DOI: 10.24275/uama.6749.9390

Algoritmos metaheurísticos aplicados a una transformada fractal en imágenes

Tesis presentada a la

Universidad Autónoma Metropolitana

Unidad Azcapotzalco

División de Ciencias Básicas e Ingeniería

Para obtener el grado de:

MAESTRA EN OPTIMIZACIÓN

por

Stephanie Pamela Avila Campos

Asesores de tesis:

Dr. Eduardo Rodríguez Martínez

Mtro. Óscar Alvarado Nava

Ciudad de México, 5 de marzo de 2022

Dedicatoria

*A mi **madre***

Gloria Campos Lara

Mi mayor tesoro y pilar fundamental de mi vida.

Agradecimientos

Quiero expresar mi más sincero y entero agradecimiento al Maestro Óscar Alvarado Nava (Q.E.P.D), asesor de esta tesis, por sus atinados consejos, su gran ayuda, excelente guía, disponibilidad y tiempo para correcciones y recomendaciones que me brindó para la realización de esta tesis, fue una persona base para que este proyecto surgiera, sin su asesoría, no hubiese podido iniciar este proceso, además de la confianza que depositó en mí para esta investigación. También agradezco a mi asesor el Dr. Eduardo Rodríguez Martínez, por sus revisiones, sugerencias y correcciones necesarias para el buen término de este trabajo.

Agradezco a mi madre Gloria Campos por el enorme apoyo y amor que me brinda en todo momento de cualquier forma y modo e incondicionalmente, por ser mi pilar y base pues siempre ha creído en mí, pero sobre todo porque siempre me ha impulsado para lograr todas mis metas; siendo ésta una de ellas. A mis hermanos Fernando Avila y Brenda Avila por apoyarme y animarme en este proceso.

De igual forma quiero agradecer a Saúl Barrera que siempre me ha apoyado en todo momento, por darme ánimos, palabras de aliento, alegría, entendimiento y amor incondicional, por brindarme ideas que me permitieron aclarar las mías en este gran camino.

Agradezco también a mi compañero y amigo de posgrado el Dr. Héctor Ricardo Gómez, pues siempre estuvo presente en esta trayectoria académica apoyándome, guiándome y argumentándome ideas.

De igual forma agradezco a mis revisores, el Dr Juan Villegas, el Dr. Marco Heredia y al Dr Javier Ramírez por sus grandes aportanciones para el mejor entendimiento de este escrito. A su vez agradezco al Consejo Nacional de Ciencia y Tecnología, CONACYT, por el apoyo

económico otorgado, así como también a la Universidad Autónoma Metropolitana - Unidad Azcapotzalco, en especial al Posgrado en Optimización de la División de Ciencias Básicas e Ingeniería, por las facilidades otorgadas para la realización de mis estudios de posgrado. Por último, agradezco a todos los maestros que compartieron conmigo sus conocimientos, ayudando con ello a mi formación profesional.

Índice general

Índice de figuras

Índice de tablas

Abreviaturas

bd Bloques dominio. 32

br Bloques rango. 32

d_H Distancia de Hausdorff. 35

AG Algoritmo genético. 28

MAE Mean Absolute Error (por sus siglas en inglés), Error Medio Absoluto. 36

PSNR Peak Signal-to-Noise Ratio (por sus siglas en inglés), Pico Señal a Ruido. 36

px Píxeles. 3

RMSE Root Mean Squared Error (por sus siglas en inglés), Raíz de Error Cuadrático Medio. 36

SFI Sistema de funciones iteradas. 13

Glosario

Atractor es el conjunto resultante, asociado al SFI, después de iterar sobre un conjunto inicial. 13

Dimensión de Hausdorff es una métrica que permite definir una dimensión fraccionaria (no entera) para un objeto fractal. 19

Fractal es el "conjunto cuya dimensión Hausdorff-Besicovitch excede estrictamente su dimensión topológica" [1]. 18

Geometría fractal es la rama de la geometría que estudia los fractales. 27

Imagen digital es la representación de una imagen en una matriz numérica de dos dimensiones con valores enteros no negativos. 13

Isometría es la aplicación o transformación geométrica que conserva las distancias existentes entre rectas, longitudes y ángulos. 21

Píxel es el elemento de color uniforme más pequeño que forma una imagen digital. Contiene información de color, saturación y brillo y no tiene un tamaño determinado. 2

Resolución es la cantidad de píxeles por unidad de medida que tiene la imagen (píxeles por cm o por pulgada). Normalmente se mide en ppp (puntos por pulgada), en inglés dpi (dots per inch). 4

Tamaño de imagen es la suma de todos los píxeles repartidos en las filas y columnas de la matriz de pixeles que representa una imagen digital. 14

Capítulo 1

Introducción

1.1. Conceptos principales de imágenes digitales

◦ **Píxel (px) e intensidad promedio**

Un Píxel (px) (acrónimo del inglés *picture element, elemento de imagen*), es la menor unidad homogénea en color que forma parte de una imagen digital. En la Figura 1.1 se observan las diferentes tonalidades dependiendo de los bits que se utilicen para codificar la imagen. En particular, para las pruebas de este trabajo de investigación haremos uso de imágenes con 8 bits.

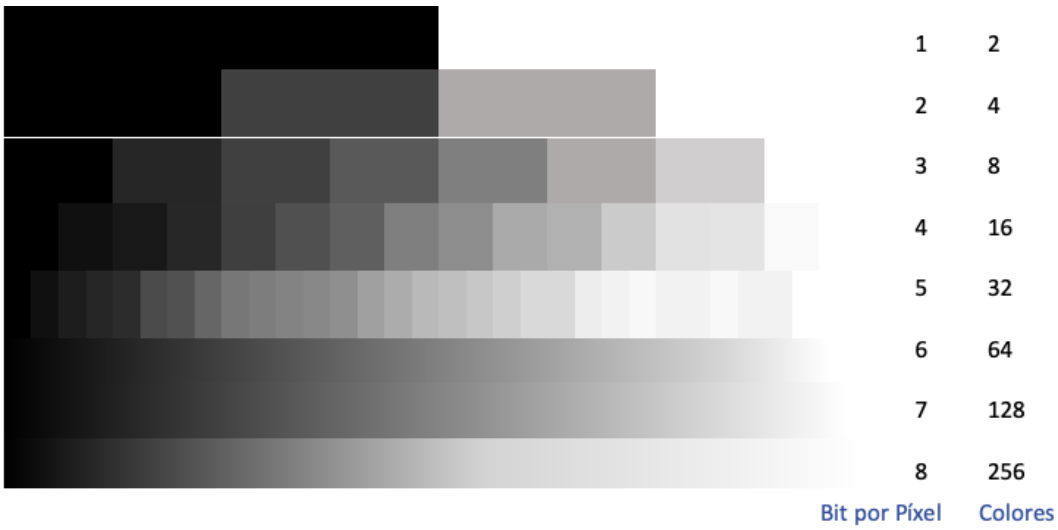


Figura 1.1: Representación de niveles de intensidad de píxel - profundidad de tono de grises.

◦ **Imagen digital**

Una imagen se define como una función de dos dimensiones $f(x,y)$ donde x e y son las coordenadas de un plano que contiene todos los puntos de la misma, y $f(x,y)$ es la amplitud en el punto (x,y) a la cual se le llama intensidad o nivel de gris de la imagen en ese punto. Cuando las coordenadas x e y así como los valores de intensidad de la función f sean discretos y finitos, se habla entonces de una *imagen digital* .

Una imagen digital está compuesta de un número finito de elementos y cada uno tiene una localidad y un valor particular. A estos elementos se les llama *"píxeles"* (px), unidad mínima de medida de una imagen digital [1].

En la Figura 1.2 se muestra una representación de una imagen con 256 niveles de intensidad. En ella, cada uno de los píxeles está representado por un número que $\in \mathbb{Z}$ que es interpretado como el nivel de intensidad luminosa en escala de grises. Ampliando la imagen en una zona cualquiera, se pueden apreciar las tonalidades de los píxeles que están representados por valores discretos, tal como se se muestra en la matriz p.

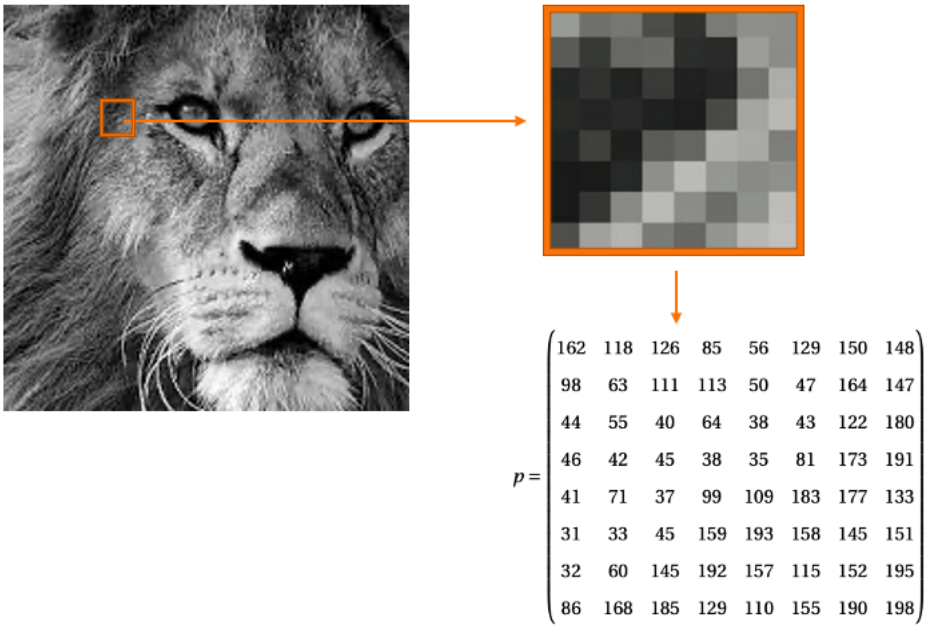
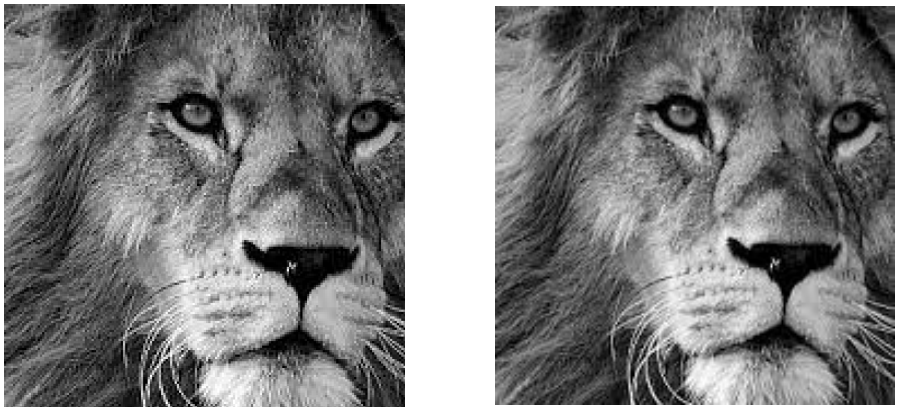


Figura 1.2: Imagen con 256 niveles de intensidad y representación numérica de un fragmento de 8×8 px.

[1] https://dictionary.cambridge.org/es-LA/dictionary/english/pixel

◦ **Resolución de la imagen**

La Resolución es la que nos permite visualizar mejor los detalles de una imagen digital debido a la cantidad de píxeles por unidad de medida que tiene la misma. Esta unidad generalmente se mide en pulgadas, aunque también puede ser en centímetros. Cuanto más píxeles por pulgada (ppp) más resolución tendrá la imagen y por tanto mejor calidad visual. Cuanto menos píxeles por pulgada tenga la imagen, tendrá menor resolución y por lo tanto se mostrará como una cuadrícula en la que los píxeles serán visibles, tal es el ejemplo de las Figuras 1.3.



(a) Imagen con “mala calidad” (b) Imagen con “buena calidad”

Figura 1.3: Diferencia entre resolución buena y mala en imágenes digitales.

Esta medida varía dependiendo del medio final de la imagen, es decir, no se necesitará la misma resolución para una imagen impresa para un anuncio que la imagen para una página web o para una fotografía. En términos generales, una imagen digital con buena resolución dependerá de la cantidad de píxeles utilizados para representarla.

◦ **Calidad de Imagen**

La calidad de una imagen está determinada por la resolución y la profundidad del color. Ambos factores están estrechamente relacionados. Cuando decimos que una imagen presenta “calidad buena” es porque es claro ver los detalles de la misma, la definición de curvas y tonalidades balanceadas de colores, la luz que presenta y la cantidad adecuada de píxeles que se usan para representarla. En cambio cuando hablamos de imágenes con “mala calidad” es porque la imagen se ve con ruido visual o pueden estar muy oscuras que imposibilita la visualización del contenido, suelen tener deficientes píxeles para su representación o incluso

mostrarse borrosas.

Algunas métricas para obtener cuantitativamente la calidad de una imagen digital son:

a) **Entropía:** es el valor que nos proporciona la cantidad de información contenida en la imagen. Está dada por la Ec. (1.1) por lo que una imagen con valor de entropía mayor tendrá una mejor calidad.

b) **Desviación estándar:** denotada por $STD = \frac{1}{n}\sum_1^n \sqrt{I,\mu^2}$ mide el contraste de la imagen I de tamaño n con base en el ruido presente en la misma, por lo que una imagen con un alto contraste tendrá una desviación estándar alta. (μ es la media).

c) **Pico señal a ruido (PSNR):** denotado por la Ec. 2.15 calcula la relación pico señal a ruido en decibelios entre dos imágenes. Esta relación se utiliza como medida de calidad entre la imagen a comparar con respecto a la imagen de referencia, por lo que entre más alto sea éste valor mejor será la calidad de la imagen a comparar.

En el ejemplo de la Figura 1.4 la imagen a) representa una calidad buena debido a que presenta un valor de entropía de 7.65 mientras que la imagen b) podemos decir que presenta una calidad mala debido a que tiene un valor de entropía de 7.07. Así mismo al calcular el valor PSNR de la imagen b) con respecto a la imagen a) como referencia, se obtiene un valor de 14.72, lo cual significa que la calidad de la imagen b) es menor con respecto a la imagen a).

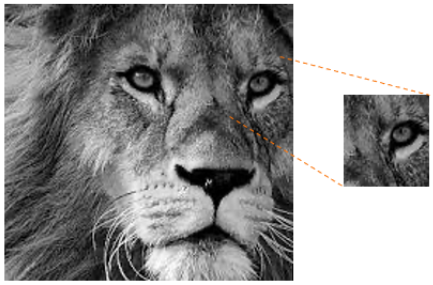
(a) Imagen con “buena calidad”, entropía: 7.65

(b) Imagen con “mala calidad”, entropía: 7.07

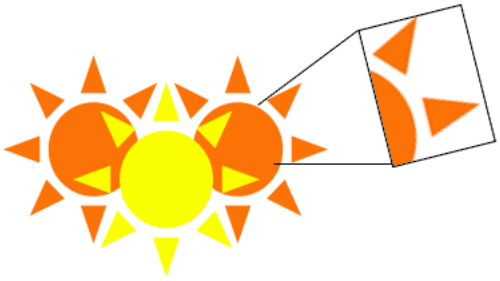
Figura 1.4: Diferencia entre calidad buena y mala en imágenes digitales.

◦ **Tipos de imágenes digitales**

Las podemos clasificar en dos grupos: **Imágenes de mapa de bits** que son imágenes formadas por píxeles e **Imágenes vectoriales** que es una imagen digital formada por objetos geométricos independientes (segmentos, polígonos, arcos, etc.), cada uno de ellos definido por distintos atributos matemáticos de forma, posición, color, etc. Para este trabajo usaremos imágenes digitales en mapa de bits.



(a) Imagen digital en mapa de bits.



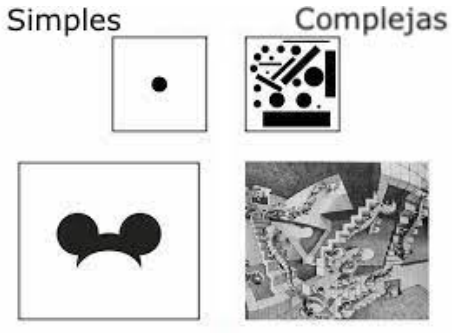
(b) Imagen digital vectorial

Figura 1.5: Ejemplo de imagen digital en mapa de bits vs imagen vectorial.

◦ **Complejidad de una imagen**

Las imágenes digitales se dividen en dos grandes grupos; imágenes de mapa de bits e imágenes vectoriales, sin embargo hay una clasificación más por mencionar que depende del contenido que presentan las imágenes; complejas y simples [2, 3].

Cuando las imágenes presentan complejidad simple, la compresión es más sencilla ya que poseen pocos elementos visuales y su decodificación por ende es más rápida, pues requieren de una menor atención. Por el contrario, cuando la imagen es compleja, su compresión, análisis y decodificación es más compleja ya que requiere mayor nivel de atención (ver Figuras 1.6); es por ello que el problema principal a atender a lo largo del presente trabajo se sustenta en imágenes complejas a escala de grises.



(a) Imágenes a escala de grises.

(b) Imágenes a color.

Figura 1.6: Simplicidad y complejidad en imágenes.

◦ **Métrica para medir la complejidad de una imagen**

Una forma para medir la complejidad de una imagen es mediante el cálculo de su **entropía**. La entropía como tal es una magnitud que se utiliza en la termodinámica para medir los cambios o propiedades de un elemento desde su momento inicial hasta el momento final. Esta métrica indica el grado de desorden de un sistema, producto de una energía que no puede ser controlada y, por lo tanto, no puede ser usada por lo que a mayor desorden, mayor entropía.

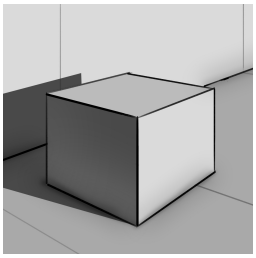
En ese sentido, la entropía aplicada a imágenes es un parámetro que permite medir cuantitativamente la cantidad de información presente en una imagen, es decir, el grado de desorden de los píxeles o nivel de utilización de los diferentes niveles de gris. Así también, como ya se mencionó anteriormente, nos proporciona qué tan buena calidad presenta una imagen.

Dicha métrica no tiene en cuenta la información que proviene de la distribución espacial de los píxeles sino del valor de éstos, es por ello que cuando se calcula el valor de entropía en una imagen con textura uniforme o pocos elementos en ella el valor, será mínimo, teniendo en cuenta que este valor oscila entre 0 y 1. En la Ec. (1.1) se define la forma de calcular la entropía de una imagen.

$$-\sum_{i=0}^{n-1} p_i \log_b p_i \tag{1.1}$$

donde n es el número de tonalidades de grises consideradas (256 para imágenes de 8 bits), p_i es la probabilidad de que un píxel tenga un nivel de gris i. En la Figura 1.7a se observa que la

imagen presenta una entropía de 4.4339, mientras que en la Figura 1.7b la imagen presenta una entropía de 7.7653; con lo que se concluye que una imagen compleja tendrá un valor de entropía muy cercano a 1 debido a los niveles de grises y textura que hay en dicha imagen.



(a) Imagen simple con entropía = 4.4339.

(b) Imagen compleja con entropía = 7.7653.

Figura 1.7: Complejidad de imagen en relación a su entropía.

1.2. La importancia de la compresión

El avance exponencial de la tecnología ha obligado a que la información digital viaje rápidamente de un dispositivo a otro con la finalidad de ser almacenada para su fácil consulta, tratamiento y transmisión. Una gran parte de la información que viaja por la red y que se almacena en las computadoras son entornos gráficos orientados a múltiples aplicaciones, de ahí que el tamaño de los archivos de imagen es cada vez mayor, por lo cual la compresión de las imágenes digitales se ha hecho imprescindible [4].

El problema emerge cuando los dispositivos de almacenamiento son de bajos recursos o llegan a su limite de almacenamiento y/o velocidad de transmisión; por lo que una solución prometedora es hacer uso de métodos de compresión, que hacen que las imágenes en general sean almacenadas en menor espacio sin deteriorar significativamente la información que contienen.

Una imagen ocupa un determinado espacio en disco y si se puede conseguir que esa imagen ocupe menos, podremos tener más imágenes en el mismo espacio de almacenamiento. La idea es, si podemos tener más en el mismo espacio, ¿Por qué no hacerlo? solo necesitamos aplicar un compresor a la información para obtener el beneficio, pues lo que hay detrás de

todo sistema de compresión es eliminar información redundante.

1.3. Compresión de imágenes

El objetivo de la compresión de imágenes es reducir los datos redundantes e irrelevantes de una imagen con la menor pérdida de calidad posible para permitir su almacenamiento o transmisión de forma eficiente. No obstante, la compresión de imágenes va ligado a la complejidad que presentan las mismas, pues ésta deriva del contenido que presenta la imagen.

En general, hay dos técnicas compresión de imágenes; la compresión sin pérdida y técnicas de compresión con pérdida de información [5, 6] por lo que la aplicación de cada una de ellas se realiza de acuerdo al tipo de imagen que se esté analizando, debido a que en algunas imágenes no es permisible la perdida de información mientras que en otras es indistinto tener un grado de error, que bien para el ojo humano pasa a ser desapercibido.

- **Compresión sin pérdida de información:** se utiliza cuando es importante conservar la calidad de la imagen. En esta clasificación, las imágenes se codifican sin necesidad de conocer la naturaleza de los datos que la representan, son de propósito general y en donde la imagen reconstruida es exactamente la misma que la imagen original.

  Estas técnicas se destacan porque emplean métodos estadísticos, basados en la teoría de Shannon [7], que permite la compresión sin pérdida, las más habituales son: Run-length encoding (RLE), codificación de Huffman, codificación aritmética y Lempel-Ziv [8].

- **Compresión con pérdida de información:** se utiliza cuando no es tan importante conservar la calidad de la imagen, pues descarta información que considera innecesaria consiguiendo en general una reducción de datos mucho mayor. Se utiliza en imagen, audio y vídeo (incluyendo animaciones renderizadas), gracias a que los sentidos de la vista y del oído son imperfectos. Un error en un píxel o en un instante de tiempo es

indetectable si se siguen los procedimientos adecuados. O lo que es lo mismo, hay píxeles y sonidos que no vemos ni oímos, con lo cual no hace falta tenerlos guardados.

El principal inconveniente, como su nombre indica, es que la calidad se ve mermada, sin embargo se intenta que la pérdida sea imprecindible para el ojo humano. Algunas técnicas que destacan son: Codificación por transformación, Vector de Cuantización y Compresión Fractal [8].

En las Figuras 1.8 y 1.9 se pueden observar los dos diferentes técnicas de compresión en donde se argumenta que dependiendo el fin que tenga la imagen es el tipo de compresión que se va a requerir.

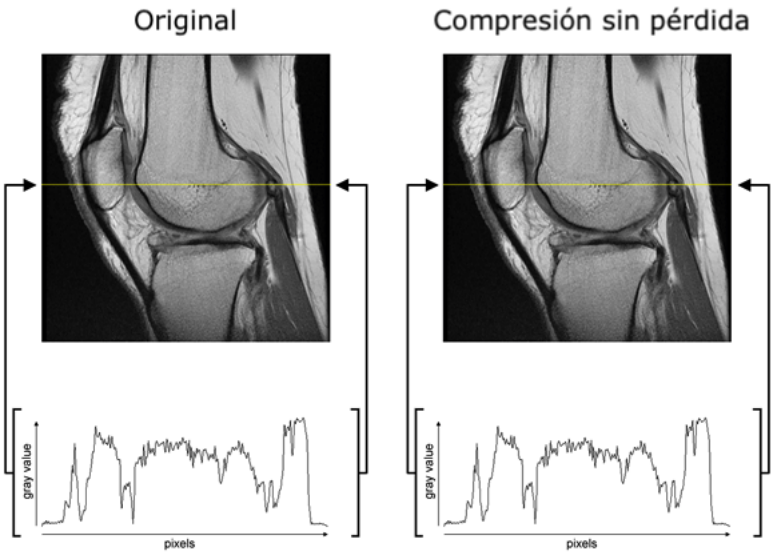


Figura 1.8: Diferencia entre imagen original e imagen con compresión sin pérdida. [1]

[1] Tomada de: https://lossyandlossless.weebly.com/lossless-compression.html

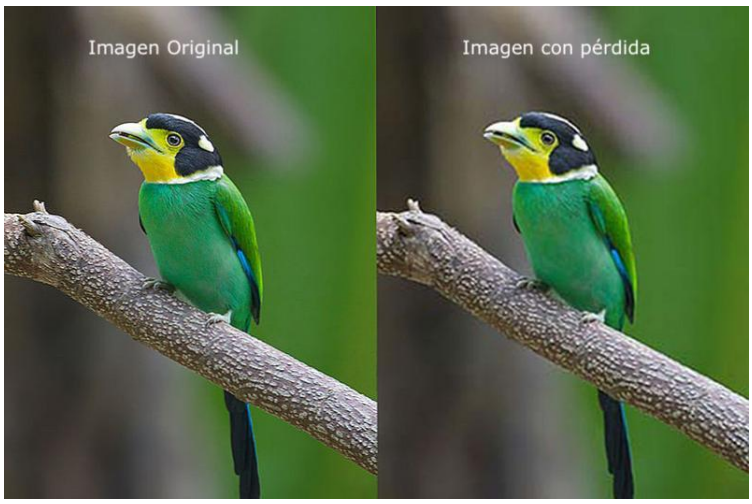


Figura 1.9: Diferencia entre imagen original e imagen con compresión con pérdida. [2]

1.3.1. Métodos compresión de imágenes digitales

Los diferentes formatos de imágenes digitales expuestos en la Tabla 1.1 soportan distintos métodos de compresión que dependiendo del objetivo a seguir es el método a utilizar.

Formato de imagen	Compresión con pérdida	Compresión sin pérdida
TIFF (Tagged Image File Format)	SÍ	SÍ
PNG (Portable Network Graphics)	No	Sí
JPEG (Joint Photographic Expert Group)	Sí	No
GIF (Graphic Interchange Format)	Sí	Sí
TGA (Truevision Graphics Adapter.)	Sí	Sí

Tabla 1.1: Formatos de imágenes digitales más comunes.

A continuación se mencionan los métodos de compresión más comunes.

◦ **Codificación Huffman:** Es un método descrito por David Huffman [9] que identifica los píxeles que aparecen con mayor frecuencia en una imagen y asigna representaciones cortas. Los píxeles que ocurren con menor frecuencia en una imagen se les asigna representaciones largas. Esta compresión es realizada a través de árboles y presenta la desventaja que junto al archivo comprimido debe estar también la tabla de códigos con las combinaciones de bits que más se repitan estadísticamente. El código de Huffman produce un ahorro cuando se

[1] Tomada de https://pediaa.com/difference-between-lossy-and-lossless-compression/

aplica directamente a una imagen, que funcionará mucho mejor si la imagen se compone de un número menor de intensidades distintas [10].

◦ **Codificador EZW:** presentado por J.Shapiro es un codificador especialmente creado para ser utilizado con la transformada wavelet y que está basado en la codificación progresiva para comprimir una imagen en una secuencia de bits con precisión creciente. Esto significa que cuantos más bits son añadidos a la cadena, la imagen reconstruida tendrá más detalles, no obstante es un método con pérdida [11].

◦ **Compresión JPEG:** método desarrollado por el comité "Joint Photographic Expert Group". Es el compresor de imagen con pérdidas más empleado en la actualidad. Dentro de los codificadores basados en la fuente, JPEG puede ser clasificado como codificador por transformación, ya que emplea la transformada discreta del coseno (DCT) para representar la imagen en el domino de frecuencias, y así poder compactar más la energía en las componentes de baja frecuencia [12]. Es un formato que pierde información al comprimir porque cuanto más se comprima mayor perdida de calidad tiene la imagen.

Aún cuando el formato de imágenes digitales más utilizado es JPEG, PNG también lo es, con la diferencia de que en este formato se da la compresión sin pérdida de información. No obstante, PNG no sustituirá a JPEG dado que éste consigue una mayor compresión en imágenes fotográficas.

◦ **Compresión fractal:** es un método de compresión con pérdida para imágenes digitales, basado en fractales (desarrollado en el capítulo 2) enunciado por Arnuad Jacquin. Este método es sumamente apropiado para texturas e imágenes naturales que no toman importancia en la perdida de información, tal como se observa en la Figura 1.10 que con muy poca información se puede recrear la imagen en su totalidad debido a las repeticiones que tiene la misma.

Figura 1.10: Helecho de Barnsley , imagen fractal recreada a partir de una hoja del helecho que lo contiene.

Su teoría se basa en el hecho de que partes de una imagen, a menudo, se parecen a otras partes de la misma imagen y por tanto puede ser eliminada alguna de ellas, por lo que los algoritmos fractales convierten estas partes en datos matemáticos llamados *"códigos fractales"*, los cuales se usan para recrear la imagen codificada. Es conveniente usar compresión fractal debido a que esta técnica no guarda píxeles sino códigos matemáticos que representan dicha información.

Para la presente investigación se usarán imágenes en formato TGA debido a que guarda datos crudos de imagen sin pasar inicialmente por ningún método de compresión. Posteriormente se utilizará el método de compresión fractal.

1.4. Planteamiento del problema

Una Imagen digital puede ser aproximada mediante un Sistema de Funciones Iteradas (SFI) que contenga un número mínimo de ecuaciones asociadas a un conjunto de coeficientes óptimos y un número mínimo de iteraciones, tales que se minimice la distancia entre ellas.(ver Sección 2.1.1.1).

De ahí que nos planteemos el siguiente escenario; si por el teorema del Collage [13] es posible encontrar un SFI que a partir del Atractor se pueda generar un nuevo conjunto que sea semejante a éste, nos resulta importante entonces saber cuál es el SFI y qué características presenta, para que la generación de la imagen a partir del sistema esté muy cercana a la imagen original.

Al momento de minimizar el coeficiente de distorsión (δ) que hay entre la imagen dada O,

y la imagen obtenida R, $0 \leq \delta = (O,R)$ por medio del SFI, deseamos que este se acerque lo mayor posible a 0 para asegurar que la imagen obtenida sea muy cercana a la imagen original, tal como se muestran en las Figuras 1.11 donde observamos el coeficiente δ que es mucho mayor que 0 con lo que se asegura que dicha imagen tiene buena aproximación aunque no muy cercana a la original.

(a) Imagen Original.

(b) Imagen obtenida del SFI con un alto $\delta =$ 28 unidades.

Figura 1.11: Comparación visual de imágenes.

Con la CF, la codificación a nivel computacional es lenta, debido a la búsqueda de similitudes propias, sin embargo, la decodificación es bastante rápida. Hacer uso de búsqueda exhaustiva para encontrar el SFI queda fuera de las posibilidades de los sistemas de cómputo actuales, pues para imágenes pequeñas sí hay una buena aproximación pero para imágenes de Tamaño de imagen mayor se vuelve complicado encontrar el Collage adecuado, es por ello la hipótesis que se plantea es *encontrar algún método metaheurístico que permita una buena reconstrucción sin perder calidad de la imagen resultante.*

1.5. Metodología propuesta

Se pretende, en este presente trabajo, reconstruir una imagen a escala de grises a partir de una imagen genérica (también a escala de grises), y que esta resultante tenga menor tamaño para su almacenamiento y transmisión. Es por ello que se adaptará e implementará un algoritmo genético (perteneciente a los algoritmos evolutivos) que es una técnica de búsqueda bioinspirada en la teoría de la evolución de Darwin [14] para acelerar la búsqueda de simi-

litudes propias en la imagen representadas por medio de SFI, aplicado a imágenes digitales con el alcance solo a escalas de grises.

En el Capítulo 2 se proporcionará un marco teórico sobre compresión fractal de imágenes y las metodologías a utilizar así como también una porción del estado del arte en cuestión sobre el mismo eje de investigación. En el Capítulo 3 se ahondará en la implementación del algoritmo de codificación fractal propuesto por el matemático Arnaud Jacquin y del algoritmo genético propuesto en esta presente tesis de investigación. En el Capítulo 4 se presentará la experimentación y resultados del algoritmo propuesto y por último en el Capítulo 5 se hará un balance final de este trabajo y se darán consideraciones finales.

De manera general, en la Figura 1.12 se observa el proceso de codificación y decodificación fractal aplicado a imágenes. Para obtener los códigos fractales que reconstruyen a la imagen es necesario que la imagen original Im_O se someta al algoritmo de codificación y éste nos devuelva estos códigos, por lo que en el Capítulo 3 se hablará del algoritmo propuesto.

Los códigos fractales que representan a la imagen entran en el proceso de decodificación implementado en el Capítulo 3, junto con una imagen genérica a escala de grises del mismo tamaño de la imagen a reconstruir. Este procedimiento da como resultado la imagen reconstruida, como se muestra en el Capítulo 4.

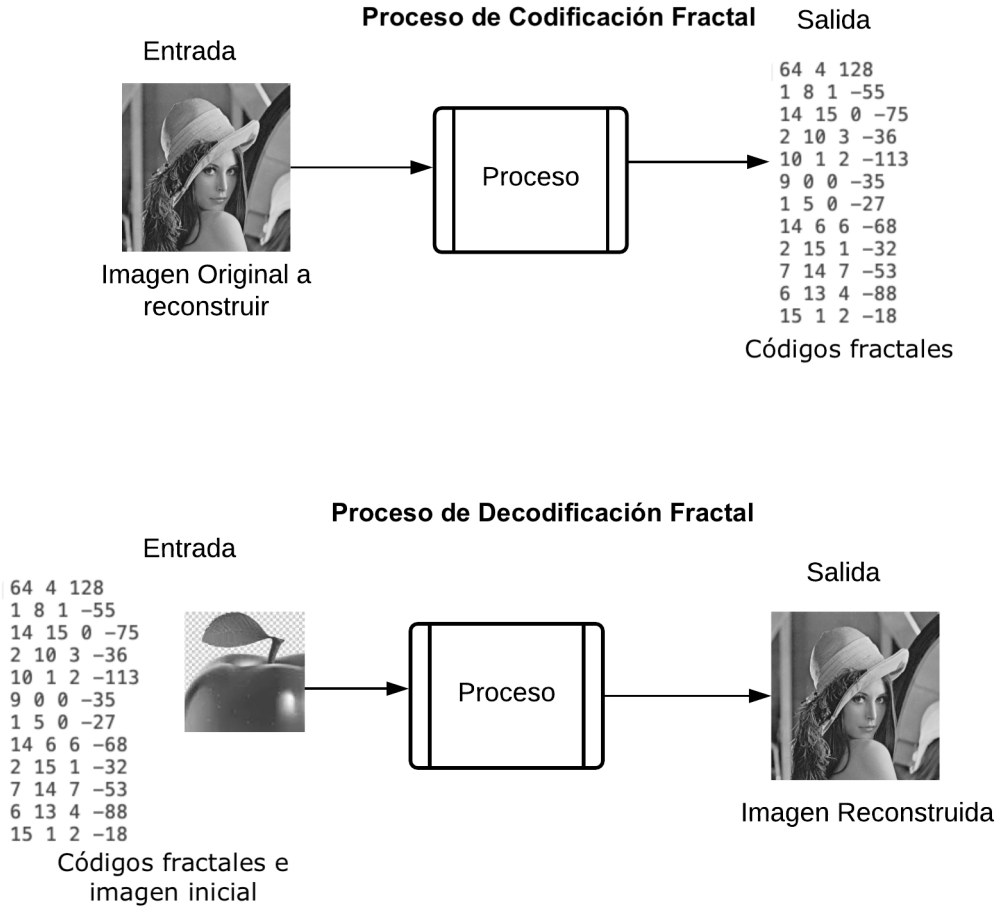


Figura 1.12: Representación visual de los procesos de codificación y decodificación fractal.

1

Capítulo 2

Marco Teórico

2.1. Teoría fractal

La geometría fractal, cuyos primeros desarrollos datan de finales del S. XX, ha recibido durante los últimos veinte años, una atención creciente. Ofrece un modelo alternativo que busca una regularidad entre un objeto y sus partes a diferentes escalas, puesto que no se pierde de la perspectiva del objeto global ni del aspecto del mismo en cada escala de observación [15–17]. De forma general, la geometría fractal busca y estudia los aspectos geométricos que son invariantes con el cambio de escala.

El término *Fractal* fue introducido por el matemático Mandelbrot para caracterizar fenómenos espaciales o temporales que son continuos pero no diferenciables debido a que en términos matemáticos, estos fenómenos presentan picos y por ende no se pueden sacar derivadas de las funciones que los representan aún cuando sean continuas en el sentido de que entre más observamos una parte del fractal a mayor escala, aparecer el objeto completo, tal como se muestra en la Figura 2.1.

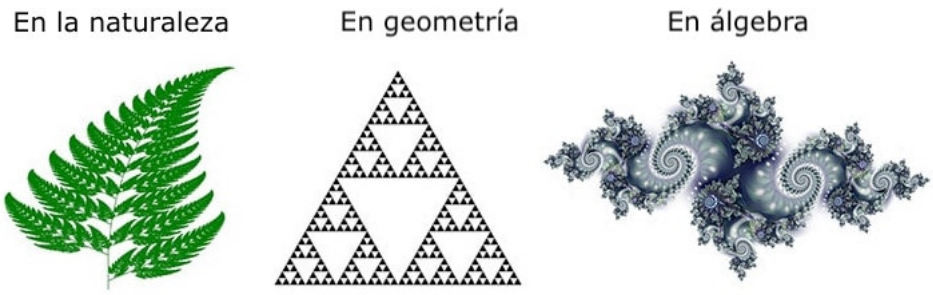


Figura 2.1: Conjuntos fractales en la naturaleza, geometría y álgebra.

La Dimensión de Hausdorff es una dimensión fraccionaria (no entera) para un objeto fractal. Está dada por la ecuación $D_H = \frac{In(\alpha)}{In(\frac{1}{r})}$ donde α es el número de copias de sí mismo a escala r. En ese sentido, el triángulo de Sierpinski está formado por 3 copias a escala $r = 1/2$ cuya dimensión no es un número entero sino una cantidad fraccionaria de 1.5850, tal como se observa en la Figura 2.2.

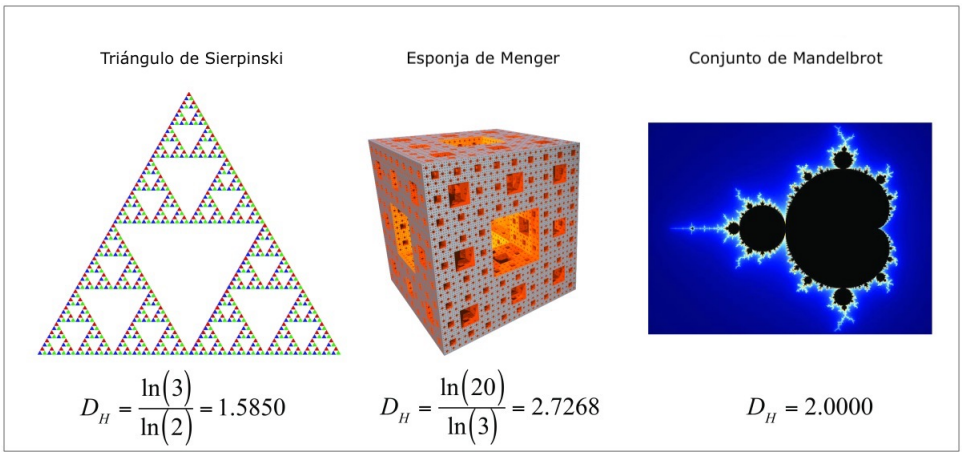


Figura 2.2: Dimensión de Hausdorff en conjuntos fractales.

Estos conjuntos que presentan autosimilitud, pueden modelarse mediante una construcción matemática llamada *Sistemas de Funciones Iteradas*, inscrita en la geometría fractal así como en la codificación fractal de imágenes introducida por el matemático Hutchinson [13, 18].

2.1.1. Transformada fractal

La aplicación de técnicas fractales para la compresión de imágenes digitales fue introducida por los matemáticos Michael Barnsley y Arnaud Jacquin en 1988 [19], y puede explotarse

modelándola como la redundancia presente en los objetos fractales, tal como se muestra en las Figuras 2.11 y 2.12, y en la Figura 3.1 para imágenes naturales. Por ello, la base matemática de las técnicas a utilizar es una teoría general de transformaciones contractivas iterativas en espacios métricos, basada en el trabajo de Barnsley y diversos autores [13, 20]. A esto es a lo que se le conoce como *transformada fractal* ; aplicar una serie de transformaciones a un conjunto compacto que nos genere un atractor del sistema de dichas transformaciones.

2.1.1.1. Sistema de funciones iteradas

Para dar una definición adecuada de qué es un sistema de funciones iteradas se dan a continuación alguna definiciones previas que permiten la introducción a esta teoría.

Definición 2.1. Un *espacio métrico* es un par (X, d), donde X es un conjunto arbitrario no vacío y $d : X \times X \to \mathbb{R}$ una aplicación, llamada *distancia o métrica.*

Definición 2.2. Sean (X, d) un espacio métrico y una sucesión $(x_n)_{n=1}^{\infty} \subset X$ es una *sucesión de Cauchy* si $\forall \epsilon > 0$ existe $n_0 \in \mathbb{N}$ tal que $d(x_n, x_m) < \epsilon$ si $n, m \geq n_0$.

Definición 2.3. Un *espacio métrico es completo* si toda sucesión de Cauchy es convergente.

Definición 2.4. Una *transformación* $t : X \to X$ es *contractiva* si existe una constante $k < 1$ tal que para cualesquiera dos puntos $x_1, x_2 \in X$ se cumple:
$d(t(x_1), t(x_2)) < k \cdot d(x_1, x_2)$. Es decir, una transformación contractiva es aquella que contrae las distancias con una razón de contracción estrictamente menor que la unidad.

Definición 2.5. Sea (X, d) un espacio métrico completo y sean $w_i : X \to X$ (con $i = 1, 2, ..., l$) un conjunto finito de ecuaciones que describen transformaciones contractivas, entonces:

$$\mathscr{S} = \{(X, w_i)\}; \qquad i = 1, 2, ..., l \tag{2.1}$$

es lo que se llama *sistema de funciones iteradas* [21].

La representación matemática de las ecuaciones w_i se define en Ec. (2.2), donde los coeficientes (a,b,c,d) definen la combinación de transformaciones de los puntos en el plano (posición y tamaño).

$$w\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} \tag{2.2}$$

Cuando a estas ecuaciones w_i se les aplica un traslado entonces se agregan los coeficientes $(u,v,)$ y si se aplican a imágenes a escala de grises, se agrega la variable z con relación a los coeficientes (f,g) que definen la combinación de brillo y contraste de las imágenes, tal como se muestra en la Ec. (2.3).

$$w\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} a & b & 0 \\ c & d & 0 \\ 0 & 0 & f \end{pmatrix}\begin{pmatrix} x \\ y \\ z \end{pmatrix} + \begin{pmatrix} u \\ v \\ g \end{pmatrix} \tag{2.3}$$

Las transformaciones isométricas son cambios de posición y orientación de una imagen determinada que no alteran la forma ni el tamaño de ésta. Por lo tanto, los coeficientes (a,b,c,d) se escogen en conjunto para efectuar una de ocho transformaciones isométricas que se muestran a continuación:

t_i	Isometría
1	identidad
2	reflejo en eje Y
3	reflejo en eje X
4	rotación en 180°
5	reflejo sobre $y = x$
6	rotación en 270°
7	rotación en 90°
8	reflejo sobre $y = -x$

Tabla 2.1: Transformaciones isométricas.

De la tabla 2.1 tenemos las siguientes representaciones con base en la matriz de los coefi-

cientes $B = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$

1. Identidad $B = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$

Figura 2.3: Identidad.

2. Reflejo en eje Y $B = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$

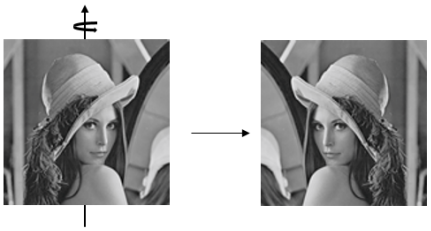


Figura 2.4: Reflejo en eje Y.

3. Reflejo en eje X $B = \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix}$

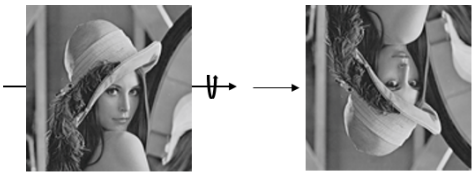


Figura 2.5: Reflejo en eje X.

4,6,7: Rotación con $\theta \in \{90, 180, 270\}$

$B = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix}$

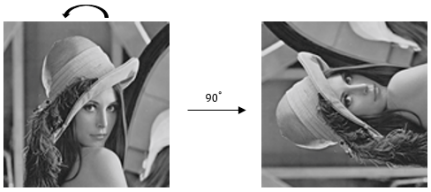


Figura 2.6: Giro 90°.

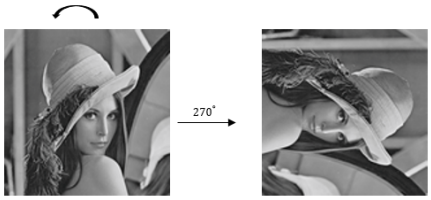


Figura 2.7: Giro 270°.

5: Reflejo sobre la diagonal $y = x$

$$B = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

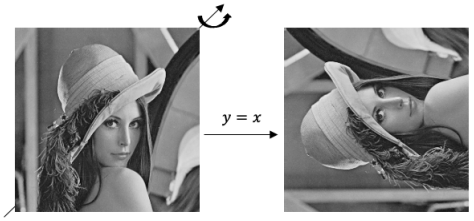


Figura 2.8: Reflejo sobre la diagonal $y = x$.

8: Reflejo sobre diagonal $y = -x$

$$B = \begin{pmatrix} 0 & -1 \\ -1 & 0 \end{pmatrix}$$

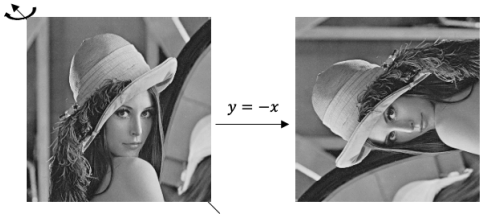


Figura 2.9: Reflejo sobre la diagonal $y = -x$.

Si se aplica la Ec. (2.3) con respecto a la Tabla 2.1 a un conjunto de puntos iniciales (x_0, y_0) (ver Figura 2.10) un cierto número de veces, las posiciones de éstos cambiaran conforme avance la iteración, ya que la nueva ecuación definida por x_{i+1}, y_{i+1} depende de los valores de los puntos x_i, y_i anteriores, cuidando que no solo se aplique la transformación de identidad, de esta manera el nuevo conjunto resultante presenta características fractales. En el ejemplo 2.1.1.1 se muestra gráficamente cómo se va formando el nuevo conjunto a partir de la aplicación de ciertas transformaciones.

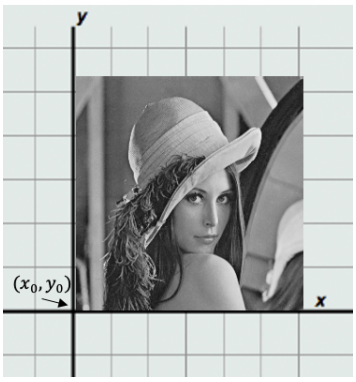


Figura 2.10: Representación de puntos iniciales (x_0, y_0) de una imagen en el plano cartesiano.

Ejemplo 2.1. Considerando la Figura 2.11, partiendo de un segmento en $\mathbb{R}^2$, se construye a partir de $\bigcup_{i=1}^{4} w_i(X)$ donde cada una de ellas representa transformaciones de traslado y rotación. De manera iterativa se aplican estas mismas transformaciones sobre el conjunto resultante de la unión, dando como resultado un nuevo segmento. Si este procedimiento se aplica de forma iterativa sobre el conjunto resultante, da origen a la llamada curva de Koch que se observa en la Figura 2.12, clasificada como conjunto fractal.

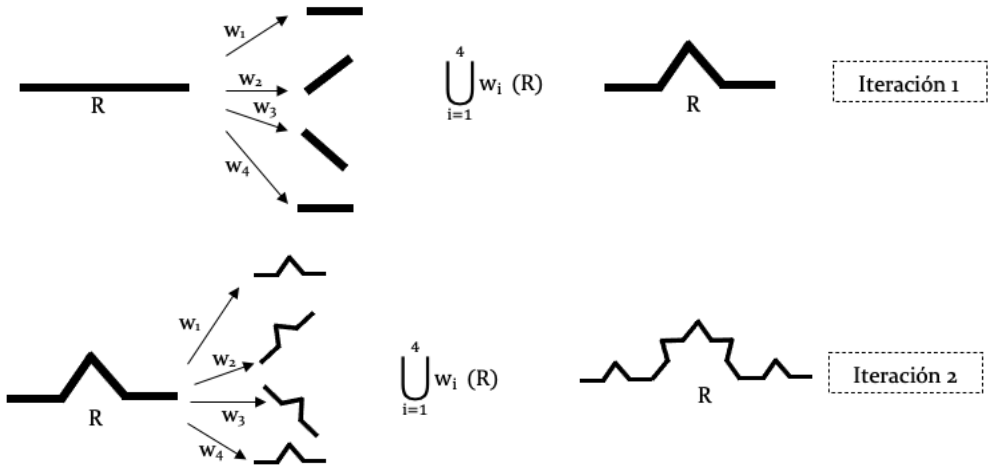


Figura 2.11: Proceso iterativo para la creación de la curva de Koch a partir de un segmento $\mathbb{R}$.

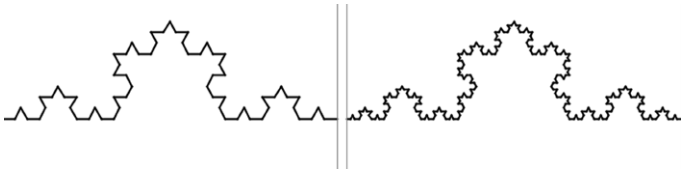


Figura 2.12: Curva de Koch final, tomada de [22].

Si un SFI, $\mathscr{S} = (F, w_i); i = 1,2,...,l$, es contractivo, entonces existe un *único* conjunto A tal que

$$W(A) = \bigcup_{i=1}^{l} w_i(A) = A \tag{2.4}$$

llamado el punto fijo del SFI, o *"atractor"* donde W es el conjunto de las transformaciones afines contractivas w.

2.1.1.2. Sistema de funciones iteradas por partes

La teoría expuesta en la Sección 2.1.1.1 queda exigua cuando se aplica a imágenes que no tienen comportamiento a simple vista fractal. Por ejemplo, es inusual ver, en una imagen

natural, pequeñas imágenes de la misma al momento de variar la distancia focal, como se ejemplifica en la Figura 2.13.

Es por ello que en lugar de verla como una serie de copias de ella misma, ésta estará formada por copias de pedazos de la misma imagen bajo transformaciones apropiadas, como se observa en la Figura 2.14 en donde la región R1, R2 y R3 son la misma solo que con diferentes transformaciones. Es así, como haremos uso de los SFI aplicado a imágenes naturales para lograr una compresión. Este método, en general se conoce como *sistema de funciones iteradas por partes, SFIP* [23].

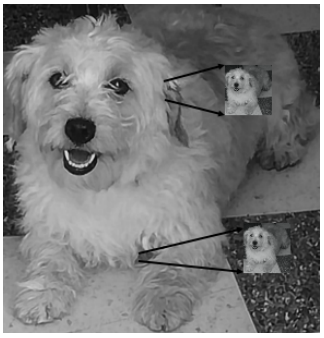


Figura 2.13: La imagen total está inmersa en pequeñas regiones a diferentes escalas.

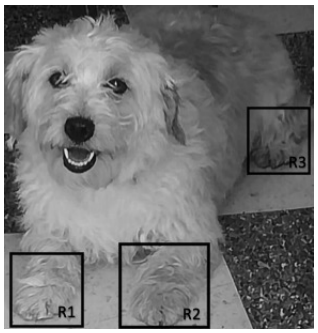


Figura 2.14: Hay varias partes de la imagen que se parecen entre sí, en particular las regiones que se encuentran enmarcadas en R1, R2 y R3.

La idea general se basa en tomar la imagen y modelar regiones que presenten autosimilitud, mediante un sistema de funciones iteradas. Un ejemplo de regiones con autosimilitud se muestra en la Figura 2.14, pues una región se dice que presenta autosimilitud si la misma forma se repite una y otra vez a escalas cada vez más pequeñas. De esta manera la teoría fractal y la compresión de imágenes se encuentran en estrecha relación, pues se está resolviendo

un SFI para cada región que se desee de la imagen.

2.1.2. Teorema del Collage

Con base en la definición enunciada en la Ec. 2.4, decimos que un conjunto $A = \bigcup_{i=1}^{l} w_i(A)$ se obtiene a partir del análisis matemático que se ejerce sobre una figura regular o un conjunto compacto no vacío representado por las transformaciones que se aplican por medio de las ecuaciones w_i.

Para satisfacer la igualdad entre el atractor y el conjunto obtenido mediante el SFI se debe cumplir semejanza entre ambos, para ello existe un teorema que estriba en esta idea, ya que hace énfasis en la relación entre ambos conjuntos a partir de la afinidad que hay entre ellos.

Teorema 2.1 (Teorema del Collage, Barnsley (1985) [13])**.** Sea (X, d) un espacio métrico completo, $L \in \mathscr{H}(X)$ una imagen digital en representación matricial situada en el conjunto de los fractales $\mathscr{H}$ y $\epsilon \geq 0$ dados.

Elegir un SFI $\{X; w_1, w_2, ..., w_l\}$ con factor de contractividad $0 \leq c < 1$, tal que:

$$d_H\left(L, \bigcup_{i=1}^{l} w_i(L)\right) \leq \epsilon,$$

es decir, $W(L)$ está suficientemente próximo a L, donde d_H es la distancia bajo la métrica de Hausdorff, entonces

$$d_H(L, A) \leq \frac{\epsilon}{(1-c)},$$

donde A es el atractor del SFI. Equivalentemente,

$$d_H(L, A) \leq \frac{1}{(1-c)} d_H\left(L, \bigcup_{i=1}^{l} w_i(L)\right) \qquad \forall \ L \in \mathscr{H}$$

El Teorema 2.1 asegura que la aproximación del atractor A al conjunto L será mejor cuanto más pequeño sea el valor del factor de contractividad c, es decir, llegará un momento en que la nueva imagen será indistinguible a L.

Entiéndase la Distancia de Hausdorff [24] como una métrica que establece qué tan lejos se encuentran dos subconjuntos compactos, uno del otro, dada por:

$$d_H(X,Y) = \text{máx}\{\sup_{x\in X} \inf_{y\in Y} d(x,y),\ \sup_{y\in Y} \inf_{x\in X} d(x,y)\}$$

donde d es la distancia euclidiana.

Una cota de la distancia de Hausdorff [25] entre el conjunto L y su aproximación $W^k(L)$, está dada por:

$$d_H(L, W^k(L)) = \frac{1}{1-c} d_H(W^k(L), W^{k+1}(L))$$

donde k es la iteración actual del SFI, $k+1$ es la iteración siguiente y L es el conjunto al que se le aplicará las transformaciones W, por lo tanto $W^{k+1}(L)$ son las transformaciones W que se le aplicarán al conjunto resultante L de la iteración anterior.

Por medio de este teorema obtenemos un conjunto de ecuaciones $\{w_i | i = 1,2,\cdots,l\}$ que aproximan a un conjunto L. Si el SFI asociado contiene pocas de estas ecuaciones, es conveniente almacenar la imagen obtenida a partir de éste, en lugar del conjunto L, que finalmente al ojo humano las pequeñas diferencias entre ellas pasan a ser desapercibidas, ganando así una reducción significativa del espacio que ocupa L.

De aquí nace la inquietud de aproximar imágenes mediante un SFI. Dada una imagen inicial I, encontrar el $\bigcup_{i=1}^{l} w_i(I)$ para obtener $W(I)$ es a lo que se le conoce como *"el problema inverso de SFI"*, enunciado por el matemático Barnsley en [13]. Formalmente se describe como: *dado un conjunto C, encontrar un SFI asociado para el cual C sea el atractor.*

2.1.3. ¿Por qué usar compresión fractal?

La compresión fractal es un método de compresión con pérdida, no obstante hacer uso de compresión por medio de Geometría fractal nos permite una alta compresión, así como también independencia de resolución debido a que partes de una imagen, por lo general, presentan similitudes con otras partes de la misma imagen. De esta manera, los algoritmos fractales convierten estas partes en datos matemáticos a través de códigos fractales que sirven para almacenar y devolver las imágenes.

Por otra parte, una de las diferencias fundamentales que tiene este método con otros mé-

todos de compresión es que el código de compresión en sí no guarda píxeles, por lo que es libre de escalas y de esta forma se puede descomprimir a cualquier escala sin tener problemas de resolución. De ahí que es conveniente guardar la imagen como un archivo que contiene códigos fractales a almacenarla como un conjunto de píxeles.

2.2. Teoría y especificaciones del Algoritmo Genético (AG)

Los algoritmos evolutivos, tienen la capacidad de explorar eficientemente el espacio de búsqueda, por esta razón, representan una estrategia prometedora a la codificación fractal. Entre ellos, AG permiten adaptar mecanismos de cruza y mutación para controlar el factor de convergencia de la búsqueda [26].

Los AG son bio-inspirados con la evolución natural [14]. Trabajan con una población de individuos, los cuales cada uno representa una solución factible y que es capaz de evolucionar hacia una solución óptima por medio de un proceso de supervivencia de los individuos mejor adaptados.

El funcionamiento de los AG se basa en la modificación iterativa de una población de individuos que son soluciones factibles, aunque no siempre óptimas, mediante el uso de operadores genéticos [27]. En la Figura 2.15 podemos ver los pasos generales a seguir por el AG así como el pseudocódigo general en el algoritmo 1. Posteriormente tendremos las definiciones que explican partes del mismo.

Resultado: función objetivo

Crear población inicial;

mientras *generaciones* ≤ *max_generaciones* **hacer**

  Evaluar a los individuos de la población;

  Asignar el valor de aptitud a cada individuo;

  si *criterio de parada* **entonces**

    población solución;

    termino (break);

  fin

  en otro caso

    aplicar operador de cruza;

    aplicar operador de mutación;

    generar nueva población;

  fin

fin

Algoritmo 1: Algoritmo genético general.

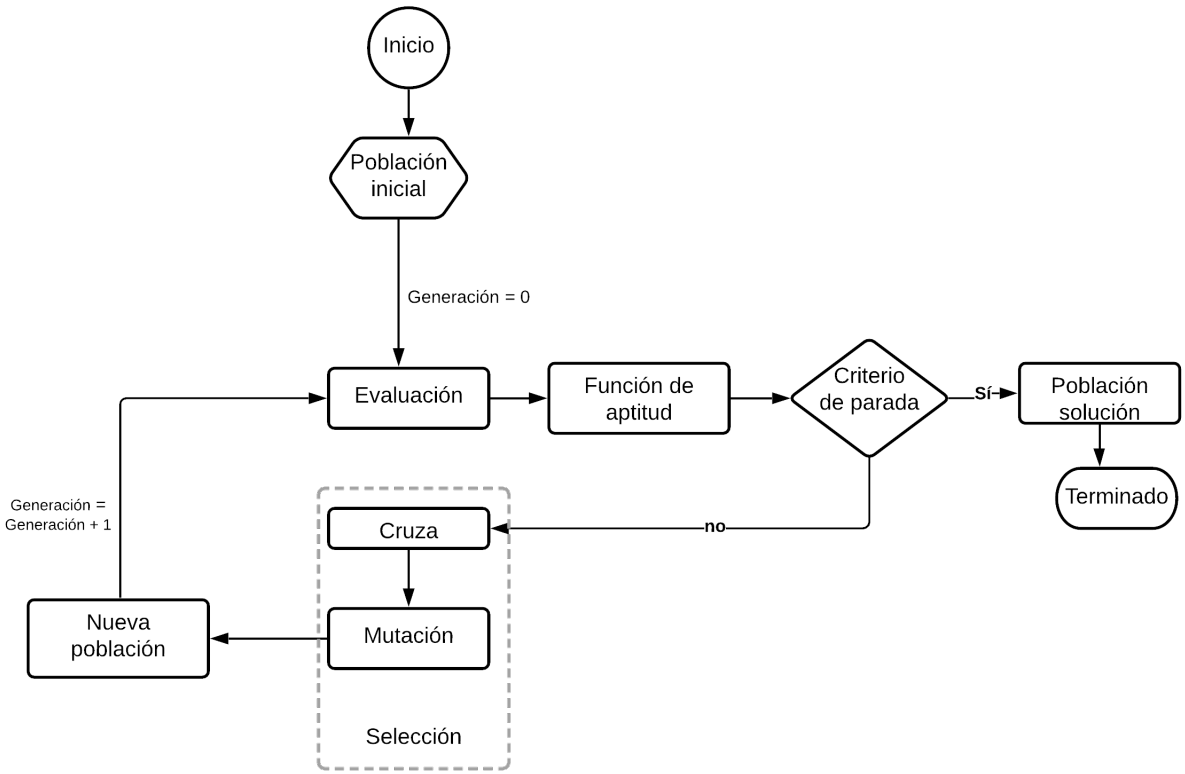


Figura 2.15: Diagrama de flujo de un AG general.

◦ **Definiciones**

- **Individuo:** conjunto de parámetros (genes) que representan una solución al problema de optimización que se intenta resolver, en particular al problema inverso de SFI.
- **Población inicial:** conjunto de individuos que pueden ser generados al azar o con información especifica que permitan ser una solución inicial factible.
- **Operadores genéticos:** son mecanismos para reproducir la información de uno o más individuos de la población para que produzcan nuevos individuos mejor adaptados.
- **Función de aptitud:** es la que permite valorar la aptitud de los individuos y debe tomar siempre valores positivos. Para el caso de un problema de minimización, la relación del valor de aptitud con la función objetivo f es inversamente proporcional; el individuo tiene un valor de aptitud mayor cuanto menor es el valor de la $f(individuo)$, es decir, cuanto mayor es el valor de la f de ese individuo, menor será valor de aptitud. Con la relación $\frac{1}{1+f(individuo)}$, el algoritmo genético selecciona a los individuos de mayor valor de aptitud.

  - **Selección:** sirve para escoger a los individuos de la población mejor adaptados, con el objetivo de participar en el proceso de reproducción. Un método muy utilizado en los AG (y que también utilizamos en el presente trabajo) es la selección por ruleta [28], que consiste en asignar a cada individuo una porción de una ruleta circular, proporcional a su valor de aptitud, tal como se muestra en el siguiente ejemplo descrito por la Figura 2.16 y la Tabla 2.2, donde Ω_i es el valor de aptitud del individuo i y ps_i es la probabilidad de selección del individuo i.

    Para cada elemento x_i de la población, se calcula su probabilidad de ser seleccionado mediante la Ec. (2.5), donde N es el número total de individuos en la población.

$$ps_i = \frac{\Omega_i(x_i)}{\sum_{i=1}^{N} \Omega_i(x_i)} \tag{2.5}$$

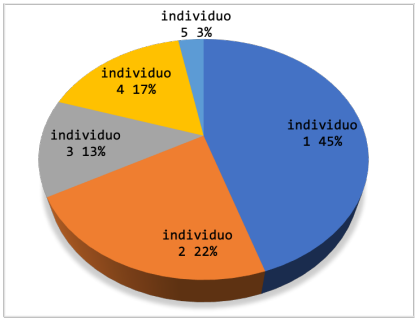


Figura 2.16: Proporción del valor de aptitud de una población con 5 individuos.

i	Ω_i	ps_i
1	3	0.45
2	5	0.22
3	7.5	0.13
4	6	0.17
5	10	0.03

Tabla 2.2: Valor de aptitud y probabilidad de selección de una población con 5 individuos.

Para el ejemplo que se muestra en la Tabla 2.2, en un problema de minimización, el peor individuo es quien tiene en Ω_i un valor de 10 y con ello es acreedor a tener una probabilidad baja de ser seleccionado.

- **Cruza:** operador genético que tiene por objetivo generar descendencia a partir de dos individuos de la población actual. Dos individuos llamados padres son seleccionados mediante el mecanismo de selección. Este operador intercambia material genético de los padres produciendo así dos nuevos individuos llamados hijos.

  Hay varias formas de realizar el procedimiento de cruza, no obstante para este trabajo se pretende emplear dos: *cruza en un punto*; ver Figura 2.17 para cromosomas binarios, donde el punto de cruza es entre el cuarto y quinto dígito de izquierda a derecha de ambos padres; y *cruza parcialmente mapeada* (PMX por sus siglas en inglés) [28] .

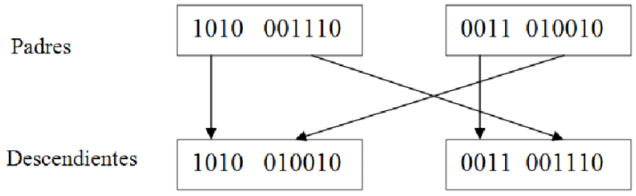


Figura 2.17: Operador de cruza en un punto.

Habitualmente el operador de cruza no se aplica a todos los pares de individuos que han sido seleccionados para reproducirse, sino que se aplica de manera aleatoria de forma que solo un porcentaje de los individuos seleccionados produzcan descendencia; a dicho porcentaje se le conoce como *probabilidad de cruza*. En el caso en que el operador de cruza no se aplique, la descendencia se obtiene simplemente duplicando los padres.

- **Mutación:** es el operador que previene que la población presente una convergencia acelerada y que existan múltiples copias de un mismo individuo. Consiste en aplicar, con cierta probabilidad, una alteración aleatoria de al menos un gen dado; a dicha probabilidad se le conoce como *probabilidad de mutación*. En la Figura 2.18 se muestra el proceso de mutación en cromosomas binarios.

Figura 2.18: Operador de mutación.

- **Condiciones de paro:** está relacionado con la progresión hacia la uniformidad; el procedimiento ha convergido cuando al menos el 95 % de los individuos de la población comparten el mismo valor por lo que el criterio de paro se ha cumplido ya que la f se ha estabilizado.

2.3. Modelación Matemática

La CFI se basa en la segmentación de la imagen a reconstruir en bloques dominio bd y bloques rango br. Los bloques dominio son secciones cuadradas de la imagen de tamaño $2p \times 2p$ px, mientras que los bloques rango son de tamaño $p \times p$ px. La segmentación de

bloques dominios es de forma continua y la segmentación para bloques rango es de forma traslapada 1 píxel sobre el eje x y 1 píxel sobre el eje y.

En la Figura 2.19 se ejemplifica la segmentación de la imagen en BD, mientras que en la Figura2.20 se observan los BR no traslapados, mostrando así la relación de tamaños entre ambos conjuntos. Por último en la Figura 2.21 se muestra la forma de segmentar la imagen por bloques traslapados.

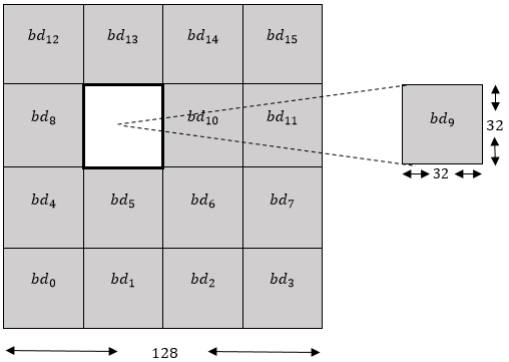


Figura 2.19: Ejemplo de segmentación de Im_O en bd en una imagen de 128 × 128px.

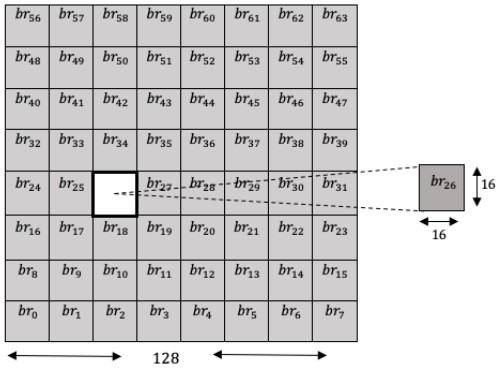


Figura 2.20: Ejemplo de segmentación de Im_O en br sin traslapar, en una imagen de 128 × 128px.

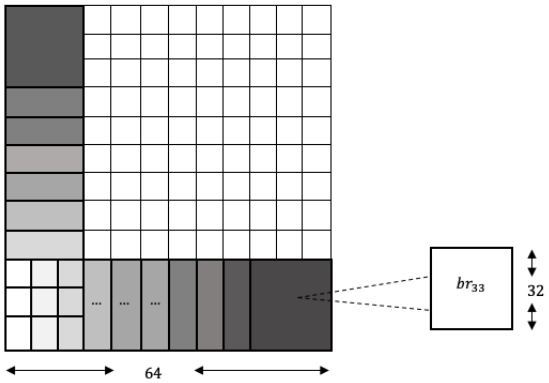


Figura 2.21: Ejemplo de segmentación de Im_O en br traslapados en ejes x y y.

La Ec. 2.6 representa la función objetivo, que es minimizar la distancia que hay entre bd_i y br_j^*; donde bd_i representa el $i-$ésimo bloque dominio para $i = \{1, ..., p\}$, siendo p el número máximo de bloques dominio, ubicado en la posición (x_i, y_i) y br_j^* para $j = \{1, ..., q\}$, siendo q el número máximo de bloques rango transformados, ubicado en la posición (x_j, y_j).

Esta ecuación está restringida por la desigualdades de 2.7 a 2.11. La desigualdad 2.7 representa la cantidad de bloques dominio a tener, siendo 4 bloques como mínimo y $(\frac{n}{m})^2$ como máximo, debido a que la cantidad depende del tamaño de la imagen a reconstruir (n) y del tamaño del bloque (m). La desigualdad 2.8 representa la cantidad de bloques rango a tener, siendo 4 bloques como mínimo y $(\frac{n}{m} - m + 1)^2$ como máximo, debido a que son bloques traslapados 1 px en 2 dimensiones. La desigualdad 2.9 ejemplifica el tamaño de la imagen a reconstruir. Haciendo hincapié que son imágenes cuadradas, la más pequeña es de tamaño 64×64 px mientras que la más grande es de tamaño 1024×1024 px. Por último tenemos las restricciones 2.10 y 2.11 que hacen alusión a las coordenadas en el plano de los bloques dominio y rango.

$$\min \quad \delta = \sqrt{\frac{1}{n^2} \sum_{i=1}^{p} \sum_{j=1}^{q} (bd_i(x_i, y_i) - br_j^*(x_j, y_j))^2} \tag{2.6}$$

sujeto a:

$$4 \leq bd_i \leq \left(\frac{n}{m}\right)^2 \tag{2.7}$$

$$4 \leq br_j \leq \left(\frac{n}{2} - m + 1\right)^2 \tag{2.8}$$

$$64 \leq n \leq 1024 \tag{2.9}$$

$$x_i, y_i \geq 1 \quad \text{para} \quad i = \{1, ..., k\} \tag{2.10}$$

$$x_j, y_j \geq 1 \quad \text{para} \quad j = \{1, ..., r\} \tag{2.11}$$

2.4. Análisis sobre la función objetivo general

La calidad de imagen que se obtiene en el método de codificación fractal depende del tamaño de los bloques o segmentos de la imagen en que se va a particionar y en el contenido de la imagen inicial o base con el que iniciará el proceso de decodificación. Para ello, se necesita medir este factor por medio de ciertas métricas que nos permitan evaluar la calidad de imagen obtenida R en función de la partición utilizada, en comparación con la imagen original O. Las métricas más habituales en el procesamiento de imágenes se enuncian a continuación donde $O(x, y)$ representa el píxel con coordenadas (x, y) de la imagen original y $R(x, y)$ el píxel con coordenadas (x, y) de la imagen reconstruida:

- *Distancia de Hausdorff:*

$$d_H(O(x,y), R(x,y)) = \text{máx}\{\sup_{x\in X} \inf_{y\in Y} d(O(x,y), R(x,y)), \sup_{y\in Y} \inf_{x\in X} d(O(x,y), R(x,y))\}, \tag{2.12}$$

  donde d es la distancia euclidiana.

  d_H aplicado a imágenes, proporciona una medida del grado de similitud entre ellas, en este caso la imagen original O y la imagen reconstruida R.

  Esta distancia presenta consideraciones particulares que alteran de forma significativa la función objetivo como consecuencia de pequeñas variaciones en los coeficientes del SFI. Por ejemplo si en la transformación de rotación se aplica rotar solo un grado a la imagen, la d_H dará un valor que bien puede ser mayor y que perjudique el valor de la función objetivo tratándose de que el problema es minimizar el valor de esta función.

  Otro ejemplo realizado fue calcular esta distancia entre una imagen con píxeles negros en su totalidad y una imagen solo con algunos píxles negros y el restante con píxeles blancos. Evidentemente sabíamos que la distancia debía ser muy grande, pues ambas

imágenes no se parecían en nada no obstante, la d_h dio un valor de 0 con lo cual concluimos que la imagen con pocos píxeles en negro y el resto en blanco, se contenía en la imagen en su totalidad con píxeles negros. De ahí que descartáramos esta métrica para la evaluación de nuestra función objetivo.

Esta desventaja hace presente la sensibilidad al hacer pequeños cambios en las transformaciones W, ya que los píxeles pintados por medio del SFI pueden desbordar el lienzo definido para pintar la imagen, pintar un píxeles extra o bien no pintarlo, siendo que visualmente pasan desapercibido para el ojo humano no obstante, cuando se calcula la d_H el valor puede, o no, ser el esperado en comparación con la imagen resultante.

- *Error Medio Absoluto* (MAE, por sus siglas en inglés Mean Absolute Error):

$$MAE = \frac{1}{n^2} \sum_{x=1}^{n} \sum_{y=1}^{n} |O(x,y) - R(x,y)| \tag{2.13}$$

Otra forma de evaluar la distancia entre la imagen original y la imagen reconstruida es con la métrica MAE, que calcula el promedio de las diferencias absolutas de los píxeles entre la imagen original O y la imagen reconstruida R.

- *Error Cuadrático Medio* (RMSE, por sus siglas en inglés Root Mean Square Error):

$$RMSE = \sqrt{\frac{1}{n^2} \sum_{x=1}^{n} \sum_{y=1}^{n} (O(x,y) - R(x,y))^2} \tag{2.14}$$

Es la métrica más utilizada en la compresión de imágenes que calcula la raíz de la diferencia cuadrada entre la imagen original O y la reconstruida R para luego promediar esos valores. Esta distancia presenta la ventaja de que los valores de error grandes se magnifican, mientras que los pequeños se ignoran, lo que genera estabilidad y por tanto nos muestra una métrica mejor adaptable.

- *Error Pico Señal a Ruido* (PSNR, por sus siglas en inglés Peak Signal-to-Noise Ratio):

$$PSNR = 10\log_{10} \frac{255}{\frac{1}{n^2}\sum_{x=1}^{n}\sum_{y=1}^{n}(O(x,y) - R(x,y))^2} \tag{2.15}$$

Calcula la relación pico de señal a ruido, en decibelios, entre dos imágenes. El uso más habitual es como una medida de la calidad de la reconstrucción en la compresión de imágenes. Cuanto mayor sea el valor dado por PSNR, mejor será la calidad de la imagen reconstruida. Los valores típicos que adopta este parámetro están entre 30 y 50 dB.

2.4.1. Análisis de las imágenes mediante histogramas

Un histograma es un gráfico que nos muestra en general la forma en cómo están distribuidas las intensidades de los valores de los píxeles de la imagen y es útil cuando se desea hacer un análisis visual de la composición de la imagen. Numéricamente los histogramas se comparan para saber qué tanto se parece uno de otro, por lo que las métricas más habituales de comparación son las siguientes:

- Correlación:

$$d_{correl}(H_1, H_2) = \frac{\sum_I (H_1(I) - \bar{H}_1)(H_2(I) - \bar{H}_2)}{\sqrt{\sum_I (H_1(I) - \bar{H}_1)^2 \sum_I (H_2(I) - \bar{H}_2)^2}} \tag{2.16}$$

Donde

$$\bar{H}_k = \sum_J H_k(J) \tag{2.17}$$

y $H_k(J)$ es el histograma k con $k = 1$ ó $k = 2$ de la imagen J y N es el número total de intervalos de clase, es decir el eje x dividido en rangos de números. Para cada intervalo, se dibuja una barra en la que el ancho de la barra representa el rango del intervalo y la altura de la barra representa el número de datos incluidos en ese rango [29].

Siendo $H_1(I)$ y $H_2(I)$ histogramas de la imagen I, describen el grado de dependencia lineal que existe entre el histograma de una imagen y otra. El coeficiente de correla-

ción oscila entre −1 y +1 encontrándose en medio el valor 0 que indica que no existe asociación lineal entre los dos histogramas de las imágenes en estudio.

- Intersección:

$$d_{inter}(H_1, H_2) = \sum_I (H_1(I), H_2(I)) \tag{2.18}$$

Describe el número de píxeles que coinciden entre ambos histogramas así, si el número resultante es el número total de los píxeles en la imagen original entonces estaremos teniendo que ambas imágenes son la misma [30].

- Distancia Bhattacharyya:

$$d_{bhatta}(H_1, H_2) = \sqrt{1 - \frac{1}{\sqrt{\bar{H}_1 \bar{H}_2 N^2}} \sum_I \sqrt{H_1(I) \cdot H_2(I)}} \tag{2.19}$$

Esta métrica nos permite observar qué tan cercanos están un histograma de otro mostrando como máximo el valor 1 que no hay coincidencia ó 0 donde hay una alta coincidencia entre las imágenes [29].

2.5. Antecedentes de la compresión fractal

La fuerza y popularidad que ha tomado la compresión fractal de imágenes, ha hecho que diversos autores realicen contribuciones para obtener una solución al problema inverso del SFI. Por citar algunos autores, S. Abenda *et al.* en [31] analiza una subclase de SFI, los sistemas lineales de Cantor (LCSs), que se dan simplemente por conjuntos finitos de mapas lineales con imágenes que no se superponen, centrándose en la hipótesis de que si el problema está restringido a un LCSs en una línea real, entonces existe una solución y puede hacerse única, es decir que un conjunto arbitrario se pueda aproximar por medio de LCSs con escalas iguales. En comparación, en el presente trabajo de tesis sí abordaremos la superposición de regiones de la imagen (bloques traslapados) para una mejora captación de información de la misma.

Por otro lado, Niclas Wadströmer en [32] junto con varios colegas, analizan las dos fases del algoritmo de Barnsley, para acelerar el tiempo de ejecución y obtener mejores aproximaciones entre imágenes, con lo cual logran reducir tiempos de codificación no obstante, las

imágenes a trabajar por estos autores son solo binarias, es decir a blanco y negro e imágenes fractales.

Desde otra perspectiva diversos autores [33, 34] que utilizan transformación wavelet y momentos de la imagen, así como probabilidades dependientes de lugar [35] reducen tiempo de cómputo en la codificación y aumentan la semejanza entre imágenes.

Al-Bundi *et. al.*, en [36] proponen un algoritmo genético mejorado a partir de un SFIP y con ayuda del error cuadrático medio buscan similitudes entre cada región sin embargo, no analizan imágenes de mayor tamaño que 512×512 px y tampoco mencionan cuántos píxeles traslaparon los br así como tampoco la unidad de medida del tiempo de codificación y decodificación. Así, numerosos trabajos como en [17,37,38] proponen mejoras a la CFI a partir de segmentar la imagen y encontrar similitudes entre regiones para eliminar información redundante pero sin mejora en el tiempo de codificación de la imagen.

Por otro lado, en [39] modelan ciertas regiones de la imagen con una hipergráfica ponderada, donde cada arista tiene el número de bytes guardados, con el objetivo de optimizar las tasas de compresión, es decir, el número final de bytes que usan las imágenes, por lo que encuentran el conjunto de aristas con el peso máximo que cubre toda la imagen dividida por regiones. Otra forma de abordar el problema inverso de los SFI sin tener mejoras en el tiempo de búsqueda de los códigos fractales asociados a la imagen.

Por su parte, los AG han sido usados para resolver los problemas de optimización relacionados con la compresión fractal de imágenes. Por citar un ejemplo, Al-Bundi y otros autores, en su trabajo denominado “Crowding Optimization Method to Improve Fractal Image Compressions Based Iterated Function Systems" [36] alteran los mecanismos de selección y cruza para obtener mejores individuos, a este método le llaman “método de aglomeración". Reportan tiempos de codificación como de decodificación sin embargo cometen el error de no mencionar la métrica con la que han documentado estos tiempos por lo que podría ponerse en duda si realmente logran la reducción significativa de dichos tiempos.

Es importante mencionar que dichos autores en sus trabajos de investigación han mejorado, en algunos casos, el tiempo de codificación fractal, sin embargo no ahondan en el número

de iteraciones que requiere el SFI ni sobre los coeficientes óptimos, así como tampoco en el análisis de imágenes complejas mayores a 512×512 px. Es por ello que con la adaptación e implementación del AG aplicado a este problema se pretende reportar una mejora considerable en la búsqueda de dichos coeficientes para una mejor compresión.

2

Capítulo 3

Metodología CFI

Diversos autores están inmersos en esta área de investigación debido a que al hacer compresión de imágenes mediante codificación fractal se generan grandes retos tales como lograr una reconstrucción de la imagen lo suficientemente aceptable y con un nivel de compresión alto y encontrar un algoritmo que permita una codificación en tiempo razonable. Es por ello que la metodología propuesta en este trabajo de investigación hace frente a encontrar la codificación fractal adecuada de la imagen en un tiempo sumamente razonable. A su vez se desea que la imagen resultante en el proceso de decodificación sea lo suficientemente cercana a la original.

Como primer instancia se hablará del algoritmo de CFI por segmentación de bloques y finalmente se implementará un AG que use este algoritmo previo, como base para ciertos parámetros del propio AG.

3.1. CFI mediante segmentación por bloques

La técnica para la compresión de imágenes basada en la CFI es completamente diferente de las técnicas tradicionales de compresión de imágenes, pues encontrar un SFI adecuado y óptimo que permita la reconstrucción de la imagen, es un problema que sigue abierto y más aún encontrar un algoritmo eficiente que consiga o encuentre ese SFI resulta un reto en este campo de la investigación.

La idea general de segmentar por bloques haciendo uso de SFI, se basa en tomar una par-

tición de la imagen original para encontrar autosemejanza entre estas secciones y lograr una reconstrucción total con pocas de ellas, mejorando así la calidad de compresión. Por ejemplo, la Figura 3.1 muestra una imagen dividida en bloques domino representados por las regiones enmarcadas en color negro, así como también las regiones enmarcadas en color azul y rojo; éstas representan redundancia de información, por lo que es posible que si se le aplica una transformación afín al bloque amarillo, obtengamos alguno de los bloques verdes, es entonces donde podremos eliminar redundancia de información en la imagen.

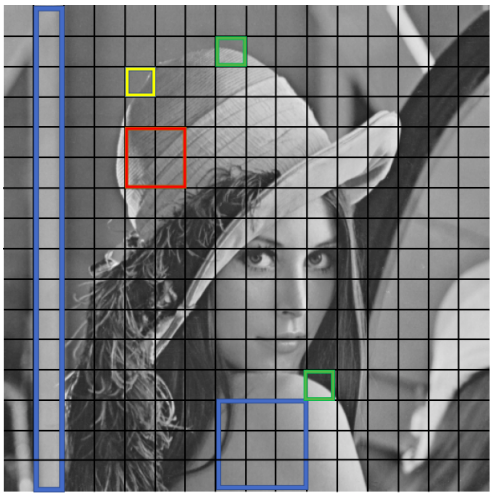


Figura 3.1: Representación de bloques, Lenna 256 × 256px.

Algunos autores abordan el tema a partir de buscar la mejor forma de segmentación de la imagen [40, 41]. Davione y Chassery en [42] presentan una nueva partición basada en triángulos y cuadriláteros, ganando ventaja en términos de adaptabilidad en la imagen sin embargo, los tiempos de codificación no mejoraron y los valores de la función objetivo no tuvieron mucha diferencia entre ellos, por otra parte, visualmente la reconstrucción no fue la mejor.

Para resolver el problema de la CFI requerimos de dos procesos, codificar la imagen para obtener los códigos fractales y, posteriormente con esos códigos, reconstruir la imagen usando el proceso de decodificación tal como se muestra de forma general, en la figura 3.2.

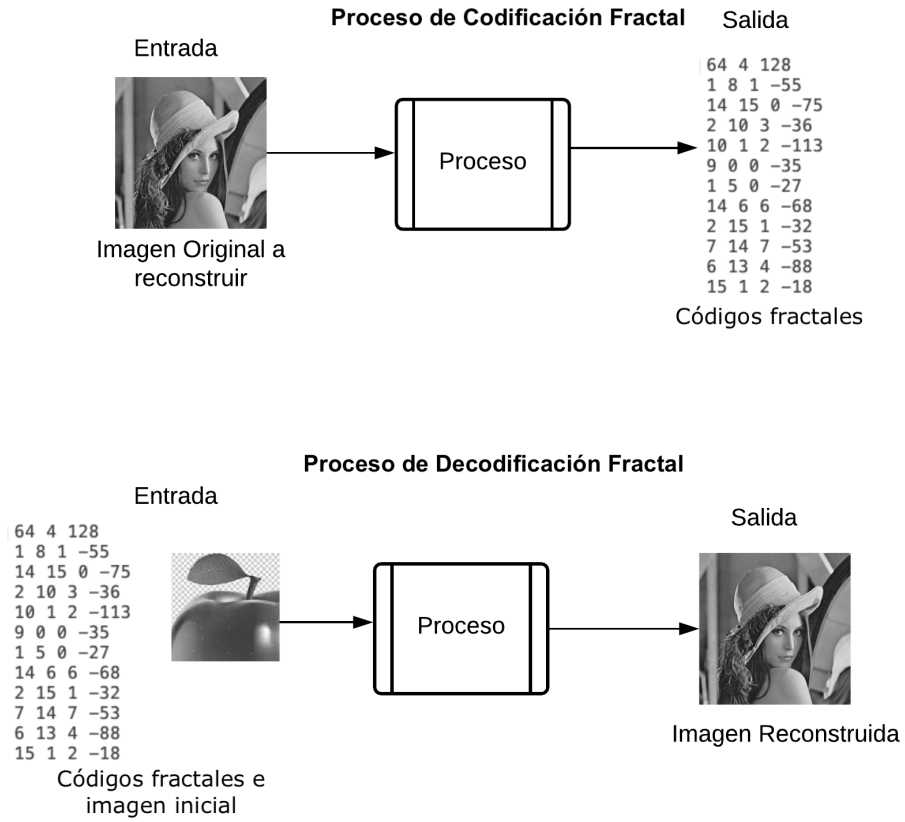


Figura 3.2: Representación visual de los procesos de codificación y decodificación fractal.

La técnica propuesta por el matemático Arnaud Jacquin [43] anuncia particionar la imagen original en bloques dominio y bloques rango traslapados, de tal manera que en el proceso de búsqueda se encuentren los bloques rango que mejor reconstruyan los bloques dominio y por medio de la Ec. (2.4) del SFI se reconstruya la imagen original.

3.1.1. Algoritmo de codificación

1. La imagen original Im_O se particiona en dos grupos de bloques cuadrados: bloques rango BR y bloques dominio BD. Si consideramos r como el número de píxeles, entonces cada **bloque dominio** $\{bd_i | bd_i \in BD\}$ para $i = 1, ..., p$; será de tamaño $2r \times 2r$, mientras que para cada **bloque rango** $\{br_j | br_j \in BR\}$ para $j = 1, ..., q$; el tamaño será de $r \times r$ píxeles, es decir la mitad de tamaño de los bloques dominio.

   Debido a que los BD son del doble de tamaño de los BR, el factor de contractividad

c se mantiene en $\frac{1}{2}$. Con ello aseguramos que la Ec. (2.1) del Teorema del Collage se cumple como sigue:

$$d_H(Im\_O, Im\_R) \leq 2d_H(Im\_O, \bigcup_{i=1}^{l} w_i(Im\_O)) \tag{3.1}$$

Si el c fuera mayor, es decir que se encuentre en $\frac{1}{2} < c < 1$, entonces se tendría del lado derecho de la ecuación un valor mayor a 2 y cambiaría así la forma de segmentación de la imagen. Por otro lado, si pasa que c se encuentre en $0 < c < \frac{1}{2}$, se tendría entonces que aumentar el conjunto de transformaciones W necesarias para reconstruir la imagen.

2. Para que se pueda comparar un br_j con un bd_i, debido a que se desea almacenar el br_j que más se asemeja al bd_i, es necesario que ambos cuenten con el mismo tamaño, por lo cual previo al procedimiento, se hace una reducción de los bd_i tomando el valor promedio de cuatro píxeles contiguos (no en linea recta sino de forma cuadrada) debido a que el número de píxeles de un bd_i es el cuádruple de un br_j. Con ello aseguramos que ambos bloques tienen el mismo tamaño y así obtenemos los bloques dominio escalados bde_i.

   Puesto que hay regiones que suelen ser más claras o más oscuras, es por ello que se debe calcular su valor de **luminancia promedio** $s_i \in [0,...,255]$ para $i = 1,...,l$. Este cálculo se hace a partir de la diferencia del valor promedio de píxeles de cada bde_i y cada br_j. Este parámetro s_i, se suma al bloque rango br_j de la comparación, dando así un bloque rango modificado br_j^*.

3. A cada br_j^* se le aplica cada una de las transformaciones mencionadas en la Sección 2.1.1.1 de forma consecutiva. Los ocho bloques br_j^* resultantes son comparados con el bloque dominio bd_i mediante el criterio de distorsión RMSE en la Ec. (2.14) y finalmente se selecciona el bloque rango transformado br_j^* con el menor valor de distorsión, es decir, el bloque rango transformado que más se pareció al bloque dominio

4. En un archivo (con extensión `.fic` de Fractal Image Compression por ejemplo) se almacenan las coordenadas (x_j, y_j) del br_j^*, la transformación t_i y el cambio de luminancia s_i, que mejor tuvo similitud con el bde_i en cuestión, estos parámetros (x_j, y_j, t_i, s_i) son denominados código fractal o mapa de afinidad del bloque br_j^*.

En la Figura 3.3 se visualiza un ejemplo de la estructura del archivo que contiene los mapas de afinidad como sigue: los primeros dos valores son las coordenadas (x, y) del br^* que tuvo mayor similitud al bde. El tercer valor numérico es la transformación t y el último valor es el cambio de luminancia s. De tal archivo, el primer renglón menciona cuántos bloques se utilizaron para segmentar la imagen, el tamaño de los bloques y el tamaño de la imagen respectivamente. Del segundo renglón en adelante, se muestran los mapas afines.

```
64 4 128
1 8 1 -55
14 15 0 -75
2 10 3 -36
10 1 2 -113
9 0 0 -35
1 5 0 -27
14 6 6 -68
2 15 1 -32
7 14 7 -53
6 13 4 -88
15 1 2 -18
7 6 2 -80
10 2 0 -93
14 6 1 -59
8 14 6 -22
13 15 2 -46
12 1 5 -18
14 8 5 -92
0 14 0 -84
```

Figura 3.3: Parte de los mapas de afinidad para una imagen de 128 × 128px con partición de bd de 16 × 16px.

El algoritmo de codificación arrojará tantos renglones en el archivo como se tengan de bd debido a que cada bloque se compara con un br^* y se guarda el mejor. Más aún, la cantidad de bd depende del tamaño que estos tengan debido a la relación $(\frac{n}{m})^2$ siendo n el tamaño de la imagen original y m el tamaño del bloque. La cantidad de bloques rango viene dada por $(\frac{n}{2} - m + 1)^2$. Con una imagen de tamaño 128 × 128px y una partición de bd de tamaño 32 × 32px, se tiene un total de $\frac{128}{32} = 16$ bd. Para este ejemplo, el tamaño de los br es entonces de 16 × 16px (la mitad de los bd), por lo que se dispondrá de un total de $(\frac{128}{2} - 16 + 1)^2 = 2401$ br.

En el Algoritmo 2 observamos el algoritmo de codificación fractal a partir de realizar segmentación por bloques traslapados en imágenes a escala de grises y que da como resultado el mapa de afinidad para la imagen original. A su vez en la Figura 3.4 podemos observar los pasos a seguir por la CFI.

Resultado: mapa de afinidad

leer_imagen(Im_O);

extracción de bloques dominio BD de Im_O;

extracción de bloques rango BR de Im_O;

Escala $\frac{1}{2}$ de bloques dominio $\rightarrow bde$;

para bde_i **hacer**

  para br_j **hacer**

    $s_i \leftarrow$ cambio de luminancia(bde_i, br_j);

    $br_j^* \leftarrow$ suma(br_j, s_i);

    para *transformación* $t = 1$ *hasta* $t = 8$ **hacer**

      $br_t \leftarrow$ copia(br_j^*);

      $br_t^* \leftarrow$ transformación(br_t, t_i);

      dist $\leftarrow$ RMSE(bde_i, br_t^*);

      si *dist* < *dist_min* **entonces**

        almacena_información(br_t^*, t, s);

      fin

    fin

  fin

  almacena_mapa(x_j, y_j, t_i, s_i);

fin

Algoritmo 2: CFI a partir de bloques cuadrados traslapados.

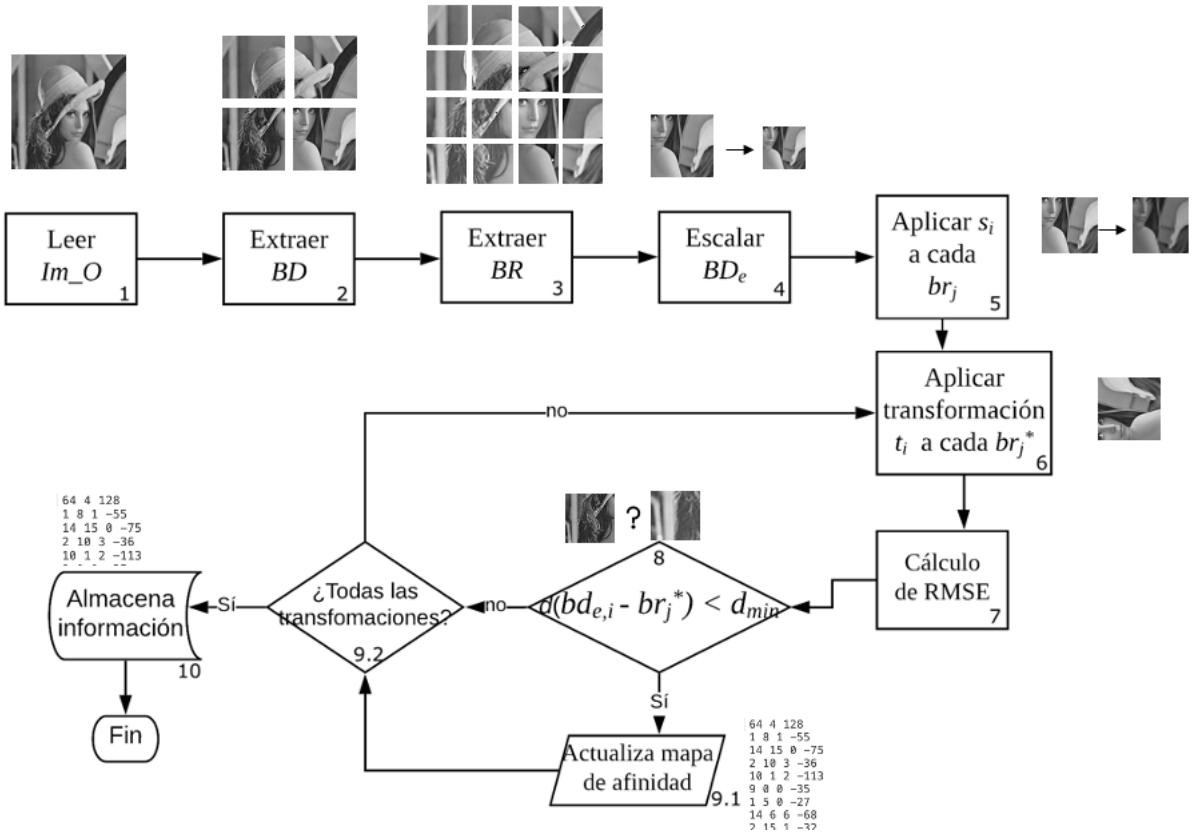


Figura 3.4: Diagrama de flujo de codificación para la CFI por bloques traslapados.

3.1.2. Algoritmo de decodificación

El proceso de decodificación requiere de los siguientes pasos:

1. Primero se lee el archivo con los mapas de afinidad y se identifica el tamaño de la imagen que se va a reconstruir para delimitar el espacio donde se pintarán los píxeles. Este espacio lo nombramos como lienzo.

2. Posteriormente se particiona el lienzo en el número de regiones que tendrá la segmentación, es decir el número de entradas en el mapa de afinidad.

3. De una imagen arbitraria se identifica el br_j que resultó más cercano al bd_i por medio de sus coordenadas (x_j, y_j).

4. Se suma el cambio de luminancia s_i, asociado de ambos bloques, al br_j.

5. Se aplica la transformación t_i y por último se coloca el bloque en la posición correspondiente en el lienzo. Este proceso se hace para todas las entradas del mapa de afinidad que están en el archivo emitido por la CFI contando como una iteración.

6. Se calcula la distancia que hay entre la imagen original y la nueva imagen por medio de alguna de las métrica mencionadas en la Sección 2.4. Si la distancia aún es muy grande se aumenta el número de iteraciones, α, de la siguiente forma: la imagen obtenida con una iteración ahora será la imagen inicial arbitraria para aplicar de nuevo CFI. El criterio de parada se da cuando la distancia entre ambas imágenes se estabiliza a un valor.

En el Algoritmo 3 podemos observar el pseudocódigo para decodificación fractal de imágenes que da como resultado la imagen reconstruida R. A su vez en la Figura 3.5 se muestran los pasos a seguir por la decodificación fractal.

Resultado: Imagen reconstruida (Im_R)
leer mapas de afinidad;
leer imagen inicial (Im_I);
para *iteraciones* $1 < \alpha$ **hacer**
  para $1 <$ *mapas* **hacer**
    $br_j \leftarrow$ extracción de bloques(Im_I, x_j, y_j);
    $br_j^* \leftarrow$ suma(br_j, s_i);
    $br_j^* \leftarrow$ aplicar_transformación(br_j^*, t);
  fin
  colocar(Lienzo, br_j,*);
  $Im\_I \leftarrow$ Lienzo;
  si *calcula_distancia(Im_O, Lienzo)* $< d_{min}$ **entonces**
    break;
  fin
  en otro caso
    $\alpha++$;
  fin
fin
pintar_Imagen(Lienzo);

Algoritmo 3: Decodificación para compresión fractal de imágenes.

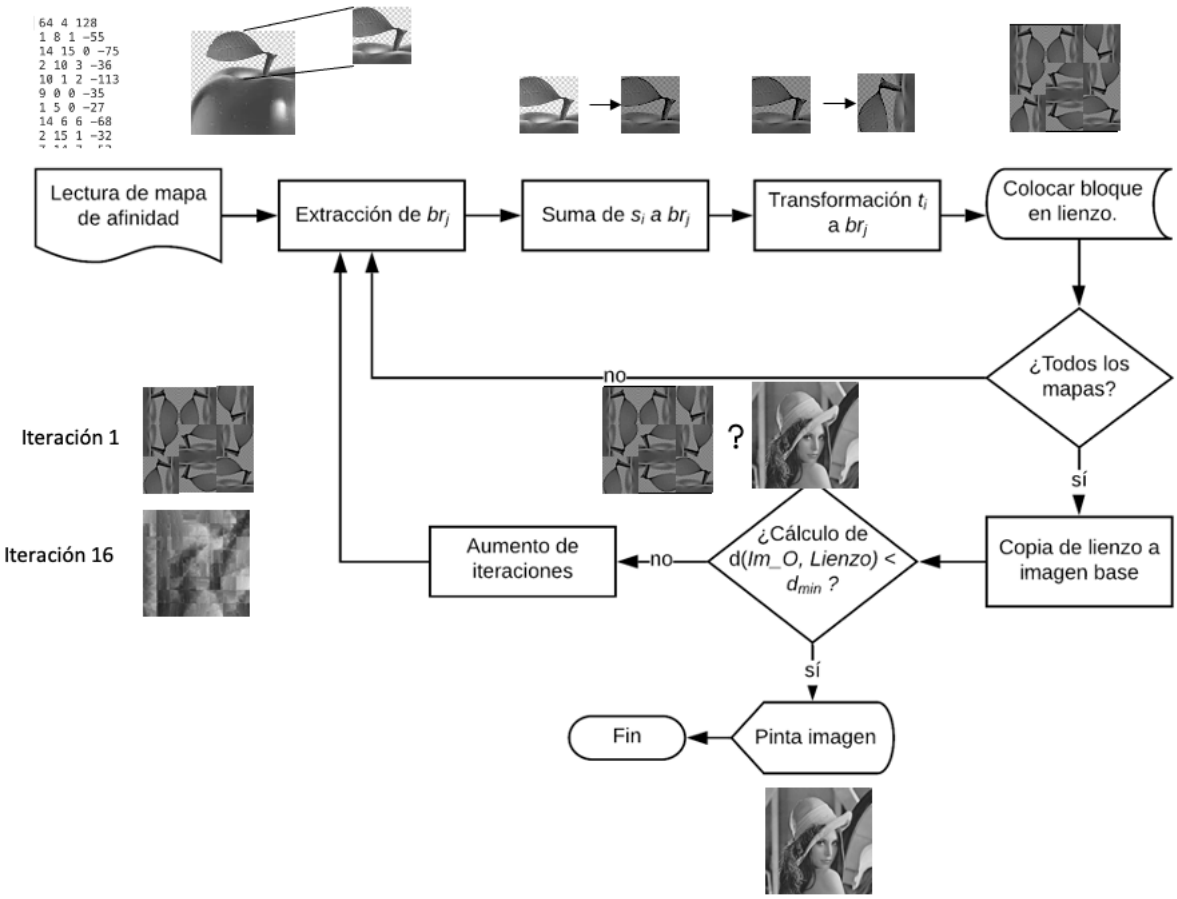


Figura 3.5: Diagrama de flujo de decodificación para la CFI.

Si aún cuando se aumente el número de iteraciones la imagen resultante no es una buena aproximación a la imagen original, entonces se prueba otra partición más pequeña de bloques en el algoritmo de codificación.

3.1.3. Complejidad de la CFI

Como se ha mencionado, la compresión fractal es una técnica de compresión de datos con pérdida que atrajo mucha atención en los últimos años, principalmente por su uso en la compresión de imágenes. Trata de eliminar información redundante de partes que presenten semejanzas o correlaciones entre píxeles debido a las partes similares que hay en las imágenes. Hannes Hartenstein, *et. al*, en su publicación *"On The Inverse Problem of Fractal Compression"* [44] demuestran que el problema inverso de la codificación fractal es NP-Duro.

Realizar un buen algoritmo de codificación de modo que éste nos arroje los coeficientes óptimos requiere de tiempos de cómputo elevado debido a que el espacio de búsqueda es muy grande y éste depende de las imágenes que se quieran reconstruir. En contraste, realizar un algoritmo de decodificación para que se reconstruya la imagen, tarda un tiempo mucho menor que la parte de codificación.

Por lo tanto, en tiempos de computo, este problema es desproporcionado debido a que el proceso de codificación depende del número de los bloques en los que se divide la imagen; si éste aumenta, también lo hará el tiempo que se emplea para realizar un gran número de operaciones entre bloques y píxeles, en contraste con el proceso de decodificación, que se hace de manera iterativa solo para los bloques no traslapados que se desean cubrir.

Si aplicamos el algoritmo de codificación para encontrar el mapa de afinidad en una imagen de 1024×1024 px con bloques dominio de tamaño 4×4 px, bloques rango traslapados de 2×2 px y 8 transformaciones afines se tendrían que hacer $136,902,606,848$ comparaciones, pues recordemos que el número de bloques dominio está dado por $\left(\frac{n}{m}\right)^2$ y la cantidad de bloques rango traslapados por $\left(\frac{n}{m} - m + 1\right)^2$ por lo tanto tenemos lo siguiente:

$$\left(\frac{n}{m}\right)^2 = \left(\frac{1024}{4}\right)^2 = 65,536 \tag{3.2}$$

$$\left(\frac{n}{m} - m + 1\right)^2 = \left(\frac{1024}{2} - 2 + 1\right)^2 = 261,121 \tag{3.3}$$

$$65,536 \times 261,121 \times 8 = 136,902,606,848 \tag{3.4}$$

Hasta la actualidad, no se conoce ningún método de bajo costo para la resolución de este problema. Cuando al matemático Barnsley, en 1988 anunció que lo había resuelto, tomaba alrededor de 100 horas codificar una imagen y alrededor de 30 minutos decodificarla, aún cuando se guiaba el proceso.

En el siguiente capítulo veremos que el tiempo de codificación para la CFI es aceptable debido a que es mucho menor que estos 30 minutos mencionados y más aún, hacer uso de bloques traslapados 1 píxel logramos obtener una mejor compresión de la imagen debido a que el coeficiente de distorsión δ es muy pequeño.

Debido a que el algoritmo de compresión fractal es un algoritmo de compresión con pérdida del que no se exige obtener un óptimo global de la función objetivo, es deseable entonces encontrar métodos de búsqueda que se acerquen a una solución potencial y que aceleren el proceso de codificación fractal.

3.2. Algoritmo Genético aplicado a la CFI

El espacio de búsqueda para la reconstrucción de imágenes a escala de grises, se encuentra definido por: la cantidad de bloques dominio, la cantidad de bloques rango traslapados, el número de isometrías y el espectro de intensidad del píxel. Por ejemplo para una imagen de 1024 × 1024px se tendrían que analizar 2^{31} posibilidades, ya que $2^{10} \times 2^{10} \times 2^3 \times 2^8 = 2^{31}$ porque $1024 = 2^{10}$, con 8 isometrías son $8 = 2^3$ y $256 = 2^8$ son los valores posibles a tomar los píxeles incluyendo al 0, pues hay 256 valores entre el 0 y 255 y haciendo búsqueda exhaustiva, el procedimiento se vuelve muy lento. Es por ello que describimos a continuación las estructuras básicas que empleamos para adaptar un AG a este problema de minimización y poder mejorar tiempos y valores de la f en comparación con el método original de Jacquin.

3.2.1. Modelo de cromosoma aplicado a nuestro problema.

Debido a que se desea buscar un emparejamiento entre el bloque dominio bd_i y el mejor bloque rango br_j, la longitud del cromosoma queda determinado por la cantidad de bd_i que se deseen obtener de la imagen a reconstruir. Más aún, cada cromosoma codificará un mapa de afinidad.

Cada gen representa los códigos fractales del bloque o sección de la imagen que mejor se asemeja al bd_i, por lo tanto, tendrán tantos genes como bd_i tengamos. Cada gen contiene una transformación w la cual engloba a las coordenadas x_i^k y y_i^k, la transformada $t_i^k \in \{0, ..., 7\}$ y el cambio de luminancia $s_i^k \in \{0, \ldots, 255\}$ asociados al i-ésimo bloque del $k-$ésimo cromosoma.

En la Figura 3.6 se puede observar la representación del cromosoma 1. La imagen original se divide en bloques bd_i y a cada uno le corresponde una transformación w, con la unión

de ellas se forma un SFI. Cada SFI es un cromosoma del AG y por tanto un individuo de la población.

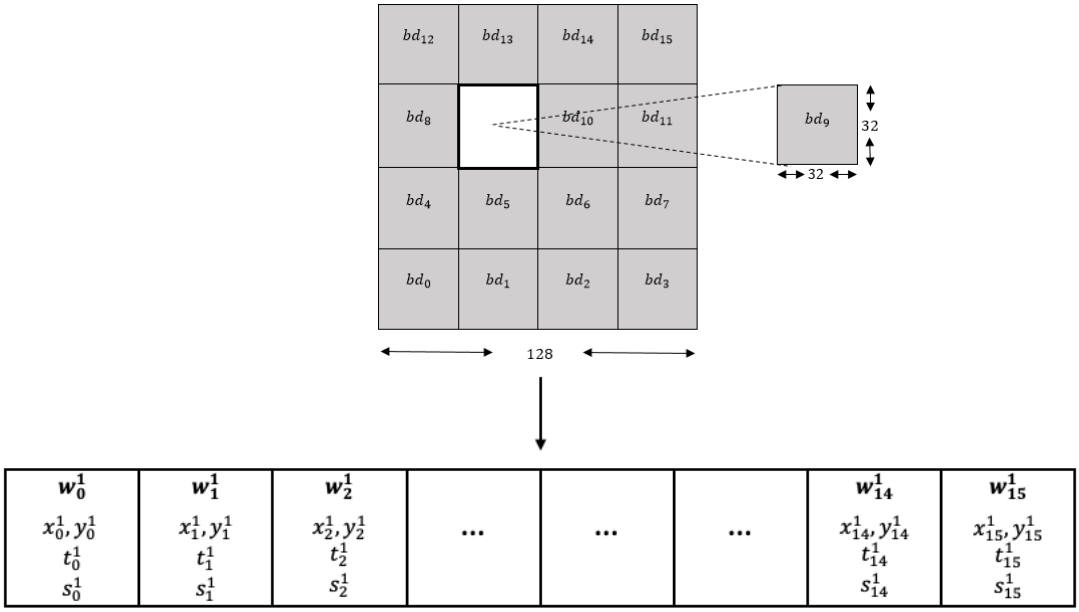


Figura 3.6: Representación del cromosoma 1.

3.2.2. Población inicial

La población inicial es un conjunto de individuos o cromosomas elaborados de tal forma que representen el espacio de búsqueda a explorar. En ese sentido, se probaron dos formas de inicializar la población. Con una población de 20 individuos, la primera forma se da como sigue: Los primeros 5 individuos se crearon a partir de extraer de cada bloque dominio la sección como sigue: del individuo 1 se extrajo la parte central de cada bloque, para el individuo 2 se extrajo la esquina inferior izquierda de cada bloque, para el individuo 3 con la esquina superior izquierda, para el individuo 4 con la esquina inferior derecha y por último para el individuo 5 con la esquina superior derecha, tal como se observa en la Figura 3.7, y con la transformación de identidad.

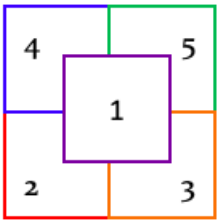


Figura 3.7: De cada bd se extrajo la sección que se enmarca con los colores respectivos, para cada uno de los primeros 5 individuos.

Los quince individuos restantes se construyeron de forma aleatoria. En estos individuos, al $i-$ésimo gen se le asigna una tupla aleatoria $\{(x_i, y_i), t_i\}$; el parámetro s_i se calcula una vez que se asigna la tupla anterior. A 5 de esos 15 individuos seleccionados de forma aleatoria, con búsqueda exhaustiva, se encontró el bloque rango transformado que más se le asemejó a un bloque dominio seleccionado al azar por medio de la distancia RMSE.

La segunda forma se implementó como sigue: para cada bloque del individuo 1 se le asignó el primer bloque rango traslapado correspondiente al conjunto de bloques dado por la división entre el número total de bloques traslapados y el número total de bloques dominio. Por ejemplo, para una imagen de 256×256px con bd_i de 4×4px, cada conjunto tendría 3 bloques rango traslapados. En el primer conjunto estarían los bloques del 0 al 2, en el segundo del 3 al 5 y así sucesivamente, tal como se muestra en la Figura 3.8.

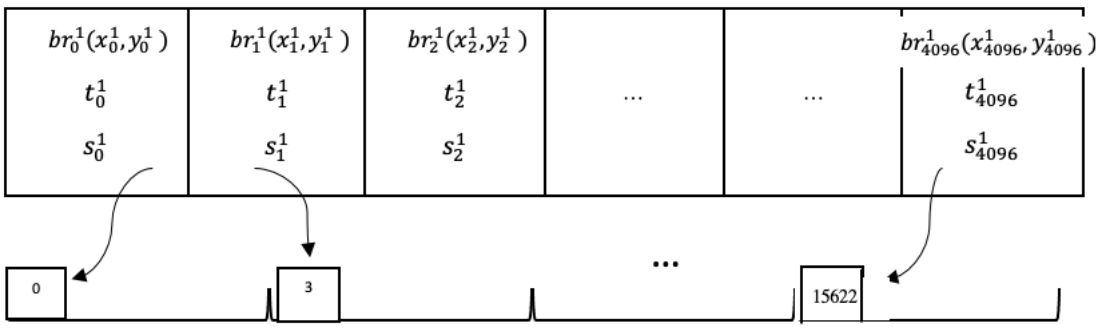


Figura 3.8: Representación de individuo 1 para una imagen de 256×256px con bd de 4×4px.

Los individuos del 2 al 6 se construyeron a partir de asignar al $i-$ésimo gen una tupla aleatoria $\{(x_i^k, y_i^k), t_i^k\}$. Aunado a ello, a cinco genes de estos individuos se les aplicó la operación

adicional de en encontrar, mediante búsqueda exhaustiva, el bloque rango transformado que minimice la distancia RMSE a un bloque dominio seleccionado al azar. Los 14 individuos restantes se generaron del mismo modo anterior a excepción de la operación adicional de búsqueda exhaustiva.

3.2.3. Función objetivo

Al calcular la distancia euclidiana que hay entre la imagen original y la resultante dada por el SFI, $d(Im\_O, W(Im\_O))$ se está calculando la distancia entre la imagen original y el atractor del sistema $d(Im\_O, A)$, por ende podemos estimar esta distancia por medio de alguna de las métricas ya mencionadas.

Con base en ello, en una variante del AG, probamos evaluar la función objetivo de cada cromosoma o individuo a partir de generar la reconstrucción de la imagen como parte del AG, es decir, ocupar el algoritmo de decodificación como parte del AG, con lo que tenemos la función objetivo en la Ec. (3.5).

En la segunda variante del AG, el algoritmo de decodificación no forma parte del AG, sino solo la evaluación de cada gen en el cromosoma. Recordemos que se desea encontrar el mejor bloque rango transformado que se asemeje al bd_i, de modo que la función objetivo queda en la Ec. (3.6) donde k es el cromosoma.

$$d(Im\_O, Im\_R) = \sqrt{\frac{1}{n^2}\sum_{x=1}^{n}\sum_{y=1}^{n}(Im\_O(x,y) - Im\_R(x,y))^2} \tag{3.5}$$

$$\sum_{k=1}^{l}(d_k(BD,BR)) = \sum_{k=1}^{l}\left(\sqrt{\frac{1}{m^2}\sum_{i=1}^{p}\sum_{j=1}^{q}\left(bd_i^k(x_i^k,y_i^k) - br_j^{*k}(x_j^k,y_j^k)\right)^2}\right) \tag{3.6}$$

3.2.4. Operadores genéticos

3.2.4.1. Selección

El operador de selección nos permite dar una probabilidad a cada individuo de la población conforme a su función de aptitud. Para este problema de minimización, deseamos que el individuo con el menor valor de función objetivo tenga una probabilidad mayor de selección y el que tenga el peor valor tenga una probabilidad muy baja de selección.

La Ec. (3.7) representa la probabilidad de selección (ps) que se le asignó a cada individuo k a partir de reconstruir la imagen en el AG y le Ec. (3.8) representa la probabilidad de selección que se le asignó a cada individuo k a partir de la evaluación de cada gen en el cromosoma donde la reconstrucción no forma parte del AG.

$$ps_k = \frac{\exp(-d_k(Im\_O, Im\_R))}{\sum_{k=1}^{l} \exp(-d_k(Im\_O, Im\_R))} \tag{3.7}$$

$$ps^k = \frac{\exp(-d^k(bd_i^k, br_j^{*k})}{\sum_{i=1}^{p}\sum_{j=1}^{q} \exp(-d^k(bd_i^k, br_j^{*k}))} \tag{3.8}$$

Posteriormente por medio de *selección por ruleta* se escogen los individuos a los que se les aplicarán los operadores genéticos de cruza y mutación.

3.2.4.2. Cruza

Todos los individuos de la población tienen el mismo tamaño por ende, el punto de cruza puede ser aleatorio. Cabe mencionar que la cruza PMX arrojó resultados no esperados debido a que, para algunos individuos, la recombinación de genes se ciclaba y el algoritmo no terminaba y en otras ocasiones, la cruza PMX le asignaba el mismo gen al nuevo descendiente, con lo cual afectaba de forma significativa la solución. Más aún afectaba la diversidad que se esperaba en la población; por esta razón es que decidimos solo utilizar el operador de cruza en un punto pues resultó factible arrojando buenos resultados.

Para tener una mayor diversidad todos los individuos se cruzan en parejas, por esta razón se requiere que el número de individuos en la población sea par. En la Figura 3.9 se selecciona el padre 1 y padre 2, al azar se genera el punto de corte entre un gen y otro para intercambiar genes dejando dos hijos como descendientes.

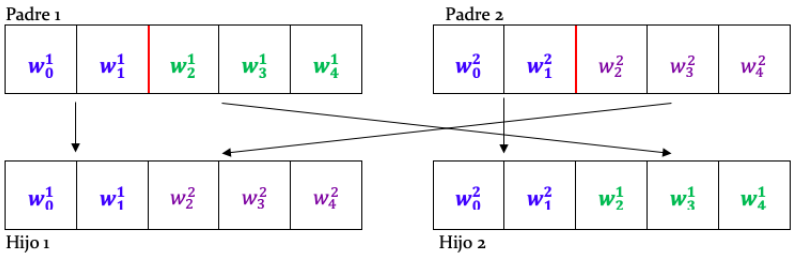


Figura 3.9: Cruza en un punto.

3.2.4.3. Mutación

El operador de mutación requiere de una probabilidad para saber cuál hijo es el que se va a mutar. Para tomar esta decisión, inicialmente se fija una probabilidad de mutación pm, en este caso de fue 0.4 ya que fue la que mejor resultados dio ya que, se generó un número aleatorio σ entre 0 y 1 y si $\sigma \leq pm$ entonces el hijo se muta, de lo contrario no es afectado por esta decisión y pasa directamente a la siguiente generación. Con ello aseguramos que haya diversidad en la población y que tampoco haya una convergencia prematura. Si el hijo es mutado, entonces se genera un número aleatorio entero entre 0 y el número total de bd, para seleccionar el gen que será alterado.

La alteración consiste en encontrar, mediante búsqueda exhaustiva, el bloque rango transformado que minimice la distancia RMSE a este bloque. Este procedimiento se realiza para uno o cinco genes, por lo que es importante mencionar que evidentemente el tiempo de codificación aumentará cuando se alteren 5 genes.

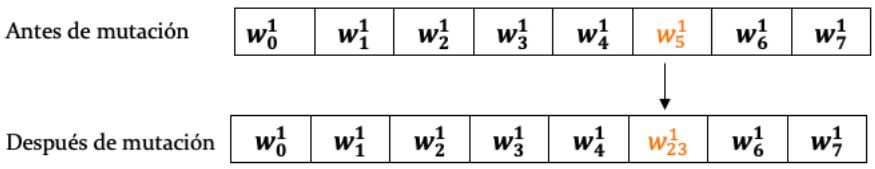


Figura 3.10: Ejemplo del cromosoma 1 con 8 genes. Se selecciona el gen a alterar y se aplica búsqueda exhaustiva para encontrar el bloque rango que se acerca más al bloque dominio.

En el Algoritmo 4 se encuentra el pseudocódigo del AG implementado para la CFI; éste hace uso del algoritmo de decodificación para reconstruir la imagen en un mismo proceso.

1. Inicialmente se segmenta la imagen en bd y br.
2. Posteriormente se establecen los parámetros del SFI tales como el número de iteraciones y el tamaño de los bloques. A su vez se establecen los parámetros de algoritmo genético tales como el número de generaciones, la pm y el tamaño de la población.
3. Se crea la población y comienza el algoritmo a operar hasta que no haya una condición de paro, que es el número de generaciones.
4. Una vez terminado el proceso genético, para cada individuo se reconstruye la imagen a partir de sus códigos fractales y se calcula la distancia que hay entre la imagen original Im_O y la reconstruida Im_R_k, almacenando la imagen que se acerca más a la original.

Del mismo modo, en el Algoritmo 5 se encuentra el pseudocódigo del algoritmo genético sin usar el algoritmo de decodificación para reconstruir la imagen. Una vez terminado el proceso genético se almacena el mapa de afinidad que contiene los códigos fractales de la imagen a reconstruir.

Resultado: Imagen reconstruida

Segmentar Im_O en BD y BR traslapados;

Establecer parámetros para SFI;

Establecer parámetros para AG;

Crear población inicial;

mientras *generaciones* ≤ *max_generaciones* **hacer**

  aplicar operador de selección;

  aplicar operador de cruza;

  aplicar operador de mutación;

  para $k = 1$ *hasta individuos en la población* **hacer**

    decodificación(x^k, y^k, t^k, s^k);

    si *dist(*Im_O, Im_R_k*)* < *dist_min* **entonces**

      alamacena(Im_R_k);

    fin

    en otro caso

      $generaciones++$;

    fin

  fin

fin

Algoritmo 4: AG para codificación fractal de imágenes con decodificación como parte del AG.

Resultado: mapa de afinidad

Segmentar Im_O en BD y BR traslapados;

Establecer parámetros para SFI;

Establecer parámetros para AG;

Crear población inicial;

mientras *generaciones* $\leq$ *max_generaciones* **hacer**

  para $k = 1$ *hasta individuos en la población* **hacer**

    aplicar operador de selección;

    aplicar operador de cruza;

    aplicar operador de mutación;

    dist $\leftarrow$ RMSE(bd^k, br^{*k});

    si *dist* < *dist_min* **entonces**

      almacena_información(x^k, y^k, t^k, s^k);

    fin

    almacena_mapa(x^k, y^k, t^k, s^k);

  fin

fin

Algoritmo 5: AG para codificación fractal de imágenes sin decodificación como parte del AG.

De este modo hemos explicado parte de la teoría y pasos a seguir para resolver el problema inverso del SFI a partir de la CFI, con lo cual no lleva a presentar los resultados siguientes obtenidos a partir de seguir dicha metodología.

3

Capítulo 4

Implementación y resultados

Los siguientes resultados son producto de implementar los algoritmos de CFI en lenguaje de programación de alto nivel C, y probados en una computadora MacBook Pro en sistema operativo OSx, con un procesador a 2.9 GHz Intel Core i7 y con 8GB en memoria RAM. Para el análisis de las imágenes por medio de histogramas, utilizamos lenguaje de programación Python con ayuda de una librería multiplataforma llamada OpenCv (Open Source Computer Vision).

La imagen a reconstruir es Lenna, Figura 4.1 a escalas de 128×128, 256×256 y 512×512 px a escala de grises en formato TGA y por considerarse una imagen compleja debido a que presenta una entropía de 7.54 y es apta para probar la eficiencia del algoritmo propuesto. El problema modelado a partir de SFI, se sustenta en el Teorema 2.1, por lo que el factor de contractividad queda definido en $\frac{1}{2}$.

Figura 4.1: Imagen original Im_O a reconstruir.

◦ **Espacio de búsqueda**

El espacio de búsqueda depende de la cantidad de bloques dominio, del número de bloques rango traslapados y el número de transformaciones w. En la Tabla 4.1observamos que la cantidad de bloques a examinar aumenta conforme la partición se hace más pequeña y el tamaño de la imagen original crece.

Tamaño de Im_O	BD Tamaño	Cantidad	BR Tamaño	Cantidad
128 × 128px	32 × 32px	16	16 × 16px	2401
	16 × 16px	64	8 × 8px	3249
	8 × 8px	256	4 × 4px	3721
256 × 256px	32 × 32px	64	16 × 16px	12769
	16 × 16px	256	8 × 8px	14641
	8 × 8px	1024	4 × 4px	15625
512 × 512px	32 × 32px	256	16 × 16px	58081
	16 × 16px	1024	8 × 8px	62001
	8 × 8px	4096	4 × 4px	64009

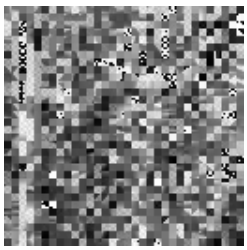
Tabla 4.1: Cantidad de bloques dependiendo de la partición y de la imagen.

Es importante mencionar que la cantidad de códigos fractales resultantes, después de aplicar el algoritmo de codificación, está en función del tamaño y cantidad de los bloques dominio, es decir, si se va a reconstruir una imagen de tamaño 512 × 512px con bd de 8 × 8px se tendrán 4096 códigos fractales.

4.1. Resultados de CFI mediante Búsqueda Exhaustiva

◦ **Reconstrucción de Imágenes**

Las imágenes de la Figura 4.2 son la reconstrucción de la imagen original de Lenna 128 × 128px a partir de hacer la segmentación de bd de 4 × 4px con 1, 4, 8, 16 y 32 iteraciones respectivamente. En la Tabla 4.2 se puede observar que la reconstrucción se estabiliza a partir de 32 iteraciones.



(a) 1 iteración, RMSE: 67.059

(b) 4 iteraciones, RMSE: 17.445

(c) 8 iteraciones, RMSE: 8.456

(d) 16 iteraciones RMSE: 8.343

(e) 32 iteraciones, RMSE: 8.335

Figura 4.2: Reconstrucción imagen de tamaño 128 × 128px con *bd* de 4 × 4px.

Del mismo modo, hemos realizado el procedimiento para la imagen de 256 × 256 px y 512 × 512 px, de ahí que se concluye que la imagen con mayor estabilidad en distancia y mayor cercanía a la imagen original es la imagen cuyo tamaño es 512 × 512 px con bloques de tamaño 4 × 4 px y 32 iteraciones, así como se muestra en de la Figura 4.3, en la Tabla 4.2 y en la Tabla4.4.

(a) *bd* de 32×32px, RMSE: 16.776

(b) *bd* de 16×16px, RMSE: 10.669

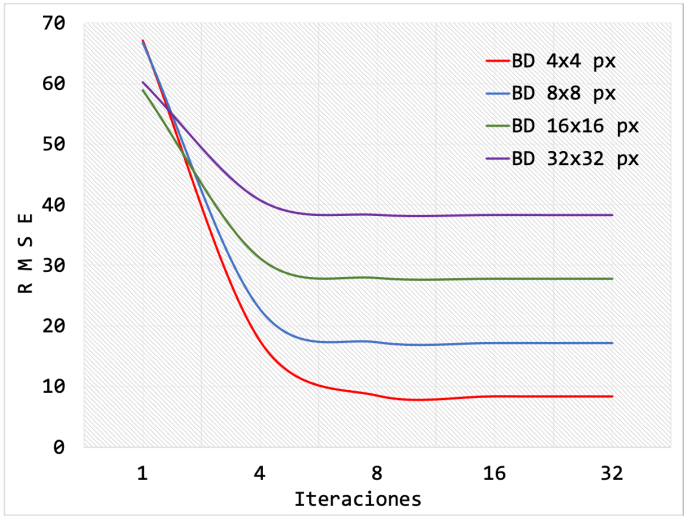
(c) *bd* de 8 × 8px, RMSE: 6.274

(d) *bd* de 4 × 4px, RMSE: 2.823

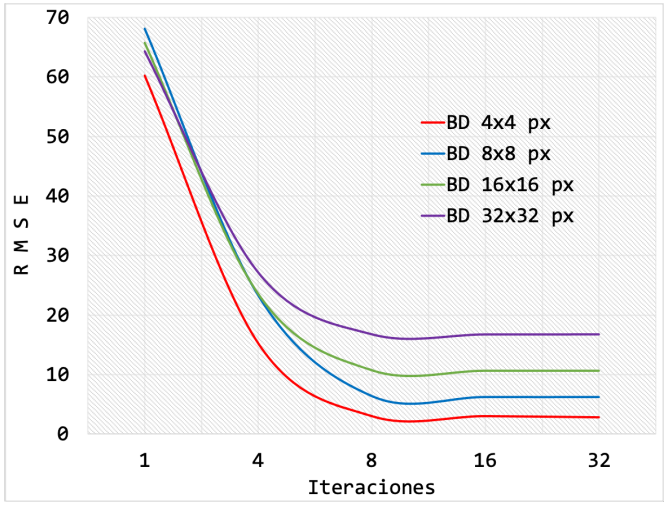
Figura 4.3: Reconstrucción de imagen de tamaño 512×512px con 32 iteraciones, a partir de diferentes tamaños de *bd*.

◦ **Función objetivo** f

La f a desarrollar es minimizar el coeficiente de distorsión δ entre la imagen original y la imagen reconstruida, dado por la Ec. (2.6). Con uso de la métrica RMSE, se observa que δ es grande cuando el tamaño del bloque también lo es, aunque se aumenten las iteraciones para el SFI. Conforme disminuye el tamaño del bloque y aumentan las iteraciones, δ es cada vez más pequeño, mostrando que, para bloques de 4×4px e iteraciones a partir de 16 en adelante se llega una imagen suficientemente aceptable como se observa en la Figura 4.4.



(a) RMSE entre Im_O y Im_R para imagen de tamaño 128×128px.



(b) RMSE entre Im_O y Im_R para imagen de tamaño 512×512px.

Figura 4.4: RMSE entre Im_O y Im_R a partir de bd de tamaño 4×4, 8×8, 16×16 y 32×32px con iteraciones de 1, 4, 8, 16 y 32.

bd	iter 1	iter 4	iter 8	iter 16	iter 32
32 × 32	60.224	40.726	38.311	38.295	38.280
16 × 16	58.912	31.154	27.902	27.781	27.780
8 × 8	66.768	22.695	17.263	17.145	17.143
4 × 4	67.059	17.445	8.456	8.343	8.335

Tabla 4.2: RMSE entre imagen original y reconstruida (Lenna) de tamaño 128 × 128px.

bd	iter 1	iter 4	iter 8	iter 16	iter 32
32 × 32	45.659	31.875	30.205	30.144	30.143
16 × 16	41.142	20.856	18.632	18.542	18.543
8 × 8	43.524	19.072	12.678	12.125	12.123
4 × 4	47.295	15.388	7.346	7.343	7.325

Tabla 4.3: RMSE entre imagen original y reconstruida (Lenna) de tamaño 256 × 256px.

bd	iter 1	iter 4	iter 8	iter 16	iter 32
32 × 32	64.319	27.162	16.792	16.767	16.776
16 × 16	65.775	23.576	10.754	10.669	10.669
8 × 8	68.078	23.313	6.433	6.276	6.274
4 × 4	60.224	15.239	3.020	3.017	2.823

Tabla 4.4: RMSE entre imagen original y reconstruida (Lenna) de tamaño 512 × 512px.

◦ **Comparativa visual de imágenes**

Para la imagen de Lenna de 512 × 512 px podemos observar en la imagen de la Figura 4.19 la diferencia de píxeles entre la imagen original y la reconstruida, es decir, los px que no se pintaron en la imagen reconstruida con 32 iteraciones y bloques de 4 × 4 px, por lo que podemos asegurar que la CFI con bloques traslapados ha dado buenos resultados.

Figura 4.5: Píxeles no pintados en la imagen reconstruida de tamaño 512 × 512 px con bd de 4 × 4 px.

Así mismo, hemos realizado un acercamiento focal de la imagen reconstruida para observar a detalle las variaciones con respecto a la imagen original y hemos notado que estas diferencias pasan a ser desapercibidas tal como se observa en la Figura 4.6.

(a) Porción de la Imagen original Im_O.

(b) Porción de la imagen reconstruida Im_R.

Figura 4.6: Comparativa visual de una parte de la imagen original y la imagen reconstruida.

◦ Análisis por medio de histogramas

En imágenes de 8 bits, puede haber hasta 256 tonos de gris. Cada píxel de una imagen en escala de grises tiene un valor de brillo comprendido entre 0 (negro) y 255 (blanco) por lo que

en un histograma, el eje x está comprendido entre estos valores, mientras que los valores del eje y son la cantidad de píxeles de la imagen que presentan ese nivel de gris concreto. Cada imagen tiene su propio histograma, pero como regla general se considera que una imagen tiene un buen contraste si su histograma se extiende ocupando casi todo el rango de tonos.

Así, el histograma que se muestra en la Figura 4.7 referente a la imagen original de Lenna en tamaño 128 × 128px en Figura 4.1, se observa que la imagen está muy bien distribuida en sus valores de píxeles, pues la gráfica no está ni más cargada hacia los valores mínimos con negros, ni a valores máximos con blancos, sino que mantienen una buena distribución, lo cual podemos observar que es una imagen con información confiable para aplicar y probar codificación fractal.

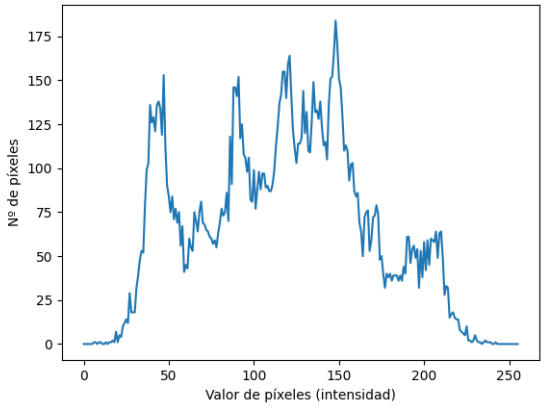


Figura 4.7: Histograma de Im_O de tamaño 128 × 128px.

En la Figura 4.8 se observa en color naranja el histograma de la imagen reconstruida con bd de 16×16px y 1 iteración, por lo que aún está alejado del histograma de la imagen original. En la Figura 4.9 se observa que el histograma de la imagen reconstruida con bd de 4 × 4 px y 32 iteraciones, es muy semejante al de la imagen original, lo cual observamos que la partición de bloques en 4×4px resultó mejor, con una reconstrucción muy cercana a la imagen original con resolución de 128 × 128px.

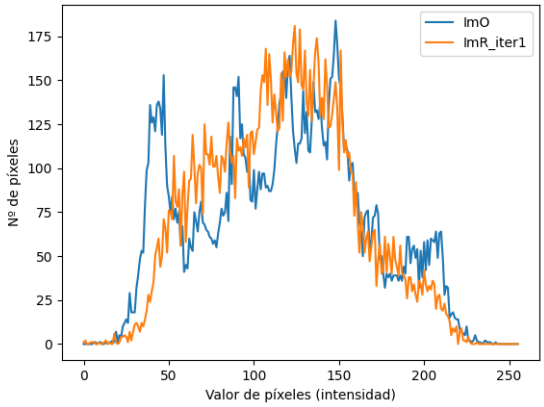


Figura 4.8: Histograma de Im_R (Lenna) de tamaño 128 × 128px con bd de 16 × 16px y 1 iteración.

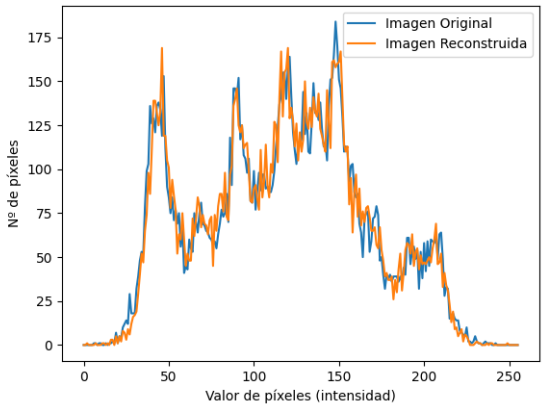


Figura 4.9: Histograma de Im_R (Lenna) de tamaño 128 × 128px con bd de 4 × 4px y 32 itercaiones.

En las Figuras 4.10 y 4.11 se muestra en color azul el histograma de la imagen original de tamaño 512 × 512px y en color naranja el histograma de las imagen reconstruida a partir de códigos fractales obtenidos con la CFI, con bd de 32 × 32 y 4 × 4px respectivamente. Nos percatamos que la mejor reconstrucción es aquella que presenta una partición de bd de 4 × 4px a partir de 16 iteraciones en la codificación fractal.

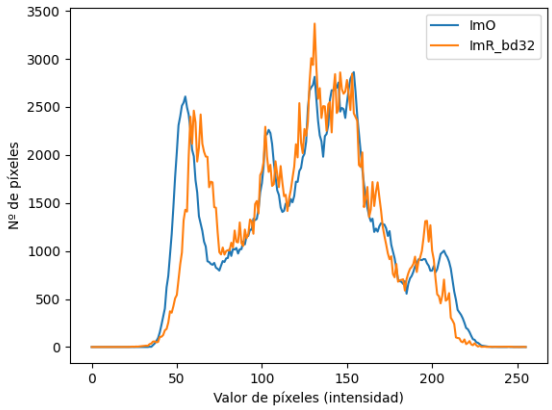


Figura 4.10: Histograma de Im_R (Lenna) de tamaño 512 × 512px con bd de 32 × 32px con 32 iteraciones.

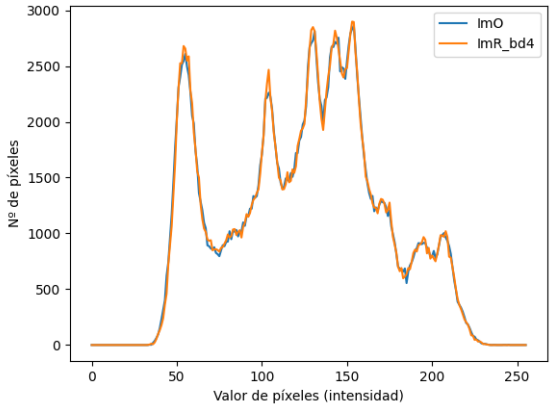


Figura 4.11: Histograma de Im_R (Lenna) de tamaño 512 × 512px con bd de 4 × 4px con 32 iteraciones.

Numéricamente realizamos la comparación de estos histogramas por 3 métodos: **Correlación**; que describe el grado de dependencia lineal que existe entre una imagen y otra; el coeficiente de correlación oscila entre -1 y +1 encontrándose en medio el valor 0 que indica que no existe asociación lineal entre las dos imágenes en estudio [45], **Intersección**; que nos muestra cuántos píxeles han coincidido en la misma posición con el mismo valor, y por último la distancia **Bhattacharyya**; la cual mide la similitud de dos distribuciones de probabilidad [46]. Al observar las Tablas 4.5 y 4.6 donde se comparan numéricamente los histogramas de las imágenes reconstruidas con diferentes tamaños de bloques y 32 iteraciones, nos encontramos que en efecto, la mejor partición es con bloques de 4 × 4px, pues hay una mejor correlación e intersección entre las imágenes, tanto para imagen de tamaño 128 × 128px como para 512 × 512px.

bd	32×32	16×16	8×8	4×4
Correlación (uni)	0.7755	0.8299	0.9684	0.9146
Intersección (px)	13114	13652	14484	1531
Bhattacharyya (uni)	0.2015	0.1653	0.1318	0.0678

Tabla 4.5: Comparativa de histogramas de Im_O a Im_R (Lenna 128×128px) con diferentes particiones.

bd	32×32	16×16	8×8	4×4
Correlación (uni)	0.9141	0.9756	0.9929	0.9985
Intersección (px)	2334	2481	2540	2581
Bhattacharyya (uni)	0.1238	0.0673	0.0362	0.0162

Tabla 4.6: Comparativa de histogramas de Im_O a Im_R (Lenna 512×512px) con diferentes particiones.

◦ **Tiempos de codificación vs decodificación**

En la Sección 3.1.3 hablamos de la complejidad del problema, por lo que un tema que resalta es el tiempo que se tarda el algoritmo en realizar la codificación fractal y el tiempo que tarda en realizar la decodificación. Con los experimentos realizados, observamos que para codificar una imagen se tarda un tiempo relativamente alto, no obstante para el proceso de decodificación se logra de forma rápida, tal como se muestra en la Tabla 4.7. Un factor importante a recalcar es que este tiempo depende también del tamaño de la imagen a reconstruir así como de la partición que se haga, pues el espacio de búsqueda aumenta cuando aumenta la partición.

Tamaño de imagen	*bd*	Cod. Tiempo (s)	Decod. Tiempo (s)
128 × 128px	32 × 32	2.0364	0.0380
	16 × 16	4.5644	0.1310
	8 × 8	6.7184	0.5055
	4 × 4	9.3206	1.8978
256 × 256px	32 × 32	66.2281	0.5133
	16 × 16	90.8560	2.0864
	8 × 8	113.4619	7.7552
	4 × 4	147.1137	31.3793
512 × 512px	32 × 32	1460.7058	7.9524
	16 × 16	1648.0601	31.3311
	8 × 8	1918.4337	127.7528
	4 × 4	2473.2578	502.0804

Tabla 4.7: Tiempos de codificación vs decodificación para imagen de diferentes tamaños y particiones con 32 iteraciones.

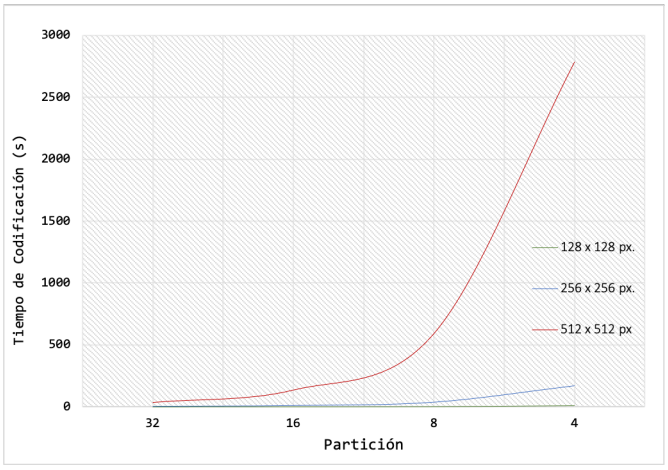


Figura 4.12: Tiempo en segundos de codificación para imágenes de tamaño 128 × 128, 256 × 256 y 512 × 512px con diferentes particiones.

Debido a que este algoritmo hace una búsqueda exhaustiva de orden $O(n^3)$ para lograr obtener los mejores mapas que minimicen el error entre la imagen original y la reconstruida, requiere de tiempo de cómputo muy elevado. Por mencionar un ejemplo, un SFI como el del helecho que se observa en la Figura 4.13 junto con sus 4 transformaciones (I_1, I_2, I_3, I_4) en la Figura 4.14 modeladas a partir de una trasnformación w consta de 24 parámetros; encontrar sus respectivos valores por fuerza bruta requeriría un tiempo de cómputo sumamente alto

cuando no hay una discretización fija para cada parámetro de w.

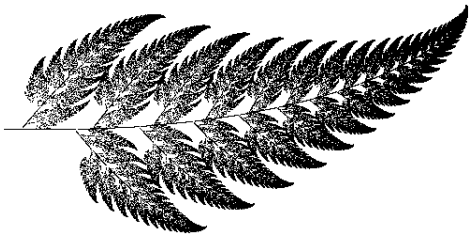


Figura 4.13: Helecho de Barnsley.

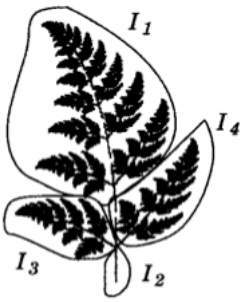


Figura 4.14: Helecho de Barnsley a partir de 4 transformaciones.

En ese sentido, de la Tabla 4.7 concluimos que el tiempo de codificación fractal está muy por encima del tiempo que se requiere para decodificar la imagen. Conforme crece el tamaño de la imagen a reconstruir, el espacio de búsqueda también y por ende el tiempo de codificación. En contraste, el tiempo de decodificación también aumenta pero en cifras más pequeñas.

4.2. Resultados de CFI a partir del Algoritmo Genético

Con el objetivo de tener una administración eficiente de la memoria, fue necesario implementar AG por medio de estructuras y memoria dinámica en lenguaje de alto nivel C. Cada individuo en la población es una estructura y cada elemento es otra estructura que tiene la información de cada gen como sigue: las coordenadas (x, y) y la transformación t se representan con un tipo de dato “int", el cambio de luminancia s se representa con un tipo de dato “short". Aunado a ello cada bloque se representa por medio de un arreglo bidimensional de tipo “unsigned char"debido a que en cada entrada del arreglo hay un valor decimal entero entre 0 y 255, donde 0 representa al color negro y 255 al color blanco.

Cuando el operador genético de cruza y de mutación se realizan, las estructuras de los bloques seleccionados no son copiadas en una nueva localidad de memoria, sino que solo se ocupan apuntadores; preservando así la memoria. Del mismo modo, la imagen es cargada en una arreglo bidimencional usando memoria dinámica, lo que permite que se adapte tanto la CFI por medio de búsqueda exhaustiva como por AG, para una imagen que va desde

tamaño 64 × 64px hasta 1024 × 1024px.

Todas las pruebas se realizaron en una computadora MacBook Pro en sistema operativo OSx, con un procesador a 2.9 GHz Intel Core i7, con 8GB en memoria RAM y con imágenes cuadradas en formato TGA a escala de grises con 8 bits. Las diferencias en las imágenes resultantes fueron en calidad y tiempo empleado. La medida de error se tomó como la distancia entre la imagen original y la imagen resultante bajo el cálculo de RMSE y PSNR.

◦ **Selección de parámetros**

La selección de los parámetros óptimos para el algoritmo genético se realizaron a prueba y error. Los parámetros considerados fueron: (1) el mecanismo de generación de la población inicial, (2) el número de genes mutados y (3) el número de iteraciones para el SFI. Con ello dejamos 20 individuos para la población inicial y 20 generaciones con una probabilidad de mutación de 0.4.

- A : población inicial.
  - A_1:inicializar población con 5 individuos a partir de la extracción central y esquinas de cada bloque como se ejemplifica en la Figura 3.7 y a 5 de los individuos restantes aplicarles búsqueda exhaustiva a 5 genes.
  - A_2: cada bloque del individuo 1 se empareja con el bloque rango traslapado para la sección correspondiente de la imagen. Los individuos del 2 al 6 se generan con bloques de forma aleatoria y se aplica la operación adicional de búsqueda exhaustiva solo a estos 5 individuos.
- B : cantidad de genes para mutar.
  - B_1: 1 gen
  - B_2: 5 genes
- C : iteraciones para SFI.
  - C_1: 4 iteraciones
  - C_2: 8 iteraciones
  - C_3: 16 iteraciones

◦ **Reconstrucción de Imágenes**

En la Tabla 4.8 se observan los resultados experimentales de ejecutar el AG a partir de las 12 combinaciones obtenidas con la selección de parámetros. La columna 1 ejemplifica el número de combinación, la columna 2 el mecanismo de generación de la población inicial, la columna 3 el número de genes mutados, la columna 4 el número de iteraciones para el SFI, la columna 5 la distancia por medio de RMSE, la columna 6 el tiempo de ejecución del AG tomando en cuenta la decodificación como parte de éste, la columna 7 el tiempo total de ejecución del AG con 20 generaciones y la última columna el tiempo que le tomó al AG sin decodificación y por generación.

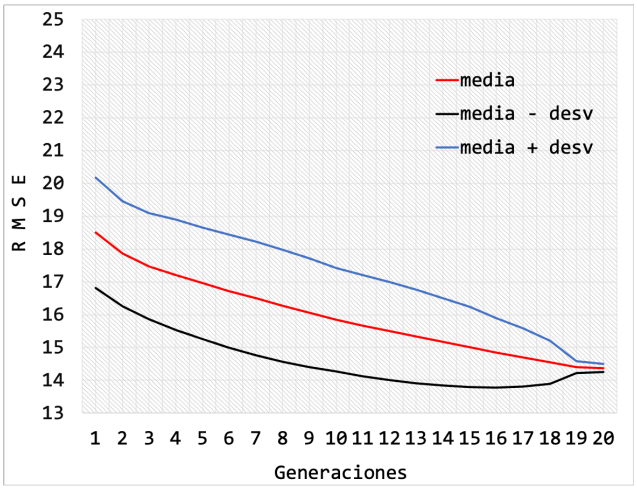
id	**A**	**B**	**C**	RMSE	tiempo 1 (s)	tiempo total (s)	tiempo 2(s)
1	A_1	B_1	C_1	14.3441	1.6175	28.0313	1.3146
2	A_1	B_1	C_2	14.3414	1.5146	28.6824	1.4357
3	A_1	B_1	C_3	14.3438	1.5978	31.7854	1.4398
4	A_1	B_2	C_1	14.3406	4.5103	79.7487	3.8671
5	A_1	B_2	C_2	14.3424	3.9898	84.0203	3.0145
6	A_1	B_2	C_3	14.3360	4.2175	82.8525	3.5892
7	A_2	B_1	C_1	23.9207	1.3254	28.7275	1.0989
8	A_2	B_1	C_2	24.2352	1.2115	31.5248	0.9982
9	A_2	B_1	C_3	24.1990	1.0848	31.3545	0.9712
10	A_2	B_2	C_1	24.1573	3.5477	80.0198	2.0165
11	A_2	B_2	C_2	24.4200	3.6884	78.1845	2.0369
12	A_2	B_2	C_3	24.2256	3.6189	81.3893	2.1942

Tabla 4.8: Resultados de la ejecución del AG probando las 12 combinaciones. La mejor distancia de cada combinación es la que se ejemplifica la segunda columna para una imagen de 512×512 px con bd de 4×4px.

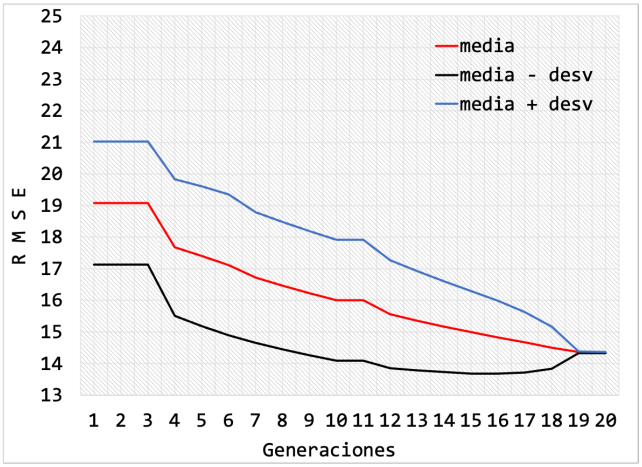
Las Figuras 4.15 y 4.16 representan la reconstrucción de las imágenes del mejor individuo tanto de la 1era generación como de la última para la combinación 6, que es a partir de implementar la forma 1 de la población, con 5 génes a mutar y 16 iteraciones para el SFI. En la Figura 4.17 se observa cómo la distancia entre la imagen original y la imagen reconstruida disminuye en cada generación, ejemplificando solo las combinaciones 1, 6 y 12.

Figura 4.15: Generación 1, RMSE: 14.34503 de la combinación 6.

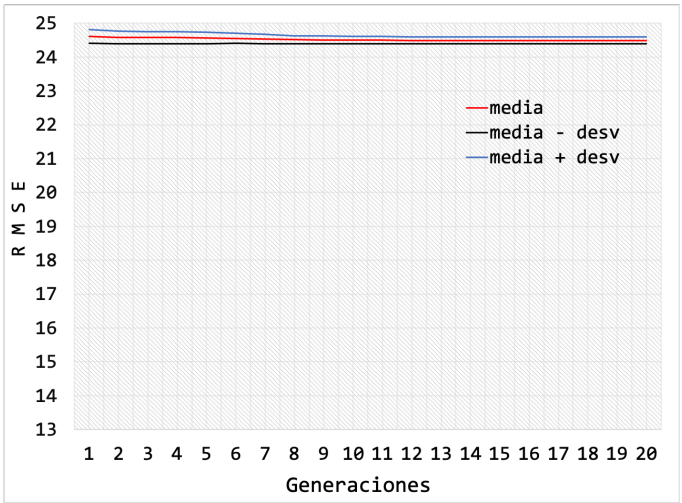
Figura 4.16: Generación 20, RMSE: 14.33608 de la combinación 6.



(a) Combinación 1, A_1, B_1 : 1 gen a mutar y C_1 : 4 iteraciones para el SFI, RMSE: 14.344104.



(b) Combinación 6, A_1, B_2 : 5 gen a mutar y C_3 : 16 iteraciones para el SFI, RMSE: 14.336088.



(c) Combinación 12, A_2, B_2 : 5 gen a mutar y C_3 : 16 iteraciones para el SFI, RMSE: 24.225648.

Figura 4.17: Comportamiento del valor de aptitud del mejor individuo a lo largo de cada generación.

De la Tabla 4.8 podemos observar que la combinación que mejor resultó está representada por id 6 con A_2, B_2, C_3, lo que indica que la población se genera de la primer forma, con 5 bloques a mutar para el operador de mutación y 16 iteraciones para el SFI. Vemos que cuando las iteraciones para el SFI aumentan, el tiempo de decodificación también lo hace por lo que es esperable que el tiempo también aumente cuando los bloques a mutar son 5, en lugar 1, y más aún cuando el tamaño de estos bloques es más grande que 4×4px.

Los tiempos de codificación para cada una de las técnicas de búsqueda comparadas en este trabajo se muestran en la Tabla 4.9; se comparan la búsqueda exhaustiva, búsqueda por medio del AG con el algoritmo de decodificación como parte de éste, (AG_1) y búsqueda por medio del AG sin el algoritmo de decodificación como parte de la reconstrucción (AG_2). La imagen a reconstruir es de tamaño 256×256 px a partir de segmentar con bloques de tamaño 4×4 px.

Los parámetros que se usaron para el AG son dados por la combinación 6, que es la que mejor reconstruye a la imagen, con A_1: población con 20 individuos donde 5 de ellos son formados como se muestra en la Figura 3.7 y a 5 de los individuos restantes se le aplicó búsqueda exhaustiva a 5 genes, B_2: 5 genes a mutar con una probabilidad de mutación de 0.44 y C_1: 16 iteraciones para el SFI.

Se observa que se redujo significativamente el tiempo de codificación a 4.2175 segundos para el AG ya con el algoritmo de decodificación como parte de éste, mientras que para el algoritmo por búsqueda exhaustiva tenemos que la reconstrucción de la imagen tarda 178.4447 segundos; con ello cubrimos uno de los objetivos que resulta sumamente importante para la resolución del problema inverso de la codificación fractal; que es disminuir el tiempo de codificación.

Es importante mencionar que aunque el tiempo de búsqueda por medio de AG sin el algoritmo de decodificación resultó menor, el tiempo final ya con la reconstrucción de la imagen se compensa debido a que el mapa de afinidad emitido por este AG entra a un proceso independiente de reconstrucción de la imagen, el cual tarda entre entre 0.5 segundos a 20 segundos aproximadamente, es por ello que dejar el algoritmo de decodificación como parte del AG simplifica y facilita el proceso de reconstrucción.

RMSE CFI	RMSE AG_1	tiempo CFI (s)	tiempo AG_1 (s)	tiempo AG_2(s)
7.343	14.336	178.4457	4.2175	4.0021

Tabla 4.9: Comparativa de resultados en tiempo medido en segundos y medida de distorsión RMSE para Lenna 256 × 256px.

◦ **Comparativa visual de imágenes**

En la Figura 4.18 se puede observar una comparativa visual entre la imagen original, la imagen reconstruida para CFI por búsqueda exhaustiva y por AG respectivamente para una imagen de tamaño 512 × 512 px con *bd* de 4 × 4 px y 32 iteraciones en el SFI.

(a) Imagen original de tamaño 512 × 512 px

(b) Imagen reconstruida a partir de CFI por búsqueda exhaustiva, RMSE: 2.823

(c) Imagen reconstruida a partir de CFI por AG, RMSE: 4.336

Figura 4.18: Imagen original en comparativa con reconstruidas a partir de Búsqueda Exhaustiva y AG.

Para la imagen de Lenna de 512 × 512 px podemos observar en la imagen de la Figura 4.19 la diferencia de píxeles entre la imagen original y la reconstruida, es decir, los píxeles que no se pintaron en la imagen reconstruida con 32 iteraciones y bloques de 4 × 4 px, por lo que podemos asegurar que el AG aplicado a CFI con bloques traslapados ha dado aceptables resultados.

Figura 4.19: Píxeles no pintados en la imagen reconstruida de tamaño 512 × 512 px con bd de 4 × 4 px.

Así mismo, hemos realizado un acercamiento focal de la imagen reconstruida para observar a detalle las variaciones con respecto a la imagen original y hemos notado que estas diferencias son aceptables como se muestra en la Figura 4.20.

(a) Porción de la Imagen original Im_O.

(b) Porción de la imagen reconstruida Im_R.

Figura 4.20: Comparativa visual de una parte de la imagen original y la imagen reconstruida.

◦ Análisis por medio de histogramas

Numéricamente realizamos la comparación de histogramas por 3 métodos: **Correlación**, **Intersección** y la distancia **Bhattacharyya** [46]. En la Tabla 4.10 y 4.11 se muestran los valores

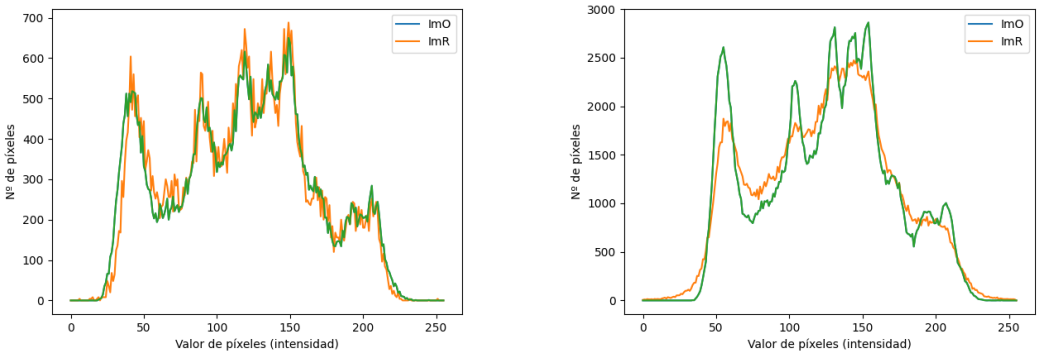
de estas métricas para una imagen de tamaño 256 × 256 px y 512 × 512 px respectivamente a partir de la reconstrucción por la combinación 1, 6 y 12. Nos encontramos que en efecto, la mejor combinación es la número 6, pues hay una mejor correlación e intersección entre las imágenes. Por último en la Figura 4.21 se observa en color verde el histograma de la imagen original y en color naranja el histograma de la imagen reconstruida del mejor individuo por AG de 256 × 256 px y 512 × 512 px respectivamente.

combinación	correlación	intersección [px]	Bhattacharyya
1	0.97034	91.48933	0.07225
6	0.97899	91.81233	0.07225
12	0.91604	95.87274	0.12538

Tabla 4.10: Comparativa entre el histograma de la imagen original y la reconstruida para 256 × 256 px.

combinación	correlación	intersección [px]	Bhattacharyya
1	0.96478	88.81288	0.09608
6	0.965812	88.81279	0.09530
12	0.96582	88.81278	0.09533

Tabla 4.11: Comparativa entre el histograma de la imagen original y la reconstruida con AG para 512 × 512 px.



(a) Histograma de Im_R Lenna de 256 × 256px. (b) Histograma de Im_R Lenna de 512 × 512px.

Figura 4.21: Histogramas de imágenes reconstruidas con AG en comparación con la imagen original de 256 × 256 y 512 × 512 px.

Empleamos la medida de distorsión de señal a ruido pico, PSNR (Peak Signal-to-Noise Ratio), como otro criterio de comparación entre imágenes, ejemplificado en la Sección2.4 para una imagen de 512×512 px por lo que en la Tabla 4.12 observamos que en la primer columna, PSNR 1, el mejor valor de la función objetivo al aplicar el AG con decodificación como parte de éste y en la segunda columna, PSNR2, el mejor valor encontrado por el autor Óscar Alvarado Nava documentado en su tesis de maestría en [47].

Así mismo en la tercer columna, tiempo AG (s), se muestra el tiempo, medido en segundos, empleado por el AG de la presente tesis, la cuarta columna muestra el tiempo empleado por nuestra implementación de búsqueda exhaustiva y la última columna el tiempo empleado por el autor Óscar Alvarado Nava documento en [47].

PSNR 1 [dB]	PSNR 2 [dB]	tiempo AG_1 (s)	tiempo CFI 1 (s)	tiempo CFI 2(s)
24.1527	81.4820	17.254	2473	7740

Tabla 4.12: Resultados de PSNR en la ejecución del AG en el presente trabajo y la CFI documentado por [47] para la imagen de Lenna de tamaño 512×512px.

A continuación probamos imágenes de diferentes tamaño para ver el comportamiento del algoritmo.

Tamaño (px)	RMSE AG_1	tiempo AG_1 (s)	tiempo CFI (s)
1024×1024	19.253	18.026	no computado

(a) Imagen original de tamaño 1024×1024 px

(b) Imagen reconstruida por AG con bd de 4×4 y 32 iteraciones

Tamaño (px)	RMSE AG_1	tiempo AG_1 (s)	tiempo CFI (s)
512×512	0.395	12.874	3088.165

(a) Imagen original de tamaño 512×512 px

(b) Imagen reconstruida por AG con bd de 4×4 y 32 iteraciones

Tamaño (px)	RMSE AG_1	tiempo AG_1 (s)	tiempo CFI (s)
128×128	33.45	1.43	6.71

(a) Imagen original de tamaño 128×128 px.

(b) Imagen reconstruida por AG con bd de 4×4 y 32 iteraciones.

Podemos observar que el algoritmo implementado para hacer frente al problema inverso de SFI aplicado a imágenes digitales está siendo favorable en el sentido de hacer una búsqueda eficiente de los códigos fractales que permitan la reconstrucción de la imagen original. Esta reconstrucción se acepta debido a que muestra una cercanía suficiente para asegurar que la imagen reconstruida se asemeja a la imagen original.

4

Capítulo 5

Conclusiones y Trabajo Futuro

5.1. Conclusiones

La codificación fractal de imágenes es una técnica de compresión con pérdidas en donde se representa una imagen digital como un conjunto de códigos fractales, con la ventaja de reducir el espacio que éstas ocupan. Es más viable almacenar los códigos fractales que representan a una imagen, que la imagen digital en sí misma.

En el Capítulo 3 se muestra la posibilidad de adaptar la CFI para cualquier imagen como base. La complejidad aumenta cuando se deben calibrar los parámetros de iteraciones, transformaciones afines, resolución de la imagen y color de la misma. Siguiendo el algoritmo original con segmentación por bloques, los tiempos de ejecución aumentan significativamente cuando esta cantidad de bloques aumenta, debido a que se hace una búsqueda serial con todas las combinaciones posibles en el espacio de búsqueda, es por ello que solo se abarcaron imágenes cuadradas de hasta 512×512 px.

Ante ello se abre la puerta para aplicar alguna otra estrategia que optimice este tiempo ó la calidad de la reconstrucción de la imagen, por lo cual en el Capítulo 4 se muestra una de las formas de adaptación e implementación del algoritmo genético a la CFI en el que fue posible comprobar que los tiempos de codificación disminuyen bastante, dados en el Capítulo 4 donde se redujo significativamente el tiempo de codificación que ocupa el algoritmo CFI tradicional, por medio del AG, aunque éste ocupe decodificación como parte de su algoritmo. No obstante es un algoritmo que necesita ser implementado y calibrado cuidadosamente

debido a que se puede presentar convergencia prematura, aún cuando se aumentan las generaciones y probabilidades de selección.

En lo particular para este trabajo la forma de codificar a la población inicial es crucial puesto que es la base para un buen funcionamiento del algoritmo genético. Es importante mencionar que los bloques rango traslapados se obtienen de la imagen original escalada por la mitad, por lo que optar por extraer de cada bloque dominio las esquinas y el centro resultó favorable, tomando en cuenta que estos bloques son del mismo tamaño que los bloques rango traslapados, pues para no estar escalando cada bd por la mitad y emparejarlo con cada br, es viable escalar la imagen original y de ahí tomar los br.

Finalmente, otro aspecto importante es la forma de cruzar y mutar a la población, es por ello que utilizar búsqueda exhaustiva con el algoritmo propuesto por Jacquin resulto favorable pues éste método ya probado aseguraba dar soluciones factibles.

5.2. Trabajo a futuro

El problema de cómo implementar una aplicación de forma eficiente en un sistema de cómputo, es aún un problema abierto. Cualquier aplicación se podría implementar cien por ciento en software desde la simulación hasta la práctica, si embargo, esto está limitado a los recursos computacionales con los que se cuenten. Una solución que podría ser favorable para resolver el problema inverso de la CFI es paralelizar el algoritmo de búsqueda y observar si es que los tiempos de codificación disminuyen más que lo reportado por AG. De ser factible, seguramente la codificación, así como la decodificación, se harían más rápidos y se abriría el espacio de búsqueda para más transformaciones afines, incluso con imágenes a color en el espacio RGB y no solo se queden en escala de grises.

Como se ha mencionado, el problema de codificación fractal a imágenes tiene diversos niveles de complejidad y cada uno con cierta profundidad. Por ejemplo, el espacio de búsqueda para el cambio de luminancia s, solo hemos tomado las imágenes que se encuentran en escala de grises con 8 bits entre [0, 255] sin embargo aún está latente el problema para imágenes a color tomando en cuenta que hay tres variables más que calibrar.

Surge una nueva investigación cuando no solo se trabajan imágenes cuadradas sino también rectangulares y más aún, si se trabaja por segmentación, y no solo de forma cuadrada sino triangular o rectangular.

En general, el mayor trabajo para una propuesta de diseño o implementación y aplicación de un algún algoritmo que resuelva este problema, es encontrar un balance óptimo en el particionamiento, transformaciones e iteraciones que requiere la transformada fractal, así como también los tiempos de codificación y decodificación.

Finalmente agrego que, si bien la flexibilidad que ofrecen las metaheurísticas las han vuelto una opción recurrente para resolver problemas de optimización de alta complejidad, puede evitarse su uso y que no sea un sinónimo de utilizarlas en cualquier problema de optimización, pues deben reservarse precisamente para problemas complejos, debido a que algunos de los problemas se pueden resolver eficientemente usando algoritmos exactos. No obstante, para este caso sí fue necesario aplicarlo debido a la complejidad que presenta el problema abordado en este trabajo de investigación, pues se ha indagado en la literatura para encontrar algún método exacto que lo resuelva sin embargo no ha habido registro alguno.

5

Apéndice A

Archivos de configuración A

En el archivo `cfi.c` se encuentra el algoritmo principal de codificación para CFI

```
#include<stdlib.h>
#include<time.h>
#include"bloques.h"
#include"transformaciones.h"
#include"tga.h"
int main(int argc, char *argv[])
{
        int i,j,k;
        int n = atoi(argv[1]); //resolucion de la imagen
        int tamR = atoi(argv[2]); //tamano de BR
        int tamD = atoi(argv[3]);  //tamano de BD
        int numR = ((n/2)-tamR+1); //numero de BR de la imagen
        int numD = n/tamD * n/tamD; //numero de BD de la imagen
        int numr = numR*numR;
        FILE *archivo;
        datos *trans;
        long aux, dist_minima, *dist;
        short prom, *lumProm;
        bloque *M = creaBloque(1,n,8); //imagen original
        bloque *Me = creaBloque(1,n/2,13); //imagen escalada 1/2
        bloque *bloqueD = creaBloque(numD,tamD,10); //bloques
           Dominio
        bloque *bloqueR = creaBloque(numr,tamR,9); //bloques
```

```
            Rango
23          bloque *bloquesRT= creaBloque(numr,tamR,9); //bloques
               Rango para T
24          bloque *temp = creaBloque(numr,tamR,9); /bloques Rango
               temporales
25          bloque *temp2 = creaBloque(numr,tamR,9);
26          dist =(long *)calloc(numr,sizeof(long)); //distancia
27          trans=(datos *)calloc(numr,sizeof(datos));  //
               transformacion
28          lumProm=(short *)calloc(numr,sizeof(short));  //
               luminancia promedio
29
30          imagenOriginal(M,argv[4]); //cargar imagen original
31          BD(bloqueD,M,numD); //BD de la imagen original
32          escala(Me,M); //se escala la imagen original
33          BR(bloqueR,Me,numr); //NR de la imagen original
34
35          archivo = fopen("mapas.fic", "w+");
36          for(i=0; i<numD; i++) //cada BD
37          {
38                  for(j=0; j<numr; j++) //cada BR
39                  {
40                          copia(&bloquesRT[j],&bloqueR[j],9);
41                          prom = (short)bloqueD[i].media - (short)
                               bloqueR[j].media;
42                          cambioLuminancia(&bloquesRT[j],prom);
43                          dist_minima = 1024*1024*1024;
44                          for(k=0; k<8; k++) // 8 transformaciones
45                          {
46                                  copia(&temp2[j],&bloquesRT[j],9);
47                                  transformacion(&temp2[j],&temp[j
                                       ],k);
48                                  aux = distancia(&bloqueD[i],&
                                       temp2[j]);
49
50                                  if(aux < dist_minima)
```

```
51                                      {
52                                              dist_minima = aux;
53                                              dist[j] = dist_minima;
54                                              lumProm[j]= prom;
55                                              trans[j]=k;
56                                      }
57                              }
58                      }
59                      mapa(archivo,bloqueR,numr,dist,trans,lumProm);
60              }
61             fclose(archivo);
62           libera_mem(M,Me,bloqueR,bloquesRT,bloqueD);
63           return 0;
64 }
```

Se compila usando el archivo `Makefile` que tiene la siguiente estructura:

```
1  all:bloq
2
3  bloq:bloques.o transformaciones.o tga.o principal.o
4  gcc $^ -o $@
5
6  bloques.o: bloques.c
7  gcc -c $^ -o $@
8
9  transformaciones.o:transformaciones.c
10 gcc -c $^ -o $@
11
12 tga.o: tga.c
13 gcc -c $^ -o $@
14
15 principal.o: principal.c
16 gcc -c $^ -o $@
17
18 lenna256_4:
19 ./bloq 256 256 4 lenna256.tga
20
21 clean:
```

```
rm *.o bloq B*.tga D*.tga im*.tga mapa_*.fic
```

En el archivo `decodificación.c` se encuentra el algoritmo principal de decodificación para CFI

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#include<math.h>
#include"mapas.h"
#include"tga.h"
#include"transformaciones.h"

int main(int argc, char *argv[])
{
        int i,j,n=0,m=0, *x, *y;
        int iter = atoi(argv[2]);
        datos *trans;
        short *cl, tamI=0, tamB=0, **M;
        double tiempo;
        FILE *mapa;

        mapa = fopen(argv[1], "r");
        fscanf(mapa, "%d", &n);      //filas (mapas)
        fscanf(mapa, "%d", &m);      //columnas
        fscanf(mapa, "%hu", &tamI);  //tamano de la imagen a
           reconstruir
        M=creaMatriz(n,m);           //matriz para mapas

        for(i=0; i < n; i++)
         for(j = 0; j < m; j++)
                fscanf(mapa, "%hd", &M[i][j]);

        fclose(mapa);
        tamB = tamI/sqrt(n);         //tamano de Bloque

        bloque *imT = creaBloque(1,tamI,1);   //crear imagen base
```

```
32          bloque *imTe = creaBloque(1,tamI/2,2); //imagen a
               tranformar, escala
33          bloque *lienzo = creaBloque(1,tamI,3);
34          bloque *bloquesD = creaBloque(n,tamB,1); //bloques
               continuos en lienzo
35          bloque *temp = creaBloque(n,tamB,1);
36          imagenOriginal(imT,argv[3]);  //cargar imagen base
37
38          trans=(datos *)calloc(n,sizeof(datos)); //transformacion
39          x=(int *)calloc(n,sizeof(int));
40          y=(int *)calloc(n,sizeof(int));
41          cl=(short *)calloc(n,sizeof(short));    //luminancia
               promedio
42
43          for(i=0; i < n; i++)
44          {
45             x[i] = M[i][0];
46                  y[i] = M[i][1];
47                  trans[i] = (datos)M[i][2];
48                  cl[i] = (short)M[i][3];
49          }
50          for(i=0; i<iter; i++)
51          {
52                  for(j=0; j<n; j++)
53                  {
54                          escala(imTe,imT);
55                          extraer(&bloquesD[j],imTe,x[j],y[j]);
56                          cambioLuminancia(&bloquesD[j],cl[j]);
57                          transformacion(&bloquesD[j],&temp[j],
                               trans[j]);
58                  }
59                  Lienzo(lienzo,bloquesD);
60                  copia(imT,lienzo);
61          }
62          pintaImagen(lienzo,tamI,tamB,iter);
63          libera_mem(lienzo,imT,imTe,bloquesD,temp,x,y);
```

```
64          tiempo = (double)(fin - inicio)/CLOCKS_PER_SEC;
65          return 0;
66 }
```

Se compila usando el archivo `Makefile` que tiene la siguiente estructura:

```
1  all:deco
2
3  deco:mapas.o tga.o transformaciones.o principal.o
4  gcc $^ -o $@
5
6  mapas.o: mapas.c
7  gcc -c $^ -o $@
8
9  tga.o: tga.c
10 gcc -c $^ -o $@
11
12 transformaciones.o: transformaciones.c
13 gcc -c $^ -o $@
14
15 principal.o: principal.c
16 gcc -c $^ -o $@
17
18 decodificacion:
19 ./deco mapa256.fic 16 manzana.tga
20
21 clean:
22 rm *.o deco BD*.tga
```

A

Apéndice B

Archivos de configuración B

En el archivo ag.c se encuentra el algoritmo genético para CFI

```
#include<stdlib.h>
#include<time.h>
#include<math.h>
#include"bloques.h"
#include"genetico.h"
#include"tga.h"
#include"transformaciones.h"

int main(int argc, char *argv[])
{
        int n = atoi(argv[2]); //resolucion de la imagen
        int tamB = atoi(argv[3]);   //tamano de bloques
        int nInd = atoi(argv[4]);   //indivduos
        int gen = atoi(argv[5]);
        int iter = atoi(argv[6]);       //n de iteraciones
        float pm = atof(argv[7]);   //prob de mutacion
        int numR = ((n/2)-tamB+1)*((n/2)-tamB+1);  //n de sub-BR
           de la imagen
        int numD = n/tamB * n/tamB;                          //n de
            sub-BD de la imagen
        int i,j,k,g,nB,var,*sel;
        FILE *archivo, *tiempo;
        double ts, ts1, dista;
```

```
22          char filename[30];
23
24          bloque *M = creaBloque(1,n,1);  //imagen original
25          bloque *Mtemp = creaBloque(1,n,1); //imagen original
26          bloque *Me = creaBloque(1,n/2,1);
27          bloque *lienzo = creaBloque(nInd,n,4);
28          bloque *Bdom = creaBloque(numD,tamB,2); //crear BD
29          bloque *Brang = creaBloque(numR,tamB,3);  //crear BR
30
31          poblacion *pobla = creaPoblacion(nInd,numD,numR,tamB);
32          poblacion *poblaTemp = creaPoblacion(nInd,numD,numR,tamB)
               ;
33          poblacion *poblaR = creaPoblacion(nInd,numD,numR,tamB);
34          ordenaPobla *pobSort = creaInformacion(nInd,numD);
35          ordenaPobla *pobSortM = creaInformacion(nInd,numD);
36          ordenaPobla *pobSortR = creaInformacion(nInd,numD);
37
38          nB=5;
39          sel = (int *)calloc(nB,sizeof(int));
40          srand(time(NULL));
41          imagenOriginal(M,argv[1]); //cargar im0
42          //Poblacion inicial
43          for(i=0; i<nB; i++)
44          {
45                  copia(Mtemp,M,1,n);
46                  for(j=0; j<iter; j++)
47                  {
48                          BD(Bdom,Mtemp,numD);
49                          escala(Me,Mtemp);
50                          BR(Brang,Me,numR);
51                          extraer(&pobla[i],Bdom,Brang,nInd,numD,
                               numR,i);
52                          reconstruccion(&lienzo[i],&pobla[i],Bdom,
                               numR,tamB,numD);
53                          copia(Mtemp,&lienzo[i],1,n);
54                  }
```

```
55          }
56          for(i=nB; i<nInd; i++)
57          {
58                  copia(Mtemp,M,1,n);
59                  for(j=0; j<iter; j++)
60                  {
61                          BD(Bdom,Mtemp,numD);
62                          escala(Me,Mtemp);
63                          BR(Brang,Me,numR);
64
65                          extraer(&pobla[i],Bdom,Brang,nInd,numD,
                                numR,i);
66                          if(i<10)
67                          {
68                                  aleatorio(sel,numD,nB);
69                                  busquedaEx(&pobla[i],sel,Bdom,
                                        Brang,numD,numR,nB);
70                          }
71                          reconstruccion(&lienzo[i],&pobla[i],Bdom,
                                numR,tamB,numD);
72                          copia(Mtemp,&lienzo[i],1,n);
73                  }
74          }
75          copiaPoblacion(poblaTemp,pobla,nInd,numD,tamB);
76          tiempo = fopen("tiempos123.txt", "w+");
77
78          //-----------------Algoritmo -------------------
79          RMSE(pobSort,M,lienzo,nInd);
80          clock_t inicio = clock();
81          for(g=0; g<gen; g++)
82          {
83                  clock_t inicioG = clock();
84                  probSeleccion(pobSort,nInd);
85                  ordena(pobSort,nInd);
86                  probAcumulada(pobSort,nInd);
87                  probabilidad(pobSort,nInd);
```

```
88
89                  cruzaPunto(poblaTemp,pobSort,nInd,numD,numR,tamB)
                        ;
90                  BD(Bdom,M,numD);
91                  mutacion(poblaTemp,Bdom,Brang,nInd,numD,numR,pm);
92                  for(i=0; i<nInd; i++)
93                      reconstruccion(&lienzo[i],&poblaTemp[i],Bdom,
                            numR,tamB,numD);
94
95                  RMSE(pobSortM,M,lienzo,nInd);
96                  mejorFitness(pobSortR,pobSort,pobSortM,nInd,numD)
                        ;
97                  nuevaPoblacion(poblaR,pobla,poblaTemp,pobSortR,
                        nInd,numD,tamB);
98                  for(i=0; i<nInd; i++)
99                      reconstruccion(&lienzo[i],&poblaR[i],Bdom,numR
                            ,tamB,numD);
100
101                 clock_t finG = clock();
102                 copiapobSort(pobSort,pobSortR,nInd);
103                 copiaPoblacion(pobla,poblaR,nInd,numD,tamB);
104                 copiaPoblacion(poblaTemp,poblaR,nInd,numD,tamB);
105                 sprintf(filename,"dist%d.txt",g);
106                 archivo = fopen(filename, "w+");
107
108                 dista = 1000*1000;
109                 for(i=0; i<nInd; i++)
110                 {
111                         fprintf(archivo,"%.8lf\n",pobSort->dist[i
                                ]);
112                         if(pobSort->dist[i] < dista)
113                         {
114                                 dista = pobSort->dist[i];
115                                 var=i;
116                         }
117                 }
```

```
118                     fclose(archivo);
119                     pintaImagen(&lienzo[var],n,var,g);
120                     ts = (double)(finG - inicioG)/CLOCKS_PER_SEC;
121                     fprintf(tiempo,"%f %.8lf\n",ts,pobSort->dist[var
                            ]);
122             }
123             clock_t fin = clock();
124             ts1 = (double)(fin - inicio)/CLOCKS_PER_SEC;
125             fprintf(tiempo,"\n%f\n",ts1);
126             fclose(tiempo);
127             libera_mem(M,Me,lienzo,Mtemp,Bdom,Brang,nInd);
128             return 0;
129 }
```

Se compila usando el archivo `Makefile` que tiene la siguiente estructura:

```
1  all: gen
2
3  gen:bloques.o genetico.o tga.o transformaciones.o principal.o
4  gcc $^ -o $@
5
6  bloques.o: bloques.c
7  gcc -c $^ -o $@
8
9  genetico.o: genetico.c
10 gcc -c $^ -o $@
11
12 tga.o: tga.c
13 gcc -c $^ -o $@
14
15 transformaciones.o: transformaciones.c
16 gcc -c $^ -o $@
17
18 principal.o: principal.c
19 gcc -c $^ -o $@
20
21 ag:
22 ./gen lenna512.tga 512 4 20 20 16 0.4
23
24 clean:
25 rm *.o gen  BD*.tga Im0*.tga Br*.tga ImR*.tga dist*.txt tiem*.txt
```

B

Bibliografía

[1] B. B. Mandelbrot, *The Fractal Geomtry of Nature*. New York: W. H. Freeman and Company, (1983).

[2] L. Madrid-Herrera and et. al., "Análisis de métodos de medición de complejidad de imagen," *Revista electrónica de Computación, Informática, Biomédica y Electrónica*, vol. 2, pp. 17–46, (2018).

[3] *Complexity Perception of Texture Images*. Springer, 2015.

[4] M. Al-khassaweneh and et. al. Frei-chen bases based lossy digital image compression technique.

[5] M. Singh and et. al, "Various image compression techniques: Lossy and lossless," *International Journal of Computer Applications*, vol. 142, no. 6, (2016).

[6] J. Vrindavanam and et. al, "A survey of image compression methods," *International Journal of Computer Applications*, (2012).

[7] (2021, Jul). [Online]. Available: https://www.geeksforgeeks.org/shannon-fano-algorithm-for-data-compression/

[8] C. Y. D. Nora La Serna, Mg. Luzmila Pro Concepción, "Compresión de imágenes: Fundamentos, técnicas y formatos," *Revista de Ingeniería de Sistemas e Informática*, vol. 6, no. 1, (2009).

[9] "A method for the construction of minimum-redundancy codes," *Proceedings of the IRE. IEE*, vol. 40, pp. 1098–1101, (1952).

[10] "Compresión de imágenes codificación de huffman," *Revista de educación matemática*, vol. 32, pp. 25–36, (2017).

[11] "Compresión y restauración de imágenes por técnicas no lineales," Master's thesis, Universidad de Valencia, Valencia, (2011).

[12] "El estándar de compresión de imagen jpeg," *Universidad Politécnica de Valencia*, pp. 1–7.

[13] F. B. Michael, *Fractals Everywhere*, 2nd ed. New York: Academic Press Professional, (2012).

[14] (2012, Mayo) Algoritmos genéticos. [Online]. Available: http://bibing.us.es/proyectos/abreproy/5073/fichero/Cap%C3%ADtulo+3-+Algoritmo+genético%252FCap%C3%ADtulo+3-+Algoritmo+genético.pdf

[15] C. A, P. Melville, and V. Phan, "Fractal compression," (2000).

[16] M. F. Barnsley, *Fractal Image Compression*. Mathematics, Georgia Teach, (1993).

[17] Y. Fisher, *Fractal Image Compression*. Springer-Verlag, (1995), ch. 3, Fractal Image Compression with Quadtrees, pp. 55–77.

[18] J. E. Hutchinson, "Fractals and self similarity," *Indiana University Mathematics Journal*, vol. 30, no. 5, pp. 713–747, (1981).

[19] A. Jacquin, "Image coding based on a fractal theory of iterated contractive image transformations," *IEEE Transactions on Image Processing*, vol. 1, pp. 18 – 30, (1992).

[20] S. J. Vázquez and F. de María Correa, "Sistema de funciones iteradas por partes," *Morfismos*, vol. 16, no. 1, pp. 9–27, (2012).

[21] N. Slawomir, *Iterated Function Systems for Real-Time Image Synthesis*, 1st ed. Poland: Springer, (2007).

[22] S. Daniel, *The Nature of Code: Simulating Natural Systems with Processing*. Magic Book Project, (2012).

[23] E. Grudner., "Geometría fractal y compresión de imágenes cf-sfip," *Revistas Bolivianas, Pizarra Matemática*, vol. 10, no. 16, 2014.

[24] D. P. Huttenlocher, G. A. Klanderman, and W. J. Rucklidge, "Comparing images using the hausdorff distance," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 15, no. 9, pp. 850–863, (1993).

[25] E. A. A. Sarmiento. Sistema de funciones iteradas y los fractales. [Online]. Available: http://www.konradlorenz.edu.co/images/stories/suma_digital_matematicas/SFI%20y%20los%20Fractales.pdf

[26] e. a. Carlos Artemio Coello Coello, *Computación Evolutiva*. Academia Mexicana de Computación, (2019).

[27] D. E. Goldberg. and J. H. Holland., "Genetic algorithms and machine learning," *Machine Learning*, (1988).

[28] D. E. Goldberg., "Genetic algorithms in search, optimización machine & learning," *Addison-Wesley Publishing Company, Inc.*, (1989).

[29] E. D. Eric Bazan, Petr Dokládal, "Quantitative analysis of similarity measures of distributions," *BMVC 2019*, Sep (2019).

[30] M. Name, J. Lima, F. Boff, D. Filho, and R. Falate, "Histogram comparison using intersection metric applied to digital images analysis," *Iberoamerican Journal of Applied Computing*, vol. 2, pp. 11–18, 04 (2012).

[31] S. Abenda, S. Demko, and G. Turchetti, *Local moments and inverse problem of fractal measures*, (1992), ch. 8, pp. 739–750.

[32] W. Niclas, "An automatization of barnsley's algorithm for the inverse problem of iterated fuction systems," *IEEE Transactions on Image Processing*, vol. 12, no. 11, pp. 1388–1397, (2003).

[33] R. Rinaldo and A. Zakhor, "Inverse and approximation problem for two-dimensional fractal sets," *IEEE Transactions on Image Processing*, vol. 3, no. 6, pp. 802–820, (1994).

[34] A. Arneodo, E. Bacry, and J.-F. Muzy, "Solving the inverse fractal problem from wavelet analysis," *EPL - Europhysics Letters, European Physical Society/EDP Sciences/Società Italiana di Fisica/IOP Publishing*, vol. 25, no. 7, pp. 479–484, (1994).

[35] D. L. Torre, E. Maki, F. Mendivil, and E. R. Vrscay, "Iterated function systems with place-dependent probabilities and the inverse problem of measure approximation using moments," *Fractals*, (2018).

[36] S. S. Al-Bundi, N. M. G. Al-Saidi, and N. J. Al-Jawari, "Crowding optimization method to improve fractal image compressions based iterated function systems," *International*

Journal of Advanced Computer Science and Applications, vol. 7, no. 7, pp. 392 – 401, (2016).

[37] G.-H. Ong, C.-M. Chew, and Y. Cao., "A simple partitioning aproach to fractal image compression," *SAC 01: Proceedings of the 2001 ACM symposium on Applied computing*, pp. 301–305, (2001).

[38] B. Wohlberg and G. de Jager, "A review of the fractal image coding literature," *IEEE Transactions on Image Processing*, vol. 8, no. 12, pp. 1716–1729, (1999).

[39] M. Kramm, "Image cluster compression using partitioned iterated function systems and efficient inter-image similarity features," *IEEE Xplore*, pp. 989–996, (2008).

[40] S. D and R. Matthias, "Evolutionary fractal image compression," *IEEE Xplore*, pp. 129–132, (2002).

[41] H. You, N. Shiraki, and R. Tokunag, "Image segmentation via fractal transformation: Improving idas image segmentation scheme," *Electronics and Communications in Japan*, vol. J82-A, pp. 1793–1800, (2002).

[42] Davione, Svensson, and Chassery, "A mixed triangular and quadrilateral partition for fractal image compression," *IEEE Xplore*, vol. 3, pp. 284–287, (2002).

[43] A. E. Jacquin, "A fractal theory of iterated markov operators with applications to digital image coding," Ph.D. dissertation, Georgia Institute of Technology, 1989.

[44] H. Hartenstein, R. Matthias, D. Sawpe, and R. V. Edward, "On the inverse problem of fractal compression," *Ergodic theory, analysis and efficient simulation of dynamical Systems*, pp. 617 – 647, (2001).

[45] Z. Q. and C. R., "A comparison of histogram distance metrics for content-based image retrieval," *Proceedings of SPIE - The International Society for Optical Engineering*, (2014).

[46] Naik., Patil., and Joshi., "A scale adaptive tracker using hybrid color histogram matching scheme." *In Emerging Trends in Engineering and Technology (ICETET)*, (2009).

[47] Ó. A. Nava, "Implementación en fpgas de algoritmos de compresión-descompresión para dispositivos móviles," Master's thesis, Centro de Investigación y de Estudios Avanzados del IPN Zacatenco, (2007).