CENTRO DE INVESTIGACIONES EN ÓPTICA, A.C.

AJUSTE DE FUNCIONES MEDIANTE METAHEURÍSTICAS APLICADO A LA DETECCIÓN DE FASE EN ARREGLOS DE METROLOGÍA ÓPTICA

Como Requisito para obtener el grado de:

Maestro en Optomecatrónica (Versión Definitiva)

Estudiante:
Lic. Iosvani Moré Quintero

Asesor:
Dr. Francisco Javier Cuevas de la Rosa

Julio de 2017
León, Guanajuato, México

Dedicado especialmente a mis abuelos y primeros maestros.
A mis padres, mi esposa e hija.

Agradecimientos

Al Comandante en Jefe Fidel Castro y a todos los que dieron su vida por esta Revolución Cubana porque sin ella no hubiese sido posible convertirme en un profesional de las ciencias.

Un agradecimiento especial al Dr. Francisco Javier Cuevas de la Rosa que como artífice de este trabajo me ha apoyado, guiado y corregido en mi labor científica.

Hago extensivo mi agradecimiento al Consejo Nacional de Ciencia y Tecnología, (CONACYT), y al Centro de Investigaciones en Óptica, (CIO) por haberme dado la oportunidad de estudiar la Maestría en Optomecatrónica.

A todos mis profesores que cada uno con sus particularidades han sido fuente de luz, guía e inspiración para llegar a este momento y han incidido directamente en mi formación profesional.

A mi familia, amigos y sobre todo aquellos que me brindaron su apoyo incondicional al llegar a este país y depositaron toda su esperanza y confianza en mi sueño de convertirme en un profesional de las ciencias.

A mis padres, mis suegros y mi otra familia cubana quienes supieron apoyarme incondicionalmente en todo momento y hacer que mi sueño fuese el suyo.

Especialmente a mis esposa e hija, por ser tan especiales que no tengo palabras para agradecerles y son mi motivo más fuerte para estar titulándome hoy aquí.

Resumen

El presente trabajo describe la demodulación de interferogramas como uno de los retos más importantes dentro de la industria. Debido a que este problema es extremadamente difícil aún no se conoce un método capaz de obtener mediciones precisas, sobre todo en eventos transientes.

Por lo anterior, este informe propone un método mediante el cual se puede obtener una aproximación polinomial con ajuste paramétrico del mapa de fase del objeto, utilizando técnicas de Inteligencia Artificial. En términos matemáticos se realiza una aproximación polinomial, debido a que los polinomios son capaces de generalizar una posible solución, así como un ajuste funcional en todo el mapa de fase, apoyándose en las derivadas de la fase, las cuales ayudan a obtener soluciones más aproximadas a las curvas del interferograma a demodular, ya que promedian una vecindad en el eje x y y de la función de costo.

Se proponen tres metaheurísticas, las cuales fueron modeladas como un problema de minimización para poder aproximar el término de fase del objeto en estudio. El primer algoritmo estudiado es el conocido como Recocido Simulado (RS) formulado por Kirkpatrick, Gelatt y Vecchi en 1983. Esta metaheurística simula el proceso de recocido que se realiza en la industria para obtener materiales más resistentes o más cristalinos.

La segunda metaheurística utilizada en este proyecto se refiere al algoritmo de Optimización basado en Mallas Variables (VMO). El modelo VMO utiliza un proceso de expansión formado por funciones de generación de nuevas soluciones que exploran zonas promisorias del espacio de búsqueda. Emplea, además, un proceso de contracción que se encarga de seleccionar los mejores representantes de cada zona explorada.

La tercera y última técnica utilizada se refiere a los Algoritmos Genéticos, el cual básicamente consiste en generar aleatoriamente un conjunto inicial de posibles soluciones, los cuales serán seleccionados basándose en una función de probabilidad con el objetivo de cruzar su información y crear nuevas soluciones. Este método utiliza una estrategia de mutación para generar una solución diferente al resto de las soluciones.

Nuestra propuesta es aprovechar las fortalezas de estas metaheurísticas y adaptarlas como un problema de demodulación de interferogramas. Se plantean diferentes variantes de estos métodos, así como un conjunto de disímiles configuraciones de los parámetros de entrada utilizados por los algoritmos, con la finalidad de discriminar aquellas variantes que no convergen a una solución factible y optimizar el desempeño de las mejores configuraciones.

Se realizó un proceso de comparación entre las tres metaheurísticas para evaluar la capacidad de búsqueda de cada una de ellas de forma independiente con diferentes funciones de prueba.

Después de los experimentos realizados se concluye que, sí es posible aplicar las metaheurística para aproximar el mapa de fase contenido dentro de un interferograma. Una notable ventaja de estos métodos es que no requieren de un conocimiento a fondo del problema y pueden solucionar problemáticas de alta complejidad. Finalmente, se comentan algunas ideas para mejorar la técnica propuesta, también se realizan varias propuestas de posibles líneas de investigación a acometer en el futuro.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Objetivos de la Tesis

El objetivo general del presente trabajo se basa fundamentalmente en el desarrollo de un sistema de visión para la reconstrucción tridimensional de objetos y la detección de fase modulada en un patrón de franjas (abiertas o cerradas) mediante el uso de metaheurísticas.

Para ello se desglosa el proyecto en los siguientes objetivos específicos:

- ✓ Revisar las técnicas implementadas hasta el momento y realizar un análisis comparativo.
- ✓ Desarrollar algoritmos computacionales de procesamiento de imágenes para la reconstrucción tridimensional.
- ✓ Implementar algunos algoritmos evolutivos con la finalidad de identificar la fase en arreglos de Metrología Óptica.
- ✓ Implementar una aplicación computacional capaz de procesar la información obtenida en el arreglo experimental.

1.2. Planteamiento del Problema

La Metrología Óptica (MO) es de suma importancia en la solución de problemas de medición de cantidades físicas en aplicaciones relacionadas a la industria automovilística, la aeronáutica, el análisis no destructivo de materiales compuestos, la inspección de turbinas y detección de desperfectos en elementos mecánicos, entre otras.

Mediante técnicas de MO se pueden realizar mediciones no invasivas de objetos que permiten analizar algunas de sus propiedades y cambios físicos tales como: esfuerzo, desplazamiento, distancia, presión, temperatura, velocidad, forma y dimensiones métricas de objetos y/o regiones de observación.

Como resultado final del proyecto de tesis y como una de las disímeles aplicaciones de la metrología óptica se plantea la utilización de metaheurísticas para la recuperación de la

fase óptica y realizar un análisis comparativo entre diferentes técnicas evolutivas, entre las cuales figura Recocido Simulado, Optimización Basada en Mallas Variables y Algoritmos Genéticos.

1.3. Antecedentes

Durante varios siglos, la observación de la naturaleza ha permitido al ser humano desarrollar teorías acerca del funcionamiento y comportamiento de muchos fenómenos naturales. Unido al desarrollo y progreso de la informática, se incrementó en gran medida al estudio de la naturaleza y, particularmente, de los sistemas biológicos. Al mismo tiempo, las Ciencias de la Computación se han enriquecido con la incorporación de ideas biológicas para conseguir soluciones a muchos de los problemas que se plantean en la actualidad.

Como resultado de estas interacciones entre la computación y la biología, surge así, una disciplina denominada Inteligencia Artificial, la cual abarca otras disciplinas como las Redes Neuronales Artificiales, sistemas difusos, Metaheurísticas, entre otras. Constituyendo esta última un área que recibe mucho interés debido a que se presenta en multitud de problemas de ciencia e ingeniería que, por su propia naturaleza, no pueden ser resueltos mediante un procedimiento de búsqueda exhaustivo [1] [2].

Los Algoritmos Evolutivos (AE) constituyen una técnica general de resolución de problemas de búsqueda y optimización inspirada en la teoría de la evolución de las especies y la selección natural [3]. Estos algoritmos permiten abordar problemas de búsqueda de caminos óptimos, planificación de tareas, optimización de funciones; facilitando, así las tareas de diseño e implementación.

Debido al auge en el desarrollo de procedimientos heurísticos para resolver problemas de optimización, se puede comprobar que estos algoritmos dependen en gran medida del problema en cuestión para el cual se ha diseñado. De manera que no se puede asegurar que existe el método heurístico que sea capaz de obtener los mejores resultados para cualquier problema de optimización, resultado conocido como Teorema *"No Free Lunch"* [4].

Basado en el supuesto anterior [4] y la tendencia de relacionar la Inteligencia Artificial con otras ramas de la ciencia; se modeló el problema de recuperación de fase óptica como un problema de optimización. De esta forma queda planteado la vinculación entre la Metrología Óptica y la Inteligencia Computacional.

1.3.1. Proyección de franjas con un perfil cosenoidal

En MO generalmente se usan interferómetros ópticos para medir una amplia gama de cantidades físicas. Dependiendo de la aplicación, diferentes tipos de interferómetros pueden ser usados, pero su objetivo común es producir un patrón de franjas que es modulado en fase por la cantidad física medida. Los métodos interferométricos nos permiten medir la forma de una superficie óptica con gran exactitud, usando la longitud de onda como una medida

de longitud.

Basado en el principio de superposición de ondas [5], el patrón de franjas puede ser considerado como una fluctuación de una señal sinusoidal en el espacio bidimensional. El análisis de patrones de franjas se refiere a la completa recuperación de las características originales de un patrón de franjas. En este proceso la única cantidad que se puede medir es la intensidad $I(x, y)$, la cual puede ser representada a través de su perfil cosenoidal como:

$$I(x, y) = a(x, y) + b(x, y)cos(w_x x + w_y y + \phi(x, y)) + n(x, y). \tag{1.1}$$

Donde el término $a(x, y)$ representa la iluminación de fondo, $b(x, y)$ el contraste o la modulación de la señal, w_x y w_y la frecuencia portadora en los ejes correspondientes, $\phi(x, y)$ simboliza el término de fase el cual está ligado con la cantidad física que será cuantificada y $n(x, y)$ representa el ruido multiplicativo de alta frecuencia que generalmente se presenta con aspecto granulado en las imágenes [6, 7].

El objetivo fundamental de los algoritmos evolutivos para este tipo de problema es la recuperación de la cantidad física medida a partir de $I(x, y)$, con la dificultad que existen dos funciones desconocidas $a(x, y)$ y $b(x, y)$.

Además, el hecho de que la $I(x, y)$ depende del término de fase, trae consigo que sólo puede ser obtenida en módulo 2π debido al término trigonométrico de la Ecuación (1.1). Otro inconveniente es la paridad de la función coseno $cos(\phi) = cos(-\phi)$, en la cual el signo no puede ser extraído sin algún conocimiento a priori.

Generalmente las funciones de fondo y de contraste contienen las intensidades de campos que interfieren y varias perturbaciones. El término $a(x, y)$ representa la variación en la iluminación y cambio en la reflectividad del objeto, así como el ruido electrónico dependiente del tiempo y de los componentes electrónicos relacionados con la captura de la imagen. Por otra parte, la amplitud de modulación $b(x, y)$ contiene todas las influencias multiplicativas, incluyendo la razón entre los haces de referencia y objeto.

Para una evaluación cuantitativa auxiliada por computadora el patrón de franja o interferograma es normalmente registrado por una cámara CCD y almacenado en una memoria de computadora en un formato digital. La intensidad registrada es digitalizada en un arreglo de resolución de MxN puntos imagen (píxeles) y cuantificado en valores discretos $(0 - 255)$.

El teorema de muestreo o criterio de Nyquist [8] [9], plantea que si conocemos el ancho de banda de la señal, entonces la frecuencia de muestreo f para lograr una reconstrucción casi perfecta de la señal original deberá ser:

$$f_N \geq 2 * WB, \tag{1.2}$$

esto indica la necesidad de más de dos puntos de detección por franja; así mismo, debido al uso de elementos de tamaño finito del detector, a las cargas filtradas a píxeles vecinos y al ruido en el patrón de franja, se debe suministrar al menos 3 a 5 pixeles por periodo de franja para producir una estimación de fase confiable o para sensores de 1 byte se puede incrementar

la sensibilidad si se usan cámaras de 10 o 12 bits. Es válido destacar que la técnica propuesta en este trabajo puede demodular la fase a pesar que el teorema de muestreo no cumpla con las condiciones necesarias. Otra ventaja sobre algunos métodos es la utilización de una sola imagen y sobretodo en eventos transientes o dinámicos.

1.3.2. Clasificación de los Metódos de Demodulación de Fase

El gran avance tecnológico en materia de procesamiento y adquisición de imágenes ha traído consigo el desarrollo de la metrología interferométrica, así como un aumento de los algoritmos de extracción de fase, entre los más conocidos se encuentra el de la transformada de Fourier [10] y el de corrimiento de fase [11]. Estos procedimientos pueden aplicarse tanto en eventos estáticos como transientes. Básicamente estos métodos obtienen el patrón de franjas a partir de un método óptico, lo hacen incidir sobre el objeto en cuestión para finalmente capturar la información y someterla a un análisis.

Las técnicas de medición de fase pueden ser divididas en dos grupos fundamentalmente: aquellos que toman los datos secuencialmente y los que toman los datos simultáneamente [8] [12]. La Figura 1.1 muestra una clasificación de los principales métodos de demodulación de franjas más utilizado en la literatura.

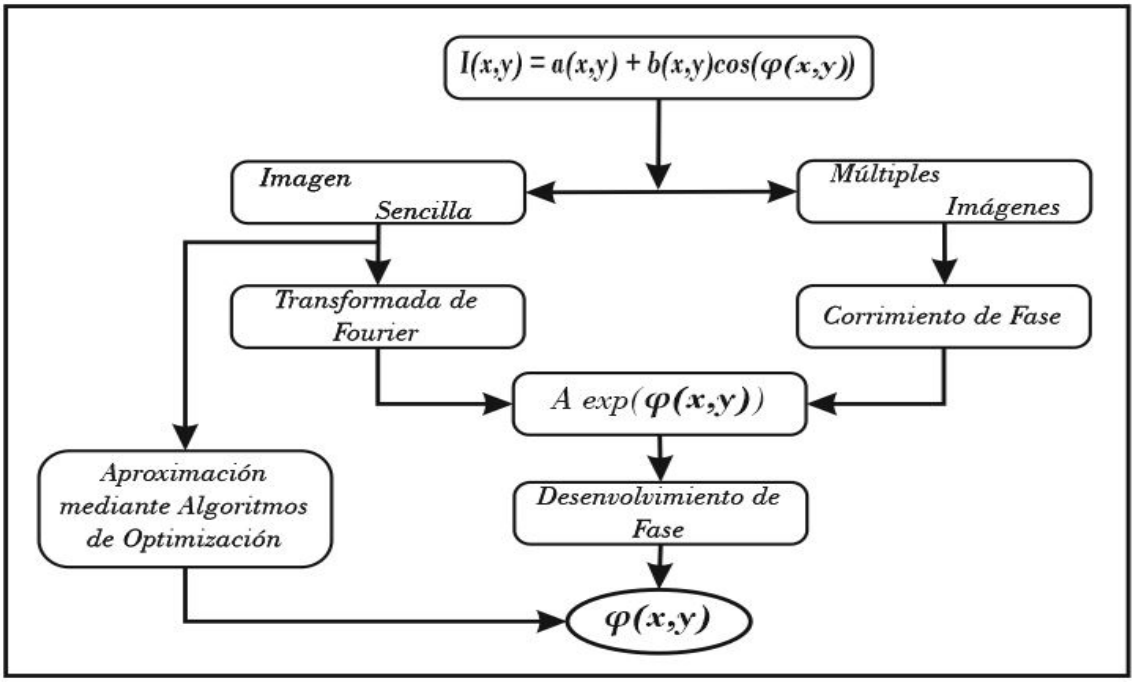


Figura 1.1: Clasificación de los principales métodos de recuperación de fase.

1.3.2.1. Métodos de medición de Fase Temporal.

Interferometría de corrimiento de fase.

El método de *"corrimiento de fase"* o *"phase shifting"* por sus siglas en inglés, es uno de los métodos más utilizados, en el cual intervienen portadores temporales. Consiste en obtener varios valores de intensidad con un incremento conocido en fase [13]. Durante el tiempo de adquisición la fase deberá permanecer estacionaria para evitar la menor cantidad de error en las mediciones [11] [14].

Asumiendo que los valores de irradiancia I_i son medidos a partir de su valor inicial $\phi(x,y)$ e incrementado en una cantidad constante δ_i, tenemos que:

$$I_i(x,y) = a(x,y) + b(x,y)cos(\phi(x,y) + \delta_i) \qquad i \in [1..N] \tag{1.3}$$

donde N indica el número de patrones de franjas que se desea adquirir. Para el número de muestras igual a N, el incremento de fase está dado por la siguiente expresión:

$$\delta_i = \frac{2\pi}{N} \tag{1.4}$$

Básicamente, mientras mayor sea el número de muestras, menor es el nivel de ruido aleatorio, que decrece por un factor $\frac{1}{\sqrt{N}}$, aunque en la práctica se utilizan entre tres y siete patrones de franjas. El número de muestra más empleado para el método de corrimiento de fase es $N = 4$, lo cual trae consigo un desplazamiento en fase de $\delta_i = \frac{\pi}{2}$.

Sustituyendo los valores anteriores en la ecuación 1.3 obtenemos 4 ecuaciones que describen los patrones de intensidad de los interferogramas medidos:

$$I_1(x,y) = a(x,y) + b(x,y)cos(\phi(x,y)) \tag{1.5}$$

$$I_2(x,y) = a(x,y) + b(x,y)cos(\phi(x,y) + \frac{\pi}{2}) \tag{1.6}$$

$$I_3(x,y) = a(x,y) + b(x,y)cos(\phi(x,y) + \pi) \tag{1.7}$$

$$I_4(x,y) = a(x,y) + b(x,y)cos(\phi(x,y) + \frac{3\pi}{2}) \tag{1.8}$$

Aplicando identidades trigonométricas y resolviendo algebraicamente las ecuaciones anteriores para obtener el valor de la fase $(\phi(x,y))$ obtenemos:

$$I_4(x,y) - I_2(x,y) = 2b(x,y)sen(\phi(x,y)) \tag{1.9}$$

$$I_1(x,y) - I_3(x,y) = 2b(x,y)cos(\phi(x,y)) \tag{1.10}$$

Dividiendo las ecuaciones 1.9 y 1.10 y despejando el término de fase llegamos a las siguientes expresiones:

$$\frac{sen(\phi(x,y))}{cos(\phi(x,y))} = \frac{I_2(x,y) - I_2(x,y)}{I_1(x,y) - I_3(x,y)} \tag{1.11}$$

$$\phi(x,y) = tan^{-1}\left[\frac{I_2(x,y) - I_2(x,y)}{I_1(x,y) - I_3(x,y)}\right] \tag{1.12}$$

Este método tiene como ventaja que no se sacrifica la resolución espacial debido a que las operaciones se realizan pixel a pixel y la precisión puede mejorarse al incrementar el número de muestras. Contradictoriamente, debido a la necesidad de adquirir múltiples imágenes este método se vuelve ineficiente para eventos transientes.

1.3.2.2. Métodos de medición de Fase Espacial.

Interferometría de Fourier.

La técnica de la transformada de Fourier se ha aplicado ampliamente en el procesamiento de señales e imágenes al transformar información de intensidad al dominio de Fourier o dominio de frecuencia. Para aplicar la transformada de Fourier a señales interferométricas se introduce una señal portadora para separar la señal de primer orden de los componentes de orden cero en el dominio de la frecuencia.

En método de la transformada de Fourier propuesta por Takeda en [10] se toma el patrón de franjas adquirido representado generalmente por la ecuación 1.13 y se reescribe como:

$$g(x,y) = a(x,y) + c(x,y)exp(2\pi f_0 x) + c^*(x,y)exp(-2\pi f_0 x), \tag{1.13}$$

donde

$$c(x,y) = \frac{1}{2}b(x,y)exp(i\phi(x,y)), \tag{1.14}$$

y $c^*(x,y)$ denota el complejo conjugado.

Posteriormente se aplica la transformada rápida de Fourier a la ecuación 1.13 con respecto a x obteniendo como resultado:

$$G(x,y) = A(f,y) + C(f - f_0, y) + C^*(f + f_0, y), \tag{1.15}$$

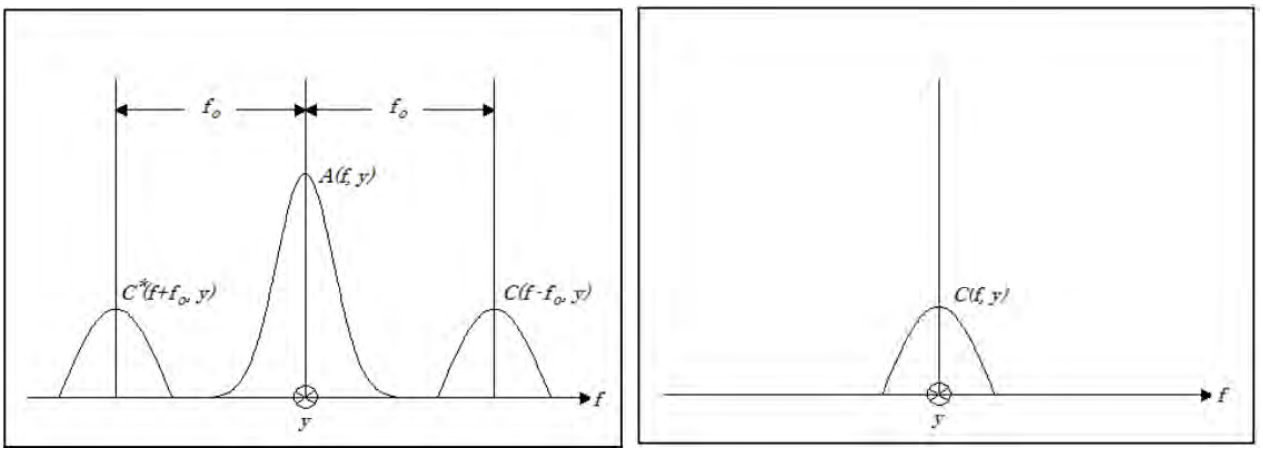
donde las letras mayúsculas denotan el espectro de Fourier y f es la frecuencia espacial en la direccin x . La Figura 1.2 muestra el resultado de aplicar la transformación, obteniéndose la separación de los espectros de Fourier en frecuencia [15].

Para un análisis posterior se puede utilizar cualquiera de los dos espectros correspondientes a la señal portadora, y se desplaza hacia el origen para obtener $C(f,y)$ como se muestra en la Figura 1.2b.

Seguidamente se calcula la transformada inversa de Fourier para $C(f,y)$ con respecto a f para obtener $c(x,y)$. Para obtener la fase a partir del término de la ecuación (1.14) es necesario aplicar la función:

$$\phi(x,y) = tan^{-1}\frac{Im\left[c(x,y)\right]}{Re\left[c(x,y)\right]}. \tag{1.16}$$

Tanto la técnica de interferometría por transformada de Fourier como la de desplazamiento de fase tienen como inconveniente que la fase obtenida se encuentra envuelta, es decir esta indeterminada en un factor de 2π lo cual se hace necesario aplicar un algoritmo de desenvolvimiento de fase [16].



(a) (b)

Figura 1.2: a) Espectro de Fourier para un patrón de Franjas. b) Espectro trasladado.

1.3.2.3. Algoritmos Evolutivos para la demodulación de fase.

El problema de demodulación de fase es considerado como un problema mal planteado *(ill-posed)* [8] [17], debido a que un interferograma puede ser modelado por diferentes funciones de fases y no existe una única solución, es decir, existen diferentes funciones que pueden generar el mismo interferograma. Explícitamente, la Ecuación (1.1) representa un problema inverso donde la información medible es la $I(x, y)$ que representa el patrón de franjas y la información deseada es el término de fase $\phi(x, y)$, dicho de otra manera, la información del objeto físico o el sistema en estudio.

Bajo el supuesto anterior es necesario que el problema modelado por la ecuación (1.1) sea reajustado mediante un proceso de regularización [18] para obtener una estimación adecuada de la fase óptica. Este proceso consiste en restringir el conjunto de soluciones admisibles con la introducción de conocimiento a *priori* adecuado a la solución.

A partir de estas condiciones, es posible obtener una aproximación paramétrica a través de la utilización de algoritmos evolutivos [19]. A pesar que el problema de demodulación ha sido tratado por varios autores, entre ellos se destaca los trabajos presentados por Servin [20–23], Rodríguez Vera [24], Takeda [15], entre otros. Ninguna de las técnicas mencionadas anteriormente utilizan AE para demodular la fase óptica, lo cual hace que este nuevo enfoque mediante la utilización de metaheurísticas abra nuevas puertas para la resolución de este problema.

El estado del arte refleja claramente la novedad de este enfoque que vincula las Ciencias Computacionales con la Metrología Óptica. Los primeros y más alentadores indicios son los trabajos realizados por Cuevas [25–32], donde se muestra la modelación de los algoritmos Genéticos y Optimización basada en Enjambre de Partículas para la demodulación de interferogramas. Entre las principales ventajas que tiene aproximar la fase mediante técnicas de inteligencia artificial, se puede mencionar: que solo se necesita una imagen lo cual es muy útil para eventos que por su naturaleza están en movimiento y no se requiere de un algorit-

mo de desenvolvimiento de fase ya que el mapa de fase se obtiene de forma directa; es una necesidad aplicar otras metaheurística a este tipo de problemas con la finalidad de obtener mejores resultados.

1.4. Organización de la tesis.

El capítulo 1 muestra una revisión del estado del arte, relacionado con las principales técnicas de demodulación de fase y el nuevo enfoque de abordar el problema de demodulación de fase como un modelo de optimización. El capítulo 2 presenta los fundamentos teóricos de los algoritmos de optimización que serán presentados en el desarrollo de la tesis. El capítulo 3 abarcará sobre la modelación matemática de los algoritmos de optimización para el problema de demodulación de fase. El capítulo 5 mostrará los resultados experimentales y un análisis comparativo entre las diferentes metaheurísticas implementadas.

1.5. Consideraciones Parciales

A partir de la revisión bibliográfica realizada para presentar este capítulo se pueden concluir que, en la actualidad la demodulación de interferogramas es comúnmente estudiada a través de técnicas tales como corrimiento de fase, interferometría de Fourier, entre otras; las cuales traen consigo algunas restricciones y requisitos especiales que delimitan el proceso de demodulación en determinadas condiciones.

Sin embargo, a pesar de lo atractivo del problema, de su importancia práctica y del considerable progreso alcanzado en las técnicas utilizadas hasta el momento, el rendimiento de los más avanzados métodos computacionales probados a la fecha sigue siendo considerado insuficiente.

Capítulo 2

Metaheurísticas

Los Algoritmos Evolutivos [33] fueron inspirados por la Teoría de la Evolución de Darwin [34], la cual establece que las especies evolucionan en base a tres conceptos básicos: replicación, variación y selección natural. Estos métodos son utilizados en la resolución de problemas de optimización, búsqueda y aprendizaje de máquina basados en modelos computacionales que emulan algún mecanismo de evolución, conocido como elemento clave en su diseño e implementación.

El inicio de la utilización de las estrategias evolutivas en la solución de este tipo de problemas data del año 1975 cuando John Holland planteó la posibilidad de incorporar los mecanismos naturales de selección y supervivencia a la resolución de problemas de Inteligencia Artificial (Adaptation in Natural and Artificial Systems) [35].

La aparición de las computadoras con gran capacidad de procesamiento de cálculo trajo consigo el incremento del número de aplicaciones de los algoritmos evolutivos en problemas más complejos de búsqueda y optimización, que surgen en la ingeniería, en los campos científicos y antes eran inabordables; a partir de entonces el desarrollo de estas técnicas se ha ido incrementando. Todo esto ha impulsado al desarrollo de procedimientos eficientes para encontrar soluciones aproximadas, donde la rapidez del proceso es tan importante como la calidad de estas. Estos procedimientos son conocidos como métodos heurísticos.

Actualmente las metaheurísticas han cobrado gran importancia por su potencial, entre las que se destacan los Algoritmos Genéticos [36], Programación Genética [37], Estrategias de Evolución y Programación Evolutiva. A continuación se abordará sobre algunas técnicas evolutivas que serán aplicada al problema planteado en el Capítulo 1.

2.1. Recocido Simulado

La técnica del Recocido Simulado (RS) fue formulada por Kirkpatrick, Gelatt y Vecchi en 1983 [38]. Esta metaheurística ha demostrado su eficiencia para resolver una serie de problemas de optimización aplicados a la ciencia e ingeniería. El RS es un enfoque de búsqueda estocástica capaz de escapar de óptimos locales mediante una modificación a la solución actual. Además, se basa fundamentalmente en una solución inicial, la cual se va transfor-

mando en otras soluciones mejores conforme avanza la búsqueda. El nombre esta basado en un algoritmo desarrollado en 1953 [39], donde se implementó un método de Monte Carlo para calcular las propiedades de cualquier material o sustancia compuesta por moléculas interactuando individualmente.

Básicamente, en física el método de recocido se utiliza en la industria para obtener materiales más resistentes, o más cristalinos, en general, para mejorar las cualidades de un material. El proceso consiste en calentar el material a muy alta temperatura. En esa situación, los átomos adquieren una distribución "azarosa" dentro de la estructura del material y la energía del sistema es máxima. Luego se hace descender la temperatura muy lentamente por etapas, dejando que en cada una de esas etapas los átomos queden en equilibrio (es decir, que los átomos alcancen una configuración óptima para esa temperatura). Al final del proceso, los átomos forman una estructura cristalina altamente regular, el material alcanza así una máxima resistencia y la energía del sistema es mínima.

Tomando como base el algoritmo de Metrópolis [39] y los conceptos físicos del proceso de recocido, Kirkpatrick desarrolló un algoritmo de Recocido Simulado para optimización global en problemas de optimización combinatoria. Desde su introducción, el RS ha ganado popularidad como un método de resolución de problemas combinatorios duros.

2.1.1. Descripción Básica

El algoritmo de RS parte de una solución y una temperatura inicial alta. A partir de este nivel la temperatura se va descendiendo lentamente, en cada nivel se explora un número N de soluciones en una vecindad predefinida, las cuales son aceptadas si superan en aptitud a la solución de partida. En caso contrario, se admitirá como nueva solución con una determinada probabilidad:

$$p(\Delta f, T) = e^{-\frac{\Delta f}{T}}, \tag{2.1}$$

donde $\Delta f = f_{SolCandidata} - f_{SolActual}$ y representa el incremento de la función de aptitud, así como T indica el nivel de temperatura. Esta función es conocida como criterio de aceptación de Metrópolis [39]

La probabilidad p de aceptar una solución peor es directamente proporcional a la temperatura mientras mayor sea la temperatura mayor probabilidad de aceptar una solución alejada; esta variante hace que el RS evite y salte a óptimos locales.

El proceso anterior es repetido hasta que se cumple una condición de parada. A continuación se muestra el pseudocódigo del algoritmo de Recocido Simulado 1.

Algoritmo 1 Algoritmo de Recocido Simulado

procedure RECOCIDO SIMULADO ▷ Sea $f(S)$ la aptitud de la solución S y $N(S)$ la vecindad
  $S_0 \rightarrow$ *Solución inicial*
  $T_0 \rightarrow$ *Temperatura inicial*
  $\alpha \rightarrow$ *Función de Temperatura*
  $nrep \rightarrow$ *Número de Iteraciones*
  $STOP \rightarrow$ *Condición de Parada*
  while condicion de STOP **do**
    Generar aleatoriamente una solución $S \in N(S)$
    $\Delta f = f(S) - f(S_0)$
    for $i : nrep$ **do**
      if $\Delta f \leq 0$ **then**
        $S_0 \leftarrow S$
      else
        Generar aleatoriamente $u \in U(0, 1)$
        if $u \leq e^{-\frac{\Delta f}{T}}$ **then**
          $S_0 \leftarrow S$
        end if
      end if
    end for
    $T = \alpha(T)$
  end while
  Retornar la mejor solución encontrada
end procedure

2.1.2. Solución inicial

Por lo general en los algoritmos de búsqueda local existen dos estrategias principales para la creación de soluciones iniciales: el enfoque aleatorio y el voraz (greedy). El enfoque aleatorio consiste básicamente en generar una solución que pertenezca al espacio de solución mediante una distribución normal, mientras la búsqueda voraz da la posibilidad de generar la solución inicial mediante algún tipo de regla heurística. Generalmente este tipo de técnicas toman decisiones en función de la información que está disponible en cada momento, por lo general suelen ser rápidos y fáciles de implementar aunque no siempre garantizan alcanzar la solución óptima. En la mayoría de los casos la inicialización voraz permite a los algoritmos de búsqueda local obtener mejores resultados al final del proceso de búsqueda. Sin embargo, esto depende de la eficiencia y efectividad del algoritmo voraz que se implemente así como de las propiedades del método de búsqueda local seleccionado [40].

Entre estos dos enfoques existen diferencias en cuanto a calidad de solución y tiempo de cómputo. No obstante, a parte de estas dos estrategias se puede generar una solución inicial a partir de un resultado conocido del problema considerado.

2.1.3. Vecindad

Uno de los factores que influyen en la eficiencia del RS es la función de vecindad utilizada [41]. El objetivo de la función de vecindad es formar una solución S' a través de un operador de movimientos que altera ligeramente la solución S. Para ello se hace necesario establecer como obtener las soluciones que conforman la vecindad dada una determinada solución y la forma de seleccionar una de ellas como candidata a ser la nueva solución. Uno de los operadores más utilizados es el llamado *swap* o intercambio de elementos, que puede ser totalmente aleatorio o mediante la aplicación de algún tipo de movimiento inteligente.

2.1.4. Esquema de enfriamiento

Es necesario definir una temperatura inicial lo suficientemente alta que permita movimientos erráticos al principio del proceso [42]. El RS requiere que la temperatura se decremente conforme avanza la búsqueda para ir generando la convergencia a una solución. Existen varios estudios enfocados en esquemas de enfriamiento entre los que se destaca [43, 44]. Los esquemas de enfriamiento pueden dividirse en dos tipos: estáticos y adapativos [45].

El esquema de enfriamiento estático es aquel que es definido complementamente a *priori*. Mientras el esquema adaptativo depende de la información obtenida durante el proceso de búsqueda, es decir, el cambio de temperatura se basa en información estadística. Existen varias funciones de enfriamiento entre las que se destacan el enfriamiento lineal [40], geométrico [46] y el logarítmico [40, 47].

El enfriamiento lineal proporciona una velocidad de descenso de la temperatura constante más o menos rápida en comparación que el resto de las funciones. Esta característica, hace que la solución obtenida probablemente sea un óptimo local. La Ecuación 2.2 muestra como se calcula temperatura actual (T_i):

$$T_i = T_0 - i \times \beta, \tag{2.2}$$

donde β indica la pendiente de la recta con la cual desciende la temperatura e i el número de generación.

El esquema de enfriamiento geométrico es una de las funciones más populares debido a que la velocidad de decremento de la temperatura es más moderado que en otros esquemas. Para calcular la temperatura actual se utiliza la Ecuación 2.3:

$$T_i = \alpha T_{i-1} \qquad \alpha \in [0, 1]\,. \tag{2.3}$$

Algunos trabajos muestran que los valores de α entre 0.8 y 0.99 arrojan los mejores resultados [48] debido a la mayor suavidad obtenida en la velocidad de descenso.

La Ecuación 2.4 exhibe el enfoque de enfriamiento logarítmico. Con esta función es posible obtener mejores resultados aunque trae como inconveniente el tiempo de cómputo requerido por el algoritmo debido al carácter lento que presenta dicha función. Sin embargo se puede combinar con diferentes criterios de parada.

$$T_i = \frac{T_o}{\log(i)}. \tag{2.4}$$

2.1.5. Condición de parada

Aunque teóricamente la condición de parada de RS se establece cuando la temperatura final arrive al valor de cero, según Talbi [40] este enfoque resulta impráctico. Lo ideal sería detener el algoritmo cuando la probabilidad de aceptar un movimiento sea insignificante. Existen algunos estudios sobre el criterio de paro [40, 49], entre los más utilizados en la bibliografía se encuentra:

- ✓ **Definición de una temperatura final**
  Por lo general se toma una temperatura cercana a cero por ejemplo ($T_{min} = 0.01$) o en base a un criterio de experimentación. Este criterio se ha convertido en el más popular debido a que se evita consumir más tiempo de cómputo y obtener soluciones considerables antes de que la temperatura llegue a cero.

- ✓ **Número de iteraciones**
  Se puede definir un número fijo de ciclos que realice el algoritmo o bien se puede tomar en cuenta el comportamiento de cada iteración. Es decir, el algoritmo se detiene cuando después de cierto número de ejecuciones no existe ningún mejoramiento en la solución actual.

- ✓ **Movimientos aceptados**
  Este criterio toma en cuenta las nuevas soluciones que son admitidas en cada nivel de temperatura. Es decir, se define un nivel de aceptación mínimo para el algoritmo.

2.2. Algoritmos Genéticos

Uno de los tipos de algoritmos evolutivos más populares son los Algoritmos Genéticos (AGs) [19], estos se debe en gran medida a John Holland, investigador de la Universidad de Michigan que a finales de 1960 implementó una técnica que simulaba la selección natural y la cual formuló en [35].

A grandes rasgos los AGs constituyen una estrategias de búqueda estocática basados en el mecanismo de selección natural y en algunos casos se involucran aspectos de genética natural, imitando a la evolución biológica como estrategia de búsqueda para la resolución de problemas.

Los AGs difieren de las estrategias de búsqueda convencionales en que estos trabajan sobre un conjunto de soluciones potenciales, llamado población. Esta población está compuesta de una serie de soluciones llamadas individuos y cada individuo esta conformado por una cadena de datos (genes) que representan cada una de las variables involucradas en los procesos de optimización. Sobre cada individuo se aplica ciertos operadores con la finalidad de obtener poblaciones sucesivas mejoradas. En cada ciclo se realizan una serie de operaciones con los individuos de la población, de entre las cuales las más comunes son: la selección, el cruce y la mutación. La aplicación de los operadores anteriores permite obtener soluciones con mejores funciones de adaptación.

En los últimos años, la comunidad científica internacional ha mostrado un creciente interés en los métodos de computación evolutiva, trayendo consigo una ruptura de las fronteras entre AGs, estrategias evolutivas y programación evolutiva. Como consecuencia, en la actualidad, el término *"Algoritmo Genético"* se utiliza para designar un concepto mucho más amplio del que concibió Holland en [35]. Entre esos trabajos se encuentran varios textos descritos por Goldberg (1989) [36], Davis (1991) [50] y Michalewicz (1996) [51].

2.2.1. Analogía entre un AG y su interpretación biológica

En Biología, el Ácido Desoxirribonucleico (ADN) es una molécula presente en casi todas nuestras células que contiene la información genética. Esta molécula posee el código que determina todas las características y el funcionamiento de un individuo. La molécula de ADN tiene forma de hélice doble formada por una combinación específica de cuatro letras, A (adenina), T (timina), C (citosina) y G (guanina) (Ver Figura 2.1).

Figura 2.1: Estructura de la molécula de ADN.

La molécula de ADN se encuentra organizada, almacenada, de una manera estructurada. Estas estructuras en las que se organiza el ADN se denominan *cromosomas*, que a su vez esta formado por *genes*, los cuales constituyen un segmento de ADN que codifica una determinada proteína. Codificar significa en este caso que cada gen contiene información para la producción de una proteína que llevará a cabo una función específica. Las diferentes posibilidades de escoger un rasgo, es decir, escoger entre ojos azules, marrones o verdes, reciben el nombre de *alelos*.

Por otro lado, el genoma es el conjunto de genes contenidos en los cromosomas, lo que puede interpretarse como la totalidad del material genético que posee un organismo o una especie en particular. Normalmente el genoma de una especie incluye numerosas variaciones o polimorfismos en muchos de sus genes, las cuales son conocidas como *genotipos*. El genotipo, unido a algunos factores ambientales que actúan sobre el ADN, determina las características del organismo, es decir, su fenotipo. De otro modo, el genotipo puede definirse como el conjunto de genes de un organismo y el fenotipo como el conjunto de rasgos de un organismo.

Por lo general las especies reproductoras sexualmente almacenan sus cromosomas por parejas. Durante la reproducción sexual se produce una recombinación o cruce: en cada padre, se intercambian los genes entre cada par de cromosomas, para formar un gameto (un cromosoma único), y entonces, los gametos de los dos padres se emparejan para constituir un conjunto completo de cromosomas. La descendencia está sujeta a mutaciones, donde se produce un cambio en algún nucleótido (bits elementales de ADN) de padre a hijo; esas modificaciones son resultado habitualmente de errores de copia.

Haciendo una analogía entre estos conceptos biológicos y la vida artificial de los AGs, un cromosoma indica una posible solución del problema, que por lo general se codifica como una cadena de bits aunque es posible tener otro tipo de codificación tal como real o entera. Los genes son bloques cortos de bits adyacentes que codifican un elemento particular de la solución. Un alelo en una cadena de bits será un 0 o un 1, para un dominio discreto el número de alelos aumenta en dependencia del problema. El cruce al igual que su analogía biológica consiste en el intercambio de material genético entre dos cromosomas padres. La mutación es una permutación de valor en un bit (1 por 0 o viceversa) en un lugar aleatorio (o, en el caso de alfabetos largos, remplazar el símbolo de un lugar escogido aleatoriamente con un símbolo nuevo escogido también aleatoriamente).

Basado en los supuestos anteriores, podemos considerar que los algoritmos genéticos tienen, al menos, estos elementos en común: poblaciones de cromosomas, selección en base a su aptitud, cruces para producir descendencia nueva, y mutación aleatoria de la nueva descendencia. A continuación mostramos un pseudocódigo de un AGs simple donde utiliza sus operadores básicos: selección, cruce y mutación 2:

Algoritmo 2 Algoritmo Genético Simple

Require:
1: *N:* Tamaño de la Población Inicial
2: **procedure** ALGORTIMO GENÉTICO
3: *Generar una población inicial*
4: *Calcular la aptitud de cada individuo*
5: **while NO** Condición de **PARADA do**
6: **for** $i : N/2$ **do**
7: *Selecionar: dos individuos para el proceso de cruce*
8: *Cruzar: los dos individuos con cierta probabilidad*
9: *Mutar: los dos descendientes con cierta probabilidad*
10: *Seleccionar: los individuos para la nueva generación*
11: **end for**
12: **if** Población converge **then**
13: Condición de **PARADA**:= VERDADERO
14: **end if**
15: **end while**
16: Retornar la mejor solución encontrada
17: **end procedure**

El Algoritmo Genético Simple, también denominado Canónico, necesita una codificación o representación del problema, que resulte adecuada al mismo. Además se requiere una función de adaptación, la cual asigna un valor real a cada posible solución codificada. Durante el proceso del algoritmo, los padres deben ser seleccionados para la reproducción, generando dos nuevos hijos, sobre cada uno de los cuales actuará un operador de mutación. El resultado de la combinación de las anteriores funciones será un conjunto de individuos, los cuales en la evolución del AG formarán parte de la siguiente población.

2.2.2. Codificación de los Individuos

Los AGs trabajan con la representación *"cromosómica"* de las soluciones realizando modificaciones sobre las mismas sin conocer la solución que representan cada individuo. Por lo tanto, el proceso desconoce las soluciones, por esto es importante que la codificación permita una representación adecuada y que las modificaciones que el algoritmo realice sobre dichas representaciones tengan significado respecto del problema que se aborde.

En sus inicios la codificación de los AGs suele hacerse mediantes valores binarios, es decir, se establece un determinado número de bits por parámetros y se realiza una discretización de la variable que representa cada gen. Por lo general todos los parámetros se codifican con la misma cantidad de bits, aunque no tienen que ser codificados iguales. La Figura 2.2 muestra un ejemplo de un individuo que posee 3 parámetros codificados con números diferentes de bits.

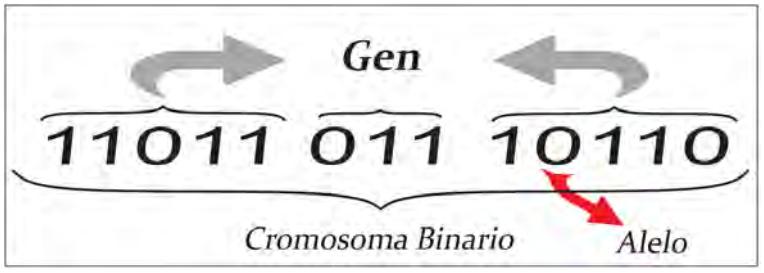


Figura 2.2: Individuo binario.

Existen otras codificaciones que representan cada parámetro mediante un valor entero, real o punto flotante. Sin embargo a pesar de que estas representaciones degradan el paralelismo implícito de la codificaión binaria, son comunmente utilizadas para el desarrollo de operadores genéticos más específicos.

2.2.3. Generación de la población inicial

Al igual que en la mayoría de las metaheurísticas, las soluciones de partida o población inicial para el caso de los AGs, se puede generar aleatoriamente o, a partir de soluciones conocidas obtenidas por experiencias anteriores aunque esta última variante es poco frecuente.

La generación de una población de forma aleatoria es ventajosa debido a que se hace imprescindible para el buen funcionamiento del AG dotar a la población de suficiente variedad de soluciones para explorar todas las zonas del espacio de búsqueda. Por otro lado, al generar

una población aleatoria, si el resultado del proceso de búsqueda es satisfactorio, esto será por las bondades del propio proceso y no por el método de generación de la primera población.

2.2.4. Condición de parada

Entre las condiciones de paradas más elementales se encuentra detener el proceso evolutivo cuando el algoritmo alcance un determinado número de generaciones. Otras condiciones que se utilizan de forma combinada son, ya sea que se logre una solución con una calidad deseada o bien que la población haya convergido, es decir, que la mayor parte de la población sea similar o bien la desviación estándar de la población sea lo suficientemente pequeña y no sea conveniente continuar con la evolución.

2.2.5. Métodos de Selección

En términos biológicos, los individuos más adaptados en un ecosistema o medio ambiente tienen más probabilidad de sobrevivir y de procrear en mayor medida. Sin embargo, en las técnicas metaheurísticas, seleccionar siempre aquellos individuos cuya función de aptitud fuera mejor, correríamos el riesgo de converger a un óptimo local de forma prematura. Por esta razón se hace necesario que la población del AG se someta a un proceso de selección aleatorio que debe tender a favorecer en cierta medida a los individuos más adaptados. Este proceso se puede realizar de diferentes formas.

2.2.5.1. Selección proporcional o por ruleta

La selección proporcional o ruleta utilizada por De Jong [52], fue el criterio de selección de la propuesta original de Holland [35]. En este tipo de elección la probabilidad de selección de cada individuo es proporcional a la aptitud (función de objetivo). Cada individuo recibe una parte proporcional de la ruleta para ser seleccionado, de forma que la suma de todos los porcentajes sea la unidad. La probabilidad de selección p_i de un individuo i-ésimo esta dada por:

$$p_i = \frac{F_i}{\sum_j F_j}, \tag{2.5}$$

donde p_i indica la probabilidad de distribución de cada individuo y F_j representa la función de adaptación del individuo j-ésimo.

Para seleccionar un individuo debemos definir las puntuaciones acumuladas de la siguiente forma:

$$\begin{aligned} &q_0 = 0, \\ &q_i = p_1 + p_2 + \ldots + p_i \quad (\forall i = 1, \ldots, n). \end{aligned} \tag{2.6}$$

Se genera un número aleatorio $a \in [0, 1]$ y se selecciona el individuo i que cumpla:

$$q_{i-1} < a < q_i. \tag{2.7}$$

2.2.5.2. Muestreo estocástico universal

El muestreo estocástico universal es un procedimiento similar al muestreo por ruleta, pero en este caso se genera un solo número aleatorio y a partir de él se obtienen los k individuos que se requieren de igual forma. Este método fue desarrollado por Baker en 1987 [53] con el fin de corregir algunos de los problemas de la selección por ruleta.

Este algoritmo es simple y eficiente. Dado un conjunto de k individuos y sus valores objetivos asociados, se ordenan en una ruleta considerando que el tamaño de los cortes asignados a cada individuo sea proporcional al valor objetivo (como en el algoritmo de ruleta). Después, una segunda ruleta, es marcada con k marcadores igualmente espaciados entre si, donde k es el número de selecciones que deseamos efectuar. Por último se gira la ruleta y se selecciona un individuo por cada marcador. Las posiciones de los marcadores indican los individuos seleccionados. El siguiente algoritmo muestra un pseudocódigo para este método.

Algoritmo 3 Muestreo Estocástico Universal

procedure MUESTREO ESTOCÁSTICO UNIVERSAL
  $k \rightarrow$ Número de individuos a Seleccionar
  $q \rightarrow$ Vector conteniendo la Probabilidad acumulada
  $ptr = Rand() \rightarrow$ Genera un valor aleatorio entre [0,1]
  $sum = 0$
  $idx = 1 \rightarrow$ Contador indicando el número de individuo
  for $i = 1; i \leq k; i++$ **do**
    $a_i = \frac{ptr+i-1}{k} \qquad (\forall i = 1, \ldots, k)$
    while $sum < a_i$ **do**
      $sum+ = q(idx++)$
    end while
    Seleccionar (idx);
  end for
end procedure

2.2.5.3. Selección por Torneo

Este método fue propuesto por Goldberg y Deb en 1991 [54], se basa fundamentalmente en la selección al azar un pequeño subconjunto de la población para efectuar una comparación entre ellos. Este algoritmo tiene la ventaja de que disminuye la velocidad de la aplicación y previene en cierto grado la convergencia prematura. Sin embargo trae como inconveniente establecer el tamaño del subconjunto.

Cuando el número de individuos que participa en el torneo es elevado aumenta la presión de selección y los peores individuos apenas tienen oportunidad de reproducirse, convirtiendose en una selección elitista. Cuando el tamaño del subconjunto es reducido, la presición disminuye y los peores individuos tienen mayor oportunidad de ser seleccionado. La Figura 2.3 muestra un ejemplo básico de una selección por Torneo.

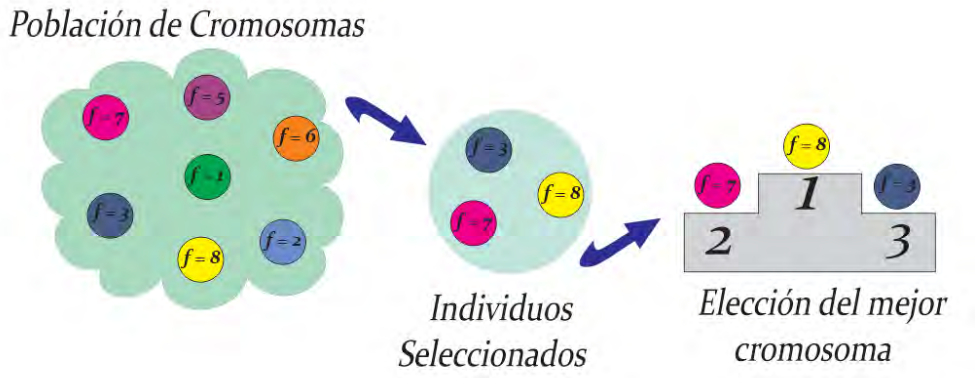


Figura 2.3: Ejemplo de selección por Torneo para un problema de maximización.

Considerando un ejemplo de funcionamiento de los métodos anteriores (Tomado del libro Algoritmos Evolutivos: Un enfoque práctico [19]). Supongamos que tenemos la siguiente población de 8 individuos:

Individuo	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Aptitud	4	1	1	2	3	2	5	2
p_i	0.20	0.05	0.05	0.10	0.15	0.1	0.25	0.10
q_i	0.20	0.25	0.30	0.40	0.55	0.65	0.90	1.00

Tabla 2.1: Población de 8 individuos.

La Tabla 2.1 aparecen las probabilidades p_i de los individuos (Ecuación 2.5), proporcionales a su adaptación, y la probabilidad acumulada q_i obtenida mediante la Ecuación 2.6. Suponiendo que se desea seleccionar 3 individuos utilizando el método de la ruleta, para ello es necesario generar 3 valores aleatorios en el rango de $[0, 1]$:

$$a_1 = 0.8751$$
$$a_2 = 0.1391$$
$$a_3 = 0.2582.$$

Según el método de selección de la ruleta las posibles soluciones seleccionadas serían aquellas que cumplan con las siguientes condiciones:

$$q_6 < a_1 < q_7 \rightarrow x_7$$
$$q_0 < a_2 < q_1 \rightarrow x_1$$
$$q_2 < a_3 < q_3 \rightarrow x_3.$$

Aplicando el procedimiento de muestreo estocástico universal generamos un valor aleatorio que supongamos que toma el siguiente valor:

$$a = 0.4578$$

Aplicando el Algoritmo 3 con $k = 3$ obtenemos:

$$a_1 = \frac{0.4578 + 1 - 1}{3} = 0.1526$$
$$a_2 = \frac{0.4578 + 2 - 1}{3} = 0.48593$$
$$a_3 = \frac{0.4578 + 3 - 1}{3} = 0.81926.$$

Al igual que en el método de la ruleta serían seleccionado aquellos valores que cumplan con las condiciones expresada en la Ecuación 2.7:

$$q_0 < a_1 < q_1 \rightarrow x_1$$
$$q_4 < a_2 < q_5 \rightarrow x_5$$
$$q_6 < a_3 < q_7 \rightarrow x_7.$$

Enfocándonos en el método de selección por torneo y tomando como tamaño del subconjunto $N = 3$, debemos generar valores aleatorios enteros entre el intervalo $[1, 8]$, que indica los individuos seleccionados. Por ejemplo la Tabla 2.2 muestra el resultado de la selección aleatoria:

Individuo	*Aptitud*
$u_1 = 5$	$\rightarrow 3$
$u_2 = 3$	$\rightarrow 1$
$u_3 = 7$	$\rightarrow 5$

Tabla 2.2: Selección aleatoria de una muestra de 3 individuos.

Considerando un problema de maximización, el individuo 7 posee la mejor aptitud por lo cual es seleccionado para el proceso de reproducción.

2.2.5.4. Selección basada en Ranking o Jerarquía

Este algoritmo fue propuesto por Baker [53], con el objetivo de prevenir una convergencia prematura ya que el valor esperado depende más de su valor de ranking que de su valor de probabilidad. Para llevar a cabo el proceso de selección se realiza un ordenamiento de la población en función de su *aptitud* el cual es necesario para asignar la probabilidad de selección en función del valor de ranking. La probabilidad asociada a su selección esta dada por:

$$VE(i,t) = Min + (Max - Min) * \frac{Rango(i,t) - 1}{N - 1}, \tag{2.8}$$

donde Max y Min están relacionados por la siguiente expresión e indica la presión de selección que no es más que un proceso de selección que elige los individuos más aptos para la reproducción. Este mecanismo garantiza un equilibrio entre la exploración (soluciones con alta penalización de la función de costo deben tener la oportunidad de pasar a la siguiente

generación) y explotación (las mejores soluciones tienen mayor probabilidad de pasar a la siguiente generación) en esta técnica de selección.

$$1 \leq Max \leq 2$$
$$Min = 2 - Max.$$

$Rango(i,t)$ representa la posición del individuo después del proceso de ordenamiento y N el número total de individuos de la población. Según Baker recomienda utilizar un $Max = 1.1$ y debe cumplirse esta restricción:

$$\sum_{i}^{N} VE(i,t) = N. \tag{2.9}$$

Este técnica de selección trae como desventaja que requiere mucho tiempo de cómputo debido a que se requiere de un algoritmo de ordenamiento, y a medida que la población aumente se incrementa la complejidad del algoritmo. El Algoritmo 4 muestra un pseudocódigo del operador basado en Ranking:

Algoritmo 4 Selección Basada en Ranking

Require:

$Rango(elemt, array)$ → Función que calcula la posición del elemento dentro del arreglo
N → Tamaño de la Población Inicial
$pob : array$ → Población de Individuos
$MaxyMin$ → Intervalo de Rankeo
k → Número de Individuos a seleccionar
procedure RANKEO
    $pob\_Ordenada := QuickSort(pob)$ → Método de Ordenamiento
    for $i = 1; i \leq N; i++$ **do**
        $VE(i) = Min + (Max - Min) * \frac{Rango(pob(i), pob\_Ordenada) - 1}{N-1}$
    end for
    for $i = 1; i \leq k; i++$ **do**
        $sum = 0$
        $idx = 1$ → Contador de la población
        $U_i := Rand(Min, Max)$
        while $sum < U_i$ **do**
            $sum+ = VE(idx++)$
        end while
        Seleccionar (idx);
    end for
end procedure

2.2.5.5. Escalamiento Sigma

El Escalamiento Sigma, tratada como *truncación sigma* en [36], es una técnica ideada para mapear la aptitud original de un individuo con su valor esperado y el valor actual de la

desviación estandar de la población , de manera que el AG sea menos susceptible a la convergencia prematura. Esta técnica mantiene la presión de selección (es decir, el grado en el que se permite a los individuos altamente ajustados a reproducirse) relativamente constante en el transcurso de todo el proceso evolutivo en lugar de depender de las variaciones de la aptitud de la población.

Usando este método de escalamiento, el valor esperado de un individuo está en función de su aptitud, la media y la desviación estándar de la población. Este técnica utiliza la desviación estándar como una medida de la variación en la información contenida en esta. Un ejemplo de escalamiento sigma es el siguiente [55].

$$VE(i,t) = \begin{cases} 1 & \text{si } \sigma(t) = 0 \\ 1 + \frac{f(i) - \hat{f}(t)}{2\sigma(t)} & \text{si } \sigma(t) \neq 0, \end{cases} \tag{2.10}$$

donde $VE(i,t)$ es el valor esperado del individuo i-ésimo en el tiempo t, $f(i)$ corresponde a la aptitud del individuo, $\hat{f}(t)$ es la aptitud media poblacional y $\sigma(t)$ indica la desviación estandar y viene dado por la Ecuación 2.11:

$$\sigma(t) = \sqrt{\frac{\sum (f(i) - \hat{f}(t))^2}{N}}. \tag{2.11}$$

Este escalamiento tiene como ventaja que al principio de las iteraciones, cuando la desviación estándar de la aptitud de la población es considerablemente alta, los individuos mejor codificados no serán tan prominentes para ser selecionado para la reproducción. Igualmente, cuando el número de iteraciones sea elevado, la convergencia será típicamente alta y la desviación estándar sería pequeña, por lo tanto, los individuos mejor codificados sobresaldrían permitiendo que la evolución continue.

2.2.5.6. Selección de Boltzman

Esta técnica planteada por Goldberg [56] es un acercamiento similar al método de recocido simulado, el cual usa una función de variación de temperatura que controla la presión de selección. Con este operador de selección se pretende mantener la presión de selección uniforme al comienzo y elitista al final, es decir, de menor presión a mayor presión de selección con el objetivo de dar mayor oportunidad a los individuos menos aptos y de esta forma realizar una búsqueda exhaustiva en el espacio de solución. Al principio la selección de Bolztman usa un valor alto de temperatura, lo cual hace que la presión de selección sea baja y la búsqueda sea prácticamente aleatoria haciendo que todos los individuos tengan la misma probabilidad de ser seleccionados y de esta forma evitar caer en mínimos locales. Con el transcurso de las generaciones, la temperatura se reduce gradualmente, lo que aumenta la presión de selección y convierte el proceso de selección en un procedimiento elitista.

Para calcular el valor esperado de un individuo se usa la siguiente expresión:

$$VE(i,t) = \frac{e^{\frac{f(i)}{T}}}{\langle e^{\frac{f(i)}{T}} \rangle^t}, \tag{2.12}$$

donde T es la temperatura y $\langle \quad \rangle^t$ denota el promedio de la población en la generación t.

2.2.6. Cruzamiento

Un operador genético de cruzamiento, conocido también como *recombinación*, es el más importante de esta técnica metaheurística ya que genera variación poblacional en la cual aparecen nuevos individuos no presente en generaciones anteriores. El objetivo es generar diversidad mediante el cruce en el cual se recombina la información de diferentes individuos, para producir la descendencia que será insertada en la próxima generación, confiando en que los individuos obtenidos de esta manera representen soluciones de mejor calidad.

Al igual que en los sistemas biológicos el cruce constituye una estrategia de reproducción sexual y suma gran importancia para los AGs debido a que la tasa de cruce con la que habitualmente se implementan los AGs es aproximadamente sobre el 90 %.

Generalmente los métodos de cruce se dividen en 2 estrategias diferentes: destructiva y no destructiva. La estrategia destructiva se caracteriza por que los descendientes se insertarán en la próxima población aunque los padres posean mejor aptitud. Por el contrario en la estrategia no destructiva la descendencia alcanzará la próxima generación si y solo si supera la aptitud de los padres. La idea principal de este operador consiste en tomar dos individuos correctamente adaptados y obtener una descendencia que comparta genes de ambos, existe la posibilidad de que los genes heredados sean precisamente los causantes de la bondad de los padres.

Existen multitud de algoritmos de cruce. Sin embargo, los más empleados son los que se detallarán a continuación:

- ✓ Cruce de N puntos
- ✓ Cruce Uniforme
- ✓ Cruce para codificaciones no binarias

2.2.6.1. Cruce de N puntos

Es una de las técnicas de cruzamiento más usuales para los individuos codificados de forma binaria. Se definen cuantos puntos de cruzamiento se desea, las localidades en donde se llevará las cruzas y el intercambio de información. El Algoritmo 5 muestra un pseudocódigo general de este tipo de cruce.

Algoritmo 5 Cruce de N puntos

Require:

$padre1 : array$ → Arreglo de binario
$padre2 : array$ → Arreglo de binario
N → Número de puntos a cruzar
l → Longitud de la cadena cromosómica
procedure CRUCE DE N PUNTOS
    Generar aleatoriamente el vector U donde: $U[1] \leq U[2] \leq \ldots \leq U[N]$
    $hijo1 := padre1$
    $hijo2 := padre2$
    if N mod 2 == 1 **then**
        $U[N+1] = l$
    end if
    for $i = 1;\ i \leq U.length/2;\ i+ = 2$ **do**
        for $j = U[i-1];\ j <= U[i];\ j++$ **do**
            $Temp = hijo1[j]$
            $hijo1[j] = hijo2[j]$
            $hijo2[j] = Temp$
        end for
    end for
    Retornar $(hijo1,\ hijo2)$
end procedure

Generalmente en esta metodología se utilizan uno o dos puntos de cruces, debido a que la utilización de más puntos de cruces en el algortimo lo hace muy destructivo [52]. El problema principal de incorporar nuevos puntos está en que los segmentos originados pierden la bondad que poseían en conjunto. Sin embargo es cierto que al añadir más puntos al proceso de cruce se abre la posibilidad de explorar más a fondo el espacio de búsqueda. A continuación describiremos los métodos particulares más utilizado en esta técnica de cruce.

Cruce de 1 punto

El cruzamiento de 1 punto es conocido como SPX por sus siglas en inglés y constituye la técnica de cruce más sencilla implementada en los AGs. Según este tipo se cruce, se selecciona aleatoriamente un punto de la cadena en cada uno de los padres y se generan nuevos individuos combinando las partes que generan los cortes anteriores. De esta manera los descendientes heredan la información genética de los padres, tal y como se muestra en la Figura 2.4.

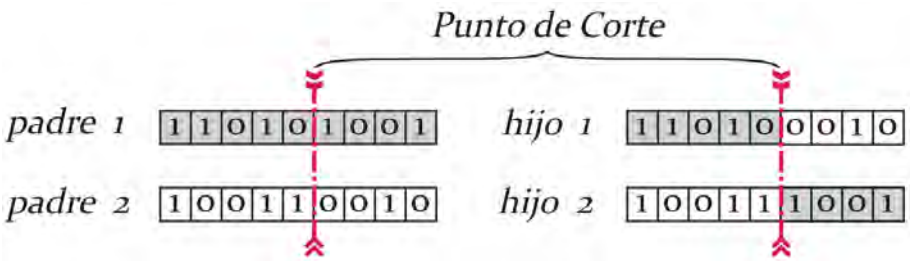


Figura 2.4: Cruce de 1 punto.

Cruce de 2 puntos

El operador con dos puntos de corte es análogo al anterior, salvo que se seleccionan dos puntos de corte y los padres intercambian los elementos de la cadena que quedan entre dichos puntos para generar los descendientes. Deberá tenerse en cuenta que ninguno de estos puntos coincida con algún extremo para garantizar la existencia de 3 segmentos. La Figura 2.5 ejemplifica este operador de cruce.

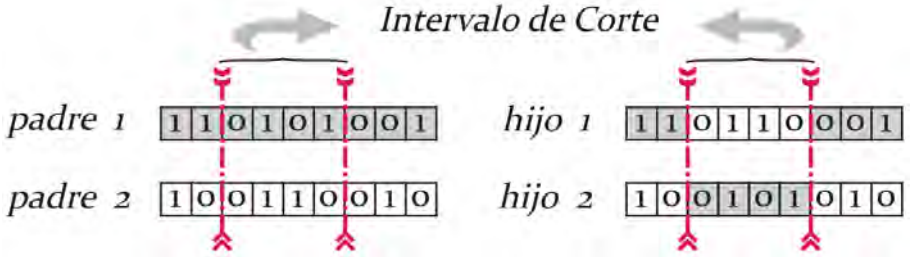


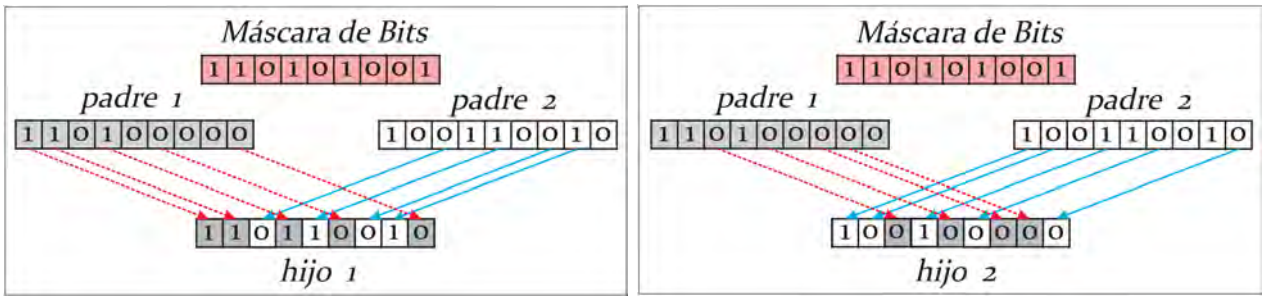
Figura 2.5: Cruce de 2 puntos.

Generalmente este tipo de cruce es conocido por sus siglas en inglés DPX (Double Point Crossover) y en comparación con el cruce de un solo punto aporta una mejora sustancial en los AGs.

2.2.6.2. Cruce Uniforme

Aunque esta técnica se puede implementar de diversas formas, una de ellas utiliza la generación de una máscara binaria de longitud igual a la cadena cromosómica de los padres. En este caso, los genes de los descendientes se obtienen de acuerdo con el criterio dado por la máscara. Para cada posición de la descendencia se tomará el gen del padre 1 si el valor de la máscara para dicha posición es 1 y del padre 2 si el valor del gen es 0 (Figura 2.6a). Para generar el segundo descendiente se debe intercambiar los roles de los padres o se puede intercambiar la interpretación de los unos y los ceros de la máscara de cruce (Figura 2.6b).

Es válido mencionar que la máscara de cruce puede no permanecer intacta durante todo el proceso evolutivo o bien se puede generar aleatoriamente para cada cruce. La siguiente figura muestra un ejemplo de este operador.



(a) (b)

Figura 2.6: Cruce uniforme a) Descendiente 1. b) Descendiente 2.

2.2.6.3. Cruce para codificaciones no binarias

Aunque los métodos vistos anteriormente se pueden utilizar para cualquier tipo de codificación del AG, si se emplea otro método de codificación donde involucre valores enteros o reales con frecuencia se define otros operadores de cruces como [57]:

✓ **Media**
Este operador trae como inconveniente que solo se genera un descendiente durante el cruzamiento. Consiste en generar una descendencia con el valor medio de los genes de los padres.

$$H_i = \frac{P1_i + P2_i}{2},$$

donde $P1$ y $P2$ representa a los padres escogidos para la reproducción y H el descendiente obtenido a partir del proceso de cruce.

✓ **Media geométrica**
Consiste en que cada gen de la descendencia toma como valor la raíz cuadrada del producto de los genes de los padres. Este operador al igual que el anterior presenta como desventaja la obtención de una sola descendencia y además se le suma el problema del signo cuando los padres tienen signos diferentes.

$$H_i = \sqrt{P1_i * P2_i}.$$

✓ **Extensión**
Esta variante resuelve el problema de generar un único hijo, es decir, radica en tomar la diferencia existente entre los genes situados en las mismas posiciones de los padres y se suma al valor más alto o se resta del valor más bajo.

$$\Delta_i = |P1_i - P2_i|,$$
$$H1_i = P1_i + max(\Delta_i),$$
$$H2_i = P2_i - min(\Delta_i),$$

donde $H1$ y $H2$ indican los hijos productos del cruzamiento y Δ_i la diferencia entre los padres.

2.2.7. Mutación

La mutación se considera un operador básico, que proporciona un pequeño elemento de aleatoridad en los individuos de la población. Es decir, imita un comportamiento natural, pues cuando se genera la descendencia siempre se produce algún tipo de diversidad en la población que nunca ha aparecido, por lo general sin mayor trascendencia, en el paso de la carga genética de padres a hijos.

El operador de mutación ocurre con muy baja probabilidad, generalmente menor al 1 % del total de genes de los individuos de la población. Esto se debe sobre todo a que los individuos suelen tener un ajuste menor después de mutados. Sin embargo se realizan mutaciones con el objetivo de escapar de mínimos locales y mantener una mayor diversidad en la población. Suponiendo que un individuo esta codificado con una cadena binaria el operador de mutación consistiría en variar aleatoriamente un gen del cromosoma, es decir, se modifica un 1 por el 0 o viceversa (Ver Figura 2.7).

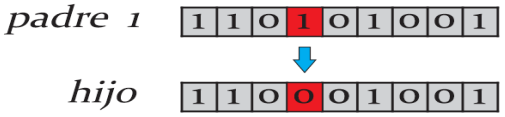


Figura 2.7: Operador de Mutación.

2.2.8. Reducción

Una vez obtenidos los individuos descendientes de una determinada población en el tiempo t, el proceso de reducción al tamaño original, consiste en escoger k individuos de entre los $2k$ individuos que forman parte de la población en el tiempo t. Dicho proceso se suele hacer fundamentalmente de varias formas distintas:

- ✓ **Aleatorio**
  El nuevo individuo se inserta en un lugar escogido de manera aleatoria en la población.

- ✓ **Simple**
  Los k individuos descendientes son los que forman parte de la población en el tiempo $t + 1$.

- ✓ **Elitista**
  Se escogen de entre los $2k$ individuos, los k individuos con mejor aptitud.

2.2.9. Teoría de esquemas

El fundamento teórico de los AGs reside en la noción de la Teoría de Esquemas [35]. Se pueden observar numerosas coincidencias entre las cadenas con las que trabaja un AG, las cuales pueden ayudar a guiar la búsqueda. Para analizar AG de una manera más cuantitativa es necesario definir algunos términos: un **esquema** es un patrón de coincidencia que se construye sobre el alfabeto $\{0, 1, *\}$ donde el símbolo $*$ representa un carácter de tipo

comodín que puede contener 0 ó 1 en dependencia del caso. En otras palabras un esquema representa a todos los individuos que coinciden con los ceros y unos del esquema, pudiendo contener cualquier símbolo en las posiciones en la que el esquema tiene el carácter $*$.

Por ejemplo, la notación $100*0$ representa el conjunto de cadenas $\{10010, 10000\}$, y el esquema $10*0*1$ simboliza el siguiente grupo de cuatro cadenas: $\{100001, 100011, 101001, 101011\}$. Los esquemas que no contienen símbolo $*$ representa a una única cadena y los que solo contienen $*$ representa a todas las cadenas de la longitud del esquema. Puesto a que cada comodín $*$ puede tomar dos valores, o 0 ó 1, un esquema con m comodines representa a 2^m cadenas.

Para analizar el comportamiento de los esquemas contenidos en una población de un AG es necesario formalizar ciertas propiedades de los esquemas: el **orden** de un esquema y la **longitud**. Estas propiedades están relacionadas con la representatividad y con la probabilidad de que el esquema sobreviva al cruce. Dado un esquema H se define:

Orden de un esquema: O(H),constituye el número de posiciones fijas contenido en dicho esquema. Por ejemplo, $O(110*1) = 4$, mientras el orden de $O(****) = 0$.

Longitud de un esquema: $\delta(H)$, se define como la distancia entre la primera y la última posiciones fijadas de la cadena. El esquema $\{110*1\}$ tiene una longitud $\delta = 4$ debido a que la última posición especificada es 5 y la primera es 1, mientras $\delta(****) = 0$.

La idea clave al explicar porqué un AG funciona se refiere a la tendencia que tiende a evolucionar la presencia de un determinado tipo de esquema en la población. La presencia $m(H, t)$ de un esquema H se define como el número de cadenas de la población en el instante t que pertenecen al esquema H. Por ejemplo, la siguiente tabla muestra la presencia de esquemas en una población determinada.

Población(t)	**Presencia de esquemas**
10101	
11101	$m(1***1) = 4$
10001	$m(**10*) = 2$
00110	$m(*0*0*) = 2$
11001	

Tabla 2.3: Ejemplo de la presencia de esquemas en una población de individuos.

Con la notación introducida hasta el momento, podemos enunciar el Teorema de los esquemas de los AGs, el cual relaciona estadísticamente la presencia de esquemas de una población P en un instante t con la presencia de la próxima generación de acuerdo a la siguiente ecuación:

$$m(H, t+1) \geq m(H, t) * \frac{f(H)}{\bar{f}}\left[1 - p_c\frac{\delta(H)}{l-1} - O(H)p_m\right], \tag{2.13}$$

donde p_c y p_m indica la probabilidad de cruce y mutación, l el número de bits del esquema, $f(H)$ es la media de las adaptaciones de las cadenas de la población que pertencen al esquema

H y $\bar{f}$ indica la aptitud media de la población y se define como:

$$\bar{f} = \sum \frac{f_i}{N}, \tag{2.14}$$

con N tamaño de la población.

Este teorema refleja que los esquemas cortos, de orden bajo adaptados por encima de la media, incrementan su presencia en la población de forma exponencial en próximas generaciones. Este resultado se evidencia claramente en el estudio de los efectos de las operaciones de selección, cruce y mutación.

- ✓ ***Efecto selección***
  Utilizando la notación anteriormente introducida, se tiene que la probabilidad de que un individuo seleccionado para cruzarse pertenezca a un esquema H, se puede calcular por medio de $\frac{f(H)}{\bar{f}}$. Tomando una población sin solapamiento y tamaño N a partir de la población $P(t)$ se espera obtener $m(H, t+1)$ representantes del esquema H en el instante $t+1$. Como conclusión de este planteamiento se deduce la siguiente ecuación:
  $$m(H, t+1) = m(H, t)\frac{f(H)}{\bar{f}}, \tag{2.15}$$
  es decir la precencia de un esquema crece en proporción a la adaptación media del esquema respecto de la aptitud de la población.

- ✓ ***Efecto cruce***
  Denotando por $P_c(H)$, la probabilidad de que el esquema H sea eliminado por el operador de cruce, se tiene que:
  $$P_c(H) = \frac{\delta(H)}{l-1}, \tag{2.16}$$
  Debido a que el operador de cruce se lleva a cabo con una cierta probabilidad, si ninguno de sus descendientes se encuentra en H, entonces dicho operador se halla restringido en su límite superior por:
  $$P_c(H) = p_c\frac{\delta(H)}{l-1}, \tag{2.17}$$
  mientras la probabilidad de sobrevivir por cruza esta dada por:
  $$P_c(H)^{-1} = 1 - p_c\frac{\delta(H)}{l-1}, \tag{2.18}$$

- ✓ ***Efecto mutación***
  La mutación se realiza alelo por alelo en toda la cadena cromosómica, lo que induce que la probabilidad de muta (p_m) tiene valores muy cercanos a cero. Bajo este supuesto la probabilidad de supervivencia de un esquema H por muta esta dada por:
  $$P_m(H) = (1 - p_m)^{O(H)}, \tag{2.19}$$
  para valores de $p_m \ll 1$ la probabilidad de sobrevivir un esquema en la muta es aproximadamente igual a:
  $$P_m(H) = (1 - O(H))p_m. \tag{2.20}$$

2.3. Optimización basada en Mallas Variables

La Optimización basada en Mallas Variables (Variable Mesh Optimization; VMO) es una metaheurística poblacional con características evolutivas donde un conjunto de nodos que representan soluciones potenciales a un problema de optimización, forman una malla (población) que dinámicamente crece y se desplaza por el espacio de búsqueda (evoluciona).

Para ello se realiza un proceso de expansión en cada ciclo, donde se generan nuevos nodos en dirección a los extremos locales (nodos de la malla con mejor calidad en distintas vecindades) y el extremo global (nodo obtenido de mejor calidad en todo el proceso desarrollado); así como a partir de los nodos fronteras de la malla. Luego se realiza un proceso de contracción de la malla, donde los mejores nodos resultantes en cada iteración son seleccionados como malla inicial para la iteración siguiente.

2.3.1. Descripción general de la metaheurística.

La esencia del método VMO es crear una malla de puntos en el espacio m dimensional, donde se realiza el proceso de optimización de una función $FO(x_1, x_2, \ldots, x_m)$; la cual se mueve mediante un proceso de expansión hacia otras regiones del espacio de búsqueda. Dicha malla se hace más "fina" en aquellas zonas que parecen ser más promisorias. Es variable en el sentido que la malla cambia su tamaño (cantidad de nodos) y configuración durante el proceso de búsqueda. Los nodos se representan como vectores de la forma $n(x_1, x_2, \ldots, x_m)$.

El proceso de generación de nodos [58] [59] está comprendido por varios pasos. El primero de ellos, ***generación de la malla inicial*** consta de Ni nodos los cuales en la primera iteración son generados de forma aleatoria o por otro método que garantice obtener soluciones diversas. El segundo es la ***generación de nodos en dirección a los extremos locales*** constituye otro tipo de exploración que se realiza en VMO y se lleva a cabo en las vecindades de cada uno de los nodos de la malla inicial. Para ello se buscan los vecinos más cercanos de cada nodo n a través de una función de distancia o semejanza.

Al igual que en el paso anterior el tercero, la ***generación de nodos en dirección hacia el extremo global*** [58] [60] tiene como propósito realizar una exploración global hacia el nodo que mejor calidad ha tenido hasta el momento (extremo global); para lo cual, se generan nuevos nodos a partir de cada nodo de la malla inicial en dirección a este.

El cuarto paso es la ***generación de nodos a partir de los nodos más externos de la malla*** [58] [60]. Esto es un proceso más de expansión de nuevos nodos; tiene lugar con el objetivo de explorar el espacio de búsqueda en dirección a las fronteras de cada dimensión.

El modelo incluye un conjunto de parámetros involucrados en el proceso de exploración de VMO, su comportamiento ha sido estudiado en Puris (Cceres, 2009). El quinto paso se centra en la ***cantidad de nodos de la malla inicial*** (N_i) ya que es muy importante la definir el tamaño de las poblaciones para obtener un buen desempeño de estos métodos debido a que no todos los individuos influyen en la evolución de la población. Para todos los casos, el tamaño total de la expansión es $N = 3Ni + Ni/2$, con el objetivo de generar con

el operador de las fronteras una cantidad de nodos, al menos, igual a la mitad de la malla inicial. Además, el tamaño de la vecindad (k) y la condición de parada (M) son parámetros que utiliza el modelo y juegan un rol importante en el desempeño del mismo.

Luego de tener bien definidos los procesos del método VMO, el siguiente algoritmo presenta un pseudocódigo con la estructura general del mismo, donde la condición de parada está descrita por un número máximo de evaluaciones de la función objetivo.

Algoritmo 6 Optimización basada en Malla Variables

```
1: procedure VMO
2:     Generación de la malla inicial (Ni) nodos de forma aleatoria.
3:     Evaluar los nodos de la malla inicial y seleccionar el mejor ng
4:     while Evaluaciones < M do
5:         for Para cada Nodo en la malla inicial del ciclo do
6:             Encontrar sus k nodos más cercanos
7:             Determinar el mejor de los vecinos ne
8:             if ne es mejor que el nodo actual then
9:                 Generar nuevo nodo en dirección a los extremos locales
10:            end if
11:        end for
12:        for Para cada Nodo en la malla inicial del ciclo do
13:            Generar un nuevo nodo en direccin al extremo global (ng)
14:        end for
15:        Seleccionar los nodos más externos de la malla
16:        Generar nuevo nodo a partir de los nodos más externos de la malla
17:        while Los N nodos de la malla en el ciclo actual no se hayan generado do
18:            Generar nuevo nodo aleatoriamente
19:        end while
20:        Construir la malla inicial actual de manera elitista
21:        Ordenar los nodos de la malla inicial según su calidad.
22:        Aplicar operador de limpieza adaptativo
23:    end while
24: end procedure
```

2.4. Consideraciones parciales.

En el estudio presentado en este capítulo se pudo encontrar una revisión teórica acerca de las metaheurísticas que serán modeladas como un problema de minimización para recuperar el campo de fase. A pesar de la sencilla implementación del Recocido simulado, se evidencia la fortaleza del método demostrada en varias aplicaiones.

Por otra parte está más que demostrado el uso de los Algoritmos Genéticos en disímiles aplicaciones, esta metaheurística poblacional ya ha sido probado para la demodulación de fase con resultados promisorios. Por último una metaheurística poblacional que a pesar de su novedad ha tenido resultados alentadores en determinadas aplicaciones será presentada.

La optimización Basada en Mallas Variables es una metaheurística surgida recientemente en los laboratorios de la Universidad Central Marta Abreu de Las Villas, esta novedad abre la posibilidad de probar su capacidad en problemas de esta índole.

Capítulo 3

Modelación Matemática

La interferometría es un método de medición que aplica el fenómeno de interferencia de las ondas. Las mediciones pueden incluir otras determinadas características de las propias ondas y los materiales por los que se propagan. Además, la interferometría se utiliza para describir las técnicas que involucran ondas luminosas y estudia los cambios de desplazamiento. La interferometría de medición de desplazamiento se utiliza ampliamente en la calibración y el control de movimiento.

En Metrología Óptica un patrón de franjas es el producto de la superposición de dos haces de luz proveniente de un láser o simplemente luz blanca (luz estructurada) en el cual se requiere un único haz. Para el caso de interferencia se debe cumplir que los dos haces de luz deben originarse en la misma fuente, para que tengan la misma frecuencia , igual longitud de onda, la misma polarización y no exceda la diferencia de camino óptico permitida.

3.1. Proyección de franjas

El método de proyección de franjas tiene la ventaja de solo requerir una sola imagen para llevar a cabo una reconstrucción adecuada del objeto bajo observación. Cuenta además con la ventaja de que pueden medirse objetos de distintas dimensiones a través de la modificación de la frecuencia de las franjas [61], el único inconveniente viene dado tanto por la resolución del sistema de proyección como de la cámara.

Normalmente, un patrón de franjas (sinusoidal o de tipo Ronchi), es proyectado sobre un plano de referencia y posteriormente sobre el objeto en estudio. Hoy en día este proceso se realiza por medios completamente digital lo que le brinda gran versatilidad y velocidad de desempeño. En la Figura 3.1 se ilustra el arreglo experimental utilizado para la obtención del patrón de franjas.

Figura 3.1: Sistema de proyección de franja.

En general, si proyectamos un patrón de franjas con periodo d perpendicular al objeto en estudio, el cual puede representarse mediante una curva S, según ilustra la Figura 3.2. Debido a la presencia de un objeto, las franjas localizada en el plano de referencia sufren un desplazamiento relacionado con la geometría del sistema. Si las franjas proyectadas presentan un perfil cosenoidal entonces este desplazamiento reflejará un cambio en el argumento de la función coseno, es decir, produce un cambio en la fase que llamaremos $\phi(x, y)$.

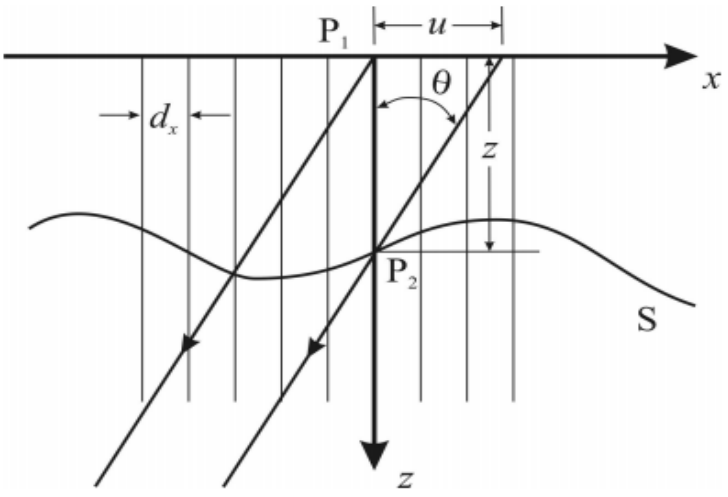


Figura 3.2: Geometría de proyección de un patrón de franjas.

3.2. Sombreado de Moiré

El principio físico de la técnica de Moiré de sombra consiste en una fuente de luz colocada en un punto, que ilumina en un ángulo oblicuo una rejilla que se coloca frente al objeto en estudio, esta rejilla produce una sombra sobre la superficie del objeto. Un observador se coloca en la posición o una cámara delante de la rejilla, las sombras formadas en la superficie (el objeto) son o bien vistos por el observador o capturados por una cámara. El resultado es un mapa topográfico, que resulta de la interferencia de la rejilla maestro y la sombra de la rejilla.

En la Figura 3.3 se hace una representación cualitativa de la intensidad de luz que se refleja a través de la rejilla que deja pasar la luz, si registramos la información con una cámara como se muestra en la figura se observan 2 rejillas una de ellas corresponde a la rejilla patrón y la otra generada por la sombra la cual se encuentra desfasada con respecto a la rejilla patrón.

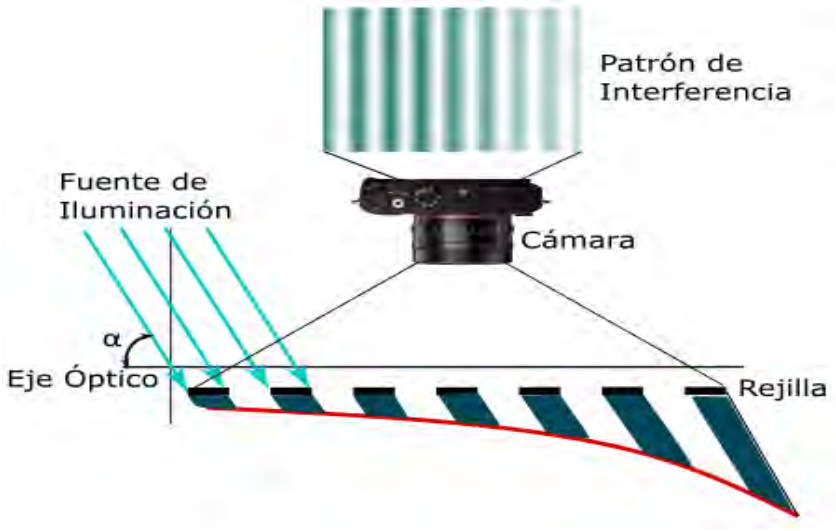


Figura 3.3: Geometría de proyección de un patrón de franjas.

Este efecto se da cuando se forman franjas de Moiré entre una rejilla y su propia sombra, por lo que se conoce como sombreado de Moiré [13, 62]. La sombra de la rejilla está distorsionada debido a la forma del objeto, por lo que se pueden observar franjas de Moiré por las rejillas distorsionadas y las de referencia.

3.3. Función Objetivo

La demodulación de interferogramas constituye un problema de ajuste de datos, cuando las franjas se cierran y existe la presencia de ruido se convierte en un problema de mayor complejidad. Bajo este supuesto surge las condiciones necesarias para abordar este problema mediante metaheurísticas.

Como se mencionó en el capítulo 1 un patrón de franjas es el producto de la superposición de dos haces proveniente de la misma fuente, que a su vez deben tener la misma polarización y la diferencia de camino óptico entre ambos no debe exceder la distancia permitida. El modelo matemático que representa un interferograma es:

$$I(x,y) = a(x,y) + b(x,y)cos(w_x x + w_y y + \phi(x,y)) + n(x,y), \tag{3.1}$$

donde $\phi(x,y)$ representa el término de fase, el cual representa la cantidad física que se desea medir y da origen a este tipo de problema.

En Investigación de Operaciones cuando los problemas que se abordan no pueden resolverse en tiempo polinomial (problemas de tipo P), se convierten en problemas de tiempo

polinomial no determinista (problemas NP) en el cual la única solución en apariencia es resolverlos mediante una búsqueda exhaustiva. Haciendo una analogía con el problema en cuestión, al surgir inconvenientes que las técnicas convencionales no pueden resolver, el problema de demodulación de fase se puede clasificar como un problema NP-completo.

En este trabajo presentamos un modelo de resolución mediante la implementación de algoritmos evolutivos. La idea fundamental del modelo es hacer un ajuste polinomial de la fase óptica la cual puede representarse mediante la siguiente expresión:

$$\phi(a,x,y) = a_0 + a_1x + a_2y + a_3xy + a_4x^2 + a_5y^2 + \ldots + a_{\frac{(n+1)(n+2)}{2}-1}y^n, \tag{3.2}$$

donde x e y representa la posición de un pixel dentro de la imagen a demodular y a representa el vector de coeficientes que representa una posible aproximación del termino de fase. Para lograr una mayor automatización del algoritmo y ampliar la aproximación a polinomios de grado n-ésimo, se implementó un método que evalua un punto a partir del vector de coeficientes sin importar el grado del mismo. La única restricción de este algoritmo es el orden en el cual se colocan los coeficientes que es de suma importancia y en caso de no existir se completa con valor cero. Ver apéndices A.

Como todo modelo de optimización necesita de una función de costo, para nuestro caso abordamos este problema como un enfoque de minimización donde el objetivo es encontrar una imagen lo más parecido al interferograma y con un perfil suave (criterio de penalización de diferencias de fase). Esta función de costo o aptitud está definida como:

$$FO = \sum_{y=1}^{R}\sum_{x=1}^{C}\Bigg\{ \Big(I(x,y) - (127 - 128\cos[\phi(a,x,y)]) \Big)^2 - m\Big[\big(\phi(a,x,y) - \phi(a,x+1,y+1) \big)^2 + \big(\phi(a,x+1,y) - \phi(a,x,y+1) \big)^2 \Big] \Bigg\}, \tag{3.3}$$

donde $I(x,y)$ representa la imagen de entrada y m controla el nivel de suavidad de la imagen a demodular. La Ecuación 3.3 representa una diferencia entre la imagen a demodular y una imagen simulada que presenta un perfil cosenoidal, donde los valores de fondo $a(x,y)$ y amplitud $b(x,y)$ toman valores iguales a 127 y 128 respectivamente debido a que la función coseno está definida en el rango de $[-1,1]$ de tal forma que considerando imágenes con pixeles de tamaño de 1 byte, se deben cubrir los valores de 0 a 255, y para lograrlo se usan estas constantes.

3.4. Rango de búsqueda de los coeficientes

Un factor importante en todo algoritmo de optimización es el espacio de búsqueda, de ahí la importancia de definir el rango de búsqueda para cada coeficiente a_i de acuerdo al conocimiento a-priori del problema a resolver. Conociendo que la ecuación que modela un patrón de interferencia tiene un perfil cosenoidal el cual indica que entre cada máximo o mínimo de la función coseno existe 2π radianes, es decir, en cada intervalo de 2π radianes hay una franja blanca y una negra (Ver Figura 3.4).

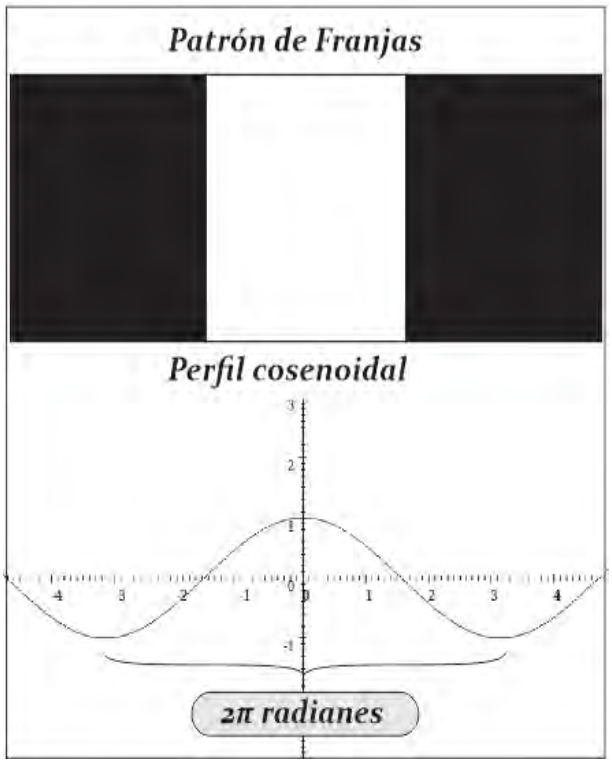


Figura 3.4: Patrón de franjas y perfil cosenoidal.

El primer paso para obtener los rangos de los intervalos consiste en contar el número total de franjas (N) en el interferogramas, para ello se puede utilizar algún algoritmo de conteo de franjas o en algunos casos donde no sea tan complicado se puede realizar de forma manual. Una vez encontrado el número total de franjas igualamos el término de fase al total de radianes presente en el interferograma (Ecuación 3.4)

$$a_0 + a_1x + a_2y + a_3xy + a_4x^2 + a_5y^2 + \ldots + a_{\frac{(n+1)(n+2)}{2}-1}y^n = 2\pi N, \tag{3.4}$$

Cada término a_i contribuye al total de radianes dentro del patrón de franjas, despejando cada término a_i de la Ecuación 3.4 y asumiendo los valores máximos de las variables x e y encontramos el intervalo de cada coeficiente. A continuación mostramos el despeje matemático realizado para encontrar los intervalos de búsqueda de cada coeficiente.

$$a_0 = 2\pi N \quad \rightarrow \quad a_0 \in [-2\pi N, 2\pi N]\,, \tag{3.5}$$

$$a_1 = \frac{2\pi N}{X_{max}} \quad \rightarrow \quad a_1 \in \left[-\frac{2\pi N}{X_{max}}, \frac{2\pi N}{X_{max}}\right], \tag{3.6}$$

$$a_2 = \frac{2\pi N}{Y_{max}} \quad \rightarrow \quad a_2 \in \left[-\frac{2\pi N}{Y_{max}}, \frac{2\pi N}{Y_{max}}\right], \tag{3.7}$$

$$a_3 = \frac{2\pi N}{X_{max}Y_{max}} \quad \rightarrow \quad a_3 \in \left[-\frac{2\pi N}{X_{max}Y_{max}}, \frac{2\pi N}{X_{max}Y_{max}}\right], \tag{3.8}$$

$$a_4 = \frac{2\pi N}{(X_{max})^2} \quad \rightarrow \quad a_4 \in \left[-\frac{2\pi N}{(X_{max})^2}, \frac{2\pi N}{(X_{max})^2}\right], \tag{3.9}$$

$$a_5 = \frac{2\pi N}{(Y_{max})^2} \quad \rightarrow \quad a_5 \in \left[-\frac{2\pi N}{(Y_{max})^2}, \frac{2\pi N}{(Y_{max})^2}\right], \tag{3.10}$$

$$\vdots \qquad\qquad\qquad\qquad \vdots$$

$$a_{\frac{(n+1)(n+2)}{2}-1} = \frac{2\pi N}{(Y_{max})^n} \quad \rightarrow \quad a_n \in \left[-\frac{2\pi N}{(Y_{max})^n}, \frac{2\pi N}{(Y_{max})^n}\right]. \tag{3.11}$$

3.5. Consideraciones Parciales

En este capítulo vimos lo referente a la modelación matemática del problema de demodulación de patrones de franjas, así como los principales aspectos a tener en cuenta cuando se quiere modelar como un problema de optimización. Recordando lo anterior podemos considerar este problema como un problema de ajuste de datos, por lo cual fue necesario definir la función de aptitud que caracteriza una posible solución. Además se tuvo en cuenta el conocimiento a *priori* para definir el espacio de búsqueda y de esta forma eliminar un conjunto de soluciones que no satisfacen al problema.

Capítulo 4

Consideraciones durante el diseño

Existen aspectos imprescindibles a la hora de aplicar un algoritmo evolutivo, la intensidad del proceso de búsqueda, la frecuencia de aplicación y la profundidad entran dentro de estos importantes aspectos en el diseño de un Algoritmo de Optimización. Luego de un minucioso estudio de la bibliografía y partiendo de la experiencia acumulada por varios creadores, a continuaciónn mostramos nuestras variantes de aplicación de los diferentes modelos utilizados en este trabajo.

Estos modelos de optimización fueron implementados en la plataforma de Java Standard Edition 8. Una de las razones principales de la elección de este lenguaje de programación es que Java constituye un lenguaje de programación de propósito general, concurrente y orientado a objetos, que fue diseñado específicamente para tener tan pocas dependencias de implementación como fuera posible [63].

Su objetivo fundamental es permitir que los desarrolladores de aplicaciones escriban el programa una vez y lo ejecuten en cualquier dispositivo, lo que quiere decir que el código que es ejecutado en una plataforma no tiene que ser recompilado para correr en otra. Java es, a partir de 2012, uno de los lenguajes de programación más populares en uso.

En conclusión, la programación orientada a objetos dispone de características muy interesantes como la herencia, el polimorfismo, la abstracción y otros elementos que sobre el papel son todos ellos deseables. Sin embargo, una implementación real supone un compromiso entre un buen diseño y una ejecución eficiente. El concepto de herencia por ejemplo a niveles tan inferiores como los alelos, genes e individuos puede repercutir fácilmente en duplicar o triplicar el espacio de memoria necesario. Igualmente, la abstracción y continua creación/copia/destrucción de instancias temporales de las clases diseñadas pueden hacer muy lento el sistema final.

4.0.1. Codificación binaria

El código binario es el sistema numérico usado para la representación de textos, o procesadores de instrucciones de computadora. En informática y telecomunicaciones, el código binario se utiliza con variados métodos de codificación de datos, tales como cadenas de caracteres, o cadenas de bits.

En la actualidad el uso de esta codificación en las metaheurística se ha incrementado considerablemente por los resultados alcanzados. Un ejemplo son los Algoritmos Genéticos que desde su creación por Holland [35], fueron implementados utilizando este tipo de codificación. En este trabajo además de presentar un AG en codificaión binaria se hace una extensión al algoritmo de Recocido Simulado el cual presenta resultados prometedores para el problema de demodulación de fase.

Tanto en el algoritmo de RS como en los AGs cada coeficiente del polinomio o gen para el caso de los AGs fueron codificados como una cadena de bit $a_i = [*, *, \ldots, l]$ con $* = [0, 1]$ y l_i indica la longitud de la cadena, cada coeficiente del polinomio está representado por una cadena binaria de longitud l, para transformar la cadena binaria a un valor real se utiliza la siguiente función de transformación:

$$a_i = Lim\_Inf_i + \left(\frac{Lim\_Sup_i - Lim\_Inf_i}{2^{l_i} - 1} \times N\_Entero\_Bin \right), \tag{4.1}$$

donde Lim_Inf_i y Lim_Sup_i representa el límite superior e inferior del rango de búsqueda del coeficiente a_i, N_Entero_Bin indica el valor entero correspondiente a la cadena binaria y esta dada por:

$$N\_Entero\_Bin = \sum_{j=0}^{l_i - 1} a_i (j)^j. \tag{4.2}$$

Con el uso de estas transformaciones garantizamos que cada coeficiente a_i este dentro del rango de valores reales permitidos ($a_i \in [Lim\_Inf_i, Lim\_Sup_i]$). En general el conjunto de cadenas binarias que modela el polinomio de ajuste representa una solución del algoritmo del RS o un cromosoma en el caso de los AGs (Ver Figura 4.1).

[11011 01011 10110 11101 10011]

Figura 4.1: Cadena binaria que modela al polinomio de ajuste.

Suponiendo que la cadena de ejemplo de la Figura 4.1 modela un polinomio de grado 2 y el valor de $l = 5$ el término de fase quedaría modelado de la siguiente manera:

$$\phi(a, x, y) = \overbrace{a_0}^{11011} + \overbrace{a_1}^{11011} x + \overbrace{a_1}^{01011} y + \overbrace{a_1}^{10110} xy + \overbrace{a_1}^{11101} x^2 + \overbrace{a_1}^{10011} y^2 \tag{4.3}$$

4.0.2. Codificación Gray

En la mayoría de los algoritmos evolutivos se requiere que una solución o un individuo sean lo más parecido como sea posible, es decir, un cambio pequeño en su codificación deberá de indicar igualmente una diminuta variación en la función de ajuste. En este trabajo implementamos un esquema de codificación Gray [64] como una variante binaria con el objetivo de evitar saltos bruscos en la vecindad de una solución.

El código Gray fue diseñado originalmente para prevenir señales ilegales de los switches electromecánicos; actualmente es usado para facilitar la corrección de errores en los sistemas de comunicaciones, tales como algunos sistemas de televisión por cable y la televisión digital terrestre.

El objetivo fundamental de la codificación Gray es asegurar que cualquier par de puntos adyacentes en el espacio de búsqueda difiera por sólo un bit en el espacio de representación [65]. Esta propiedad de adyacencia es de gran utilidad para mejorar el desempeño de muchas metaheurísticas que utilizan variables enteras, reales o complejas. La Tabla 4.1 muestra una lista de equivalencia entre valores decimales, binarios y en codificación Gray.

Decimal	**Binario**	**Gray**
0	000	000
1	001	001
2	010	011
3	011	010
4	011	100
5	101	101
6	011	110
7	011	111

Tabla 4.1: Relación entre la notación decimal, binaria y gray.

Para realizar una conversión de código binario a Gray, deben de usarse operadores OR exclusivos (XOR), una forma de implementar esta conversión sería como sigue [65]:

$$G_i = B_{i+1} \oplus B_i, \qquad i = l-1, \ldots, 0. \tag{4.4}$$

A continuación mostramos un pseudocódigo para la conversión de notación binaria a gray:

Algoritmo 7 Conversión de Binario a Gray

```
1: procedure BINARIO A GRAY( bin: array )
2:     gray[0] = XOR(0, bin[0])
3:     for i = 1; i ≤ bin.length; i++ do
4:         gray[i] = XOR(bin[i − 1], bin[i])
5:     end for
6:     Retornar gray
7: end procedure
```

El proceso inverso para transformar una codificación gray a binaria es necesario aplicar la siguiente transformación:

$$B_i = B_{i+1} \oplus G_i, \qquad i = l-1, \ldots, 0. \tag{4.5}$$

El siguiente algoritmo muestra un seudocódigo para obtener la conversión de notación gray a binario.

Algoritmo 8 Conversión de Gray a Binario

```
1: procedure GRAY A BINARIO( gray: array )
2:     bin[0] = XOR(0, gray[0])
3:     for i = 1; i ≤ gray.length; i++ do
4:         bin[i] = XOR(bin[i − 1], gray[i])
5:     end for
6:     Retornar bin
7: end procedure
```

4.0.3. Codificación númerica

En este tipo de codificación se utilizan cadenas de números que representan un número en una secuencia. Se utiliza en problemas en los que hay que ordenar algo, donde resulta muy útil. En este proyecto usamos esta codificación en una variante del algoritmo de RS y en el método de Optimización basada en Mallas Variables. Igual que en la codificación binaria cada coeficiente $a_i \in [Lim\_Inf_i, Lim\_Sup_i]$.

Aunque la codificación binaria y sus variantes es una de las notificaciones más frecuentes para resolver muchos problemas de optimización, tiene como desventaja su cuantización. Cuando se requiere resolver un problema en donde las variables son continuas y se quiere saber con toda la precisión la solución es necesario implementar este tipo de codificación.

Si se utilizara una codificación binaria, dado que cada variable requiera varios bits para ser representada, al crecer el número de variables aumenta el tamaño de la cedena de bits. La codificación continua tiene la ventaja que la implementación es más rápida que la binaria, pues las soluciones no requieren de ser decodificados antes de evaluar la función de aptitud.

4.1. Recocido Simulado

Las relaciones entre clases es un aspecto a tener en cuenta a la hora de realizar cualquier implementación haciendo uso de un lenguaje de programación orientado a objetos. Para la implementación en codificación real se implementaron 3 clases principales. La clase *Solution_Node* representa una posible solución, es decir es una aproximación polinomial del término de fase.

La clase *Algorithm_Simulate_Analing* es la clase principal encargada de llevar a cabo el funcionamiento general de la metaheurística y a su vez esta relacionado con la clase *Solution_Node* porque incluye una solución que será modificada constantemente hasta obtener la mejor solución. Por último la clase *Auxiliar* que como su nombre lo indica es una asociación que nos ayuda a facilitar el trabajo del algoritmo, ya que es la encargada de todos los métodos suplementarios. La Figura 4.2 esquematiza las relaciones entre las principales clases que intervienen la implementación real:

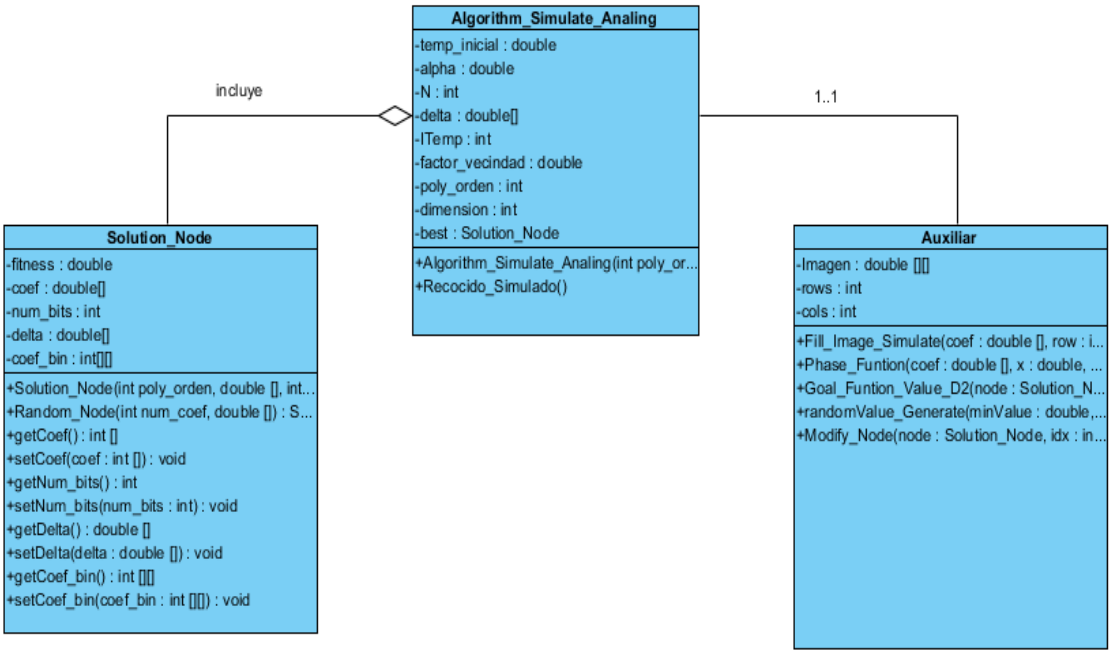


Figura 4.2: Diagrama de clases para el algoritmo de RS real.

Al igual que la implementación real la codificación binaria presenta las mismas clases con la única diferencia de la inclusión de una nueva clase nombrada *Binary*. Esta clase es la encargada de todas las operaciones a nivel de bits que realiza el algoritmo. La Figura 4.3 representa las relaciones entre clases para la implementación en codificación binaria.

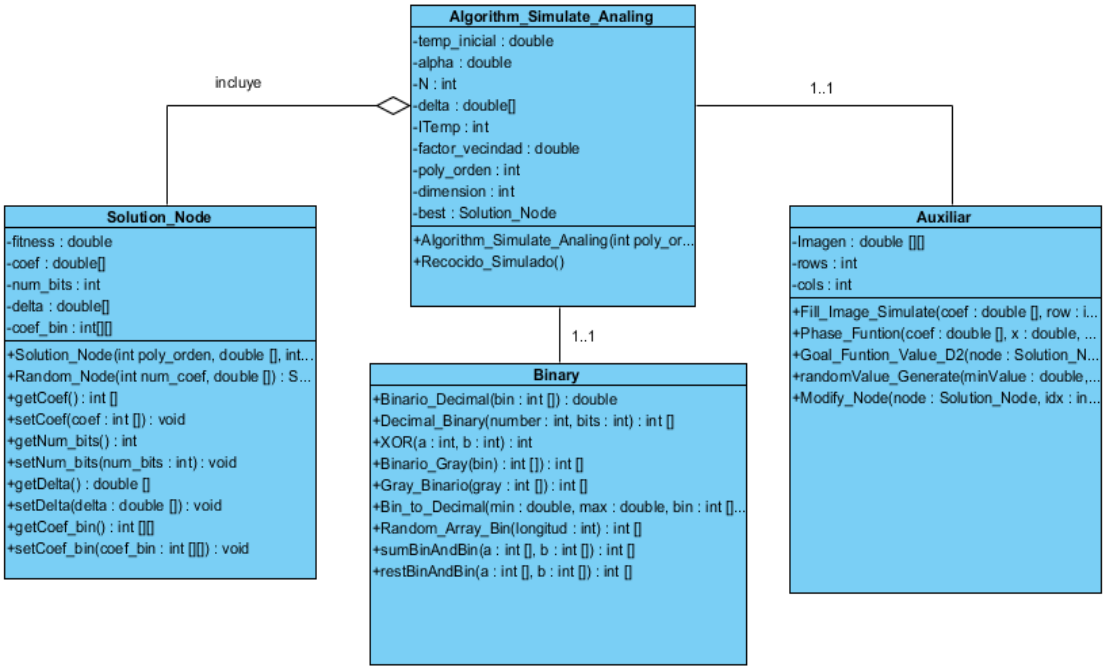


Figura 4.3: Diagrama de clases para el algoritmo de RS en codificación binaria.

Esquema de enfriamiento

Un aspecto importante a la hora de implementar un algoritmo de Recocido Simulado lo constituye el esquema de enfriamiento. Este punto involucra la necesidad de definir un valor de temperatura inicial que sea lo suficientemente robusto, es decir, que la solución alcanzada no dependa de la de partida. Esto se traduce en una temperatura inicial suficientemente alta que permita movimientos erráticos al principio del proceso.

El RS requiere que la temperatura disminuya conforme avanza la búsqueda. Existen varios estudios enfocados en esquemas de enfriamiento entre los que se destaca [44]. En este trabajo analizamos diferentes esquema de enfriamiento para comparar el desempeño del RS para cada variante.

El primer enfoque analizado fue el descenso lineal representado por la Ecuación 2.2. Este enfoque se puede ajustar para hacer que la temperatura converja a cero o a un valor cercano a cero en el número de itereaciones deseado. La Figura 4.4 muestra un gráfico donde se esquematiza el descenso de la temperatura mediante esta técnica.

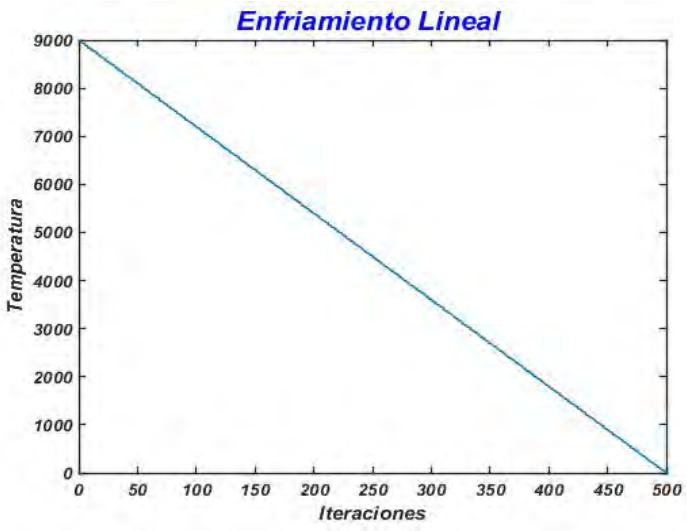


Figura 4.4: Representación gráfica de un esquema de enfriamiento lineal.

Esta técnica es sencilla de implementar y tiene ventaja de diseñarla de tal manera que converja a un valor determinado en un número de iteraciones deseadas. Por otro lado esta función hace prevalecer un enfoque aleatorio durante gran parte del proceso de búsqueda ya que su descenso depende de la pendiente de la recta y generalmente cuando se desea una cantidad grande de iteraciones el descenso se hace cada vez más lento.

El esquema de enfriamiento geométrico es uno de los más comunes e implementados para este tipo de algoritmos. Al contrario del método anterior, este esquema es muy propenso a llevar a cabo una búsqueda aleatoria en sus primeras iteraciones debido a las altas temperaturas, pero de manera acelerada desciende la temperatura haciendo la búsqueda más elitista. Este esquema tiene la desventaja que hay que estar configurando el valor inicial de

la temperatura para hacerla descender a una mínima magnitud en un determinado número de iteraciones. La siguiente figura esquematiza su comportamiento.

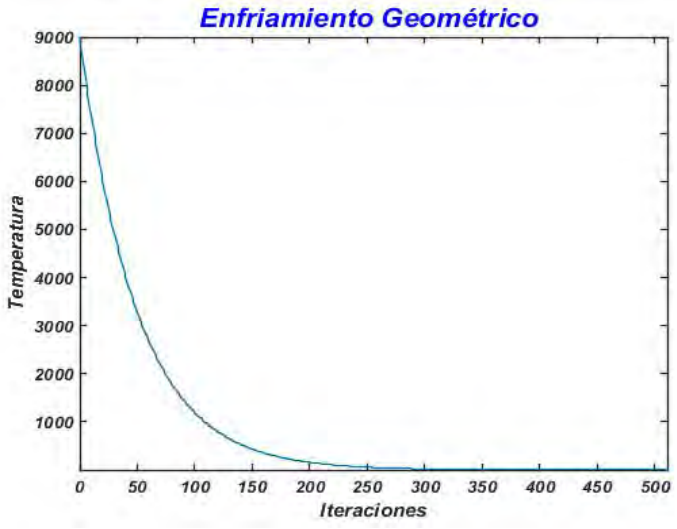


Figura 4.5: Representación gráfica de un esquema de enfriamiento geométrico.

El último enfoque fue una función exponencial donde trasladamos el máximo de la función al centro para comenzar con el máximo valor de la temperatura en la primera iteración. Con esta función se garantiza que la temperatura disminuya gradualmente con el número de iteraciones, logrando que la temperatura llegue a su ínfimo valor en la última iteración y su descenso sea gradual, tal y como se recomienda en la literatura. La siguiente ecuación muestra la ecuación que determina este comportamiento:

$$T(i) = e^{-\alpha * i^2 + \beta * i + log(T_{Inicial})}, \tag{4.6}$$

donde i representa la iteración actual, N el número de iteraciones, $T_{Inicial}$ y T_{Final} indica la temperatura inicial y final del modelo. Los términos α y β están relacionados mediante la siguiente expresión:

$$\beta = \frac{log(\frac{10 * T_{Final}}{T_{Inicial}})}{N} + \alpha * N, \tag{4.7}$$

donde α es un valor muy pequeño que modifica la velocidad de descenso de la temperatura. Tomando $N = 1000$, $T_{Inicial} = 90000$, $T_{Final} = 1$ y $\alpha = 10^{-6}$la ecuación 4.6 tiene el siguiente desempeño el cual se muestra en la Figura 4.6.

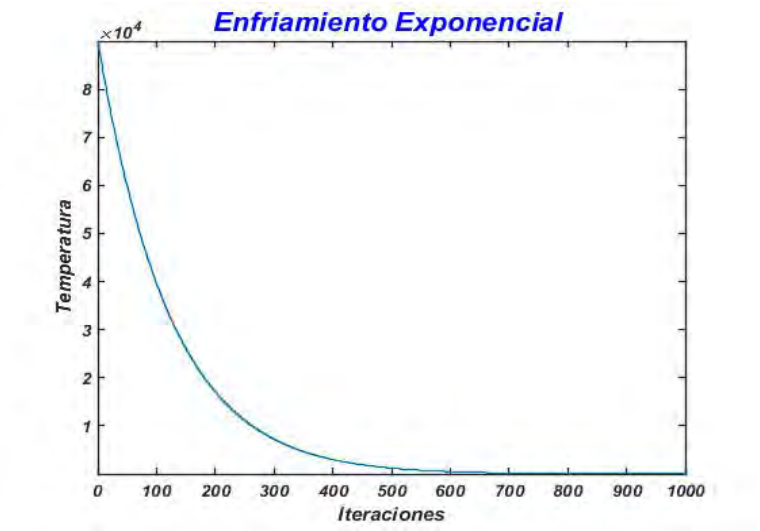


Figura 4.6: Representación gráfica de un esquema de enfriamiento exponencial.

Haciendo una comparación entre los 3 esquemas la función más flexible que permite mayor ajuste de la pendiente de la curva es el modelo exponecial. A continucación se muestra una gráfica de comparación donde se muestra el comportamiento para un total de 500 iteraciones aproximadamente. En la Figura se evidencia claramente que la exponencial permite obtener un mejor abarcamiento del espacio debido a la facilidad de manipulación del parámetro α, debido a que los gráficos representan el descenso más lento que pudiera implementar dichas funciones. La ventaja del modelo geométrico y exponencial sobre el lineal es que este último es estático una vez definido la pendiente de la recta este no cambia su comportamiento.

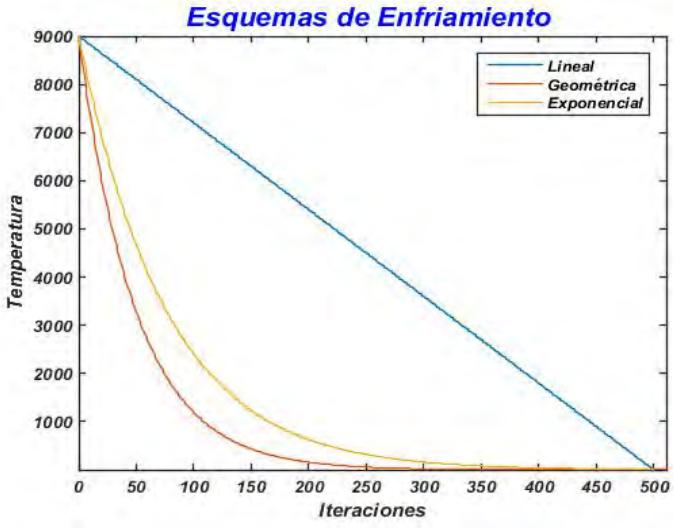


Figura 4.7: Comparación entre los esquemas de enfriamientos implementados.

Vecindad

La forma de seleccionar una solución vecina en el algoritmo de RS es un factor importante en el desempeño del algoritmo. Para la variante de codificación real del algoritmo del RS se

escoge aleatoriamente el coeficiente que será modificado, después se seleccioná un porcentaje del intervalo de búsqueda permisible y se generó un número aleatorio dentro del rango seleccionado y se le adicionó al valor del coeficiente escogido.

Este método de vecindad es una variante del algoritmo de descenso de Gradiente si se escoge un intervalo pequeño, por tal razón es de suma importancia la elección del intervalo, es decir para funciones transcendentales cuando la forma es más compleja es necesario aumentar el rango para lograr saltos más grande y aumentar el espacio de búsqueda. El siguiente algoritmo muestra la implementación de este método de vecindad.

Algoritmo 9 Obtiene un solución Vecina para una codificación real

1: **procedure** VECINDAD($a : array,\ delta : array,\ factor_v : double$)
2: $i = Random(0,\ a.length)$
3: $a_i = a_i + Random(-factor_v * delta_i,\ factor_v * delta_i)$
4: Retornar a
5: **end procedure**

La variante más utilizada para obtener una solución vecina en una codificación binaria es seleccionar al azar un bit y modificarlo, es decir, se niega el bit para obtener una solución que pertenece a su vecindad. El Algoritmo 10 describe un pseudocódigo para encontrar una solución cercana:

Algoritmo 10 Obtiene un solución Vecina para una codificación Binaria

1: **procedure** VECINDAD($a : arraybinario$)
2: $i = Random(0,\ a.length)$
3: $a_i = !a_i$
4: Retornar a
5: **end procedure**

Otra forma de obtener una solución vecina para el caso binario utilizada en este trabajo será descrita a continuación. Primeramente se selecciona aleatoriamente un coeficiente binario y después se selecciona un bit con cierta probabilidad. Para obtener este resultado se diseñó una función de probabilidad que puede describir cualquier comportamiento. Por ejemplo si se escoge una recta con pendiente positiva para privilegiar a los bit menos significativos y tenemos un total de 5 bits los valores de probabilidad estan dados por:

$$\frac{n_0}{\sum n_i^2},\ \frac{n_1}{\sum n_i^2},\ \frac{n_2}{\sum n_i^2}, \ldots,\ \frac{n_m}{\sum n_i^2}$$

donde los valores n_i están dado por la ecuación $n_i = 0.5x$. La siguiente tabla muestra un ejemplo para un coeficiente con 5 bits.

Este método garantiza que la probabilidad total sea igua a la unidad y además se puede simular cualquier otra función en dependencia de la implementación. El Algoritmo 11 muestra un pseudocódigo del método:

	1	**2**	**3**	**4**	**5**	$\sum n_i^2$
n_i	0.5	1.0	1.5	2.0	2.5	13.75
Probabilidad	0.01818	0.07272	0.1636	0.2909	0.4545	

Tabla 4.2: Ejemplo de probabilidad proporcionada a los bit para ser cambiados.

Algoritmo 11 Calcula el valor de probabilidad para ser cambiado de cada bit y encontrar un nuevo vecino

```
1: procedure VALOR_ESPERADO_BIT( n : arraya, arraybinario)
2:     n : vector que define el comportamiento de la función de probabilidad
3:     for i = 1; i ≤ n.length; i++ do
4:         sum = sum + n_i
5:     end for
6:     for i = 1; i ≤ n.length; i++ do
7:         prob_i = n_i / sum
8:     end for
9:     Retornar prob
10: end procedure
```

El tercer modelo utilizado para generar una solución vecina binaria es similar al método de vecindad implementado para la codificación real. Conociendo que el incremento entre dos valores consecutivos para un coeficiente codificado con n bits es:

$$\Delta x = \frac{b-a}{2^n} \tag{4.8}$$

donde a y b denota los límites del intervalo permisible para dicho coeficiente. Una vez conocido el incremento, si se desea generar un valor en cierto intervalo o en un porcentaje del intervalo original bajo el supuesto de la misma diferencia entre valores consecutivos se debe realizar las siguientes transformaciones a la Ecuación 4.8 para obtener el número de bits que nos permite representar dicho intervalo. Despejando la incognita n tenemos:

$$2^n = \frac{b-a}{\Delta x}. \tag{4.9}$$

Aplicando logaritmo base 2 a ambos miembros obtenemos el valor de n' para el nuevo intervalo

$$\begin{aligned} \log_2 2^{n'} &= \log_2 \left(\frac{factor_v(b-a)}{\Delta x} \right), \\ n' &= \log_2 \left(\frac{factor_v(b-a)}{\Delta x} \right). \end{aligned} \tag{4.10}$$

Generamos un número binario de n' bits (a_i')y completamos el resto con cero para igualar al número de bits del coeficiente. Si el valor generado es menor que $2^{n'-1}$ entonces el coeficiente se modificaria de esta manera:

$$a_i = Resta\_Binaria(a_i - a_i'), \tag{4.11}$$

en otro caso la transformación sería la siguiente:

$$a_i = Suma\_Binaria(a_i + a'_i). \tag{4.12}$$

Para el caso de algún desbordamiento por las operaciones de suma y resta se genera un nuevo coeficiente aleatoriamente. Para mayor información acerca de las funciones de *Suma_Binaria* y *Resta_Binaria* ver los apéndices B y C.

4.2. Algoritmos Genéticos

Tal como se introdujo, los AGs intentan imitar el proceso natural de evolución de los organismos biológicos. Para ello se considera que nuestro universo es el espacio de soluciones factibles del problema a tratar, en él existe un conjunto de individuos los cuales representan posibles soluciones al problema o aproximaciones. A cada individuo se le asigna un valor de aptitud, el cual indica que tan buena es la solución. Los individuos con mejor fitness tienen más oportunidades de "reproducirse" con otros individuos (propagar sus propiedades genéticas para generar mejores soluciones). La "reproducción" entre dos individuos produce nuevos con características heredadas de sus "padres". Los menos aptos serán los más propicios a desaparecer. Esta nueva generación contiene una mejor proporción de las características de los buenos individuos y en consecuencia, puede llevar consigo mejores resultados.

Hasta ahora se ha prestado poca atención a la calidad del software usado en el campo de la computación evolutiva. Existen numerosas implementaciones mal estructuradas, poco flexibles, y en general únicamente útiles para sus creadores. Muchos autores presentan resultados sin sugerir siquiera el tipo de implementación usada. En nuestro caso, y aunque no se trate de un objetivo primario, nos interesa discutir las ventajas innegables de usar clases de objetos. Esto nos lleva a la primera pregunta evidente: ¿qué clases deben existir y cómo podemos relacionarlas?

Es importante realizar una fase de diseño que nos permita comparar y realizar distintos modelos que nos ayude a generar un software de calidad. Aunque el objetivo principal no sea la implementación, se hace imprescindible discutir sobre el tema de implementación debido a la importancia de agilizar todo proceso evolutivo. Presentamos el diagrama de clases en la Figura 4.8.

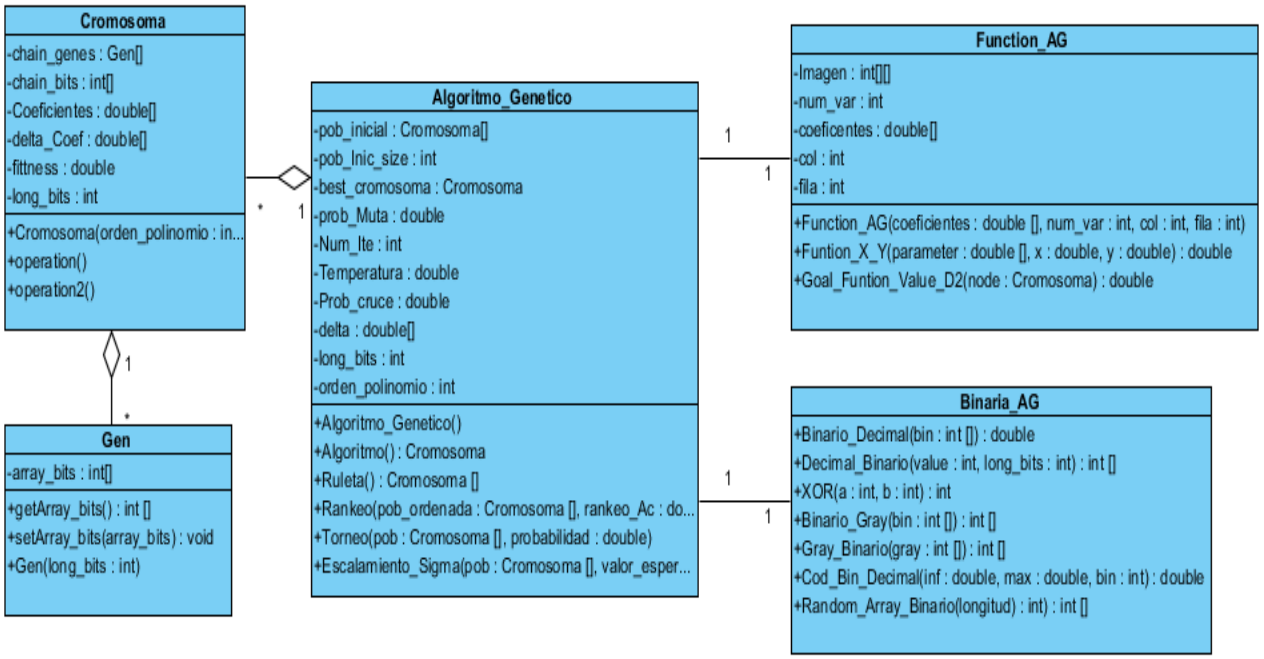


Figura 4.8: Diagrama de clases para un AG.

4.2.1. Operadores Genéticos

Para facilitar la descripción presentaremos primero los genotipos que consideramos y después el operador de selección y reemplazo en su versiones más populares. El operador de mutación que utilizaremos es unario y los operadores de cruce estudiados son binarios. Utilizaremos codificaciones en genotipo binario y real explorando así las capacidades de la caracterización ortodoxa de un AG. Nuestro interés reside en versiones canónicas de las técnicas secuenciales, aunque la extensión del estudio que abarca ciertos criterios de codificaciones como la longitud variable, constituyendo un punto de interés que motivan la necesidad de trabajar y que discutiremos en el capítulo de resultados experimentales.

Genotipo

La definición del genotipo es realizada por el usuario ya que depende estrictamente del problema a resolver y de la estructura que se quiera utilizar para tal resolución. El genotipo posee además de operaciones simples como los constructores y el calculo de la aptitud de la función objetivo. Además posee las operaciones para realizar el proceso de cruzamiento y mutación de estos, entre otras. Estas dos operaciones son particulares de cada genotipo las cuales se encargan de devolver genotipos válidos y realizar el chequeo de factibilidad en caso de que sea necesario.

El diseño de clase del genotipo asociado con el problema se muestra en la siguiente figura:

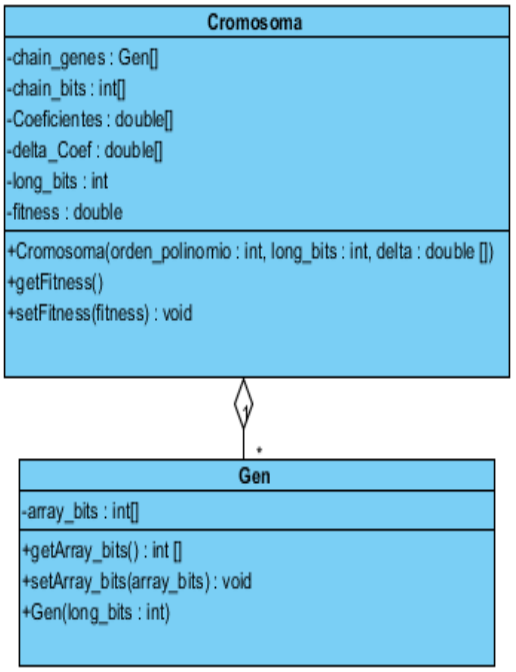


Figura 4.9: Representación binaria y real.

Como se mencionó en capítulos anteriores se utilizó el paradigma de programación orientado a objeto por las facilidades que ofrece el mismo. La figura anterior muestra ambas codificaciones separadas en dos clases y conectadas por una relación de agreagación entre ellas. La codificación binaria estaría representada por una lista de objetos de la clase *Gen* y este a su vez contendría un arreglo binario. El genotipo real sería similar al binario con la particularidad de que la clase *Gen* tendría como atributos un arreglo de valores reales. Ver Apéndices D y E.

4.2.1.1. Selección

La selección juega un papel importante en los algoritmos evolutivos secuenciales como, debido a que guía la búsqueda y provoca la convergencia de la población de individuos. Por tanto, representa el tan importante compromiso entre exploración y explotación. Ofreceremos en esta sección un resumen de los operadores de selección implementados en este trabajo.

De entre los operadores de selección proporcional, la implementación por ruleta es la más popular. Sin embargo, también es de mayor complejidad algorítmica y con tendencia a cometer errores estocásticos. Para implementar este operador de selección fue necesario la colaboración de varios métodos. Por ejemplo el Algoritmo 12 muestra la transformación realizada para convertir el AG en un problema de minimización, para ello fue necesario encontrar el peor individuo de la población y sustraer el valor de aptitud a toda la población; para evitar que el peor cromosoma tenga probabilidad cero se le adiciona una pequeño valor.

Algoritmo 12 Método que calcula las probabilidades acumulativa para un problema de minimización

```
1: procedure PROBABILIDAD ACUMULATIVA RULETA( poblacion : list Cromosoma)
2:     Cromosoma worst = Worst_Cromosoma(poblacion)  /*Función que devuelve el
   peor individuo */
3:     for (i = 1; i ≤ poblacion.length; i++) do
4:         prob_ac[i] = worst.getFitness − poblacion(i).getFitness + value
5:         sum_fittness+ = prob_ac[i]
6:     end for
7:     prob_ac[0] = prob_ac[0]/sum_fittness
8:     for (i = 1; i ≤ prob_ac.length; i++) do
9:         prob_ac[i] = prob_ac[i]/sum_fittness + prob_ac[i − 1]
10:    end for
11:    Retornar prob_ac
12: end procedure
```

Una vez calculada la probabilidad acumulativa para completar la selección por ruleta se selccionan dos individuos al azar para llevar a cabo el proceso de cruce. El Algoritmo 13 selecciona dos padres y garantiza que sean diferentes para evitar reproducir individuos iguales a sus progenitores.

Algoritmo 13 Método de selección por Ruleta

```
1: procedure SELECCIÓN( poblacion : list Cromosoma)
2:     prob_ac = ProbabilidadAcumulativaRuleta(poblacion) /*Algoritmo 12*/
3:     idx = 0
4:     while (idx < 2) do
5:         if (idx == 0) then
6:             padres[idx] = poblacion.get(Ruleta_Pos(prob_Ac))
7:             idx++
8:         else
9:             padres[idx++] = poblacion.get(Ruleta_Pos(prob_Ac))
10:            if (!padres[1].equals(padres[0])) then
11:                idx++
12:            end if
13:        end if
14:    end while
15:    Retornar: padres
16: end procedure
```

Otro método de selección utilizado es el conocido como selección por **ranking** el cual requiere de un algoritmo de ordenamiento para ordenar la población de Cromosomas a partir de su aptitud. Para realizar estas operaciones fue necesario trabajar con la Clase "Collections" de Java, la cual se utilizó para ordenar ArrayList (colección de datos que almacena los objetos de tipo Cromosomas). Para ordenarlo de forma ascendente, se tuvo que crear un

objeto de la clase Comparable (D)para que compare los elementos.

Esto se logra haciendo que la clase Cromosoma implemente la interfaz comparable "Comparable"de Java 7. Es válido destacar que las interfaces no son clases, sólo especifican requerimientos para la clase que las implemente. La diferencia entre las interfaces y las clases abstractas es que en las interfaces se especifica qué se debe hacer pero no su implementación. Serán las clases que implementen estas interfaces las que describan la lógica del comportamiento de los métodos, es decir, proporciona un mecanismo de encapsulación de los protocolos de los métodos sin forzar al usuario a utilizar la herencia.

Para ordenar el ArrayList o la población de Cromosomas en función de la aptitud o fitness, tenemos que sobreescribir el método "compareTo"de la interfaz Comparable. La Figura 4.10 muestra la implemantación donde retorna el valor de cero en caso que los cromosomas comparados sean iguales respecto a su aptitud, 1 en caso que sea mayor y -1 en caso contrario.

```
10
11      /**
12       * COMAPRA LOS FITTNESS
13       */
14      public int compareTo(Cromosoma crom) {
15          if (this.fittness == crom.getFittness()) {
16              return 0;
17          } else if (this.fittness > crom.getFittness()) {
18              return 1;
19          } else {
20              return -1;
21          }
22      }
23
```

Figura 4.10: Método *compareTo* de la interfaz Comparable.

Una vez ordenada la población se hace un ranqueo lineal dentro de un intervalo definido por el usuario. Por ejemplo si tenemos 5 Cromosomas y queremos generar el vector de probabilidad a partir de su posición obtenida después del proceso de ordenamiento y tomando un intervalo de escalamiento entre 0.5 y 1.5 obtenemos la siguiente gráfica que muestra la probabilidad acumulativa obtenida por cada individuo después del proceso de ranqueo.

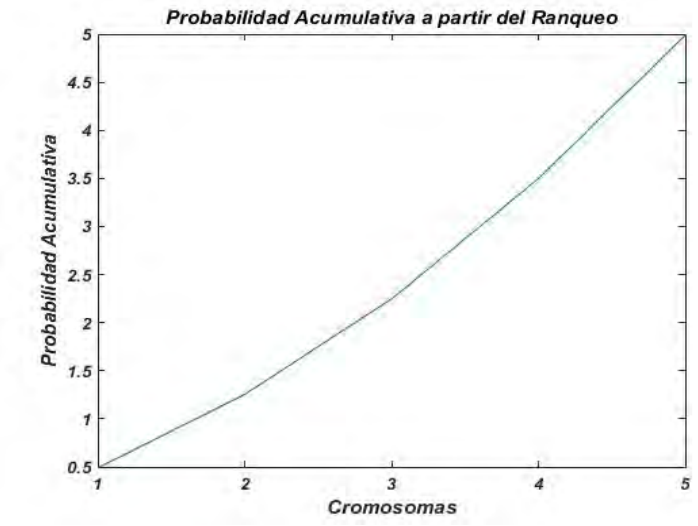


Figura 4.11: Probabilidad acumulativa obtenida por el método de Ranqueo.

Por último solo resta seleccionar aleatoriamente los cromosomas padres a partir de los valores de probabilidad de la gráfica anterior. Para mayor información el Apéndice F presenta una implementación del mismo.

La selección por **Torneo** implementada es una variante del método descrito en el Capítulo 2. Esta variante consiste en seleccionar dos individuos aleatoriamente y después realizar un torneo entre ellos con cierta probabilidad que decide cual seleccionar si el peor o el mejor caso. Esta variante evita la convergencia prematura y elimina en cierto grado el elitismo del algoritmo porque en la implementación clásica el peor individuo nunca tendrá la posibilidad de ser seleccionado. El Algoritmo 14 representa un seudocódigo de la variente implementada.

Algoritmo 14 Método de selección por Torneo

```
1: procedure TORNEO( poblacion : list Cromosoma, double : prob)
2:     while (idx < 2) do
3:         /*Se selecciona dos individuos aleatoriamente */
4:         Individuo1 = Random_Cromosoma(poblacion)
5:         Individuo2 = Random_Cromosoma(poblacion)
6:         /*Verificar que los individuos seleccionados sean diferentes */
7:         while (Individuo1 == Individuo1) do
8:             Individuo2 = Random_Cromosoma(poblacion)
9:         end while
10:        /*Ordeno los individuos segun su aptitud */
11:        if Individuo1.Fitness > Individuo2.Fitness then
12:            Temp = Individuo1
13:            Individuo1 = Individuo2
14:            Individuo2 = Temp
15:        end if
16:        if (Random(0, 1) < prob) then
17:            Individuo1 /*Recibe el mejor Individuo */
18:            idx + +
19:        else
20:            padres[idx + +] = Individuo2) /*Recibe el peor Individuo */
21:            idx + +
22:        end if
23:    end while
24:    Retornar: padres
25: end procedure
```

EL **Escalamiento Sigma** es otro método de selección estudiado en este trabajo de tesis. Para su desarrollo fue necesario de ciertas funciones adicionales como obtener el valor medio poblacional de función objetivo o la aptitud, así como la desviación estandar de la población (Apéndice G).

Este método garantiza que exista diversidad en la población durante todo el proceso evolutivo, ya que mantiene variable la desviación estandar de la población. Auxiliado de las funciones referenciadas en el Apéndice G, el siguiente algoritmo muestra la implementación de este método de selección.

Algoritmo 15 Método de selección por Escalamiento Sigma

1: **procedure** ESCALAMIENTO_SIGMA_NORMALIZADO($poblacion : list\ Cromosoma$)
2: $media = Media(poblacion)$
3: $DST = Desviacion\_Standar(poblacion, media)$
4: $worst = Worst\_Cromosoma(poblacion)$ /* Selecciona el peor Individuo*/
5: **for** $(i = 0; i < poblacion.size; i++)$ **do**
6: **if** $desv == 0$ **then**
7: $valor\_esperado(i) = 1$
8: $sum+ = valor\_esperado(i)$
9: **else**
10: $valor\_esperado(i) = \left|1 + \frac{worst.fitness - poblacion(i).fitness - media}{2 * DST}\right|$
11: $sum+ = valor\_esperado(i)$
12: **end if**
13: **end for**
14: $valor\_esperado(0) = \frac{valor\_esperado(0)}{sum}$
15: **for** $(i = 1; i < valor\_esperado.length; i++)$ **do**
16: $valor\_esperado(i) = valor\_esperado(i-1) + \frac{valor\_esperado(i)}{sum}$
17: **end for**
18: **Retornar:** $valor\_esperado$
19: **end procedure**

En este pseudocódigo es válido mencionar el por qué la búsqueda del el peor cromosoma de la población actual, necesario para convertir el AG en un algoritmo de maximización. El vector de salida que arroja el Algoritmo 15 representa la probabilidad de cada individuo la cual se usará en la selección mediante una ruleta generada aleatoriamente.

El último modelo de selección analizado en este trabajo fue la **seleccción de Boltzman**. Este algoritmo al igual que el recocido simulado utiliza una función de temperatura para mantener una presión de selección más o menos constante y dar oportunidad a individuos menos aptos. Las funciones de temperaturas utilizadas son las mismas vistas en la sección que aborda sobre al algoritmo del Recocido Simulado. Este método calcula el valor esperado o la probabilidad de cada individuo en la población que será utilizada en una selección tipo ruleta. A continuación se muestra el Algoritmo 16 el cual muestra un pseudocódigo de la implementación del mismo en cual se convierte en un problema de maximización para poder utilizar esta forma de selección.

Algoritmo 16 Método de selección de Boltzman

procedure BOLTZMAN_ACUMULATIVO($poblacion : list\ Cromosoma,\ doubleTemp$)
  $worst = Worst\_Cromosoma(poblacion)$ /* Selecciona el peor Individuo*/
  for $(i = 0; i < poblacion.size; i++)$ **do**
    $Boltzman(i) = e^{\frac{worst.fitness - poblacion(i) + 50}{Temp}}$
  end for
  $media = Media(Boltzman)$
  $Boltzman(0) = Boltzman(0)/media$
  for $(i = 1; i < poblacion.size; i++)$ **do**
    $Boltzman(0) = Boltzman(i-1) + Boltzman(i)/media$
  end for
  Retornar: $Boltzman$
end procedure

4.2.1.2. Cruzamiento

Uno de los operadores de cruzamiento más implementados en la literatura es el operador de N puntos para la codificación binaria. En este trabajo se utilizó este operador con los valores de N igual a 1 y 2 puntos respectivamente, porque al aumentar el número de puntos el algoritmo se hace más disruptivo el proceso de cruzamiento. Este algoritmo fue visto en el capítulo anterior con mayor profundidad por lo que en esta sección abordaremos más a fondos otras implementaciones.

Continuando con la codificación binaria presentamos en este trabajo una variante de la cruza uniforme y en esta ocación omitimos el uso de una máscara binaria por lo que en cada cruce se recorre completamente la cadena y un valor aleatorio decidirá para cada uno de los genes el padre que lo donará, es decir si es seleccionado se intercambian los bit entre padres y en caso contrario permanecen iguales. Para este estudio se utilizó como valor de probabilidad igual a 0.5 aunque se pudiera mostrar en estudios posteriores el desempeño del algoritmo al variar este valor. El Algoritmo muestra con mas detalles su implementación:

Algoritmo 17 Método de Cruzamiento Uniforme

procedure CRUCE_UNIFORME($padre1 : array,\ padre2 : array$)
  for $(i = 0; i < poblacion.size; i++)$ **do**
    if $Random() < 0.5$ **then** /*Si cumple intercambian los bits*/
      $temp(i) = padre1(i)$
      $padre1(i) = padre2(i)$
      $padre2(i) = temp$
    end if
  end for
  Retornar: $padre1,\ padre2$
end procedure

Para el caso de codificación real implementamos una variante del cruzamiento de N puntos en cual se selecciona aleatoriamente N puntos definido por el usuario lo cual es

equivalente a seleccionar N genes y efectuar el proceso de cruza entre ellos. Esta variante de cruzamiento se definió según la siguiente expresión:

$$a_i = mejor\_padre_i * \alpha + (1 - \alpha) * peor\_padre_i, \tag{4.13}$$

donde $\alpha \in [0, 1]$ y regularmente se utiliza un valor superior a 0.5 para beneficiar al mejor padre en cuestión de aptitud. A continuación un pseudocódigo donde muestra la impletación de esta modificación:

Algoritmo 18 Método de Cruzamiento de N Puntos para el caso real

```
1: procedure CRUCE_N_PUNTOS_CONTINUO( padre1 : array, padre2 : array, N : int, α : double)
2:     /*Se selecciona N genes Aleatoriamente y se guarda en un arreglo*/
3:     array : N_puntos = Vector_N_Puntos_Aleatorios(N)
4:     for (i = 0; i < N_puntos.size; i++) do
5:         if padre1.fitness < padre2.fitness then
6:             value = padre1(i) * α + (1 − α) * padre2(i)
7:             padre1(i) = value
8:             padre2(i) = value
9:         else
10:            value = padre2(i) * α + (1 − α) * padre1(i)
11:            padre1(i) = value
12:            padre2(i) = value
13:        end if
14:    end for
15:    Retornar: padre1, padre2
16: end procedure
```

El último operador de cruce implementado fue el conocido como cruce geométrico. En esta variante al igual que en la anetrior se selecciona N genes y se realiza un cruce geométrico entre los genes correspondientes. El Algoritmo 19 muestra su implementación:

Algoritmo 19 Método de Cruzamiento Geométrico de N Puntos para el caso real

```
1: procedure CRUCE_GEOMETRICO_N_PUNTOS_CONTINUO( padre1 : array, padre2 : array, N : int)
2:     /*Se selecciona N genes Aleatoriamente y se guarda en un arreglo*/
3:     array : N_puntos = Vector_N_Puntos_Aleatorios(N)
4:     for (i = 0; i < N_puntos.size; i++) do
5:         value = √(padre1(i) * padre2(i))
6:         padre1(i) = value
7:         padre2(i) = value
8:     end for
9:     Retornar: padre1, padre2
10: end procedure
```

4.2.1.3. Mutación

El operador de mutación ocurre con una probabilidad muy baja debido a que el objetivo de este operador es cambiar de valor como máximo a un alelo por cromosomas. Este operador se introdujo con la finalidad de evitar que el algoritmo caiga en mínimos locales y de esta forma mantener la diversidad de la población.

El algoritmo consiste en concatenar todas las cadenas binarias de los cromosomas de la población actual para evitar el problema de que el gen o bit seleccionado exceda el tamaño del cromosoma ese desplazamiento es utilizado en el siguiente cromosoma, el algoritmo que selecciona los indices que serán mutados se rige por la siguiente expresión:

$$idx = 1 - \left[\frac{ln(1-u)}{ln(1-Pm)}\right], \tag{4.14}$$

donde $u \in [0, 1]$ representa un valor generado aleatoriamente, idx indica el índice del gen que será mutado después del proceso de concatenación de la población, Pm la probabilidad de muta definida por el usuario. Este proceso es repetido secuencialmente y en cada iteración el valor de idx se actualiza añadiendo el valor anterior y el nuevo valor genarado a partir de la ecuación 4.14. Este proceso es repetido hasta que idx supere la longitud máxima de la cadena obtenida a través del proceso de concatenación.

En la mayoría de los casos la probabilidad de muta (Pm) es representada por un valor fijo; en este estudio implementamos una función de temperatura donde se hace variar en dependencia de la iteración actual. La probabilidad Pm se debe inicial con un valor relativamente alto dentro del rango permisible es decir un valor que ronde sobre los 0.05 unidades y se va disminuyendo gradualmente con la siguiente ecuación:

$$Pm_i = \frac{Pm_0 - Pm_f}{i}, \tag{4.15}$$

donde Pm_0 y Pm_f representa la probabilidad inicial y final definidas por el usuario, e i indica la iteración actual. Esta ecuación tiene el objetivo de ayudar a la convergencia de la población y asi garantizar una búsqueda aleatoria en las primeras iteraciones.

4.3. Optimización Basada en Mallas Variables

La Optimización Basada en Mallas Variables (VMO) es una meta heurística poblacional con características evolutivas donde un conjunto de nodos que representan soluciones potenciales a un problema de optimización, forman una malla (población) que crece dinámicamente y se desplaza por el espacio de búsqueda (evoluciona).

El proceso de generación de nodos en cada ciclo comprende los pasos siguientes:

- ✓ Generación de la malla inicial.
- ✓ Generación de nodos en dirección a los extremos locales.
- ✓ Generación de nodos en dirección al extremo global.

✓ Generación de nodos a partir de las fronteras de la malla.

El método incluye los siguientes parámetros:

✓ Cantidad de nodos de la malla inicial (N_{IM}).

✓ Cantidad máxima de nodos de la malla en cada ciclo (N_{TM} , donde $3 * N_{IM} \leq N_{TM}$).

✓ Tamaño de la vecindad (k).

✓ Condición de parada o número de evaluaciones (M).

4.3.1. Proceso de expansión

El algoritmo realiza un proceso de expansión mediante el cual la población se traslada a través del espacio de soluciones. Esta operación se realiza siguiendo los pasos descritos a continuación:

Generación aleatoria de la malla inicial: Para cada uno de los nodos de la malla inicial se genera aleatoriamente cada dimensión con un valor real definido dentro del espacio solución.

Generación de nodos en dirección a los extremos locales: Este modelo de exploración se basa en llevar a cabo un análisis de las vecindades para cada uno de los nodos de la malla. La vecindad de un nodo (n_i) esta definida por los k vecinos más cercanos al nodo actual. Esta vecindad puede ser obtenida a partir de una función de distancia, para nuestro trabajo se utilizó como función la distancia euclidiana definida por:

$$D_{Euclidiana}(n_1, n_2) = \sqrt{\sum_{j=1}^{M} (x_j^1 - x_j^2)^2} \tag{4.16}$$

El mejor nodo dentro de la vecindad es seleccionado como extremo local n_i^*. Para generar un nuevo nodo es necesario que el extremo local n_i^* sea diferente al nodo actual n_i, es decir que posea mejor valor de la función objetivo que el nodo n_i. Los nuevos nodos (n_z) son calculados usando la función definida por 4.17:

$$n_z = F(n_i, n_i^*, Pr(n_i, n_i^*)) \tag{4.17}$$

donde Pr representa la relación entre la aptitud del nodo actual y su extremo local. Este factor se calcula por la Ecuación 4.18y está definido en el intervalo de $[0, 1]$.

$$Pr(n_i, n_i^*) = \frac{1}{1 + |fitness(n_i) - fitness(n_i^*)|} \tag{4.18}$$

La función F definida en la Ecuación 4.17 genera un nodo n_z cuyas componentez x_j^z pueden ser calculadas como (Apéndice H):

$$x_j^z = \begin{cases} \overline{m_j} & \text{si } |\overline{m_j} - x_j^{i^*}| > \xi_j \quad \& \quad U[0,1] \leq Pr(n_i, n_i^*) \\ x_j^{i^*} + U[-\xi_j, \xi_j] & \text{si } |\overline{m_j} - x_j^{i^*}| \leq \xi_j \\ U\left[x_j^i, \overline{m_j}\right] & \text{en otro caso} \end{cases} \tag{4.19}$$

donde $\overline{m_j} = promedio(x_j^i, x_j^{i^*})$, $U[u,v]$ denota un valor aleatorio en el intervalo $[u,v]$, y ξ_j define el distancia mínima permitida para cada componente y es calculada de la siguiente manera (Ecuación 4.20):

$$\xi_j = \begin{cases} \frac{rango(a_j,b_j)}{4} & \text{si } \ c < 15\,\%C, \\ \frac{rango(a_j,b_j)}{8} & \text{si } \ 15\,\%C \leq c < 30\,\%C, \\ \frac{rango(a_j,b_j)}{16} & \text{si } \ 30\,\%C \leq c < 60\,\%C, \\ \frac{rango(a_j,b_j)}{50} & \text{si } \ 60\,\%C \leq c < 80\,\%C, \\ \frac{rango(a_j,b_j)}{100} & \text{si } \ c \geq 80\,\%C. \end{cases} \tag{4.20}$$

donde C y c denotan el número máximo de evaluaciones en la función objetivo definida por el usuario y el número de la evaluación actual y el $rango(a_j, b_j)$ denota la amplitud del dominio para cada componente.

La Ecuación 4.19 representa la función F la cual se define: para el primer caso como el valor medio entre el nodo actual y el extremo local de la $j-esima$ componente que se esté analizando. En segundo caso, la vecindad del extremo local se desplaza en función de un valor de distancia que depende de la iteración actual. En último caso, es generado un número aleatorio entre la $j-esima$ componente del valor medio y el nodo actual.

Generación de nodos en dirección al extremo global: Este paso tiene como objetivo acelerar la convergencia realizando un exploración en dirección al extremo global, es decir, el nodo con mejor valor de la función objetivo perteneciente a la población actual (n_g). Los nuevos nodos en dirección a (n_g) son generados usando al Ecuación (Apéndice I) :

$$n_g^* = G(n_i, n_g, Pr(n_i, n_g)), \tag{4.21}$$

donde Pr está definido en la por la Ecuación 4.18 y la función G se calcula como:

$$x_j^z = \begin{cases} promedio(x_j^i, x_j^g) & \text{si } \ U[0,1] < Pr(n_i, n_g) \\ U\left[promedio(x_j^i, x_j^g), x_j^g\right] & \text{en otro caso.} \end{cases} \tag{4.22}$$

Generación de nodos a partir de los nodos más externos de la malla: Este paso se realiza con el objetivo de completar los N_{TM} nodos de la malla en caso de no ser completados por los pasos anteriores. Este paso toma en cuenta los nodos más internos n_u y externos n_s de la malla que conforman la frontera interior y exterior respectivamente a partir del punto que representa el centro del espacio m dimensional. Para detectar esos nodos es empleada la norma de cada nodo que no es más que la distancia euclidiana al centro del espacio m dimensional. Partiendo de esos conjuntos, nuevos nodos son creados a partir de

cada nodo de la frontera empleando las ecuaciones 4.23 y 4.24:

$$x_j^h = \begin{cases} x_j^s + w_j & \text{si } 0 > x_j^s \\ \\ x_j^s - w_j & \text{si } 0 \leq x_j^s \end{cases} \tag{4.23}$$

$$x_j^h = \begin{cases} x_j^u + w_j & \text{si } 0 > x_j^u \\ \\ x_j^u - w_j & \text{si } 0 \leq x_j^u \end{cases} \tag{4.24}$$

donde w_j representa un desplazamiento para cada una de las componentes y es inversamente proporcional al número de iteraciones por lo cual va disminuyendo a medida que esta aumente y se puede calcular de acuerdo a la Ecuación 4.25:

$$w_j = (w_j^0 - w_j^1) * \frac{C - c}{C} + w_j^1, \tag{4.25}$$

donde w_j^0 y w_j^1 representa el desplazamiento inicial y final respectivamente, y sus valores pueden ser calculados como $w_j^0 = rango(a_j, b_j)/10$ y $w_j^1 = rango(a_j, b_j)/100$.

4.3.2. Proceso de contracción de la malla

En este proceso son seleccionados los nodos de la población que formarán parte de la malla en la próxima iteración del modelo. Primeramente es aplicado un método para incrementar la diversidad poblacional y preservar un mínimo de distancia entre los nodos de la malla. Para conseguir esto, un operador de limpieza adaptativo es propuesto. Seguidamente se brinda una descripción más detallada de dicho proceso.

Los nodos de la malla son ordenados de acuerdo con el valor de su fitness (ascendente). La diferencia entre cada nodo y su sucesor es calculada. Los nodos sucesores con una diferencia menor a ξ_j son eliminados de la malla. El término ξ_j es calculado por la Ecuación 4.20.

Por último los nodos con mejor fitness son seleccionados para la siguiente iteración del algoritmo. Si el número de nodos necesarios para la realización de una nueva iteración de la malla no es alcanzado, esta se completa con nodos generados de manera aleatoria.

4.4. Consideraciones Parciales

En este capítulo abordamos sobre las principales consideraciones tomadas en cuenta en la implementación de estas técnicas evolutivas como un problema de minimización para recuperar la fase óptica. Se implementaron 3 metaheurísticas de las cuales fueron descritas sus principales métodos, y las novedades abordadas durante el proceso de implementación. Para un mejor entendimiento se realizó un estudio de las relaciones entre clases y de esta forma ejemplificar el comportamiento de los algoritmos.

Capítulo 5

Resultados Experimentales

Este estudio centró sus bases en interferogramas simulados, así como algunas imágenes reales. Para validar el funcionamiento de las metaheurísticas se realizaron varios estudios que se presentaran a continuación, entre ellos un análisis comparativo entre los diferentes algoritmos. Se trabajó con imágenes que tienen una resolución de 20x20 pixeles e inferiores.

Se ajustaron los parámetros de entrada de cada algoritmo para obtener la mejor combinación de parámetros. Para cada combinación de parámetros se realizaron un total de 10 corridas y se tuvo como métrica para seleccionar la mejor configuración el error medio entre el mapa de fase encontrado y el simulado. Para definir la calidad de las soluciones se seleccionó como métrica el error medio entre el mapa de fase encontrado y el real. La Ecuación 5.1 representa el error calculado para cada aproximación.

$$Error = \frac{1}{RC} \sum_{x} \sum_{y} \left| \frac{f(x,y) - \phi(x,y)}{max(\phi(x,y)) - min(\phi(x,y))} \right| \times 100. \tag{5.1}$$

Por último se ralizó un estudio donde se involucran imágenes reales, donde primero se debe implementar un análisis para disminuir los tipos de ruido presentados en el interferograma para poder obtener una solución adecuada.

5.1. Experimento 1: RS para ajustar polinomios de grado 2

En el análisis experimental se seleccionó una serie de parámetros para estudiar la convergencia del Recocido Simulado para una codificación real. La siguiente tabla muestra los parámetros utilizados por el algoritmo de Recocido Simulado durante el proceso de experimentación.

Número de Iteraciones (N)	Individuos por niveles de Temperatura (ITemp)	Temperatura Inicial (T)	Factor de suavidad (m)
300	150	9000	10
500	200	90000	
1000	300	900000	
1500	500	9000000	

Tabla 5.1: Párametros del RS

Se obtuvieron todas las posibles combinaciones de parámetros para hacer una selección de la mejor configuración obtenida. Es válido destacar que el parámetro m se dejó fijo debido a un estudio previo, aunque es una consideración pendiente para estudios posteriores. Para cada combinación de parámetros se realizaron un total de 10 corridas y se tuvo como métrica para seleccionar la mejor configuración el error medio entre el mapa de fase encontrado y el simulado.

En la primera parte del proceso de experimentación se utilizaron 3 interferogramas de 6 franjas de resolución 20x20 pixeles y se realizó un ajuste para polinomios de segundo grado. De forma análoga a como se mencionó en capítulos anteriores se calculó los intervalos de búsqueda para coeficiente en particular, para ello se implementó un método (Apéndice J) que calcula de forma automática dichos intervalos a partir del número de franjas, el grado del polinomio que se desea interpolar y la resolución de la imagen. A continuación se muestras los intervalos de búsquedas para cada coeficiente:

$$a_0 \in [-37.6991 \quad 37.6991]$$
$$a_1 \in [-1.8850 \quad 1.8850]$$
$$a_2 \in [-1.8850 \quad 1.8850]$$
$$a_3 \in [-0.0942 \quad 0.0942]$$
$$a_4 \in [-0.0942 \quad 0.0942]$$
$$a_5 \in [-0.0942 \quad 0.0942]$$

Este métdo facilita el proceso de búsqueda de los intervalos de solución de cada coeficiente a partir de un conocimiento a priori.

5.1.1. Función de prueba 1

El primer interferograma simulado responde al polinomio descrito en la Ecuación 5.2.

$$\begin{aligned} f(x,y) &= 0.083765\,y^2 + 0.083765\,x^2 - 0.16753\,y \\ &\quad -0.33506\,x + 0.37778 \quad x,y \in [-15\ 15]\,. \end{aligned} \tag{5.2}$$

La Figura 5.1 muestra el interferograma en alta resolución generado por la Ecuación 5.2 para la primera simulación.

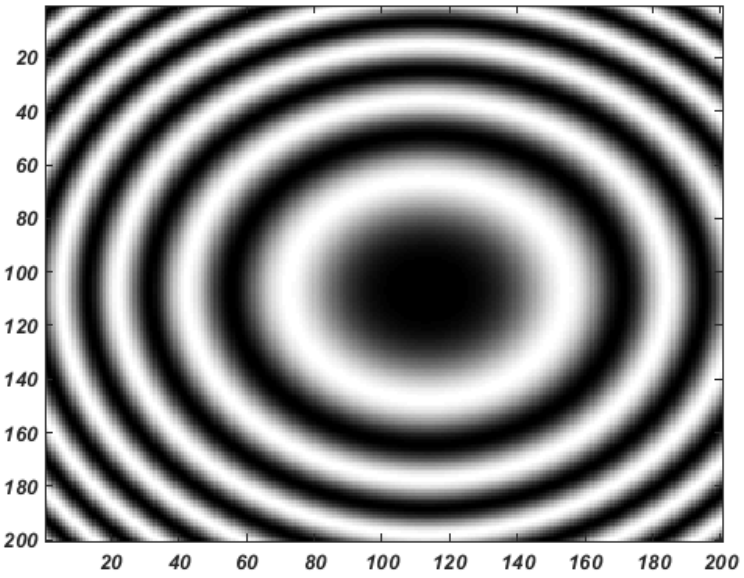
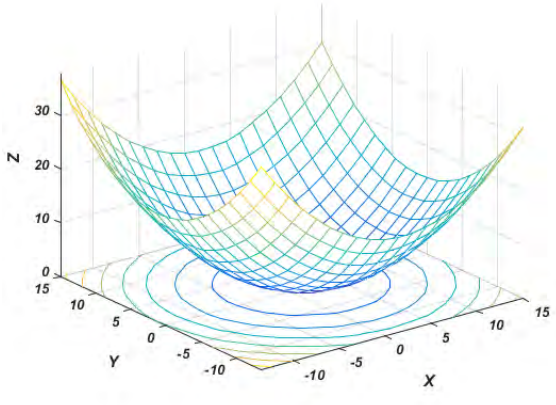
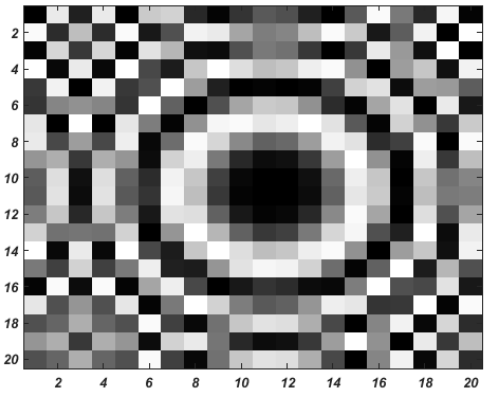


Figura 5.1: Interferograma correspondiente a la función de prueba 1 con resolución de 200 x 200 pixeles.

En este caso analizaremos cada combinación de parámetros tomando como referencia el número de iteraciones. Para cada caso mostraremos la mejor configuración obtenida. Este experimento de codificación real utilizó un factor de vecindad igual a 0.1, lo que significa que un coeficiente para encontrar una solución cercana variará su valor en un 10 % de su rango de búsqueda. La Figura 5.2 muestra el mapa de fase y el interferograma constituye una de las entrada del RS.



(a) Mapa de fase del objeto a recuperar.



(b) Interferograma de la forma a recuperar.

Figura 5.2: Representación del interferograma y el mapa de fase de la primera simulación.

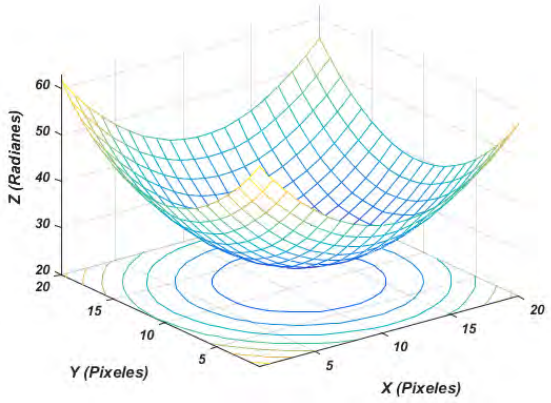
Caso 1

El primer grupo seleccionado fue para cuando el número de iteraciones (N) era igual a 300. La configuración de parámetros que arrrojó el mejor resultado fue cuando la temperatura inicial(T) comenzaba con un valor de 9000 y el número de generaciones por intervalo de temperatura fuese de 200. La Tabla 5.2 muestra los resultados de 10 ejecuciones con la configuración mencionada anteriormente:

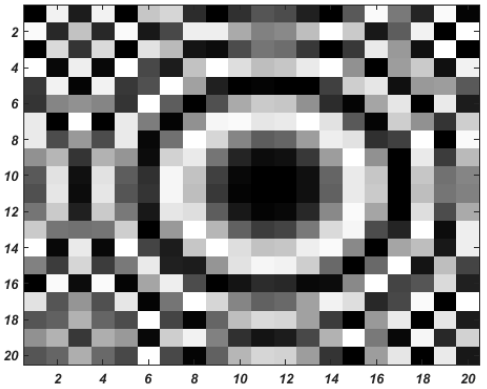
Corridas	**Fitness**	**Error**	**Iter Óptimo**
1	1694.393064	0.23368	295
2	8766.299922	3.47430	136
3	1684.560436	0.10667	230
4	1703.940234	0.25928	289
5	1685.115205	0.22348	294
6	2101.612990	0.69686	240
7	1716.842621	0.17941	267
8	1685.596070	0.12080	271
9	1679.475773	0.11529	285
10	2093.112264	0.71632	292
Promedio	2481.094858	0.612609	

Tabla 5.2: Resultados de 10 corridas para la mejor configuración con $N = 300$, $T = 9\ 000$ y $ITemp = 200$ para el interferograma de la Figura 5.2.

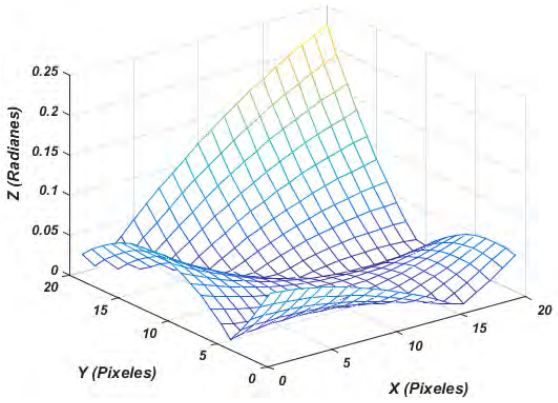
La tabla anterior muestra el mejor valor de la función objetivo para cada ejecución, el error definido por la Ecuación 5.1 y el número de la iteración donde se encontró la mejor solución. La mejor solución alcanzada es la representada por la número 3 con un tiempo de aproximadamente 28 segundos, a pesar de no tener el mejor valor de aptitud si alcanza el menor error, esto es debido al factor de suavidad utilizado. En la siguiente figura se presenta el mapa de fase y el interferograma recuperado, así como la diferencia entre la fase original y la recuperada.



(a) Mapa de fase recuperado.



(b) Interferograma recuperado.



(c) Diferencia entre la fase recuperada y la simulada.

Figura 5.3: Representación del interferograma y el mapa de fase de la primera simulación.

Para obtener la diferencia de fase fue necesario igualar los DC para ello se buscó la diferencia entre los máximos o mínimos y dicha cantidad se le adiciona o sustrae a la fase recuperada por el RS y se realiza la diferencia entre ambos mapas de fases. La interpolación polinómica que el algoritmo entregó como resultado final esta dada por la Ecuación 5.3:

$$\begin{aligned} f(x,y) &= 0.084203\,y^2 + 0.083644\,x^2 + 0.000460\,x\,y - 0.165590\,y \\ &\quad -0.333137\,x + 25.500252 \quad x, y \in [-15\ 15]\,. \end{aligned} \tag{5.3}$$

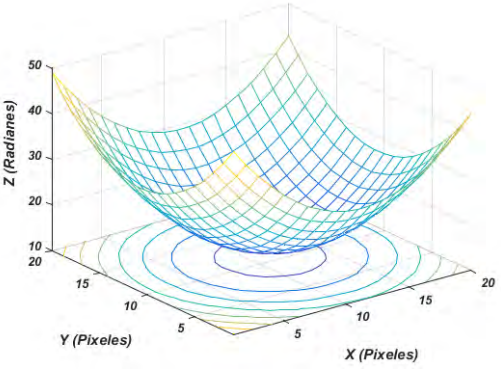
Caso 2:

Análogamente al primer caso se seleccionó la mejor configuración de parámetros cuando $N = 500$. En este caso resultó ser cuando la temperatura iniciaba con un valor de 9000 y el $Itemp = 400$. La Tabla 5.3 exhibe los resultados para 10 ejecuciones.

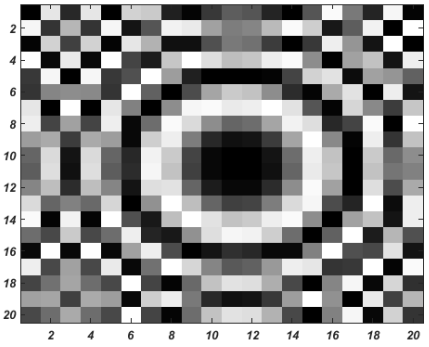
Corridas	Fitness	Error	Iter Óptimo
1	1666.078526	0.15912	459
2	1662.098003	0.10452	473
3	1664.389726	0.12286	478
4	1673.382250	0.16419	497
5	1681.787991	0.15648	449
6	1680.026483	0.12899	402
7	1675.633278	0.23417	472
8	1668.840209	0.081586	383
9	1687.360307	0.11791	431
10	1683.216263	0.10502	432
Promedio	1674.281304	0.1374846	

Tabla 5.3: Resultados de 10 corridas para la mejor configuración con $N = 500$ para el interferograma de la Figura 5.2.

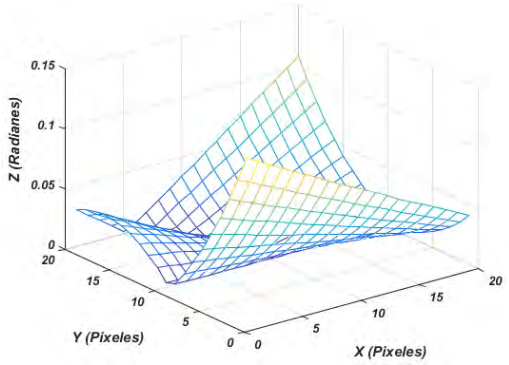
Para este grupo de configuraciones con $N = 500$ se obtuvo de forma general varias configuraciones que redujeron su error promedio por debajo de la unidad, en el Apéndice K indica los resultados experimentales de cada configuración. A continuación, la Figura 5.4 muestra la solución número 8, la cual indica un menor error al ser comparada con el mapa de fase simulado.



(a) Mapa de fase recuperado.



(b) Interferograma recuperado.



(c) Diferencia entre la fase recuperada y la simulada.

Figura 5.4: Representación del interferograma y el mapa de fase del caso 2.

La Figura 5.4a está representada por la siguiente aproximación polinomial obtenida como resultado final del algoritmo:

$$\begin{aligned} f(x,y) &= 0.083457\,y^2 + 0.083758\,x^2 - 0.000290\,x\,y - 0.165077\,y \\ &\quad -0.335550\,x + 12.986243 \quad x,y \in [-15\ 15]\,. \end{aligned} \tag{5.4}$$

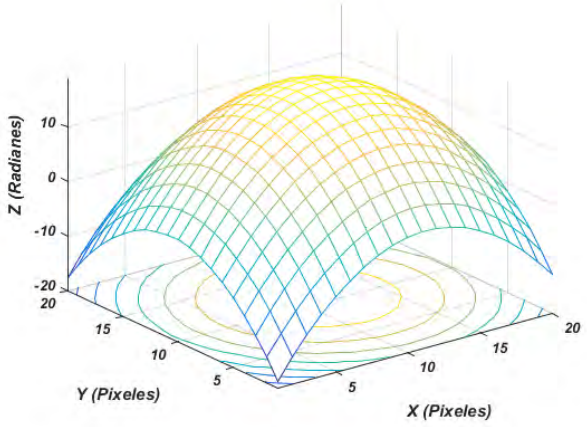
Caso 3:

El tercer caso para esta función fueron las combinaciones de parámetros cuando el valor de N era igual a = 1 000, este grupo está compuesto por 16 configuraciones de las cuales la mejor configuración está representada por la Tabla 5.4. La configuración que arrojó las mejores aproximaciones en promedio fue cuando el valor de la temperatura inicial comenzaba en 9 000, las generaciones por intervalo de temperatura era de 150, se utilizó un factor de suavidad igual a 10 y el factor de vecindad era de 0.1.

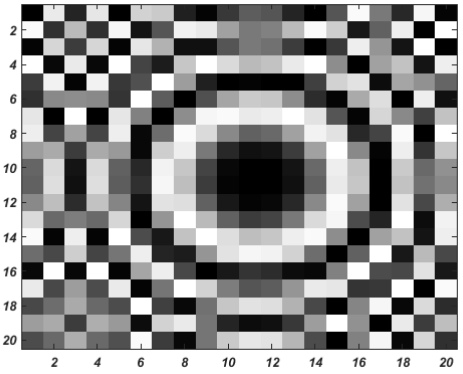
Corridas	**Fitness**	**Error**	**Iter Óptimo**
1	1672.489784	0.161560	878
2	1678.722883	0.096992	949
3	1683.094958	0.148170	970
4	1662.196769	0.202130	837
5	1674.017084	0.116640	936
6	1679.307789	0.244350	720
7	1668.251377	0.062914	998
8	1678.880717	0.110730	883
9	1670.665262	0.132040	920
10	1679.622765	0.232000	962
Promedio	1674.724939	0.150753	

Tabla 5.4: Resultados de 10 corridas para la mejor configuración con $N = 1000$ para el interferograma de la Figura 5.2.

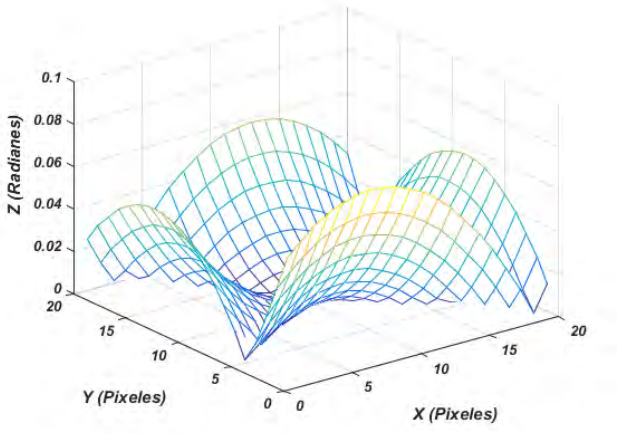
De las 10 ejecuciones ejemplificadas en la tabla anterior el mejor resultado lo muestra la corrida número 6 con un error de 0.062914 con un tiempo de ejecución de 83 segundos. La siguiente figura muestra los resultados obtenidos por el RS:



(a) Mapa de fase recuperado.



(b) Interferograma recuperado.



(c) Diferencia entre la fase recuperada y la simulada.

Figura 5.5: Representación del interferograma y el mapa de fase del caso 3.

La Figura 5.5a muestra el mapa de fase invertido, esto es debido a la paridad de la función *coseno* por lo que fue necesario invertir la fase aproximada para poderla comparar con la fase original. El mapa de fase de la Figura 5.5a está represenatdo por la solución del algoritmo y está dado por la siguiente aproximación polinomial:

$$\begin{aligned} f(x,y) &= -0.083452\,y^2 - 0.084015\,x^2 + 0.000155\,x\,y + 0.166266\,y \\ &\quad +0.335038\,x + 18.455666 \quad x,y \in [-15\ 15]\,. \end{aligned} \tag{5.5}$$

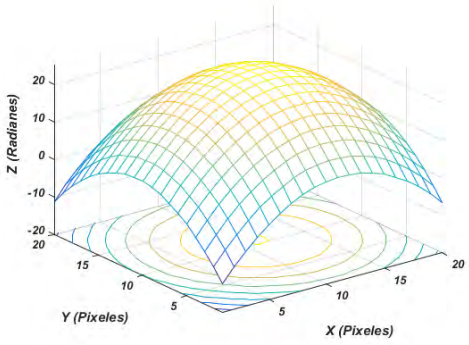
Caso 4:

El útimo caso utilizado para el interferograma generado por la Ecuación 5.2 fue el grupo de configuraciones donde el valor de N era igual a 1 500 iteraciones. La mejor combinación fue cuando la $T = 9000$, $ITemp = 300$, $m = 10$ y el factor de vecindad era de 0.1. La Tabla 5.5 muestra los resultados para 10 ejecuciones:

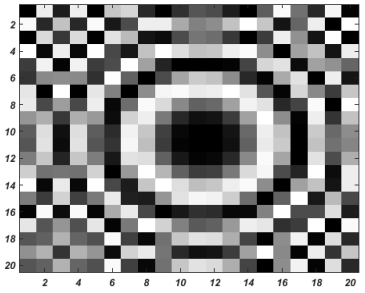
Corridas	Fitness	Error	Iter Óptimo
1	1660.729967	0.049896	1487
2	1671.047292	0.092701	1414
3	1666.252524	0.048909	1491
4	1664.435914	0.070877	1429
5	1672.434337	0.061455	1494
6	1668.267541	0.067217	1393
7	1675.625400	0.154320	1226
8	1660.121725	0.058415	1463
9	1675.591100	0.101460	1474
10	1668.120976	0.208000	1340
Promedio	1668.262678	0.091325	

Tabla 5.5: Resultados de 10 corridas para la mejor configuración con $N = 1500$ para el interferograma de la Figura 5.2.

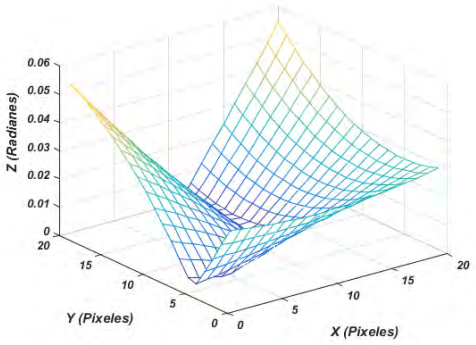
En general, esta fue la mejor configuración donde su promedio de error para 10 ejecuciones no excedió a 0.1. La mejor corrida fue la tercera con un tiempo de 250 segundos. A continuación mostramos una figura con el mapa de fase y el interferograma recuperado por el RS y una comparación con el mapa de fase original.



(a) Mapa de fase recuperado.



(b) Interferograma recuperado.



(c) Diferencia entre la fase recuperada y la simulada.

Figura 5.6: Representación del interferograma y el mapa de fase del caso 4.

La forma de la Figura 5.6a está dada por la siguiente aproximación obtenida como resultado del proceso de optimización del RS:

$$\begin{aligned} f(x,y) &= -0.083822\,y^2 - 0.083782\,x^2 - 0.000129\,x\,y + 0.168655\,y \\ &\quad +0.333263\,x + 24.745435 \quad x,y \in [-15\ 15]\,. \end{aligned} \tag{5.6}$$

5.1.2. Función de prueba 2

La segunda función de prueba utilizada en el primer experimento responde a la forma representada por el polinomio de la Ecuación 5.7:

$$\begin{aligned} f(x,y) &= 0.000579\,y^2 + 0.002319\,x^2 - 0.000579\,x\,y + 0.019714\,y \\ &\quad +0.046385\,x + 0.39717 \quad x,y \in [-100\ 100]\,. \end{aligned} \tag{5.7}$$

La Figura 5.7 muestra el interferograma en alta resolución generado por la ecuación anterior correspondiente al segundo interferograma de prueba.

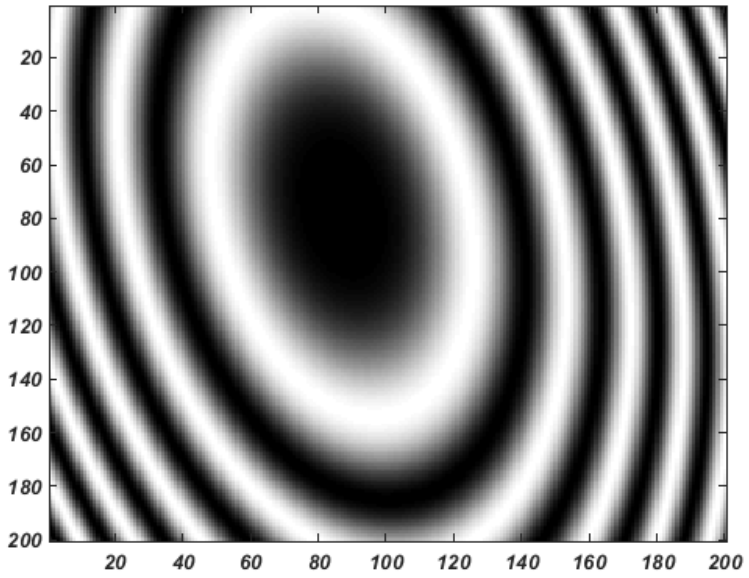
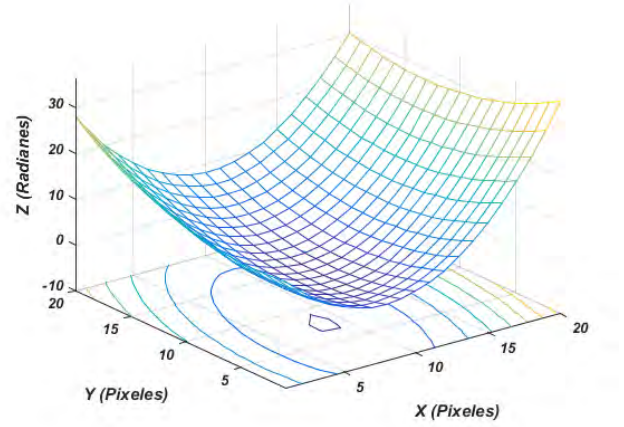
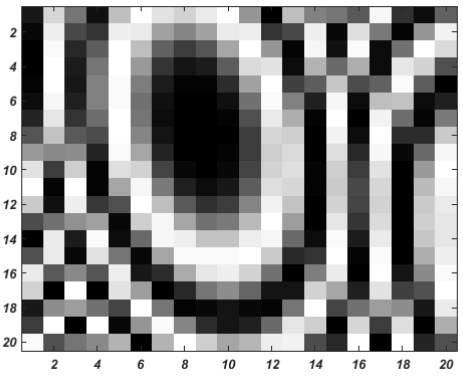


Figura 5.7: Interferograma de la función de prueba 2 con resolución de 200 x 200 pixeles.

Analizando las mejores configuraciones utilizadas en la función de prueba número 1 y teniendo en cuenta el factor tiempo se seleccionó la configuración $T = 9\ 000$; $N = 500$; $ITemp = 400$; $m = 10$ y factor de vecindad igual a 0.1 para probar los resultados con otras 2 formas. A continuación, se muestra la fase y el interferograma de la función a probar con los parámetros seleccionados.



(a) Mapa de fase del objeto a recuperar.



(b) Interferograma de la forma a recuperar.

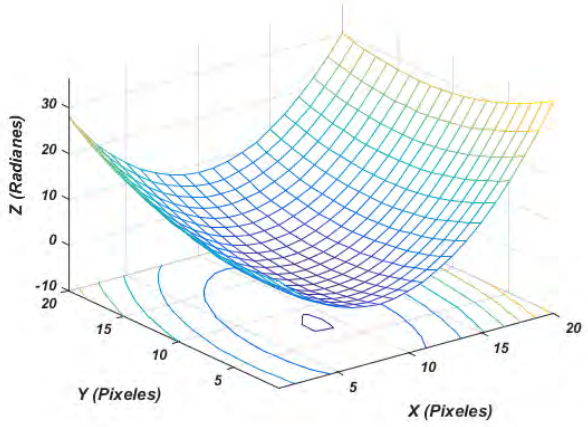
Figura 5.8: Representación del interferograma y el mapa de fase de la segunda simulación.

La siguiente tabla muestra el resultados de 10 ejecuciones del RS con la configuración seleccionada:

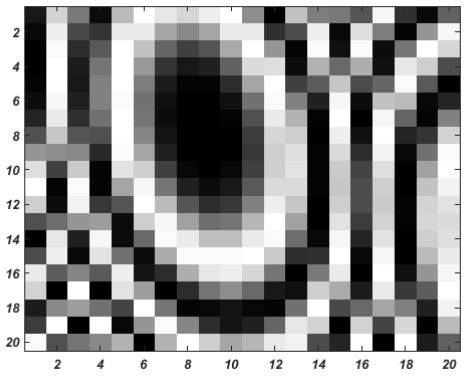
Corridas	**Fitness**	**Error**	**Iter Óptimo**
1	1302.414726	0.085314	497
2	1297.725474	0.076201	495
3	1323.082152	0.190950	489
4	1384.328234	0.470670	369
5	1355.929726	0.276650	479
6	1331.934832	0.277020	461
7	1353.438712	0.204160	485
8	1340.162541	0.198250	491
9	1354.578104	0.169680	372
10	1347.999478	0.230120	488
Promedio	1339.159398	0.217902	

Tabla 5.6: Resultados de 10 corridas para la mejor configuración con $N = 500$, $T = 9\ 000$, $ITemp = 400$, $m = 10$ y el factor de vecindad igual a 0.1, para el interferograma de la Figura 5.8.

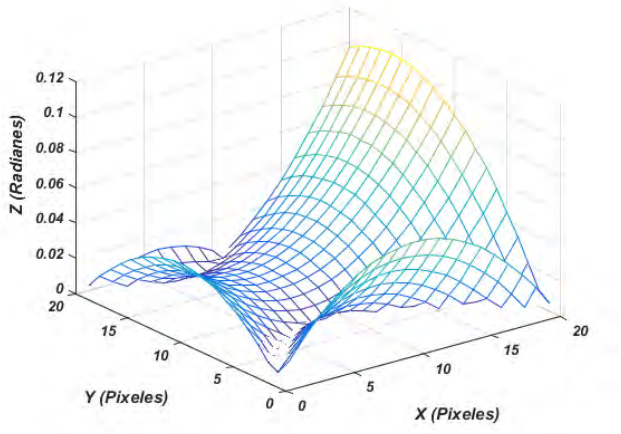
La siguiente figura muestra el mapa de fase, el interferograma recuperado por el algoritmo y una representación tridimensional de la diferencia de fase entre la forma aproximada y la original para la segunda ejecución que muestra un error de 0.076201.



(a) Mapa de fase recuperado.



(b) Interferograma recuperado.



(c) Diferencia entre la fase recuperada y la simulada.

Figura 5.9: Representación del interferograma y el mapa de fase de la configuración utilizada para la función de prueba 2.

El resultado obtenido por el algoritmo se muestra en la siguiente expresión. Esta ecuación resulta una proximación polinomial de la fase representada por la Figura 5.8a.

$$\begin{aligned} f(x,y) &= 0.000584\,y^2 + 0.002313\,x^2 - 0.000582\,x\,y + 0.019369\,y \\ &\quad +0.046185\,x + 0.381591 \quad x, y \in [-100\ 100]\,. \end{aligned} \tag{5.8}$$

5.1.3. Función de prueba 3

Al igual que en la función de prueba 2 esta función utilizó la misma combinación de parámetros y se realizaron un total de 10 ejecuciones para dicha configuración. La Figura 5.10 representa el interferograma simulado utilizado como tercera función de prueba para el proceso de experimentación.

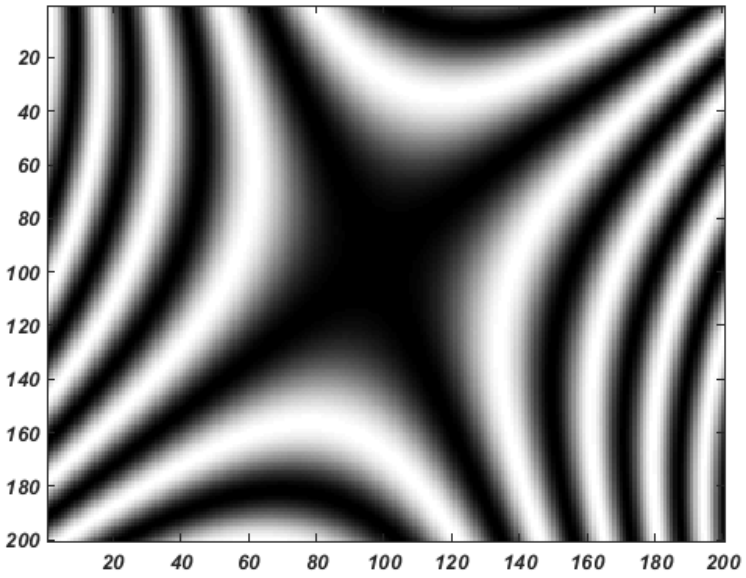
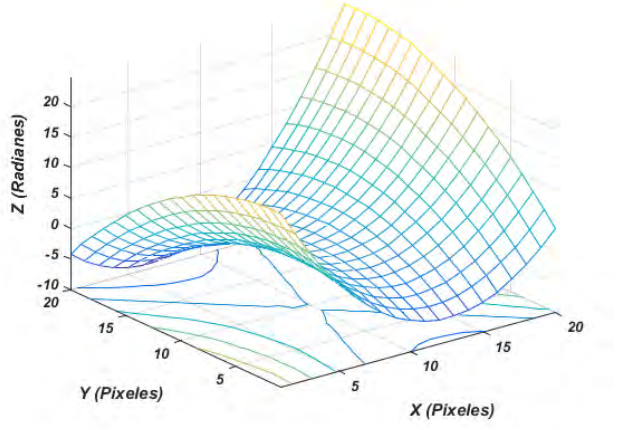


Figura 5.10: Interferograma de la función de prueba 3 con resolución de 200 x 200 pixeles.

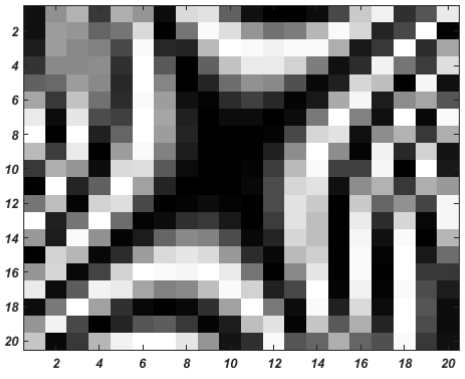
El interferograma de la figura anterior es el resultado de aplicar la Ecuación 1.1 utilizando como término de la función coseno la siguiente expresión:

$$\begin{aligned} f(x,y) &= -0.0026171\,y^2 + 0.0072698\,x^2 + 0.004798\,x\,y - 0.006979\,y \\ &\quad +0.026171\,x + 0.027043 \quad x, y \in [-50\ 50]\,. \end{aligned} \tag{5.9}$$

La Figura 5.11 muestra la fase y el interferograma de la forma representada por la ecuación 5.9 en una resolución de 20 x 20 pixeles y a su vez constituye la entrada del algoritmo RS.



(a) Mapa de fase del objeto a recuperar.



(b) Interferograma de la forma a recuperar.

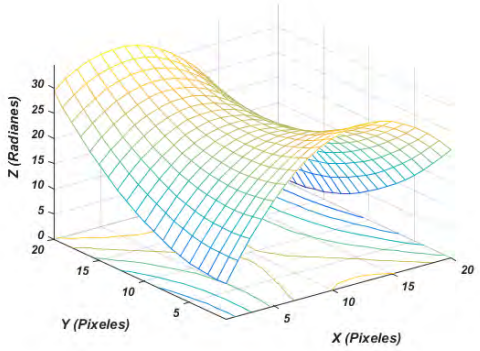
Figura 5.11: Representación del interferograma y el mapa de fase de la tercera simulación.

La Tabla 5.7 muestra los resultados de 10 corridas con la configuración seleccionada.

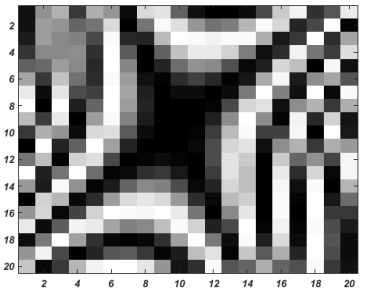
Corridas	Fitness	Error	Iter Óptimo
1	991.7600894	0.280830	424
2	1005.750229	0.655530	474
3	995.1653537	0.499540	382
4	1003.712147	0.149390	476
5	980.2318381	0.184000	462
6	987.6256447	0.193360	463
7	1009.797358	0.223760	498
8	1007.342767	0.137010	447
9	1000.495488	0.331660	495
10	1007.395283	0.248840	475
Promedio	998.9276198	0.290392	

Tabla 5.7: Resultados de 10 corridas para la mejor configuración con $N = 500$, $T = 9\ 000$, $ITemp = 400$, $m = 10$ y el factor de vecindad igual a 0.1, para el interferograma de la Figura 5.11.

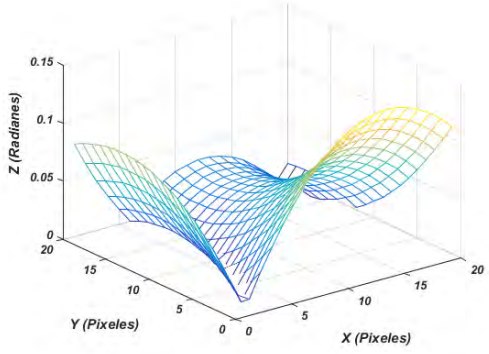
La siguiente figura muestra la fase y el interferograma obtenido por el algoritmo en la corrida número 8 y la diferencia de fase con la forma original.



(a) Mapa de fase recuperado.



(b) Interferograma recuperado.



(c) Diferencia entre la fase recuperada y la simulada.

Figura 5.12: Representación del interferograma y el mapa de fase de la configuración utilizada para la función de prueba 3.

La Ecuación 5.10 resulta una aproximación polinomial de la fase representada por la Figura 5.12a y representa la solución arrojada por el RS.

$$\begin{aligned} f(x,y) &= 0.002600\,y^2 - 0.007234\,x^2 - 0.004791\,x\,y \\ &\quad +0.008035\,y - 0.027366\,x + 25.140584. \end{aligned} \tag{5.10}$$

Se puede concluir el adecuado funcionamiento del algoritmo ya que tuvo una correcta exploración debido a que encontró la mejor solución en las iteraciones finales, esto se puede concluir en las tablas antes expuestas en el documento en la columna donde expresa la iteración óptima o donde se encontró la mejor solución.

5.2. Experimento 2: Análisis del factor de vecindad.

Un aspecto importante en el método de Recocido Simulado es la llamada función de vecindad la cual juega un papel importante a la hora de obtener una nueva solución vecina a la solución actual. Para obtener un nuevo vecino se parte de la solución actual, se genera un número aleatorio para seleccionar un coeficiente de la solución en cuestión. Después se toma una pequeña vecindad del espacio de búsqueda del coeficiente seleccionado y se le adiciona al mismo obteniendo una nueva solución vecina.

$$a_i = a_i + rand(Min(\Delta a_i),\ Max(\Delta a_i)), \tag{5.11}$$

donde α representa el porciento de la vecindad de estudio y Δa_i el rango del intervalo de búsqueda del coeficiente actual.

En este trabajo se estudió el tamaño de la vecindad para la cual se tomaron 3 dimensiones diferentes para estudiar el comportamiento del algoritmo de RS. Para llevar a cabo este análisis se tomó la configuración de parámetros utilizadas para la función 2 y 3 del primer experimento, esta configuraciónn es el resultado de un análisis de los resultados de las pruebas de la primera función donde se concluyó que esta configuración arrojaba los mejores resultados teniendo en cuenta: calidad de la solución y tiempo.

Caso 1:

El primer caso consiste en variar α tomando los siguientes valores (0.05; 0.1; 0.2). Estas nuevas variaciones fueron aplicadas a la función de prueba 1 (Ecuación 5.2). Al igual que en todos los casos se realizaron un total de 10 ejecuciones para medir la estabilidad de la metaheurística. Las siguientes tablas muestran los resultados para cada valor en estudio del parámetro α.

Corridas	Fitness	Error	Iter Óptimo
1	1661.581800	0.099790	490
2	2090.298982	0.713330	477
3	1671.242526	0.090969	481
4	1665.129604	0.072971	464
5	1663.534471	0.087666	497
6	1669.566165	0.158780	409
7	1666.370026	0.191790	390
8	1674.582531	0.052109	496
9	1673.581335	0.170240	468
10	1668.423069	0.049781	393
Promedio	1710.431051	0.168743	

Tabla 5.8: Resultados de 10 corridas para la mejor configuración con $N = 500$, $T = 9\ 000$, $ITemp = 400$, $m = 10$ y $\alpha = 0.05$, para el interferograma de la Figura 5.2.

Corridas	Fitness	Error	Iter Óptimo
1	1666.078526	0.159120	459
2	1662.098003	0.104520	473
3	1664.389726	0.122860	478
4	1673.382250	0.164190	497
5	1681.787991	0.156480	449
6	1680.026483	0.128990	402
7	1675.633278	0.234170	472
8	1668.840209	0.081586	383
9	1687.360307	0.117910	431
10	1683.216263	0.105020	432
Promedio	1674.281304	0.137485	

Tabla 5.9: Resultados de 10 corridas para la mejor configuración con $N = 500$, $T = 9\ 000$, $ITemp = 400$, $m = 10$ y $\alpha = 0.1$, para el interferograma de la Figura 5.2.

Corridas	Fitness	Error	Iter Óptimo
1	1683.558583	0.121080	496
2	1681.855941	0.150380	428
3	1677.526656	0.265600	452
4	1685.044823	0.115460	495
5	1680.740231	0.139450	459
6	1686.165735	0.256520	492
7	1678.672663	0.265980	360
8	1669.744229	0.179350	427
9	1678.158643	0.279750	484
10	1663.379236	0.073702	496
Promedio	1678.484674	0.184727	

Tabla 5.10: Resultados de 10 corridas para la mejor configuración con $N = 500$, $T = 9\ 000$, $ITemp = 400$, $m = 10$ y $\alpha = 0.2$, para el interferograma de la Figura 5.2.

El resultado fue similar para los 3 experimentos aunque el mejor resultado fue para cuando el valor de $\alpha = 0.1$.

Caso 2:

De forma análoga se varió el valor de α para la segunda función de prueba (Ecuación 5.7). A continuación mostramos las tablas donde se evidencian los resultados para cada configuración:

Corridas	Fitness	Error	Iter Óptimo
1	1319.556618	0.114280	493
2	1313.246176	0.094795	481
3	1307.417891	0.144090	471
4	1317.794934	0.124810	489
5	1360.923445	0.151190	396
6	1317.909437	0.125090	453
7	1298.481375	0.089830	457
8	1321.967124	0.189370	497
9	1338.290373	0.182430	466
10	1728.800311	0.765290	407
Promedio	1362.438768	0.198118	

Tabla 5.11: Resultados de 10 corridas para la mejor configuración con $N = 500$, $T = 9\ 000$, $ITemp = 400$, $m = 10$ y $\alpha = 0.05$, para el interferograma de la Figura 5.8.

Comparando los resultados de las tabla anteriores con el de la Tabla 5.6, observamos una similitud en el promedio de error al igual que en el caso de la función 1.

Corridas	Fitness	Error	Iter Óptimo
1	1333.902801	0.340600	434
2	1348.227628	0.243870	474
3	1525.830548	0.538510	398
4	1337.657858	0.198570	390
5	1370.038008	0.196140	400
6	1327.922459	0.134950	487
7	1297.849844	0.120260	486
8	1389.269295	0.471340	375
9	1309.961539	0.168380	448
10	1406.238395	0.283040	496
Promedio	1364.689838	0.269566	

Tabla 5.12: Resultados de 10 corridas para la mejor configuración con $N = 500$, $T = 9\ 000$, $ITemp = 400$, $m = 10$ y $\alpha = 0.2$, para el interferograma de la Figura 5.8.

Caso 3:

El último caso se aplicó la misma configuración a la función 3 de prueba con las mismas configuraciones para observar si mantenía el mismo comportamiento. Las siguientes tablas reflejan el resultado de 10 corridas para cada configuración:

Corridas	Fitness	Error	Iter Óptimo
1	1003.417828	0.198290	460
2	1012.989164	0.244100	430
3	982.7669280	0.249770	482
4	985.7036530	0.357150	385
5	977.7580740	0.156890	493
6	997.6479680	0.401610	427
7	968.4382860	0.130440	466
8	983.0428040	0.198610	495
9	1004.606670	0.356180	443
10	975.7724400	0.072317	477
Promedio	989.214381	0.2365357	

Tabla 5.13: Resultados de 10 corridas para la mejor configuración con $N = 500$, $T = 9\ 000$, $ITemp = 400$, $m = 10$ y $\alpha = 0.05$, para el interferograma de la Figura 5.8.

Corridas	Fitness	Error	Iter Óptimo
1	998.6050034	0.67972	446
2	981.6923270	0.15137	375
3	998.8885832	0.15369	397
4	973.5646239	0.11519	458
5	980.4258461	0.26521	483
6	1003.830864	0.17626	401
7	997.3466384	0.15066	490
8	992.6528804	0.28749	477
9	981.2845684	0.10029	461
10	1001.784035	0.26635	308
Promedio	991.0075369	0.234623	

Tabla 5.14: Resultados de 10 corridas para la mejor configuración con $N = 500$, $T = 9\ 000$, $ITemp = 400$, $m = 10$ y $\alpha = 0.2$, para el interferograma de la Figura 5.8.

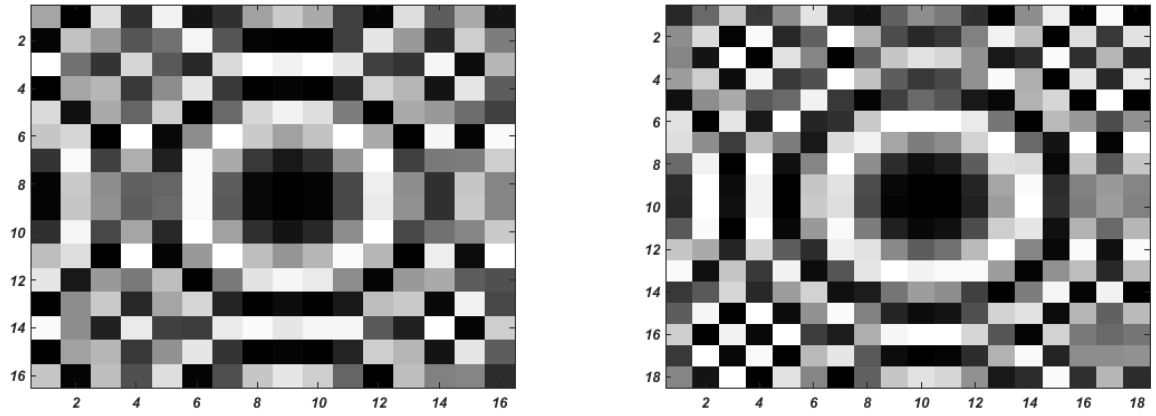
Comparando los resultados de las Tablas 5.7,5.13 y 5.14 observamos un comportamiento similar en cuanto al promedio de error de las 10 ejecuciones lo cual evidencia un comportamiento similar para los 3 casos estudiados en este experimento.

5.3. Experimento 3: Dimensión del interferograma.

Para estudiar la fortaleza de las metaheurística para la recuperación de la fase a partir de un interferograma se realizó un estudio donde se varió el tamaño del mismo para comprobar si el algoritmo de RS era capaz de aproximar la fase óptica. Los experimentos anteriores se realizaron con interferogramas de resolución de 20 x 20 pixeles, para este caso se tomaron dos resoluciones inferiores (18 x 18) y (16 x 16) donde los interferogramas no cumplían con el criterio de Nyquist, es decir existía un submuestreo de las franjas. Para este experimento se utilizó las 3 funciones de pruebas vistas en los experimentos anteriores. El RS utilizó como parámetros de entrada los valores de $T = 9\ 000$, $N = 500$, $ITemp = 400$, $m = 10$ y $\alpha = 0.1$.

Caso 1:

En el primer caso que analizaremos dos interferogramas de la primera función de prueba con resoluciones inferiores según muestra la Figura 5.13:



(a) Interferograma con resolución 16x16 pixeles.

(b) Interferograma con resolución 18x18 pixeles.

Figura 5.13: Representación de los interferogramas de entradas al algoritmo de RS.

Las siguientes tablas muestran los resultados de 10 corridas utilizando los interferograma de la Figura 5.13.

Corridas	**Fitness**	**Error**	**Iter Óptimo**
1	2503.213513	0.155060	473
2	2494.669025	0.138610	485
3	2510.477819	0.121260	493
4	2499.883141	0.138890	457
5	3697.346333	0.600950	223
6	2486.126932	0.083681	486
7	2500.392660	0.110140	452
8	2489.655271	0.129780	495
9	2499.391947	0.106890	489
10	2489.774077	0.142590	487
Promedio	2617.093072	0.172785	

Tabla 5.15: Resultados de 10 corridas para el interferograma de resolución de 16 x 16 píxeles

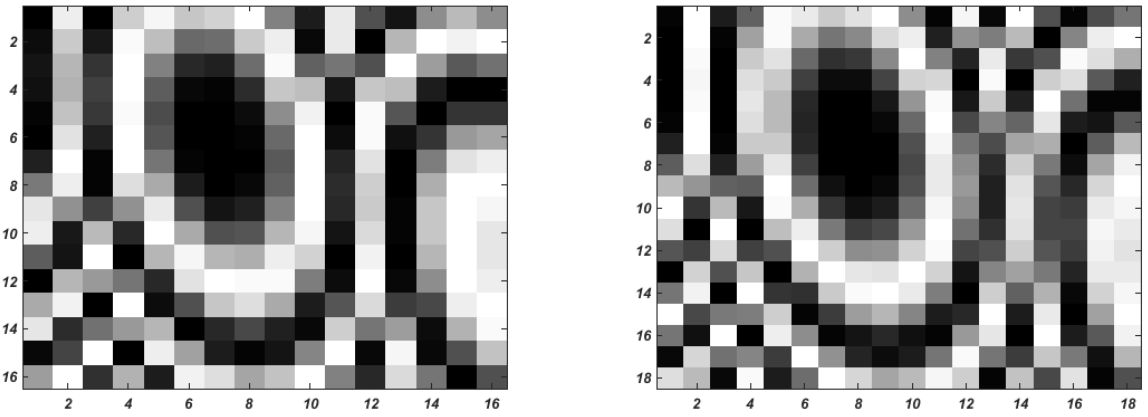
Corridas	Fitness	Error	Iter	Óptimo
1	2029.815398	0.172280		493
2	2026.103200	0.111280		469
3	2031.902695	0.161360		478
4	2021.675507	0.097647		469
5	2028.085511	0.102570		478
6	2016.330230	0.114670		447
7	2038.504022	0.132380		429
8	2031.526475	0.158490		463
9	2019.699560	0.147520		453
10	2051.103016	0.142190		460
Promedio	2029.474561	0.134039		

Tabla 5.16: Resultados de 10 corridas para el interferograma de resolución de 18 x 18 pixeles

Si comparamos los resultados de las tablas anteriores con la Tabla 5.9 que muestra los resultados de 10 ejecuciones con el interferograma representa la función con una resolución de 20 x 20 pixeles, observamos que los resultados son similares incluso se obtuvo un mejor promedio de error con el interferograma de resolución 18 x 18 pixeles.

Caso 2:

El segundo caso analiza 2 interferogramas de igual resolución que el primer caso, la única diferencia es que estos interferogramas corresponden a la función de prueba número 2. La Figura 5.14 corresponde a los interferogramas que se utilizan como entrada del RS.



(a) Interferograma con resolución 16x16 pixeles. (b) Interferograma con resolución 18x18 pixeles.

Figura 5.14: Representación de los interferogramas de entradas al algoritmo de RS.

A continuación se presenta las tablas de resultados de 10 corridas del RS para los interferogramas de la Figura 5.14.

Corridas	Fitness	Error	Iter Óptimo
1	1926.048551	0.096461	482
2	1926.084679	0.118370	467
3	1941.863821	0.182550	431
4	1955.348067	0.139560	493
5	1925.129606	0.071956	429
6	1915.857086	0.102310	437
7	1953.367007	0.124510	384
8	9113.077453	11.974000	347
9	1906.162931	0.062599	453
10	2001.799221	0.248310	344
Promedio	2656.473842	1.312063	

Tabla 5.17: Resultados de 10 corridas para el interferograma de resolución de 16 x 16 pixeles de la función de prueba 2

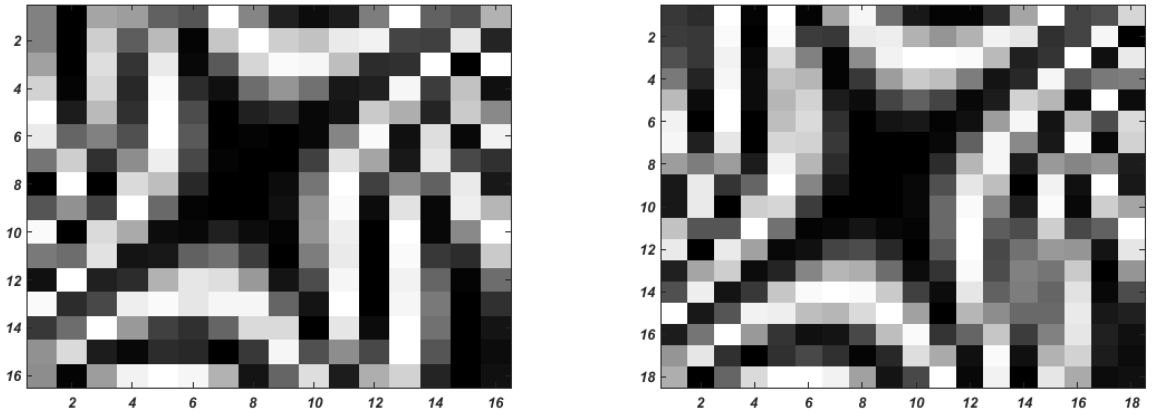
Corridas	Fitness	Error	Iter Óptimo
1	1617.559020	0.159440	457
2	1607.076284	0.293050	430
3	1631.852940	0.187550	492
4	1602.811424	0.169970	490
5	1584.227162	0.153410	444
6	1588.979648	0.183840	486
7	1580.400801	0.210110	485
8	1564.090503	0.165940	496
9	1566.031849	0.051664	454
10	1592.319163	0.124270	477
Promedio	1593.534879	0.169924	

Tabla 5.18: Resultados de 10 corridas para el interferograma de resolución de 18 x 18 pixeles de la función de prueba 2

Al igual que el primer caso el interferograma con resolución de 18 x 18 pixeles obtuvo los mejores resultados si lo comparamos con los resultados de la imagen de 20 x 20 pixeles mostrado en la Tabla 5.6, aunque de forma general el algoritmo pudo recuperar la fase para todos los casos.

Caso 3:

Este caso analizamos igual 2 interferogramas con las resoluciones analizadas en los casos anteriores. La Figura 5.15 muestras las imágenes que representan la entrada del RS para el proceso de optimización:



(a) Interferograma con resolución 16x16 pixeles.

(b) Interferograma con resolución 18x18 pixeles.

Figura 5.15: Representación de los interferogramas de entradas al algoritmo de RS.

Las siguientes tablas muestran los resultados de 10 ejecuciones para cada interferograma en particular.

Corridas	**Fitness**	**Error**	**Iter Óptimo**
1	1462.877722	0.384080	463
2	1457.412709	0.126790	411
3	1458.463434	0.151790	375
4	1486.113527	0.149040	497
5	1476.979753	0.099208	442
6	1484.092690	0.192960	462
7	1457.860142	0.204180	477
8	1453.739295	0.077299	496
9	1479.281831	0.319830	399
10	1459.332056	0.082134	470
Promedio	1467.615316	0.178731	

Tabla 5.19: Resultados de 10 corridas para el interferograma de resolución de 16 x 16 pixeles de la función de prueba 3

Corridas	Fitness	Error	Iter Óptimo
1	1185.365373	0.346540	485
2	1177.823786	0.070341	467
3	1172.239429	0.036773	494
4	1189.669728	0.085811	426
5	1215.964847	0.326520	424
6	1200.208464	0.165060	462
7	1179.723030	0.174190	489
8	1168.577560	0.098306	336
9	1186.100306	0.091337	430
10	1191.159597	0.247450	462
Promedio	1186.683212	0.164233	

Tabla 5.20: Resultados de 10 corridas para el interferograma de resolución de 18 x 18 pixeles de la función de prueba 3

De forma general el algoritmo mostró igual desempeño, logró recuperar el mapa de fase a pesar de existir un submuestreo de las franjas y la resolución con la cual se obtuvo los mejores resultados fue con la de 18 x 18 pixeles.

5.4. Experimento 4: RS para ajustar polinomios de grado 3

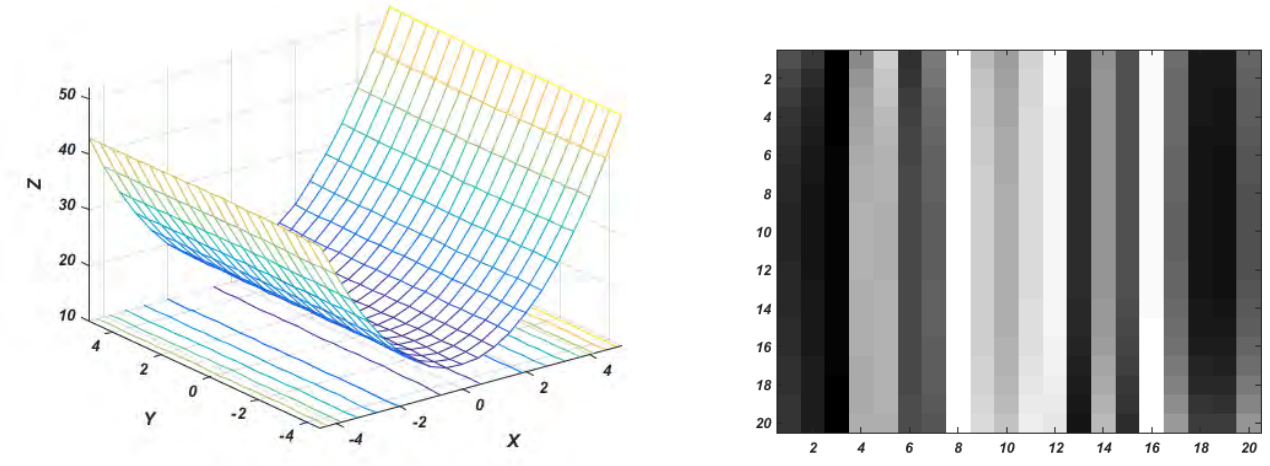
Al igual que el primer experimento se seleccionaron tres funciones de tercer grado para mostrar el comportamiento del RS con funciones más complejas. Se utilizaron imágenes de 20 x 20 pixeles y se dividió el experimento en tres casos donde se muestran las mejores configuraciones de parámetros. Para cada función se ejecutaron 10 corridas para demostrar la estabilidad de la metaheurística. Los interferogramas en alta resolución para cada una de las funciones de pruebas son mostrado en el Apéndice M.

5.4.1. Función de prueba 4

La cuarta función en estudio corresponde al polinomio descrito en la Ecuación 5.12.

$$\begin{aligned} f(x,y) &= 0.001449\,y^3 + 0.001449\,x^3 + 0.002898\,x\,y^2 + 1.449200\,x^2 \\ &\quad +0.144920\,x + 14.492001 \quad x,y \in [-5\ 5]\,. \end{aligned} \tag{5.12}$$

La Figura 5.16 muestra el mapa de fase y el interferograma simulado que es utilizado como entrada del RS.



(a) Mapa de fase del objeto a recuperar.

(b) Interferograma de la forma a recuperar.

Figura 5.16: Representación del interferograma y el mapa de fase de la función de prueba 4.

Para llevar a cabo el proceso de optimización se definió el espacio solución de cada componente, la cual viene representada de la siguiente manera:

$$a_0 \in [-37.6991 \quad 37.6991]$$

$$a_1 \in [-1.8850 \quad 1.8850]$$

$$a_2 \in [-1.8850 \quad 1.8850]$$

$$a_3 \in [-1.942 \quad 1.942]$$

$$a_4 \in [-1.942 \quad 1.942]$$

$$a_5 \in [-0.0942 \quad 0.0942]$$

$$a_6 \in [-0.00471 \quad 0.00471]$$

$$a_7 \in [-0.00471 \quad 0.00471]$$

$$a_8 \in [-0.00471 \quad 0.00471]$$

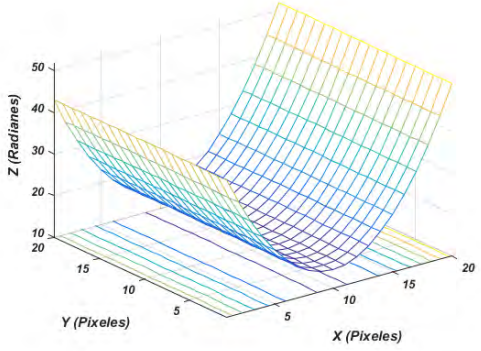
$$a_9 \in [-0.00471 \quad 0.00471]$$

La siguiente tabla muestra el mejor resultado de 10 ejecuciones, donde de forma general se obtuvo en conjunto el mejor promedio de error:

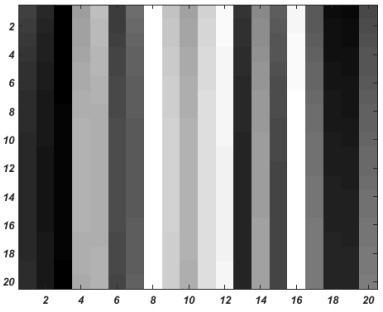
Corridas	Fitness	Error	Iter Óptimo
1	372.241267	0.282758	464
2	330.660849	0.151804	480
3	363.641565	0.202250	439
4	364.181474	0.230801	480
5	340.765234	0.269751	371
6	335.634570	0.179578	457
7	328.031898	0.153878	495
8	303.801642	0.173631	459
9	373.559959	0.223259	499
10	344.417664	0.183663	488
Promedio	345.693612	0.205137	

Tabla 5.21: Resultados de 10 corridas para la mejor configuración con $N = 500$, $T = 90\ 000$, $ITemp = 300$, $\alpha = 0.3$ (Factor de vecindad) y $m = 10$ para el interferograma de la Figura 5.16.

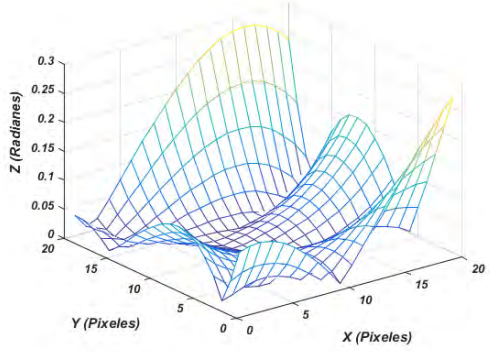
La mejor solución alcanzada es la representada por la número 2 donde alcanza el menor valor de error. En la siguiente figura se presenta el mapa de fase y el interferograma recuperado, así como la diferencia entre la fase original y la recuperada.



(a) Mapa de fase recuperado.



(b) Interferograma recuperado.



(c) Diferencia entre la fase recuperada y la simulada.

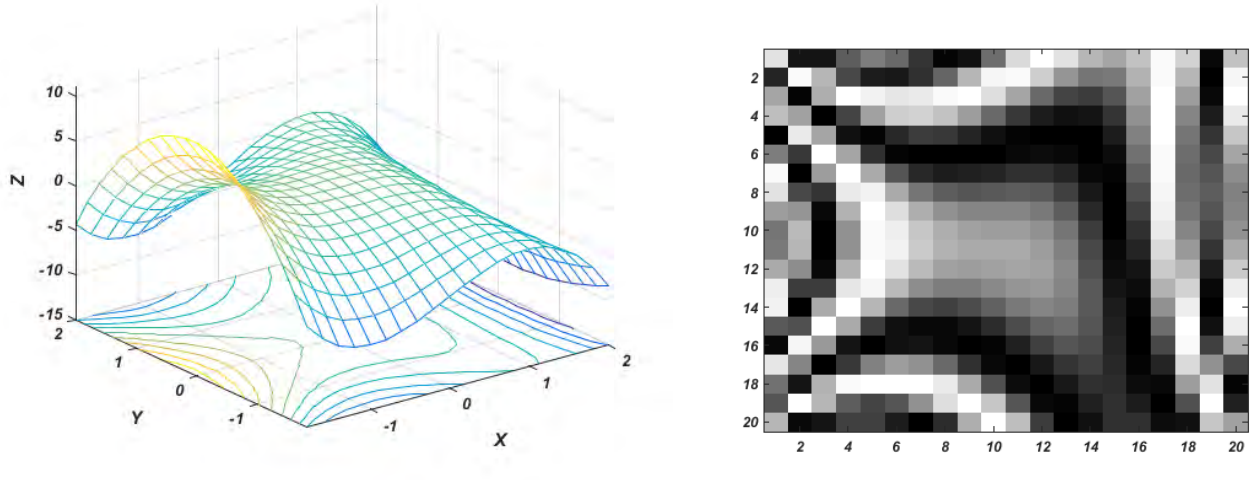
Figura 5.17: Representación del interferograma y el mapa de fase de la primera simulación.

La ejecución número 1 entregó como resultado la siguiente aproximación polinomial:

$$\begin{aligned} f(x,y) &= 0.000161\,y^3 + 0.000740\,x^3 + 0.000864\,x^2\,y + 0.000337\,y^2\,x - 0.003840\,y^2 \\ &\quad +1.449500\,x^2 + 0.001940\,x\,y + 0.009630\,y + 0.154790\,x + 27.103740 \end{aligned} \tag{5.13}$$

5.4.2. Función de prueba 5

Esta función representa el segundo polinomio de tercer grado utilizado durante el proceso de experimentación. La Figura 5.18 muestra el interferograma simulado utilizado en el algoritmo de RS y el mapa de fase a aproximar por el algoritmo.



(a) Mapa de fase del objeto a recuperar. (b) Interferograma de la forma a recuperar.

Figura 5.18: Representación del interferograma y el mapa de fase de la función de prueba 5.

La forma de la Figura 5.18a representa el siguiente polinomio:

$$\begin{aligned} f(x,y) &= 0.018228\,y^3 - 1.585909\,x^3 + 0.000018\,x^2\,y + 1.421850\,x\,y^2 - 1.640596\,y^2 \\ &\quad +0.018228\,x\,y + 0.692696\,y + 1.822885 \quad x,y \in [-2\ 2]\,. \end{aligned} \tag{5.14}$$

Debido a la complejidad de la función de prueba fue necesario ampliar el intervalo de búsqueda de los coeficientes lo que hace más complejo el proceso de experimentación. Estos intervalos fueron tomados empíricamente con el objetivo de hacer el dificultar el proceso de búsqueda y demostrar la capacidad de las metaheurísticas. A continuación se indica el rango

de factibilidad de cada coeficiente:

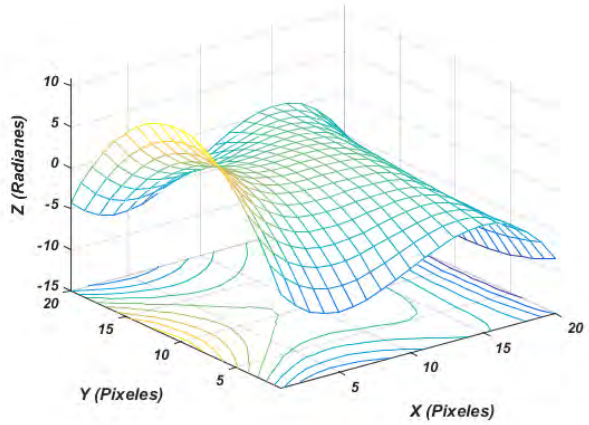
$$a_0 \in [-37.6991 \quad 37.6991]$$
$$a_1 \in [-1.8850 \quad 1.8850]$$
$$a_2 \in [-1.8850 \quad 1.8850]$$
$$a_3 \in [-0.0942 \quad 0.0942]$$
$$a_4 \in [-1.942 \quad 1.942]$$
$$a_5 \in [-2.0942 \quad 2.0942]$$
$$a_6 \in [-2.00471 \quad 2.00471]$$
$$a_7 \in [-2.00471 \quad 2.00471]$$
$$a_8 \in [-2.00471 \quad 2.00471]$$
$$a_9 \in [-2.00471 \quad 2.00471]$$

Al igual que en el resto de los experimentos se realizaron 10 corridas para cada configuración de parámetros. Los resultados de la configuración que computó los mejores resultados son mostrados en la siguiente tabla:

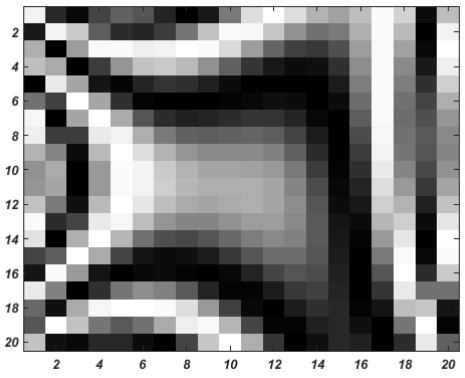
Corridas	**Fitness**	**Error**	**Iter Óptimo**
1	156.986541	1.215400	489
2	131.417767	0.590080	490
3	293.694156	0.705426	498
4	324.132837	0.765986	478
5	186.418955	1.031604	459
6	153.525024	0.325781	498
7	116.626056	0.374107	496
8	131.424008	0.580728	493
9	162.242572	0.475001	499
10	241.242270	0.654481	499
Promedio	189.771019	0.671859	

Tabla 5.22: Resultados de 10 corridas para la mejor configuración con $N = 500$, $T = 900\,000$, $ITemp = 200$, $\alpha = 0.3$ (Factor de vecindad) y $m = 10$ para el interferograma de la Figura 5.16.

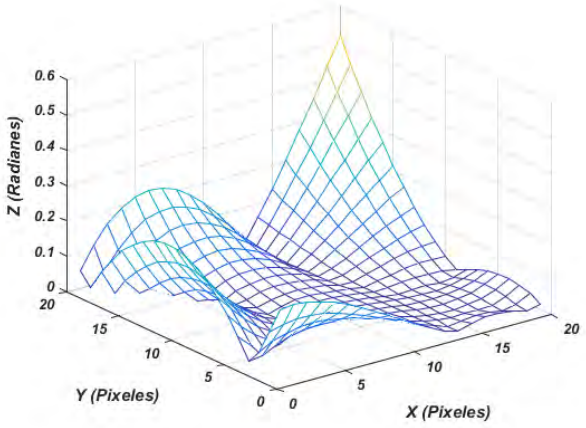
Analizando la tabla anterior observamos que la mejor solución se alcanzó en la sexta ejecución debido a que se alcanzó un error de 0.325781. La Figura 5.19 muestra una representación tridimensional del objeto recuperado, el interferograma por el algoritmo y la diferencia entre la fase original y la obtenida por el RS.



(a) Mapa de fase recuperado.



(b) Interferograma recuperado.



(c) Diferencia entre la fase recuperada y la simulada.

Figura 5.19: Representación del interferograma y el mapa de fase de la función de prueba número 5.

El vector de coeficientes obtenido como solución del algoritmo representa el siguiente polinomio:

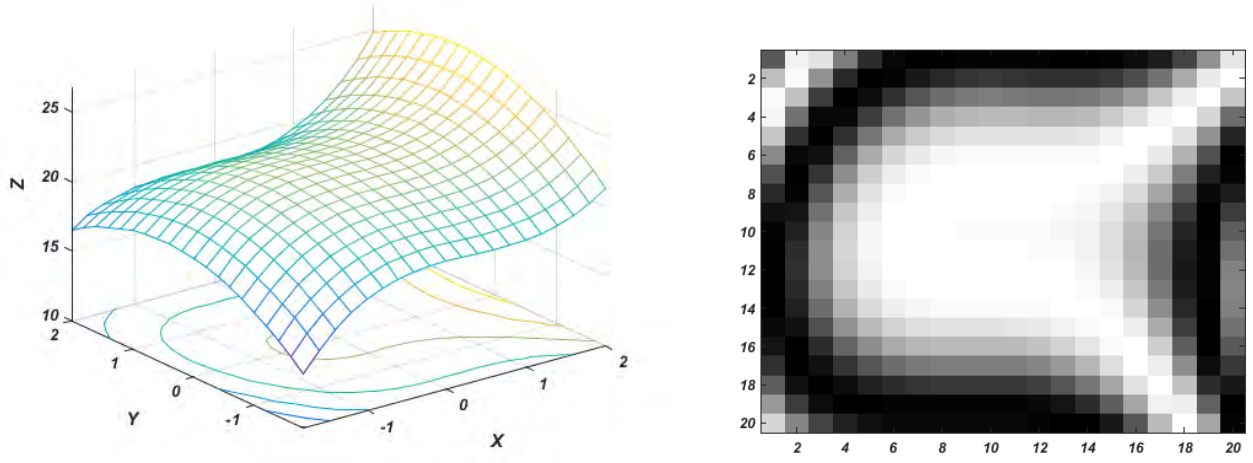
$$\begin{aligned} f(x,y) &= -0.018977\,y^3 + 1.568593\,x^3 + 0.029504\,x^2\,y - 1.380444\,y^2\,x + 1.616317\,y^2 \\ &\quad +0.045970\,x^2 - 0.006025\,x\,y - 0.703901\,y + 0.000782\,x - 20.719066. \end{aligned} \tag{5.15}$$

5.4.3. Función de prueba 6

La sexta función que fue seleccionada para el proceso de experimentación esta representada por el siguiente polinomio de grado 3:

$$\begin{aligned} f(x,y) &= 0.576478\,x^3 + 0.278328\,x^2\,y - 0.953254\,y^2 \\ &\quad -0.113313\,x^2 + 22.266270 \qquad x,y \in [-2\ 2]\,. \end{aligned} \tag{5.16}$$

La Figura 5.20 representa el interferograma y el mapa de fase que se utilizan como entradas para el algoritmo de optimización:



(a) Mapa de fase del objeto a recuperar.

(b) Interferograma de la forma a recuperar.

Figura 5.20: Representación del interferograma y el mapa de fase de la función de prueba 6.

Esta función de prueba utiliza los mismos rangos de búsquedas utilizados en la función 5, lo cual hace más engorroso el proceso debido al aumento del espacio solución. La siguiente tabla muestra los resultados de 10 corridas para la mejor configuración de parámetros obtenidos a través del proceso de experimentación.

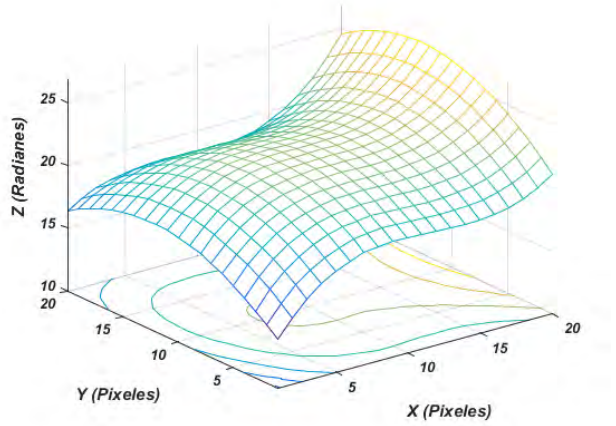
Corridas	**Fitness**	**Error**	**Iter Óptimo**
1	95.134412	0.844956	498
2	92.730214	0.927453	434
3	92.569789	0.889603	491
4	92.636671	1.249151	493
5	58.942886	0.541819	497
6	128.109107	0.997244	476
7	107.955886	0.876878	487
8	119.327371	1.243946	493
9	111.209240	1.361371	384
10	69.390262	1.472873	497
Promedio	96.800584	1.040529	

Tabla 5.23: Resultados de 10 corridas para la mejor configuración con $N = 500$, $T = 900\,000$, $ITemp = 300$, $\alpha = 0.3$ (Factor de vecindad) y $m = 10$ para el interferograma de la Figura 5.16.

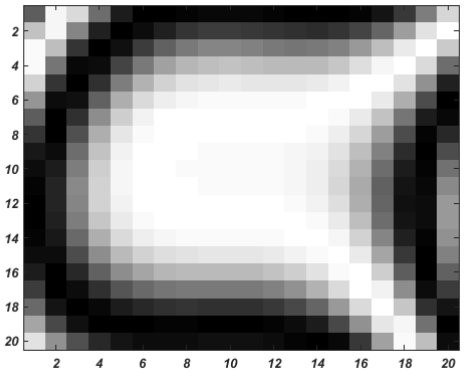
De la tabla anterior se deduce que la mejor solución obtenida por el RS fue la corrida 5. La siguiente ecuación muestra el polinomio obtenido a través del proceso de optimización.

$$\begin{aligned} f(x,y) \;=\; & -0.012441\,y^3 + 0.597323\,x^3 + 0.289703\,x^2\,y - 0.032588\,y^2\,x - 0.975898\,y^2 \\ & -0.131835\,x^2 + 0.014890\,x\,y - 0.012827\,y + 0.007501\,x + 34.8895863. \end{aligned} \quad (5.17)$$

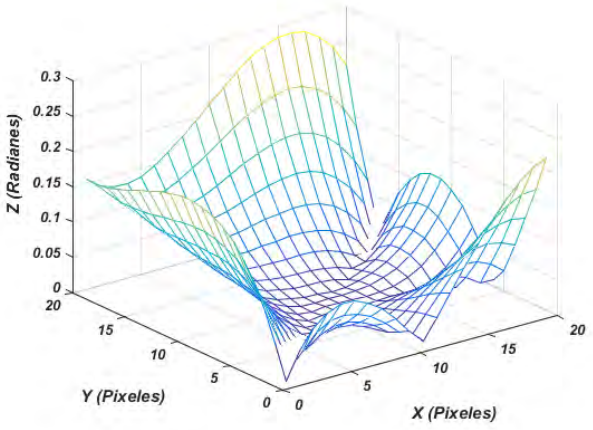
La Figura 5.21 esquematiza una representación tridimensional de la fase del objeto recuperado, así como su interferograma y una compración con el mapa de fase original.



(a) Mapa de fase recuperado.



(b) Interferograma recuperado.



(c) Diferencia entre la fase recuperada y la simulada.

Figura 5.21: Representación del interferograma y el mapa de fase de la función de prueba número 6.

5.5. Experimento 5: RS codificación binaria

El quinto experimento consiste en poner a prueba la implementación del RS con una codificación binaria. Para ellos se seleccionó la función de prueba 1 con la mejor configuración de parámetros obtenida para el caso de la codificación real. Para la codificación binaria fue necesario introducir un nuevo parámetro, es decir la cantidad de bits que se utilizarán para codificar cada componente. A continuación muestran las tablas con los resultados de 10 corridas con dos tamaños diferentes de bits:

Corridas	Fitness	Error	Iter Óptimo
1	439.441494	0.543715	366
2	410.909323	0.414563	290
3	310.982481	0.276415	351
4	392.482439	0.414563	332
5	306.928643	0.276415	321
6	392.482439	0.414563	282
7	400.963034	0.414564	311
8	304.359147	0.276415	283
9	436.195338	0.385691	292
10	316.520699	0.276415	264
Promedio	371.126504	0.369339	

Tabla 5.24: Resultados de 10 corridas para la mejor configuración con $N = 500$, $T = 9\ 000$, $ITemp = 400$, $\alpha = 0.1$ (Factor de vecindad), $Nbits = 7$ (número de bits) y $m = 10$ para el interferograma de la Figura 5.2.

Corridas	Fitness	Error	Iter Óptimo
1	182.060424	0.089083	458
2	180.833953	0.074264	440
3	183.047837	0.100843	395
4	181.925774	0.093454	458
5	175.387402	0.056359	407
6	186.956340	0.138803	338
7	186.559754	0.083770	445
8	180.235083	0.077924	414
9	185.096029	0.107922	475
10	189.649367	0.143938	479
Promedio	183.175196	0.096636	

Tabla 5.25: Resultados de 10 corridas para la mejor configuración con $N = 500$, $T = 9\ 000$, $ITemp = 400$, $\alpha = 0.1$ (Factor de vecindad), $Nbits = 15$ (número de bits) y $m = 10$ para el interferograma de la Figura 5.2.

Como se puede observar en ambas tablas el error estuvo inferior a la unidad aunque los mejores resultados se obtuvieron cuando se incrementó el número de bits, al igual que la codificación esta variante es capaz de encontrar el mapa de fase de un interferograma.

5.6. Experimento 6: VMO para el ajuste de funciones

Para llevar a cabo este experimento se escogieron una serie de combinaciones de parámetros del algoritmo y se seleccionó la que entregó mejor resultado. Todas estas combinaciones se probaron con las 6 funciones de pruebas antes descritas en experimentos anteriores, al

igual que en el RS estas pruebas se realizaron en igualdad de condiciones , es decir, utilizaron los mismos rangos para la búsqueda de los coeficientes. A continuación mostraremos las gráficas con las 10 ejecuciones y una representación tridimensional de la solución encontrada.

5.6.1. Resultados de VMO para la función de prueba 1.

La Tabla 5.26 muestra los resultados de 10 ejecuciones del algoritmo VMO aplicado al interferograma de la Figura 5.2. Además se mencionan los parámetros de configuración de la metaheurística que arrojaron los mejores resultados.

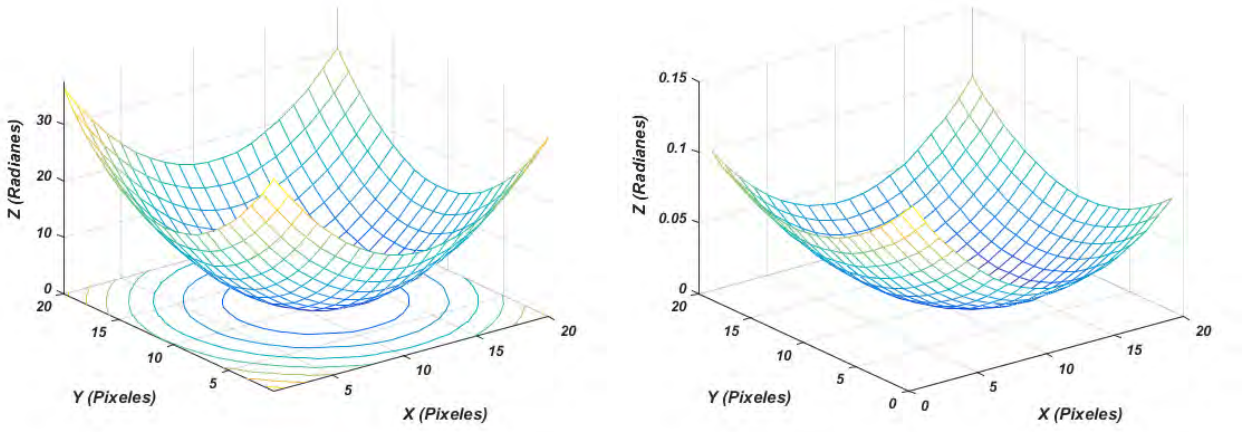
Corridas	**Fitness**	**Error**
1	186.623108	0.178389
2	171.730900	0.099519
3	197.851256	0.231858
4	184.582873	0.119756
5	182.993673	0.181409
6	188.920025	0.205039
7	271.578747	0.481715
8	177.549543	0.101468
9	190.160936	0.232185
10	186.519854	0.158671
Promedio	197.519613	0.214013

Tabla 5.26: Resultados de 10 corridas de VMO para la mejor configuración con $N_{IM} = 20$, $N_{TM} = 70$, $k = 9$, $M = 100\ 000$ y $m = 10$ para el interferograma de la Figura 5.2.

De la tabla anterior se deduce que la mejor ejecución fue la número 2 con un error de 0.099519, esta solución tuvo como resultado la siguiente aproximación polinomial:

$$\begin{aligned} f(x,y) &= -0.083572\,y^2 - 0.083462\,x^2 + 0.000058\,x\,y \\ &\quad +0.167071\,y + 0.333852\,x + 5.862597. \end{aligned} \tag{5.18}$$

La Figura 5.22 muestra la forma recuperada por VMO la cual está dada por la Ecuación 5.18.



(a) Mapa de fase del objeto a recuperar.

(b) DIferencia entre la fase obtenida por VMO y la original.

Figura 5.22: Mapa y diferencia de fase de la solución obtenida por VMO de la función de prueba 1.

5.6.2. Resultados de VMO para la función de prueba 2.

Análogamente a los experimentos del RS se seleccionaron una serie de parámetros y se formaron ciertas combinaciones de las cuales se escogió la mejor configuración. La Tabla 5.27 muestra los resultados de 10 ejecuciones para la configuración seleccionada:

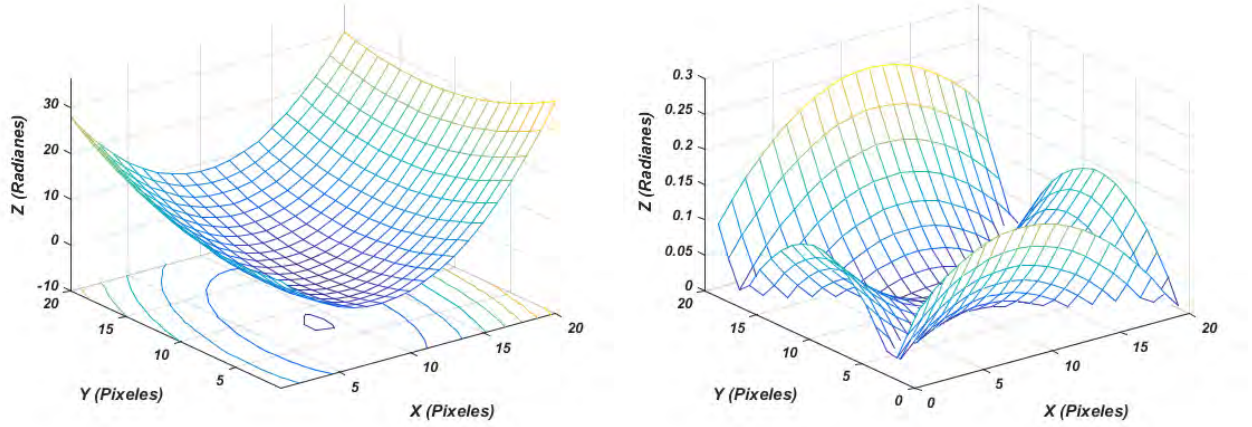
Corridas	Fitness	Error
1	1492.738915	0.412393
2	1371.247021	0.245120
3	2250.970403	1.667687
4	3843.349370	2.753748
5	1384.679889	0.227286
6	1568.959004	0.580199
7	1831.707083	0.712581
8	1998.407040	1.083633
9	10271.540449	19.975761
10	1357.049422	0.190836
Promedio	2737.064860	2.784924

Tabla 5.27: Resultados de 10 corridas de VMO para la mejor configuración con $N_{IM} = 40$, $N_{TM} = 124$, $k = 7$, $M = 400\ 000$ y $m = 100$ para el interferograma de la Figura 5.8.

La mejor solución fue la corrida número 10 la siguiente ecuación muestra la aproximación polinomial obtenida como resultado de VMO:

$$\begin{aligned} f(x,y) &= -0.083572\,y^2 - 0.083462\,x^2 + 0.000058\,x\,y \\ &\quad +0.167071\,y + 0.333852\,x + 5.862597. \end{aligned} \tag{5.19}$$

La Figura 5.23 ejemplifica la forma tridimensional que representa la Ecuación 5.19 y la diferencia de fase entre la original y la recuperada.



(a) Mapa de fase del objeto a recuperar.

(b) DIferencia entre la fase obtenida por VMO y la original.

Figura 5.23: Mapa y diferencia de fase de la solución obtenida por VMO de la función de prueba 2.

5.6.3. Resultados de VMO para la función de prueba 3.

La Tabla 5.28 muestra los resultados obtenidos por VMO para la tercera función de prueba (Figura 5.11). Aunque la corrida número 6 no pudo encontrar la forma, el algoritmo de forma general pudo encontrar la fase en el resto de las ejecuciones.

Corridas	Fitness	Error
1	276.644924	1.059843
2	222.160471	0.827386
3	272.380843	0.907505
4	402.705341	1.631045
5	537.370480	0.746252
6	1591.167251	36.592319
7	222.544772	1.207330
8	278.273109	0.289327
9	140.646542	0.340429
10	336.608624	0.782309
Promedio	428.050236	4.438375

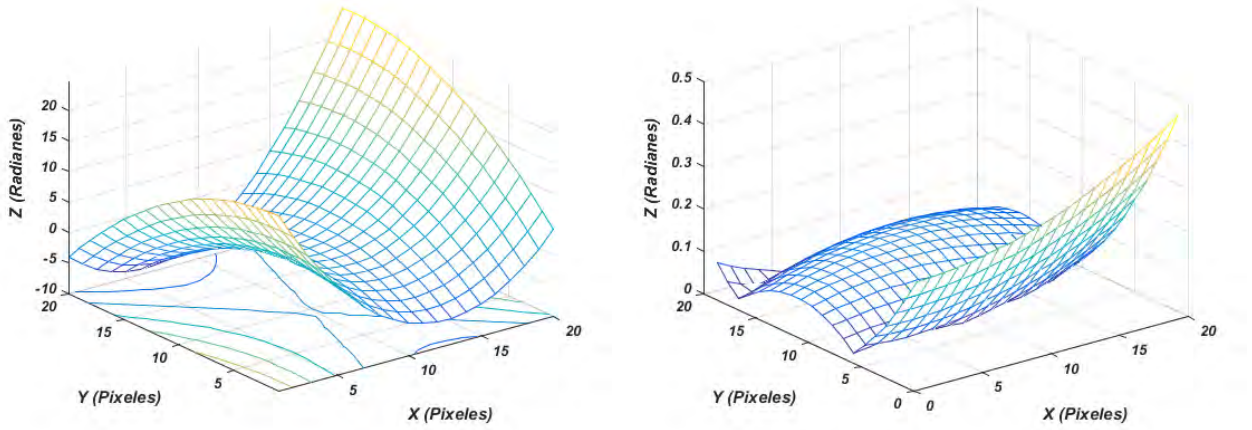
Tabla 5.28: Resultados de 10 corridas de VMO para la mejor configuración con $N_{IM} = 40$, $N_{TM} = 124$, $k = 9$, $M = 150\ 000$ y $m = 10$ para el interferograma de la Figura 5.11.

La corrida número 8 muestra la mejor aproximación con un error de 0.289327. La Ecua-

ción 5.20 representa la aproximación polinomial encontrada por VMO:

$$\begin{aligned} f(x,y) &= 0.002523\,y^2 - 0.007293\,x^2 - 0.004763\,x\,y \\ &\quad +0.010820\,y - 0.026857\,x + 6.338387. \end{aligned} \tag{5.20}$$

La siguiente figura representa la fase aproximada encontrada por el algoritmo y una comparación entre las fases mostrando la diferencia entre la fase original y la aproximada.



(a) Mapa de fase del objeto a recuperar.

(b) Diferencia entre la fase obtenida por VMO y la original.

Figura 5.24: Mapa y diferencia de fase de la solución obtenida por VMO de la función de prueba 3.

5.6.4. Resultados de VMO para la función de prueba 4.

La función de prueba 4 corresponde a un polinomio de tercer grado y al igual que en RS se utilizó el intervalo de búsqueda ampliado para corroborar la fortaleza de la metaheurística. La Tabla 5.29 muestra los resultados de 10 ejecuciones, donde se evidencia que para esta función de tercer grado el algoritmo fue capaz de demodular la fase y mantener una estabilidad reduciendo su error promedio hasta el valor de 0.252520.

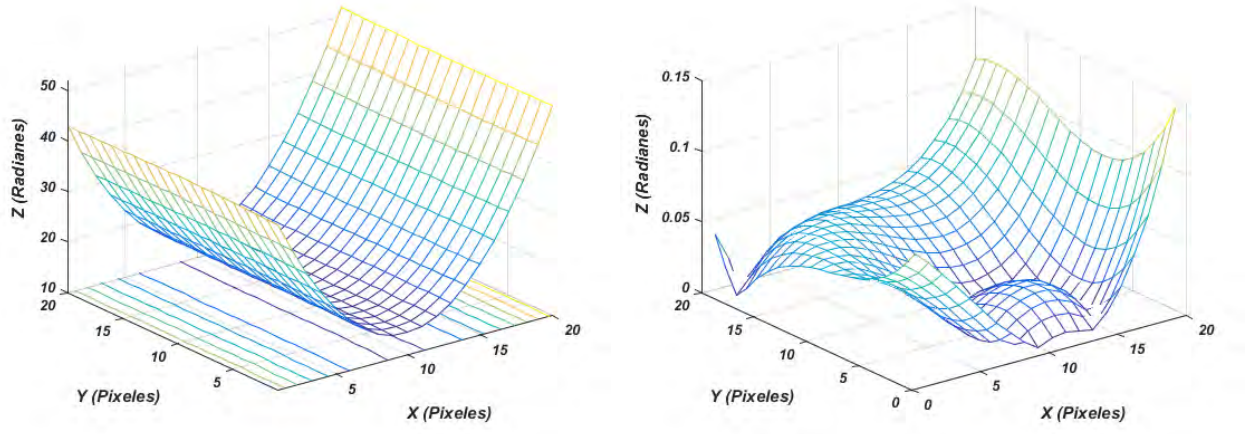
Corridas	Fitness	Error
1	371.184668	0.312467
2	350.581179	0.174965
3	363.551243	0.287163
4	300.167458	0.173424
5	382.526865	0.280411
6	357.551947	0.308525
7	348.116101	0.201150
8	411.304460	0.446958
9	292.832010	0.099902
10	387.584296	0.222624
Promedio	355.454298	0.252520

Tabla 5.29: Resultados de 10 corridas de VMO para la mejor configuración con $N_{IM} = 40$, $N_{TM} = 150$, $k = 9$, $M = 70\ 000$ y $m = 10$ para el interferograma de la Figura 5.16.

De la tabla anterior se interpreta que la mejor ejecución fue la número 9 la cual alcanzó un error de 0.099902. La solución encontrada por VMO viene dada por la siguiente ecuación:

$$\begin{aligned} f(x,y) \;=\; & -0.001893\,y^3 + 0.000479\,x^3 - 0.000541\,x^2\,y - 0.002627\,y^2\,x + 0.000614\,y^2 \\ & -1.446832\,x^2 + 0.001217\,x\,y + 0.013388\,y - 0.155909\,x - 1.936700. \end{aligned} \tag{5.21}$$

Una representación tridimensional de la fase aproximada mediante el polinomio de la Ecuación 5.21 es presentada en la Figura 5.25, así como la diferencia entre la fase aproximada obtenida por VMO y la fase original de la Figura 5.16.



(a) Mapa de fase del objeto a recuperar.

(b) Diferencia entre la fase obtenida por VMO y la original.

Figura 5.25: Mapa y diferencia de fase de la solución obtenida por VMO de la función de prueba 4.

5.6.5. Resultados de VMO para la función de prueba 5.

VMO en esta función también sostuvo una estabilidad en sus resultados debido a que el promedio de error para 10 ejecuciones fue de 0.979923. La siguiente tabla muestra los valores alcanzado por la función objetivo así como los valores de errores cuando se comparan con la fase original.

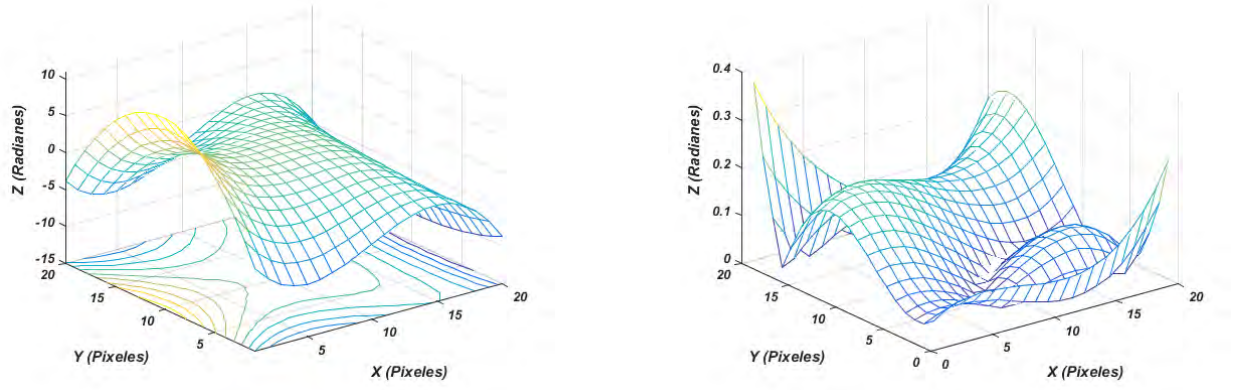
Corridas	Fitness	Error
1	326.893329	0.899522
2	549.669009	1.373472
3	210.134662	0.751236
4	378.186168	1.202258
5	118.793517	0.410440
6	196.506504	1.545797
7	403.125840	0.934140
8	397.524932	1.152262
9	142.514176	0.806316
10	337.232225	1.036935
Promedio	273.002253	0.979923

Tabla 5.30: Resultados de 10 corridas de VMO para la mejor configuración con $N_{IM} = 40$, $N_{TM} = 150$, $k = 9$, $M = 70\ 000$ y $m = 10$ para el interferograma de la Figura 5.18.

El mejor resultado corresponde a la ejecución número 5 con un error de 0.410440, la solución del algoritmo en esta corrida corresponde al siguiente polinomio:

$$\begin{aligned} f(x,y) &= -0.001893\,y^3 + 0.000479\,x^3 - 0.000541\,x^2\,y - 0.002627\,y^2\,x + 0.000614\,y^2 \\ &\quad -1.446832\,x^2 + 0.001217\,x\,y + 0.013388\,y - 0.155909\,x - 1.936700. \end{aligned} \tag{5.22}$$

La representación gráfica que describe el polinomio de la Ecuación 5.22 y la diferencia entre las fases son representadas tridimensionalmente en la siguiente figura:



(a) Mapa de fase del objeto a recuperar.

(b) Diferencia entre la fase obtenida por VMO y la original.

Figura 5.26: Mapa y diferencia de fase de la solución obtenida por VMO de la función de prueba 5.

5.6.6. Resultados de VMO para la función de prueba 6.

Los resultados obtenidos por VMO en la sexta función de prueba se pueden considerar relativamente buenos, su promedio de error fue inferior a la unidad. La Tabla 5.31 muestra los resultados para 10 ejecuciones:

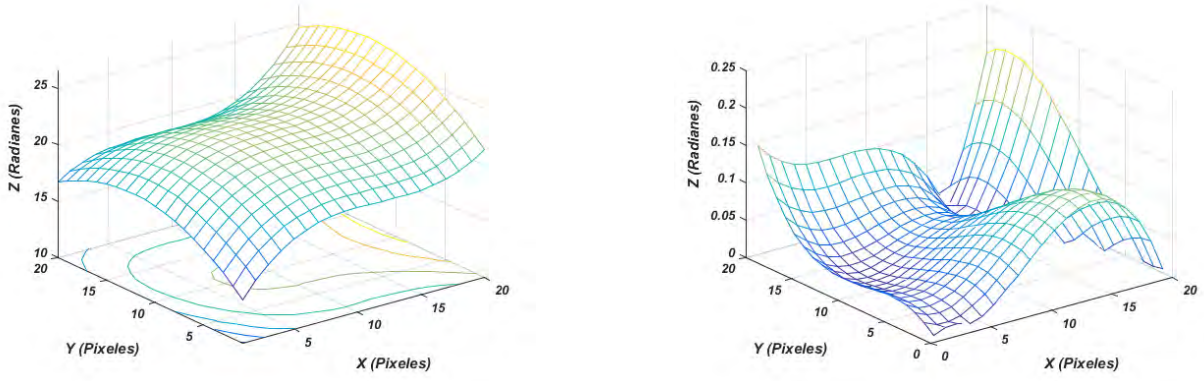
Corridas	Fitness	Error
1	52.733072	0.672503
2	55.445040	1.496875
3	34.389687	0.470482
4	34.189834	0.345212
5	25.224054	0.419455
6	26.389228	0.512810
7	97.666806	0.853596
8	61.880120	0.850602
9	33.809850	0.834871
10	78.078303	0.765641
Promedio	48.953485	0.631583

Tabla 5.31: Resultados de 10 corridas de VMO para la mejor configuración con $N_{IM} = 20$, $N_{TM} = 70$, $k = 9$, $M = 100\ 000$ y $m = 10$ para el interferograma de la Figura 5.20.

La mejor corrida computada en la tabla anterior corresponde a la ejecución número 5 la cual representa la siguiente aproximación polinomial como resultado del algoritmo.

$$\begin{aligned} f(x,y) \;=\; & 0.011196\,y^3 + 0.556762\,x^3 + 0.275897\,x^2\,y - 0.001518\,y^2\,x - 0.944890\,y^2 \\ & -0.135830\,x^2 - 0.022563\,x\,y - 0.037532\,y + 0.041174\,x + 9.720237. \end{aligned} \quad (5.23)$$

La representación tridimensional a la cual corresponde el polinomio de la Ecuación 5.23, así como la diferencia entre la fase original y la recuperada son mostradas en la Figura 5.27.



(a) Mapa de fase del objeto a recuperar.

(b) Diferencia entre la fase obtenida por VMO y la original.

Figura 5.27: Mapa y diferencia de fase de la solución obtenida por VMO de la función de prueba 5.

5.7. Experimento 7: AG para el ajuste de funciones

La última metaheurística utilizada en este trabajo fue un Algoritmo Genético en su codificación real a modo de realizar una comparación entre los diferentes métodos. Al igual que en los experimentos anteriores se seleccionó la variante que arrojó los mejores resultados en cuanto al promedio de error obtenido entre las fases. Para el AG se realizó un estudio previo para seleccionar el método de selección utilizado y para todos los casos los mejores resultados se obtuvieron para la selección de Boltzman. A partir del método de selección se variaron los parámetros del algoritmo hasta obtener la mejor solución. A continuación se muestran los resultados obtenidos en cada una de las funciones de pruebas utilizadas en el proceso de experimentación. Para todas las pruebas se utilizó una probabilidad de cruce igual a 1.

5.7.1. AG para la función de prueba 1

Al igual que en el resto de los experimentos para cada configuración se realizaron un total de 10 ejecuciones. La siguiente tabla muestra el valor de error y la actitud del mejor individuo obtenidos a partir del proceso evolutivo generado por el AG. El tiempo promedio para cada corrida fue de 34.625 segundos, el cual es un tiempo muy bueno para este tipo de problemas.

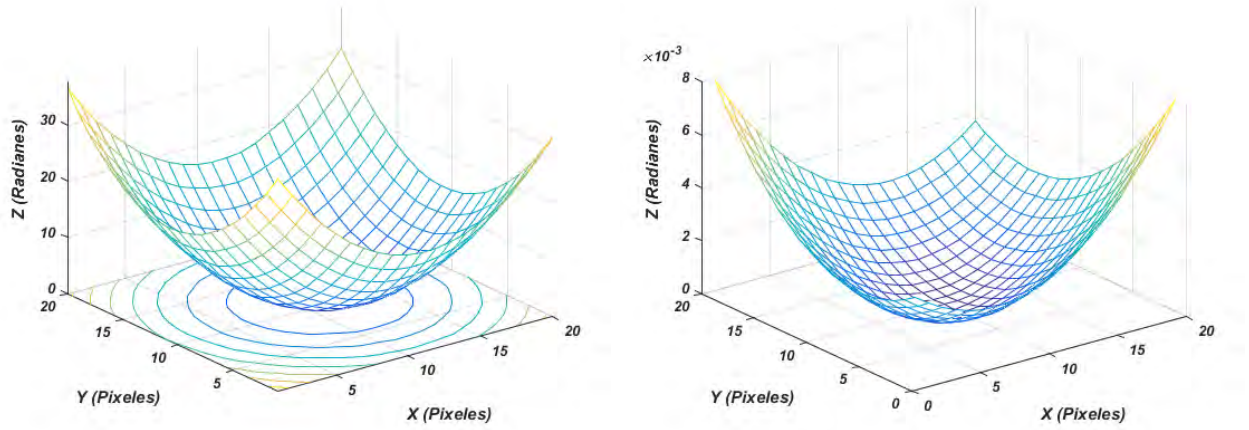
Corridas	**Fitness**	**Error**
1	165.970822	0.010374
2	165.942681	0.011125
3	165.941507	0.009706
4	165.957171	0.006225
5	165.945227	0.011217
6	165.940891	0.010577
7	165.944459	0.010155
8	165.950462	0.012452
9	165.957731	0.005538
10	165.948765	0.011945
Promedio	165.948277	0.009727

Tabla 5.32: Resultados de 10 corridas de un AG para el interferograma de la Figura 5.2, los parámetros utilizados corresponde a $N = 100$ (número de cromosomas de la población inicial), $Iter = 1000$ (número de iteraciones), $T = 9\ 000$ (Temperatura de Boltzman), $P_M = 0.002$ (probabilidad de mutación) y $m = 10$ (factor de suavidad).

La mejor ejecución es la número 9 la cual alcanzó reducir su error hasta un valor de 0.005538. La Ecuación 5.24 representa la aproximación polinomial que entregó como resultado el algoritmo:

$$\begin{aligned} f(x,y) &= 0.083752\,y^2 + 0.083749\,x^2 + 0.000011\,x\,y \\ &\quad -0.167526\,y - 0.335055\,x - 24.752773. \end{aligned} \tag{5.24}$$

La Figura 5.28 muestra la forma recuperada por el AG la cual está dada por la Ecuación 5.24.



(a) Mapa de fase del objeto a recuperar.

(b) Diferencia entre la fase obtenida por VMO y la original.

Figura 5.28: Mapa y diferencia de fase de la solución obtenida por AG de la función de prueba número 1.

De forma general el AG mostró una estabilidad y generó soluciones similares en las 10 ejecuciones reduciendo su promedio de error para todas las corridas hasta 0.009727.

5.7.2. AG para la función de prueba 2

La Tabla 5.33 muestra los resultados alcanzados por la metaheurística para la función de prueba número 2. Se cuantificaron las mejores soluciones obtenidas en 10 ejecuciones, así como el valor de la función de aptitud y su error con respecto al mapa de fase original.

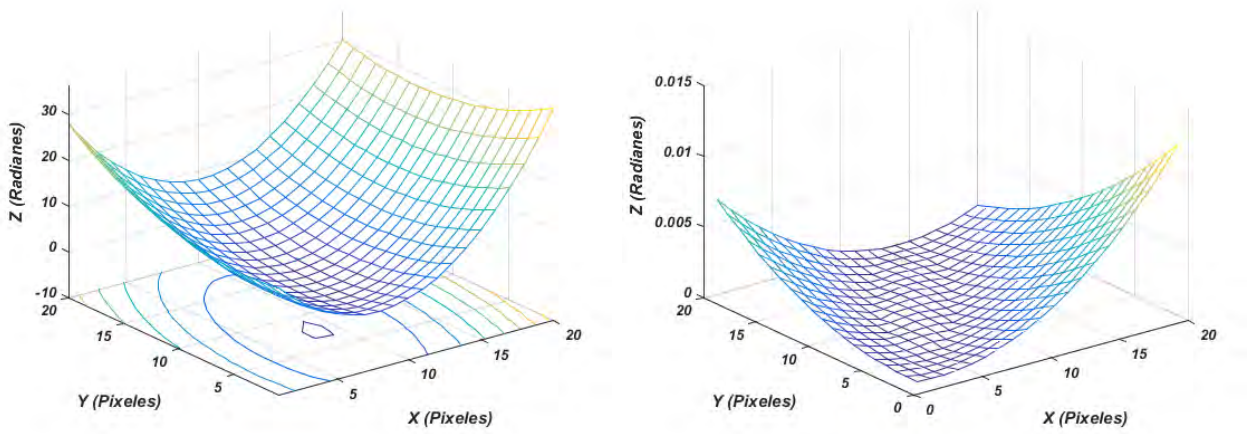
Corridas	Fitness	Error
1	129.301846	0.013653
2	129.197973	0.012119
3	129.157862	0.005134
4	129.153524	0.012277
5	129.217391	0.013785
6	129.210422	0.011368
7	129.147735	0.012338
8	129.156846	0.012584
9	129.186280	0.014667
10	129.132504	0.005764
Promedio	129.170320	0.010989

Tabla 5.33: Resultados de 10 corridas de un AG para el interferograma de la Figura 5.8, los parámetros utilizados corresponde a $N = 400$, $Iter = 1000$, $T = 90\ 000$, $P_M = 0.002$ y $m = 10$.

La mejor aproximación se obtuvo en la tercera ejecución con un error de 0.005134. El tiempo promedio por cada ejecución estuvo aproximadamente alrededor de los 144.375 segundos, debido a que para esta función fue necesario ampliar la población inicial y así el espacio de búsqueda. El resultado obtendido por el AG se muestra en la siguiente ecuación:

$$\begin{aligned} f(x,y) &= 0.083752\,y^2 + 0.083749\,x^2 + 0.000011\,x\,y \\ &\quad -0.167526\,y - 0.335055\,x - 24.752773. \end{aligned} \tag{5.25}$$

La Figura 5.29 indica una representación tridimensional de la Ecuación 5.25 y la diferencia entre el mapa de fase original y el aproximado.



(a) Mapa de fase del objeto a recuperar.

(b) Diferencia entre la fase obtenida por VMO y la original.

Figura 5.29: Mapa y diferencia de fase de la solución obtenida por AG de la función de prueba número 2.

5.7.3. AG para la función de prueba 3

Al igual que en las funciones anteriores el AG fue capaz de encontrar una solución factible mediante una aproximación polinomial de segundo orden. La Tabla 5.34 contiene el valor de la función de aptitud y el error de las aproximaciones encontradas en 10 ejecuciones.

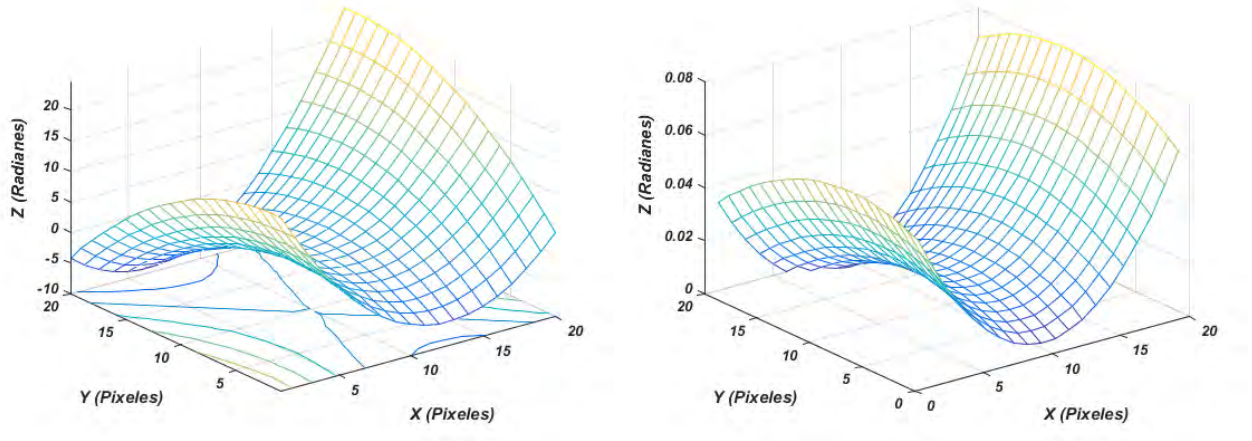
Corridas	Fitness	Error
1	963.709291	0.091595
2	964.183898	0.104753
3	964.078734	0.124772
4	963.693242	0.097896
5	963.808478	0.086141
6	963.672340	0.087622
7	963.774490	0.085475
8	968.607395	0.131918
9	963.675962	0.089423
10	963.839062	0.115101
Promedio	964.393713	0.102294

Tabla 5.34: Resultados de 10 corridas de un AG para el interferograma de la Figura 5.11, los parámetros utilizados corresponde a $N = 200$, $Iter = 1000$, $T = 900\ 000$, $P_M = 0.02$ y $m = 10$.

El tiempo promedio por ejecución fue de aproximadamente 79 segundos por ejecución. De la tabla anterior se concluye que la mejor aproximación obtenida por el AG fue en la séptima corrida, la cual representa la siguiente aproximación de segundo grado:

$$\begin{aligned} f(x,y) &= 0.002610\,y^2 - 0.00724\,5x^2 - 0.004794\,x\,y \\ &\quad +0.006940\,y - 0.02607\,x - 0.039859. \end{aligned} \tag{5.26}$$

El mapa de fase correspondiente a la Ecuación 5.26 y la diferencia entre el mapa de fase original y el aproximado se muestra en la siguiente Figura 5.30:



(a) Mapa de fase del objeto a recuperar.

(b) Diferencia entre la fase obtenida por VMO y la original.

Figura 5.30: Mapa y diferencia de fase de la solución obtenida por AG de la función de prueba número 3.

5.7.4. AG para la función de prueba 4

La función de prueba número 4 representa un polinomio de tercer grado. Al igual que en los otros experimento el AG utilizó los mismos intervalos de búsqueda para encontrar los resultados que se muestran en la Tabla 5.35.

Corridas	Fitness	Error
1	285.969145	0.012194
2	285.695491	0.009817
3	285.769637	0.020445
4	285.681296	0.011910
5	285.675191	0.032603
6	286.242256	0.022624
7	285.623346	0.017437
8	285.722042	0.028347
9	285.654315	0.010406
10	285.805187	0.011302
Promedio	285.771659	0.019384

Tabla 5.35: Resultados de 10 corridas de un AG para el interferograma de la Figura 5.16, los parámetros utilizados corresponde a $N = 200$, $Iter = 1000$, $T = 90\ 000$, $P_M = 0.002$ y $m = 10$.

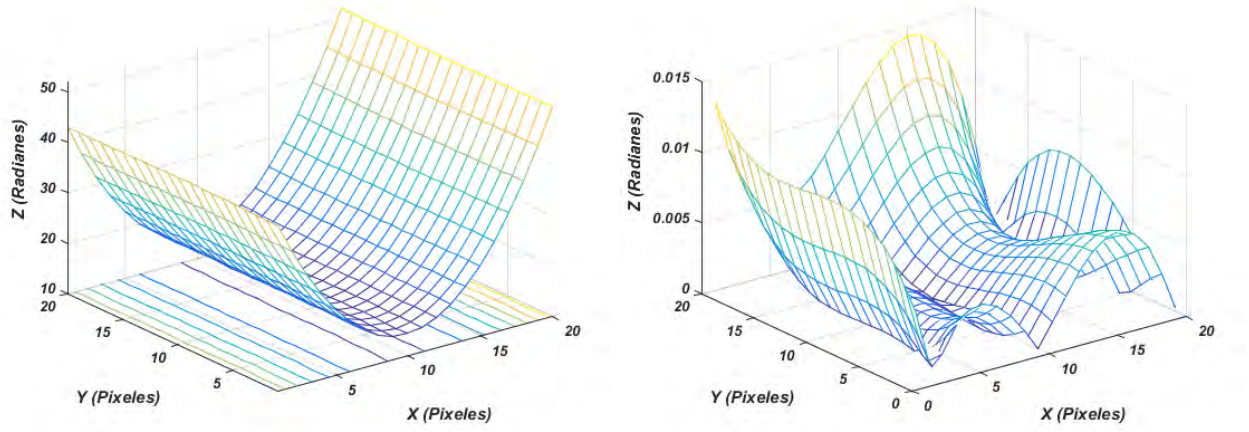
La mejor solución fue la aproximación encontrada en la segunda corrida, aunque de forma general el algoritmo logró converger a una solución factible. La Ecuación 5.27 representa el polinomio de grado 3 encontrado por el AG en la tercera ejecución:

$$\begin{aligned} f(x,y) &= -0.001403\,y^3 - 0.000313\,x^3 - 0.000009\,x^2\,y - 0.002847\,y^2\,x + 0.000155\,y^2 \\ &\quad -1.449043\,x^2 - 0.000049\,x\,y - 0.000175\,y - 0.142992\,x - 20.776504. \end{aligned} \quad (5.27)$$

Las ejecuciones promediaron un tiempo de 120.125 segundos, y de forma general todas convergieron a soluciones similares. La Figura 5.31 muestra el mapa de fase encontrado por el AG en la segunda ejecución y la diferencia entre el mapa de fase original y el encontrado por el algoritmo.

5.7.5. AG para la función de prueba 5

Esta función representa un polinomio de tercer grado, que al igual que la función 5 se necesita un mayor número de coeficientes lo que hace más engorroso el proceso de búsqueda del AG. La Tabla 5.36 muestra los resultados de 10 ejecuciones utilizando un AG y como función de entrada el interferograma correspondiente a la Figura 5.18.



(a) Mapa de fase del objeto a recuperar.

(b) Diferencia entre la fase obtenida por VMO y la original.

Figura 5.31: Mapa y diferencia de fase de la solución obtenida por AG de la función de prueba número 4.

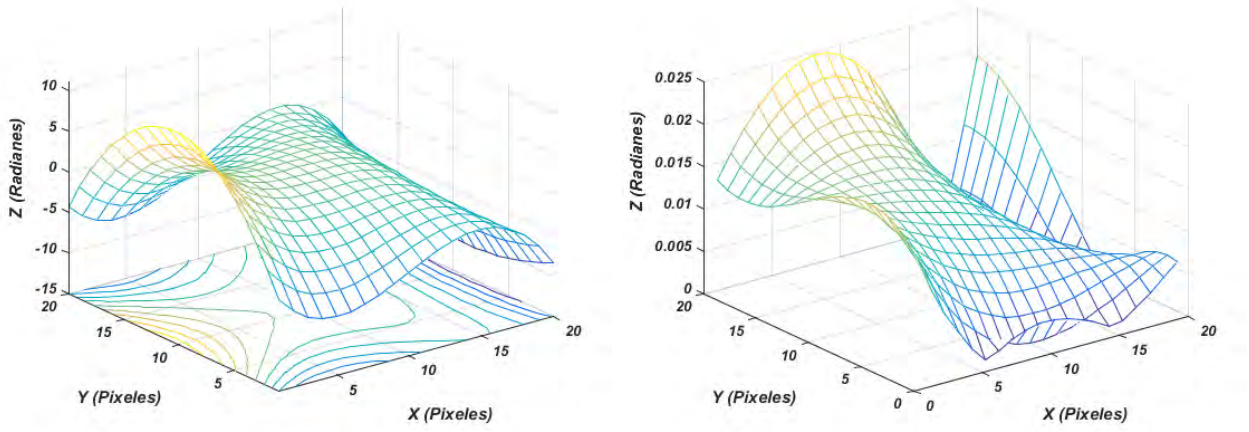
Corridas	**Fitness**	**Error**
1	54.481724	0.102797
2	62.040554	0.129961
3	53.802526	0.047396
4	72.742631	0.253615
5	93.594564	0.243072
6	55.232243	0.092857
7	102.226598	0.509442
8	54.249107	0.090711
9	54.853655	0.062340
10	54.256238	0.124892
Promedio	67.619695	0.178041

Tabla 5.36: Resultados de 10 corridas de un AG para el interferograma de la Figura 5.18, los parámetros utilizados corresponde a $N = 200$, $Iter = 1000$, $T = 900\ 000$, $P_M = 0.002$ y $m = 10$.

El mejor resultado fue el encontrado en la corrida número 3 con un error de 0.047396. Aproximadamente cada ejecución requirió de 121 segundos para encontrar una aproximación polinomial. La Ecuación 5.28 representa la solución encontrada en la tercera corrida:

$$\begin{aligned} f(x,y) &= 0.017238\,y^3 - 1.585248\,x^3 + 0.002913\,x^2\,y + 1.421822\,y^2\,x - 1.639701\,y^2 \\ &\quad +0.001358\,x^2 + 0.019290\,x\,y + 0.689066\,y + 0.002671\,x - 17.030626. \end{aligned} \quad (5.28)$$

La representación gráfica de la mejor solución encontrada por el AG para esta función se muestra en la siguiente figura, además la Figura 5.32b muestra la diferencia entre la fase original y la encontrada por el algoritmo.



(a) Mapa de fase del objeto a recuperar.

(b) Diferencia entre la fase obtenida por VMO y la original.

Figura 5.32: Mapa y diferencia de fase de la solución obtenida por AG de la función de prueba número 5.

5.7.6. AG para la función de prueba 6

Los resultados del AG para la última función del estudio experimental están representados en la siguiente tabla:

Corridas	Fitness	Error
1	9.949611	0.055582
2	9.932017	0.083988
3	10.003066	0.057154
4	9.863623	0.015990
5	9.892921	0.035918
6	9.828337	0.012121
7	9.897446	0.095530
8	9.994860	0.055599
9	9.921683	0.038555
10	9.844549	0.015379
Promedio	9.891917	0.038441

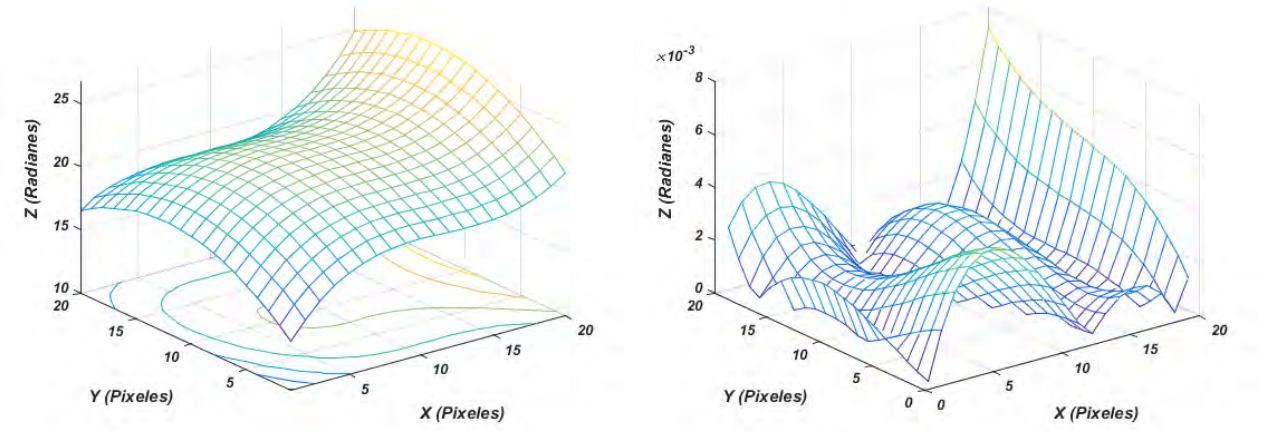
Tabla 5.37: Resultados de 10 corridas de un AG para el interferograma de la Figura 5.20, los parámetros utilizados corresponde a $N = 400$, $Iter = 1000$, $T = 90\ 000$, $P_M = 0.002$ y $m = 10$.

A pesar de la complejidad de la función y del intervalo de búsqueda ampliado el AG logró converger a una buena solución demostrando una estabilidad para el conjunto de 10 ejecuciones. En promedio cada corrida necesitó un total de 247 segundos para encontrar la solución. La mejor ejecución fue la número 6 la cual alcanzó un error de 0.012121. La

Ecuación 5.29 representa el polinomio encontrado por el algoritmo.

$$\begin{aligned} f(x,y) &= -0.000239\,y^3 + 0.575257\,x^3 + 0.277893\,x^2\,y + 0.000247\,y^2\,x - 0.953571\,y^2 \\ &\quad -0.113388\,x^2 - 0.000216\,x\,y + 0.001426\,y + 0.002788\,x - 15.432729. \end{aligned} \quad (5.29)$$

La representación tridimensional del mapa de fase encontrado por el AG y la diferencias entre las fases son mostrada en la siguiente figura:



(a) Mapa de fase del objeto a recuperar.

(b) Diferencia entre la fase obtenida por VMO y la original.

Figura 5.33: Mapa y diferencia de fase de la solución obtenida por AG de la función de prueba número 6.

5.8. Comparación de los resultados

Esta sección aborda de forma general el desempeño de cada metaheurística y hace una comparción entre ellas. Cada función de prueba será analizada de forma independiente para determinar cual metaheurística obtuvo los mejores resultados. La siguiente tabla muestra los resultados obtenidos por cada uno de los algoritmos para las funciones de segundo grado.

Función 1	**RS**		**VMO**		**AG**	
	Fitness	**Error**	**Fitness**	**Error**	**Fitness**	**Error**
Mejor Corrida	1668.840209	0.081586	171.730900	0.099519	165.957731	0.005538
Promedio	1674.281304	0.1374846	197.519613	0.214013	165.948277	0.009727
Función 2	**RS**		**VMO**		**AG**	
	Fitness	**Error**	**Fitness**	**Error**	**Fitness**	**Error**
Mejor Corrida	1297.725474	0.076201	1357.049422	0.190836	129.157862	0.005134
Promedio	1339.159398	0.217902	2737.064860	2.784924	129.170320	0.010989
Función 3	**RS**		**VMO**		**AG**	
	Fitness	**Error**	**Fitness**	**Error**	**Fitness**	**Error**
Mejor Corrida	1007.342767	0.137010	278.273109	0.289327	963.774490	0.085475
Promedio	998.9276198	0.290392	428.050236	4.438375	964.393713	0.102294

Tabla 5.38: Comparación entre las 3 metaheurística para la funciones de segundo grado.

Análogamente la Tabla 5.39 muestra una comparación entre las 3 metaheurísticas para las funciones de mayor orden y más complejas para el proceso de demodulación.

Función 4	RS		VMO		AG	
	Fitness	Error	Fitness	Error	Fitness	Error
Mejor Corrida	330.660849	0.151804	292.832010	0.099902	285.695491	0.009817
Promedio	345.693612	0.205137	355.454298	0.252520	285.771659	0.019384

Función 5	RS		VMO		AG	
	Fitness	Error	Fitness	Error	Fitness	Error
Mejor Corrida	153.525024	118.793517	0.410440	0.190836	53.802526	0.047396
Promedio	189.771019	0.671859	273.002253	0.979923	67.6196950	0.178041

Función 6	RS		VMO		AG	
	Fitness	Error	Fitness	Error	Fitness	Error
Mejor Corrida	58.942886	0.541819	25.224054	0.419455	9.828337	0.012121
Promedio	96.800584	1.040529	48.953485	0.631583	9.891917	0.038441

Tabla 5.39: Comparación entre las 3 metaheurística para la funciones de tercer grado.

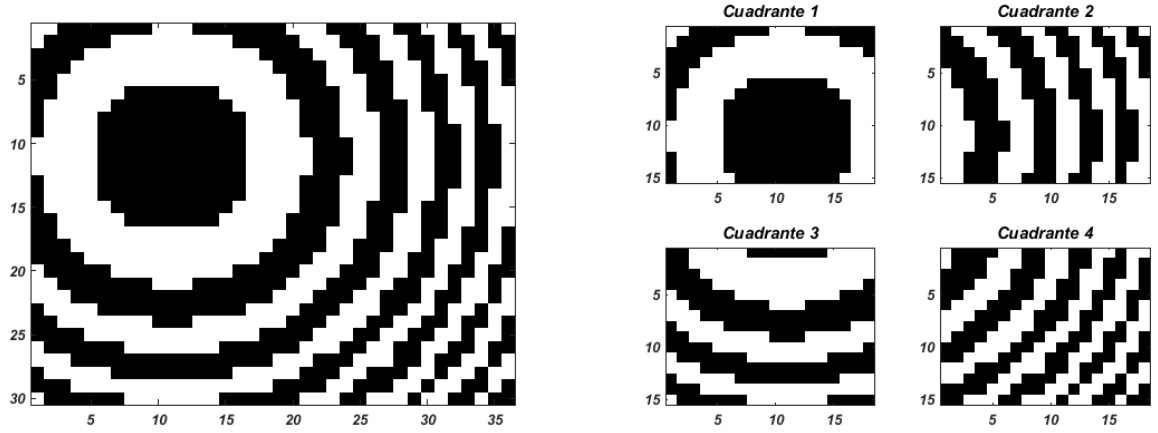
De forma general el algoritmo que alcanzó los mejores resultados para ambos grupos de funciones fue el Algoritmo Genético, mostrando una estabilidad y las mejores aproximaciones. A pesar de la superioridad del AG no se puede despreciar el desempeño de las otras 2 metaheurísticas que de igual forma fueron capaces de demodular los interferogramas con resultados muy alentadores. En conclusión a pesar de que el AG fue superior tanto VMO como RS lograron encontrar buenas aproximaciones en todas las funciones con errores inferiores a la unidad.

5.9. Experimento 8: Demodulación de interferogramas mediante ventana usando un RS

Este último experimento se realizó con el objetivo de sentar bases para trabajos futuros. Básicamente consiste en demodular interferogramas de mayor resolución aplicando una técnica de divide y vencerás, es decir la idea fundamental es dividir el interferograma en pequeñas subimágenes y demodularlo de forma independiente, y al final realizar un proceso de reconstrucción. Este método tiene como ventaja que uno puede subdividir el interferograma en cuantas subimágenes o instancias se quiera.

Para probar este método se simuló un interferograma de resolución 30 x 36 pixeles, este a su vez fue dividido en 4 cuadrantes, los cuales cada uno de forma independiente sirvieron de

entrada para las meteheurísticas. Primeramente la imagen fue sometida a un procesamiento para eliminar algún tipo de ruido, es decir, se binarizó el interferograma inicial utilizando el método de Otsu para encontrar el umbral adecuado, para ello se utilizó la implementación de Matlab R1204b. La Figura 5.34 muestra el interferograma binarizado y la división de los cuadrantes utilizados.

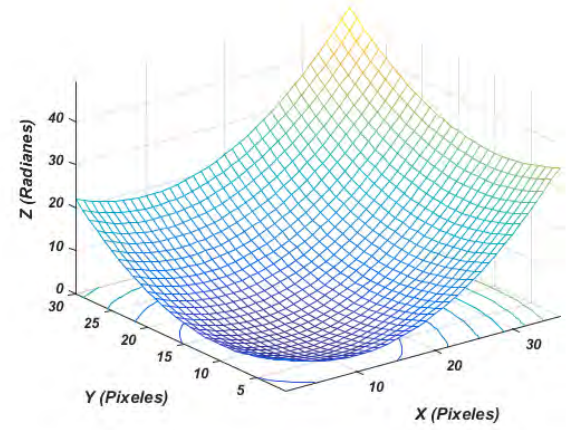


(a) Interferograma simulado.

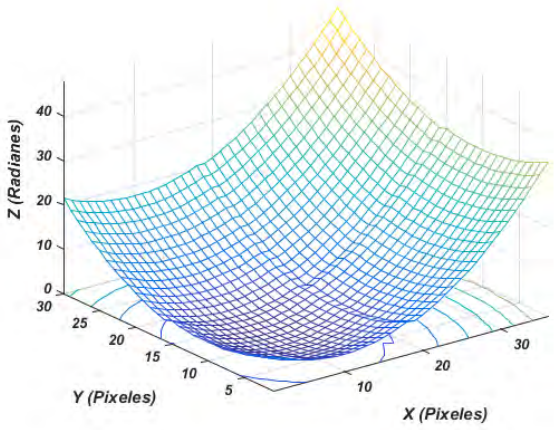
(b) División en cuadradrantes del interferograma simulado.

Figura 5.34: Interferograma binarizado y mapa de fase para el interferograma simulado para utilizarlo como entrada para el RS utilizado división de cuadrantes.

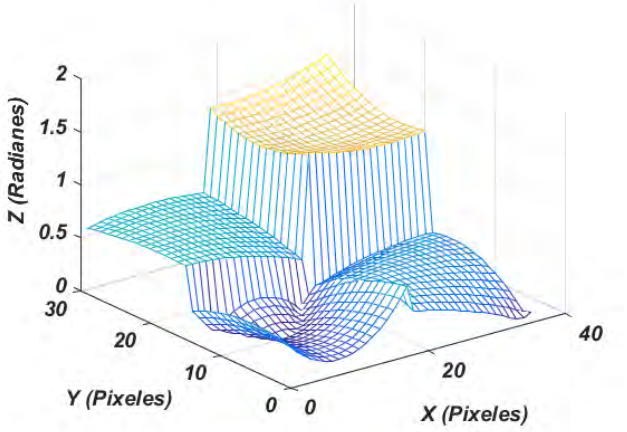
Como resultado del algoritmo de RS para cada cuadrante del interferograma mostramos la Figura 5.35, donde se expone los resultados finales , es decir, el resultado del proceso de unificación de cada cuadrante.



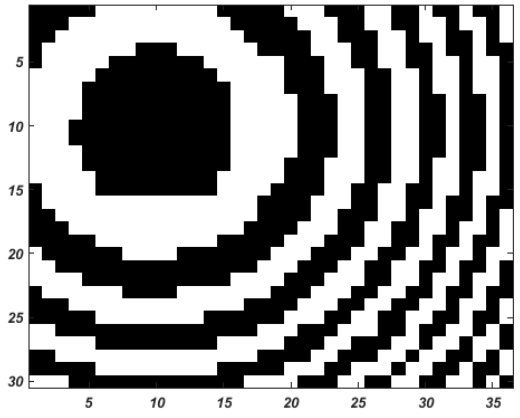
(a) Mapa de fase original.



(b) Mapa de fase obtenido por la técnida de división de ventanas.



(c) Diferenica entre el mapa original y el aproximado.

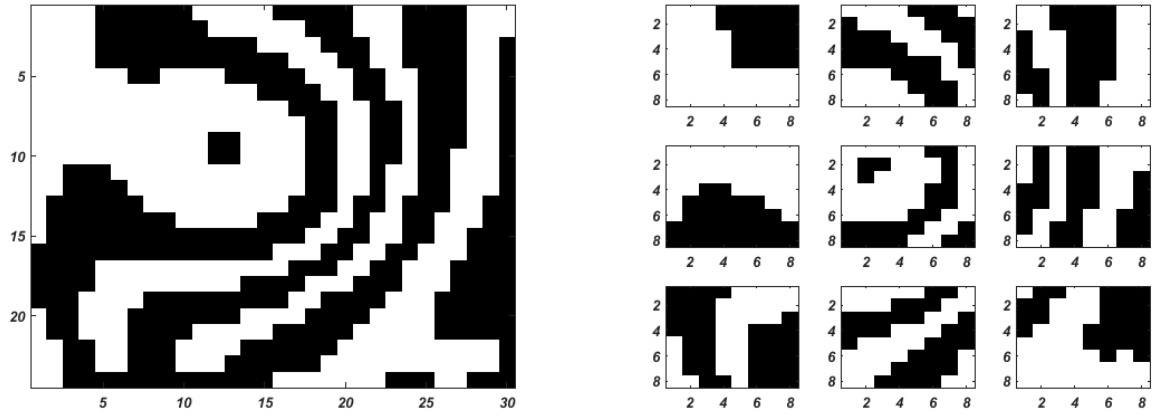


(d) Interferograma recuperado por la técnica.

Figura 5.35: Resultados obtenidos al dividir el ingterferograma original en cuadrantes.

Los resultados que se muestran en la figura anterior, evidencia la capacidad de esta técnica de divide y vencerás utilizando una metaheurística , es capaz de demodular interferogramas de mayor resolución. Una de las limitaciones de esta técnica es el proceso de unificación de las partes, lo cual se convierte en un reto para trabajos futuros. El Apendice L muestra el algoritmo utilizado para el proceso de unificación de los cuadrantes.

Este mismo proceso se utilizó para un interferograma real, al igual que en el proceso anterior la imagen fue binarizada y dividida en cuadrantes. La Figura 5.36 muestra la imagen binarizada y la división de las subimagenes.

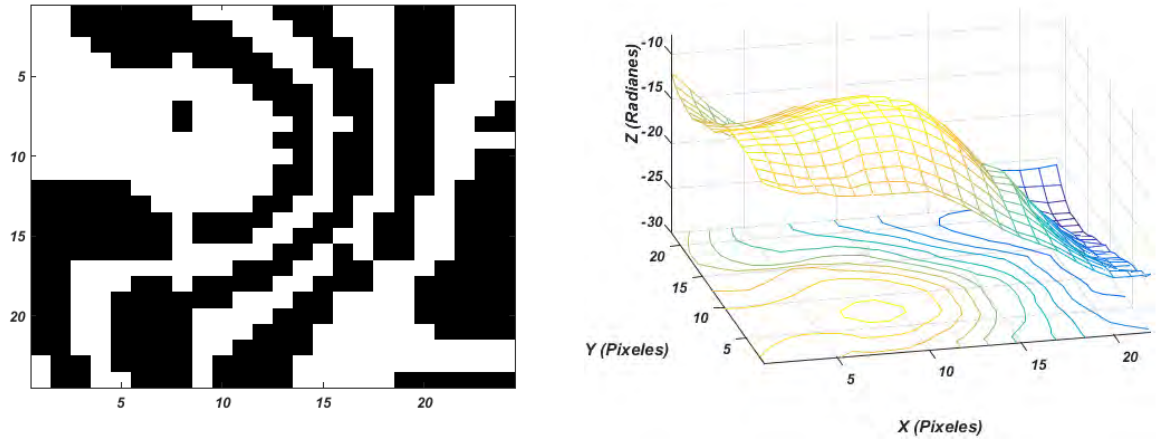


(a) Interferograma real

(b) División en cuadrantes del interferograma original.

Figura 5.36: Interferograma real utilizado en la técnica de demodulación mediante ventanas.

Los resultados de esta técnica se muestran en la siguiente figura.



(a) Interferograma recuperado por el RS

(b) Fase recuperada por el RS utilizando la técnica de división de cuadrnates.

Figura 5.37: Resultado obtenido por el RS utilizando en la técnica de demodulación mediante ventanas.

Aunque no se pudo contar con un análisis perfilométrico del objeto en estudio se pudo comprobar que la técnica fue capaz de aproximar el interferograma y mantener el mismo número de franjas, lo que indica que el mapa de fase recuperado en dimensión es lo suficientemente aproximado al real. Este resultado demuestra la capacidad de las metaheurísticas enfocados en este tipo de problemas, lo que implica un punto de interés para trabajos futuros.

Otro ejemplo de una imagen real utilizando esta técnica fue un objeto semi-esférico localizado detrás de una rejilla de Ronchi el cual fue iluminado con una fuente de luz blanca colimada, la Figura 5.38 muestra la imagen obtenida después de un proceso de filtrado donde se elimina la propagación rectilínea de la luz.

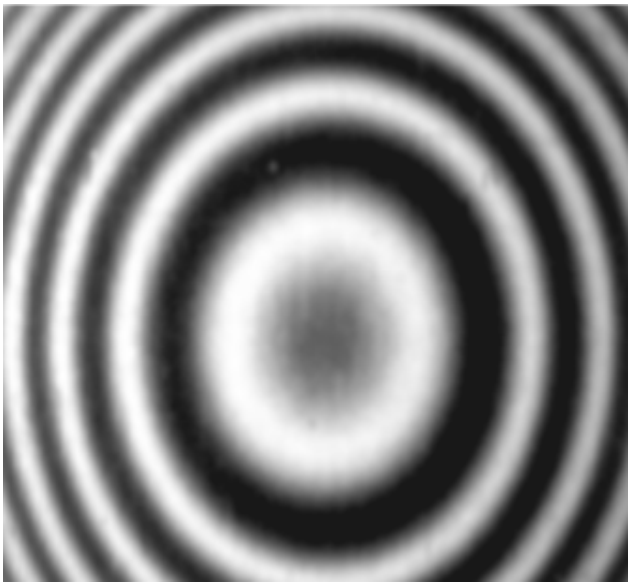
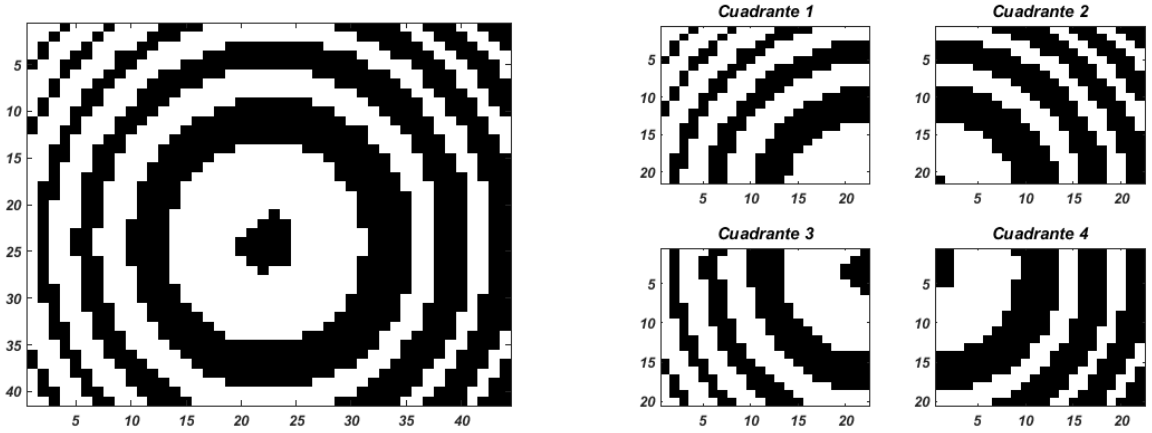


Figura 5.38: Patrón de franjas generado por el objeto semi-esférico con una resolución de 255 x 271 pixeles.

Para encontrar la fase del objeto anterior utilizamos AGs y VMO con dos interferogramas de resoluciones diferentes. Para el AG utilizamos un interferograma de resolución de 41 x 44 pixeles. La Figura 5.39 muestra el interferograma reducido y la división en cuadrantes que se utilizó como entrada al Algoritmo Genético.



(a) Interferograma binarizado y reducido el utilizó para obtener los cuadrantes que constituyeron la entrada del AG.

(b) División en cuadrantes.

Figura 5.39: Patrón de franjas binarizado utilizado en la recuperación del mapa de fase del objeto semi-esférico.

La Figura 5.40 muestra el mapa de fase recuperado por el AG donde se evidencia que tiene una forma esférica similar a la del objeto.

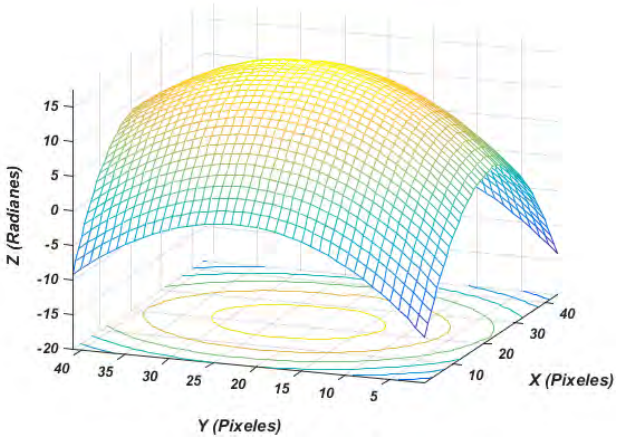
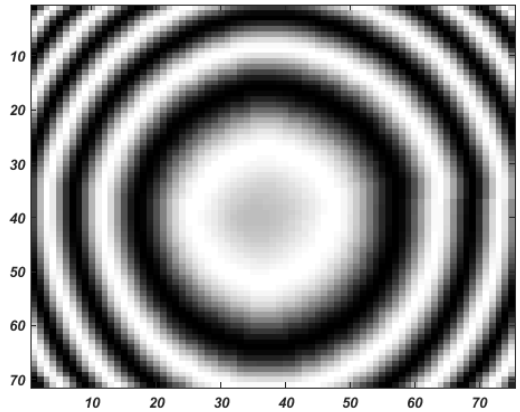
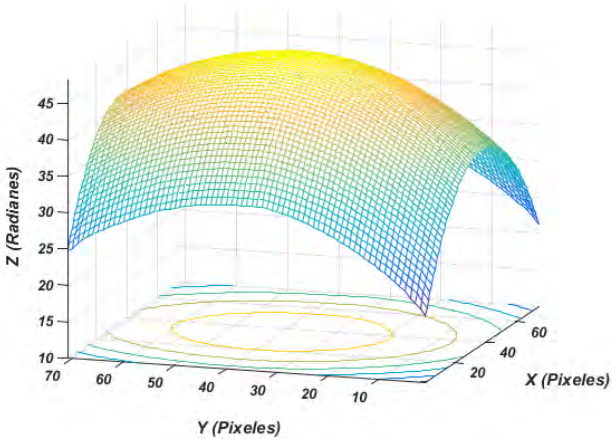


Figura 5.40: Mapa de fase del objeto semi-esférico encontrado por el AG.

Análogamente se realizó el mismo experimento con la diferencia de que se utilizó VMO para obtener la fase del objeto, y además la imagen de entrada tenía mayor resolución que la utilizada en el AG. La Figura 5.41 muestra el interferograma y el mapa de fase recuperado por el algoritmo.



(a) Patrón de franjas recuperado por VMO con resolución de 78 x 81 píxeles.

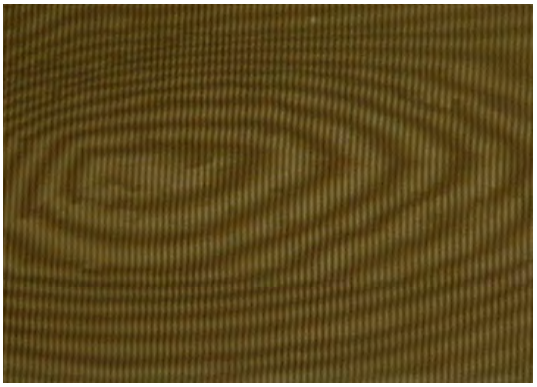


(b) Mapa de fase recuperado por VMO.

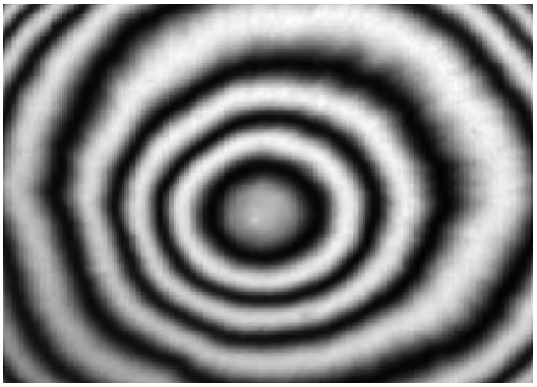
Figura 5.41: Patrón de franjas y el mapa de fase del objeto semi-esférico recuperados por VMO.

Tanto AG como VMO aproximaron con suficiente similitud la forma del objeto en cuestión. Este experimento evidencia la capacidad de las metaheurísticas en este tipo de problemas.

Los últimos experimentos relacionados con imágenes reales involucran la siguiente figura donde se muestra los interferogramas obtenidos del arreglo experimental después del proceso de filtrado donde se elimina en gran medida el efecto del ruido.



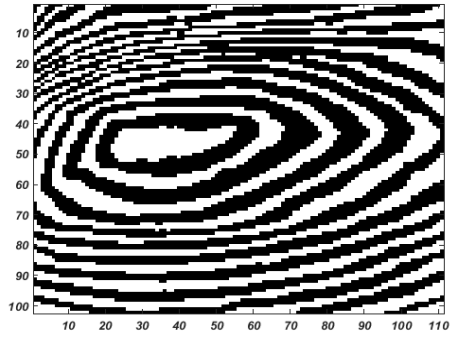
(a) Patrón de franjas original utilizando moteado de Moiré con una resolución de 343 x 381 pixele.



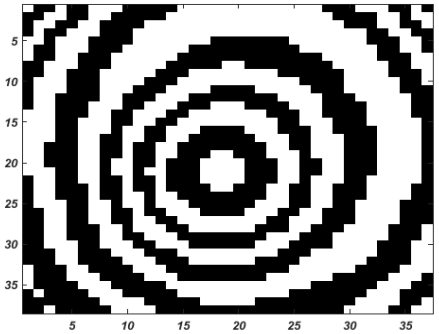
(b) Patrón de franjas obtenido en el arreglo experimental con una resolución de 533 x 521 pixeles.

Figura 5.42: Patrones de franjas originales obtenidos en el arreglo experimental.

Para ambas imágenes fue necesario reducir la resolución y binarizar las imágenes de la Figura 5.42 para obtener un mejor desempeño de las metaheurísticas y hacer constante la amplitud de las franjas. La Figura 5.43 muestra el resultado después de aplicar este proceso.



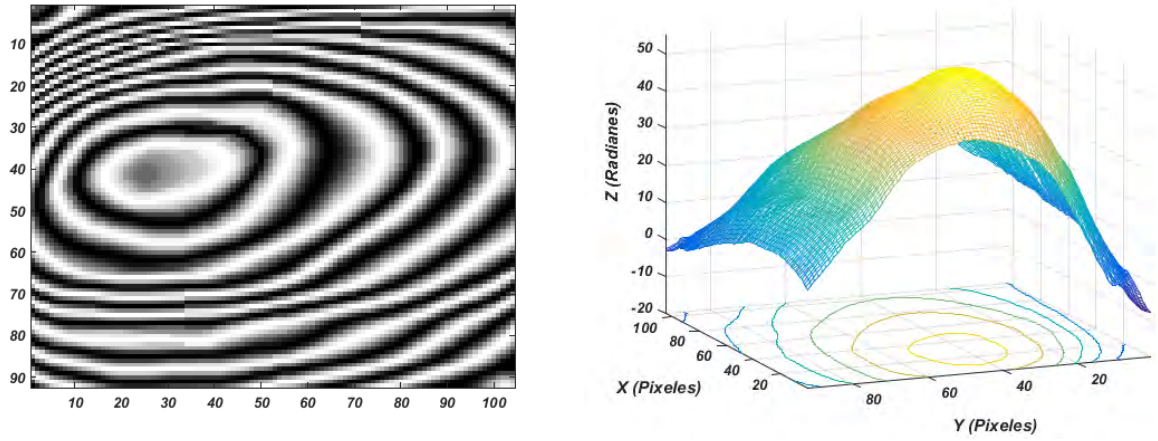
(a) Patrón de franjas binarizado correspondiente a la Figura 5.42a con una resolución de 102 x 111 pixeles.



(b) Patrón de franjas binarizado correspondiente a la Figura 5.42b con una resolución de 38 x 37 pixeles.

Figura 5.43: Patrones de franjas utilizados por las metaheurísticas en el proceso de demodulación.

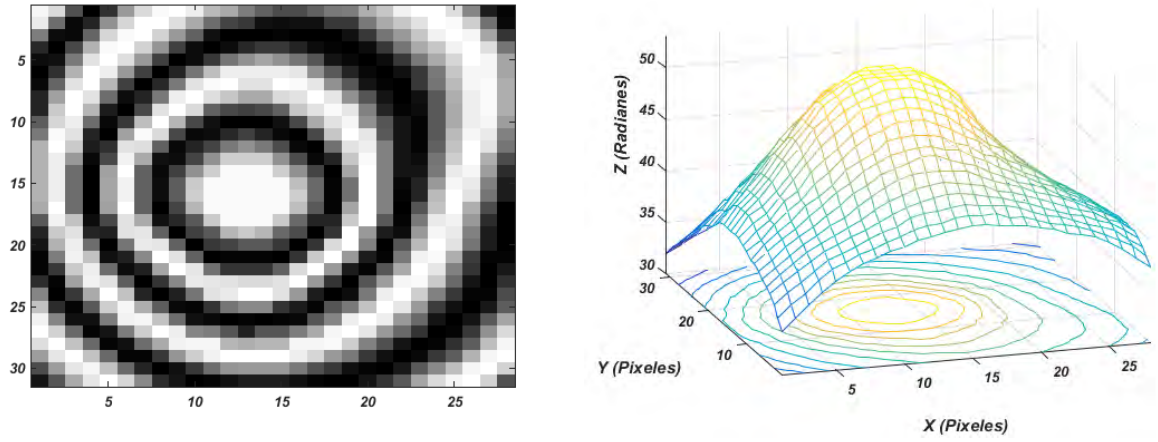
La Figura 5.43a fue dividida en 36 cuadrantes los cuales se utilizaron como entrada para las metaheurísticas. La Figura 5.44 muestra los resultados obtenidos después de aplicar el proceso de unificación de los cuadrantes.



(a) Interferograma aproximado. (b) Mapa de fase aproximada.

Figura 5.44: Resultados obtenidos por VMO utilizando la técnica de división de cuadrantes para el interferograma de la Figura 5.43a.

Por último la Figura 5.43b fue dividida en 9 cuadrantes, los cuales al igual que la anterior fueron utilizados de forma independiente como entradas al algoritmo de optimización. La Figura 5.45 muestra los resultados obtenidos por VMO:



(a) Interferograma aproximado. (b) Mapa de fase aproximada.

Figura 5.45: Resultados obtenidos por VMO utilizando la técnica de división de cuadrantes para el interferograma de la Figura 5.43b.

Este experimento reafirma la capacidad de recuperar el término de fase a partir de interferogramas reales. Esta técnica tiene como ventaja que se puede reducir el interferograma hasta la resolución adecuada donde el algoritmo de optimización pueda converger a una solución factible en un intervalo de tiempo razonable, y a la hora de recuperarlo, podemos escalar el término de fase tanto como se desee.

Capítulo 6

Conclusiones y Trabajos a Futuro

Como resultado de esta investigación se demostró que la metaheurística poblacional Optimización basada en Mallas Variables y el algoritmo de Recocido Simulado son posible modelarlo como un problema para la demodulación de interferogramas mediante una aproximación polinomial con ajuste parámetrico. Se modeló además una variante de Algroritmo Genético en su codificación real con el objetivo de llevar a cabo un proceso de comparación. Se utilizó un conocimiento a priori con el objetivo de restringir el espacio de búsqueda y evitar que las metaheurísticas no se pierdan en mínimos locales. Se probó que estas técnicas pueden encontrar soluciones factibles, incluso cuando se incrementa el espacio solución.

Esta técnica de demodulación de ajuste de funciones mediante heurísticas tiene la ventaja de requerir una única imagen en comparación con el conocido método de desplazamiento de fase, el cual, necesita al menos 3 imágenes con un desplazamiento en fase conocido. Otra ventaja de esta técnica es la posibilidad de demodular los interferogramas con la presencia de franjas cerradas y no requiere de ningún algoritmo de desenvolvimiento de fase, ya que la aproximación se refiere directamente al término de fase.

La mejor metaheurística que ofreció los mejores resultados fue el AG con selección de Boltzman. Este algoritmo demostró su solidez y capacidad para este tipo de problemática. Adicionalmente VMO y RS demostraron su capacidad de converger a buenas soluciones.

6.1. Trabajos futuros

Hoy día uno de los principales retos de la ingeniería consiste en obtener una medición precisa fundamentalmente en eventos transcientes, donde se dificulta en gran medida el registro de una medición. Es por ello que uno de los desafíos en Metrología Óptica es encontrar el mapa de fase a partir de un interferograma en tiempo real. Es por ello que uno de las motivaciones consiste en el diseño e implementación de una aplicación a nivel de hardware de tal forma que pueda entregar mediciones en tiempo real.

No es menos cierto que esta técnica causa novedad en la resolución o enfoque para este tipo de problema. Es por ello que uno de los objetivos de la tesis era probar las funciona-

lidades de algunas metaheurísticas, con el objetivo de demodular en un futuro la fase de un interferograma mediante el uso de hiper-heurística, es decir seleccionar, combinar, generar o adaptar varias heurísticas más simples para resolver eficientemente problemas de búsqueda computacional. Una de las motivaciones para estudiar la hiper-heurística es construir sistemas que puedan manejar clases de problemas en lugar de resolver un solo problema.

Otra línea de investigación a futuro es la paralelización de los algoritmos, con el fin de disminuir en tiempo y además encontrar soluciones para interferogramas de mayor resolución. Es decir, aplicar una técnica de divide y vencerás particionando el interferograma en pequeñas subimágenes con un menor número de franjas y hacer que las técnicas antes mencionadas alcance un mejor desempeño y que el proceso de demodulación sea menos engorroso.

Otro enfoque de investigación sería la implementación de un algoritmo híbrido donde se combinen dos o más algoritmos que solucionan el mismo problema, ya sea escogiendo uno (a merced de los datos), o cambiando entre ellos sobre el curso del algoritmo. Esto es hecho generalmente para combinar características deseadas de cada uno, a fin de que el algoritmo global sea mejor que los componentes individuales.

Por último trabajar en la obtención de una función objetivo que ofrezca mayores bondades a la hora de enfrentar un problema tan complejo como el de demodulación de fase a partir de interferogramas.

Apéndice A

Método para evaluar un punto en un polinomio de grado n

```
1     /**
2      * METODO QUE CALCULA EL VALOR DE LA FASE A PARTIR DEL VECTOR DE
3      * COEFICIENTES
4      */
5     public static double Phase_Funtion(double[] coef, double x, double y) {
6         double phase = coef[0];
7         int k, exp, c = 0;
8         int grado = (int) (-3 + Math.sqrt(9 + 8 * (coef.length - 1))) / 2;
9         for (int i = 1; i <= grado; i++) {
10            k = 0;
11            for (int j = i + c; j <= 2 * i + c; j++) {
12                exp = (i + k + 2) % (i + 1);
13                phase += coef[j] * Math.pow(x, exp) * Math.pow(y, i - exp);
14                k++;
15            }
16            c += i;
17        }
18        return phase;
19    }
```

Apéndice B

Método para sumar dos arreglos binarios

```
1     /*
2      * Suma dos numeros binarios.
3      * Recibe dos arreglos de tipo entero y retorna un
4      * arreglo de tipo entero con el resultado de la suma.
5      */
6     public static int[] sumBinAndBin(int a[], int b[]) {
7         int acarreo = 0;//variable de control.
8         int[] c = new int[a.length];//Arreglo a retornar.
9         //Se recorre el arreglo a de manera inversa.
10        for (int i = a.length - 1; i > -1; i--) {
11            int suma = a[i] + b[i]; //Se realiza la suma una vez por ciclo
12            if (suma == 2) {//Si la suma da 2 los elementos en el arreglo son 1 y 1.
13                c[i] = 0;//Por lo tanto debe ir como resultado un 0
14                acarreo = 1;//Y se lleva un acarreo
15            } //Si por el contrario la suma es 0 quiere decir que los elementos
16            //eran 0 y 0.
17            else if (suma == 0) {
18                c[i] = 0;//Entonces debe ir 0 como resultado
19            } //Si por el contrario la suma es 1, entonces los elementos
20            //eran 0 y 1 o viceversa.
21            else if (suma == 1) {
22                c[i] = 1;//Y debe ir como resultado un 1.
23            }
24            if (acarreo == 1) {//Si existe un acarreo en este ciclo
25                if (i == 0) { // Excedio el intervalo y geenro un numero Aleatorio
26                    c = Binary.Random_Array_Bin(a.length);
27                    break;
28                } else {
29                    //Se comprueba si la suma del elemento siguiente del
30                    //arreglo "a" mas el acarreo es igual a 2.
31                    if (a[i - 1] + 1 == 2) {
32                        a[i - 1] = 0;//Si lo es, se asigna un 0 a "a".
33                        acarreo = 1;//Y se sigue llevando un acarreo.
34                    } else {//Si la suma no da 2, solo puede dar 1.
35                        a[i - 1] = 1;//Y entonces se asigna 1 a "a".
36                        acarreo = 0;//Y el acarreo desaparece.
37                    }
```

```
38                  }
39              }
40
41          }
42          return c;//Se regresa el arreglo.
43      }
```

Apéndice C

Método para restar dos arreglos binarios

```
1     /*
2      * Resta dos numeros binarios.
3      * Recibe dos arreglos de tipo entero y retorna un
4      * arreglo de tipo entero con el resultado de la suma.
5      */
6     public static int[] restBinAndBin(int a[], int b[]) {
7         int acarreo = 0;//variable de control.
8         int[] c = new int[a.length];//Arreglo a retornar.
9         //Se recorre el arreglo a de manera inversa.
10        for (int i = a.length - 1; i > -1; i--) {
11            int resta = a[i] - b[i]; //Se realiza la suma una vez por ciclo
12            if (resta == -1) {//Si la suma da 2 los elementos en el arreglo son 1 y 1.
13                c[i] = 1;//Por lo tanto debe ir como resultado un 0
14                acarreo = 1;//Y se lleva un acarreo
15            } //Si por el contrario la suma es 0 quiere decir que los elementos
16            //eran 0 y 0.
17            else if (resta == 0) {
18                c[i] = 0;//Entonces debe ir 0 como resultado
19            } //Si por el contrario la suma es 1, entonces los elementos
20            //eran 0 y 1 o viceversa.
21            else if (resta == 1) {
22                c[i] = 1;//Y debe ir como resultado un 1.
23            }
24            if (acarreo == 1) {//Si existe un acarreo en este ciclo
25                if (i == 0) { // Excedio el intervalo y geenro un numero Aleatorio
26                    c = Binary.Random_Array_Bin(a.length);
27                    break;
28                } else {
29                    //Se comprueba si la suma del elemento siguiente del
30                    //arreglo "a" mas el acarreo es igual a 2.
31                    if (b[i - 1] + 1 == 2) {
32                        b[i - 1] = 0;//Si lo es, se asigna un 0 a "a".
33                        acarreo = 1;//Y se sigue llevando un acarreo.
34                    } else {//Si la suma no da 2, solo puede dar 1.
35                        b[i - 1] = 1;//Y entonces se asigna 1 a "a".
36                        acarreo = 0;//Y el acarreo desaparece.
37                    }
```

```
38                  }
39              }
40
41          }
42          return c;//Se regresa el arreglo.
43      }
```

Apéndice D

Clase Cromosoma

```
1  public class Cromosoma implements Comparable<Cromosoma> {
2
3      private int[] chain_bits;
4      private ArrayList<Gen> chain_genes;
5      private double[] Coeficientes;
6      private double[] delta_Coef; // arreglo que contiene los limites de cada Coeficiente
7      private double fittness;
8      private int orden_polinomio;
9      private int long_bits;
10
11     /**
12      * COMAPRA LOS FITTNESS
13      */
14     public int compareTo(Cromosoma crom) {
15         if (this.fittness == crom.getFittness()) {
16             return 0;
17         } else if (this.fittness > crom.getFittness()) {
18             return 1;
19         } else {
20             return -1;
21         }
22     }
23
24     /**
25      * CONSTRUCTOR PARA OBTENER UN CROMOSOMA ALEATORIO
26      */
27     public Cromosoma(int orden_polinomio, int long_bits, double[] delta) {
28         int num_Coef = (orden_polinomio + 1) * (orden_polinomio + 2) / 2;
29         int tam_chain = long_bits * num_Coef;
30
31         chain_bits = new int[tam_chain];
32         chain_genes = new ArrayList<>();
33         Coeficientes = new double[num_Coef];
34         delta_Coef = delta;
35         this.orden_polinomio = orden_polinomio;
36         this.long_bits = long_bits;
37
38         // Genero los genes aleatoriamente
39         for (int i = 0; i < num_Coef; i++) {
```

```
40              chain_genes.add(new Gen(long_bits));
41          }
42
43          // Convierto los Genes en una sola cadena Cromosomica (Cadena de Bits)
44          int idx = 0; // indice del arreglo que almacenara la cadena cromosomica
45          for (Gen gen : chain_genes) {
46              for (int i = 0; i < long_bits; i++) {
47                  chain_bits[idx++] = gen.getArray_bits()[i];
48              }
49          }
50  //        chain_bits = Auxiliar_Binaria_AG.Binario_Gray(chain_bits);
51          // Convierto los coeficientes a numeros Reales
52          idx = 0;
53          for (Gen gen : chain_genes) {
54              Coeficientes[idx] = Binaria_AG.Cod_Bin_Decimal(-delta_Coef[idx],
                    delta_Coef[idx], gen.getArray_bits());
55              idx++;
56          }
57          this.fittness = Function_AG.Goal_Funtion_Value_D2(this);
58      }
59
60      /**
61       * CONSTRUCTOR PARA CREAR UN CROMOSOMA A PARTIR DE UN ARRAY DE BITS
62       */
63      public Cromosoma(int orden_polinomio, int long_bits, int[] chain_bits, double[]
            delta) {
64          int num_Coef = (orden_polinomio + 1) * (orden_polinomio + 2) / 2;
65
66          this.chain_bits = chain_bits;
67          chain_genes = new ArrayList<>();
68          Coeficientes = new double[num_Coef];
69          delta_Coef = delta;
70          this.orden_polinomio = orden_polinomio;
71          this.long_bits = long_bits;
72
73          // Genero los genes a partir de la cadena cromosomica
74          int[] array_temp;
75          int idx = 0;
76          for (int i = 0; i < num_Coef; i++) {
77              array_temp = new int[long_bits];
78              for (int j = 0; j < long_bits; j++) {
79                  array_temp[j] = chain_bits[idx++];
80              }
81              chain_genes.add(new Gen(array_temp));
82          }
83
84          chain_bits = Binaria_AG.Binario_Gray(chain_bits);
85          // Convierto los coeficientes a numeros Reales
86          idx = 0;
87          for (Gen gen : chain_genes) {
88              int[] bit_Coef = Binaria_AG.Gray_Binario(gen.getArray_bits());
89  //            int[] bit_Coef = gen.getArray_bits();
90              Coeficientes[idx] = Binaria_AG.Cod_Bin_Decimal(-delta_Coef[idx],
                    delta_Coef[idx], bit_Coef);
91              idx++;
```

```
92          }
93          this.fittness = Function_AG.Goal_Funtion_Value_D2(this);
94      }
95
96      /**
97       * CONSTRUCTOR PARA CREAR UN CROMOSOMA A PARTIR DE UN ARRAY DE BITS
98       */
99      public Cromosoma(int orden_polinomio, int long_bits, double[] coeficientes, double[]
            delta) {
100         int num_Coef = (orden_polinomio + 1) * (orden_polinomio + 2) / 2;
101
102         this.chain_bits = new int[num_Coef * long_bits];
103         chain_genes = new ArrayList<>();
104         Coeficientes = coeficientes;
105         delta_Coef = delta;
106         this.orden_polinomio = orden_polinomio;
107         this.long_bits = long_bits;
108
109         // Genero los genes a partir de la cadena cromosomica
110         int[] array_temp;
111         int idx = 0;
112         for (int i = 0; i < num_Coef; i++) {
113             int coef_tmp = (int) ((Math.pow(2, long_bits) - 1)*(this.Coeficientes[i] -
                    (-this.delta_Coef[i]))/(this.delta_Coef[i] - (-this.delta_Coef[i])));
114             array_temp = Binaria_AG.Decimal_Binario(coef_tmp, long_bits);
115             for (int j = 0; j < this.long_bits; j++) {
116                 this.chain_bits[idx++] = array_temp[j];
117             }
118             this.chain_genes.add(new Gen(array_temp));
119         }
120         this.fittness = Function_AG.Goal_Funtion_Value_D2(this);
121     }
```

Apéndice E

Clase Gen

```
1  public class Gen {
2
3      private int[] array_bits;
4      public Gen(int long_bits) {
5          array_bits = new int[long_bits];
6          for (int i = 0; i < long_bits; i++) {
7              if (Math.random() < .5) {
8                  array_bits[i] = 1;
9              } else {
10                 array_bits[i] = 0;
11             }
12         }
13     }
14     public Gen(int[] array_bits) {
15         this.array_bits = array_bits;
16     }
17     public int[] getArray_bits() {
18         return array_bits;
19     }
20
21 }
```

Apéndice F

Método de selección a partir de probabilidad acumulativa

```
1
2       /**
3        * SELECCION POR RULETA RANKEO DEVUELVE CUAL FUE EL SELECIONADO
4        */
5       private int Ruleta_Valor_Esperado_Pos(double[] prob_Ac, int N) {
6           int pos = 0;
7           double random = this.rnd.nextDouble() * N;
8           while (prob_Ac[pos] < random) {
9               pos++;
10          }
11          return pos;
12      }
```

Apéndice G

Métodos suplementarios utilizados en la selección por Escalamiento Sigma

```
1
2       /**
3        * CALCULA LA MEDIA DE LA FUNCION OBJETIVO DE LA POBLACION
4        */
5       private double Media(ArrayList<Cromosoma> pob) {
6           double media = 0;
7           for (Cromosoma elemt : pob) {
8               media += elemt.getFittness();
9           }
10          return media / pob.size();
11      }
12
13      /**
14       * CALCULA LA DESVIACION STANDAR DE LA FUNCION OBJETIVO DE LA POBLACION
15       */
16      private double Desviacion_Standar(ArrayList<Cromosoma> pob, double media) {
17          double desv = 0;
18          for (Cromosoma elemt : pob) {
19              desv += Math.pow(elemt.getFittness() - media, 2);
20          }
21          return Math.sqrt(desv / (pob.size() - 1));
22      }
```

Apéndice H

Método que genera un nodo a partir de su extremo local

```
1
2     public static Node vmo_F_Funtion(Node current, Node best_local, double[]
          distanceShare, double Pr) {
3
4         double[] phase = new double[current.getPhase_coef().length];
5         double halfValue;
6
7         for (int i = 0; i < current.getPhase_coef().length; i++) {
8             halfValue = Auxiliar.halfPoint_Calculate(current.getPhase_coef()[i],
                  best_local.getPhase_coef()[i]);
9
10            if ((Math.random() <= Pr) && (Math.abs(halfValue -
                  best_local.getPhase_coef()[i]) > distanceShare[i])) {
11                phase[i] = halfValue;
12            } else if (Math.abs(halfValue - best_local.getPhase_coef()[i]) <=
                  distanceShare[i]) {
13                phase[i] = best_local.getPhase_coef()[i] +
                      Auxiliar.randomValue_Generate(-distanceShare[i], distanceShare[i]);
14            } else {
15                phase[i] = Auxiliar.randomValue_Generate(halfValue,
                      best_local.getPhase_coef()[i]);
16            }
17        }
18        return new Node(phase);
19    }
```

vmoglobal

Apéndice I

Método que genera un nodo a partir de su extremo globall

```
1
2     public static Node vmo_G_Funtion(Node current, Node global_n, double Pr) {
3
4         double[] phase = new double[current.getPhase_coef().length];
5         double halfValue;
6
7         for (int i = 0; i < current.getPhase_coef().length; i++) {
8
9             halfValue = Auxiliar.halfPoint_Calculate(current.getPhase_coef()[i],
                  global_n.getPhase_coef()[i]);
10
11            if (Math.random() < Pr) {
12                phase[i] = halfValue;
13            } else {
14                phase[i] = Auxiliar.randomValue_Generate(halfValue,
                      global_n.getPhase_coef()[i]);
15            }
16
17        }
18
19        return new Node(phase);
20    }
```

Apéndice J

Método para calcular el intervalo de búsqueda de cada coeficiente

```
1     /* Metodo que calucula los intervalos para cada coeficientes **/
2     public static double[] Phase_Delta_COeficientes(double numFranjas, double orden,
          double xMax, double yMax) {
3         int num_coef = (int) ((orden + 1) * (orden + 2) / 2);
4         double delta[] = new double[num_coef];
5         delta[0] = 2* Math.PI*numFranjas;
6         int k, exp, c = 0;
7         int grado = (int) (-3 + Math.sqrt(9 + 8 * (num_coef - 1))) / 2;
8         for (int i = 1; i <= grado; i++) {
9             k = 0;
10            for (int j = i + c; j <= 2 * i + c; j++) {
11                exp = (i + k + 2) % (i + 1);
12                delta[j] = 2* Math.PI*numFranjas/ (Math.pow(xMax, exp) * Math.pow(yMax, i
                      - exp));
13                k++;
14            }
15            c += i;
16        }
17        return delta;
18    }
```

Apéndice K

Tablas de Resultados para la función 1

Corridas	Fitness	Error	Iter Óptimo	Tiempo Promedio(s)
1	1692.549480	0.162320	488	34.9
2	10574.19537	30.43900	447	34.9
3	1700.388634	0.245960	476	34.9
4	1671.022074	0.051853	457	34.9
5	12000.25569	7.399600	193	34.9
6	1672.388887	0.107950	453	34.9
7	1699.245147	0.212600	413	34.9
8	2092.203817	0.699730	434	34.9
9	1669.606021	0.187020	466	34.9
10	1679.166929	0.085902	497	34.9
Promedio	3.9591935	0.612609		

Tabla K.1: Resultados de 10 corridas para la mejor configuración con $N = 500$, $T = 9\ 000$, $ITemp = 150$, $m = 10$ y el factor de vecindad iguala 0.1, para el interferograma de la Figura 5.2.

Corridas	Fitness	Error	Iter Óptimo	Tiempo Promedio(s)
1	11163.32575	32.23800	490	35
2	11110.09645	30.92500	377	35
3	11211.62301	34.15100	485	35
4	11893.64686	13.95300	436	35
5	1671.377048	0.052556	398	35
6	1693.773297	0.108690	461	35
7	2107.459821	0.731800	469	35
8	8144.607330	1.659500	119	35
9	1694.008046	0.137470	471	35
10	1737.064115	0.407470	444	35
Promedio	6242.698173	11.4364486		

Tabla K.2: Resultados de 10 corridas para la mejor configuración con $N = 500$, $T = 90\ 000$, $ITemp = 150$, $m = 10$ y el factor de vecindad iguala 0.1, para el interferograma de la Figura 5.2.

Corridas	Fitness	Error	Iter Óptimo	Tiempo Promedio(s)
1	10969.39093	33.73400	491	34.9
2	8571.076250	2.436100	250	34.9
3	1685.927247	0.117310	467	34.9
4	10477.70358	5.574800	293	34.9
5	1683.415593	0.051683	476	34.9
6	1705.468101	0.177980	483	34.9
7	10382.09987	6.151300	135	34.9
8	1690.851350	0.121930	465	34.9
9	11446.99137	18.67100	477	34.9
10	5015.198038	1.538000	369	34.9
Promedio	6362.812233	6.8574103		

Tabla K.3: Resultados de 10 corridas para la mejor configuración con $N = 500$, $T = 900\ 000$, $ITemp = 150$, $m = 10$ y el factor de vecindad iguala 0.1, para el interferograma de la Figura 5.2.

Corridas	Fitness	Error	Iter Óptimo	Tiempo Promedio(s)
1	10969.39093	33.73400	491	34.9
2	8571.076250	2.436100	250	34.9
3	1685.927247	0.117310	467	34.9
4	10477.70358	5.574800	293	34.9
5	1683.415593	0.051683	476	34.9
6	1705.468101	0.177980	483	34.9
7	10382.09987	6.151300	135	34.9
8	1690.851350	0.121930	465	34.9
9	11446.99137	18.67100	477	34.9
10	5015.198038	1.538000	369	34.9
Promedio	6362.812233	6.8574103		

Tabla K.4: Resultados de 10 corridas para la mejor configuración con $N = 500$, $T = 9\ 000\ 000$, $ITemp = 150$, $m = 10$ y el factor de vecindad iguala 0.1, para el interferograma de la Figura 5.2.

Corridas	Fitness	Error	Iter Óptimo	Tiempo Promedio(s)
1	1674.447583	0.08977	441	55.4
2	1693.902291	0.28410	416	55.4
3	9748.638482	5.98450	319	55.4
4	1684.214344	0.10028	432	55.4
5	1667.380724	0.12442	494	55.4
6	1688.035491	0.25619	467	55.4
7	2099.430202	0.74343	478	55.4
8	1674.148470	0.20118	466	55.4
9	1672.765528	0.18289	490	55.4
10	1689.581487	0.10061	452	55.4
Promedio	2529.25446	0.806737		

Tabla K.5: Resultados de 10 corridas para la mejor configuración con $N = 500$, $T = 9\ 000$, $ITemp = 200$, $m = 10$ y el factor de vecindad iguala 0.1, para el interferograma de la Figura 5.2.

Corridas	Fitness	Error	Iter Óptimo	Tiempo Promedio(s)
1	1671.513021	0.082274	495	55.4
2	1672.963872	0.129900	497	55.4
3	10076.06518	8.055400	275	55.4
4	1681.225630	0.118760	442	55.4
5	1667.380724	0.146830	452	55.4
6	10856.88894	30.10100	452	55.4
7	1674.734672	0.087759	486	55.4
8	2100.214925	0.760310	437	55.4
9	11554.36875	3.750100	140	55.4
10	10798.13580	40.33500	466	55.4
Promedio	5376.976882	8.3567333		

Tabla K.6: Resultados de 10 corridas para la mejor configuración con $N = 500$, $T = 90\ 000$, $ITemp = 200$, $m = 10$ y el factor de vecindad iguala 0.1, para el interferograma de la Figura 5.2.

Corridas	Fitness	Error	Iter Óptimo	Tiempo Promedio(s)
1	10903.12798	33.45300	490	55.3
2	8099.215285	7396.100	489	55.3
3	1672.572364	0.052909	484	55.3
4	1674.319907	0.171750	461	55.3
5	1681.855450	0.210570	499	55.3
6	1688.625960	0.233400	483	55.3
7	1690.848705	0.127730	448	55.3
8	2095.567300	0.709900	477	55.3
9	11727.35545	31.78600	373	55.3
10	1673.191064	0.123720	476	55.3
Promedio	4290.667947	746.2968979		

Tabla K.7: Resultados de 10 corridas para la mejor configuración con $N = 500$, $T = 900\ 000$, $ITemp = 200$, $m = 10$ y el factor de vecindad iguala 0.1, para el interferograma de la Figura 5.2.

Corridas	Fitness	Error	Iter Óptimo	Tiempo Promedio(s)
1	1678.863643	0.13334	445	55.4
2	10523.73362	7.50120	421	55.4
3	1700.891835	0.16014	487	55.4
4	1688.384539	0.24171	490	55.4
5	10393.58908	5.06780	245	55.4
6	11986.74673	9.17180	275	55.4
7	11016.54166	8.47090	118	55.4
8	11583.77414	8.20810	231	55.4
9	9343.250084	3.29820	493	55.4
10	1675.563981	0.16235	459	55.4
Promedio	7159.133931	4.241554		

Tabla K.8: Resultados de 10 corridas para la mejor configuración con $N = 500$, $T = 9\ 000\ 000$, $ITemp = 200$, $m = 10$ y el factor de vecindad iguala 0.1, para el interferograma de la Figura 5.2.

Corridas	Fitness	Error	Iter Óptimo	Tiempo Promedio(s)
1	1666.419462	0.111530	482	83
2	1664.204022	0.172230	455	83
3	1679.718366	0.282720	367	83
4	1688.384539	0.120530	446	83
5	10393.58908	0.256970	491	83
6	1676.487155	0.081482	461	83
7	1658.638315	0.099338	465	83
8	1677.397602	0.087686	450	83
9	1695.822797	0.290930	459	83
10	2093.448667	0.745890	445	83
Promedio	1716.213726	0.2249306		

Tabla K.9: Resultados de 10 corridas para la mejor configuración con $N = 500$, $T = 9\ 000$, $ITemp = 300$, $m = 10$ y el factor de vecindad iguala 0.1, para el interferograma de la Figura 5.2.

Corridas	Fitness	Error	Iter Óptimo	Tiempo Promedio(s)
1	1694.264651	0.113890	499	83
2	1678.982473	0.176770	462	83
3	1700.139171	0.248890	464	83
4	1675.308706	0.091805	478	83
5	1674.271254	0.168290	497	83
6	1664.762538	0.139340	485	83
7	1680.293576	0.099852	462	83
8	1663.396581	0.104230	471	83
9	1669.051390	0.155640	487	83
10	1696.347921	0.131500	493	83
Promedio	1679.681826	0.1430207		

Tabla K.10: Resultados de 10 corridas para la mejor configuración con $N = 500$, $T = 90\,000$, $ITemp = 300$, $m = 10$ y el factor de vecindad iguala 0.1, para el interferograma de la Figura 5.2.

Corridas	Fitness	Error	Iter Óptimo	Tiempo Promedio(s)
1	10901.67532	29.36400	495	83
2	1678.343908	0.090635	486	83
3	1684.315490	0.101790	498	83
4	1663.163165	0.190130	447	83
5	1695.504709	0.141780	478	83
6	1672.373669	0.235180	435	83
7	1684.641444	0.086922	489	83
8	10378.52282	2.403100	347	83
9	1685.216906	0.201130	444	83
10	1663.926369	0.165220	494	83
Promedio	3470.76838	3.2979887		

Tabla K.11: Resultados de 10 corridas para la mejor configuración con $N = 500$, $T = 900\,000$, $ITemp = 300$, $m = 10$ y el factor de vecindad iguala 0.1, para el interferograma de la Figura 5.2.

Corridas	Fitness	Error	Iter Óptimo	Tiempo Promedio(s)
1	1667.975874	0.059016	486	83
2	10419.16020	10.61000	472	83
3	1711.835969	0.264950	486	83
4	1678.175762	0.079262	481	83
5	1677.791457	0.145480	470	83
6	10919.93156	30.15000	496	83
7	1681.430736	0.285260	489	83
8	1715.436061	0.243180	491	83
9	1680.522923	0.118900	493	83
10	1673.332736	0.183760	490	83
Promedio	3482.559327	4.2139808		

Tabla K.12: Resultados de 10 corridas para la mejor configuración con $N = 500$, $T = 9\ 000\ 000$, $ITemp = 300$, $m = 10$ y el factor de vecindad iguala 0.1, para el interferograma de la Figura 5.2.

Corridas	Fitness	Error	Iter Óptimo	Tiempo Promedio(s)
1	1671.827725	0.119650	465	111
2	1669.579551	0.173350	450	111
3	1675.439227	0.210520	485	111
4	1681.636653	0.152330	485	111
5	1675.691789	0.283550	427	111
6	1687.242819	0.116950	452	111
7	10383.52462	5.601900	378	111
8	1673.531811	0.092510	496	111
9	1663.447419	0.069548	468	111
10	1674.522982	0.070481	466	111
Promedio	2545.644459	0.6890789		

Tabla K.13: Resultados de 10 corridas para la mejor configuración con $N = 500$, $T = 90\ 000$, $ITemp = 300$, $m = 10$ y el factor de vecindad iguala 0.1, para el interferograma de la Figura 5.2.

Corridas	Fitness	Error	Iter Óptimo	Tiempo Promedio(s)
1	1674.073294	0.103620	467	111
2	1676.108563	0.264840	490	111
3	1688.307242	0.102480	497	111
4	1691.381780	0.120820	498	111
5	1693.406789	0.169790	499	111
6	1672.148441	0.097538	493	111
7	1687.161033	0.070433	443	111
8	10667.73644	31.02000	460	111
9	1680.205937	0.156110	480	111
10	8099.101771	6261.200	497	111
Promedio	3222.963129	629.3305631		

Tabla K.14: Resultados de 10 corridas para la mejor configuración con $N = 500$, $T = 900\,000$, $ITemp = 300$, $m = 10$ y el factor de vecindad iguala 0.1, para el interferograma de la Figura 5.2.

Corridas	Fitness	Error	Iter Óptimo	Tiempo Promedio(s)
1	1672.653524	0.187010	474	111
2	1675.377982	0.276770	485	111
3	2097.940330	0.703140	494	111
4	1678.733713	0.089744	437	111
5	1689.736386	0.126840	496	111
6	1708.766297	0.276930	476	111
7	1692.156850	0.125800	453	111
8	1670.288288	0.152050	495	111
9	1697.511719	0.167050	469	111
10	1669.117566	0.069688	479	111
Promedio	1725.228266	0.2175022		

Tabla K.15: Resultados de 10 corridas para la mejor configuración con $N = 500$, $T = 9\,000\,000$, $ITemp = 300$, $m = 10$ y el factor de vecindad iguala 0.1, para el interferograma de la Figura 5.2.

Apéndice L

Algoritmo de empalme de cuadrantes

Este método fue tomado de [66] e implementado en Matlab con el objetivo de unificar las fases de cada cuadrante recuperado por las metaheurística.

```
function fase = Join_Phase(row, col, phase)
for i = 1:row
    for j = 1:col-1
        current = squeeze(phase(i,j,:,:));
        next = squeeze(phase(i,j+1,:,:));
        nextInv = -1*next;
        ColOverlap = size(current,2);
        Area = size(current,1);
        DC1 = sum( ((current(:,ColOverlap) - next(:,1) )/Area )');
        DC2 = sum( ((current(:,ColOverlap) - nextInv(:,1) )/Area)');
        RMS1 = sum( ((current(:,ColOverlap) - next(:,1) - DC1).^2/Area )');
        RMS2 = sum( ((current(:,ColOverlap) - nextInv(:,1) - DC2).^2/Area)');

        if RMS1 < RMS2
            phase(i,j+1,:,:) = next + DC1;
        else
            phase(i,j+1,:,:) = nextInv + DC2;
        end
    end
    if i > 1
        current = squeeze(phase(i-1,1,:,:));
        next = squeeze(phase(i,1,:,:));
        nextInv = -1*next;
        RowOverlap = size(current,1);
        Area = size(current,2);
        DC1 = sum( ((current(RowOverlap,:) - next(1,:) )/Area )');
        DC2 = sum( ((current(RowOverlap,:) - nextInv(1,:) )/Area)');

        RMS1 = sum( ((current(RowOverlap,:) - next(1,:) - DC1).^2/Area )');
        RMS2 = sum( ((current(RowOverlap,:) - nextInv(1,:) - DC2).^2/Area)');

        if RMS1 < RMS2
            for k = 1: col
                next = squeeze(phase(i,k,:,:));
                phase(i,k,:,:) = next + DC1;
            end
        else
```

```
            for k = 1: col
                next = squeeze(phase(i,k,:,:));
                nextInv = -1*next;
                phase(i,k,:,:) = nextInv + DC1;
            end
        end
    end
    end

for i = 1:row
    fila = squeeze(phase(i,1,:,:));
    for j = 2:col
        next = squeeze(phase(i,j,:,:));
        fila = horzcat(fila,next);
    end
    if i > 1
        fase = vertcat(fase,fila);
    else
        fase = fila;
    end
    clear fila;
end
```

Apéndice M

Interferogramas Simulados

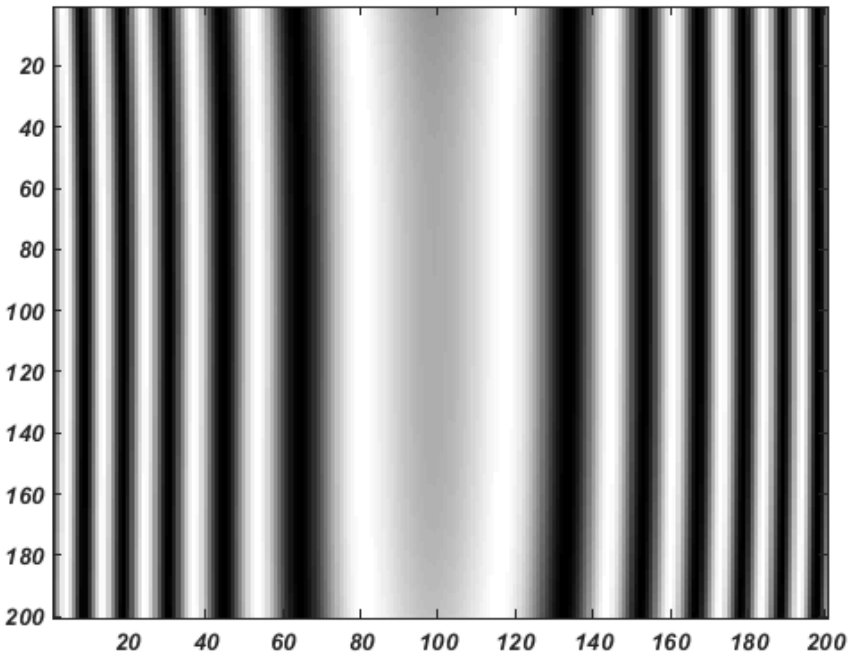


Figura M.1: Interferograma de la función de prueba 4 con resolución de 200 x 200 pixeles.

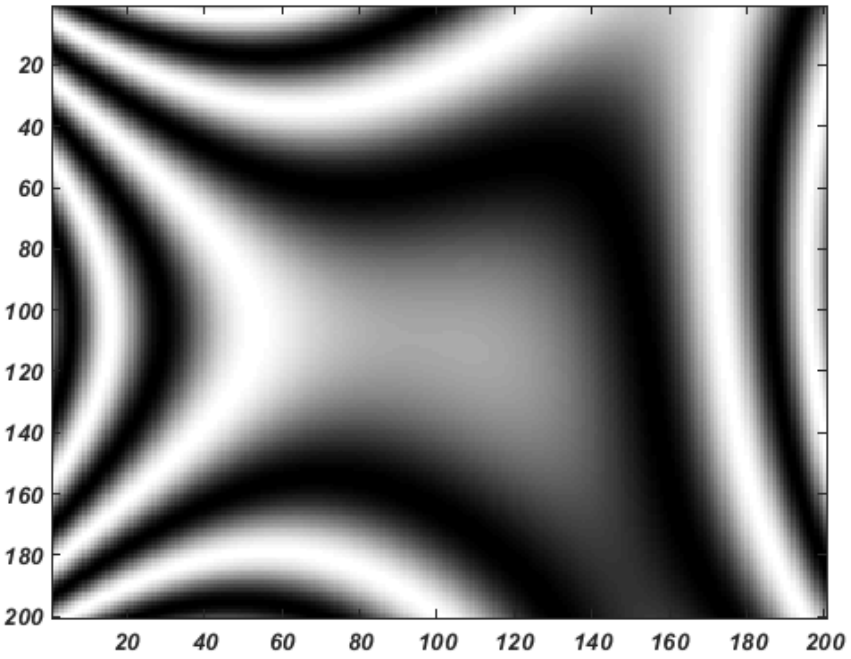


Figura M.2: Interferograma de la función de prueba 5 con resolución de 200 x 200 pixeles.

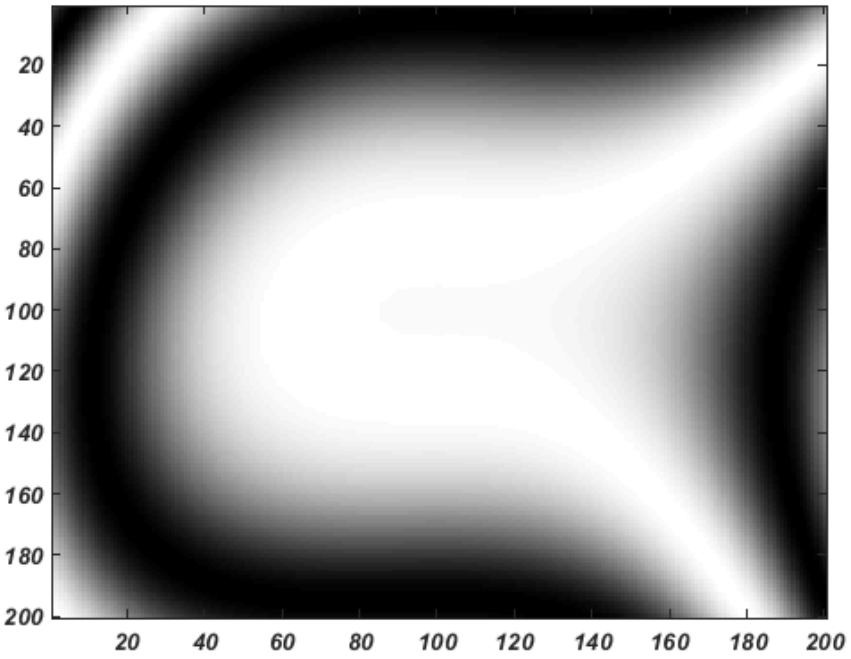


Figura M.3: Interferograma de la función de prueba 6 con resolución de 200 x 200 pixeles.

Bibliografía

[1] F. Cuevas, O. Gonzalez, Y. Susuki, D. Hernandez, M. Rocha, and N. Alcala, "Genetic algorithms applied to optics and engineering - art. no. 60461K," *Fifth Symposium Optics in Industry* **6046** (2006) 60461K.

[2] T. W. Liao, P. J. Egbelu, and P. C. Chang, "Two hybrid differential evolution algorithms for optimal inbound and outbound truck sequencing in cross docking operations," *Applied Soft Computing Journal* **12** no.˜11, (2012) 3683–3697. http://dx.doi.org/10.1016/j.asoc.2012.05.023.

[3] J. E. and others Eiben, Agoston E and Smith, *Introduction to Evolutionary Computing*. Springer, 2003.

[4] D. H. Wolpert and W. G. Macready, "No Free Lunch Theorems for Optimization,".

[5] E. A. U. Hecht, *Optics Hetch 4th ed.pdf*. Addisson Wesley Iberoamericana, Madrid, 2000.

[6] R. C. Gonzalez and R. E. Woods, *Digital Image Processing (3rd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2006.

[7] R. A. Ryerson, F. M. Henderson, A. J. Lewis, A. S. for Photogrammetry, and R. Sensing, *Manual of Remote Sensing, Principles and Applications of Imaging Radar*. Manual of Remote Sensing - Third Edition. Wiley, 1998. https://books.google.com.mx/books?id=E9AsAQAAMAAJ.

[8] M. Servín, J. A. Quiroga, and M. Padilla, *Fringe pattern analysis for optical metrology: theory, algorithms, and applications*. John Wiley & Sons, 2014.

[9] J. Morlet, *Sampling Theory and Wave Propagation*, pp.˜233–261. Springer Berlin Heidelberg, Berlin, Heidelberg, 1983. http://dx.doi.org/10.1007/978-3-642-82002-1{_}12.

[10] M. Takeda, H. Ina, and S. Kobayashi, "Fourier-transform method of fringe-pattern analysis for computer-based topography and interferometry," *Journal of the Optical Society of America* **72** no. 1, (1982) 156–160.

[11] D. Malacara, "Phase shifting interferometry *," *Revista Mexicana de Física* **36** no. 1, (1990) 6–22.

[12] D. W. Robinson and G. T. Reid, *Interferogram Analysis, Digital Fringe Pattern Measurement Techniques*. Taylor & Francis, 1993. https://books.google.com.mx/books?id=KrbvAAAAMAAJ.

[13] D. Malacara, *Optical shop testing*, vol. 59. John Wiley & Sons, 2007.

[14] M. W. U. o. T. Kujawinska, J. I. o. D. o. P. Wójciak, Optical, and Instruments), "Spatial-carrier phase shifting technique of fringe pattern analysis," *IndustrialApplications of Holographic and Speckle Measuring Techniques* **1508** (1991) 61–67.

[15] M. Takeda and K. Mutoh, "Fourier transform profilometry for the automatic measurement of 3-D object shapes," *APPLIED OPTICS* **22** no. August, (1983) 1–5.

[16] R. Juarez-Salazar, C. Robledo-Sanchez, and F. Guerrero-Sanchez, "Phase-unwrapping algorithm by a rounding-least-squares approach," *Optical Engineering* **53** no.˜2, (2014) 24102. http://dx.doi.org/10.1117/1.OE.53.2.024102.

[17] J. Marroquin, S. Mitter, and T. Poggio, "Probabilistic Solution of Ill-Posed Problems in Computational Vision," *Journal of the American Statistical Association* **82** no.˜397, (1987) 76–89. http://www.tandfonline.com/doi/abs/10.1080/01621459.1987.10478393.

[18] A. N. Tikhonov and V. I. A. Arsenin, *Solutions of ill-posed problems / Andrey N. Tikhonov and Vasiliy Y. Arsenin ; translation editor, Fritz John.* Winston ; distributed solely by Halsted Press Washington : New York, 1977.

[19] L. Araujo and C. Cervigón, *Algoritmos Evolutivos: un enfoque práctico.* Alfaomega, primera ed ed., 2009.

[20] A. Sánchez, *Técnicas Avanzadas de demodulación de patrones de franjas.* PhD thesis, Centro de Investigaciones en Ópticas, 2007.

[21] M. Servin, J. L. Marroquin, and F. J. Cuevas, "Demodulation of a single interferogram by use of a two-dimensional regularized phase-tracking technique," *Appl. Opt.* **36** no.˜19, (1997) 4540–4548. http://ao.osa.org/abstract.cfm?URI=ao-36-19-4540.

[22] M. Servin, J. L. Marroquin, D. Malacara, and F. J. Cuevas, "Phase unwrapping with a regularized phase-tracking system," *Applied Optics* **37** no. 10, (1998) 1917–1923.

[23] M. Servin, F. J. Cuevas, D. Malacara, J. L. Marroquin, and R. Rodriguez-Vera, "Phase unwrapping through demodulation by use of the regularized phase-tracking technique," *Applied Optics* **38** no. 10, (1999) 1934–1941.

[24] R. Rodriguez-Vera and M. Servin, "Phase locked loop profilometry," *Optics & Laser Technology* **26** no. 6, (1994) 393–398.

[25] F. J. Cuevas, F. Mendoza, M. Servin, and J. H. Sossa-Azuela, "Window fringe pattern demodulation by multi-functional fitting using a genetic algorithm," *Optics Communications* **261** no.˜2, (2006) 231–239.

[26] L. E. Espinosa, J. M. Valadez, and F. J. Cuevas, "Characterize the parameters of genetic algorithms based on zernike polynomials for recovery of the phase of interferograms of closed fringes using hybrid technique," *Advances in Soft Computing* **41** (2007) 277–289.

[27] L. Espinosa and J. Valadez, "Demodulation of Interferograms of Closed Fringes by Zernike Polynomials using a technique of Soft Computing," *Engineering Letters* no. August, (2007) . `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.149.2946{&}rep=rep1{&}type=pdf`.

[28] J. F. Jimenez, F. J. Cuevas, J. H. Sossa, and L. E. Gomez, "Fringe-Pattern Demodulation Using a Parametric,".

[29] F. J. Cuevas, J. H. Sossa-Azuela, and M. Servin, "A parametric method applied to phase recovery from a fringe pattern based on a genetic algorithm," *Optics Communications* **203** no.˜3-6, (2002) 213–223.

[30] F. Cuevas and M. Servin, "Closed fringe demodulation by multi-polynomial fitting," *Proceedings of SPIE - The International Society for Optical Engineering* **5531** (2004) 405–412.

[31] J. Jiménez, H. Sossa, F. Cuevas, and L. Gómez, "Demodulation of Interferograms based on Particle Swarm Optimization,".

[32] F. J. Cuevas, J. H. S. Azuela, G. Garnica, J. Rayas, M. Carpio, and O. Gonzalez, "Fringe normalization by using of an interpolation algorithm," *Eighth International Symposium on Laser Metrology: MACRO-, MICRO-, AND NANO-TECHNOLOGIES APPLIED IN SCIENCE, ENGINEERING, AND INDUSTRY* **5776** (2005) 684–691.

[33] T. Back, U. Hammel, and H. P. Schwefel, "Evolutionary Computation: Comments on the History and Current State," *Trans. Evol. Comp* **1** no.˜1, (1997) 3–17. `http://dx.doi.org/10.1109/4235.585888`.

[34] C. Darwin, *The Origin of Species*. P. F. Collier & Son, 1909. `https://books.google.ca/books?id=YY4EAAAAYAAJ`.

[35] J. H. Holland, *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, 1975.

[36] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st ed., 1989.

[37] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.

[38] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by Simulated Annealing," *American Association for the Advancement of Science* **220** no. 4598, (1983) 671–680.

[39] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller, "Equation of State Calculations by Fast Computing Machines," *\jcp* **21** (1953) 1087–1092.

[40] E.-G. Talbi, *Metaheuristics: From Design to Implementation*. Wiley Publishing, 2009.

[41] P. Moscato, "An Introduction to Population Approaches for Optimization and Hierarchical Objective Functions: A Discussion on the Role of Tabu Search," *Ann. Oper. Res.* **41** no. 1-4, (May, 1993) 85–121. `http://dl.acm.org/citation.cfm?id=160231.160242`.

[42] F. Romeo and A. Sangiovanni-Vincentelli, "A theoretical framework for simulated annealing," *Algorithmica* **6** no.˜1, (1991) 302. `http://dx.doi.org/10.1007/BF01759049`.

[43] M. F. Cardoso, R. L. Salcedo, S. F. De Azevedo, and Others, "Nonequilibrium simulated annealing: a faster approach to combinatorial minimization," *Industrial and Engineering Chemistry Research* **33** (1994) 1908.

[44] Y. Nourani and B. Andresen, "A comparison of simulated annealing cooling strategies," *Journal of Physics A: Mathematical and General* **31** no. 41, (1998) 8373.

[45] D. Henderson, S. H. Jacobson, and A. W. Johnson, "The theory and practice of simulated annealing," in *Handbook of metaheuristics*, pp. 287–319. Springer, 2003.

[46] M. A. S. Elmohamed, P. Coddington, and G. Fox, *A comparison of annealing techniques for academic course scheduling*, pp.˜92–112. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998. `http://dx.doi.org/10.1007/BFb0055883`.

[47] A. Albrecht and C.-K. Wong, *On Logarithmic Simulated Annealing*, pp.˜301–314. Springer Berlin Heidelberg, Berlin, Heidelberg, 2000. `http://dx.doi.org/10.1007/3-540-44929-9{_}23`.

[48] E. Rodriguez-Tello, J.-K. Hao, and J. Torres-Jimenez, "An effective two-stage simulated annealing algorithm for the minimum linear arrangement problem," *Computers & Operations Research* **35** no. 10, (2008) 3331–3346.

[49] R. Otten and L. Van Ginneken, "Stop criteria in simulated annealing," in *Computer Design: VLSI in Computers and Processors, 1988. ICCD'88., Proceedings of the 1988 IEEE International Conference on*, pp. 549–552, IEEE. 1988.

[50] L. Davis, "Handbook of genetic algorithms,".

[51] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs (3rd Ed.)*. Springer-Verlag, London, UK, UK, 1996.

[52] K. A. De Jong, *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*. PhD thesis, Ann Arbor, MI, USA, 1975.

[53] J. E. Baker, "Reducing Bias and Inefficiency in the Selection Algorithm," in *Proceedings of the Second International Conference on Genetic Algorithms on Genetic Algorithms and Their Application*, pp. 14–21. L. Erlbaum Associates Inc., Hillsdale, NJ, USA, 1987. `http://dl.acm.org/citation.cfm?id=42512.42515`.

[54] D. E. Goldberg and K. Deb, "A comparative analysis of selection schemes used in genetic algorithms," *Foundations of genetic algorithms* **1** (1991) 69–93.

[55] M. Mitchell, *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, MA, USA, 1996.

[56] D. E. Goldberg and Others, "A note on Boltzmann tournament selection for genetic algorithms and population-oriented simulated annealing," *Complex Systems* **4** no. 4, (1990) 445–460.

[57] M. Gestal, D. Rivero, J. R. Rabuñal, J. Dorado, and A. Pazos, "Introducción a los Algoritmos genéticos y la Programación genética," *Universidade da Coruña Servizo de Publicacións* (2010) .

[58] A. Y. Puris, *Desarrollo de meta-heurísticas poblacionales para la solución de problemas complejos Desarrollo de meta-heurísticas poblacionales para la solución de problemas complejos*. PhD thesis, Universidad Central Marta Abreu de Las Villas, 2009.

[59] A. Puris and R. Bello, "Optimización basada en Mallas Dinámicas. Su aplicación en la solución de problemas de optimización continuos.," *Memorias del VI Congreso Español sobre meta-heurísticas, Algoritmos Evolutivos y Bioinspirados.* (2009) 441\n448.

[60] R. Bello, A. Puris, R. Falcón, and Y. Gómez, "Feature Selection Through Dynamic Mesh Optimization," in *Proceedings of the 13th Iberoamerican Congress on Pattern Recognition: Progress in Pattern Recognition, Image Analysis and Applications*, CIARP '08, pp. 348–355. Springer-Verlag, Berlin, Heidelberg, 2008. `http://dx.doi.org/10.1007/978-3-540-85920-8{_}43`.

[61] C. Joenathan and B. M. Khorana, "Phase Measurement by Differentiating Interferometric Fringes," *Journal of Modern Optics.* **39:10** (1992) 2075–2087.

[62] K. Gasvik, *Optical metrology*, vol. C. 2002. `http://books.google.com/books?hl=en{&}lr={&}id=6lNzbY2NlkkC{&}oi=fnd{&}pg=PR11{&}dq=Optical+Metrology{&}ots=c2OFRRp1xH{&}sig=CnzvDNlhwsnXnsqJZPRNZWeMT6A`.

[63] J. Gosling, J. Bill, S. Guy, and B. Gilad, *The Java Language Specification*. ADDISON-WESLEY, 2005.

[64] F. Gray, "Pulse code communication," 1953. `https://www.google.com/patents/US2632058`.

[65] R. W. Doran, "The Gray Code," *Journal of Universal Computer Science* **13** no. 11, (Nov, 2007) 1573–1597.

[66] L. E. Toledo, F. J. Cuevas, J. F. J. Vielma, and J. H. Sossa, "Fringe Pattern Demodulation Using Evolutionary Algorithms," in *Advanced Topics in Measurements*. InTech, 2012.