp. 143- 152.

Tian J. y Wells R.O. (1996). A lossy image codec based on index coding. En *Proceedings of the IEEE Data Compression Conference, (DCC)*, pp. 456-460.

Tsukimaya y cols. (12/07/1989). US patente nº 4.872.009. *Method y apparatus for data compression y restoration*".

Tsukimaya y cols. (1986).Hitachi (08/07/1986). US patente nº 4.586.027. *Method and system for data compression and restoration*.

Vitter J. S. (1987). Design and analysis of dynamic Huffman codes. *Journal of the ACM*, 34(4), pp 825–845.

Vitter J. S. (1989). Algorithm 673 Dynamic Huffman coding, *ACM Transactions on Mathematical Software*, 15(2), pp, 158–167.

Wallace G.K. (1991). The JPEG still picture compression standard. *Communications of the ACM*, 34(4), pp. 31-44.

Wang B. R. (2012). *Introduction to Orthogonal Transforms*. Cambridge University Press. ISBN-13: 978-0-521-51688-4.

Wang Z., Chanda D., Simon S. y Richter, T. (2012a). Memory efficient lossless compression of image sequences with JPEG-LS and temporal prediction. En *Proceedings of Picture Coding Symposium (PCS)*, pp. 305-308.

Wang Z., Klaiber, M., Gera, Y., Simon, S. y Richter, T. (2012b). Fast lossless image compression with 2D Golomb parameter adaptation based on JPEG-LS. En *Proceedings Signal Processing Conference (EUSIPCO),* pp. 1920- 1924.

Weinberger M. J., Seroussi G. y Sapiro G. (1996). LOCO-I: A low complexity, context-based, lossless image compression algorithm. En *Proceedings of the IEEE Data Compression Conference (DCC)*, pp. 140–149.

Welch T. A. (1984), A Technique for High-Performance Data Compression. *IEEE Computer*. 17(6), pp. 8-19.

Witten, I. H., Neal, R. M. y Cleary, J. G. (1987). Arithmetic Coding for data compression. *Communications of the ACM*, 30(6), pp. 520-540.

Wu X y Memon N. (1996). CALIC-a context based adaptive lossless image codec. *Proceedings of 1996 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP),* 4, pp. 1890–1893.

Yang X., Zhu Z. y Yang B. (2008). Adaptive Lifting Scheme for image compression. En *Proceedings of the Fifth International Conference on Fuzzy Systems and Knowledge Discovery (FSKD '08)*, pp. 547-551.

Yaguchi, K., Kobayashi, N. y Shinohara, A. (2014). Efficient Algorithm and Coding for Higher-Order Compression. *Proceedings of the IEEE Data Compression Conference (DCC)*, pp. 434-439.

Zhong Z., Hosokawa S., Toda H., Imamura T. y Miyake, T. (2013). Application of the Lifting Scheme to variable filter band discrete wavelet transform. *Proceedings of Wavelet Analysis and Pattern Recognition (ICWAPR)*, pp. 103-109.

Ziv J. y Lempel A. (1977). A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3), pp. 337-342.

Ziv J. y Lempel A. (1978). Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory*, 24(5), pp. 530-536.