UNIVERSIDAD AUTÓNOMA METROPOLITANA
UNIDAD AZCAPOTZALCO

División de Ciencias Básicas e Ingeniería

Proyecto de integración de Ingeniería en Computación

Modalidad: Proyecto tecnológico
Trimestre 2015 Primavera

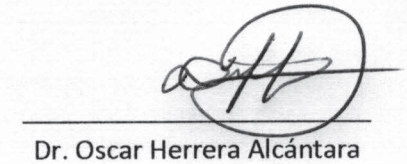
Aplicación de filtros digitales en la compresión de imágenes

Proyecto que presenta:
Moisés Chávez Pérez
Matricula:
206241606

Asesor:
Dr. Oscar Herrera Alcántara

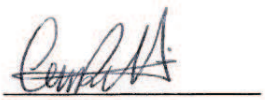
México, D.F. Septiembre, 2015

Yo, Oscar Herrera Alcántara, *declaro que aprobé el contenido del presente Reporte de Proyecto de Integración* y *doy mi autorización para su* publicación en la Biblioteca Digital, así como en el Repositorio Institucional de UAM Azcapotzalco.



Dr. Oscar Herrera Alcántara

Yo, Moisés Chávez Pérez*, doy mi autorización* a la Coordinación de Servicios de Información de la Universidad Autónoma Metropolitana, Unidad Azcapotzalco, para publicar el presente documento en la Biblioteca Digital, así como en el Repositorio Institucional de UAM Azcapotzalco.



Moisés Chávez Pérez

Tabla de contenido

1 Informacion general

1.1 Resumen

En este proyecto se mide la factibilidad al aplicar un filtro optimizado de una imagen pequeña en una imagen de mayor tamaño. La implemantacion de los algoritmos se realizo en el lenguaje de programación Java y las imágenes están en formato PGM en escala de grises.

Se aplica la Transformada Wavelet Discreata Haar sobre un conjunto de imágenes de 512x512 *pixeles* para generar otro conjunto de imágenes de 64x64 *pixeles*, para lo cual se implemento la Transformada Wavelet Discreta en dos dimensiones.
Se aplicó un algoritmo de Evolución Diferencial, que utiliza la Transformada Wavelet Discreta (aplicada sobre imágenes de 64x64 *pixeles*) como función de aptitud para optimizar los fitltros digitales.
Se aplica la Transformada Wavelet Discreta a imágenes de 512x512 *pixeles* utilizando filtros digitales optimizados cuyos parámetros fueron optimizados con imágenes de 64x64 *pixeles* y también filtros digitales fijos (Haar, Daubechies 4 y Daubechies 6) prara obtener el número de coeficientes irrelevantes que genera cada uno de los filtros empleados, y asi poder medir la factibilidad de aplicacar filtros digitales optimizados con imágenes pequeñas sobre imágenes de mayor tamaño.

1.2 Introducción

El procesamiento de imágenes se define como el mejoramiento de la imagen para aumentar su utilidad [1].
El procesamiento de imágenes tiene diversas aplicaciones como son el procesamiento de huellas digitales[1], el reconocimiento de rostros[2] y el procesamiento de imágenes satelitales[3], entre otras.
Dentro de las técnicas para el procesamiento de imágenes se encuentran la adquisición de la imagen, el realce de la imagen, la restauración de la imagen, el procesamiento de la imagen en color, el análisis multiresolución, la aplicación de procesos morfológicos, la segmentación, y la compresión de imágenes.

En particular, la compresión de imágenes se define como la manipulación de una representación original para obtener una versión equivalente que tenga una representación mínima. Un gran número de técnicas de compresión consideran a una imagen como una matriz bidimensional de pixeles M, en dondeM_{ij} es el valor del elemento de imageno *pixel*,

[1] Frecuentemente usado en dispositivos con pantallas táctiles.
[2] Por ejemplo, en aplicaciones de reconocimiento de rostros en redes sociales como Facebook.
[3] Google Maps ofrece la capacidad de realizar acercamientos y alejamientos para mostrar mapas.

en donde los índices i, j determinan la posición en la imagen completa. Los valores máximos de i, j definen al tamaño de la imagen que típicamente son llamadas "el ancho" y "lo alto" de la imagen respectivamente. A la matriz de *pixeles* se le aplica una trasformación lineal con el objetivo de que la representación obtenida tenga *entropía* mínima. La *entropía* de la imagen transformada se define como el valor esperado de una variable aleatoria, que en el caso de la compresión corresponde a la información $I(P_{ij})$ asociada a los coeficientes de transformación, los cuales siguen una distribución de probabilidad. Si un coeficiente de transformación tiene una probabilidad de ocurrencia P_{ij} entonces su información está dada por $I(P_{ij}) = \log_b \frac{1}{P_{ij}}$, en donde el valor esperado (*entropía* de los coeficientes de transformación) está dada por $E(I(Pij)) = \sum P_{ij} \log_b \frac{1}{P_{ij}}$ para cierto valor de b, específicamente si $b = 2$ la unidad de medida se denomina *bit*.

Es sabido que una imagen a procesar suele tener alta entropía, en particular si corresponden a imágenes fotográficas de la naturaleza, en contraste con las imágenes generadas por computadora.

Para imágenes en escala de grises con 256 niveles de gris la entropía máxima es 8, y la mínima es cero. La entropía máxima se logra cuando los pixeles (o coeficientes de transformación) siguen una distribución uniforme, y es mínima cuando la imagen tiene un único valor para todos los *pixeles* (o coeficientes de transformación).

Otro resultado interesante es que existe una estrecha relación entre la entropía de los coeficientes de transformación y el número de coeficientes de transformación irrelevantes, es decir, aquellos que pueden llevarse a cero sin que repercutan significativamente en la imagen reconstruida.

Aprovechando esto, en lugar de evaluar y minimizar la entropía se puede minimizar el número de coeficientes de transformación relevantes.

Para obtener una representación con entropía mínima suelen aplicarse transformaciones lineales. Algunas transformaciones son: la transformada Coseno Discreta, la Transformada Discreta de Fourier y la Transformada Wavelet Discreta. Estas transformadas poseen la propiedad de que los coeficientes de transformación tendrán una menor entropía, y por lo tanto facilitan su compresión.

Las transformadas anteriores tienen en común que usan funciones base de espacios vectoriales, similar a los vectores base i, j con los cuales se pueden expresar vectores en dos dimensiones. Las transformadas ya mencionadas usan funciones coseno, funciones seno, y funciones wavelet respectivamente.

La Transformada Wavelet Discreta para una señal $f(x)$ se define mediante la siguiente ecuación:

$$\sum_m \sum_n f(x) 2^{-\frac{m}{2}} \psi(2^{-m}x - n) \qquad (1.1)$$

que indica la proyección de la señal de entrada en el espacio wavelet respecto a las funciones $\psi(2^{-m}x - n)$ obtenidas mediante traslaciones en n y dilataciones en 2^{-k} de una función wavelet principal ψ.[2]

Uno de los resultados más sorprendentes sobre la Transformada Wavelet Discreta es que se pueden implementar usando los así llamados bancos de filtros de reconstrucción perfecta. Estos bancos de reconstrucción perfecta constan de un filtro pasabajas y un filtro pasaaltas. Se han propuesto varios filtros "estándar" como son los filtros *Daubechies*[2], *Coiflets*[3], *Symlets*[4] y *Curvelets[5]* por mencionar algunos.

1.3 Justificación

Típicamente, y para minimizar el costo computacional en la compresión de imágenes, se usan filtros "estándar" sin embargo, a manera de analogía, esto sería como "pensar que a todos nos queden los mismos zapatos". Alternativamente se plantea mejorar la compresión de imágenes al diseñar filtros específicos (para cada imagen). Para ello se propone usar filtros paramétricos de reconstrucción perfecta.

Los filtros paramétricos de reconstrucción perfecta tienen parámetros que al variar en cierto intervalo generan funciones wavelets con los cuales se puede implementar la Transformada Wavelet Discreta de imágenes. La correcta elección de los valores de estos parámetros permite maximizar la compresión, minimizar la entropía de los coeficientes de transformación, y minimizar el número de coeficientes de transformación relevantes[6][7][8].

Dado que la optimización de estos parámetros se complica para imágenes de gran tamaño (mayor a 512x512 *pixeles*) se plantea la posibilidad de optimizarlos en imágenes pequeñas (64x64 *pixeles*), y aplicarlos a imágenes más grandes (512x512 *pixeles*).
Con lo anterior se pueden generar filtros optimizados que ofrezcan mejores resultados en lugar de usar filtros "estándar".

1.4 Objetivo general

Realizar un estudio experimental para medir la factibilidad para aplicar un filtro optimizado de una imagen pequeña en una imagen de mayor tamaño.

1.5 Objetivos particulares

1. Identificar un conjunto de imágenes de 512x512 *pixeles* en escalas de grises que puedan reducirse a imágenes de 64x64 *pixeles*.

2. Implementar la Transformada Wavelet Discreta en dos dimensiones en lenguaje Java.
3. Aplicar un algoritmo de optimización de parámetros de filtros digitales que implementen la Transformada Wavelet Discreta sobre imágenes de 64x64 *pixeles*.
4. Aplicar la Transformada Wavelet Discreta a imágenes de 512x512 *pixeles* utilizando filtros digitales optimizados cuyos parámetros fueron optimizados con imágenes de 64x64 *pixeles*.
5. Realizar pruebas comparativas entre los filtros optimizados obtenidos con imágenes de 64x64 *pixeles* y filtros *Daubechies*.

1.6 Organización del reporte

El reporte del proyecto está organizado de la siguiente manera:

- **Información general.** Presenta elementos fundamentales para este trabajo como son: Introducción, justificación, objetivo general, objetivos específicos, y una breve descripción del contenido en este reporte.

- **Antecedentes.** Presenta antecedentes que introducen al tema de procesamiento de imágenes, específicamente la compresión de imágenes usando la transformada wavelet discreta.

- **Desarrollo del proyecto.** Presenta los módulos principales que componen este proyecto y una descripción técnica.

- **Recursos. Hardware y Software.** Herramientas Software para el desarrollo de las aplicaciones Java y características del equipo de cómputo utilizado.

- **Analisis y discusión de los resultados.** Se presenta el conjunto de resultados obtenidos en los experimentos realizados al concluir éste proyecto.

2 Antecedentes

2.1 Trabajos internos

2.1.1 Proyectos terminales

1. **Implementación de una aplicación software para procesamiento de imágenes con wavelets y bancos de filtros paramétricos de reconstrucción perfecta** [9]. En este trabajo se aplican filtros de reconstrucción perfecta a imágenes y a través de

una interfaz gráfica de usuario se proporcionan valores de los parámetros en forma manual.
En el proyecto propuesto un algoritmo evolutivo se usa para optimizar los parámetros de los filtros.

2. **Estudio experimental de la aproximación de filtros digitales paramétricos para implementar la transformada wavelet discreta con polinomios evolutivos** [10]. En este trabajo se aproximan filtros digitales paramétricos con polinomios cuyos coeficientes se calculan usando un algoritmo evolutivo.
   En el proyecto propuesto los filtros evolutivos se aplican para procesar imágenes y un algoritmo evolutivo se usa para optimizar los parámetros de los filtros.

3. **Clasificación de llantos de bebé con wavelets** [11]. En este trabajo se usa la transformada wavelet en una dimensión para procesar señales de audio.
   En el proyecto propuesto se aplica la transformada wavelet en dos dimensiones para procesar imágenes en escala de grises.

2.2 Trabajos externos

2.2.1 Artículos

1. **Frequency selective parameterized wavelets of length ten** [12]. En este artículo solo se comparan imágenes pero no se usa la entropía de los coeficientes de transformación sino que se calcula la energía de los coeficientes de transformación y con un solo nivel de descomposición (la transformada wavelet discreta solo se aplica una vez en sentido vertical y horizontal).
   En el trabajo propuesto se aplicará la transformada wavelet discreta con tres niveles de descomposición, y los parámetros optimizados se aplicarán a imágenes de mayor tamaño.

2.2.2 Tesis

1. **Image Compression By Wavelet Transform** [13]. La diferencia radica en que se aplican filtros del tipo "splines-biortogonales", y los que se usan en el proyecto de investigación son filtros ortogonales. También usan la codificación Embedded Zerotree Wavelet y en este trabajo se trabaja con la cantidad de coeficientes de transformación relevantes.

2.2.3 Software

1. **JPEG 2000**[14]. JPEG2000 es un estándar de codificación de imágenes que utiliza técnicas de compresión basados en wavelets.

2. **MATLAB** [15]. Implementa la Transformada Wavelet Discreta para realizar la compresión de imágenes en escala de grises o color verdadero de imágenes.

3. **DjVu** [16]. DjVu utiliza el algoritmo IW44 basado en wavelets para la compresión de imágenes.

2.3 Marco teórico

2.3.1 Formato PGM

PGM [17]. Es un acrónimo derivado de "Portable Gray Map".
El formato de un archivo PGM consiste en lo siguiente:

1. Un número mágico para identificar el tipo de archivo. En este proyecto se usa P5.
2. # comentario, las líneas que comienzan con "#" pueden ser comentarios.
3. El alto y el ancho son dos números enteros que representan las dimensiones horizontal y vertical respectivamente.
4. El valor máximo de gris (Maxval).
5. Continúan los datos de la imagen pixel0 pixel1 …

2.3.2 Compresión de la imagen

La compresión de imágenes se define como la manipulación de una representación original para obtener una versión equivalente que tenga una representación mínima.
En este trabajo se aplica la transformada wavelet discreta a una imagen aprovechando la propiedad de que primero se puede aplicar en un solo sentido (*horizontal*/*vertial*), y luego en el otro sentido (*vertical*/*horizontal*) por lo que se puede decir que para calcular la transformada wavelet discreta en dos dimensiones (sobre imágenes) se puede aplicar dos veces en una sola dimensión a la vez. Esto genera cuatro cuadrantes LL, LH, HL, y HH, que corresponden a la aplicación de los filtros pasabajas (*Low*) y pasaaltas (*High*) en los sentidos horizontal y vertical (ver Figura 2.1).

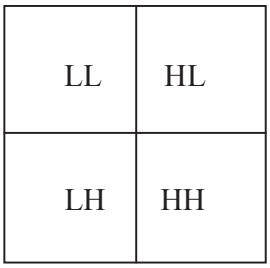


Figura 2.1. Primer nivel de transformación.

Al cuadrante LL se le puede volver a aplicar otro nivel de transformación, y tantos niveles como se pueda siempre que el tamaño del cuadrante (en sentido horizontal o vertical) sea mayor que la longitud del filtro empleado.

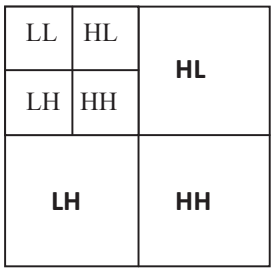


Figura 2.2. Segundo nivel de transformación.

En el caso de usar imágenes de longitud 64, al aplicar un primer nivel de transformación se genera un cuadrante LL de dimensiones 32x32, un segundo nivel de transformación genera un cuadrante LL de 16x16, y un tercer nivel da lugar a un cuadrante de 8x8 *pixeles*.

Puesto que la longitud de los filtros que se usan en este trabajo son de 4 y 6, sólo se aplican 3 niveles de transformación.
Para reconstruir la imagen se usan las bandas LL, LH, HL y HH del tercer nivel de transformación para generar el cuadrante LL del segundo nivel, y así sucesivamente se combinan las bandas del segundo nivel para generar la banda LL del primer nivel de transformación, con lo cual se puede obtener la imagen original.

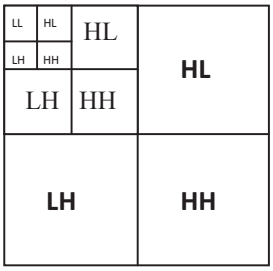


Figura 2.3. Tercer nivel de transformación.

Este es justamente el algoritmo de la Transformada Wavelet Discreta Inversa.

2.3.3 Transformada Wavelet Discreta

Uno de los resultados más importantes en las aplicaciones de la Transformada Wavelet Discreta es que se puede implementar usando un par de filtros ortogonales L y H, pasabajas y pasaaltas respectivamente que generan la bandas de frecuencias bajas, y frecuencias altas respectivamente.

A manera de ejemplo considérese el filtro pasabajas $h = \left\{\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right\}$ y el filtro pasaaltas $h = \left\{\frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}}\right\}$.

Cuando se aplica a un vector unidimensional de longitud 2^n, se generan dos subbandas (*vectores*) de longitud 2^{n-1}. Sea por ejemplo el vector $[a, b]$, con longitud 2 (n=1) que al aplicarle el filtro h se obtiene el vector $\left[\frac{(a+b)}{\sqrt{2}}\right]$ y al aplicarle el filtro g se obtiene el vector $\left[\frac{(a-b)}{\sqrt{2}}\right]$, entonces el vector resultante de la transformada wavelet unidimensional es el vector $\left[\frac{a+b}{\sqrt{2}}, \frac{a-b}{\sqrt{2}}\right]$.

En donde, como se precisó previamente, el número de coeficientes de transformación con baja frecuencia es $2^{1-1} = 1$, y también el número de coeficientes de alta frecuencia es 1.

Al combinar estos dos coeficientes de baja y alta frecuencia usando los mismos filtros de reconstrucción perfecta se tiene, para el filtro de baja frecuencia:

$$\left[\frac{\frac{a+b}{\sqrt{2}} + \frac{a-b}{\sqrt{2}}}{\sqrt{2}}\right] = a \tag{2.1}$$

Y para el filtro de alta frecuencia:

$$\left[\frac{\frac{a+b}{\sqrt{2}} + \frac{a-b}{\sqrt{2}}}{\sqrt{2}}\right] = b \tag{2.2}$$

En donde el error raíz cuadrático medio de reconstrucción está dado por:

$$RMS = \sqrt{(a-a)^2 + (b-b)^2} = 0 \tag{2.3}$$

Que es precisamente cero, por tratarse de filtros de reconstrucción perfecta. En la práctica podría haber posibles errores que no sea idénticamente cero atribuibles a los cálculos con números de punto flotante.

2.3.4 Algoritmos evolutivos

Los algoritmos evolutivos se basan principalmente el los conceptos de evolución de los seres vivos[18]. Se imitan por computadora los procesos de reproducción, selección, mutación, y cruzamiento entre otros. Existen múltiples técnicas evolutivas que han mostrado dar soluciones aceptables en tiempo razonable, a diferencia de otros métodos de optimización de búsqueda local, que tienen la desventaja de quedar frecuentemente atrapados en óptimos locales.

En este trabajo se usa la técnica de evolución diferencial.

2.3.4.1 Evolucion diferencial

La implementación más simple de la Evolución diferencial (ED) mantiene un par de poblaciones de vectores, ambos de los cuales contienen Np vectores (de D dimensiones) de parámetros con valores reales. La población actual, simbolizada por P_x, se compone de los vectores $x_{i,g}$, que ya se han encontrado para ser aceptables, ya sea como puntos iníciales, o por comparación con otros vectores:

$$\begin{aligned} \boldsymbol{P}_{\boldsymbol{X},g} &= (\boldsymbol{X}_{i,g}), \qquad i = 0,1,\dots,Np-1, \qquad g = 0,1,\dots,g_{max}, \\ \boldsymbol{X}_{i,g} &= (x_{j,i,g}), \qquad j = 0,1,\dots,D-1. \end{aligned} \tag{2.4}$$

Los índices empiezan con 0 para simplificar el trabajo con matrices.
El índice $g = 0,1,\dots,g_{max}$, indica la generación a la que un vector pertenece. Además, cada vector se le asigna un índice de población i, que va de 0 a $Np-1$. Los parámetros dentro de los vectores son indexados con j, que va de 0 a $D-1$. Una vez inicializado, el algoritmo de ED muta aleatoriamente a los vectores para producir una población intermedia $P_{V,g}$, de Np vectores mutantes $V_{i,g}$:

$$\begin{aligned} P_{V,g} &= (V_{i,g}), \qquad i = 0,1,\dots,Np-1, \qquad g = 0,1,\dots,g_{max}, \\ V_{i,g} &= (v_{j,i,g}), \qquad j = 0,1,\dots,D-1. \end{aligned} \tag{2.5}$$

Cada vector en la población actual se cruza con un vector mutante para producir una población del experimental $P_{\mathbf{u}}$, de Np vectores de prueba $\mathbf{u}_{i,g}$:

$$\begin{aligned} \mathrm{P}_{\mathbf{u},g} &= (\mathbf{u}_{i,g}), \quad i = 0,1,\dots,Np-1, \qquad g = 0,1,\dots,g_{max}, \\ \mathbf{u}_{i,g} &= (u_{j,i,g}), \quad j = 0,1,\dots,D-1. \end{aligned} \tag{2.6}$$

Durante la combinación, los vectores de prueba sobrescriben la población mutante, por lo que una única matriz puede contener ambas poblaciones.

2.3.4.2 Inicialización

Antes de la población, se pueden inicializar ambos límites superior e inferior, cada parámetro debe ser especificado. Estos valores $2D$ se pueden recoger en dos vectores de inicialización de D dimensiones, b_L y b_U, donde los subíndices L y U indican los límites inferior y superior,

respectivamente. Una vez que los límites se han especificado, un generador de números aleatorios asigna a cada parámetro de cada vector un valor dentro del rango establecido. Por ejemplo, el valor inicial ($g = 0$) del $j-$ésimo parámetro (elemento) del $i-$ésimo vector es

$$x_{j,i,0} = rand_j(0,1) \cdot \left(b_{j,U} - b_{j.L}\right) + b_{j.L} \tag{2.7}$$

El generador de números aleatorios $rand_j(0,1)$, devuelve un número aleatorio uniformemente distribuido dentro de la gama $[0,1)$, es decir, $0 \leq rand_j(0,1) < 1$. El subíndice j, indica que un nuevo valor aleatorio se genera para cada parámetro (*elemento de vector*). Incluso si una variable es discreta o integral, debe ser inicializada con un valor real, ya que la ED trata internamente todas las variables como de valores de punto flotante, independientemente de su tipo.

2.3.4.3 Mutación

Una vez inicializada la población, la ED muta y cruza la población para producir una población de Np vectores de prueba. En particular, la evolución diferencial añade una escala, un muestreo aleatorio, una diferencia vectorial a un tercer vector. La ecuación 2.8 muestra cómo combinar tres vectores diferentes, elegidos al azar para crear un vector mutante $\mathbf{V}_{i,g}$:

$$\mathbf{V}_{i,g} = \mathbf{X}_{r0,g} + F \cdot \left(\mathbf{X}_{r1,g} - \mathbf{X}_{r2,g}\right) \tag{2.8}$$

El factor de escala $F \in (0,1+)$, es un número real positivo que controla la velocidad en el que la población evoluciona. Mientras que no hay límite superior en F, valores eficaces son rara vez superior a 1.0.
El índice de *vector base*, $r0$, se puede determinar de varias maneras, pero por ahora se supone que es índice de un vector elegido al azar, que es diferente del índice i del *vector destino*. Excepto por ser distintos tanto de los índices de base como el vector objetivo, el *vector de diferencia* indexado por, $r1$ y $r2$, también se seleccionan al azar.

2.3.4.4 Cruzamiento

La ED cruza cada vector con un vector mutante.

$$\mathbf{u}_{i,g} = u_{j,i,g} = \begin{cases} v_{j,i,g} & si\ \left(rand_j(0,1) \leq Cr\ o\ j = j_{rand}\right) \\ x_{j,i,g} & en\ caso\ contrario. \end{cases} \tag{2.9}$$

La probabilidad de cruce, $Cr \in [0,1]$, es un valor definido por el usuario que controla la fracción de los valores de los parámetros que se copian del vector mutante. Para determinar qué fuente aporta un parámetro dado, el cruce uniforme compara Cr con la salida de un generador de números aleatorios uniforme, $rand_j$ (0,1). Si el número aleatorio es menor o igual a Cr, el parámetro de prueba se hereda del vector mutante $v_{i,g}$; de lo contrario, el parámetro se copia del vector $x_{i,g}$. Además, el parámetro es elegido al azar con índice j_{rand}, se toma del vector mutante para asegurar que el vector de prueba no duplica a $x_{i,g}$. Debido a esta condición adicional, Cr solamente se aproxima a la verdadera probabilidad, la p_{Cr}, de un parámetro de prueba que será heredado a partir del vector mutante.

2.3.4.5 Selección

Si el vector de prueba $\mathbf{u}_{i,g}$, tiene un valor de la función objetivo igual o inferior al de su vector objetivo $\mathbf{X}_{i,g}$, sustituye al vector objetivo en la próxima generación; de lo contrario, el vector objetivo conserva su lugar en la población durante al menos una generación más (Ecuación 2.10). Mediante la comparación de cada vector de prueba con el vector objetivo del que hereda parámetros, La ED vuelve a realizar combinación y selección de otros vectores de prueba:

$$\mathbf{X}_{i,g+1} = \begin{cases} \mathbf{u}_{i,g} & \text{si } f(\mathbf{u}_{i,g}) \leq f(\mathbf{X}_{i,g}) \\ \mathbf{X}_{i,g} & en\ caso\ contrario \end{cases} \tag{2.10}$$

Una vez generada la nueva población, el proceso de mutación, recombinación y la selección se repite hasta que el óptimo se encuentra, o un criterio de terminación se cumple, por ejemplo, el número de generaciones alcanza un máximo preestablecido g_{max}.

C – pseudocódigo

Se presenta al estilo C un pseudocódigo de Evolución diferencial clásico. Los índices vector $r0$, $r1$ y $r2$ son todos diferentes y distintos del índice destino i. Adicionalmente, la selección se retrasa hasta que la población de prueba esta completa.

```
// inicializacion

do //genera una poblacion de prueba
{
      for(i=0; i<Np; i++)  //r0!=r1!=r2!=i
      {
            do r0=floor(rand(0,1)*Np); while(r0 == i);
            do r1=floor(rand(0,1)*Np); while(r1 == r0 or r1 == i);
            do r2=floor(rand(0,1)*Np); while(r2 == r1 or r2 == r0 or r2
            == i);
            jrand = floor(D*rand(0,1));
```

```
            for(j=0; j<D; j++) //genera un vector de prueba
            {
                  if(rand(0,1) <= Cr or  j == jrand)
                  {

                        u_{j,i} = X_{j,r0} + F*(X_{j,r1} - X_{j,r2});
                  }
                  else
                  {
                        u_{j,i} = X_{j,i};
                  }
            }
      }

      // seleccion de la siguiente generacion
      for(i=0; i<Np; i++)
      {
            if( f(u_i) <= f(x_i)) ui = x_i;
      }
}while(criterio de paro);
```

3 Desarrollo del proyecto

3.1 Colección de imágenes en formato PGM

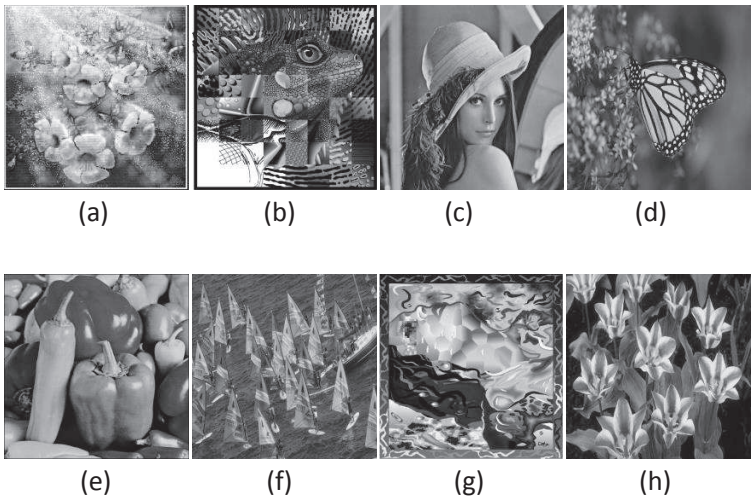


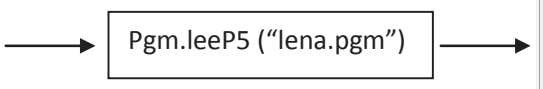
Figura 3.1. (a) clegg, (b) frymire, (c) lena, (d) monarch, (e) peppers, (f) sail, (g) serrano, (h) tulips.

3.2 Módulos principales

3.2.1 Lectura de la imagen en formato PGM

La entrada es un archivo de imagen en formato PGM en escala de grises y la salida es un arreglo bidimensional de punto flotante que dependerá del tamaño de la imagen de entrada el cual pude ser de 512x512 pixeles o de 64x64 pixeles.

Figura 3.2. Archivo pgm.



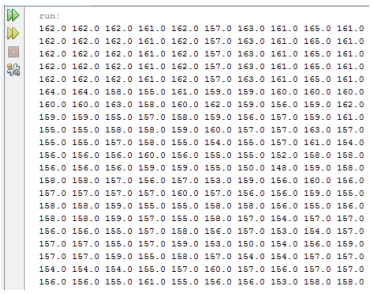


Figura 3.3. Arreglo 2d en patalla.

3.2.2 Reducción de la imagen

Se utilizó la Transformada Wavelet Discreta Haar [19] para redimensionaran una imagen de 512x512 pixeles a 64x64 pixeles en escala de grises.

Figura 3.4. Entrada pgm.

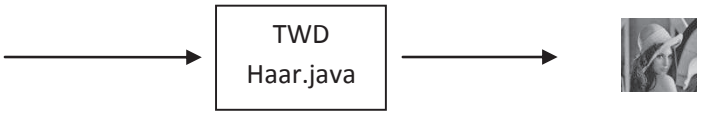


Figura 3.5. Salida pgm.

3.2.3 Implementacion de la Transformada Wavelet Discreta

En este módulo se implementó la Transformada Wavelet Discreta sobre un arreglo bidimensional de números de punto flotante que corresponden a los coeficientes de transformación por cada una de las imágenes de 64x64 pixeles para obtener el mayor número de coeficientes que se pueden hacer cero sin afectar la reconstrucción de la imagen.

3.2.4 Optimización de los filtros paramétricos utilizando un algoritmo de Evolución Diferencial

Se optimizaron los filtros digitales para imágenes de 64x64 *pixeles* y los resultados se enviaron a un archivo de texto para ser utilizados posteriormente.

El algoritmo de Evolución Diferencial usado para la optimización de los filtros digitales es una adaptación de un algoritmo de libre acceso, disponible en [20].

- Se genera una población con N individuos.
- Se inicia un ciclo. Mientras el número de iteraciones no sea mayor a un número máximo establecido se hace lo siguiente.
- Fase de mutación, para cada individuo en la población se genera un individuo experimental.
- Cruzamiento, se hace el cruce del individuo actual y el individuo experimental.
- Se evalúa a cada uno de los individuos de la población contra los individuos generados para quedarse con la mejor solución, en este caso con el mayor número de coeficientes irrelevantes.
- Si la desviación estándar obtenida hasta el momento es menor que el valor esperado entonces el ciclo termina, de lo contrario el ciclo continúa.
- Envía los resultados a un archivo de texto, es decir, la mejor solución (número de coeficientes irrelevantes) y los parámetros que generan dicha solución.

Como se había mencionado anteriormente, la optimización de estos parámetros se complica para imágenes de tamaño mayor a 512x512 *pixeles,* se plantea la posibilidad de optimizarlos en imágenes de 64x64 *pixeles*, y aplicarlos a imágenes de 512x512 *pixeles*.

3.2.5 Transformada Wavelet Discreta a imágenes de 512x512 pixeles utilizando filtros digitales optimizados

Con lo anterior se pueden generar filtros optimizados que ofrezcan mejores resultados en lugar de usar filtros "estándar".
Se probaron los filtros optimizados con imágenes de 64x64 *pixeles* sobre sus respectivas imágenes de 512x512 *pixeles* para obtener cual es el mayor número de coeficientes de transformación que pueden llevarse cero.

3.2.6 Transformada Wavelet Discreta a imágenes de 512x512 pixeles utilizando filtros digitales fijos

Se aplicó la Transformada Wavelet Discreta con filtros fijos sobre imágenes de 512x512 *pixeles* para obtener el número de coeficientes irrelevantes que genera cada filtro y tener un punto de comparación con los filtros optimizados.

3.2.7 Presentación de resultados

Finalmente se obtuvo un archivo por cada uno de los filtros digitales aplicados sobre las imágenes de 512x512 *pixeles*, el archivo contiene el nombre de la imagen, el número de coeficientes irrelevantes y los filtros empleados.

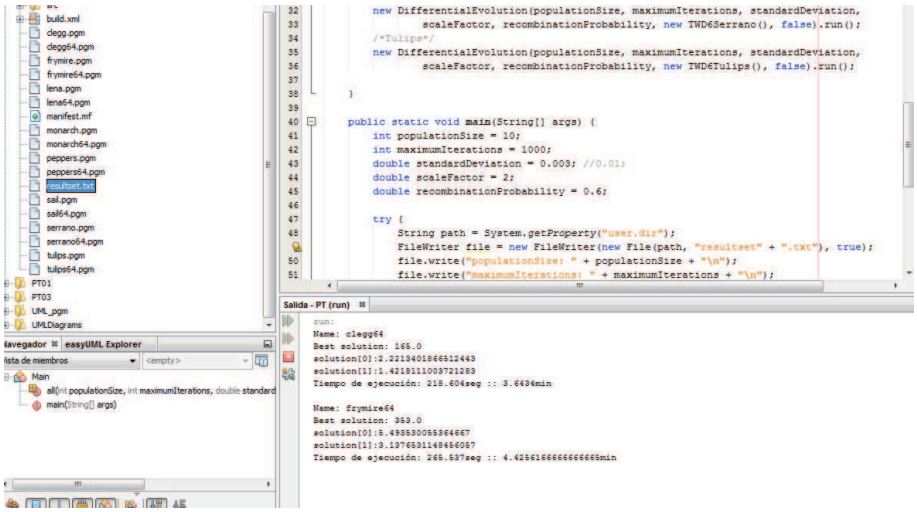


Figura 3.6. Ejecución del algoritmo de evolución diferencial.

3.3 Especificación técnica

En todos los módulos se usa el lenguaje de programación Java.
El formato de imágenes a procesar será PGM, que es un formato de imágenes en escala de grises.
Las imágenes a procesar tendrán un tamaño de 64x64 *pixeles* y de 512x512 *pixeles*.
El conjunto de imágenes de prueba está formado por "clegg", "frymire", "lena", "monarch", "peppers", "sail", "serrano" y "tulips" que son así conocidas en el contexto del procesamiento de imágenes [21].
Los filtros "estándar" a usar en los experimentos son Haar, Daubechies 4 y Daubechies 6.
Los filtros paramétricos a usar se forman de cuatro coeficientes.

4 Recursos

4.1 Aplicación software

Dichas aplicaciones están codificadas en lenguaje Java lo cual permite repetir el experimento sobre Sistemas Operativos y hardware diferentes a los empleados en este proyecto sin realizar modificación alguna en el código fuente del proyecto.

4.2 Hardware y software usado

Los experimentos se realizaron sobre una computadora de escritorio con las siguientes características:

1. Procesador: AMD Athlon 64 X2 Dual Core Processor 4200 2,9 GHz
2. Memoria RAM: 2.00 GB
3. Sistema operativo: GNU/Linux

El proyecto fue realizado utilizando el IDE de desarrollo NetBeans 7.4.

5 Resultados y discusión de los resultados

5.1 Resultados de la aplicación

Se realizaron al menos 10 ejecuciones de la aplicación para el conjunto de imágenes de 64x64 *pixeles* con los siguientes parámetros:

- Tamaño de la población: 50
- Máximo número de iteraciones: 1000
- Desviación estándar: 0.003
- Factor de escala: 2.0
- Probabilidad de combinación: 0.6

Los mejores resultados obtenidos se muestran en la Tabla 5.1.

Imagen	Filtro 1 optimizado	Filtro 2 optimizado	Coeficientes irrelevantes
Clegg	2.398218579925798	1.6311084841810302	180
Frymire	3.9253434284958484	4.109907531258745	253
Lena	1.2485241197436663	0.984708173607773	270
Monarch	1.3667110214706648	1.0287653541256685	322
Peppers	6.14206797039259	3.5560471358563848	230
Sail	5.991046887166441	3.5829858687033185	147
Serrano	3.961174559334411	1.3959662391765448	153
Tulips	4.286003583144051	3.862738318040813	142

Tabla 5.1. Filtros optimizados y número de coeficientes irrelevantes.

5.2 Conclusiones

Después de obtener los filtros optimizados con imágenes de 64x64 *pixeles* (ver tabla 5.1) se aplicaron sobre imágenes de 512x512 *pixeles*.

En una comparación con filtros "estándar" se genera la tabla 5.2. En la tabla 5.2 se muestra la comparación de los coeficientes de transformación que se pueden llevar a cero sin afectar la reconstrucción de las imágenes utilizando los diferentes filtros: Haar, Daubechies 4, Daubechies 6 y filtros optimizados con imágenes pequeñas.

imagen	Clegg	Frymire	Lena	Monarch	Peppers	Sail	Serrano	Tulips
filtro	NA	NA	NA	NA	NA	NA	NA	NA
Haar	24264	74288	21527	30417	22580	10223	86838	20692
Daubechies4	22449	54338	18097	23597	19902	10263	60633	18237
Daubechies6	21581	44485	18060	23831	20003	10413	50455	18741
Optimizados	23471	73791	17983	23517	19502	10322	76146	16497

Tabla 5.2. Número de coeficientes irrelevantes en imágenes de 512x512 *pixeles*.

La conclusión se puede formular de la manera siguiente: la aplicación de filtros optimizados con imágenes pequeñas sobre imágenes de mayor tamaño no favorece la compresión de dichas imágenes respectivamente; pero si genera un número de coeficientes irrelevantes muy parecido a los que se obtienen con los filtros Daubechies 6.

Podrían usarse estos filtros para realizar pruebas en algunas otras aplicaciones debido a su rapidez, ya que la evolución de los filtros digitales para imágenes de 64x64 *pixeles* es por mucho más rápida que para imágenes de 512x512 *pixeles*.

Apéndice

Código fuente

Pgm.java

```
public class Pgm {

    public static double[][] leeP5(String filePath) {
        double[][] data2D = null;
        try {
            FileInputStream fileInputStream = new
            FileInputStream(filePath);
            Scanner scan = new Scanner(fileInputStream);
            // Descartar el numero magico
            scan.nextLine();
            // Descartar la linea comentada
            scan.nextLine();
            // Lee el alto, el ancho y el valor maximo
            // de intensidad de gris
            int picWidth = scan.nextInt();
            int picHeight = scan.nextInt();
            int maxvalue = scan.nextInt();
            fileInputStream.close();

            fileInputStream = new FileInputStream(filePath);
            DataInputStream dis = new DataInputStream(fileInputStream);
            // ignora las primeras 4 lineas del archivo pgm
            int numnewlines = 4;
            while (numnewlines > 0) {
                char c;
                do {
                    c = (char) (dis.readUnsignedByte());
                } while (c != '\n');
                numnewlines--;
            }
            // leer en un arreglo bidimencional la imagen
            data2D = new double[picHeight][picWidth];
            for (int row = 0; row < picHeight; row++) {
                for (int col = 0; col < picWidth; col++) {
                    data2D[row][col] = dis.read();
                }
            }
        } catch (FileNotFoundException fnfe) {
            System.out.println(fnfe);
        } catch (IOException ioe) {
            System.out.println(ioe);
        }
        return data2D;
    }


    public static void escribeP5(String filePath, double[][] data2D) {
```

```
        try {
            FileOutputStream fos = new FileOutputStream(filePath);
            DataOutputStream dos = new DataOutputStream(fos);

            //Estructura de una imagen P5
            dos.writeBytes("P5");
            dos.writeBytes("\n# Nueva imagen P5\n");
            dos.writeBytes(data2D.length + " " + data2D[0].length);
            dos.writeBytes("\n255\n");
            //Escribe valores de pixeles
            for (int i = 0; i < data2D.length; i++) {
                for (int j = 0; j < data2D[0].length; j++) {
                    dos.write((int) data2D[i][j]);
                }
            }

            dos.close();

        } catch (Exception e) {
            System.err.println("Error : " + e.getMessage());
        }
    }

}
```

TestPgm.java

```
public class TestPgm {
    // Lee un archivo pgm y en un arreglo de dos dimensiones
    // y los despliega en pantalla.
    public static void main(String[] args) {
        String path = System.getProperty("user.dir");
        String filePath = path + "\\lena.pgm";
        double[][] data = new double[512][512];
        data = Pgm.leeP5(filePath);

        for (int i = 0; i < data.length; i++) {
            for (int j = 0; j < data[0].length; j++) {
                System.out.print(data[i][j] + " ");
            }
            System.out.println("");
        }
    }
}
```

Haar2D.java

```
public class Haar2D {

    public static void reduceImagen(String name, int level) {
        String path = System.getProperty("user.dir");
        String filePath = path + "\\" + name + ".pgm";
        double[][] matrix = Pgm.leeP5(filePath);
        FI f = new FI();
        double[][] copy = f.TDWLevels(matrix, level);
        String fileOut = path + "\\" + name + "64.pgm";
        Pgm.escribeP5(fileOut, copy);
    }
}

class FI {

    double[] h4;
    double[] g4;
    double[] Ih4;
    double[] Ig4;

    void AlfaToHs(double alfa4) {
        int filtersize = 4;
        double[] hr4 = new double[filtersize];
        hr4[0] = 1 / 4.0 + Math.cos(alfa4) / (2 * Math.sqrt(2));
        hr4[1] = 1 / 4.0 + Math.sin(alfa4) / (2 * Math.sqrt(2));
        hr4[2] = 1 / 4.0 - Math.cos(alfa4) / (2 * Math.sqrt(2));
        hr4[3] = 1 / 4.0 - Math.sin(alfa4) / (2 * Math.sqrt(2));
        h4[0] = hr4[0];
        h4[1] = hr4[1];
        h4[2] = hr4[2];
        h4[3] = hr4[3];
        hr4 = null;
    }

    public FI(){
        h4 = new double[4];
        g4 = new double[4];
        Ih4 = new double[4];
        Ig4 = new double[4];

        AlfaToHs(Math.PI / 4);
        calculahs4(4);
        calculaihs4(4);
    }


    public double[][] TDWLevels(double[][] a, int level) {
        int y = a.length;
        int x = a[0].length;

        int filtersize4 = 4;
        for (int nivel = 0; nivel < level; nivel++) {
            int n = y >> nivel;
```

```
        //Dimension vertical TRANSFORMACION VERTICAL

        for (int jy = 0; jy < n; jy++) {
            double[] arreglo = new double[n];
            for (int ix = 0; ix < n; ix++) {
                arreglo[ix] = a[jy][ix];
            }
            twd4(arreglo, n, filtersize4);
            for (int ix = 0; ix < n; ix++) {
                a[jy][ix] = arreglo[ix];
            }
            arreglo = null;
        }

        //Dimension horizontal TRANSFORMACION HORIZONTAL
        for (int ix = 0; ix < n; ix++) {
            double[] arreglo = new double[n];
            for (int jy = 0; jy < n; jy++) {
                arreglo[jy] = a[jy][ix];
            }
            twd4(arreglo, n, filtersize4);
            for (int jy = 0; jy < n; jy++) {
                a[jy][ix] = arreglo[jy];
            }
            arreglo = null;
        }
    }

    int new_length = a.length >> level;
    double[][] new_a = new double[new_length][new_length];
    for (int i = 0; i < new_length; i++) {
        System.arraycopy(a[i], 0, new_a[i], 0, new_length);
    }
    a = null;
    return new_a;
}

public void ITWDLevels(double[][] a) {
    int level = 3;
    int y = 512;
    int x = 512;
    int filtersize4 = 4;
    for (int nivel = level - 1; nivel >= 0; nivel--) {
        int n = y >> nivel;

        //Dimension vertical TRANSFORMACION VERTICAL
        for (int ix = 0; ix < n; ix++) {
            double[] arreglo = new double[n];
            for (int jy = 0; jy < n; jy++) {
                arreglo[jy] = a[jy][ix];
            }
            invtwd4(arreglo, n, filtersize4);
            for (int jy = 0; jy < n; jy++) {
                a[jy][ix] = arreglo[jy];
            }
            arreglo = null;
        }
```

```
        //Dimension vertical TRANSFORMACION VERTICAL
        for (int jy = 0; jy < n; jy++) {
            double[] arreglo = new double[n];
            for (int ix = 0; ix < n; ix++) {
                arreglo[ix] = a[jy][ix];
            }
            invtwd4(arreglo, n, filtersize4);
            for (int ix = 0; ix < n; ix++) {
                a[jy][ix] = arreglo[ix];
            }
            arreglo = null;
        }
    }
}

void error(double[][] a, double[][] ori) {
    int x = 512;
    int y = 512;
    double rms = 0.0;

    for (int jy = 0; jy < y; jy++) {
        for (int ix = 0; ix < x; ix++) {
            rms += (a[jy][ix] - ori[jy][ix]) * (a[jy][ix] -
            ori[jy][ix]);
        }
    }

    rms = rms / ((double) x * y);//(rowmax-rowmin+1));
    rms = Math.sqrt(rms);
    System.out.println("RMS = " + rms);
}

void calculahs4(int filtersize4) {
    for (int i = 0; i < filtersize4; i++) {
        if (i % 2 == 0) {
            g4[i] = h4[filtersize4 - 1 - i];
        } else {
            g4[i] = -h4[filtersize4 - 1 - i];
        }
    }
}

void calculaihs4(int filtersize4) {
    for (int i = 0; i < filtersize4; i++) {
        int j = filtersize4 - 1 - i;
        int j2 = (int) (j / 2); // 2*(int)((8-1-i)/2)

        if (i % 2 == 0) { //pares h, impares g
            Ih4[i] = h4[2 * j2];
            Ig4[i] = h4[2 * j2 + 1];
        } else {
            Ih4[i] = g4[2 * j2];
            Ig4[i] = g4[2 * j2 + 1];
        }
    }
}
```

```
    //usa filtros como apuntadores h4[i]
    void twd4(double[] a, int n, int filtersize4) {
    /*”a” es el arreglo unidimensional, n es la longitud del vector a
      transformar*/
        if (n >= filtersize4) {
            int i, j;
            int ndiv2 = n / 2; // n entre 2
            double[] tmp = new double[n];
            i = 0;

            for (j = 0; j <= n - 2; j = j + 2) {
                tmp[i] = 0;
                tmp[i + ndiv2] = 0;
                for (int z = 0; z < filtersize4; z++) {
                    tmp[i] += a[(j + z) % n] * h4[z];
                    tmp[i + ndiv2] += a[(j + z) % n] * g4[z];
                }
                i++;
            }
            for (i = 0; i < n; i++) {
                a[i] = tmp[i];
            }
            tmp = null;
        }
    }

    void invtwd4(double[] a, int n, int filtersize4) {
        /*"a" es el arreglo unidimensional, n es la longitud del vector
recuperado*/
        if (n >= filtersize4) {
            int i, j;
            int ndiv2 = n / 2; // n entre 2
            double[] tmp = new double[n];
            j = filtersize4 - 2;
            for (i = 0; i <= ndiv2 - 1; i++) {
                tmp[(j) % n] = 0;
                for (int w = 0,q = 0; q < filtersize4 / 2; q++, w += 2) {
                    int ii = (i + w / 2) % ndiv2;
                    tmp[(j) % n] += a[ii] * Ih4[w] + a[ii + ndiv2] *
                    Ih4[w + 1];
                }
                j++;
                tmp[(j) % n] = 0;
                for (int w = 0; w < filtersize4; w += 2) {
                    tmp[(j) % n] += a[(i + w / 2) % ndiv2] * Ig4[w] +
                    a[(i + w / 2) % ndiv2 + ndiv2] * Ig4[w + 1];
                }
                j++;
            }
            for (i = 0; i < n; i++) {
                a[i] = tmp[i];
            }
            tmp = null;
        }
    }
}
```

TestHaar2D.java

```
public class TestHaar2D {

    public static void main(String[] args) {
        int level = 3;

        Haar2D.reduceImagen("clegg", level);
        Haar2D.reduceImagen("frymire", level);
        Haar2D.reduceImagen("lena", level);
        Haar2D.reduceImagen("monarch", level);
        Haar2D.reduceImagen("peppers", level);
        Haar2D.reduceImagen("sail", level);
        Haar2D.reduceImagen("serrano", level);
        Haar2D.reduceImagen("tulips", level);

    }
}
```

Haar.java

```
public class Haar {

    public int haar(String fileName, double alfa) {
        String filePath = System.getProperty("user.dir") + "\\" +
        fileName;
        double[][] copy = Pgm.leeP5(filePath);
        int level = 3;
        FI pgmData = new FI(copy, level, alfa);
        int ea = pgmData.energiaAcumulada(copy);
        return ea;
    }
}

class FI {

    double[] h4;
    double[] g4;
    double[] Ih4;
    double[] Ig4;

    void AlfaToHs(double alfa4) {
        int filtersize = 4;
        double[] hr4 = new double[filtersize];
        hr4[0] = 1 / 4.0 + Math.cos(alfa4) / (2 * Math.sqrt(2));
        hr4[1] = 1 / 4.0 + Math.sin(alfa4) / (2 * Math.sqrt(2));
        hr4[2] = 1 / 4.0 - Math.cos(alfa4) / (2 * Math.sqrt(2));
        hr4[3] = 1 / 4.0 - Math.sin(alfa4) / (2 * Math.sqrt(2));
        h4[0] = hr4[0];
        h4[1] = hr4[1];
        h4[2] = hr4[2];
        h4[3] = hr4[3];
        hr4 = null;
    }
```

```
public int energiaAcumulada(double[][] a) {
    double Ef = 0;
    int x = a[0].length;
    int y = a.length;
    double[] Ls = new double[x * y];
    int ii = 0;
    for (int jy = 0; jy < y; jy++) {
        for (int ix = 0; ix < x; ix++) {
            Ls[ii++] = a[jy][ix];
            Ef += a[jy][ix] * a[jy][ix];
        }
    }
    for (ii = 0; ii < x * y - 1; ii++) {
        for (int jj = ii + 1; jj < x * y; jj++) {
            if (Ls[ii] * Ls[ii] < Ls[jj] * Ls[jj]) {
                //< de mayor a menor
                double tmp = Ls[ii];
                Ls[ii] = Ls[jj];
                Ls[jj] = tmp;
            }
        }
    }

    double Ep = 0.0;
    int lai = x * y - 1; //peor caso

    for (ii = 0; ii < x * y; ii++) {
        Ep += Ls[ii] * Ls[ii];
        if (Ep / Ef <= 1.0 && Ep / Ef > 0.999999) {
            lai = ii;
            break;
        }
    }
    int ceros = 0;
    if (lai - 1 < 0) {
        lai = 4095;
        System.out.println("negativo " + (lai - 1));
    } else {
        double Th = Math.sqrt(Ls[lai] * Ls[lai]);
    }
    Ls = null;
    ceros = x * y - (lai + 1);

    return ceros;
}
```

```
public FI(double[][] matrix, int level, double alfa) {
    h4 = new double[4];
    g4 = new double[4];
    Ih4 = new double[4];
    Ig4 = new double[4];
    //AlfaToHs(Math.PI / 4);
    AlfaToHs(alfa);
    calculahs4(4);
    calculaihs4(4);
```

```
    double[][] ori = new double[matrix.length][matrix[0].length];
    for (int r = 0; r < matrix.length; r++) {
        for (int rr = 0; rr < matrix[0].length; rr++) {
            ori[r][rr] = matrix[r][rr];
        }
    }

    TDWLevels(matrix, level);
    //ITWDLevels(matrix);
    //error(ori, matrix);
}



public void TDWLevels(double[][] a, int level) {
    int y = a.length;
    int x = a[0].length;
    int filtersize4 = 4;
    for (int nivel = 0; nivel < level; nivel++) {
        int n = y >> nivel;
        //Dimension vertical TRANSFORMACION VERTICAL
        for (int jy = 0; jy < n; jy++) {
            double[] arreglo = new double[n];
            for (int ix = 0; ix < n; ix++) {
                arreglo[ix] = a[jy][ix];
            }
            twd4(arreglo, n, filtersize4);
            for (int ix = 0; ix < n; ix++) {
                a[jy][ix] = arreglo[ix];
            }
            arreglo = null;
        }
        //Dimension horizontal TRANSFORMACION HORIZONTAL
        for (int ix = 0; ix < n; ix++) {
            double[] arreglo = new double[n];
            for (int jy = 0; jy < n; jy++) {
                arreglo[jy] = a[jy][ix];
            }
            twd4(arreglo, n, filtersize4);
            for (int jy = 0; jy < n; jy++) {
                a[jy][ix] = arreglo[jy];
            }
            arreglo = null;
        }
    }
}
```

```
public void ITWDLevels(double[][] a) {
    int level = 3;
    int y = 512;
    int x = 512;
    int filtersize4 = 4;
    for (int nivel = level - 1; nivel >= 0; nivel--) {
        int n = y >> nivel;
        //Dimension vertical TRANSFORMACION VERTICAL
        for (int ix = 0; ix < n; ix++) {
        double[] arreglo = new double[n];
            for (int jy = 0; jy < n; jy++) {
                arreglo[jy] = a[jy][ix];
            }
            invtwd4(arreglo, n, filtersize4);
            for (int jy = 0; jy < n; jy++) {
                a[jy][ix] = arreglo[jy];
            }
            arreglo = null;
        }

        //Dimension vertical TRANSFORMACION VERTICAL
        for (int jy = 0; jy < n; jy++) {
            double[] arreglo = new double[n];
            for (int ix = 0; ix < n; ix++) {
                arreglo[ix] = a[jy][ix];
            }
            invtwd4(arreglo, n, filtersize4);
            for (int ix = 0; ix < n; ix++) {
                a[jy][ix] = arreglo[ix];
            }
            arreglo = null;
        }
    }
}

void error(double[][] a, double[][] ori) {
    int x = 512;
    int y = 512;
    double rms = 0.0;

    for (int jy = 0; jy < y; jy++) {
        for (int ix = 0; ix < x; ix++) {
            rms += (a[jy][ix] - ori[jy][ix]) * (a[jy][ix] -
            ori[jy][ix]);
        }
    }

    rms = rms / ((double) x * y);//(rowmax-rowmin+1));
    rms = Math.sqrt(rms);
    System.out.println("RMS = " + rms);
}
```

```
    void calculahs4(int filtersize4) {
        for (int i = 0; i < filtersize4; i++) {
            if (i % 2 == 0) {
                g4[i] = h4[filtersize4 - 1 - i];
                //System.out.println("g4[" + i + "]: " + g4[i]);
            } else {
                g4[i] = -h4[filtersize4 - 1 - i];
                //System.out.println("g4[" + i + "]: " + g4[i]);
            }
            //Equivalente g[i]=pow(-1,i)*h4[filtersize-1-i];
        }
    }

    void calculaihs4(int filtersize4) {
        for (int i = 0; i < filtersize4; i++) {
            int j = filtersize4 - 1 - i;
            int j2 = (int) (j / 2); // 2*(int)((8-1-i)/2)

            if (i % 2 == 0) { //pares h, impares g
                Ih4[i] = h4[2 * j2];
                Ig4[i] = h4[2 * j2 + 1];
            } else {
                Ih4[i] = g4[2 * j2];
                Ig4[i] = g4[2 * j2 + 1];
            }
        }
    }

    //usa filtros como apuntadores h4[i]
    void twd4(double[] a, int n, int filtersize4) {
    /* "a" es el arreglo unidimensional, n es la longitud del vector a
transformar*/
        if (n >= filtersize4) {
            int i, j;
            int ndiv2 = n / 2; // n entre 2
            double[] tmp = new double[n];
            i = 0;

            for (j = 0; j <= n - 2; j = j + 2) {
                tmp[i] = 0;
                tmp[i + ndiv2] = 0;
                for (int z = 0; z < filtersize4; z++) {
                    tmp[i] += a[(j + z) % n] * h4[z];
                    tmp[i + ndiv2] += a[(j + z) % n] * g4[z];
                }
                i++;
            }
            for (i = 0; i < n; i++) {
                a[i] = tmp[i];
            }
            tmp = null;
        }
    }
```

```
    void invtwd4(double[] a, int n, int filtersize4) {
        /* "a" es el arreglo unidimensional, n es la longitud del vector
recuperado*/
        if (n >= filtersize4) {
            int i, j;
            int ndiv2 = n / 2; // n entre 2

            double[] tmp = new double[n];
            j = filtersize4 - 2;
            for (i = 0; i <= ndiv2 - 1; i++) {
                tmp[(j) % n] = 0;
                for(int w = 0, q = 0; q < filtersize4 / 2; q++, w += 2) {
                    int ii = (i + w / 2) % ndiv2;
                    tmp[(j) % n] += a[ii] * Ih4[w] + a[ii + ndiv2] *
                    Ih4[w + 1];
                }
                j++;
                tmp[(j) % n] = 0;
                for (int w = 0; w < filtersize4; w += 2) {
                    tmp[(j) % n] += a[(i + w / 2) % ndiv2] * Ig4[w] +
                    a[(i + w / 2) % ndiv2 + ndiv2] * Ig4[w + 1];
                }
                j++;

            }
            for (i = 0; i < n; i++) {
                a[i] = tmp[i];
            }
            tmp = null;
        }
    } // invtransform
}
```

TestHaar.java

```
public class TestHaar {

    // transformada wavelet discreta sobre imagenes de 512x512 pixeles

    public static void haar(String fileName) {

        double alfa = 0.78539816; //Math.PI / 4;
        Haar h = new Haar();
        int ea = h.haar(fileName, alfa);
        System.out.println("file: " + fileName);
        System.out.println("alfa: " + alfa);
        System.out.println("ceros: " + ea);
        System.out.println("");
```

```
            try {
                String path = System.getProperty("user.dir");
                FileWriter file = new FileWriter(new File(path, "haar" +
                ".txt"), true);
                file.write("file: " + fileName + "\n");
                file.write("alfa: " + alfa + "\n");
                file.write("ceros: " + ea + "\n\n");
                file.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }

        public static void main(String[] args) {
            TestHaar.haar("clegg.pgm");
            TestHaar.haar("frymire.pgm");
            TestHaar. haar("lena.pgm");
            TestHaar.haar("monarch.pgm");
            TestHaar.haar("peppers.pgm");
            TestHaar.haar("sail.pgm");
            TestHaar.haar("serrano.pgm");
            TestHaar.haar("tulips.pgm");
        }
    }
```

TWD4.java

```
public class TWD4 {

    public int twd4(String fileName, double alfa) {
        String filePath = System.getProperty("user.dir") + "\\" +
        fileName;
        double[][] copy = Pgm.leeP5(filePath);
        int level = 3;
        FI pgmData = new FI(copy, level, alfa);
        int ea = pgmData.energiaAcumulada(copy);
        return ea;
    }
}

class FI {

    double[] h4;
    double[] g4;
    double[] Ih4;
    double[] Ig4;

    void AlfaToHs(double alfa4) {
        int filtersize = 4;
        double[] hr4 = new double[filtersize];
        hr4[0] = 1 / 4.0 + Math.cos(alfa4) / (2 * Math.sqrt(2));
        hr4[1] = 1 / 4.0 + Math.sin(alfa4) / (2 * Math.sqrt(2));
        hr4[2] = 1 / 4.0 - Math.cos(alfa4) / (2 * Math.sqrt(2));
        hr4[3] = 1 / 4.0 - Math.sin(alfa4) / (2 * Math.sqrt(2));
```

```
        h4[0] = hr4[0];
        h4[1] = hr4[1];
        h4[2] = hr4[2];
        h4[3] = hr4[3];
        hr4 = null;
    }

    public int energiaAcumulada(double[][] a) {
        double Ef = 0;
        int x = a[0].length;
        int y = a.length;
        double[] Ls = new double[x * y];
        int ii = 0;
        for (int jy = 0; jy < y; jy++) {
            for (int ix = 0; ix < x; ix++) {
                Ls[ii++] = a[jy][ix];
                Ef += a[jy][ix] * a[jy][ix];
            }
        }
        for (ii = 0; ii < x * y - 1; ii++) {
            for (int jj = ii + 1; jj < x * y; jj++) {
                if (Ls[ii] * Ls[ii] < Ls[jj] * Ls[jj]) {
                    //< de mayor a menor
                    double tmp = Ls[ii];
                    Ls[ii] = Ls[jj];
                    Ls[jj] = tmp;
                }
            }
        }

        double Ep = 0.0;
        int lai = x * y - 1; //peor caso

        for (ii = 0; ii < x * y; ii++) {
            Ep += Ls[ii] * Ls[ii];
            if (Ep / Ef <= 1.0 && Ep / Ef > 0.999999) {
                lai = ii;
                break;
            }
        }
        int ceros = 0;
        if (lai - 1 < 0) {
            lai = 4095;
            System.out.println("negativo " + (lai - 1));
        } else {
            double Th = Math.sqrt(Ls[lai] * Ls[lai]);
        }
        Ls = null;
        ceros = x * y - (lai + 1);

        return ceros;
    }
```

```
public FI(double[][] matrix, int level, double alfa) {
    h4 = new double[4];
    g4 = new double[4];
    Ih4 = new double[4];
    Ig4 = new double[4];

    AlfaToHs(alfa);
    calculahs4(4);
    calculaihs4(4);
    double[][] ori = new double[matrix.length][matrix[0].length];
    for (int r = 0; r < matrix.length; r++) {
        for (int rr = 0; rr < matrix[0].length; rr++) {
            ori[r][rr] = matrix[r][rr];
        }
    }

    TDWLevels(matrix, level);
    //ITWDLevels(matrix);
    //error(ori, matrix);
}


public void TDWLevels(double[][] a, int level) {
    int y = a.length;
    int x = a[0].length;
    int filtersize4 = 4;
    for (int nivel = 0; nivel < level; nivel++) {
        int n = y >> nivel;

        //Dimension vertical TRANSFORMACION VERTICAL
        for (int jy = 0; jy < n; jy++) {
            double[] arreglo = new double[n];
            for (int ix = 0; ix < n; ix++) {
                arreglo[ix] = a[jy][ix];
            }
            twd4(arreglo, n, filtersize4);
            for (int ix = 0; ix < n; ix++) {
                a[jy][ix] = arreglo[ix];
            }
            arreglo = null;
        }

        //Dimension horizontal TRANSFORMACION HORIZONTAL
        for (int ix = 0; ix < n; ix++) {
            double[] arreglo = new double[n];
            for (int jy = 0; jy < n; jy++) {
                arreglo[jy] = a[jy][ix];
            }
            twd4(arreglo, n, filtersize4);
            for (int jy = 0; jy < n; jy++) {
                a[jy][ix] = arreglo[jy];
            }
            arreglo = null;
        }
    }
}
```

```
public void ITWDLevels(double[][] a) {
    int level = 3;
    int y = 512;
    int x = 512;
    int filtersize4 = 4;
    for (int nivel = level - 1; nivel >= 0; nivel--) {
        int n = y >> nivel;

        //Dimension vertical TRANSFORMACION VERTICAL
        for (int ix = 0; ix < n; ix++) {
            double[] arreglo = new double[n];
            for (int jy = 0; jy < n; jy++) {
                arreglo[jy] = a[jy][ix];
            }
            invtwd4(arreglo, n, filtersize4);
            for (int jy = 0; jy < n; jy++) {
                a[jy][ix] = arreglo[jy];
            }
            arreglo = null;
        }

        //Dimension vertical TRANSFORMACION VERTICAL
        for (int jy = 0; jy < n; jy++) {
            double[] arreglo = new double[n];
            for (int ix = 0; ix < n; ix++) {
                arreglo[ix] = a[jy][ix];
            }
            invtwd4(arreglo, n, filtersize4);
            for (int ix = 0; ix < n; ix++) {
                a[jy][ix] = arreglo[ix];
            }
            arreglo = null;
        }
    }
}

void error(double[][] a, double[][] ori) {
    int x = 512;
    int y = 512;
    double rms = 0.0;

    for (int jy = 0; jy < y; jy++) {
        for (int ix = 0; ix < x; ix++) {
            rms += (a[jy][ix] - ori[jy][ix]) * (a[jy][ix] -
            ori[jy][ix]);
        }
    }

    rms = rms / ((double) x * y);//(rowmax-rowmin+1));
    rms = Math.sqrt(rms);
    System.out.println("RMS = " + rms);
}

void calculahs4(int filtersize4) {
    for (int i = 0; i < filtersize4; i++) {
        if (i % 2 == 0) {
            g4[i] = h4[filtersize4 - 1 - i];
```

```
                //System.out.println("g4[" + i + "]: " + g4[i]);
            } else {
                g4[i] = -h4[filtersize4 - 1 - i];
                //System.out.println("g4[" + i + "]: " + g4[i]);
            }
            //Equivalente g[i]=pow(-1,i)*h4[filtersize-1-i];
        }
    }

    void calculaihs4(int filtersize4) {
        for (int i = 0; i < filtersize4; i++) {
            int j = filtersize4 - 1 - i;
            int j2 = (int) (j / 2); // 2*(int)((8-1-i)/2)

            if (i % 2 == 0) { //pares h, impares g
                Ih4[i] = h4[2 * j2];
                Ig4[i] = h4[2 * j2 + 1];
            } else {
                Ih4[i] = g4[2 * j2];
                Ig4[i] = g4[2 * j2 + 1];
            }
        }
    }

    void twd4(double[] a, int n, int filtersize4) {
    //usa filtros como apuntadores h4[i]
    /* "a" es el arreglo unidimensional, n es la longitud del vector a
transformar*/
        if (n >= filtersize4) {
            int i, j;
            int ndiv2 = n / 2; // n entre 2
            double[] tmp = new double[n];
            i = 0;

            for (j = 0; j <= n - 2; j = j + 2) {
                tmp[i] = 0;
                tmp[i + ndiv2] = 0;
                for (int z = 0; z < filtersize4; z++) {
                    tmp[i] += a[(j + z) % n] * h4[z];
                    tmp[i + ndiv2] += a[(j + z) % n] * g4[z];
                }
                i++;
            }
            for (i = 0; i < n; i++) {
                a[i] = tmp[i];
            }
            tmp = null;
        }
    }

    void invtwd4(double[] a, int n, int filtersize4) {
        /* "a" es el arreglo unidimensional, n es la longitud del vector
recuperado*/
        if (n >= filtersize4) {
            int i, j;
            int ndiv2 = n / 2; // n entre 2
```

```
            double[] tmp = new double[n];
            j = filtersize4 - 2;
            for (i = 0; i <= ndiv2 - 1; i++) {
                tmp[(j) % n] = 0;
                for(int w = 0, q = 0; q < filtersize4 / 2; q++, w += 2) {
                    int ii = (i + w / 2) % ndiv2;
                    tmp[(j) % n] += a[ii] * Ih4[w] + a[ii + ndiv2] *
                    Ih4[w + 1];
                }
                j++;
                tmp[(j) % n] = 0;
                for (int w = 0; w < filtersize4; w += 2) {
                    tmp[(j) % n] += a[(i + w / 2) % ndiv2] * Ig4[w] +
                    a[(i + w / 2) % ndiv2 + ndiv2] * Ig4[w + 1];
                }
                j++;

            }
            for (i = 0; i < n; i++) {
                a[i] = tmp[i];
            }
            tmp = null;
        }
    } // invtransform
}
```

TestTWD4.java

```
public class TestTWD4 {
    // transformada wavelet discreta sobre imagenes de 512x512 pixeles
    public static void twd4(String fileName) {
        double alfa = 5*Math.PI/12; //α = 5*Math.PI/12 or 13*Math.PI/12
        TWD4 lena = new TWD4();
        int ea = lena.twd4(fileName, alfa);
        System.out.println("file: " + fileName);
        System.out.println("alfa: " + alfa);
        System.out.println("ceros: " + ea);
        System.out.println("");
         try {
            String path = System.getProperty("user.dir");
            FileWriter file = new FileWriter(new File(path, "twd4" +
            ".txt"), true);
            file.write("file: " + fileName + "\n");
            file.write("alfa: " + alfa + "\n");
            file.write("ceros: " + ea + "\n\n");
            file.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

```
    public static void main(String[] args) {

        TestTWD4.twd4("clegg.pgm");
        TestTWD4.twd4("frymire.pgm");
        TestTWD4.twd4("lena.pgm");
        TestTWD4.twd4("monarch.pgm");
        TestTWD4.twd4("peppers.pgm");
        TestTWD4.twd4("sail.pgm");
        TestTWD4.twd4("serrano.pgm");
        TestTWD4.twd4("tulips.pgm");
    }
}
```

TWD6.java

```
public class TWD6 {

    public int lena64(double[][] copy, double alfa, double beta) {
        String filePath = System.getProperty("user.dir") +
        "\\lena_twd.pgm";
        FI6 flena = new FI6(copy, filePath, alfa, beta);
        int ea = flena.energiaAcumulada(copy);
        return ea;
    }

    public int twd6(String fileName, double alfa, double beta) {
        String filePath = System.getProperty("user.dir") + "\\" +
        fileName;
        double[][] copy = Pgm.leeP5(filePath);
        FI6 pgmData = new FI6(copy, null, alfa, beta);
        int ea = pgmData.energiaAcumulada(copy);
        return ea;
    }

}

class FI6 {

    double[][] copy;
    double[] h6;
    double[] g6;
    double[] Ih6;
    double[] Ig6;

    void AlfaBetaToHs(double alfa6, double beta6) {
        int filtersize = 6;
        double p6 = 0.5 * Math.sqrt(1 + Math.sin(alfa6 + Math.PI / 4.0));
        double[] hr6 = new double[filtersize];

        hr6[0] = 1 / 8.0 + Math.cos(alfa6) / (4 * Math.sqrt(2)) + (p6 /
                 2.0) * Math.cos(beta6);
        hr6[1] = 1 / 8.0 + Math.sin(alfa6) / (4 * Math.sqrt(2)) + (p6 /
                 2.0) * Math.sin(beta6);
        hr6[2] = 1 / 4.0 - Math.cos(alfa6) / (2 * Math.sqrt(2));
        hr6[3] = 1 / 4.0 - Math.sin(alfa6) / (2 * Math.sqrt(2));
        hr6[4] = 1 / 8.0 + Math.cos(alfa6) / (4 * Math.sqrt(2)) - p6 /
```

```
        2.0 * Math.cos(beta6);
        hr6[5] = 1 / 8.0 + Math.sin(alfa6) / (4 * Math.sqrt(2)) - p6 /
        2.0 * Math.sin(beta6);
        h6[0] = hr6[0];
        h6[1] = hr6[1];
        h6[2] = hr6[2];
        h6[3] = hr6[3];
        h6[4] = hr6[4];
        h6[5] = hr6[5];
        hr6 = null;
    }

    public int energiaAcumulada(double[][] a) {
        double Ef = 0;
        int x = a[0].length;
        int y = a.length;
        double[] Ls = new double[x * y];
        int ii = 0;
        for (int jy = 0; jy < y; jy++) {
            for (int ix = 0; ix < x; ix++) {
                Ls[ii++] = a[jy][ix];
                Ef += a[jy][ix] * a[jy][ix];
            }
        }
        for (ii = 0; ii < x * y - 1; ii++) {
            for (int jj = ii + 1; jj < x * y; jj++) {
                if (Ls[ii] * Ls[ii] < Ls[jj] * Ls[jj]) {
                    //< de mayor a menor
                    double tmp = Ls[ii];
                    Ls[ii] = Ls[jj];
                    Ls[jj] = tmp;
                }
            }
        }

        double Ep = 0.0;
        int lai = x * y - 1; //peor caso

        for (ii = 0; ii < x * y; ii++) {
            Ep += Ls[ii] * Ls[ii];
            if (Ep / Ef <= 1.0 && Ep / Ef > 0.999999) {
                lai = ii;
                break;
            }
        }
        int ceros = 0;
        if (lai - 1 < 0) {
            lai = 4095;
            System.out.println("negativo " + (lai - 1));
        } else {
            double Th = Math.sqrt(Ls[lai] * Ls[lai]);
        }
        Ls = null;
        ceros = x * y - (lai + 1);

        return ceros;
    }
```

```
public FI6(double[][] copy, String filePath, double alfa, double
          beta) {
    this.copy = copy;
    h6 = new double[6];
    g6 = new double[6];
    Ih6 = new double[6];
    Ig6 = new double[6];


    AlfaBetaToHs(alfa, beta);
    calculahs6(6);
    calculaihs6(6);
    TDWLevels(copy);
}

public void TDWLevels(double[][] a) {
    int level = 3;
    int y = a[0].length;
    int x = a.length;
    int filtersize6 = 6;
    for (int nivel = 0; nivel < level; nivel++) {
        int n = y >> nivel;
        for (int jy = 0; jy < n; jy++) {
        //Dimension vertical TRANSFORMACION VERTICAL
            double[] arreglo = new double[n];
            for (int ix = 0; ix < n; ix++) {
                arreglo[ix] = a[jy][ix];
            }
            twd6(arreglo, n, filtersize6);
            for (int ix = 0; ix < n; ix++) {
                a[jy][ix] = arreglo[ix];
            }
            arreglo = null;
        }

        for (int ix = 0; ix < n; ix++) {
        //Dimension horizontal TRANSFORMACION HORIZONTAL
            double[] arreglo = new double[n];
            for (int jy = 0; jy < n; jy++) {
                arreglo[jy] = a[jy][ix];
            }
            twd6(arreglo, n, filtersize6);
            for (int jy = 0; jy < n; jy++) {
                a[jy][ix] = arreglo[jy];
            }
            arreglo = null;
        }
    }

}
```

```
public void ITWDLevels(double[][] a) {
    int level = 3;
    int y = 512;
    int x = 512;
    int filtersize6 = 6;
    for (int nivel = level - 1; nivel >= 0; nivel--) {
        int n = y >> nivel;

        for (int ix = 0; ix < n; ix++) {
        //Dimension vertical TRANSFORMACION VERTICAL
            double[] arreglo = new double[n];
            for (int jy = 0; jy < n; jy++) {
                arreglo[jy] = a[jy][ix];
            }
            invtwd6(arreglo, n, filtersize6);
            for (int jy = 0; jy < n; jy++) {
                a[jy][ix] = arreglo[jy];
            }
            arreglo = null;
        }

        for (int jy = 0; jy < n; jy++) {
        //Dimension vertical TRANSFORMACION VERTICAL
            double[] arreglo = new double[n];
            for (int ix = 0; ix < n; ix++) {
                arreglo[ix] = a[jy][ix];
            }
            invtwd6(arreglo, n, filtersize6);
            for (int ix = 0; ix < n; ix++) {
                a[jy][ix] = arreglo[ix];
            }
            arreglo = null;
        }
    }
}

void error(double[][] a, double[][] ori) {
    int x = 512;
    int y = 512;
    double rms = 0.0;

    for (int jy = 0; jy < y; jy++) {
        for (int ix = 0; ix < x; ix++) {
            rms += (a[jy][ix] - ori[jy][ix]) * (a[jy][ix] -
            ori[jy][ix]);
        }
    }

    rms = rms / ((double) x * y);//(rowmax-rowmin+1));
    rms = Math.sqrt(rms);
    System.out.println("RMS = " + rms);
}
```

```
    void calculahs6(int filtersize6) {
        for (int i = 0; i < filtersize6; i++) {
            if (i % 2 == 0) {
                g6[i] = h6[filtersize6 - 1 - i];
            } else {
                g6[i] = -h6[filtersize6 - 1 - i];
            }
            //Equivalente g[i]=pow(-1,i)*h6[filtersize-1-i];
        }
    }

    void calculaihs6(int filtersize6) {
        for (int i = 0; i < filtersize6; i++) {
            int j = filtersize6 - 1 - i;
            int j2 = (int) (j / 2); // 2*(int)((8-1-i)/2)

            if (i % 2 == 0) { //pares h, impares g
                Ih6[i] = h6[2 * j2];
                Ig6[i] = h6[2 * j2 + 1];
            } else {
                Ih6[i] = g6[2 * j2];
                Ig6[i] = g6[2 * j2 + 1];
            }
        }
    }

    //usa filtros como apuntadores h6[i]
    void twd6(double[] a, int n, int filtersize6) {
    /* "a" es el arreglo unidimensional, n es la longitud del vector a
transformar*/
        if (n >= filtersize6) {
            int i, j;
            int ndiv2 = n / 2; // n entre 2
            double[] tmp = new double[n];
            i = 0;

            for (j = 0; j <= n - 2; j = j + 2) {
                tmp[i] = 0;
                tmp[i + ndiv2] = 0;
                for (int z = 0; z < filtersize6; z++) {
                    tmp[i] += a[(j + z) % n] * h6[z];
                    tmp[i + ndiv2] += a[(j + z) % n] * g6[z];
                }
                i++;
            }
            for (i = 0; i < n; i++) {
                a[i] = tmp[i];
            }
            tmp = null;
        }
    }
```

```
    void invtwd6(double[] a, int n, int filtersize6) {
        /* "a" es el arreglo unidimensional, n es la longitud del vector
recuperado*/
        if (n >= filtersize6) {
            int i, j;
            int ndiv2 = n / 2; // n entre 2
            double[] tmp = new double[n];
            j = filtersize6 - 2;
            for (i = 0; i <= ndiv2 - 1; i++) {
                tmp[(j) % n] = 0;
                for(int w = 0, q = 0; q < filtersize6 / 2; q++, w += 2) {
                    int ii = (i + w / 2) % ndiv2;
                    tmp[(j) % n] += a[ii] * Ih6[w] + a[ii + ndiv2] *
                    Ih6[w + 1];
                }
                j++;
                tmp[(j) % n] = 0;
                for (int w = 0; w < filtersize6; w += 2) {
                    tmp[(j) % n] += a[(i + w / 2) % ndiv2] * Ig6[w] +
                    a[(i + w / 2) % ndiv2 + ndiv2] * Ig6[w + 1];
                }
                j++;
            }
            for (i = 0; i < n; i++) {
                a[i] = tmp[i];
            }
            tmp = null;
        }
    } // invtransform
}
```

TestTWD6.java

```
public class TestTWD6 {

    // transformada wavelet discreta sobre imagenes de 512x512 pixeles
    public static void twd6(String fileName) {
        double alfa = 1.78508070;
        double beta = 1.07424683;
        TWD6 pgm = new TWD6();
        int ea = pgm.twd6(fileName, alfa, beta);
        System.out.println("file: " + fileName);
        System.out.println("alfa: " + alfa);
        System.out.println("beta: " + beta);
        System.out.println("ceros: " + ea);
        System.out.println("");
```

```
        try {
            String path = System.getProperty("user.dir");
            FileWriter file = new FileWriter(new File(path, "twd6" +
            ".txt"), true);
            file.write("file: " + fileName + "\n");
            file.write("alfa: " + alfa + "\n");
            file.write("beta: " + beta + "\n");
            file.write("ceros: " + ea + "\n\n");
            file.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
        TestTWD6.twd6("clegg.pgm");
        TestTWD6.twd6("frymire.pgm");
        TestTWD6.twd6("lena.pgm");
        TestTWD6.twd6("monarch.pgm");
        TestTWD6.twd6("peppers.pgm");
        TestTWD6.twd6("sail.pgm");
        TestTWD6.twd6("serrano.pgm");
        TestTWD6.twd6("tulips.pgm");
    }
}
```

TestTWD6_optimizados.java

```
public class TestTWD6_optimizados {

    // transformada wavelet discreta sobre imagenes de 512x512 pixeles
    public static void twd6_mejor(String fileName, double alfa, double
    beta) {
        TWD6 pgm = new TWD6();
        int ea = pgm.twd6(fileName, alfa, beta);
        System.out.println("file: " + fileName);
        System.out.println("alfa: " + alfa);
        System.out.println("beta: " + beta);
        System.out.println("ceros: " + ea);
        System.out.println("");
        try {
            String path = System.getProperty("user.dir");
            FileWriter file = new FileWriter(new File(path, "twd6_mejor"
            + ".txt"), true);
            file.write("file: " + fileName + "\n");
            file.write("alfa: " + alfa + "\n");
            file.write("beta: " + beta + "\n");
            file.write("ceros: " + ea + "\n\n");
            file.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

```
    public static void main(String[] args) {
        // twd6 con filtros optimizados para imaganes de 64x64 pixeles
        // con los mejores filtros obtenidos evolutivamente
        twd6_mejor("clegg.pgm", 2.398218579925798, 1.6311084841810302);
        twd6_mejor("frymire.pgm", 3.9253434284958484, 4.109907531258745);
        twd6_mejor("lena.pgm", 1.2485241197436663, 0.984708173607773);
        twd6_mejor("monarch.pgm", 1.3667110214706648,
        1.0287653541256685);
        twd6_mejor("peppers.pgm", 6.14206797039259, 3.5560471358563848);
        twd6_mejor("sail.pgm", 5.991046887166441, 3.5829858687033185);
        twd6_mejor("serrano.pgm", 3.961174559334411, 1.3959662391765448);
        twd6_mejor("tulips.pgm", 4.286003583144051, 3.862738318040813);
    }
}
```

DifferentialEvolution.java

```
public class DifferentialEvolution {

    private int dimensions;
    private Individual[] population;
    private int populationSize;
    private int maximumIterations;
    private double standardDeviation;
    private double scaleFactor;
    private double[] allFitness;
    private double bestSolutionFitness;
    private double[] bestSolution;
    private double recombinationProbability;
    private IProblem problem;
    private boolean delayExecution;


    public DifferentialEvolution(int populationSize, int
           maximumIterations, double standardDeviation,
           double scaleFactor, double recombinationProbability, IProblem
           problem,
           boolean delayExecution) {

        this.dimensions = problem.getDimensionsNumber();
        this.populationSize = populationSize;
        this.maximumIterations = maximumIterations;
        this.standardDeviation = standardDeviation;
        this.scaleFactor = scaleFactor;
        this.recombinationProbability = recombinationProbability;
        this.problem = problem;
        this.delayExecution = delayExecution;
        population = new Individual[populationSize];
        allFitness = new double[populationSize];
    }
```

```java
/**
 * se ejecuta hasta cumplir el criterio de paro
 */
public void run() {
    // toma el timepo de inicio
    double Ti, Tf, Tt;
    Ti = System.currentTimeMillis();

    init();
    //System.out.println("Tudo pronto...");
    //System.exit(0);
    double currentStandardDeviation;
    for (int i = 0; i < maximumIterations; i++) {
        iterate();
        currentStandardDeviation =
        Statistics.getStandardDeviation(allFitness);
        if (currentStandardDeviation < standardDeviation) {
            break;
        }
    }
    // toma el tiempo después de encontrar una solución optima
    Tf = System.currentTimeMillis();
    Tt = Tf - Ti; // obtiene el tiempo de ejecucion
    // envia la salida a un arvhivo de texto
    System.out.println("Name: " + problem.getName());
    System.out.println("Best solution: " + bestSolutionFitness);
    for (int i = 0; i < bestSolution.length; i++) {
        System.out.println("solution[" + i + "]:" + bestSolution[i]);
    }
    // envía la salida a un arvhivo de texto
    System.out.println("Tiempo de ejecución: " + Tt / 1000 + "seg ::
    " + (Tt / 1000) / 60 + "min");
    System.out.println("");
    try {
        String path = System.getProperty("user.dir");
        FileWriter file = new FileWriter(new File(path, "resultset" +
        ".txt"), true);
        file.write("Name: " + problem.getName() + "\n");
        file.write("Best solution: " + bestSolutionFitness + "\n");
        for (int i = 0; i < bestSolution.length; i++) {
            file.write("solution[" + i + "]:" + bestSolution[i] +
            "\n");
        }
        file.write("Tiempo de ejecución: " + Tt / 1000 + "seg :: " +
        (Tt / 1000) / 60 + "min\n");
        file.write("\n");
        file.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}

public IProblem getProblem() {
    return problem;
}
```

```
    // Inicializa el algoritmo
    private void init() {

        for (int i = 0; i < populationSize; i++) {
            createIndividual(i);
        }

        bestSolutionFitness = population[0].getSolutionFitness();
        for (Individual individual : population) {
            calculateBestSolution(individual);
        }
    }

    private void createIndividual(int i) {
        double[] initialSolution;
        population[i] = new Individual(dimensions, true);
        initialSolution = getRandomSolution();
        population[i].updateSolution(initialSolution,
        problem.getFitness(initialSolution));
        allFitness[i] = population[i].getSolutionFitness();
    }
    // Realiza las iteraciones del algrotimo
    private void iterate() {
        Individual experimentalIndividual;
        double[] recombinationIndividualSolution;
        double recombinationIndividualSolutionFitness;

        for (int i = 0; i < populationSize; i++) {
            experimentalIndividual = mutation(i);
            recombinationIndividualSolution = crossover(population[i],
            experimentalIndividual);
            recombinationIndividualSolutionFitness =
            problem.getFitness(recombinationIndividualSolution);

            // Si el resultado del individuo experimental esta fuera
            // de lo esperado, se crea un nuevo individuo
            If
(!problem.verifyConstraints(recombinationIndividualSolution)) {
                createIndividual(i);
            } else if
(problem.compareFitness(population[i].getSolutionFitness(),
                    recombinationIndividualSolutionFitness)) {

population[i].updateSolution(recombinationIndividualSolution.clone(),
                        recombinationIndividualSolutionFitness);
                allFitness[i] = recombinationIndividualSolutionFitness;
                calculateBestSolution(population[i]);
            }
        }
        try {
            if (delayExecution) {
                Thread.sleep(250);
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
```

```
    // face de mutacion para el vector experimental
    private Individual mutation(int currentIndividualIndex) {
        int targetIndividualIndex =
getRandomIndex(currentIndividualIndex, -1, -1);
        int individualAIndex = getRandomIndex(currentIndividualIndex,
targetIndividualIndex, -1);
        int individualBIndex = getRandomIndex(currentIndividualIndex,
targetIndividualIndex, individualAIndex);

        double[] experimentalSolution = new double[dimensions];
        double[] targetIndividualSolution =
population[targetIndividualIndex].getSolution();
        double[] individualASolution =
population[individualAIndex].getSolution();
        double[] individualBSolution =
population[individualBIndex].getSolution();

        Individual experimentalIndividual = new Individual(dimensions,
true);
        double position;

        for (int i = 0; i < dimensions; i++) {
            position = targetIndividualSolution[i] + scaleFactor *
(individualASolution[i] - individualBSolution[i]);
            experimentalSolution[i] =
trimPositionToProblemLimits(position, i);
            //experimentalSolution[0]=Math.PI/4;
        }

        experimentalIndividual.updateSolution(experimentalSolution,
problem.getFitness(experimentalSolution));

        return experimentalIndividual;
    }

    // Face de cruzamiento del vector experimental y el individuo actual
    private double[] crossover(Individual currentIndividual, Individual
experimentalIndividual) {
        Random random = new Random(System.nanoTime());
        double[] recombinationIndividualSolution =
currentIndividual.getSolution().clone();
        double[] experimentalIndividualSolution =
experimentalIndividual.getSolution();
        double randomValue;

        for (int i = 0; i < dimensions; i++) {
            randomValue = random.nextDouble();
            if (randomValue <= recombinationProbability) {
                recombinationIndividualSolution[i] =
experimentalIndividualSolution[i];
            }
        }

        return recombinationIndividualSolution;
    }
```

```
    // devuelve un índice aleatorio entre 0 y el numero de
    // dimesiones del problema actal
    private int getRandomIndex(int currentIndividualIndex, int
individualAIndex, int individualBIndex) {
        Random random = new Random(System.nanoTime());
        int randomIndex = random.nextInt(populationSize);

        while (randomIndex == currentIndividualIndex || randomIndex ==
individualAIndex
                || randomIndex == individualBIndex) {
            randomIndex = random.nextInt(populationSize);
        }

        return randomIndex;
    }

    private void calculateBestSolution(Individual individual) {
        if (problem.compareFitness(bestSolutionFitness,
individual.getSolutionFitness())) {
            bestSolutionFitness = individual.getSolutionFitness();
            bestSolution = individual.getSolution().clone();
        }
    }

    private double trimPositionToProblemLimits(double position, int
dimension) {
        double retorno = position;

        if (retorno > problem.getUpperLimit(dimension)) {
            retorno = problem.getUpperLimit(dimension);
        } else if (retorno < problem.getLowerLimit(dimension)) {
            retorno = problem.getLowerLimit(dimension);
        }

        return retorno;
    }

    private double[] getRandomSolution() {
        double[] position = new double[dimensions];
        Random random = new Random(System.nanoTime());
        do {
            for (int i = 0; i < dimensions; i++) {
                double rightBound, leftBound, value =
                random.nextDouble();
                rightBound = problem.getUpperLimit(i);
                leftBound = problem.getLowerLimit(i);
                value = leftBound + (rightBound - leftBound) * value;
                position[i] = value;
                if (position[i] > rightBound) {
                    position[i] = rightBound;
                }
                if (position[i] < leftBound) {
                    position[i] = leftBound;
                }
            }
        } while (!problem.verifyConstraints(position));
        return position;}}
```

Individual.java

```
public class Individual {
	private double[] solution;
	private int dimensions;
	private double solutionFitness;

	public Individual(int dimensions, boolean flag) {
		this.dimensions = dimensions;
		this.solution = new double[dimensions];
		    if(flag){
			for (int i = 0; i < solution.length; i++) {
				solution[i]= Math.PI/4;
			}
		    }
	}

	public double[] getSolution() {
		return solution;
	}

	public void updateSolution(double[] solution, double
solutionFitness) {
		this.solution = solution;
		this.solutionFitness = solutionFitness;
	}

	public double getSolutionFitness() {
		//System.out.println("this.solutionFitness =" +
this.solutionFitness);
		return this.solutionFitness;
	}
}
```

Statistics.java

```
public class Statistics {

	public static double getStandardDeviation(double[] data) {
		return Math.sqrt(getVariance(data));
	}

	public static double getVariance(double[] data) {
		double media = getArithmeticAverage(data);
		double sum1 = 0, sum2 = 0;

		for (double x : data) {
			sum1 += Math.pow((x - media), 2);
			sum2 += (x - media);
		}
		return (sum1 - (Math.pow(sum2, 2) / data.length)) /
		(data.length - 1);
	}
```

```
	public static double getArithmeticAverage(double[] data) {
		double sum = 0;
		for (double x : data) {
			sum += x;
		}
		return (sum / data.length);
	}

}
```

IProblem.java interface

```
public interface IProblem {
	static final double MINIMUM_DIMENSION_VALUE = 0.01;

	/**
	 * retorna el nombre del problema
	 */
	String getName();

	/**
	 * retorna el numero de dimensiones del problema
	 */
	int getDimensionsNumber();

	/**
	 * retorna el limite inferior del espacio de búsqueda para el
	 * problema en cuestion
	 */
	double getLowerLimit(int dimension);

	/**
	 * retorna el limite superior del espacio de búsqueda para el
	 * problema en cuestion
	 */
	double getUpperLimit(int dimension);

	/**
	 * compara la mejor solución, si la mejor es la que es la actual
	 * devuelve true, de lo contrario devuelve false
	 */
	boolean compareFitness(double bestSolutionFitness, double
currentSolutionFitness);

	/**
	 * calcula el fitnes de la solución dada
	 */
	double getFitness(double ... solution);

	public boolean verifyConstraints(double... variables);
}
```

TWD6Clegg.java

```
public class TWD6Clegg implements IProblem {

    static String filePath = System.getProperty("user.dir") +
    "\\clegg64.pgm";
    static final double[][] data = Pgm.leeP5(filePath);


    public String getName() {
        return "clegg64";
    }
    public int getDimensionsNumber() {
        return 2;
    }

    public double getLowerLimit(int dimension) {
        return 0;
    }

    public double getUpperLimit(int dimension) {
        return 2 * Math.PI;
    }

    public boolean compareFitness(double pBestFitness, double
currentPositionFitness) {
        return currentPositionFitness > pBestFitness;
    }

    public double getFitness(double... dimension) {
        double result = 0;
        double alfa =  dimension[0]; //Math.PI / 4;
        double beta =  dimension[1];
        //result = Math.abs( alfa-beta);
        TWD6 tw = new TWD6();
        double[][] copy = new double[data.length][data[0].length];

        for (int i = 0; i < copy.length; i++) {
            for (int j = 0; j < copy[0].length; j++) {
                copy[i][j] = data[i][j];
            }
        }
        result = tw.lena64(copy, alfa, beta);
        copy = null;
        return result;
    }

    public boolean verifyConstraints(double... variables) {
        return true;
    }
}
```

TWD6Frymire.java

```
public class TWD6Frymire implements IProblem {

    static String filePath = System.getProperty("user.dir") +
    "\\frymire64.pgm";
    static final double[][] data = Pgm.leeP5(filePath);

    public String getName() {
        return "frymire64";
    }

    public int getDimensionsNumber() {
        return 2;
    }

    public double getLowerLimit(int dimension) {
        return 0;
    }

    public double getUpperLimit(int dimension) {
        return 2 * Math.PI;
    }

    public boolean compareFitness(double pBestFitness, double
    currentPositionFitness) {
        return currentPositionFitness > pBestFitness;
    }

    public double getFitness(double... dimension) {
        double result = 0;
        double alfa =  dimension[0]; //Math.PI / 4;
        double beta =  dimension[1];
        //result = Math.abs( alfa-beta);
        TWD6 tw = new TWD6();
        double[][] copy = new double[data.length][data[0].length];

        for (int i = 0; i < copy.length; i++) {
            for (int j = 0; j < copy[0].length; j++) {
                copy[i][j] = data[i][j];
            }
        }
        result = tw.lena64(copy, alfa, beta);
        copy = null;
        return result;
    }

    public boolean verifyConstraints(double... variables) {
        return true;
    }
}
```

TWD6Lena. Java

```
public class TWD6Lena implements IProblem {

    static String filePath = System.getProperty("user.dir") +
    "\\lena64.pgm";
    static final double[][] data = Pgm.leeP5(filePath);

    public String getName() {
        return "lena64";
    }

    public int getDimensionsNumber() {
        return 2;
    }

    public double getLowerLimit(int dimension) {
        return 0;
    }

    public double getUpperLimit(int dimension) {
        return 2 * Math.PI;
    }

    public boolean compareFitness(double pBestFitness, double
currentPositionFitness) {
        return currentPositionFitness > pBestFitness;
    }

    public double getFitness(double... dimension) {
        double result = 0;
        double alfa =  dimension[0]; //Math.PI / 4;
        double beta =  dimension[1];
        //result = Math.abs( alfa-beta);
        TWD6 tw = new TWD6();
        double[][] copy = new double[data.length][data[0].length];

        for (int i = 0; i < copy.length; i++) {
            for (int j = 0; j < copy[0].length; j++) {
                copy[i][j] = data[i][j];
            }
        }
        result = tw.lena64(copy, alfa, beta);
        copy = null;
        return result;
    }

    public boolean verifyConstraints(double... variables) {
        return true;
    }
}
```

TWD6Monarch.java

```
public class TWD6Monarch implements IProblem {

    static String filePath = System.getProperty("user.dir") +
    "\\monarch64.pgm";
    static final double[][] data = Pgm.leeP5(filePath);

    public String getName() {
        return "monarch64";
    }

    public int getDimensionsNumber() {
        return 2;
    }

    public double getLowerLimit(int dimension) {
        return 0;
    }

    public double getUpperLimit(int dimension) {
        return 2 * Math.PI;
    }

    public boolean compareFitness(double pBestFitness, double
currentPositionFitness) {
        return currentPositionFitness > pBestFitness;
    }

    public double getFitness(double... dimension) {
        double result = 0;
        double alfa =  dimension[0]; //Math.PI / 4;
        double beta =  dimension[1];
        //result = Math.abs( alfa-beta);
        TWD6 tw = new TWD6();
        double[][] copy = new double[data.length][data[0].length];

        for (int i = 0; i < copy.length; i++) {
            for (int j = 0; j < copy[0].length; j++) {
                copy[i][j] = data[i][j];
            }
        }
        result = tw.lena64(copy, alfa, beta);
        copy = null;
        return result;
    }

    public boolean verifyConstraints(double... variables) {
        return true;
    }
}
```

TWD6Peppers.java

```
public class TWD6Peppers implements IProblem {

    static String filePath = System.getProperty("user.dir") +
    "\\peppers64.pgm";
    static final double[][] data = Pgm.leeP5(filePath);

    public String getName() {
        return "peppers64";
    }

    public int getDimensionsNumber() {
        return 2;
    }

    public double getLowerLimit(int dimension) {
        return 0;
    }

    public double getUpperLimit(int dimension) {
        return 2 * Math.PI;
    }

    public boolean compareFitness(double pBestFitness, double
currentPositionFitness) {
        return currentPositionFitness > pBestFitness;
    }

    public double getFitness(double... dimension) {
        double result = 0;
        double alfa =  dimension[0]; //Math.PI / 4;
        double beta =  dimension[1];
        //result = Math.abs( alfa-beta);
        TWD6 tw = new TWD6();
        double[][] copy = new double[data.length][data[0].length];

        for (int i = 0; i < copy.length; i++) {
            for (int j = 0; j < copy[0].length; j++) {
                copy[i][j] = data[i][j];
            }
        }
        result = tw.lena64(copy, alfa, beta);
        copy = null;
        return result;
    }

    public boolean verifyConstraints(double... variables) {
        return true;
    }
}
```

TWD6Sail.java

```
public class TWD6Sail implements IProblem {

    static String filePath = System.getProperty("user.dir") +
    "\\sail64.pgm";
    static final double[][] data = Pgm.leeP5(filePath);

    public String getName() {
        return "sail64";
    }

    public int getDimensionsNumber() {
        return 2;
    }

    public double getLowerLimit(int dimension) {
        return 0;
    }

    public double getUpperLimit(int dimension) {
        return 2 * Math.PI;
    }

    public boolean compareFitness(double pBestFitness, double
currentPositionFitness) {
        return currentPositionFitness > pBestFitness;
    }

    public double getFitness(double... dimension) {
        double result = 0;
        double alfa =  dimension[0]; //Math.PI / 4;
        double beta =  dimension[1];
        //result = Math.abs( alfa-beta);
        TWD6 tw = new TWD6();
        double[][] copy = new double[data.length][data[0].length];

        for (int i = 0; i < copy.length; i++) {
            for (int j = 0; j < copy[0].length; j++) {
                copy[i][j] = data[i][j];
            }
        }
        result = tw.lena64(copy, alfa, beta);
        copy = null;
        return result;
    }

    public boolean verifyConstraints(double... variables) {
        return true;
    }
}
```

TWD6Serrano.java

```
public class TWD6Serrano implements IProblem {

    static String filePath = System.getProperty("user.dir") +
    "\\serrano64.pgm";
    static final double[][] data = Pgm.leeP5(filePath);

    public String getName() {
        return "serrano64";
    }

    public int getDimensionsNumber() {
        return 2;
    }

    public double getLowerLimit(int dimension) {
        return 0;
    }

    public double getUpperLimit(int dimension) {
        return 2 * Math.PI;
    }

    public boolean compareFitness(double pBestFitness, double
currentPositionFitness) {
        return currentPositionFitness > pBestFitness;
    }

    public double getFitness(double... dimension) {
        double result = 0;
        double alfa =  dimension[0]; //Math.PI / 4;
        double beta =  dimension[1];
        //result = Math.abs( alfa-beta);
        TWD6 tw = new TWD6();
        double[][] copy = new double[data.length][data[0].length];

        for (int i = 0; i < copy.length; i++) {
            for (int j = 0; j < copy[0].length; j++) {
                copy[i][j] = data[i][j];
            }
        }
        result = tw.lena64(copy, alfa, beta);
        copy = null;
        return result;
    }

    public boolean verifyConstraints(double... variables) {
        return true;
    }
}
```

TWD6Tulips.java

```
public class TWD6Tulips implements IProblem {

    static String filePath = System.getProperty("user.dir") +
    "\\tulips64.pgm";
    static final double[][] data = Pgm.leeP5(filePath);


   public String getName() {
        return "tulips64";
    }

    public int getDimensionsNumber() {
        return 2;
    }

    public double getLowerLimit(int dimension) {
        return 0;
    }

    public double getUpperLimit(int dimension) {
        return 2 * Math.PI;
    }

    public boolean compareFitness(double pBestFitness, double
currentPositionFitness) {
        return currentPositionFitness > pBestFitness;
    }

    public double getFitness(double... dimension) {
        double result = 0;
        double alfa =  dimension[0]; //Math.PI / 4;
        double beta =  dimension[1];
        //result = Math.abs( alfa-beta);
        TWD6 tw = new TWD6();
        double[][] copy = new double[data.length][data[0].length];

        for (int i = 0; i < copy.length; i++) {
            for (int j = 0; j < copy[0].length; j++) {
                copy[i][j] = data[i][j];
            }
        }
        result = tw.lena64(copy, alfa, beta);
        copy = null;
        return result;
    }

    public boolean verifyConstraints(double... variables) {
        return true;
    }
}
```

Clase principal invoca las clases anteriores para evolucionar los filtros paramétricos en cada una de las imágenes aplicando la transformada wavelet discreta.

Main.java

```
public class Main {

    static void all(int populationSize, int maximumIterations, double
standardDeviation,
            double scaleFactor, double recombinationProbability) {
        /*clegg*/
        new DifferentialEvolution(populationSize, maximumIterations,
standardDeviation,
                scaleFactor, recombinationProbability, new TWD6Clegg(),
false).run();
        /*Frymire*/
        new DifferentialEvolution(populationSize, maximumIterations,
standardDeviation,
                scaleFactor, recombinationProbability, new TWD6Frymire(),
false).run();
        /*Lena*/
        new DifferentialEvolution(populationSize, maximumIterations,
standardDeviation,
                scaleFactor, recombinationProbability, new TWD6Lena(),
false).run();
        /*Monarch*/
        new DifferentialEvolution(populationSize, maximumIterations,
standardDeviation,
                scaleFactor, recombinationProbability, new TWD6Monarch(),
false).run();
        /*Peppers*/
        new DifferentialEvolution(populationSize, maximumIterations,
standardDeviation,
                scaleFactor, recombinationProbability, new TWD6Peppers(),
false).run();
        /*Sail*/
        new DifferentialEvolution(populationSize, maximumIterations,
standardDeviation,
                scaleFactor, recombinationProbability, new TWD6Sail(),
false).run();
        /*Serrano*/
        new DifferentialEvolution(populationSize, maximumIterations,
standardDeviation,
                scaleFactor, recombinationProbability, new TWD6Serrano(),
false).run();
        /*Tulips*/
        new DifferentialEvolution(populationSize, maximumIterations,
standardDeviation,
                scaleFactor, recombinationProbability, new TWD6Tulips(),
false).run();

    }

    public static void main(String[] args) {
        // inicializamos las condiciones para el algoritmo elvolutivo
        int populationSize = 10;
        int maximumIterations = 1000;
        double standardDeviation = 0.003;
```

```
        double scaleFactor = 2;
        double recombinationProbability = 0.6;

        try {
            // cabecera del arvhivo que contiene el conjunto de
            // resultados
            String path = System.getProperty("user.dir");
            FileWriter file = new FileWriter(new File(path, "resultset" +
            ".txt"), true);
            file.write("populationSize: " + populationSize + "\n");
            file.write("maximumIterations: " + maximumIterations + "\n");
            file.write("standardDeviation: " + standardDeviation + "\n");
            file.write("scaleFactor: " + scaleFactor + "\n");
            file.write("recombinationProbability: " +
            recombinationProbability + "\n\n");
            file.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
        all(populationSize, maximumIterations, standardDeviation,
        scaleFactor, recombinationProbability);
    }
}
```

BIBLIOGRAFÍA

[1] Maher A. Sid Ahmed, *Image Processing: Theory, Algorithms, and Architectures*.McGraw-Hill. 1995.

[2] I. Daubechies, Ten Lectures on Wavelets. Society for Industrial and Applied Mathematics, 1992.

[3] R. Coifman, y. Meyer, s. Quake, and m. V. Wickerhauser, "signal processing and compression with wavelet packets," in *wavelets and their applications*, vol. 442, j. S. Byrnes, j. Byrnes, k. Hargreaves, and k. Berry, eds. Springer netherlands, 1994, pp. 363–379

[4] "Symlet wavelet filter computation - MATLAB symaux." [En línea]. Disponible en: http://www.mathworks.com/help/wavelet/ref/symaux.html. [Accedido: 05-mar-2015].

[5] Emmanuel Candès, Laurent Demanet, David Donoho y Lexing Ying , "Fast Discrete Curvelet Transforms," Applied and Computational Mathematics,2006.

[6] Oscar Herrera y Roman Mora, "Aplicación de Algoritmos Genéticos a la Compresión de Imágenes con Evolets", en *Avances Recientes en Sistemas Inteligentes*, Miguel González Mendoza y Oscar Herrera Alcántara, SMIA, 2011, pp. 157 - 166.

[7] O. Herrera Alcántara y M. González Mendoza, "Optimization of Parameterized Compactly Supported Orthogonal Wavelets for Data Compression", en *Advances in Soft Computing*, vol. 7095, I. Batyrshin y G. Sidorov, Eds. Springer Berlin Heidelberg, 2011, pp. 510-521.

[8] O. Herrera Alcántara, "On the Best Evolutionary Wavelet Based Filter to Compress a Specific Signal", en *Advances in Soft Computing*, vol. 6438, G. Sidorov, A. Hernández Aguirre, y C. Reyes García, Eds. SpringerBerlin Heidelberg, 2010, pp. 394-405.

[9] Claudio Vargas, Oscar. Implementación de una aplicación software para procesamiento de imágenes con wavelets y bancos de filtros paramétricos de reconstrucción perfecta. Oscar Herrera Alcántara.

[10] Garduño Martínez, Javier Adrián. Estudio experimental de la aproximación de filtros digitales paramétricos para implementar la transformada wavelet discreta con polinomios evolutivos, Oscar Herrera Alcántara.

[11] Hernández Nieves, María Sara. Clasificación de llantos de bebé con wavelets. Oscar Herrera Alcántara.

[12] Roach, D. W., “Frequency selective parameterized wavelets of length ten", *Journal of Concrete and Applicable Mathematics*, vol. 8, no. 1, pp. 165-179, 2010.

[13] Xiao, Panrong, "Image Compression by Wavelet Transform." (2001). Electronic Theses and Dissertations. Paper 58. http://dc.etsu.edu/ etd/58

[14] “JPEG - JPEG 2000”. [En línea]. Disponible en: http://www.jpeg.org/jpeg2000/. [Accedido: 06-mar-2015].

[15] “True Compression for Images - MATLAB & Simulink”. [En línea]. Disponible en: http://www.mathworks.com/help/wavelet/ug/true-compression-for-images.html. [Accedido: 05-mar-2015].

[16] “C44”. [En línea]. Disponible en: http://djvu.sourceforge.net/doc/man/c44.html. [Accedido: 05-mar-2015].

[17] “Anexo: Formatos de archivo de gráficos”. [En línea]. Disponible en: http://es.wikipedia.org/wiki/Anexo:Formatos_de_archivo_de_gráficos. [Accedido: 10-abr-2015].

[18] M. Gestal, D. Rivero, J. R. Rabuñal, J. Dorado, and A. Pazos, *Introducción a los algoritmos genéticos y a la programación genética*. Universidade da Coruña, 2010.

[19] H. L. Resnikoff y R. O. N. Wells, Wavelet Analysis: The Scalable Structure of Information : with 92 Figures. Springer New York, 1998.

[20] “Differential Evolution.” [En línea]. Disponible en: https://code.google.com/p/cnde/source/browse/trunk/ED/src/br/upe/dsc/de/algorithm/DifferentialEvolution.java?r=11. [Accedido: 30-Jun-2015].

[21] “Polyomino Compressed Format Benchmarks - The Waterloo image set”. [En línea]. Disponible en: http://www.researchandtechnology.net/pcif/waterloo_benchmarks.php. [Accedido: 10-abr-2015].