FACULTAD DE MATEMÁTICA ASTRONOMÍA FÍSICA Y COMPUTACIÓN
Universidad Nacional de Córdoba

Trabajo Especial de Licenciatura en Ciencias de la Computación

Desarrollo de un algoritmo de compresión de datos optimizado para imágenes satelitales

Autor: Cruz, Kouichi Julián Andrés
Director: Lanfri, Mario Alberto
Profesor representante: Bustos, Oscar Humberto
Córdoba, 2017

Agradecimientos

Esta tesis marca el final de la Licenciatura en Ciencias de Computación. Esta es la mejor oportunidad para agradecer a todas las personas que me han apoyado a cumplir este objetivo.

Agradezco a mi tía Alejandra y a mi tío Gustavo, quienes fueron, son y serán como padres para mí. Su incontable apoyo, paciencia, afectos y buena energía que recibí durante toda mi vida me ha sido muy importante para llegar a este momento. A mi abuela Rosa y a mi tío Aldo, por todo el soporte que me han dado desde mi infancia.

A mis amigos Nico, Iván, Mariano y Migue, quienes han sido buena onda conmigo desde el comienzo de esta carrera, y con quienes compartí innumerables juntadas. A mis amigos Nacho y Maico, por todas las experiencias compartidas en este camino, y por todas las que quedan. Mi paso por esta casa de estudios no sería lo mismo sin ellos.

A Oscar, por ser una excelente persona. A Daniel, por su útil consejo.

Y a toda la gente que he conocido a lo largo de estos años, compañeros, profesores y amigos.

Resumen

Las imágenes satelitales son cada vez de mayor tamaño, de tal manera que hoy en día hablar en términos de gigabytes ya es normal. A la hora de generar productos que formen un mosaico también nos encontramos con grandes volúmenes lo cual no solo dificulta el procesamiento sino también la transferencia o distribución de los mismos hacia los usuarios. Finalmente, también tenemos el problema del manejo de los datos tanto a bordo de la plataforma del satélite como de su bajada a tierra.

En esta tesis, se desarrollará e implementará un algoritmo de compresión con pérdida orientado a resolver esta problemática, utilizando la Transformada Discreta de Wavelets y la codificación Huffman.

Abstract

Satellite images are increasing in size, to the point where speaking in the order of Gigabytes is normal. We also find big volumes of data when generating products such as mosaics, which makes both their processing and their transfer to end users more difficult. Finally, we also have large volumes of data in both the satellites themselves and in their transfer back to Earth.

In this thesis, we design and implement a lossy compression algorithm targeted to solve this topic, making use of the Discrete Wavelet Transform and the Huffman coding.

Índice general

Capítulo 1

Introducción

1.1. Motivación

En las últimas décadas se ha incrementado exponencialmente el volumen de datos generados por sensores remotos a bordo de satélites, a tal punto que los desarrolladores de sistemas satelitales de resolución espacial submétrica han solicitado a la comunidad científica aportes al desarrollo de nuevos paradigmas para un manejo, almacenamiento y transmisión más inteligente de los datos.

En Argentina, el lugar de ingestión, preprocesamiento, almacenamiento y catalogación de los datos espaciales es la Estación Terrena Córdoba (ETC) con asiento en el Centro Espacial Teófilo Tabanera (CETT).

Operando desde 1977, la ETC ha ido incorporando sucesivamente a la grilla de captación de datos a satélites cada vez más modernos y por ende con mayores capacidades de almacenamiento a bordo y con sensores de mayor resolución espacial y radiométrica, lo cual a dado como resultado que hoy sea normal tener una imagen con un tamaño del orden de los Gigabytes.

Este resultado genera la necesidad de realizar la transferencia de los datos desde los satélites a la Tierra de manera más eficiente. En esta tesis se desarrolla, implementa y verifica un algoritmo de compresión de imágenes satelitales orientado a resolver esta problemática.

1.2. Conceptos básicos

Antes de entrar en el detalle de esta tesis, se introducirán conceptos esenciales que serán utilizados a lo largo de todo el documento.

1.2.1. Generalidades de compresión de datos

El objetivo de la compresión es eliminar la redundancia de la información para transmitir datos más eficientemente. Existen dos tipos de compresión: compresión sin pérdida y compresión con pérdida. En el caso de la compresión sin pérdida, los datos pueden recuperarse en su totalidad. En el caso de compresión con pérdida, algunos datos no podrán recuperarse de manera exacta. Ésto ocurre ya que la compresión con pérdida elimina la irrelevancia, es decir, elimina datos que pueden ser obtenidos en función de otros datos. El procedimiento de eliminación de irrelevancia se denomina *cuantización*.

1.2.2. Generalidades de imágenes satelitales

Las imágenes satelitales pueden ser de varios tipos, por ejemplo: multiespectrales, hiperespectrales y de radar.

- Las imágenes multiespectrales e hiperespectrales son aquellas que contienen información procedente de analizar algunas frecuencias del espectro electromagnético de manera pasiva, es decir, en función del reflejo de una onda electromagnética procedente de otra fuente (en este caso, la luz solar). Esta información se coloca en *bandas* que componen una imagen. La cantidad de bandas de una imagen espectral será la que determine si es multiespectral o hiperespectral.

- Un radar (**RA**dio **D**etection **A**nd **R**anging) es un sensor activo, el cual emite pulsos de microondas y mide la energía reflejada. Tanto los pulsos recibidos como los transmitidos están polarizados horizontalmente o verticalmente. Estos canales polarizados se denominan *canal H* y *canal V*, respectivamente. En la imagen se coloca la intensidad del pulso recibido para una o más combinaciones de canales de transmisión y canales de recepción.

Para comprimir una imagen, se debe comprimir cada banda individualmente.

1.2.2.1. Resolución

En teledetección, existen cuatro tipos de resoluciones para las imágenes satelitales [Chuvieco, 1995, p.90-97]:

Resolución espacial La resolución espacial se asocia a la distancia representada por cada píxel de una imagen. Típicamente, este número varía entre 1 y 1000 metros.

Resolución espectral La resolución espectral es la cantidad de bandas de longitudes de onda almacenadas en una imagen.

Resolución radiométrica La resolución radiométrica es la cantidad de bits por cada píxel por cada banda. Esto indica la cantidad de intensidades que el sensor es capaz de distinguir. Típicamente, este número varía entre 8 y 14 bits.

Resolución temporal La resolución temporal se asocia a la frecuencia con la que un satélite toma imágenes de una misma área. Evidentemente, esta resolución es irrelevante para los fines de esta tesis.

La calidad de los datos obtenidos estará dada por las cuatro resoluciones mencionadas.

1.3. Estado del arte: otros algoritmos de compresión

1.3.1. Algoritmos que no utilizan wavelets

Uno de los algoritmos más notables y más conocidos que no utilizan transformada de wavelets es el algoritmo JPEG. Este algoritmo utiliza la transformada discreta del coseno, otra transformada matemática que pasa funciones a una base de espacio de funciones conformada por cosenos.

1.3.2. Algoritmos que utilizan wavelets

Entre los algortimos más conocidos que utilizan wavelets se encuentran:

JPEG2000 Este algoritmo se considera el sucesor de JPEG y permite tanto compresión con pérdida como sin pérdida.

MrSID Este método fue específicamente desarrollado para *Sistemas de Información Geográfica* (GIS, por sus siglas en inglés). Si bien obtiene en promedio ratios de compresión 2:1 en modo sin pérdida, obtiene ratios mucho más altos para compresión con pérdida.

ECW Otro algoritmo más reciente que utiliza wavelets. Usualmente obtiene ratios desde 25:1 a 40:1. [Ueffing, 2001]

Es importante notar que todos éstos son de propietario.

1.4. Estructura de la tesis

En esta sección se detalla el orden de los temas a presentar en esta tesis, detallando que contendrá cada capítulo.

Capítulo 2: Base teórica. Antecedentes. Técnicas básicas En el capítulo 2 se introducirán los conceptos teóricos utilizados. Se explicarán algoritmos de codificación, tales como RLE, y transformadas matemáticas, tales como la transformada de Fourier y la transformada de wavelets.

Se entrará en profundidad en este último, ya que es la técnica que se utiliza en este trabajo.

Capítulo 3: Desarrollo de la solución. Tecnologías adoptadas. Arquitectura En el capítulo 3 se explicará el diseño de la solución. Se especificarán los requerimientos y la arquitectura de la implementación.

Se mostrarán diagramas que muestren la estructura del programa a implementar, tales como el diagrama de arquitectura, y los diagramas de secuencia.

Capítulo 4: Implementación En el capítulo 4 se entrará en profundidad en cuestiones de implementación.

Se mostrará la elección de las herramientas para la implementación, junto con una implementación en pseudocódigo.

Capítulo 5: Verificación de la solución. Elección del set de imágenes de testeo. Análisis de los resultados En el capítulo 5 se mostrará la verificación de la solución. Se evaluará la correctitud de la implementación del algoritmo con algunas imágenes satelitales, cuya elección también estará documentada en este capítulo.

En función de las imágenes elegidas, se dará un análisis de performance. Se dará una comparación con un algoritmo de uso general, ZIP.

Capítulo 6: Conclusiones y trabajo futuro En el capítulo 6 se explicará el trabajo futuro a realizar para este trabajo, y se cerrará con una conclusión.

Capítulo 2

Base teórica. Antecedentes. Técnicas básicas

Previamente a la explicación del proyecto en sí, se deben explicar conceptos básicos que se utilizarán a lo largo de todo el desarrollo. Estos conceptos incluyen elementos referentes a las imágenes en sí, algoritmos matemáticos de codificación, y transformadas matemáticas que permitan pasar los datos a otra representación, con su respectivo algoritmo de cálculo. Todos estos conceptos necesarios estarán presentes en esta sección.

2.1. Imágenes

Una imagen es un arreglo rectangular de puntos, de m filas y n columnas. La expresión $m \times n$ se denomina *tamaño* de la imagen, y cada punto se lo denomina *píxel*. Usualmente, las imágenes suelen clasificarse en [Salomon et al., 2010, p.446,447]:

1. Imagen *bi-nivel* (o *monocromática*): en estas imágenes, los píxeles pueden tener sólo dos posibles valores (usualmente referidos como blanco y negro). Cada píxel es representado por 1 bit.

2. Imagen *de escala de grises*: un píxel puede tener valores entre 0 y $2^n - 1$, donde n es la cantidad de bits.

3. Imagen *de tono continuo*: Estas imágenes tienen muchos colores similares. Si los píxeles difieren en una unidad, la diferencia es casi imperceptible al ojo humano. Cada píxel es representado o por un único número de gran tamaño o por múltiples componentes. Suelen ser imágenes "no artificiales", generalmente obtenidas por una cámara digital.

4. Imagen *de tono discreto* o imagen *sintética*: Estas imágenes usualmente son generadas artificialmente. Los objetos artificiales poseen bordes bien definidos que contrastan con el resto de la imagen, el fondo de la misma. Esto implica que píxeles contiguos pueden o bien ser idénticos o bien variar significativamente. [Salomon et al., 2010, p.447]

5. Imagen de tipo *cartoon*[1]: Estas imágenes consisten de múltiples áreas. Cada área tiene un color uniforme, pero áreas adyacentes pueden ser idénticas o muy diferentes.

Todos estos tipos de imágenes pueden tener redundancia. Sin embargo, ésta es diferente para cada tipo de imágenes. La imposibilidad de realizar un algortimo de compresión que sea útil para todos los tipos de redundancia es la razón por la cual no existe un algoritmo para *todos* los tipos de imágenes. Por este motivo, existen diferentes algoritmos para cada tipo de imagen.

2.2. Algoritmos básicos

2.2.1. Primera aproximación: RLE

El algoritmo **R**un **L**ength **E**ncoding es el algoritmo de compresión más simple, y que sirve como base para algoritmos más sofisticados. La idea del algoritmo es reemplazar un valor d que aparece n veces consecutivas por el par nd. Dicho algoritmo aplica esta idea, pero teniendo algunas consideraciones.

Realizar la aplicación directa de la idea no funciona, pues, por ejemplo, reemplazar la cadena `"2._gg_izi_tutorial_well"` por `"2._2g_izi_tutorial_we2l"` no puede ser decodificada inambiguamente (no puede diferenciar el primer `"2"` como un caracter independiente o una cantidad de repeticiones de otro caracter).

Una solución es utilizar un caracter delimitador (en este ejemplo, se utilizara `"#"`). La cadena entonces quedaría `"2._#2g_izi_tutorial_we#2l"`. Sin embargo, esta cadena es más larga, pues se reemplazan dos caracteres consecutivos por tres caracteres. Para solucionar esto, se opta por no convertir repeticiones de 2 caracteres.

Evidentemente, no se obtienen altas relaciones de compresión con este algoritmo si los datos a comprimir tienen baja repetición.

[1] Dibujos animados

2.2.2. Códigos de Huffman

A diferencia del método anterior, éste es un método probabilístico. Este algoritmo representa cada elemento que se desee codificar como una cadena binaria. Las cadenas generadas por este algoritmo no poseen la misma longitud, otorgándole códigos más cortos a los elementos más probables. Este tipo de código se denomina *código de longitud variable*.

Los códigos de Huffman pertenecen a la categoría de *prefix codes*, la cual se verá a continuación.

2.2.2.1. Prefix codes

En los códigos de prefijo (*prefix codes*), se cumple la propiedad que ninguna codificación de un dato es un prefijo de la codificación de otro dato. Se puede demostrar que con un código de prefijos se obtiene la compresión más óptima de un conjunto de datos. [Cormen et al., 2009, p.429].

Codificar una cadena de datos es simple para todo tipo de códigos binarios: se concatena la codificación de cada dato individualmente. Es decir, para una función de codificación $E(a)$ y una cadena de datos $a_1 \dots a_m$, se cumple que:

$$E(a_1 \dots a_m) = E(a_1) \dots E(a_m)$$

Con los códigos de prefijos, la descompresión de una cadena de datos también resulta más sencilla, ya que hace que el primer elemento de la cadena codificada pueda ser determinada inambiguamente. Una vez detectada, se la reemplaza por el dato original, y se repite el procedimiento con el resto de la cadena.

Para el decodificador, se busca que el primer dato codificado sea fácil de obtener. Esto se puede lograr con un árbol binario que tenga los datos en las hojas [Cormen et al., 2009, p.430], como se verá a continuación.

2.2.2.2. Construcción de los códigos

Dados unos posibles datos $a_1, \dots, a_n$, donde cada uno tiene una probabilidad de ocurrencia $p_1, \dots, p_n$ se define un árbol con costos en las aristas, constuído de la siguiente forma:

1. Se crea una lista con todos los datos $a_1, \dots, a_n$ y sus respectivas probabilidades.

2. En las hojas se colocan los símbolos con sus respectivas probabilidades de ocurrencia.

3. Se toman los dos nodos a_i y a_j de la lista que tienen la menor probabilidad de ocurrencia, y se crea un nodo a_{ij}, el cual tendra una probabilidad de ocurrencia $p_i + p_j$. Luego se eliminan los nodos a_i y a_j de la lista y se añade el nodo a_{ij} con su probabilidad.

Se aplica este algoritmo recursivamente hasta que quede sólo 1 elemento en la lista. Este elemento corresponderá a la raíz del árbol, denominada r.

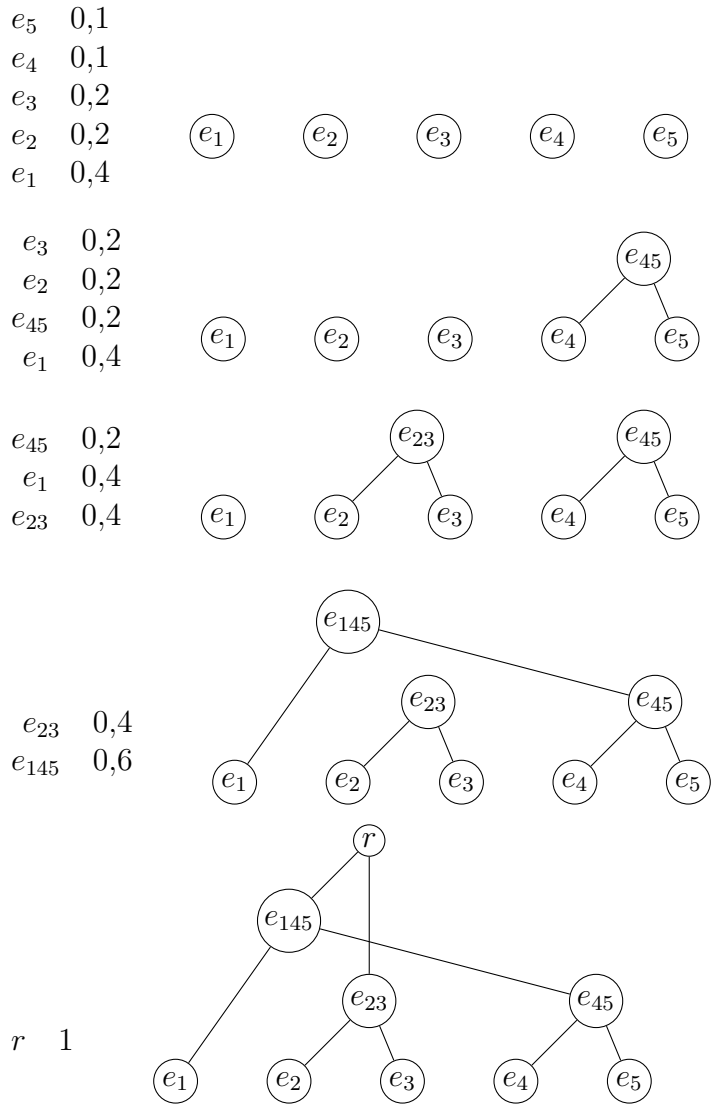


Figura 2.1: Ejemplo de construcción del árbol del algoritmo de Huffman

A modo de ejemplo, suponer 5 símbolos e_1, e_2, e_3, e_4, e_5 con probabilidades $0{,}4; 0{,}2; 0{,}2; 0{,}1; 0{,}1$, respectivamente. La construcción del árbol para este ejemplo se muestra en la figura 2.1.

Una vez completo el árbol, se recorre el arbol asignandole arbitrariamente 1's y 0's a cada arista. El camino desde la raíz hasta la hoja i determinará la representación que tendrá el valor a_i. La reconstrucción del ejemplo se muestra en la figura 2.2

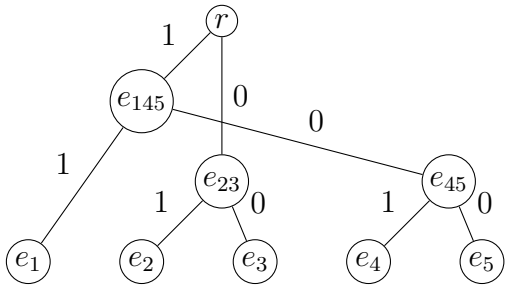


Figura 2.2: Ejemplo de reconstrucción de códigos

Luego se reemplaza cada a_i por el valor obtenido por el árbol. Al final de la nueva cadena, se colocan las probabilidades de ocurrencia, para ser usadas por el descompresor, el cual realiza el mismo procedimiento para obtener los valores codificados. Para el ejemplo, se obtienen los códigos mostrados en la tabla 2.3.

e_1	11
e_2	01
e_3	00
e_4	101
e_5	100

Figura 2.3: Códigos obtenidos

Es importante notar que el árbol construido no es único. Para el ejemplo anterior, el árbol de la figura 2.4, por ejemplo, también es válido.

Estos árboles generarán dos codificaciones diferentes al realizar la reconstrucción ¿Cuál de los códigos generados es mejor? Para conocer la respuesta, se calcula el *costo* del árbol generado [Cormen et al., 2009, p.431]. Si llamamos al árbol generado por la figura 2.1 como T_1 y al árbol generado por la figura 2.4 como T_2, $B(T)$ denota el costo del árbol T, y $|e|$ como la longitud de la cadena e, entonces

$$B(T_1) = \sum_{i=1}^{n} p_i|e_i| = 0{,}4 \cdot 2 + 0{,}2 \cdot 2 + 0{,}2 \cdot 2 + 0{,}1 \cdot 3 + 0{,}1 \cdot 3 = 2{,}2$$

e_1	1
e_2	01
e_3	001
e_4	0001
e_5	0000

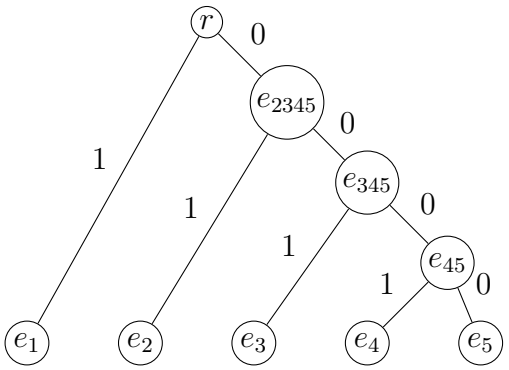


Figura 2.4: Otro árbol válido

$$B(T_2) = \sum_{i=1}^{n} p_i |e_i| = 0{,}4 \cdot 1 + 0{,}2 \cdot 2 + 0{,}2 \cdot 3 + 0{,}1 \cdot 4 + 0{,}1 \cdot 4 = 2{,}2$$

Ambos códigos otorgan la misma media. Se puede demostrar que todo código generado por el algoritmo de Huffman es óptimo [Cormen et al., 2009, p. 433-435].

Una implementación estándar utilizando cola de prioridades tendrá complejidad $\mathcal{O}(n \log(n))$. Esto es así porque:

- La construcción del árbol requiere $n-1$ nodos adicionales. Realizar una operación en una cola de prioridades tiene complejidad $\mathcal{O}(\log(n))$, la cual se realiza una vez por cada nodo. Es decir, se realiza $n-1$ veces una operación $\mathcal{O}(\log(n))$, lo cual tiene complejidad $\mathcal{O}(n \log(n))$

- Todo árbol de n nodos tiene $n-1$ aristas [Cormen et al., 2009, p. 1174-1176]. En este caso, el árbol generado tiene n nodos iniciales más $n-1$ nodos añadidos, para un total de $2n-1$ nodos. Esto implica que hay $2n-2$ aristas. Para obtener la codificación de cada símbolo, se deben recorrer todas las aristas una única vez, lo cual tiene una complejidad $\mathcal{O}(2n) = \mathcal{O}(n)$

2.3. Transformadas matemáticas

Las transformadas matemáticas son utilizadas para transformar valores de una representación donde estén correlacionados a otra representación donde no lo estén. [Salomon et al., 2010] Entre las transformadas más útiles se encuentran la transformada del coseno, la transformada rápida de Fourier y la transformada de wavelets.

2.3.1. Transformada del coseno

La transformada del coseno es una de las más utilizadas en compresión. Entre otros, es utilizado por los formatos JPEG (imágenes) y MPEG (audio).

Se utiliza tanto la versión unidimensional y la bidimensional.

2.3.1.1. Transformada del coseno unidimensional

Se toman n funciones $w(f)$, tal que $w(f) = \cos(f\theta)$, para $0 \leq \theta \leq \pi$ y $f = 0, \ldots, n-1$. A cada una de estas funciones se las muestrea en los puntos:

$$\theta = \frac{(2k+1)\pi}{2n}$$

para $k = 0, \ldots, n-1$. Cada uno de estos valores será un valor en el vector $\mathbf{v}_f$, para $f = 0, \ldots, n-1$. Este conjunto de n vectores forman una base ortonormal, luego se puede expresar cualquier vector como una combinación lineal de los $\mathbf{v}_i$. La forma mas simple de calcular la transformada de un vector $\mathbf{p}$ es:

$$G_f = \sqrt{\frac{2}{n}} C_f \sum_{j=0}^{n-1} p_j \cos\left(\frac{(2j+1)f\pi}{2n}\right) \tag{2.1}$$

$$\text{donde } C_f = \begin{cases} \frac{1}{\sqrt{2}}, & f = 0 \\ 1\ , & f > 0 \end{cases} \text{ para } f = 0, \ldots, n-1$$

La forma más simple de calcular la transformada inversa es:

$$p_t = \sqrt{\frac{2}{n}} \sum_{j=0}^{n-1} C_j G_j \cos\left(\frac{(2t+1)j\pi}{2n}\right), \text{ para } t = 0, \ldots, n-1 \tag{2.2}$$

2.3.1.2. Transformada del coseno bidimensional

Usualmente, los píxeles de las imágenes suelen tener correlación en ambas dimensiones, por lo cual se utiliza la transformada bidimensional del coseno.

La fórmula para la transformada bidimensional es:

$$G_{ij} = \frac{2}{n} C_i C_j \sum_{x=0}^{n-1} \sum_{y=0}^{n-1} p_{xy} \cos\left(\frac{(2y+1)j\pi}{2n}\right) \cos\left(\frac{(2x+1)i\pi}{2n}\right) \tag{2.3}$$

para $i, j = 0, \ldots, n-1$. La transformada inversa es:

$$p_{xy} = \frac{2}{n} \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} C_i C_j G_{ij} \cos\left(\frac{(2x+1)i\pi}{2n}\right) \cos\left(\frac{(2y+1)j\pi}{2n}\right) \tag{2.4}$$

2.3.2. Transformada de Fourier

La transformada de Fourier descompone una función del tiempo[2] en su equivalente en frecuencia.

2.3.2.1. Transformada de Fourier continua

Las funciones que usualmente se utilizan suelen estar en el dominio del tiempo, es decir, tienen el tiempo como parámetro. Para las funciones periódicas, se las puede analizar con la frecuencia como un parámetro. Esto se puede realizar con la *transformada continua de Fourier*, cuya formula para una función $g(t)$ es:

$$G(f) = \int_{-\infty}^{\infty} g(t) e^{-i2\pi ft} \, \mathrm{d}t$$

$$G(f) = \int_{-\infty}^{\infty} g(t)[\cos(2\pi ft) - i\sin(2\pi ft)] \, \mathrm{d}t \tag{2.5}$$

La transformada inversa es:

$$g(t) = \int_{-\infty}^{\infty} G(f) e^{i2\pi ft} \, \mathrm{d}f$$

$$g(t) = \int_{-\infty}^{\infty} G(f)[\cos(2\pi ft) + i\sin(2\pi ft)] \, \mathrm{d}f \tag{2.6}$$

2.3.2.2. Transformada de Fourier discreta

En el caso de los datos computacionales, se utiliza la transformada discreta de Fourier, ya que usualmente tienen una cantidad finita n de datos. La transformada es:

$$G(f) = \sum_{t=0}^{n-1} g(t) e^{-i\frac{2\pi}{n}ft}$$

$$G(f) = \sum_{t=0}^{n-1} g(t) \left[\cos\left(\frac{2\pi ft}{n}\right) - i\sin\left(\frac{2\pi ft}{n}\right) \right] \tag{2.7}$$

para $f = 0, 1, \ldots, n-1$. La transformada inversa es:

$$g(t) = \frac{1}{n} \sum_{f=0}^{n-1} G(f) e^{i\frac{2\pi}{n}ft}$$

[2]En imágenes, no existe el concepto de "tiempo". Sin embargo, se utiliza el tiempo como variable en base a los formalismos del análisis de señales

$$g(t) = \frac{1}{n}\sum_{f=0}^{n-1} G(f)\left[\cos\left(\frac{2\pi ft}{n}\right) + i\sin\left(\frac{2\pi ft}{n}\right)\right] \tag{2.8}$$

para $t = 0, 1, \ldots, n-1$.

2.4. Wavelets

Al igual que la transformada de Fourier, la transformada de wavelets permite transformar una función de una base canónica a otra base. Mientras que la transformada de Fourier utiliza una base de senos y cosenos, las transformadas de wavelets utilizan funciones denominadas *wavelets*, $\psi(t)$. Un wavelet es una función continua (ya sea real o compleja) tal que:

1. La función es de energía finita, es decir,

$$\int_{-\infty}^{\infty} |\psi(t)|^2 \, \mathrm{d}t < \infty \tag{2.9}$$

2. Si $\Psi(f)$ es la transformada de Fourier de la función, se debe cumplir que

$$C_g = \int_0^{\infty} \frac{|\Psi(f)|^2}{f} \, \mathrm{d}f < \infty$$

   Esta condición se denomina *condición de admisibilidad* y el valor C_g se denomina *constante de admisibilidad*.

3. (*sólo para funciones complejas*) La transformada de Fourier de la función debe ser real, y ser igual a 0 para frecuencias negativas.

Entre los wavelets más comunes, se encuentran el *wavelet sombrero mejicano* (figura 2.5):

$$\psi(t) = (1 - t^2)e^{-\frac{t^2}{2}} \tag{2.10}$$

el *wavelet gaussiano*, equivalente a la derivada de una gaussiana estándar (figura 2.6):

$$\psi(t) = -te^{-\frac{t^2}{2}} \tag{2.11}$$

y el *wavelet de Haar* (figura 2.7):

$$\psi(t) = \begin{cases} 1 \ , & 0 \le t < \frac{1}{2} \\ -1, & \frac{1}{2} \le t < 1 \\ 0 \ , & \text{caso contrario} \end{cases} \tag{2.12}$$

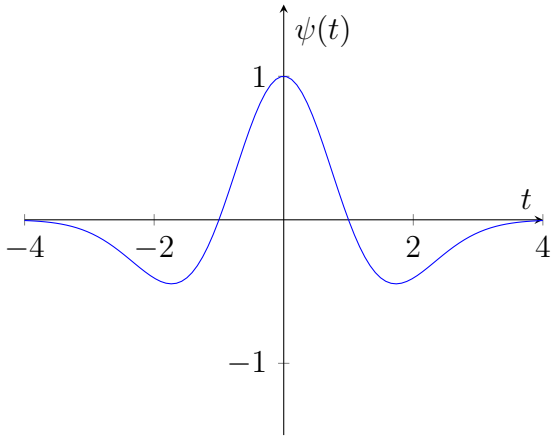


Figura 2.5: Gráfica del wavelet sombrero mejicano

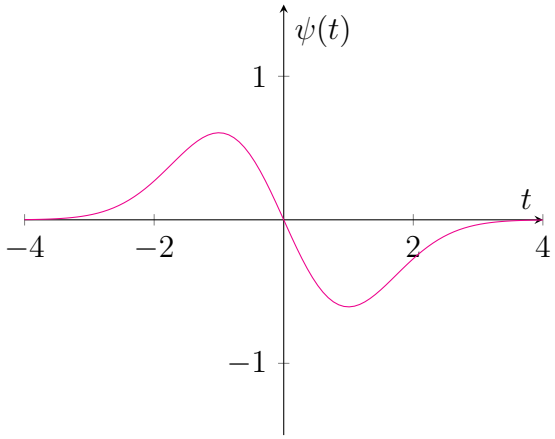


Figura 2.6: Gráfica del wavelet gaussiano

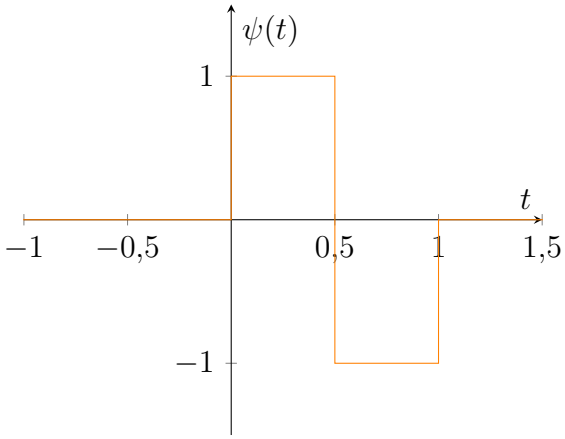


Figura 2.7: Gráfica del wavelet de Haar

2.4.1. Transformada continua de wavelets

Una vez elegido un wavelet, se desea poder expandirlo/comprimirlo en tiempo y poder desplazarlo en tiempo. Para ello, sean dos coeficientes a, coeficiente de expansión, y b, coeficiente de traslación, los cuales modificarán el wavelet de la manera $\psi(\frac{t-b}{a})$. Para el wavelet sombrero mejicano (Ecuación 2.10), esto equivale a:

$$\psi\left(\frac{t-b}{a}\right) = \left(1 - \left(\frac{t-b}{a}\right)^2\right) e^{-\frac{\left(\frac{t-b}{a}\right)^2}{2}}$$

Utilizando esto, se llega a que la transformada continua de wavelets (usualmente abreviada CWT) de una función $x(t)$ es:

$$T(a,b) = w(a) \int_{-\infty}^{\infty} x(t)\psi^*\left(\frac{t-b}{a}\right) \mathrm{d}t$$

donde * denota el conjugado y $w(a)$ una función de compensación. Normalmente, se toma $w(a) = \frac{1}{\sqrt{a}}$ para asegurar que todos los wavelets tengan la misma energía para todo a. Esto deja la transformada continua de wavelets como:

$$T(a,b) = \frac{1}{\sqrt{a}} \int_{-\infty}^{\infty} x(t)\psi^*\left(\frac{t-b}{a}\right) \mathrm{d}t \tag{2.13}$$

Si llamamos al wavelet normalizado:

$$\psi_{a,b}(t) = \frac{1}{\sqrt{a}}\psi\left(\frac{t-b}{a}\right) \tag{2.14}$$

la transformada de wavelets finalmente queda como:

$$T(a,b) = \int_{-\infty}^{\infty} x(t)\psi^*_{a,b}(t)\ \mathrm{d}t \tag{2.15}$$

Tomando la noción de que el producto interno entre dos funciones es:

$$\langle f(t), g(t)\rangle = \int_{-\infty}^{\infty} f(t)g^*(t)\ \mathrm{d}t$$

La CWT es equivalente a:

$$\langle x(t), \psi_{a,b}(t)\rangle = \int_{-\infty}^{\infty} x(t)\psi^*_{a,b}(t)\ \mathrm{d}t \tag{2.16}$$

2.4.2. Transformada continua de wavelets inversa

Desde la función transformada, se obtiene la función original de la siguiente manera [Addison, 2017, p.24]:

$$x(t) = \frac{1}{C_g}\int_{-\infty}^{\infty}\int_{0}^{\infty}\frac{1}{a^2}T(a,b)\psi^*_{a,b}(t)\ \mathrm{d}a\ \mathrm{d}b \tag{2.17}$$

2.4.3. Transformada discreta de wavelets para funciones continuas

2.4.3.1. Discretización de wavelets

Para pasar al tiempo discreto, tenemos que discretizar la función del wavelet normalizado (ecuación 2.14).

Una forma consiste en discretizar los parámetros a y b de manera logarítmica, vinculando a a la distancia entre b valores. Este tipo de discretización nos permite obtener el siguiente wavelet:

$$\psi_{m,n}(t) = \frac{1}{\sqrt{a_0^m}}\psi\left(\frac{t - nb_0a_0^m}{a_0^m}\right) \tag{2.18}$$

donde n controla la traslación, m controla la expansión, a_0 es un valor de expansión fijo mayor a 1, y b_0 es un valor de traslación mayor que 0.

Reemplazando $\psi_{m,n}(t)$ a transformada discreta de wavelets (DWT) de una señal continua $x(t)$ queda como:

$$T_{m,n} = \int_{-\infty}^{\infty} x(t) \frac{1}{\sqrt{a_0^m}} \psi\left(\frac{t - nb_0a_0^m}{a_0^m}\right) \mathrm{d}t \tag{2.19}$$

donde $T_{m,n}$ son los valores de la DWT dados en el índice (m, n) de una matriz. Los valores $T_{m,n}$ se denominan *coeficientes de wavelet* o *coeficientes de detalle*.

Para determinar la correctitud de la representación del wavelet con esta discretización, se utiliza la teoría de *wavelet frames*, los cuales son construídos muestreando los parámetros de tiempo y escala del wavelet, como se hizo anteriormente. Se determina la *familia de wavelets* que componen un frame a aquellos tal que tienen energía de sus coeficientes acotada respecto al wavelet original, es decir:

$$AE \leq \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} |T_{m,n}|^2 \leq BE \tag{2.20}$$

donde A y B son los límites del frame y E es la energía del wavelet original, es decir (para una función $x(t)$):

$$\int_{-\infty}^{\infty} |x(t)|^2 \, \mathrm{d}t = ||x(t)||^2$$

La ecuación anterior tiene sentido pues los wavelets deben tener energía finita.

Los valores de A y B serán dependientes de la elección de a_0 y b_0 [Daubechies, 1999].

Si $A = B$, se define el frame como *ajustado*. Si un frame es ajustado, entonces se puede reconstruir la señal original $x(t)$ de la siguiente manera:

$$x(t) = \frac{1}{A} \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} T_{m,n}\psi_{m,n}(t) \tag{2.21}$$

Un frame ajustado con $A = B > 1$ es redundante, donde A es el orden de redundancia. Si en cambio $A = B = 1$, la familia de wavelets forma una base ortonormal.

Si $A \neq B$, se puede aproximar la señal original con la fórmula

$$x'(t) = \frac{2}{A+B} \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} T_{m,n}\psi_{m,n}(t) \tag{2.22}$$

la cual tendrá un error proporcional al ratio $\frac{B}{A}$. Mientras más cercano sea éste a 1, más ajustado será el frame y más exacta será la aproximación.

2.4.3.2. Wavelets diádicos

Para la discretización, una propiedad deseable es que los cambios de expansión sean potencia de 2. Esto se puede conseguir tomando $a_0 = 2$ y $b_0 = 1$. Para estos valores, el wavelet queda:

$$\psi_{m,n}(t) = \frac{1}{\sqrt{2^m}}\psi\left(\frac{t - n2^m}{2^m}\right) \tag{2.23}$$

Este tipo de wavelets se denominan *diádicos* [Addison, 2017].

Otra propiedad deseable de un wavelet diádico es que sea ortonormal. Es decir:

$$\int_{-\infty}^{\infty} \psi_{m,n}(t)\psi_{m',n'}(t)\ \mathrm{d}t = \begin{cases} 1, & m = m' \text{ y } n = n' \\ 0, & \text{caso contrario} \end{cases}$$

La DWT de una señal continua $x(t)$ queda entonces:

$$T_{m,n} = \int_{-\infty}^{\infty} x(t)\psi_{m,n}(t)\ \mathrm{d}t \tag{2.24}$$

Ya que la familia de wavelets es ortonormal, por lo visto en la sección anterior, $A = B = 1$ y se puede reconstruir la señal de manera exacta como:

$$x(t) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} T_{m,n}\psi_{m,n}(t) \tag{2.25a}$$

Esta es la *transformada inversa de wavelets discreta*.

Recordando que la DWT puede ser vista como un producto interno, nos queda que:

$$x(t) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} \langle x(t), \psi_{m,n}(t)\rangle\psi_{m,n}(t) \tag{2.25b}$$

Usando que $A = B = 1$ podemos determinar que la energía de la señal es:

$$\int_{-\infty}^{\infty} |x(t)|^2 \, \mathrm{d}t = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} |T_{m,n}|^2$$

2.4.3.3. Función de escala

Los wavelets diádicos ortonormales tienen asociada una *función de escala* definida como:

$$\phi_{m,n}(t) = \frac{1}{\sqrt{2^m}} \phi\left(\frac{t}{2^m} - n\right) \tag{2.26}$$

Esta función tiene la misma forma que el wavelet. A su vez, cumple la siguiente propiedad:

$$\int_{-\infty}^{\infty} \phi_{0,0}(t) \, \mathrm{d}t = 1$$

donde a $\phi_{0,0}(t) = \phi(t)$ se lo suele llamar *función de escala* o *wavelet padre*. Esta función es ortogonal a traslaciones de si misma, pero no a expansiones de si misma.

Convolucionando la función de escala con la señal, se pueden generar *coeficientes de aproximación* de la siguiente manera:

$$S_{m,n} = \int_{-\infty}^{\infty} x(t) \phi_{m,n}(t) \, \mathrm{d}t \tag{2.27}$$

Es decir, los coeficientes de aproximación son medias ponderadas de la función factorizadas por $\sqrt{2^m}$.

El conjunto de coeficientes de aproximación para un m_0 fijo se denominan *aproximación discreta* de la función con escala m_0. Para generar una aproximación continua con escala m_0, se suma una secuencia de funciones de escala multiplicada por coeficientes de aproximación, de la siguiente manera:

$$x_{m_0}(t) = \sum_{n=-\infty}^{\infty} S_{m_0,n} \phi_{m_0,n}(t)$$

En base a esto, se puede generar una representación de $x(t)$ usando los coeficientes de wavelet (ecuación 2.19) y los coeficientes de aproximación:

$$x(t) = \sum_{n=-\infty}^{\infty} S_{m_0,n}\phi_{m_0,n}(t) + \sum_{m=-\infty}^{m_0}\sum_{n=-\infty}^{\infty} T_{m,n}\psi_{m,n}(t) \tag{2.28}$$

Si definimos los *detalles de señal* para una escala m

$$d_m(t) = \sum_{n=-\infty}^{\infty} T_{m,n}\psi_{m,n}(t) \tag{2.29}$$

La ecuación anterior nos queda:

$$x(t) = x_{m_0}(t) + \sum_{m=-\infty}^{m_0} d_m(t) \tag{2.30}$$

En base a esta ecuación, se puede demostrar que:

$$x_{m-1}(t) = x_m + d_m(t) \tag{2.31}$$

lo cual indica que si aproximamos una señal a un índice m_0 y le sumamos su detalle de señal, se obtiene una aproximación de mayor resolución (es decir, menor m [Addison, 2017, p.100]). A esto se lo llama *representación multiresolución.*

2.4.3.4. Relación entre la función de wavelets y la función de escala

Se define la *ecuación de escala* [Addison, 2017, p.101] como una relación de la función de escala $\phi(t)$ calculada en base a expansiones y traslaciones de sí misma, definida como:

$$\phi(t) = \sum_k c_k\phi(2t-k) \tag{2.32}$$

es decir, la suma de funciones de escala trasladadas una distancia entera k y ponderadas por *coeficientes de escala* c_k. Esto implica que se puede obtener la función de escala $\phi(t)$ solucionando un sistema de ecuaciones en diferencias. Integrando ambos lados de la igualdad, queda:

$$\sum_k c_k = 2 \tag{2.33}$$

Si además se desea que el sistema sea ortonormal, se debe añadir la condición

$$\sum_k c_k c_{k+2k'} = \begin{cases} 2, & k' = 0 \\ 0, & \text{caso contrario} \end{cases} \tag{2.34}$$

Esto es equivalente a decir que:

$$\sum_k c_k^2 = 2 \tag{2.35}$$

Estos coeficientes se utilizan para la reconstrucción del wavelet de la siguiente manera:

$$\psi(t) = \sum_k (-1)^k c_{1-k} \phi(2t - k) \tag{2.36}$$

Si asumimos que tenemos un número finito de coeficientes de escala N_k, la ecuación anterior es equivalente a:

$$\psi(t) = \sum_k (-1)^k c_{N_k - 1 - k} \phi(2t - k)$$

Tomando b_k como:

$$b_k = (-1)^k c_{N_k - 1 - k} \tag{2.37}$$

la ecuación anterior queda:

$$\psi(t) = \sum_k b_k \phi(2t - k) \tag{2.38}$$

Tomando las ecuaciones 2.26 y 2.32 para un índice de escala $m + 1$, se puede demostrar que la siguiente igualdad es válida:

$$\frac{1}{\sqrt{2^{m+1}}} \phi\left(\frac{t}{2^{m+1}} - n\right) = \frac{1}{\sqrt{2}\sqrt{2^m}} \sum_k c_k \phi\left(\frac{2t}{2 \cdot 2^m} - (2n + k)\right) \tag{2.39}$$

Lo cual es equivalente a:

$$\phi_{m+1,n}(t) = \frac{1}{\sqrt{2}} \sum_k c_k \phi_{m,2n+k}(t) \tag{2.40}$$

De manera similar, se obtiene que:

$$\psi_{m+1,n}(t) = \frac{1}{\sqrt{2}} \sum_k b_k \phi_{m,2n+k}(t) \tag{2.41}$$

2.4.3.5. *Fast Wavelet Transform*

En base a la ecuacion 2.27, podemos determinar que los coeficientes de aproximación para un índice de escala $m+1$ es:

$$S_{m+1,n} = \int_{-\infty}^{\infty} x(t)\phi_{m+1,n}(t)\ \mathrm{d}t$$

Usando la ecuación 2.40 esto equivale a:

$$S_{m+1,n} = \int_{-\infty}^{\infty} x(t)\left(\frac{1}{\sqrt{2}}\sum_k c_k\phi_{m,2n+k}(t)\right)\ \mathrm{d}t$$

Aplicando propiedades de la integral se llega a:

$$S_{m+1,n} = \frac{1}{\sqrt{2}}\sum_k c_k\left(\int_{-\infty}^{\infty} x(t)\phi_{m,2n+k}(t)\ \mathrm{d}t\right)$$

Esto es equivalente a:

$$S_{m+1,n} = \frac{1}{\sqrt{2}}\sum_k c_k S_{m,2n-k} \tag{2.42a}$$

Realizando un cambio de variable se llega a que:

$$S_{m+1,n} = \frac{1}{\sqrt{2}}\sum_k c_{k-2n} S_{m,n} \tag{2.42b}$$

Similarmente, se puede llegar a que

$$T_{m+1,n} = \frac{1}{\sqrt{2}}\sum_k b_k S_{m,2n-k} \tag{2.43a}$$

Lo que es equivalente a

$$T_{m+1,n} = \frac{1}{\sqrt{2}}\sum_k b_{k-2n} S_{m,n} \tag{2.43b}$$

Estas ecuaciones nos permiten determinar, conociendo todos los coeficientes $S_{m_0,n}$ para un m_0 dado, los coeficientes de aproximación y detalle para todo m mayor que m_0. Estas ecuaciones se denominan *ecuaciones de descomposición* y representan la primera parte de la Fast Wavelet Transform (FWT).

Nótese como estas ecuaciones nos permiten calcular los coeficientes sin necesidad de conocer la forma de $x(t)$, basta con los $S_{m_0,n}$; y permite calcularlos sin necesidad de calcular una convolución (como en la ecuación 2.24).

Nótese también como la ecuación 2.42 equivale a un *filtro pasa bajos* ya que deja pasar bajas frecuencias de la señal, es decir, una versión suavizada de la señal. Adicionalmente, la ecuación 2.43 equivale a un *filtro pasa altos* ya que deja pasar altas frecuencias de la señal, es decir, los detalles de la señal.

Ahora, se desea realizar el camino inverso, es decir, reconstruir $S_{m,n}$ a partir de $S_{m+1,n}$ y $T_{m+1,n}$. Partiendo de la ecuación 2.31 y expandiéndola, se obtiene:

$$x_{m-1}(t) = \sum_n S_{m,n}\phi_{m,n}(t) - \sum_n T_{m,n}\psi_{m,n}(t) \tag{2.44}$$

Usando las ecuaciones 2.40 y 2.41 se obtiene que

$$x_{m-1}(t) = \sum_n S_{m,n}\frac{1}{\sqrt{2}}\sum_k c_k\phi_{m-1,2n+k}(t) - \sum_n T_{m,n}\frac{1}{\sqrt{2}}\sum_k b_k\phi_{m-1,2n+k}(t) \tag{2.45}$$

Realizando un cambio de variables, se obtiene

$$x_{m-1}(t) = \sum_n S_{m,n}\frac{1}{\sqrt{2}}\sum_k c_{k-2n}\phi_{m-1,n}(t) - \sum_n T_{m,n}\frac{1}{\sqrt{2}}\sum_k b_{k-2n}\phi_{m-1,n}(t) \tag{2.46}$$

Recordando que $x_{m-1}(t)$ equivale:

$$x_{m-1}(t) = \sum_n S_{m-1,n}\phi_{m-1,n}(t)$$

se puede llegar a que (luego de intercambiar n y k en la ecuación 2.45 y algunos pasos de álgebra):

$$S_{m-1,n} = \frac{1}{\sqrt{2}}\sum_k c_{n-2k}S_{m,k} + \frac{1}{\sqrt{2}}\sum_k b_{n-2k}T_{m,k} \tag{2.47}$$

Esta ecuación se denomina *algoritmo de reconstrucción* y compone la segunda parte de la FWT.

2.4.4. Transformada discreta de wavelets para funciones discretas finitas

A lo largo de la sección anterior, se derivaron todos los calculos para una señal continua $x(t)$, donde se mostró que una señal puede ser expresada como una serie de wavelets para todas las expansiones y traslaciones (ecuación 2.25b) o una expansión que incluye las funciones de wavelet y de escala (ecuación 2.28). Para obtener una discretización multiresolución, se necesita que la entrada de la transformada sean muestras de la señal para una escala $m = 0$, es decir:

$$S_{0,n} = \int_{-\infty}^{\infty} x(t)\phi(t-n)\ \mathrm{d}t \tag{2.48}$$

Usando la primera parte de la FWT, esto nos permitirá generar todos los $S_{m,n}$ y $T_{m,n}$, con $m > 0$. Se va a asumir que se tiene una señal $S_{0,n}$ de entrada de longitud finita $N = 2^M$. Es decir, se requiere realizar los cálculos para escalas m tal que $0 < m < M$.

2.4.4.1. Ejemplo práctico: Wavelet de Haar

El *wavelet de Haar* es el wavelet ortonormal más sencillo. Su función de escala tiene sólo dos coeficientes distintos de 0, su ecuación es:

$$\phi(t) = \phi(2t) + \phi(2t-1) \tag{2.49}$$

es decir, sus coeficientes son $c_0 = c_1 = 1$. La solución a la ecuación anterior está dado por la función:

$$\phi(t) = \begin{cases} 1 \ , & 0 \leq t < 1 \\ -1, & \text{caso contrario} \end{cases} \tag{2.50}$$

Utilizando la ecuación 2.37 se obtiene que $b_0 = 1$ y $b_1 = -1$, luego la ecuación queda:

$$\psi(t) = \phi(2t) - \phi(2t-1) \tag{2.51}$$

La solución de esta ecuación es el wavelet de Haar:

$$\psi(t) = \begin{cases} 1 \ , & 0 \leq t < \frac{1}{2} \\ -1, & \frac{1}{2} \leq t < 1 \\ 0 \ , & \text{caso contrario} \end{cases} \tag{2.52}$$

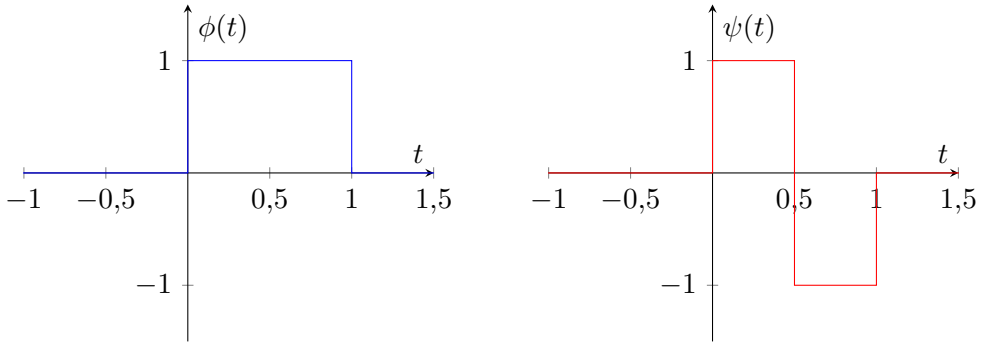


Figura 2.8: Función de escala $\phi(t)$ y función de wavelet $\psi(t)$

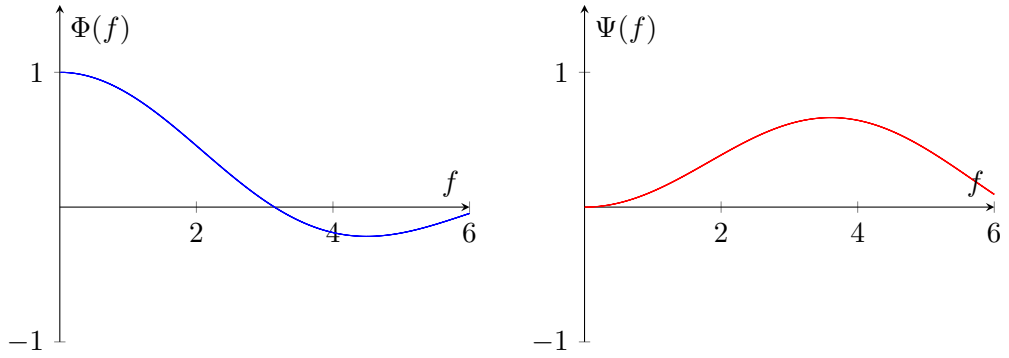


Figura 2.9: Espectro en frecuencia de $\phi(t)$ y $\psi(t)$

En base a los coeficientes de escala obtenidos, la fórmula para obtener los coeficientes de aproximación queda:

$$S_{m+1,n} = \frac{1}{\sqrt{2}}(S_{m,2n} + S_{m,2n+1}) \tag{2.53}$$

y los coeficientes de detalle quedan:

$$T_{m+1,n} = \frac{1}{\sqrt{2}}(S_{m,2n} - S_{m,2n+1}) \tag{2.54}$$

Se mostrará una aplicación de las fórmulas obtenidas con un ejemplo: se desea calcular la descomposición con wavelets de Haar del vector $\mathbf{W}^{(0)} = (1, 2, 3, 4)$. La primera aproximación $\mathbf{W}^{(1)} = (S_{1,0}, S_{1,1}, T_{1,0}, T_{1,1})$ será:

$$S_{1,0} = \frac{1+2}{\sqrt{2}} = \frac{3}{\sqrt{2}} \qquad S_{1,1} = \frac{3+4}{\sqrt{2}} = \frac{7}{\sqrt{2}}$$

$$T_{1,0} = \frac{1-2}{\sqrt{2}} = -\frac{1}{\sqrt{2}} \qquad T_{1,1} = \frac{3-4}{\sqrt{2}} = -\frac{1}{\sqrt{2}}$$

Es decir, $\mathbf{W}^{(1)} = (\frac{3}{\sqrt{2}}, \frac{7}{\sqrt{2}}, -\frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}})$. La siguiente aproximación $\mathbf{W}^{(2)} = (S_{2,0}, T_{2,0}, T_{1,0}, T_{1,1})$ tendrá:

$$S_{2,0} = \frac{\frac{3}{\sqrt{2}} + \frac{7}{\sqrt{2}}}{\sqrt{2}} = 5 \qquad T_{2,0} = \frac{\frac{3}{\sqrt{2}} - \frac{7}{\sqrt{2}}}{\sqrt{2}} = -2$$

Es decir, $\mathbf{W}^{(2)} = (5, -2, -\frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}})$. La media de la señal puede ser encontrada realizando una iteración sobre $S_{2,0}$, es decir,

$$\bar{x} = \frac{S_{2,0}}{(\sqrt{2})^2} = 5$$

Para realizar la transformada inversa, se tiene que:

$$S_{m,2n} = \frac{S_{m+1,n} + T_{m+1,n}}{\sqrt{2}} \qquad S_{m,2n+1} = \frac{S_{m+1,n} - T_{m+1,n}}{\sqrt{2}}$$

Aplicando estas ecuaciones a $\mathbf{W}^{(2)} = (S_{2,0}, T_{2,0}, T_{1,0}, T_{1,1}) = (5, -2, -\frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}})$ queda que:

$$S_{1,0} = \frac{5 + (-2)}{\sqrt{2}} = \frac{3}{\sqrt{2}} \qquad S_{1,1} = \frac{5 - (-2)}{\sqrt{2}} = \frac{7}{\sqrt{2}}$$

Con lo cual se obtiene $\mathbf{W}^{(1)} = (\frac{3}{\sqrt{2}}, \frac{7}{\sqrt{2}}, -\frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}})$. Realizando un paso más, se obtiene que:

$$S_{0,0} = \frac{\frac{3}{\sqrt{2}} + (-\frac{1}{\sqrt{2}})}{\sqrt{2}} = 1 \qquad S_{0,1} = \frac{\frac{3}{\sqrt{2}} - (-\frac{1}{\sqrt{2}})}{\sqrt{2}} = 2$$

$$S_{0,2} = \frac{\frac{7}{\sqrt{2}} + (-\frac{1}{\sqrt{2}})}{\sqrt{2}} = 3 \qquad S_{0,3} = \frac{\frac{7}{\sqrt{2}} - (-\frac{1}{\sqrt{2}})}{\sqrt{2}} = 4$$

Es decir, $\mathbf{W}^{(0)} = (1, 2, 3, 4)$, recuperando así la señal original.

2.4.5. Wavelets de Daubechies

Como se vio en el caso de los wavelets de Haar, los coeficientes se ordenan en dos secuencias: una de suavizado de la entrada, y otra que extra los detalles de escala. En esta sección se verá una familia de wavelets donde el wavelet de Haar es el más simple: los *wavelets de Daubechies*. Las funciones de escala

cumplen todas las condiciones de ortonormalidad vista anteriormente, pero estos wavelets satisfacen una condición adicional: que la función de escala tenga cierto grado de suavizado dado por una función de momentos [Addison, 2017]. Esta condición puede ser expresada en término de los coeficientes como:

$$\sum_{k=0}^{N_k-1} (-1)^k c_k k^m = 0 \tag{2.55}$$

para todo $m = 0, 1, \ldots, \frac{N_k}{2} - 1$, donde N_k es la cantidad de coeficientes de escala.

El wavelet de Haar puede ser visto como el wavelet de Daubechies con 2 coeficientes (usualmente se denota D2). Se mostrará ahora la derivación para 4 coeficientes, D4. Utilizando la ecuación 2.32, la función de escala es:

$$\phi(t) = c_0\phi(2t) + c_1\phi(2t-1) + c_2\phi(2t-2) + c_3\phi(2t-3) \tag{2.56}$$

y de la ecuación 2.36, la función de wavelet es:

$$\psi(t) = c_3\phi(2t) - c_2\phi(2t-1) + c_1\phi(2t-2) - c_0\phi(2t-3) \tag{2.57}$$

Para encontrar los valores de los coeficientes, se realiza un sistema de ecuaciones. Usando la ecuación 2.33 se obtiene:

$$c_0 + c_1 + c_2 + c_3 = 2 \tag{2.58}$$

Por la ecuación 2.35 se obtiene:

$$c_0^2 + c_1^2 + c_2^2 + c_3^2 = 2 \tag{2.59}$$

Usando la ecuación 2.55 para $m = 0$:

$$c_0 - c_1 + c_2 - c_3 = 0 \tag{2.60}$$

Y usando la misma para $m = 1$:

$$-c_1 + 2c_2 - 3c_3 = 0 \tag{2.61}$$

Este sistema de ecuaciones tiene 2 posibles soluciones:

$$c_0 = \frac{1+\sqrt{3}}{4} \qquad c_1 = \frac{3+\sqrt{3}}{4} \qquad c_2 = \frac{3-\sqrt{3}}{4} \qquad c_3 = \frac{1-\sqrt{3}}{4}$$

la cual da $\phi(t)$, y

$$c_0 = \frac{1-\sqrt{3}}{4} \qquad c_1 = \frac{3-\sqrt{3}}{4} \qquad c_2 = \frac{3+\sqrt{3}}{4} \qquad c_3 = \frac{1+\sqrt{3}}{4}$$

la cual da $\phi(-t)$. Para el resto de este documento, se utilizará la primera solución.

Usando la ecuación 2.56 con los coeficientes anteriores, se puede aproximar la función de escala de la siguiente manera:

$$\phi_j(t) = c_0\phi_{j-1}(2t) + c_1\phi_{j-1}(2t-1) + c_2\phi_{j-1}(2t-2) + c_3\phi_{j-1}(2t-3) \quad (2.62)$$

La función de escala tiene aproximadamente la forma de la figura 2.10. Su espectro en frecuencia viene dado por la figura 2.11.

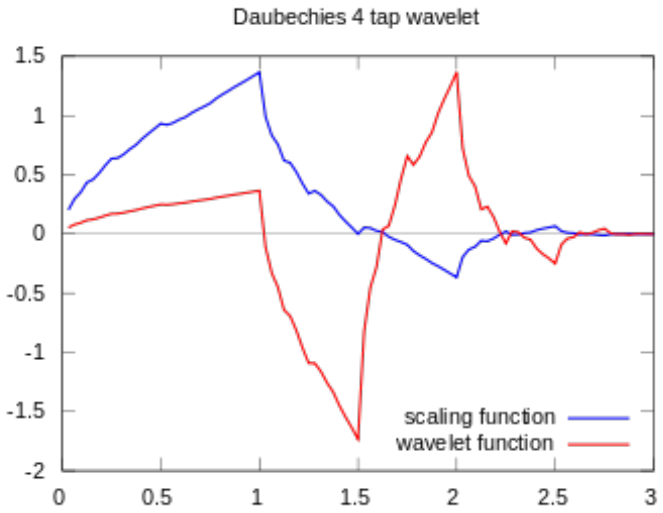


Figura 2.10: Aproximación de la funcion de escala de 4 coeficientes

Sin embargo, por lo visto anteriormente, no hace falta calcular la función de escala para calcular la transformada; basta con utilizar el algoritmo multiresolución con los coeficientes obtenidos.

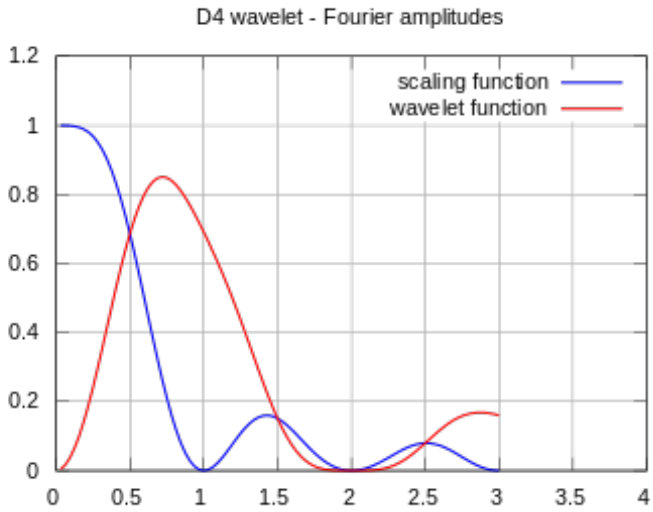


Figura 2.11: Aproximación de la funcion de escala de 4 coeficientes

Capítulo 3

Desarrollo de la solución. Tecnologías adoptadas. Arquitectura

Antes de realizar una implementación del proyecto, es necesario realizar algunos pasos previos.

Primero, se deben detallar y especificar los requerimientos del proyecto. Esto permite establecer un "acuerdo" de qué debe realizar el software, para proveer una referencia para su posterior verificación, y para reducir costos. En segunda instancia, sobre los requerimientos establecidos, se debe realizar un diseño de la arquitectura del software, es decir, la definición de la organización y estructura del mismo. Esta etapa es importante ya que actúa como "puente" entre la especificación de requerimientos y la etapa previa final, que consiste en realizar un diseño de la implementación basado en la arquitectura previamente diseñada.

3.1. Requerimientos de software

Los requerimientos de un sistema dan una descripción que debería hacer dicho sistema. El proceso de identificación, análisis, documentación y verificación de los requerimientos se denomina *ingeniería de requerimientos*. Normalmente se desea que una especificación de requerimientos sea: [IEEE, 1998]

- Correcta: Todo requerimiento explicitado en el documento representa algo requerido en el sistema.

- Inambigua: Toda línea escrita en el documento puede ser interpretada de una y sólo una manera.
- Completa: Todo lo que el sistema debe poder realizar y las repuestas a toda clase de datos de entrada están especificadas en el documento. [Jalote, 2008]
- Consistente: Ningún requerimiento debe entrar en conflicto con otro.
- Todos los requerimientos deben tener especificada su importancia y/o estabilidad.
- Verificable: Todos los requerimientos especificados son verificables, es decir, para cada requerimiento existe algún procedimiento costo-eficiente de tiempo finito que permite verificar si el programa final cumple dicho requerimiento.
- Modificable: La estructura y estilo del documento debe permitir realizar modificaciones de manera fácil, completa y consistente, preservando su estructura y estilo.
- Trazable: El origen de cada requerimiento debe ser claro. A su vez, cada requerimiento debe ser fácil de referenciar posteriormente.

3.1.1. Especificación de los requerimientos de software

A continuación se especificarán los requerimientos del software, siguiendo los criterios explicitados en [European Space Agency, 1991, p. 1-19 - 1-28].

3.1.1.1. Descripción general

Perspectiva del producto La aplicación será un producto independiente y funcionará correctamente en sistemas LINUX.

Funciones del producto El producto tendrá dos funciones:

Función 1 Compresión de una imagen

Entrada Una imagen

Salida Una versión comprimida de la imagen.

Función 2 Descompresión de una imagen previamente comprimida con esta aplicación.

Entrada Una imagen comprimida por esta aplicación.

Salida Una imagen reconstruída a partir de la entrada.

3.1.1.2. Requerimientos específicos funcionales

Los requerimientos funcionales describen que debe realizar un sistema. [Sommerville, 2011]. Para este proyecto, los requerimientos funcionales son:

- El algoritmo debe funcionar con imágenes multiespectrales. Este requerimiento es **esencial**.
- El algoritmo debe poder realizarse únicamente en modo con pérdida. Este requerimiento es **condicional** [IEEE, 1998, p.7].
- Se debe utilizar la Transformada de Wavelets Discreta. Este requerimiento es **opcional**.

3.1.1.3. Requerimientos específicos no funcionales

Los requerimientos no funcionales son aquellos que no se refieren a la funcionalidad del producto en sí, si no a atributos del producto como, por ejemplo, confiabilidad, tiempo de respuesta y espacio en memoria [Sommerville, 2011]. Para este proyecto, los requerimientos no funcionales son:

- El producto debe ejecutarse en un tiempo menor a 10 minutos para imágenes de tamaño menor a 8192×8192 píxeles. Este requerimiento es **condicional**.

3.2. Arquitectura del software

Con los requerimientos ya establecidos, se realiza un diagrama que contenga los módulos a implementar. Se opta por realizar una implementación modular ya que es considerada una buena práctica de programacion, además de facilitar la implementación y el debugging [Jalote, 2008, p.122,123].

Los módulos deberán implementar las funciones necesarias para realizar la compresión y la descompresión. El diseño, el cual es un desarrollo propio, a desarrollar, necesitará de implementar la codificación Huffman, la transformada Daubechies-4, algún método de cuantización y algún método de lectura/escritura de imágenes. El diagrama de módulos está dado en la figura 3.1.

A continuación se describirá la funcionalidad esperada de cada módulo:

Main Este módulo será el módulo principal y el que se ejecutará primero cuando se ejecute el programa. Llama según la funcionalidad a los módulos **Compression** y **Decompression**.

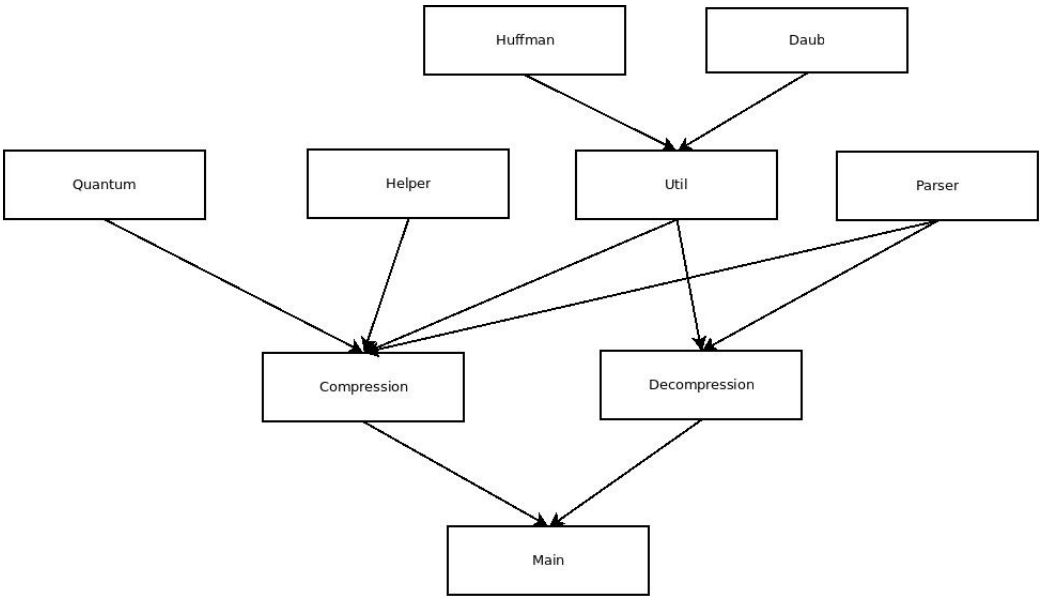


Figura 3.1: Diagrama de arquitectura

Compression Este módulo se encarga de realizar el procedimiento de compresión. Llama a los módulos **Parser**, **Helper**, **Util** y **Quantum**.

Decompression A diferencia del anterior, realiza la descompresión. Llama a los módulos **Parser** y **Util**.

Quantum Este módulo realiza la cuantización de la imagen. Dado que este paso sólo se realiza para la compresión, este módulo es usado sólo por el módulo **Compression**.

Util Este módulo tendrá funciones variadas que se refieren a la compresión por wavelets y a la codificación Huffman. Usada jerárquicamente por **Compression** y **Decompression**, y usa a **Huffman** y **Daub**.

Helper Este módulo tendrá funciones adicionales no referidas a algoritmos matemáticos. Usado sólo por el módulo **Compression**.

Parser Este módulo se encargará de leer la imagen desde el archivo original (cuando es utilizado por **Compression**), y de escribir el archivo de salida en formato de imagen (cuando es utilizado por **Decompression**).

Huffman Este módulo tendrá implementado la codificación Huffman vista anteriormente. Usado por el módulo **Util**.

Daub Este módulo tendrá implementado el cálculo de la transformada de wavelets discreta usando el wavelet Daubechies-4 visto anteriormente. Usado por el módulo **Util**.

3.3. Diseño de implementación

3.3.1. Compresión

Para realizar la compresión, se sigue el siguiente procedimiento:

1. Leer la imagen
2. Calcular la transformada de wavelets utilizando el wavelet Daubechies-4
3. Cuantizar el resultado (*sólo si se realiza compresión con pérdida*)
4. Aplicar la codificación Huffman
5. Escribir en un archivo binario el resultado.

El diagrama de secuencia de la compresión es el de la figura 3.2. Dicho diagrama corresponde al procedimiento mencionado anteriormente.

3.3.2. Descompresión

Para realizar la descompresión, se sigue el siguiente procedimiento:

1. Leer el archivo binario y extraer los datos codificados
2. Reconstruir la codificación Huffman y recuperar los datos originales.
3. Aplicar la transformada de wavelets inversa utilizando el wavelet Daubechies-4.
4. Escribir en formato de imagen el resultado.

El diagrama de secuencia de la compresión es el de la figura 3.3. Dicho diagrama corresponde al procedimiento explicitado anteriormente.

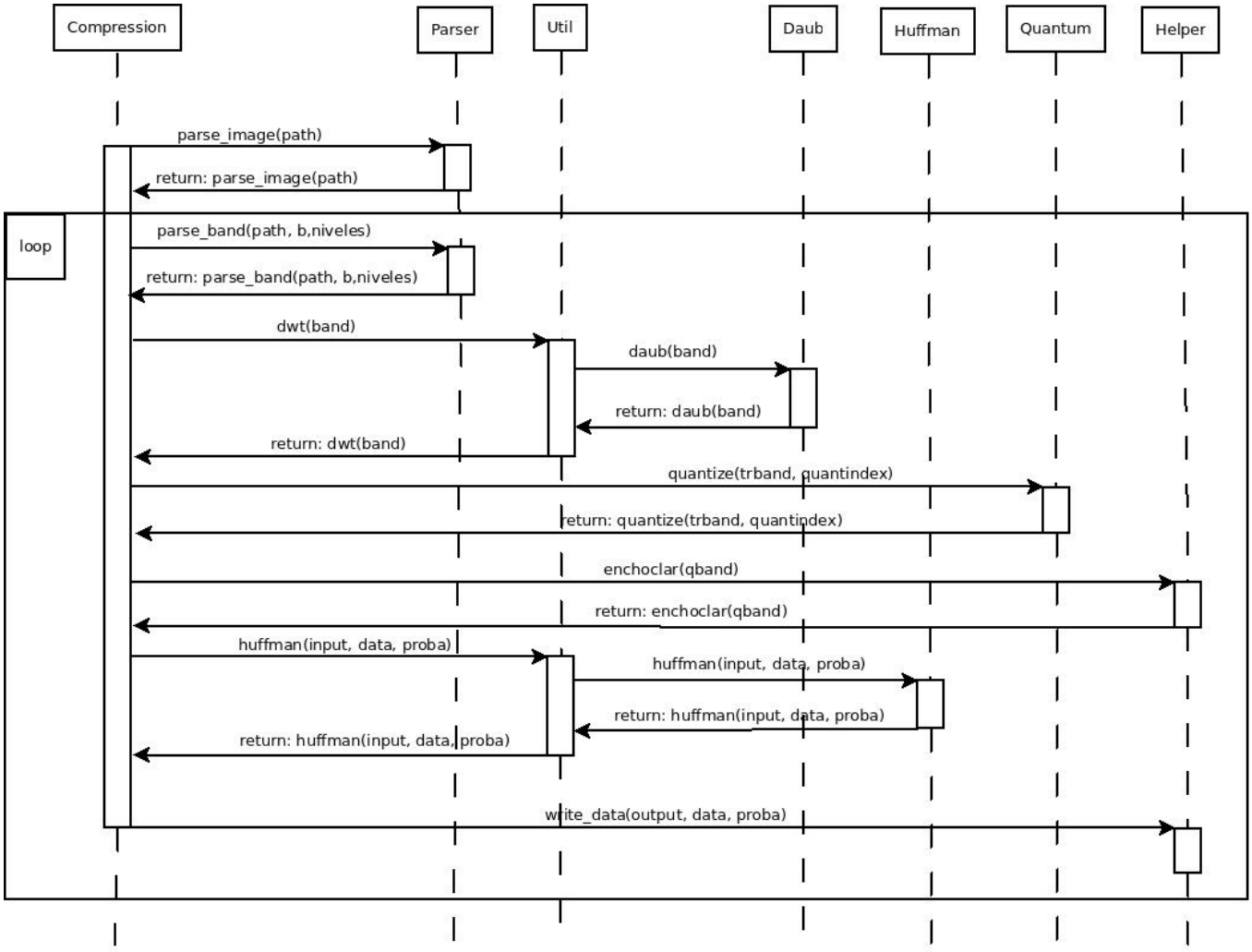


Figura 3.2: Diagrama de secuencia de la compresión

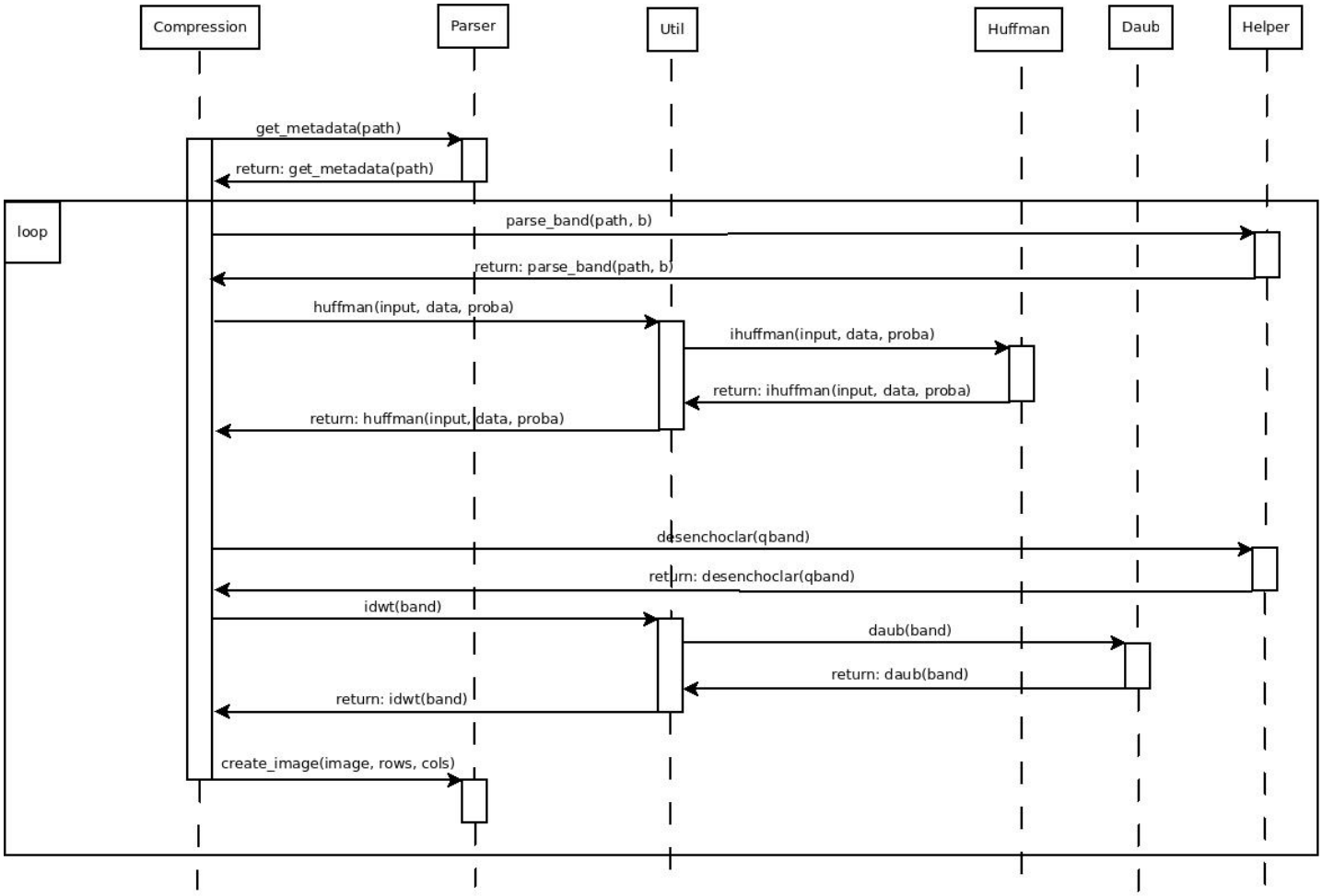


Figura 3.3: Diagrama de secuencia de la descompresión

Capítulo 4

Implementación

En el capítulo anterior se diseñó la solución al problema planteado para este proyecto. Este diseño se utiliza para la implementación del proyecto, la cual se abordará de manera general en este capítulo.

4.1. Herramientas utilizadas

Se utiliza el lenguaje C++ para la implementación de todo el proyecto. Para la compilación, se utiliza un archivo Makefile. Para la redacción de este documento, se utiliza LaTeX.

4.2. Pseudocódigo

El pseudocódigo es una descripción de alto nivel del funcionamiento de un algoritmo o un programa computacional. Se escribe utilizando convenciones estructurales de un lenguaje de programación real, pero utilizando también lenguaje natural, ya que esta diseñado para comprensión humana en lugar de comprensión para máquinas.

Los algoritmos 1 y 2 representan una implementación en pseudocódigo de la compresión y de la descompresión, respectivamente. Están fuertemente basados en los diagramas de secuencia vistos en el capítulo anterior.

En los pseudocódigos se mostrará que el algoritmo realiza el procedimiento de compresión/descompresión para cada banda individualmente.

Es importante notar que, para la compresión, el parámetro Q y la cantidad de niveles de la transformada deben ser inicializados de antemano. El uso de ambos parámetros se explicará más adelante.

Algorithm 1 Compresión

Data: Path a la imagen a comprimir
Result: Imagen comprimida
inicialización
parse_image(path)
for *b = 1 . . . bands* **do**
    band := parse_band(path, b, niveles)
    band := dwt(band)
    band := quantizar(band, Q)
    band := enchoclar(band)
    (elementos, probabilidades) := extraer elementos de *band* y calcular su probabilidad
    codificación := huffman(band, elementos, probabilidades)
    write_data(codificación)
end

Algorithm 2 Descompresión

Data: Path a la imagen comprimida a descomprimir
Result: Imagen original
inicialización
(codificacion, elementos, probabilidades) := get_metadata(path)
for *b = 1 . . . bands* **do**
    codificación := parse_binary_band(path, b)
    band := ihuffman(codificación, elementos, probabilidades)
    band := desenchoclar(band)
    band := idwt(band)
    create_image(band)
end

4.2.1. Cuantización

La *cuantización*, en el contexto de procesamiento de imágenes, es un procedimiento que reduce un rango de valores a un único valor o a un rango más pequeño.

La cuantización puede ser vista como el procedimiento que da pérdida a los algoritmos de compresión.

Para este algoritmo, el procedimiento de cuantización realiza un cambio selectivo dependiendo del valor del dato a cuantizar. El criterio de selección y la salida vienen dados por la función 4.1.

$$quant(x) = \begin{cases} 0 \quad , & |x| < Q \\ \lfloor x \rfloor, & Q \leq |x| < 4Q \\ x \quad , & 4Q \leq |x| \end{cases} \tag{4.1}$$

para algún $Q > 0$. Notar que $\lfloor \ \rfloor$ denota la función *piso*. Esta función permite eliminar los valores pequeños, reducir la precisión sin eliminar a los valores "medianos" y conservar los valores grandes.

La idea detrás de esta función reside en que, por lo general, los coeficientes de detalle suelen ser valores pequeños, mientras que los coeficientes de escala suelen ser más grandes.

Reducir el nivel de detalle de los coeficientes medianos permite incrementar la frecuencia de cada valor en sí, sin una gran pérdida de detalle de la imagen que sí se perdería en caso de eliminar dichos valores completamente. Esto, a su vez, mejora el resultado de la codificación Huffman, ya que habrá una menor cantidad de valores, con mayor repetitibilidad para cada uno.

Algorithm 3 Cuantización

Data: x, Q
Result: y
if $x < Q$ **then**
  $y := 0$
else if $Q \leq x < 4Q$ **then**
  $y := \lfloor x \rfloor$
else
  $y := x$

4.2.2. Util

El módulo Util, como se vio en el capítulo anterior, contiene las funciones que implementan la transformada de wavelets y la codificación Huffman. Posee cuatro funciones, dos para cada una de estas funcionalidades.

Estas funciones serán de transformación/codificación y transformación inversa/decodificación, respectivamente. Los pseudocódigos referidos a Huffman vienen dados por los algoritmos 4 y 5.

Algorithm 4 Codificación Huffman

```
Data: datos[M]: datos a codificar
Data: elementos[N]: lista con los elementos que aparecen en datos
Data: probabilidades[N]: probabilidad de cada elemento
Result: Cadena binaria
inicialización
arbol = elementos
for n = 1 ... N do
    insert(cola_de_prioridades, (elementos[i], probabilidades[i]))
end
for n = 1 ... N - 1 do
    x := front(cola_de_prioridades)
    y := front(cola_de_prioridades)
    crear un nuevo nodo z en arbol, padre de x y y
    z.probabilidad = x.probabilidad + y.probabilidad
    insert(cola_de_prioridades, (z, z.probabilidad))
end
for n = 2N-1 ... 1 do
    nodo := arbol[n];
    if nodo tiene hijos then
        asignar a x y y los hijos de nodo
        representacion[x] := representacion[nodo] + '0'
        representacion[y] := representacion[nodo] + '1'
    end
end
for m = 1 ... M do
    cadena := cadena + representacion[datos[m]]
end
return cadena
```

Algorithm 5 Decodificación Huffman

```
Data: cadena: cadena binaria codificada
Data: elementos[N]: lista con los elementos que aparecen en datos
Data: probabilidades[N]: probabilidad de cada elemento
Result: datos[M]: datos decodificados
inicialización
arbol = elementos
for n = 1 ... N do
    insert(cola_de_prioridades, (elementos[i], probabilidades[i]))
end
for n = 1 ... N - 1 do
    x := front(cola_de_prioridades)
    y := front(cola_de_prioridades)
    crear un nuevo nodo z en arbol, padre de x y y
    z.probabilidad = x.probabilidad + y.probabilidad
    insert(cola_de_prioridades, (z, z.probabilidad))
end
for n = 2N-1 ... 1 do
    nodo := arbol[n];
    if nodo tiene hijos then
        asignar a x y y los hijos de nodo
        representacion[x] := representacion[nodo] + '0'
        representacion[y] := representacion[nodo] + '1'
    end
end
candidato := ''
tamaño := 0
for c = 1 ... longitud(cadena) do
    candidato := candidato + cadena[c]
    if candidato esta en representacion then
        elem := elemento con representación candidato
        tamaño := tamaño + 1
        insert(datos, elem)
        candidato := ''
    end
end
return datos
```

4.2.2.1. Wavelets

A la hora de implementar la DWT, se presenta un problema. En todos los casos de la FDWT, se asumió que el vector a transformar es potencia de 2. Sin embargo, las imágenes satelitales no necesariamente tienen tamaños potencias de 2. Esto se puede compensar fácilmente extendiendo la imagen con ceros, para que llegue la longitud requerida.

Sin embargo, tomando esta idea, se puede realizar una implementación de la transformada por *niveles*. Este tipo de implementación realiza una variante en los cálculos de los $S_{m,n}$ y $T_{m,n}$: en lugar de calcular estos coeficientes para todos los m, calcula únicamente los $S_{k,n}$ y $T_{k,n}$ tal que $k < l < M$, donde l es la cantidad de niveles de la transformada. Esta implementación permite extender la imagen hasta el menor múltiplo de 2^l, en lugar de hasta 2^M, lo cual reduce el tamaño de la imagen extendida.

Es importante notar, que si $l = M$, se está extendiendo lo suficiente la imagen como para realizar el cálculo de todos los coeficientes. Si $l > M$ este paradigma carece de sentido y no se ejecutará la compresión.

El pseudocódigo de esta idea se muestra en los algoritmos 6 y 7.

Algorithm 6 Transformada de wavelets Daub-4

Data: datos[N][N]: datos a transformar
Data: niv: cantidad de niveles transformar
Result: transf[N][N]: datos transformados
inicialización
C0 := $\frac{1+\sqrt{3}}{4}$
C1 := $\frac{3+\sqrt{3}}{4}$
C2 := $\frac{3-\sqrt{3}}{4}$
C3 := $\frac{1-\sqrt{3}}{4}$
transf := datos
for each *fila* **in** *datos* **do**
  datos := calcular coeficientes sobre *fila* aplicando la *Fast Wavelet Transform* hasta el nivel *niv* y los coeficientes $C0 \dots C3$
end
for each *columna* **in** *datos* **do**
  datos := calcular coeficientes sobre *columna* aplicando la *Fast Wavelet Transform* hasta el nivel *niv* y los coeficientes $C0 \dots C3$
end
return *transf*

Algorithm 7 Transformada inversa de wavelets Daub-4

Data: transf[N][N]: datos transformados
Data: niv: cantidad de niveles transformar
Result: datos[N][N]: datos originales
inicialización
C0 := $\frac{1+\sqrt{3}}{4}$
C1 := $\frac{3+\sqrt{3}}{4}$
C2 := $\frac{3-\sqrt{3}}{4}$
C3 := $\frac{1-\sqrt{3}}{4}$
datos := transf
for each *fila* **in** *datos* **do**
  datos := calcular coeficientes sobre *fila* aplicando la *Fast Wavelet Transform* inversa desde el nivel *niv* y los coeficientes $C0 \dots C3$
end
for each *columna* **in** *datos* **do**
  datos := calcular coeficientes sobre *columna* aplicando la *Fast Wavelet Transform* inversa desde el nivel *niv* y los coeficientes $C0 \dots C3$
end
return *datos*

4.2.3. Código

El código implementado se encuentra disponible en `https://github.com/kouichicruz/compression-thesis`.

Capítulo 5

Verificación de la solución. Elección del set de imágenes de testeo. Análisis de los resultados

Luego de haber realizado la implementación del proyecto como se vio en el capítulo anterior, se debe realizar su correspondiente verificación para asegurarse de que la salida sea la esperada y cumpla con todos los requerimientos establecidos en el capítulo 3.

Una forma de realizar una verificación es realizar los procedimientos de compresión y descompresión para una serie de imágenes, las cuales serán seleccionadas siguiendo criterios que se verán en este capítulo. Para asegurar la correctitud del proyecto, se espera que haya una reducción del tamaño en disco de la imagen después de su compresión; y que se recupere la imagen original después de su descompresión.

5.1. Imágenes de test elegidas

Para las imágenes de testeo, se buscaron imágenes (satelitales *o no*) que cumplan con alguno de los siguientes criterios:

- La imagen es considerada estándar para la verificación. Esto quiere decir que la imagen ha sido utilizada para la verificación de otros algoritmos. Este tipo de imágenes dan la ventaja de que permiten obtener una comparación del programa a verificar con respecto a otros programas.

- La imagen proviene de algún satélite, donde dicha imagen tiene algunos elementos fácilmente apreciables a simple vista. Esto permite dar una apreciación instantánea de la correctitud de la imagen.

Existe una imágen estándar muy frecuentemente utilizada en algoritmos de compresión de imágenes: las imágenes Lena. Éstas vienen dadas en la figura 5.1.

(a) Lena (1 banda) (b) Lena (3 bandas)

Figura 5.1: Imágenes Lena

Estas imágenes de 512×512 píxeles se utilizan como un criterio de comparación entre múltiples algortimos de compresión de imágenes.

Si bien las imágenes Lena son consideradas estándar, no califican como imágenes satelitales. Para asegurar el correcto funcionamiento del algoritmo, se probará también con una imagen dada en la figura 5.2. Esta es una imagen de San Francisco procedente del satélite Landsat 8, de una resolución espacial de 15×15 metros, una resolución espectral de 3 bandas (bandas 4, 3, 2; color real) y un tamaño de 7204×7204 píxeles.

5.2. Resultados

Todas las imágenes anteriormente mostradas fueron comprimidas y descomprimidas con la implementación del proyecto. Las imágenes fueron comprimidas en modo con pérdida con un coeficiente de cuantización $Q = 16$.

5.2.1. Criterios a analizar

Para analizar la correctitud y la performance de la implementación del proyecto, se analizarán los siguientes dos criterios:

Figura 5.2: Imagen satelital (SF.tif)

- Relación de compresión. Ésta se define como [Salomon et al., 2010, p.12]:

$$factordecompresion = \frac{tamañodeimagenoriginal}{tamañodeimagencomprimida} \tag{5.1}$$

  Esta relación permite medir cuántas veces más pequeño es el archivo comprimido.

- Calidad de reconstrucción de la imagen. Al comprimirse en modo con pérdida, existen algunos datos que son aproximados en lugar de reconstruidos de manera exacta. Existen muchos métodos de medición, en este trabajo se utilizará la *relación señal-ruido de pico*, usualmente abreviada *PSNR* por sus siglas en inglés. La forma más fácil de calcular la PSNR es calculando el *error cuadrático medio* (usualmente abreviado *MSE*) [Salomon et al., 2010, p.463,464]. Para dos imágenes L y K de tamaño $m \times n$, donde L es la imagen original[1] y K es la imagen comprimida[2], el MSE se define como:

$$MSE = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} (L[i,j] - K[i,j])^2 \tag{5.2}$$

  Con esta definición, la PSNR equivale a:

$$PSNR = 10 \log_{10} \left(\frac{MAX_i^2}{MSE} \right) \tag{5.3}$$

[1] En análisis de señales, esto equivale a "señal sin ruido"
[2] En análisis de señales, esto equivale a "señal ruidosa"

$$PSNR = 20 \log_{10} \left(\frac{MAX_i}{\sqrt{MSE}} \right) \tag{5.4}$$

$$PSNR = 20 \log_{10}(MAX_i) - 10 \log_{10}(MSE) \tag{5.5}$$

donde MAX_i es el máximo valor posible de píxel de la imagen. Es decir, si se usan 8 bits, $MAX_i = 255$. Si se utilizan 16 bits, $MAX_i = 65535$. Si se utilizan B bits, $MAX_i = 2^B - 1$. Si se tienen b bandas, la PSNR será la media aritmética de las PSNR de cada banda.
Notar que mientras más alta sea la PSNR, menor será el MSE, lo cual indica una aproximación más exacta de la imagen original. En el caso de la compresión sin pérdida, $MSE = 0$, lo cual implicará que la PSNR será infinita.

5.2.2. Resultados obtenidos

Las relaciones de compresión y las PSNR para cada imagen se detallan en la tabla 5.1. Las imágenes descomprimidas se muestran en la figura 5.3.

Imagen	Tamaño original [bytes]	Tamaño comprimido [bytes]	Relación	PSNR [dB]
lena512.bmp	263222	88422	2.9768:1	35.7346
lena512color.tiff	786572	258094	3.0476:1	33.1350
SF.tif	155851845	80356664	1.9395:1	33.9852

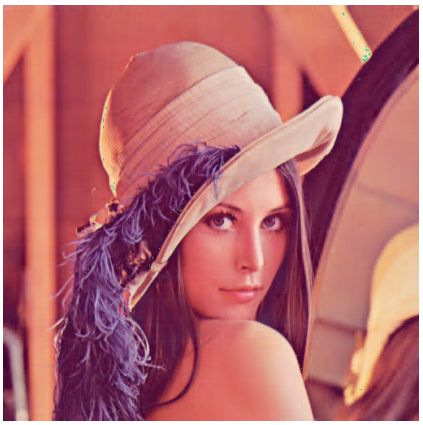
Cuadro 5.1: Resultados de relaciones de compresión y PSNR

A modo de comparación, se comprimió las imágenes con el compresor ZIP. El resultado obtenido para cada imagen se detalla en la tabla 5.2.

Imagen	Tamaño original [bytes]	Algoritmo desarrollado		Algoritmo ZIP	
		Tamaño comprimido [bytes]	Relación	Tamaño comprimido [bytes]	Relación
lena512.bmp	263222	88422	2.9768:1	212438	1.2391:1
lena512color.tiff	786572	258094	3.0476:1	715144	1.0999:1
SF.tif	155851845	80356664	1.9395:1	109197495	1.4272:1

Cuadro 5.2: Resultados comparados con ZIP

(a) Lena (1 banda)



(b) Lena (3 bandas)

(c) SF

Figura 5.3: Descompresión de las imágenes

Es importante aclarar que no se compararon las PSNR ya que ZIP es un algoritmo de compresión *sin* pérdida. Como se vio anteriormente, en los algoritmos sin pérdida la PSNR siempre será infinita.

5.2.3. Análisis de los resultados

Primero se analizará la correctitud de la reconstrucción de la imagen, es decir, la PSNR. Las PSNR usuales para algoritmos de compresión con pérdida en imágenes de 8 bits están entre 30 dB y 50 dB [Welstead, 1999, p.155,156]. Como se puede apreciar, la PSNR resultante del algoritmo del proyecto se mantiene dentro de este margen, indicando que tiene niveles de pérdida similares a otros algoritmos usados en la industria.

En segundo lugar, se analizará la relación de compresión obtenida. Si

bien se obtiene un nivel de compresión apreciable que supera algoritmos de compresión generales (en particular, al algoritmo ZIP), es significativamente menor al de otros algoritmos más sofisticados, tales como MrSID o ECW.

Capítulo 6

Conclusiones y trabajo futuro

6.1. Conclusiones

En esta tesis se desarrolló, diseñó, implementó y verificó un algoritmo de compresión de imágenes satelitales multiespectrales. Las imágenes son cada vez de mayor tamaño, y esa tendencia no parece que vaya a cambiar en los años que siguen. El objetivo de esta tesis fue generar dicho algoritmo para contrarrestar este problema.

Se realizó el diseño, la implementación y la verificación siguiendo patrones de ingeniería de software utilizados muy frecuentemente en el desarrollo de proyectos de este ámbito. Para la implementación, se utilizaron conceptos de algoritmos y estructuras de datos para realizar una implementación más eficiente. La compresión se realiza utilizando la transformada de wavelets discreta con el wavelet Daubechies-4, y la codificación Huffman. En el caso de la transformada de Wavelets, se eligió por uno de los wavelets más comunes en la actualidad, mientras que se eligió la codificación Huffman para codificar de manera mas efectiva los datos que resultan de la transformación de la imagen por medio de la DWT.

El resultado de esta tesis es un algoritmo funcional de compresión y descompresión con pérdida de imágenes. Este algoritmo posee niveles de pérdida similares a los algoritmos más usados en la industria. Ya que se generó un algoritmo menos sofisticado, se obtienen menores relaciones de compresión en comparación con los algoritmos líderes de la industria (tales como MrSID y ECW), pero que de todas maneras ofrece buenas relaciones de compresión. Las imágenes comprimidas por este método gozan de mayores relaciones de compresión con respecto a algoritmos de uso general como ZIP.

6.2. Trabajo futuro

Como trabajo futuro, se puede reemplazar tanto la codificación como la transformada matemática por alguna variante de mayor complejidad y mayor rendimiento.

Para la codificación, se puede utilizar, por ejemplo, codificación LZW [Welch, 1984] o Arithmetic Coding [MacKay, 2002]. Estos algoritmos se han utilizado en algunos formatos existentes en la actualidad, por ejemplo, GIF.

Para la transformada, se puede incrementar el número de coeficientes del wavelet de Daubechies. Se puede también utilizar otros wavelets, tales como *symlets* [Daubechies, 1999], los cuales son versiones simétricas de los wavelets de Daubechies, y *coiflets* [Daubechies, 1999], que tienen tanto para la función de escala como para la función de wavelet $\frac{N_k}{3}$ momentos.

Adicionalmente, la implementación de la transformada de wavelets 2D puede ser mejorada eligiendo otra descomposición del wavelet en lugar de la descomposición clásica. Una de las opciones más eficientes es la descomposición adaptativa [Meyer et al., 1999], el cual da resultados muy buenos en imágenes de tono continuo, pero el costo computacional de esta descomposición es significativamente mayor [Salomon et al., 2010].

Bibliografía

[Addison, 2017] Addison, P. S. (2017). *The Illustrated Wavelet Transform Handbook: Introductory Theory and Applications in Science, Engineering, Medicine and Finance*. CRC Press, 2nd edition.

[Chuvieco, 1995] Chuvieco, E. (1995). *Fundamentos de Teledetección Espacial*. Ediciones RIALP, 2 edition.

[Cormen et al., 2009] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2009). *Introduction to Algorithms*. The MIT Press, 3rd edition.

[Daubechies, 1999] Daubechies, I. (1999). *Ten Lectures On Wavelets*. CBMS-NSF Regional Conference Series in Applied Mathematics. SIAM: Society for Industrial and Applied Mathematics.

[European Space Agency, 1991] European Space Agency (1991). ESA Software Engineering Standards. Number no. 2 in ESA PSS-05-0 Issue 2.

[IEEE, 1998] IEEE (1998). IEEE 830-1998 Recommended Practice for Software Requirements Specifications.

[Jalote, 2008] Jalote, P. (2008). *A Concise Introduction to Software Engineering*. Undergraduate Topics in Computer Science. Springer-Verlag London, 1st edition.

[Jian Guo Liu, 2009] Jian Guo Liu, P. M. (2009). *Essential Image Processing and GIS for Remote Sensing*. Wiley, 1 edition.

[MacKay, 2002] MacKay, D. J. C. (2002). *Information Theory, Inference and Learning Algorithms*. Cambridge University Press, 1st edition.

[Meyer et al., 1999] Meyer, F. G., Averbuch, A., and Strömberg, J.-O. (1999). Fast Adaptive Wavelet Packet Image Compression.

[Oppenheim and Schafer, 2009] Oppenheim, A. V. and Schafer, R. W. (2009). *Discrete-Time Signal Processing*. Prentice-Hall Signal Processing Series. Prentice Hall, 3rd edition.

[Oppenheim et al., 1996] Oppenheim, A. V., Willsky, A. S., and with S. Hamid (1996). *Signals and Systems*. Prentice Hall, 2nd edition.

[Press et al., 2007] Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. (2007). *Numerical recipes: the art of scientific computing*. Cambridge University Press, 3rd edition.

[Salomon et al., 2010] Salomon, D., Motta, G., and Bryant, D. (2010). *Handbook of Data Compression*. Springer, 5th edition.

[Sommerville, 2011] Sommerville, I. (2011). *Software engineering*. Pearson, 9th edition.

[Ueffing, 2001] Ueffing, C. (2001). Wavelet based ECW image compression.

[Welch, 1984] Welch, T. (1984). A Technique for High-Performance Data Compression.

[Welstead, 1999] Welstead, S. (1999). *Fractal and Wavelet Image Compression Techniques*. SPIE Tutorial Texts in Optical Engineering Vol. TT40. SPIE Publications.

Aprobación del Tribunal

Los abajo firmantes, miembros del Tribunal de Evaluación de tesis, damos Fe que el presente ejemplar impreso, se corresponde con el aprobado por éste Tribunal.