UNIVERSIDAD MIGUEL HERNÁNDEZ DE ELCHE

ESCUELA POLITÉCNICA SUPERIOR DE ELCHE

GRADO EN INGENIERÍA INFORMÁTICA EN
TECNOLOGÍAS DE LA INFORMACIÓN

"Diseño de algoritmos híbridos de optimización para la mejora del ratio de convergencia"

TRABAJO FIN DE GRADO

Septiembre -
2024

AUTOR: María José Moscardó Doménech
DIRECTOR: Héctor Francisco Migallón Gomis

AGRADECIMIENTOS

Me gustaría dar las gracias:

A mi familia, en especial a mi madre, por la paciencia, la comprensión, el apoyo incondicional y el sacrificio que ha supuesto para ella que yo haya podido llegar a la universidad y estudiar una carrera lejos de mi hogar.

A mis amigos y compañeros que me han acompañado y ayudado durante estos años, ellos son la familia que se elige.

A mi tutor Hector Migallón, por brindarme la oportunidad de poder formar parte en un proyecto de investigación más allá de este TFG durante estos dos años anteriores, por su dedicación y paciencia para abordar todos los inconvenientes que he tenido durante ese periodo y por ser un excelente docente.

RESUMEN

Este proyecto tiene como objetivo diseñar algoritmos híbridos que combinen los algoritmos SMA (Slime Mould Algorithm), HHO (Harris Hawk Optimization) y HGS (Hunger Games Search), para obtener una convergencia mejor de la que pueden alcanzar cada uno de ellos por separado.

Para ello el primer objetivo es estudiar la convergencia de cada uno de estos algoritmos a los que se les llama operadores. Es decir, la capacidad de cada uno de ellos para acercarse a la solución óptima en el menor número de evaluaciones de función posibles.

El segundo objetivo consta de evaluar la convergencia de cada uno de los modelos híbridos propuestos, y así proponer la solución que se considere óptima.

Los desarrollos realizados permitirán la inclusión futura de nuevos operadores o sustitución de los operadores propuestos.

Los modelos híbridos desarrollados seguirán tanto una filosofía estática como dinámica.

En la fase de experimentación se utilizarán varias funciones comunes para caracterizar este tipo de algoritmos (heurísticos de optimización). Y evaluarán resultados experimentales para analizar los diferentes algoritmos propuestos.

Cabe destacar que el objetivo no es diseñar un algoritmo prevalente sino diseñar un algoritmo que combine de forma óptima cada algoritmo.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

Capítulo 1
Introducción

En este capítulo se detallan los objetivos del proyecto y los objetivos personales, además de un breve resumen de la estructura del trabajo.

1.1.- Objetivos

1.1.1.- Objetivos del proyecto

Este proyecto tiene como objetivo diseñar un algoritmo híbrido que combine los algoritmos SMA (Slime Mould Algorithm), HHO (Harris Hawk Optimization) y HGS (Hunger Games Search), para obtener una mejor convergencia de la que pueden alcanzar cada uno de ellos por separado.

Para ello el primer objetivo es estudiar la convergencia de cada uno de estos algoritmos a los que se les llama operadores. Es decir, la capacidad de cada uno de ellos para acercarse a la solución óptima en el menor número de evaluaciones de función posibles.

El segundo objetivo consta de evaluar la convergencia de cada uno de los modelos híbridos propuestos, para determinar cuál de ellos presenta una ratio mayor.

Por último, se propone un modelo adaptativo de estos algoritmos híbridos capaz de explotar las fortalezas de cada operador, ya que según para qué función de optimización se aplique estos operadores el rendimiento puede variar drásticamente.

1.1.2.- Objetivos personales

En el ámbito personal, el principal objetivo es ampliar y profundizar en los conocimientos adquiridos durante el grado sobre optimización y algoritmia, adquirir una mayor soltura y comprensión del lenguaje de programación C.

1.2- Estructura del trabajo

En este proyecto se realizará una introducción para contextualizar s qué son los algoritmos heurísticos y metaheurísticos e introducir el marco teórico.

En el capítulo 3 se describe cada operador, su modelo matemático y su funcionamiento basado en procesos naturales.

En los capítulos 4 y 5 se describen los algoritmos de hibridación y el modelo adaptativo respectivamente. En ellos se detallará tanto su funcionamiento como ecuaciones necesarias y pseudocódigo.

En el capítulo 6 se recogerán las tablas con la información resultante de los experimentos realizados sobre los operadores, los modelos híbridos y el modelo adaptativo.

Por último, en el capítulo 7 se recogerán las ideas principales que desprenden de la investigación y sugerencias de mejora para futuras investigaciones.

Capítulo 2
Preliminares

En este capítulo se realizará una introducción del marco teórico de este proyecto. En primer lugar, se expondrá una breve introducción a la optimización y cuál es su relación con los problemas computacionales. Por último, se profundizará en los algoritmos heurísticos y los algoritmos metaheurísticos.

2.1.- Optimización

2.1.1.- Definición y contexto

Hoy en día, la optimización no es una herramienta que esté relegada al ámbito de la ciencia de la computación exclusivamente. Existen numerosos sectores que hacen uso de esta, como el mundo de los negocios en el proceso de toma de decisiones, por ejemplo la mejora de las rutas de transporte.

La optimización consiste en la búsqueda y selección del individuo óptimo según unos criterios concretos, dentro de un grupo de individuos a los cuales se les llama población. En el caso de las rutas de transporte, mencionadas en el párrafo anterior, se trataría de la ruta más rápida o aquella que consuma menos recursos. Todo dependerá de aquellos parámetros que se especifiquen.

2.1.2.- Problemas de optimización

Un problema de optimización es aquel donde se pretenda encontrar el valor óptimo de una función objetivo, función *fitness* o función de evaluación. Dependiendo del tipo de problema de optimización que se desee abordar, el objetivo puede ser maximizar o minimizar la función dada.

Los valores de entrada de estas funciones (variables de decisión) vienen definidos por un límite inferior y un límite superior. Es decir, se encuentran dentro de un dominio acotado. A su vez, estas funciones pueden estar sujetas a restricciones. Lo cual afectaría directamente al conjunto de soluciones factibles, reduciéndolo en este caso.

2.1.3.- Optimización continua

La optimización continua o programación matemática continua, abarca los problemas de optimización donde las variables de decisión toman valores reales. Por tanto, en estos problemas, las soluciones del tipo “fuerza bruta” no son válidas.

Cada función objetivo definirá estas variables dentro de un dominio concreto para cada una de ellas. Poniendo como ejemplo la función *Rosenbrock*, las variables de esta función están definidas dentro del siguiente intervalo $-5 \leq x_i \leq 10$, siendo i el número de variables.

Además, como se menciona en el apartado anterior, estos problemas pueden estar sujetos a restricciones. Las restricciones juegan un papel crucial en los problemas de optimización ya que reducen el conjunto de soluciones factibles.

Según la naturaleza de estas y la forma en la que afecten al problema se pueden clasificar de diversas maneras, pero las más comunes son $g_i(x)$: Restricciones de desigualdad y $h_i(x)$: Restricciones de igualdad. Estos dos tipos de restricciones, al igual que la función objetivo, pueden ser lineales o no lineales. En el caso de ser no lineales aumentarían la complejidad del problema.

La naturaleza de un problema de optimización la define el tipo de función objetivo y sus restricciones. Clasificándose así en problemas lineales o no lineales. Para que un problema sea considerado no lineal al menos una de esas funciones tiene que serlo. No importa que la función objetivo sea lineal, si una de las restricciones no lo es, se consideraría no lineal. Dicho de otro modo, para que un problema sea considerado lineal tanto su función objetivo como sus restricciones deben ser lineales.

A grandes rasgos se podría decir que los problemas lineales debido a que por lo general suelen ser problemas de menor complejidad, son más tratables con métodos deterministas y estructurados. Mientras que los problemas no lineales requieren de métodos más sofisticados y con un enfoque heurístico. Cabe puntualizar que la afinidad a un tipo de método u otro dependerá del grado de complejidad del problema, no del tipo de problema. En términos generales los problemas no lineales suponen un grado mayor de complejidad frente a los lineales.

2.2.- Algoritmos heurísticos

A medida que la complejidad del problema aumenta, aumenta a su vez de forma exponencial el tiempo requerido para llegar a una solución óptima. Los algoritmos heurísticos, en contraposición a los métodos deterministas, ofrecen una mejora significativa en el tiempo de ejecución, pero, a su vez, la solución obtenida puede distar considerablemente de la solución óptima, además, para algunos problemas, estos algoritmos son difíciles de definir.

No todas las heurísticas llegan a la solución óptima del problema. Estos algoritmos son recomendables para aquellos problemas que no requieran encontrar necesariamente la solución más efectiva, sino una solución aceptable, y que lo hagan con una frecuencia relativamente alta.

En muchos problemas de alta complejidad para llegar a la solución óptima requieren de un número elevado de evaluaciones de las diversas posibilidades. Tomando como ejemplo el juego del ajedrez, la complejidad de su árbol de búsqueda es 10^{120}. Durante una partida de ajedrez el jugador evalúa los posibles movimientos que puede realizar para intentar hallar el movimiento óptimo. Pero para ello requeriría la evaluación de todas las posibles jugadas que puede hacer y las posibles jugadas con las que el rival puede

responder. En un primer vistazo las peores jugadas llamadas "*bunders"* se descartan, pero, aun así, se despliega un amplio abanico de posibilidades. El cual será mayor o menor dependiendo de cuánto avanzada esté la partida y cómo de abierta o cerrada estén las posiciones. Por ello el jugador elegirá aquel movimiento que "aparentemente sea el mejor", pudiendo no serlo, pero sí estar cerca de él. En el caso de poder analizar todas las posibilidades se estaría en un ejemplo de "fuerza bruta".

Los algoritmos heurísticos poblacionales se pueden dividir en dos grandes grupos: Algoritmos Evolutivos e Inteligencia de enjambre. Ambos requieren de ciertos parámetros comunes como el tamaño de población, el número de generaciones, el tamaño de las variables de cada individuo de la población (es decir, el tamaño del problema), etc.

Según (Rao, 2016) los algoritmos evolutivos basan su funcionamiento en la evolución genética dentro de la biología. Son una clase de algoritmos de optimización que utilizan una población de posibles soluciones, sobre la cual aplican métodos de selección, cruce y mutación para evolucionar esta población inicial, con el fin de mejorar el conjunto de soluciones.

Los algoritmos de inteligencia de enjambre se basan en el comportamiento de los enjambres en la naturaleza, es decir, cada individuo de esta población se extrapolaría a un individuo que compone el enjambre. El funcionamiento consiste en ajustar el comportamiento de cada individuo de la población en base a unos parámetros y a la interacción con otros individuos.

Para problemas de optimización donde se requiera una mayor exactitud en la solución, los algoritmos heurísticos podrían ser insuficientes.

2.3.- Algoritmos metaheurísticos

Los algoritmos metaheurísticos, considerados una subclase de los algoritmos heurísticos por algunos autores, intentan solventar estos dos problemas de los algoritmos heurísticos.

El prefijo "meta" proviene del griego "μετά" que significa "más allá" o "sobre". El término "metaheurístico" se popularizó en la comunidad científica como una manera de describir algoritmos de optimización aplicables a una amplia gama de problemas.

Estos tipos de algoritmos son estrategias de optimización de un nivel superior. Son una mejora de estos algoritmos heurísticos, los cuales consiguen una mejor exploración del espacio de búsqueda haciendo uso de estrategias como utilizar procesos iterativos o determinísticos para poder escapar de óptimos locales. Es decir, incluyen mecanismos para evitar quedar atrapados en soluciones subóptimas.

Se tratan de algoritmos más flexibles. Aplicables a una variedad de problemas más amplio ya que no dependen de características específicas del problema y utilizan estrategias generales.

Aunque estos algoritmos supongan una mejora frente a los algoritmos heurísticos, cabe destacar que no por ello los algoritmos heurísticos hayan quedado relegados a un segundo plano. Los algoritmos heurísticos consiguen buenos resultados con los problemas de menor complejidad.

Estos algoritmos hacen uso de dos procesos clave para la búsqueda de la solución óptima: el proceso de exploración y el proceso de explotación. En el proceso de exploración se buscan nuevas áreas del espacio de soluciones para evitar caer en mínimos locales, mientras que el proceso de explotación trata de refinar las mejores soluciones encontradas, centrándose en áreas más prometedoras.

Capítulo 3
Algoritmos de optimización seleccionados

En este apartado se explicará y se detallará el pseudocódigo de los tres algoritmos metaheurísticos seleccionados: SMA (*Slime Mould Algorithm*), HHO (*Harris Hawk Optimization*) y HGS (*Hunger Games Search*).

3.1.- Algoritmo SMA

En este apartado se explica el origen y el funcionamiento del algoritmo SMA (*Slime Mould Algorithm*).

3.1.1.- Origen y descripción

El algoritmo SMA basa su funcionamiento en emular el algoritmo de búsqueda que realiza el moho *Physarum polycephalum* para encontrar alimento.
Este moho se trata de una colonia de organismos unicelulares eucariotas, que se desarrolla en lugares fríos y húmedos. Forma redes venosas amarillentas y de apariencia viscosa como se puede observar en la figura 1.

Figura 1: Slime Mould cultivado en una placa Petri

Se ha podido observar que el algoritmo de búsqueda que realiza el moho puede ser aplicable a problemas de optimización. En el estudio (Tero, 2010) se observó que este moho explora inicialmente el espacio de búsqueda con un margen relativamente contiguo para maximizar el área explorada. Resultando en una red tubular que conecta fuentes de alimento descubiertas mediante conexiones directas, puntos de conexión intermedios adicionales (puntos Steiner) que reducen la longitud total de la red de conexión y la formación de enlaces cruzados.

El crecimiento del plasmodio está limitado a las características del sustrato, puede ser limitado por barreras físicas e incluso por el régimen de luz. Así, por ejemplo *Physarum* puede encontrar el camino más corto a través de un laberinto (Figura 2).

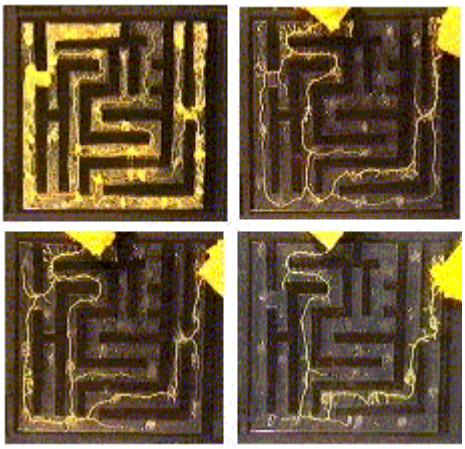

Figura 2: Slime Mould resolviendo un laberinto

Uno de los experimentos que se realizaron en esta investigación fue comprobar si el moho era capaz de replicar la red ferroviaria de los pueblos aledaños a Tokio. Para ello utilizaron una plantilla donde situaron treinta y seis fuentes de alimento en las ubicaciones donde se encontraban estas localidades, dejando crecer el moho desde Tokio haciendo uso de las limitaciones mencionadas anteriormente para representar limitaciones físicas como pueden ser montañas o lagos.

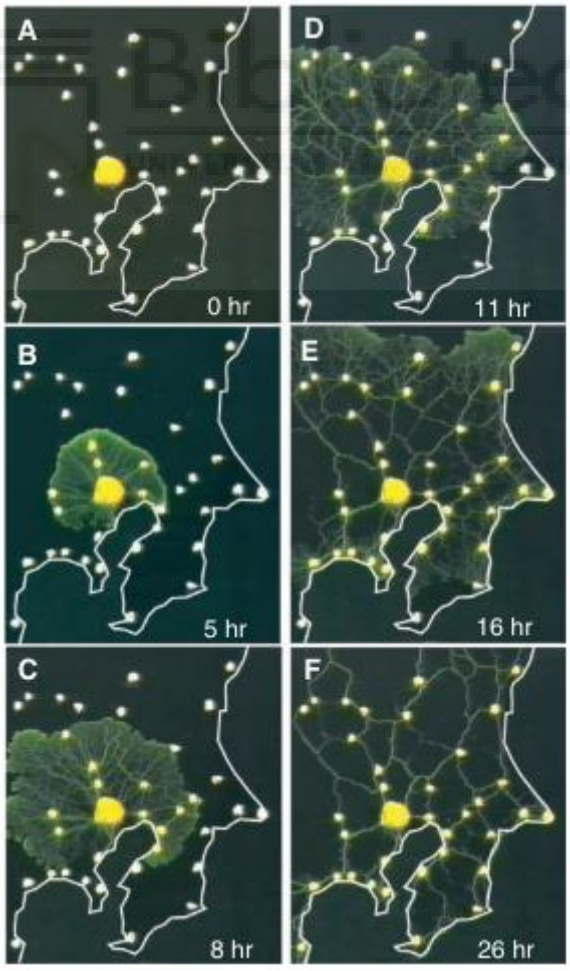

Figura 3: Proceso de resolución del Slime Mould de la red ferroviaria de Tokio

Y se pudo observar que el costo de soluciones encontradas por el moho coincidía estrechamente con el de la red ferroviaria ya existente.

Según (Shimin LI, 2020), el ciclo de vida de este organismo se basa en diferentes etapas, pero su etapa nutricional principal es el plasmodio, que es la etapa activa y dinámica. Durante esta etapa el moho busca alimentos, los rodea y secreta enzimas para digerirlos.

El moho desarrolla su estructura venosa a través de contracciones, durante su diferencia de fase.

La anchura de cada una de las venas la determina la respuesta del flujo citoplasmático del modelo del *Physarum*. El aumento del flujo de citoplasma conduce a un aumento del grosor de las venas. Si el flujo disminuye las venas se contraen disminuyendo el diámetro de estas.

El *physarum* puede construir una ruta más fuerte donde la concentración de alimento es mayor, para asegurar que obtiene la mayor concentración de nutrientes.

Estudios recientes también han revelado que el moho tiene la capacidad de hacer modificaciones en la búsqueda de alimentos basadas en la teoría de la optimización. Cuando la calidad de distintas fuentes de alimentos es diferente, puede elegir la fuente con la concentración de nutrientes más alta. No obstante, hay que tener en cuenta la velocidad y el nivel de riesgo durante la búsqueda. La experimentación ha demostrado que cuanto más rápida es la velocidad de toma de decisiones, las probabilidades de que el moho encuentre la fuente de alimento óptima son menores.

El moho debe decidir cuándo abandonar esta área y buscar en otra durante la búsqueda. Esta, junto con la capacidad de hacer rectificaciones en la búsqueda son características que se puede observar en los algoritmos metaheurísticos mencionado en el capítulo anterior.

Si no dispone de información completa, la mejor manera de saber cuándo abandonar la posición actual es adoptando reglas heurísticas o empíricas basadas en la información insuficiente disponible en ese momento. Cuando encuentra alimentos de alta calidad, se reduce la probabilidad de abandonar el área.

Debido a sus características biológicas, también puede utilizar varias fuentes de alimentos al mismo tiempo. Por tanto, aunque haya encontrado una fuente mejor, aún puede dividir una parte de su biomasa para poder explotar ambos recursos al mismo tiempo cuando encuentre alimento de mayor calidad.

El moho también puede ajustar de forma dinámica sus patrones de búsqueda según la calidad de las fuentes. Si la calidad es alta, utilizará el método de búsqueda limitado por región, para enfocar la búsqueda en las fuentes que ya se han encontrado. Si la densidad

de la fuente encontrada en un inicio es baja, la abandonará para explorar otras alternativas en la región. Esta estrategia de búsqueda adaptativa puede ser más evidente cuando distintos bloques de alimentos de diferente calidad están dispersos en una región.

3.1.2.- Modelo matemático

En este apartado se describe el modelo matemático del SMA para las fases de conseguir alimento, envolver alimento y oscilación.

3.1.2.1- Conseguir alimento

Como se explica en (Shimin LI, 2020) dependiendo del olor en el aire el moho puede acercarse al alimento. Este comportamiento puede ser expresado en las siguientes fórmulas matemáticas, las cuales tratan de replicar el modelo de contracción:

$$\overrightarrow{x(t+1)} = \begin{cases} \overrightarrow{X_b(t)} + \overrightarrow{vb} \cdot \left(\overrightarrow{W} \cdot \overrightarrow{X_A(t)} - \overrightarrow{X_B(t)}\right), & r < p \\ \overrightarrow{vc} \cdot \overrightarrow{X(t)}, & r \geq p \end{cases} \quad \textbf{(1)}$$

donde $\overrightarrow{vb}$ es un parámetro comprendido en el rango de $[-a, a]$, ver ecuación (4), $\overrightarrow{vc}$ decrece de forma lineal de uno a cero. t representa la iteración actual, $\overrightarrow{X_b}$ representa la localización individual con la mayor concentración de olor encontrada actualmente, $\vec{X}$ representa la localización del moho, $\overrightarrow{X_A}$ y $\overrightarrow{X_B}$ representan dos individuos seleccionados al azar, $\overrightarrow{W}$ representa el peso del moho.

La formula de p viene dada por la siguiente ecuación:

$$p = \tanh\ |S(i) - DF| \quad \textbf{(2)}$$

Donde $i \in 1,2,\ldots,n, S(i)$ representa la calidad de $\vec{X}, DF$ representa la mejor calidad obtenida en todas las iteraciones.

La fórmula de $\overrightarrow{vb}$ es la siguiente:

$$vb = [-a, a] \quad \textbf{(3)}$$

$$a = \operatorname{arctanh}\left(-\left(\frac{t}{\max\_t}\right) + 1\right) \quad \textbf{(4)}$$

La fórmula de $\overrightarrow{W}$ es la siguiente:

$$\overrightarrow{W(IndiceOlor(i))} = \begin{cases} 1 + r \cdot \log\left(\frac{bF - S(i)}{bF - wF} + 1\right), condición \\ 1 - r \cdot \log\left(\frac{bF - S(i)}{bF - wF} + 1\right), \ otros \end{cases} \quad \textbf{(5)}$$

$$IndiceOlor = sort(S) \quad \textbf{(6)}$$

Donde la condición indica que $S(i)$ se encuentra en la primera mitad de la población, r denota el valor aleatorio en el intervalo [0,1], max $\_t$ muestra la iteración máxima, bF representa la aptitud óptima obtenida en el proceso iterativo actual, wF representa el peor valor de aptitud obtenido en el proceso iterativo actual, $IndiceOlor$ representa el vector de valores de aptitud ordenados.

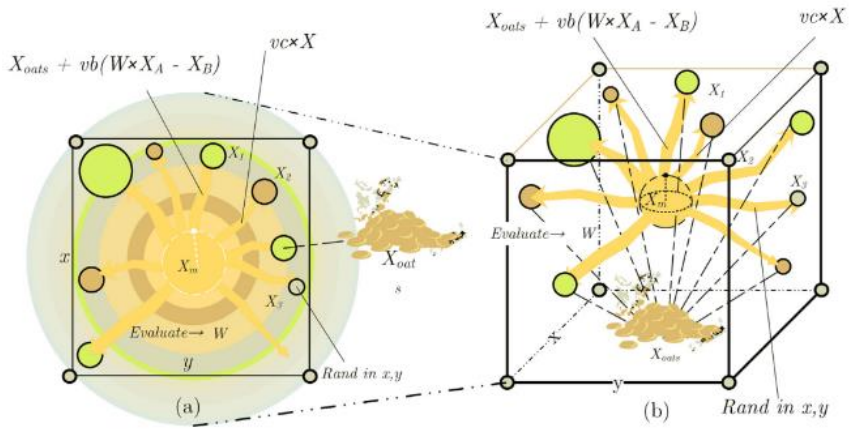

Figura 4: Posibles localizaciones en 2 dimensiones y 3 dimensiones

La figura 4 representa la aplicación de la ecuación (1). La región de búsqueda de $\vec{X}$ puede actualizarse de acuerdo con la mejor ubicación de $\overrightarrow{X_b}$ obtenida hasta el momento y el ajuste fino de los parámetros $\overrightarrow{vb}, \overrightarrow{vc}$ y $\overrightarrow{W}$ puede cambiar la ubicación del individuo.

La figura 4 se utiliza también para ilustrar el cambio de posición del individuo de búsqueda en el espacio tridimensional. La utilización de números aleatorios permite que los individuos formen vectores de búsqueda en cualquier ángulo, es decir, que exploren el espacio de soluciones en cualquier dirección, de modo que el algoritmo tenga la posibilidad de encontrar la solución óptima. Por lo tanto, la ecuación (1) permite que los individuos de búsqueda exploren en todas las direcciones posibles cerca de la solución óptima, simulando así la estructura de sector circular del moho al acercarse a la comida. También se puede aplicar extender este concepto al espacio hiperdimensional.

3.1.2.2- Envolver alimento

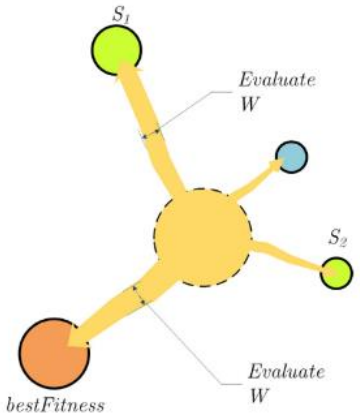

Figura 5: Evaluación de la calidad

Esta parte simula matemáticamente el modo de concentración de la estructura del tejido venoso del moho durante la búsqueda. Cuanto mayor es la concentración de comida con la que entra en contacto la vena, más fuerte es la onda generada por el bio-oscilador, más rápido fluye el citoplasma y más gruesa es la vena. La ecuación (5) simula matemáticamente la retroalimentación positiva y negativa entre el ancho de la vena del moho y la concentración de comida que se explora. El componente r en la ecuación (5) simula la incertidumbre del modo de contracción venosa. Se utiliza el logaritmo para asuavizar la tasa de cambio de valor numérico de modo que el valor de la frecuencia de contracción no cambie demasiado. Esta condición simula cómo el moho ajusta sus patrones de búsqueda según la calidad de la comida. Cuando la concentración de comida es alta, el peso cerca de la región es mayor; cuando la concentración de la comida es baja, el peso de la región se reducirá, lo que hará que explore otras regiones.

La figura 5 muestra el proceso de evaluación de los valores de aptitud para el moho.

La fórmula matemática para actualizar la ubicación del moho es la siguiente:

$$\overrightarrow{X^*} = \begin{cases} rand \cdot (UB - LB) + LB, \ rand < z \\ \overrightarrow{X_b(t)} + \overrightarrow{vb} \cdot \left(W \cdot \overrightarrow{X_A(t)} - \overrightarrow{X_{B(t)}}\right), \ r < p \\ \overrightarrow{vc} \cdot \overrightarrow{X(t)}, \ r \geq p \end{cases} \quad \textbf{(7)}$$

Dónde LB y UB denotan los limites superior e inferior del rango de búsqueda, y $rand$ y r representan valores aleatorios comprendidos entre [0,1]. El valor de z es un parámetro que se indicará a la hora de la experimentación y que será un valor muy pequeño.

3.1.2.3- Oscilación

El moho depende principalmente de la onda de propagación generada por el oscilador biológico para cambiar el flujo citoplasmático en las venas, de modo que tienda a ubicarse en la mejor posición de concentración de comida. Con el propósito de simular las variaciones del ancho de las venas del moho, se utilizan $\overrightarrow{W}, \overrightarrow{vb}$ y $\overrightarrow{vc}$ para realizar estas variaciones. $\overrightarrow{W}$ simula matemáticamente la frecuencia de oscilación del moho cerca de una concentración de alimento diferente.

De modo que el moho puede acercarse más rápidamente a la comida de alta calidad cuando la encuentra, y más lentamente cuando la concentración de comida es baja en la posición individual, mejorando así la eficiencia del moho en la elección de la fuente óptima de alimento.

3.1.3.- Pseudocódigo

En este apartado se muestra el pseudocódigo empleado para el algoritmo SMA

Inicializacion de los parametros *popsize,* Max_itereacion;
Inicializacion de la posición del *slime mould* $X_i(i = 1,2, \dots , n)$;
While ($t \leq$ Max_iteration)
    Calcular la función objetivo para todo el *slime Mould*;
    Actualizar *bestFitness,* X_b
    Calcular *W* según ecuación (5)
    For each región de búsqueda
        Actualizar *p,vb,vc*
        Actualizar la posición según ecuación (7)
    Fin for each
    $T = t+1$
End while
Return *bestFitness,* X_b

Figura 6: Pseudocódigo del SMA

Cabe remarcar que el algoritmo SMA ha sido utilizado con éxito en numerosas aplicaciones reales, por ejemplo, para calcular la calidad y el coste en construcciones como se puede apreciar en el artículo (Son, 2023) .

3.2.- Algoritmo HHO

En este apartado se describe el origen y el funcionamiento del algoritmo HHO (*Harris Hawk Optimization*).

3.2.1.- Origen y descripción

El halcón Harris *parabuteo unicinctus*, figura 7, es un ave rapaz que habita la mitad sur de Arizona. Algo que convierte a estas aves en una especie única es su particular forma de cazar y conseguir alimento. El halcón Harris caza en conjunto y comparte el alimento. Este método de caza se ha podido observar previamente solo en ciertas especies de mamíferos carnívoros.

Figura 7: Halcón Harris

Estas aves son capaces de reconocer a sus familiares y organizar reuniones para cenar estando fuera de la época reproductiva.

La literatura (Ali Asghar Heidari, 2019) habla de que los halcones Harris inician su caza al amanecer, posándose sobre grandes árboles, nidos o grandes postes eléctricos, y permanecen pendientes de los movimientos de los otros miembros del grupo. El grupo se reúne y un halcón tras otro realiza vuelos cortos y se vuelven a posar. Resultando en un movimiento de “salto de rana” donde los miembros de un equipo se mueven de manera intercalada para avanzar y asegurar una posición por toda la región de búsqueda. Se reúnen y dividen varias veces para buscar de forma activa el animal cubierto, generalmente un conejo.

La principal estrategia para capturar a una presa de estas aves es atacar por sorpresa. Para ello varios miembros atacaran desde diferentes posiciones cuando la presa haya sido detectada. En algunas ocasiones y dependiendo de las características de la presa, esta podrá escabullirse. Los halcones Harris presentan varios estilos de persecución, que adaptarán según las circunstancias. Uno de ellos es la táctica de cambio, cuando el mejor individuo falla en la captura la persecución continuará por otro de los miembros del equipo. Por otro lado, al tratarse de un método de caza cooperativo pueden perseguir al conejo hasta el agotamiento.

En el algoritmo HHO los halcones serían las soluciones candidatas y la mejor solución candidata en cada paso representaría a la presa.

3.2.2.- Modelo matemático

En este apartado se describe el modelo matemático del proceso explicado en el apartado anterior.

3.2.2.1- Fase de exploración

Como se explica en (Ali Asghar Heidari, 2019) Los halcones se posan sobe ubicaciones aleatorias y esperan a alertar una presa basada en dos estrategias. Considerando que estas dos estrategias tengan la misma probabilidad q, los halcones se posicionaran lo suficientemente cerca de los otros miembros de la familia (para estar lo suficientemente cerca de ellos al atacar) y del conejo. Como está modelado en la ecuación (8) para la condición $q < 0{,}5$ o en el caso de que se posen sobre arboles al azar como está modelado en la ecuación (8) para la condición $q \geq 0{,}5$

$$X(t+1) = \begin{cases} X_{rand}(t) - r_1 |X_{rand}(t) - 2r_2 X(t)| & q \geq 0{,}5 \\ \left(X_{rabit}(t) - X_m(t)\right) - r_3 (LB + r_4 (UB - LB)) & q < 0{,}5 \end{cases} \quad \textbf{(8)}$$

Dónde $X(t+1)$ es el vector de posición del halcón en la siguiente iteración t, $X_{rabbit}(t)$ es la posición del conejo, $X(t)$ es el vector de posición actual de los halcones, r_1, r_2, r_3, r_4, y q son números aleatorios entre $[0,1]$ que se actualizan en cada iteración, LB y UB son los limites inferior y superior de las variables, $X_{rand}(t)$ es un halcón elegido al azar de la población actual, y X_m es la posición media de la población actual.

La posición media de los halcones se obtiene según la ecuación (9):

$$X_m(t) = \frac{1}{N} \sum_{i=1}^{N} X_i(t) \quad \textbf{(9)}$$

Dónde $X_i(t)$ indica la localización para cada halcón en la iteración t y N corresponde al número total de halcones.

3.2.2.2- Transición entre fases de exploración

El algoritmo HHO puede hacer la transición entre fase de exploración a fase de exploración, y luego cambiar entre diferentes comportamientos explorativos según la energía de escape de la presa. La energía viene modelada en la ecuación (10):

$$E = 2E_0\left(1 - \frac{t}{T}\right) \quad \textbf{(10)}$$

Donde E indica la energía de escape de la presa, T es el número máximo de iteraciones, y E_0 es el estado inicial de su energía.

3.2.2.3- Fase de exploración

En esta fase los halcones realizarán el ataque a la presa detectada. La presa va a intentar escapar, y en situaciones reales ocurren diferentes estilos de persecución. Según los comportamientos de escape de la presa y las estrategias de persecución de los halcones, se proponen cuatro posibles estrategias en el HHO para modelar la etapa de ataque.

Suponiendo que r es la probabilidad de que una presa escape con éxito ($r < 0{,}5$) o no lo haga $r(\geq 0{,}5)$ antes del ataque. Hagan lo que hagan las presas, los halcones siempre realizarán un asedio duro o suave dependiendo de la energía de escape de la presa. En una situación real los halcones se acercarían cada vez más a la presa. Después de varios minutos, la presa que escapa perderá cada vez más energía; entonces los halcones intensificarán el proceso de asedio para atrapar sin esfuerzo a la presa ya agotada. Para poder modelar esta estrategia permitiendo que el HHO cambie entre procesos de asedio duro y suave, se utiliza el parámetro E. Cuando $|E| \geq 0{,}5$, ocurre el asedio suave, y cuando $|E| < 0{,}5$ ocurre el asedio duro.

I. Asedio suave

Cuando $r \geq 0{,}5$ y $|E| \geq 0{,}5$, el conejo todavía tiene suficiente energía e intenta escapar mediante algunos saltos engañosos aleatorios, pero finalmente no lo logra. Durante estos intentos, los halcones lo rodean suavemente para agotar más al conejo y luego realizan el ataque sorpresa. Este comportamiento viene modelado en la ecuación (11) y la ecuación (12):

$$X(t+1) = \Delta X(t) - E|JX_{rabbit}(t) - X(t)| \quad \textbf{(11)}$$

$$\Delta X(t) = X_{rabbit}(t) - X(t) \quad \textbf{(12)}$$

Dónde $\Delta X(t)$ es la diferencia entre el vector de posición del conejo y la posición actual en la iteracion t, r_5 es un número aleatorio entre $[0{,}1]$ y $J = 2(1 - r_5)$ representa la fuerza del salto aleatorio del conejo durante el proceso de escape.

II. Asedio duro

Cuando $r \geq 0{,}5$ y $|E| < 0{,}5$, la presa está muy agotada y tiene una energía baja de escape. Además, los halcones rodean a la presa firmemente para realizar el ataque sorpresa. Las posiciones actuales se actualizan utilizando la ecuación (13).

$$X(t+1) = X_{rabbit}(t) - E|\Delta X(t)| \quad \textbf{(13)}$$

III. Asedio suave con inmersiones rápidas progresivas

Cuando $|E| \geq 0{,}5$ pero $r < 0{,}5$ el conejo tiene la suficiente energía para escapar con éxito y aún se está realizando un asedio suave antes del ataque.

Para poder modelar los patrones de "salto de rana" se utiliza el concepto *levy flight* (LF) en el algoritmo HHO. LF es utilizado para imitar los movimientos engañosos de las presas.

Para realizar un asedio suave, se supuso que los halcones pueden evaluar y decidir su próximo movimiento basado en la regla de la ecuación (14):

$$Y = X_{rabbit}(t) - E|JX_{rabbit}(t) - X(t)| \quad \textbf{(14)}$$

Después comparan el posible resultado de dicho movimiento con la inmersión anterior para detectar si será una buena inmersión o no. Si el movimiento no es razonable debido a que la presa esta realizando movimientos muy abruptos, estos también empiezan a realizar inmersiones irregulares y rápidos. Se lanzarán en picado basándose en los patrones de LF según la ecuación (15):

$$Z = Y + S \times LF(D) \quad \textbf{(15)}$$

Donde D es la dimensión del problema y S es un vector al azar de tamaño $1 \times D$ y LF es la función *Lévy flight* la cual se calcula según la ecuación (16):

$$LF(x) = 0{,}01 \times \frac{u \times \sigma}{|v|^{\beta}}, \left(\frac{\Gamma(1+\beta) \times \sin\frac{\pi\beta}{2}}{\Gamma\left(\frac{1+\beta}{2}\right) \times \beta \times 2^{\left(\frac{\beta-1}{2}\right)}} \right)^{\frac{1}{\beta}} \quad \textbf{(16)}$$

Dónde u y v son valores aleatorios dentro del intervalo $[0,1]$ y β es una constante predeterminada establecida en 1,5, σ es el valor que controla la escala de la distribución de Lévy y Γ es el índice de la distribución de Lévy.

La ecuación final para establecer la posición de los halcones durante esta estrategia es la ecuación (17):

$$X(t+1) = \begin{cases} Y & if\ F(Y) < F(X(t)) \\ Z & if\ F(Z) < F(X(t)) \end{cases} \quad \textbf{(17)}$$

Dónde Y y Z se obtienen mediante las ecuaciones (14) y (15).

IV. Asedio duro con inmersiones rápidas progresivas

Cuando $|E| < 0{,}5$ y $r < 0{,}5$ el conejo no tiene suficiente energía para escapar y se forma un asedio duro antes del ataque sorpresa para atrapar y matar a la presa. La situación de la presa en este caso es similar a la del asedio suave, pero esta vez los halcones intentan reducir su distancia entre su ubicación promedio y la presa. En este caso se aplica la regla modelada en la anteriormente en la ecuación (17), pero esta vez Y y Z se obtienen mediante la ecuación (18):

$$Y = X_{rabbit}(t) - E|JX_{rabbit(t)} - X_m(t)| \quad \textbf{(18)}$$

Dónde $X_m(t)$ se obtiene mediante la ecuación (9) y Z de la ecuación (15).

3.2.3.- Pseudocódigo

Calcular la función objetivo para toda la población de halcones
Establecer X_{rabbit} como la la mejor localización del conejo
For each halcón X_i
    Actualizar el valor inicial de E_0 y la fuerza del salto J
    Actualizar E según la ecuación (10)
    If ($|E| \geq 1$)
        Actualizar el vector de ubicación según la ecuación (8)
    If($|E| < 1$)
        If ($r \geq 0{,}5\ y\ |E| \geq 0{,}5$)
            Actualizar el vector de posición según la ecuación (11)
        Else if ($r \geq 0{,}5\ y\ |E| < 0{,}5$)
            Actualizar el vector de posición según la ecuación (13)
        Else if ($r < 0{,}5\ y\ |E| \geq 0{,}5$)
            Actualizar el vector de posición según la ecuación (17)
        Else if ($r < 0{,}5\ y\ |E| < 0{,}5$)
            Actualizar el vector de posición según la ecuación (17)
        End if
    End if
Return X_{rabbit}

Figura 8: Pseudocódigo del HHO

3.3.- Algoritmo HGS

En este apartado se describe el origen y el funcionamiento del algoritmo HGS (*Hunger Games Search*).

3.3.1.- Origen y descripción

El algoritmo de optimización HGS está inspirado en la competencia de recursos, esencialmente de alimentos, observable en el reino animal. Basándose en los comportamientos y en las estrategias de los animales cuando hay escasez de alimentos, donde la competencia y la necesidad de tomar decisiones rápidas y eficientes son crucial para la supervivencia.

Este método sigue el concepto simple del "hambre" como la necesidad más crucial, siendo la motivación detrás de los comportamientos y decisiones en la vida de los animales. Para ello se implementa el hambre como una característica más utilizando un peso adaptativo para simular como afecta el hambre en cada paso de la búsqueda.

3.3.2.- Modelo matemático

En este apartado se describe el modelo matemático del proceso explicado en el apartado anterior que Según (Yutao Yang, 2021) se modela con las siguientes etapas.

3.3.2.1.- Aproximación al alimento

Los animales sociales a menudo cooperan entre sí durante la búsqueda de alimento, pero no se excluye la posibilidad de que algunos individuos no cooperen.

La regla principal del algoritmo HGS se modela en la ecuación (19) para la comunicación cooperativa individual y el comportamiento de búsqueda de alimento.

$$X(t+1) = \begin{cases} Game_1: \overrightarrow{X(t)} \times \left(1 + randn(1)\right), r_1 < l \\ Game_2 \overrightarrow{W_1} \times \overrightarrow{W_b} + \vec{R} \times \overrightarrow{W_2} \times \left|\overrightarrow{X_b} - \overrightarrow{X(t)}\right| , r_1 > l,\ r_2 > E \\ Game_3: \overrightarrow{W_1} \times \overrightarrow{X_b} - \vec{R} \times \overrightarrow{W_2} \times \left|\overrightarrow{X_b} - \overrightarrow{X(t)}\right|, r1 > l, r_2 < E \end{cases} \quad \textbf{(19)}$$

Dónde $\vec{R}$ está comprendida en el rango [0,2] y se calcula según la ecuación (21), r_1 y $r_2$representan dos números aleatorios en el rango [0,1], $randn(1)$ es un número aleatorio según la distribución normal, t indica la iteración actual, $\overrightarrow{W_1}$ y $\overrightarrow{W_2}$ representan los pesos del hambre, $\overrightarrow{X_b}$ es la ubicación del mejor individuo de esa iteración, $\overrightarrow{X(t)}$ es la ubicación de cada individuo.

La expresión $\overrightarrow{X(t)} \times (1 + randn(1))$ indica como un individuo hambriento busca comida de forma aleatoria en la ubicación actual.

La expresión $|\overrightarrow{X_b} - \overrightarrow{X(t)}|$ representa el rango de actividad del individuo actual que es multiplicado por $\overrightarrow{W_2}$ influenciando el hambre en el factor de actividad.

$\vec{R}$ es un controlador de rango añadido para limitar el rango de actividad, en el cual el rango de $\vec{R}$ se reduce gradualmente a 0.

Sumar o restar el rango de activad basado en $\overrightarrow{W_1} \times \overrightarrow{X_b}$ simula al individuo actual siendo informado por sus compañeros al llegar a la ubicación del alimento, y luego buscando nuevamente en la ubicación actual después de adquirir el alimento. $\overrightarrow{W_1}$ se introduce como el error en captar la posición real en la actualidad. E es un control de variación para todas las posiciones que viene dado por la ecuación (20).

$$E = \text{sech}(|F(i) - BF|) \quad \textbf{(20)}$$

Dónde $i \in 1,2,\dots,n$, $F(i)$ representa el valor de la función objetivo para cada individuo, BF es el valor óptimo obtenido hasta la iteración actual.

La fórmula para $\vec{R}$ viene dada por la ecuación (21).

$$\vec{R} = 2 \times shrink \times rand - shrink \quad \textbf{(21)}$$

$$shrink = 2 \times \left(1 - \frac{t}{T}\right) \quad \textbf{(22)}$$

Dónde $rand$ es un numero aleatorio en el rango de $[0,1]$ y T representa el número máximo de iteraciones.

3.3.2.2.- El papel del hambre

El nivel de inanición de los individuos es simulado según el modelo matemático que se propone a continuación.

La fórmula para $\overrightarrow{W_1}$ en la ecuación (19) es la siguiente:

$$\overrightarrow{W_1(i)} = \begin{cases} hungry(i) \times \frac{N}{SHungry} \times r_4, & r_3 < l \\ 1 r_3 > l \end{cases} \quad \textbf{(23)}$$

La fórmula para $\overrightarrow{W_2}$ en la ecuación (19) es la siguiente:

$$W_2(i) = (1 - \exp(-|hungry(i) - SHungry|)) \times r_5 \times 2 \quad \textbf{(24)}$$

Dónde $hungry$ representa el hambre de cada individuo, N representa el número de individuos, $SHungry$ es el sumatorio del hambre de todos los individuos, r_3, r_4 y r_5 son números aleatorios en el rango de $[0,1]$.

La fórmula para $hungry(i)$ es la siguiente:

$$hungry(i) \begin{cases} 0, & AllFitness(i) == BF \\ hungry(i) + H, & Allfitness(i)! = BF \end{cases} \quad \textbf{(25)}$$

Dónde $AllFitness(i)$ preserva el valor de la función objetivo (la aptitud) de cada individuo en la iteración actual. En cada iteración el hambre del mejor individuo se establece en 0. Para los otros individuos se añade un nuevo valor de hambre H basado en el hambre original. Por lo tanto, se entiende que el H correspondiente a diferentes individuos será diferente.

La fórmula para H se encuentra en la ecuación (27):

$$TH = \frac{F(i)-BF}{WF-BF} \times r_6 \times 2 \times (UB - LB) \quad \textbf{(26)}$$

$$H = \begin{cases} LH \times (1+r), & TH < LH \\ TH, & TH \geq LH \end{cases} \quad \textbf{(27)}$$

Dónde r_6 es un valor aleatorio en el rango de $[0,1]$, $F(i)$ representa el valor de la función objetivo para cada individuo, BF es el mejor valor obtenido de entre todos los individuos hasta la iteración actual, WF es el peor valor obtenido de entre todos los individuos en la iteración actual (hasta ese momento), UB y LB son los limites superior e inferior.

3.3.3.- Pseudocódigo

Inicialización de los parámetros $N, T, l, D, SHungry$
Inicialización de las posiciones de los individuos $X_i (i = 1,2, \dots, N)$
While (t ≤ T)
    Calcular la función objetivo de cada individuo
    Actualizar BF, WF, X_b, BI
    Calcular $hungry$ según la ecuación (25)
    Calcular W_1 según la ecuación (23)
    Calcular W_2 según la ecuación (24)
    For each individuos
        Calcular E según la ecuación (20)
        Actualizar R según la ecuación (21)
        Actualizar las posiciones según ecuación (19)
    End for
    $t = t + 1$
End while
Return BF, X_b

Figura 9: Pseudocódigo del HGS

Capítulo 4
Métodos híbridos diseñados

En este capítulo se describirán tres algoritmos híbridos que combinan los operadores: HHO, SMA y HGS, los cuales fueron definidos en el capítulo anterior. Estos operadores son la base para el desarrollo de los algoritmos que se describen en este capítulo. Todas las versiones parten de la idea de combinar los operadores de distintas formas para comprobar si existe una mejora de la convergencia media.

4.1.- Algoritmo híbrido 1: Algoritmo Híbrido Aleatorio de Operadores con Población Completa

En esta sección se describe el funcionamiento del Algoritmo híbrido 1. En el cual en cada iteración la población será procesada por un algoritmo diferente.

4.1.1 Descripción

Como se ha dicho, el algoritmo diseñado para esta versión propone que por cada iteración la población sea procesada por el mismo algoritmo en su totalidad.

La población está representada por una matriz de dimensiones $N \times m$, donde N equivale al número de individuos y m al número de variables. A su vez esta matriz va asociada a un vector $of(X)$ el cual almacena el valor de la función objetivo de cada individuo.

Estas variables no son semejantes a lo que es conocido como variable en el ámbito de la programación. Sino que hacen referencia a una serie de atributos cuantificables que son relevantes para la evaluación del individuo dentro de la búsqueda. Tomando una matriz como un vector de vectores cada fila de la matriz sería un vector que almacena los atributos de un individuo.
Por comodidad y para evitar confusiones a estas variables se les llamará atributos o dimensión de la función.

El número de atributos dependerá de cuál sea la función objetivo, ya que existen algunas donde el número de atributos ya está predefinido, como puede ser la función *Foxholes* o la función *Michalewicz*. Por tanto, la matriz no siempre será del mismo tamaño.

Al inicio de cada iteración se calcularán los valores de los parámetros de cada operador. El algoritmo está diseñado para recorrer la matriz individuo a individuo y ser procesados uno a uno por los operadores. Para ello se utiliza una estructura de control de tipo *switch* que dependerá de un valor $decision$ en el rango de 1 a 3, se decidirá qué operador procesará la matriz esa iteración. Siendo el 1 el algoritmo SMA, el 2 el algoritmo HHO y el 3 el algoritmo HGS. Este valor se calculará según la ecuación (28):

$$decision = (t \ mod \ 3) + 1 \quad \textbf{(28)}$$

Siendo t la iteración actual.

Con esta ecuación, aunque los individuos sean procesados uno a uno, se asegura que siempre sean procesados por el mismo operador durante la iteración actual. Y este operador se irá alternando de manera secuencial según la iteración en la que se encuentre.

4.1.2 Pseudocódigo

Inicializar $UB, LB, n_{vars}, N, F(i), matrix, dest_{fitness}, fes,$ max $\_fes$
Calcular $F(i)$ para todos los individuos
While (fes<max_fes)
    Calcular T // T=max_fes/N
    Actualizar *dest_fitness, best_pos, order, index*
    Ordenar los vectores *order, index* de manera ascendente
    Actualizar *bigger, lowest y worstAllTime*
    Calcular *s y W*
    Inicializar W_1 *y* W_2 a 1
    Inizializar *hungry* y sumHungry a 0
    For i desde 0 hasta N-1
        If (*dest_fitness == F(i))*
            hungry(i) =0;
            Actualizar *tempPosition*
        Else
            Calcular *c*
            If (*c* < 100)
                Calcular *b*
            Else
                b = c
            End if
            hungry(i)+=b
            sumHungry += hungry(i)
        End if
    End for
    Calcular W_1 *y* W_2
    For i desde 0 hasta N-1
        Calcular *decision* según ecuación (28)
        Switch *decision*
            Case 1
                SMA(i)
            Case 2
                HHO(i)
            Case 3
                HGS(i)
        End switch
    End for
    Comprobar que las variables estén dentro de *UP* y *LB*
    Calcular $F(i)$
End while
    Return *dest_fitness*

Figura 10: Pseudocódigo del Algoritmo híbrido1

4.2.- Algoritmo híbrido 2.1: Algoritmo Híbrido de Operadores con Subpoblaciones Fijas

Esta sección explica y describe el funcionamiento del algoritmo híbrido 2.1, en el cual la población se dividirá en tantas subpoblaciones como operadores, en este caso 3, y cada subpoblación será procesada siempre por el mismo operador.

4.2.1 Descripción

El algoritmo diseñado para esta versión, propone que, por cada iteración, la matriz sea dividida en tantas subpoblaciones como número de operadores existan, en el caso de este proyecto, serán tres de tamaño $N_{sup} = N/3$. Siendo N el número de individuos. Cada subpoblación será procesada por un operador distinto, y estas siempre serán procesadas por el mismo operador durante todas las iteraciones.

Al inicio de cada iteración se calcularán los valores de las variables de cada operador. Posteriormente se recorrerá la matriz individuo a individuo y mediante una estructura de control de tipo *switch* que dependerá de un valor $decision$ en el rango de 1 a 3, se decidirá qué operador procesará ese individuo concreto. Siendo el 1 el algoritmo SMA, el 2 el algoritmo HHO y el 3 el algoritmo HGS. Este valor se calculará según la ecuación (29):

$$decision = \frac{i}{\frac{N}{3}} + 1 \quad \textbf{(29)}$$

Donde $i \in 1,2,\dots,N,$ representando el individuo que se está procesando y N el número de individuos.

Esta ecuación permite asignar siempre los mismos individuos a cada operador sin necesidad de dividir y acotar realmente la matriz en subpoblaciones.

4.2.2 Pseudocódigo

Inicializar $UB, LB, n_{vars}, N, F(i), matrix, dest_{fitness}, fes,$ max $\_fes$
Calcular $F(i)$ para todos los individuos
While (fes<max_fes)
    Calcular T // T=max_fes/N
    Actualizar *dest_fitness, best_pos, order, index*
    Ordenar los vectores *order, index* de manera ascendente
    Actualizar *bigger, lowest y worstAllTime*
    Calcular *s y W*
    Inicializar W_1 y W_2 a 1
    Inizializar *hungry* y sumHungry a 0

```
        For i desde 0 hasta N-1
            If (dest_fitness == F(i))
                hungry(i) =0;
                Actualizar tempPosition
            Else
                Calcular c
                If (c < 100)
                    Calcular b
                Else
                    b = c
                End if
                hungry(i)+=b
                sumHungry += hungry(i)
            End if
        End for
        Calcular W1 y W2
        For i desde 0 hasta N-1
            Calcular decision según la ecuación (29)
            Switch decision
                Case 1
                    SMA(i)
                Case 2
                    HHO(i)
                Case 3
                    HGS(i)
            End switch
        End for
        Comprobar que las variables estén dentro de UP y LB
        Calcular F(i)
End while
Return dest_fitness
```

Figura 11: Pseudocódigo del Algoritmo híbrido 2.1

4.3.- Algoritmo híbrido 2.2: Algoritmo Híbrido Secuencial de Operadores con Subpoblaciones

Esta sección explica y describe el funcionamiento del algoritmo híbrido 2.2, en el cual la población se dividirá en tantas subpoblaciones como operadores, en este caso 3, y cada subpoblación será procesada por un algoritmo diferente en cada iteración, siguiendo una rotación estática.

4.3.1 Descripción

El algoritmo diseñado para esta versión propone que, por cada iteración, la matriz sea dividida en tantas subpoblaciones como número de operadores existan, en el caso de este proyecto serán tres de tamaño $N_{sup} = N/3$. Siendo N el número de individuos. Cada subpoblación será procesada por un operador distinto, en cada iteración de manera secuencial.

Al inicio de cada iteración se calcularán los valores de las variables de cada operador. Posteriormente se recorrerá la matriz individuo a individuo y mediante una estructura de control de tipo *switch* que dependerá de un valor $decision$ en el rango de 1 a 3, se decidirá qué operador procesará ese individuo concreto. Siendo el 1 el algoritmo SMA, el 2 el algoritmo HHO y el 3 el algoritmo HGS. Este valor se calculará según la ecuación (30).

$$decision = \left(\left(\frac{i}{\frac{N}{3}} + 1\right) + (t-1)\right) mod\ 3 + 1 \quad \textbf{(30)}$$

Donde $i \in 1,2,\dots,N,$ representando el individuo que se está procesando y N el numero total de individuos.

Esta ecuación es una variación de la ecuación (29), añadiendo la expresión $t-1$ permitirá que cada iteración la subpoblación sea procesada por un operador distinto de manera secuencial.

4.3.2 Pseudocódigo

Inicializar $UB, LB, n_{vars}, N, F(i), matrix, dest_{fitness}, fes,$ max $\_fes$
Calcular $F(i)$ para todos los individuos
While (fes<max_fes)
    Calcular T // T=max_fes/N
    Actualizar *dest_fitness, best_pos, order, index*
    Ordenar los vectores *order, index* de manera ascendente
    Actualizar *bigger, lowest y worstAllTime*
    Calcular *s y W*
    Inicializar W_1 *y* W_2 a 1
    Inizializar *hungry* y sumHungry a 0
    For i desde 0 hasta N-1

```
            If (dest_fitness == F(i))
                hungry(i) =0;
                Actualizar tempPosition
            Else
                Calcular c
                If (c < 100)
                    Calcular b
                Else
                    b = c
                End if
                hungry(i)+=b
                sumHungry += hungry(i)
            End if
        End for
        Calcular W1 y W2
        For i desde 0 hasta N-1
            Calcular decision según la ecuación (30)
            Switch decision
                Case 1
                    SMA(i)
                Case 2
                    HHO(i)
                Case 3
                    HGS(i)
            End switch
        End for
        Comprobar que las variables estén dentro de UP y LB
        Calcular F(i)
End while
Return dest_fitness
```

Figura 12 : Pseudocódigo del Algoritmo híbrido 2.2

4.4.- Algoritmo híbrido 3: Algoritmo Híbrido con Asignación Aleatoria de Individuos

4.4.1 Descripción

El algoritmo diseñado para esta versión propone que por cada iteración se recorrerá la matriz de población y cada individuo será procesado por un operador diferente dependiendo de un número aleatorio que puede tomar valores del 1 al 3.

A diferencia de la versión 2, la matriz no se divide en 3 subpoblaciones exactas que irán rotando en función de las iteraciones, sino que esa subpoblación cada vez tendrá un tamaño distinto, puesto que dependerá de un numero aleatorio que se calculará para cada individuo a la hora de ser procesado. Teóricamente, estos tamaños deberán ser similares ya que existe la misma probabilidad para cada número aleatorio.

Al inicio de cada iteración se calcularán los valores de las variables de cada operador. Posteriormente se recorrerá la matriz individuo a individuo y mediante una estructura de control de tipo *switch* que dependerá de un valor $decision$ en el rango de 1 a 3, se decidirá qué operador procesará ese individuo concreto. Siendo el 1 el algoritmo SMA, el 2 el algoritmo HHO y el 3 el algoritmo HGS. Este valor se calculará según la ecuación (31).

$$decision = (rand \; mod \; 3) + 1 \quad \textbf{(31)}$$

Donde $rand$ es un numero entero aleatorio en el rango de $[0, RAND\_MAX]$, RAND_MAX toma el máximo valor de un entero de 32 bits.

Con esta versión se proporciona una variación de las subpoblaciones. Es decir, los individuos que conforma cada subpoblación varían en cada iteración, y a su vez no siguen una rotación como en las versiones vistas anteriormente.

4.4.2 Pseudocódigo

Inicializar $UB, LB, n_{vars}, N, F(i), matrix, dest_{fitness}, fes,$ max $\_fes$
Calcular $F(i)$ para todos los individuos
While (fes<max_fes)
    Calcular T // T=max_fes/N
    Actualizar *dest_fitness, best_pos, order, index*
    Ordenar los vectores *order, index* de manera ascendente
    Actualizar *bigger, lowest y worstAllTime*
    Calcular *s y W*
    Inicializar W_1 y W_2 a 1
    Inizializar *hungry* y sumHungry a 0

```
        For i desde 0 hasta N-1
            If (dest_fitness == F(i))
                hungry(i) =0;
                Actualizar tempPosition
            Else
                Calcular c
                If (c < 100)
                    Calcular b
                Else
                    b = c
                End if
                hungry(i)+=b
                sumHungry += hungry(i)
            End if
        End for
        Calcular W1 y W2
        For i desde 0 hasta N-1
            Calcular decision según la ecuación (31)
            Switch decision
                Case 1
                    SMA(i)
                Case 2
                    HHO(i)
                Case 3
                    HGS(i)
            End switch
        End for
        Comprobar que las variables estén dentro de UP y LB
        Calcular F(i)
End while
Return dest_fitness
```

Figura 13: Pseudocódigo del Algoritmo híbrido 3

Capítulo 5 Método adaptativo

En este capítulo se describirán dos versiones del modelo adaptativo que combinan los operadores HHO, SMA y HGS, que fueron definidos en el capítulo 3. Ambos modelos incorporan estrategias dinámicas de asignación de subpoblaciones.

5.1.- Descripción

Cómo se ha podido observar en el capítulo 4, cada método combinatorio está diseñado de manera que procese una subpoblación del mismo tamaño o tamaños muy similares.

Con el método adaptativo basado en ENMODE (Karam M. Sallam, 2020) se pretende mejorar el reparto de las subpoblaciones. Ya que para ello este método incorpora estrategias donde el tamaño de la subpoblación asignada a cada algoritmo irá variando, dependiendo de cuál de ellos esté dando mejores resultados durante las iteraciones, asignando de esta manera un mayor número de individuos al algoritmo con mejores resultados.

5.2.- EnMODE

Según (Karam M. Sallam, 2020) Estas estrategias se basan en cuantificar la mejora de cada operador mediante dos indicadores: la calidad de las soluciones y la diversidad de cada subpoblación.

El algoritmo EnMODE comienza dividiendo la población inicial en varias subpoblaciones cada una de las cuales evoluciona utilizando su propia variante DE (*Differential Evolution*). Basado en el índice de mejora de cada operador, el tamaño de la subpoblación varía de forma adaptativa. Al final de cada generación todas las subpoblaciones se reúnen y luego se vuelven a dividir según los nuevos tamaños de subpoblación. Además, se lleva a cabo una reducción lineal del tamaño de la población, donde se establece la población inicial en un valor grande y al inicio del proceso evolutivo y después se reduce linealmente a un valor pequeño.

El rendimiento relativo de un operador DE puede variar durante las etapas de optimización; es decir, el rendimiento de un operador puede funcionar bien para ciertos tipos de problemas, pero su rendimiento puede ser deficiente para otros. Como resultado, es necesario utilizar un DE de múltiples operadores, dando más peso al que tenga un mejor rendimiento en cada etapa evolutiva.

El EnMODE utiliza las siguientes ecuaciones para poder calcular los indicadores de calidad y diversidad y así poder calcular las nuevas subpoblaciones.

La reducción lineal de la población se calcula mediante la ecuación (32):

$$NP_{t+1} = round[\left(\frac{NP_{min} - NP_{init}}{MAX_{FES}}\right) \times FES + NP_{init}] \quad \textbf{(32)}$$

Dónde NP_{t+1} es el valor de la población inicial en la próxima iteración, NP_{min} es el valor mínimo de individuos que puede contener la población, FES es el número actual de evaluaciones de función, MAX_{FES} es el número máximo de evaluaciones de función.

Para la calidad de las soluciones se utiliza el mejor individuo al final de cada generación en cada subpoblación, basado en el cual se calcula la tasa de calidad según la ecuación (33):

$$Qual_{op} = \frac{f_{t,op}^{best}}{\sum_{op=1}^{n_{op}} f_{t,op}^{best}}, \forall op = 1,2,\dots,n_{op} \quad \textbf{(33)}$$

Dónde $f_{t,op}^{best}$ es el mejor valor de la función objetivo obtenido por cada operador op durante la iteración t. En caso de trabajar con funciones con restricciones $f_{t,op}^{best}$ no será solo el individuo optimo, si no que también se tendrá en cuenta las violaciones de las restricciones.

De manera similar, para la diversidad obtenida de cada operador, se calcula la desviación media de cada solución obtenida de cada operador.

$$Div_{op} = \frac{1}{NP_{op}}\left(\sum_{i=1}^{NP_{op}} dis\left(\vec{y}_{op}, i - \vec{y}_{op}^{best}\right)\right), \forall op = 1,2,\dots,n_{op} \quad \textbf{(34)}$$

Dónde $dis\left(\vec{y}_{op}, i - \vec{y}_{op}^{best}\right)$ es la distancia Euclídea entre la solución i y la mejor obtenida por ese operador op.

Teniendo la diversidad, el índice de diversidad se calcula según la ecuación (35):

$$DI_{op} = \frac{Div\_op}{\sum_{op=1}^{n_{op}} Div\_op}, \forall op = 1,2,\dots,n_{op} \quad \textbf{(35)}$$

Basándose en las anteriores ecuaciones el índice de mejora se calcula mediante la ecuación (36):

$$IIV_{op} = \left(1 - Quad_{op}\right) + DI_{op}, \forall op = 1,2,\dots,n_{op} \quad \textbf{(36)}$$

Finalmente, el número de individuos que cada operador procesará en la siguiente iteración se calcula según la ecuación (37)

$$PS_{op} = \max\left(0.1, min\left(0.9\frac{IIV_{op}}{\sum_{op=1}^{n_{op}} IIV_{op}}\right)\right) \times NP, \forall op = 1,2,\dots,n_{op} \quad \textbf{(37)}$$

Para evitar que una subpoblación de un determinado operador tenga cero soluciones, se utiliza un valor mínimo que es igual a $0{,}1 \times NP$.

5.3.- Modelo adaptativo propuesto

A diferencia del EnMODE original, la primera versión el modelo adaptativo no realizará la reducción lineal de población. Y por último el modelo adaptativo está diseñado para un número de operadores igual a 3 y no a 2 como el algoritmo original, ya que los operadores serán los algoritmos SMA, HHO y HGS descritos previamente en el capítulo 3.

5.3.1 – Modelo adaptativo sin reducción poblacional

El modelo adaptativo diseñado puede ser considerado una mejora del algoritmo híbrido 3. Es decir, no se realizará una división de la matriz, si no que se recorrerá la matriz individuo a individuo asignando de forma aleatoria cada individuo a un operador. A diferencia del algoritmo híbrido 3, se limitará el numero de individuos que procese cada operador según el tamaño que indique la ecuación (37). En una primera instancia estos tamaños habrán sido inicializados a $PS_{op} = \frac{NP}{3}$, y en caso de que NP no sea divisible entre 3, se reparten los individuos restantes de manera aleatoria entre los tres operadores.

Al inicio del proceso iterativo se calcularán todas las variables necesarias para cada operador. Posteriormente se recorrerá la matriz individuo a individuo y para cada uno de ellos se generará un número aleatorio $decison$ que se calcula mediante la ecuación (31) que decide qué operador aplicar, siempre que el número de individuos procesados por ese operador no haya alcanzado el límite prestablecido NP_{op}. Siendo el 1 el algoritmo SMA, el 2 el algoritmo HHO y el 3 el algoritmo HGS.

Una vez la población ha sido procesada, se aplicarán las ecuaciones de desde la (33) hasta la (37). Y se calcularán los nuevos tamaños de las subpoblaciones.

5.3.1.1.- Pseudocódigo

Inicializar $UB, LB, n_{vars}, N, F(i), matrix, dest_{fitness}, fes,$ max $\_fes$
Calcular $F(i)$ para todos los individuos
While (fes<max_fes)
    Calcular T // T=max_fes/N
    Actualizar *dest_fitness, best_pos, order, index*
    Ordenar los vectores *order, index* de manera ascendente
    Actualizar *bigger, lowest y worstAllTime*
    Calcular *s y W*
    Inicializar W_1 y W_2 a 1
    Inizializar *hungry* y sumHungry a 0
    For i desde 0 hasta N-1

If (*dest_fitness == F(i))*
hungry(i) =0;
Actualizar *tempPosition*
Else
Calcular *c*
If (*c* < 100)
Calcular *b*
Else
b = c
End if
hungry(i)+=b
sumHungry += hungry(i)
End if
End for
Calcular W_1 *y* W_2
For i desde 0 hasta N-1
While procesado == 0
Calcular *decision* según la ecuación (31)
if *decision* ==1 y *count_sma* ≤ *ps_sma*
count_sma++
procesado=1
SMA(i)
If else *decision* ==2 y *count_hho* ≤ *ps_hho*
count_hho++
procesado=1
HHO(i)
If esle *decision* ==3 y *count_hgs* ≤ *ps_hgs*
count_hgs++
procesado=1
HGS(i)
Else
Procesado=0
End if
End while
End for
Inicializar los contadores *count_sma, count_hho, count_hgs* a 0
Comprobar que las variables estén dentro de *UP* y *LB*
Calcular $F(i)$
Calcular $Qual_{op}, Div_{op}, DI_{op}, IIV_{op}$ según las ecuaciones (33), (34), (35) y (36)
Calcular los nuevos valores de *ps_sma, ps_hho* y *ps_hgs* según la ecuación (37)
End while
Return *dest_fitness*

Figura 14: Pseudocódigo del Algoritmo Adaptativo

5.3.2 – Modelo adaptativo con reducción poblacional

Esta versión implementa la reducción de población mencionada anteriormente en el apartado 5.2. El tamaño de la población es uno de los factores clave que afectan en el rendimiento de los algoritmos de optimización, siendo más efectiva la población pequeña en la fase de explotación.

La inicialización de población a un número elevado de individuos favorece el espacio de búsqueda inicial y garantiza una exploración adecuada. No obstante, a medida que el algoritmo avanza podría ser beneficioso reducir el espacio de búsqueda, descartando a los peores individuos mediante la reducción lineal de la población y centrándose en las soluciones más prometedoras.

El permitir ajustar dinámicamente el tamaño de la población puede mejorar la eficiencia computacional sin comprometer la calidad de los resultados y al realizarse de una manera lineal evita que el algoritmo pierda diversidad prematuramente. La implementación de esta técnica en algoritmos combinatorios puede lograr un mayor equilibrio entre las fases de exploración y explotación, acelerando la convergencia.

La reducción lineal propuesta se define en la ecuación (32). Donde NP_{min} será un parámetro predefinido, evitando que la población no decaiga por debajo de ese umbral.

5.3.2.1.- Pseudocódigo

Inicializar $UB, LB, n_{vars}, NP, F(i), matrix, dest_{fitness}, fes,$ max $\_fes$
Calcular $F(i)$ para todos los individuos
Asignar $NP = NP_{ini}$
While (fes<max_fes)
    Calcular T // T=max_fes/N
    Actualizar *dest_fitness, best_pos, order, index*
    Ordenar los vectores *order, index* de manera ascendente
    Actualizar *bigger, lowest y worstAllTime*
    Calcular *s y W*
    Inicializar W_1 y W_2 a 1
    Inizializar *hungry* y sumHungry a 0
    For i desde 0 hasta $NP - 1$
        If (*dest_fitness == F(i))*
            hungry(i) =0;
            Actualizar *tempPosition*
        Else

Calcular *c*
If (*c* < 100)
Calcular *b*
Else
b = *c*
End if
hungry(i)+=b
sumHungry += hungry(i)
End if
End for
Calcular W_1 *y* W_2
For i desde 0 hasta $NP - 1$
While procesado == 0
Calcular *decision* según la ecuación (31)
if *decision* ==1 y *count_sma* ≤ *ps_sma*
count_sma++
procesado=1
SMA(i)
If else *decision* ==2 y *count_hho* ≤ *ps_hho*
count_hho++
procesado=1
HHO(i)
If esle *decision* ==3 y *count_hgs* ≤ *ps_hgs*
count_hgs++
procesado=1
HGS(i)
Else
Procesado=0
End if
End while
End for
Inicializar los contadores *count_sma, count_hho, count_hgs* a 0
Comprobar que las variables estén dentro de *UP* y *LB*
Calcular $F(i)$
Calcular $Qual_{op}, Div_{op}, DI_{op}, IIV_{op}$ según las ecuaciones (34), (35), (36) y (37)
Calcular los nuevos valores de *ps_sma, ps_hho* y *ps_hgs* según la ecuación (37)
Calcular N_{t+1}según la ecuación (32)
While $N > NP_{t+1}$ y $NP_{min} \leq NP_{t+1}$
$aux = F(i)$
For *i* = 0 hasta NP
If $aux > F[i]$
$worst\_pos = 1$
$aux = F(i)$

$aux\_matrix = matrix[i]$

End if

End for

$F[worst\_position] = i$

$F[NP-1] = aux$

$Matrix[worst\_pos] = Matrix[NP-1]$

$Matrix[NP-1] = aux\_matrix$

$NP = NP - 1$

End while

End while

Return *dest_fitness*

Figura 15: Pseudocódigo del Algoritmo Adaptativo con reducción de población

Capítulo 6
Resultados experimentales

En este capítulo se expondrán los resultados obtenidos a partir de los experimentos realizados. Los datos caracterizados se recopilan en tablas para facilitar su análisis. Se describirán también los aspectos más relevantes hallados en cada prueba.

6.1.- Funciones propuestas

En el ámbito de la optimización y los algoritmos evolutivos, las funciones de optimización elegidas son de crucial importancia para realizar los resultados experimentales y evaluar el rendimiento y la convergencia. Ya que cada función presenta ciertas peculiaridades las cuales pueden favorecer la convergencia para algunos algoritmos, o por el contrario no lograr una buena convergencia.

Las funciones elegidas para la experimentación de este proyecto son las funciones Rosenbrock, Foxholes, trid10 y Michalewicz.

6.1.1 Función Rosenbrock

La función Rosenbrock también conocida como "Valle de Rosenbrock" es una de las más populares para probar algoritmos de optimización. Ya que no presenta una naturaleza convexa.

Según (Mia Jian, 2019) la función Rosenbrock es una función clásica en la teoría y el método de optimización sin restricciones. Es una herramienta importante para medir las ventajas y desventajas de los algoritmos.

La función Rosenbrock se define según la ecuación (38) para casos bidimensionales y según la ecuación (39) para casos multidimensionales.

$$F(x,y) = (1-x)^2 + 100(y-x^2)^2 \quad \textbf{(38)}$$

$$F(x_1, x_2, \dots, x_n) = \sum_{i=1}^{n-1}[100(x_{i+1} - x_i^2)^2 + (1-x_i)^2], \; x_i \in [-5{,}10] \quad \textbf{(39)}$$

Cada contorno de la función Rosenbrock es aproximadamente parabólico, cuyo mínimo global también se encuentra en un valle parabólico. Es fácil encontrar el valle, pero bastante difícil encontrar el mínimo global porque el valor en el valle no varía mucho.

La función Rosenbrock presenta características de una función multimodal cuando sus dimensiones son mayores que 3 y características unimodales con otras dimensiones.

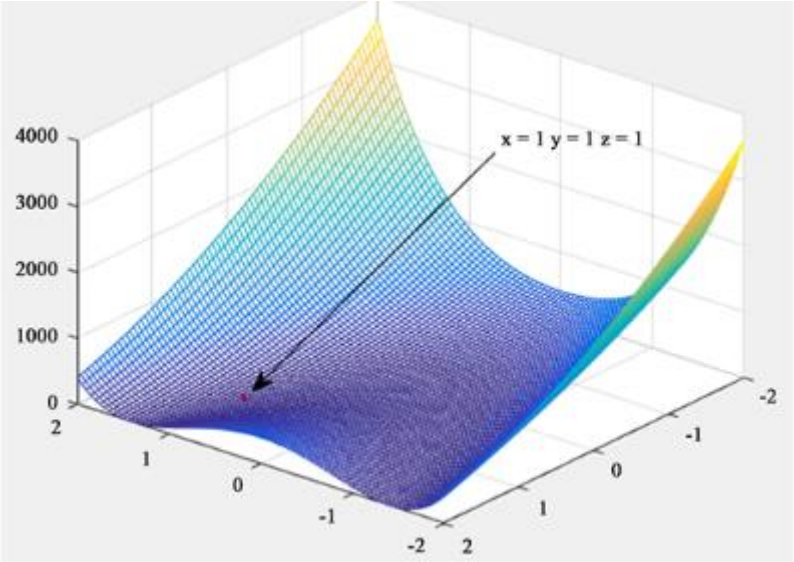

Figura 16: Gráfica 3D de la función Rosenbrock

El mínimo global de esta función se encuentra en $(x_1, x_2, \dots, x_n) = (1,1, \dots,1)$ con un valor óptimo de $f(1,1, \dots,1) = 0$.

6.1.2 Función Foxholes

La función Foxholes conocida como Shekel's foxholes o Shekel's function es una función bidimensional. También es una de las más utilizadas para poner a prueba este tipo de algoritmos evolutivos. Se trata de una función multimodal, por tanto, presenta múltiples óptimos locales. Lo cual supone una mayor dificultad para los algoritmos de encontrar el mínimo global ya que sin una buena estrategia es fácil caer en mínimos locales y encontrar el óptimo local.

La función se define según la Ecuación (41):

$$f(x_1, x_2) = \left[\sum_{i=1}^{25} \frac{1}{i+\sum_{j=1}^{2}(x_j - a_{ij})^6}\right]^{-1}, x_1, x_2 \in [-65.536, 65.536] \quad \textbf{(41)}$$

Dónde a_{ji} es una matriz de tamaño 2x25 que almacena los valores que definen los agujeros *foxholes* del paisaje de la función. En la figura 18 se puede apreciar con mayor claridad los agujeros.

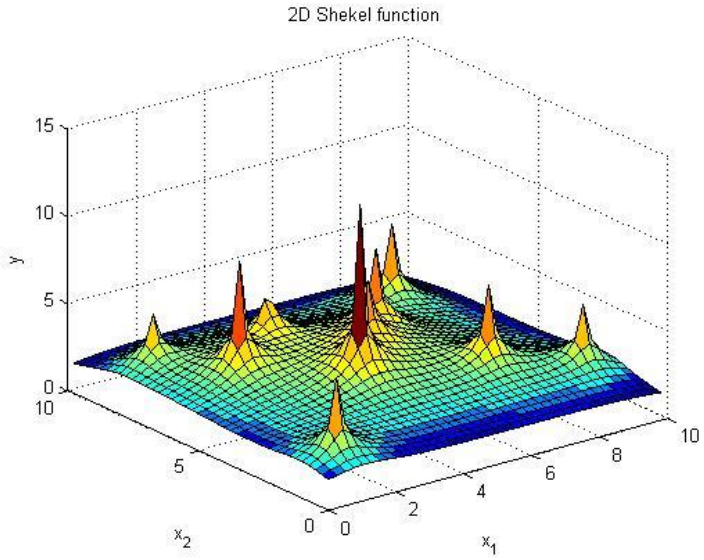

Figura 17: Gráfica 3D de la función Foxholes

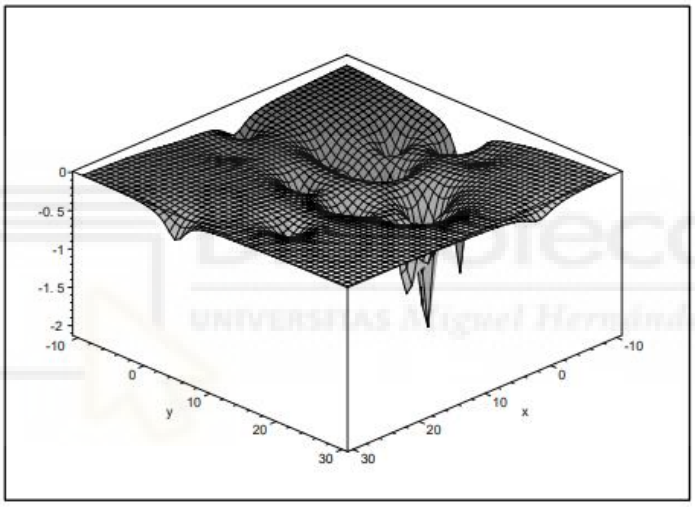

Figura 18: Gráfica 3D invertido de la función Foxholes

6.1.3 Función Trid10

La función Trid10 es una variante de 10 dimensiones de la función Trid. Esta es utilizada para poner a prueba algoritmos de optimización debido a su complejidad por la presencia de múltiples mínimos locales, por lo que se trata de una función multimodal.

La función Trid se define según la Ecuación (42):

$$f(x) = \sum_{i=1}^{n}(x_1 - 1)^2 - \sum_{i=2}^{n} x_i x_{i+1},\ x_i \in [-100,100] \quad \textbf{(42)}$$

En el caso de la versión Trid10 $n = 10$ y su valor óptimo global es $f(x) = -210$.

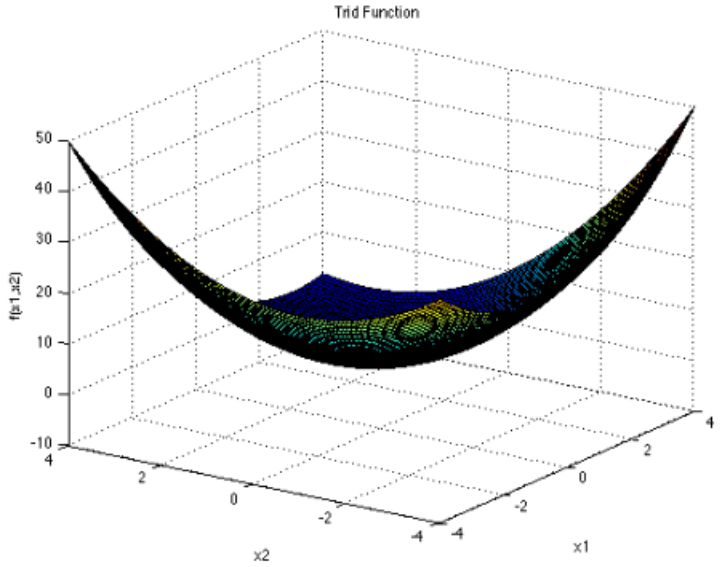

Figura 19: Gráfica 3D de la función Trid

6.1.4 Función Michalewicz

La función Michalewicz se trata de una función multimodal, es decir, presenta multiples mínimos locales añadiendo complejidad al algoritmo evolutivo para escapar de ellos y encontrar el mínimo global. La topología de la función presenta múltiples valles, alguno de ellos muy profundos como se puede observar en la figura 20.

La función Michalewicz se define según la Ecuación (43):

$$f(x_1, x_2, \ldots, x_n) = -\sum_{i=1}^{n} \sin(x_i) \left[\sin\left(\frac{ix_i^2}{\pi}\right)\right]^{2n}, \; x_i \in [0, \pi] \quad \textbf{(43)}$$

El valor óptimo global depende del número de dimensiones, para este proyecto su dimensión $d = 5$, y su mínimo global $f(x) = -4.687658$.

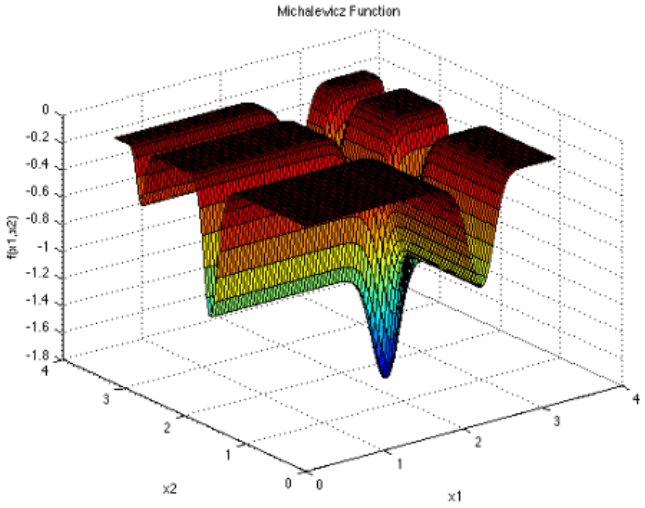

Figura 20: Gráfica 3D de la función Michalewicz

6.2.- Resultados de cada versión

Para poder evaluar el rendimiento y el comportamiento de cada algoritmo combinatorio, es necesario evaluar el comportamiento de cada función con cada operador. De esta manera sabiendo como converge cada función se podrá comprobar si el método adaptativo selecciona los pesos de las subpoblaciones de manera adecuada.

También se comparará la convergencia de los modelos híbridos frente a los modelos adaptativos.

Para la realización de estos experimentos cabe aclarar que todos han sido realizados con los valores $MAX_{FES} = 15000$ y $n_{ex} = 15$. Siendo el primero el número máximo de evaluaciones de función y el segundo el numero de ejecuciones que se ha realizado cada experimento. Las dimensiones o atributos de cada función están recogidas en la columna de variables y la columna de población está indicado el tamaño inicial de la población.

6.2.1 Resultados del operador SMA

Población	Función	Variables (v)	Mejor	Peor	Media	Moda	D. Típica
18	Rosenbrock	50	4.70E+01	4.84E+01	4.78E+01	4.70E+01	5.09E-01
		100	9.78E+01	9.84E+01	9.82E+01	9.78E+01	2.09E-01
	Foxholes	2	9.98E-01	9.98E-01	9.98E-01	9.98E-01	3.44E-13
	Trid10	10	-2.10E+02	-2.09E+02	-2.10E+02	-2.10E+02	1.13E-01
	Michalewicz	5	-4.69E+00	-2.95E+00	-4.12E+00	-4.69E+00	5.41E-01
30	Rosenbrock	50	4.75E+01	4.84E+01	4.80E+01	4.75E+01	2.47E-01
		100	9.80E+01	9.86E+01	9.83E+01	9.80E+01	1.69E-01
	Foxholes	2	9.98E-01	9.98E-01	9.98E-01	9.98E-01	1.44E-12
	Trid10	10	-2.10E+02	-2.09E+02	-2.09E+02	-2.10E+02	2.62E-01
	Michalewicz	5	-4.69E+00	-3.54E+00	-4.16E+00	-4.69E+00	4.40E-01

Tabla 1: Resultados del operador SMA

Observando los datos de la tabla 1, se puede observar que todas las funciones han llegado a su mínimo global exceptuando la función Rosenbrock. Algo que también es apreciable es que la función Rosenbrock en este algoritmo es sensible al número de variables y no se ve afectada por el aumento de población. Observando los valores óptimos obtenidos tanto para v=50 como para v=100 coinciden en ambas poblaciones, con una desviación típica cercana a 0, lo cual indica que, durante las 15 ejecuciones, los valores óptimos encontrados no distaban considerablemente los unos de los otros. Por lo que se puede deducir que ambos valores son mínimos locales y el algoritmo SMA no ha conseguido escapar de ellos.

Por otro lado, el resto de las funciones converge adecuadamente y llegan a valores iguales o muy cercanos a su mínimo global con una desviación típica extremadamente baja en el caso de la función Foxholes en ambas poblaciones.
La función Trid 10 muestra una pequeña mejora de su desviación típica para NP=30.

En el caso de la función Michalewicz consigue llegar a su mínimo global pero su desviación típica es ligeramente superior y apenas presenta variaciones de un tamaño de población a otro. Por tanto, no todas las ejecuciones han llegado al óptimo, aunque no se han quedado alejadas.

Hay que remarcar que la función Rosenbrock no converge de manera deseable con el operador SMA.

6.2.2 Resultados del operador HHO

Población	Función	Variables (v)	Mejor	Peor	Media	Moda	D. Típica
18	Rosenbrock	50	1.37E-04	1.80E-02	6.85E-03	1.37E-04	5.78E-03
		100	1.17E-04	1.38E-01	2.99E-02	1.17E-04	4.26E-02
	Foxholes	2	9.98E-01	1.99E+00	1.06E+00	9.98E-01	2.57E-01
	Trid10	10	-2.10E+02	-2.08E+02	-2.09E+02	-2.10E+02	3.64E-01
	Michalewicz	5	-4.64E+00	-3.32E+00	-3.89E+00	-4.64E+00	4.29E-01
30	Rosenbrock	50	2.68E-03	6.20E-02	1.73E-02	2.68E-03	1.74E-02
		100	9.14E-05	1.03E-01	1.81E-02	9.14E-05	2.77E-02
	Foxholes	2	9.98E-01	1.99E+00	1.26E+00	9.98E-01	4.55E-01
	Trid10	10	-2.10E+02	-2.09E+02	-2.09E+02	-2.10E+02	2.52E-01
	Michalewicz	5	-4.51E+00	-2.88E+00	-3.70E+00	-4.51E+00	5.11E-01

Tabla 2: Resultados del operador HHO

Observando la tabla 2 se puede apreciar una mejora considerable de la convergencia de la función Rosenbrock frente a la tabla 1. La función alcanza valores cercanos su mínimo global y no cae en mínimos locales, además su desviación típica es considerablemente baja, especialmente para v=50 y NP=18. Otra diferencia apreciable es que no muestra una gran sensibilidad al incremento de variables, aunque muestra mejores resultados con v=100.

Mientras tanto la función Foxholes ha aumentado considerablemente su desviación típica con respecto del operador SMA, pese a que sigue siendo baja y consigue llegar a su mínimo global.

La función Trid10 no presenta grandes variaciones de un tamaño de población a otro, pero consigue llegar a su valor optimo como se puede observar en el mejor valor obtenido.

Michalewicz es la función con mayor desviación típica, no consigue llegar a su óptimo cuando el tamaño de población es 30 y se queda muy cercano cuando el tamaño de población es 18.

Se destaca que el operador HHO muestra mejor convergencia con la función Rosenbrock y una peor convergencia con el algoritmo Foxholes.

6.2.3 Resultados del operador HGS

Población	Función	Variables (v)	Mejor	Peor	Media	Moda	D. Típica
18	Rosenbrock	50	4.63E+01	4.86E+01	4.80E+01	4.63E+01	8.15E-01
		100	9.79E+01	9.86E+01	9.83E+01	9.79E+01	3.04E-01
	Foxholes	2	9.98E-01	1.08E+01	7.13E+00	1.08E+01	4.65E+00
	Trid10	10	-1.29E+02	1.17E+03	2.38E+01	-1.29E+02	3.21E+02
	Michalewicz	5	-3.91E+00	-2.57E+00	-3.31E+00	-3.91E+00	4.12E-01
30	Rosenbrock	50	4.66E+01	4.86E+01	4.76E+01	4.66E+01	6.28E-01
		100	9.68E+01	9.85E+01	9.80E+01	9.68E+01	5.66E-01
	Foxholes	2	9.98E-01	1.08E+01	4.00E+00	9.98E-01	4.31E+00
	Trid10	10	-1.35E+02	-8.85E+00	-7.28E+01	-1.35E+02	4.14E+01
	Michalewicz	5	-3.91E+00	-2.56E+00	-3.33E+00	-3.91E+00	3.65E-01

Tabla 3: Resultados del operador HGS

Observando la tabla 3, la función Rosenbrock no consigue llegar a su valor optimo, cayendo en mínimos locales, de igual manera que en el algoritmo SMA, además también aumenta la desviación típica. Es decir, no solo no es capaz de escapar de mínimos locales, además no en todas las ejecuciones llega a ellos. Respecto a los resultados de los anteriores operadores también muestra gran sensibilidad al aumento de variables.

Otro aspecto relevante es el aumento considerable de la desviación típica para la función Foxholes, pese a que consigue llegar a su mínimo global, la desviación típica indica que algunos de los experimentos no llegan a la solución óptima, con una media tan distante del valor optimo y una desviación típica tan elevada, se podría deducir que en alguna de las ejecuciones no ha conseguido escapar de uno de los mínimos locales y ha caído en uno de los *foxholes*.

La función Trid10 tampoco consigue llegar a su mínimo global, muestra una media muy alejada del mejor valor obtenido, y una desviación típica extremadamente elevada. Trid10 es una función multimodal, por lo que presenta múltiples mínimos locales, en el caso de la función Foxholes presentaba una desviación típica elevada y una media distante del mejor valor, lo que podría indicar que en ocasiones cayera en mínimos locales. En el caso de la función Trid10 al ser tan elevada la desviación típica (diez veces mayor que la de Foxholes) indica que existe una gran variación entre los óptimos de cada ejecución, pudiendo indicar que cada experimento ha caído en mínimos locales muy distantes entre ellos o simplemente presente una mala convergencia.

Por último, la función Michalewicz tampoco consigue converger en su valor óptimo y presenta una desviación típica similar a los operadores anteriores. Aunque no lo haya conseguido ha alcanzado valores cercanos en todas las ejecuciones, por lo que la probabilidad de que haya quedado atrapado en un mínimo local es baja.

En resumen, el operador HGS es sin duda el que peor converge con todas las funciones respecto del resto de operadores. Esto no tiene por qué ser algo negativo ya que localizando cuál es el peor operador se podrá comprobar si el algoritmo adaptativo es capaz de detectarlo y darle un peso menor a su subpoblación.

6.2.4 Resultados del Algoritmo híbrido 1

Población	Función	Variables (v)	Mejor	Peor	Media	Mediana	D. Típica
18	Rosenbrock	50	1.45E+00	1.12E+01	4.54E+00	1.45E+00	2.52E+00
		100	1.78E+00	9.81E+01	3.38E+01	1.78E+00	4.02E+01
	Foxholes	2	9.98E-01	9.98E-01	9.98E-01	9.98E-01	3.06E-13
	Trid10	10	-2.10E+02	-2.10E+02	-2.10E+02	-2.10E+02	2.39E-02
	Michalewicz	5	-4.69E+00	-3.49E+00	-4.07E+00	-4.69E+00	4.74E-01
30	Rosenbrock	50	1.89E+00	4.63E+01	2.61E+01	1.89E+00	2.21E+01
		100	5.78E+00	9.80E+01	6.27E+01	5.78E+00	4.45E+01
	Foxholes	2	9.98E-01	9.98E-01	9.98E-01	9.98E-01	3.86E-13
	Trid10	10	-2.10E+02	-2.10E+02	-2.10E+02	-2.10E+02	2.21E-02
	Michalewicz	5	-4.69E+00	-3.52E+00	-4.17E+00	-4.69E+00	4.83E-01

Tabla 4: Resultados del Algoritmo híbrido 1

Observando la tabla 4 lo primero que se puede apreciar es el elevado valor de las desviaciones típicas. Teniendo en cuenta que se están combinando los tres operadores y que cada uno de ellos converge de manera distinta con cada función, no es algo extraño que las desviaciones típicas puedan aumentar.

La función Rosenbrock no consigue llegar a su valor optimo, y no se queda muy cerca para v=50. En cambio, para v=100 la desviación tipica aumenta extremadamente y además, para N=30 aún se aleja más del valor óptimo. Es poco probable que esta función consiga llegar a su optimo con ninguno de los métodos ya que los operadores por separado no lo han conseguido.

La función Foxholes muestra muy buena convergencia, llegando a su optimo global con una desviación típica muy cercana al 0 en ambas poblaciones.

La función Trid10 y Michalewicz también presentan buena convergencia, llegan a su valor optimo con una desviación típica baja.

En resumen, el algoritmo híbrido 1 muestra mayores problemas de convergencia con la función Rosenbrock. Es decir, no ha mejorado este efecto.

6.2.5 Resultados del Algoritmo híbrido 2.1

Población	Función	Variables (v)	Mejor	Peor	Media	Moda	D. Típica
18	Rosenbrock	50	1.74E-01	9.53E+00	3.99E+00	1.74E-01	2.44E+00
		100	2.00E-01	4.41E+01	1.64E+01	2.00E-01	1.20E+01
	Foxholes	2	9.98E-01	9.98E-01	9.98E-01	9.98E-01	4.78E-12
	Trid10	10	-2.10E+02	-2.10E+02	-2.10E+02	-2.10E+02	1.19E-01
	Michalewicz	5	-4.64E+00	-3.33E+00	-3.98E+00	-4.64E+00	4.26E-01
30	Rosenbrock	50	5.72E-01	1.11E+01	4.46E+00	5.72E-01	3.08E+00
		100	2.47E+00	3.76E+01	1.51E+01	2.47E+00	9.25E+00
	Foxholes	2	9.98E-01	9.98E-01	9.98E-01	9.98E-01	1.86E-12
	Trid10	10	-2.10E+02	-2.10E+02	-2.10E+02	-2.10E+02	1.34E-01
	Michalewicz	5	-4.68E+00	-3.53E+00	-4.05E+00	-4.68E+00	4.63E-01

Tabla 5: Resultados del Algoritmo híbrido 2.1

Según los datos de la tabla 4 las desviaciones típicas se mantienen similares al algoritmo 1.

La función Rosenbrock no consigue llegar a su valor optimo global, pero a diferencia del algoritmo anterior en este caso mejoran para v=50. Y de igual manera el peor resultado lo consigue v=100 y NP=30. También aumenta la desviación típica para v=50 y NP=18, siguiendo el mismo patrón que el algoritmo híbrido 1 pero con una ligera mejora de resultados.

La función Foxholes sigue mostrando muy buena convergencia, llegando a su optimo global con una desviación típica muy cercana al 0 en ambas poblaciones. Aunque ligeramente superior que el algoritmo anterior.

La función Trid10 también presenta buena convergencia y llega a su valor óptimo con una desviación típica baja.

En cambio, la Michalewicz no consigue llegar a su valor óptimo, pero se queda considerablemente cerca y con una desviación típica cercana al 0.

En resumen, este método supone una ligera mejora para la función Rosenbrock pero por otro lado perjudica el rendimiento de Michalewicz.

6.2.6 Resultados del Algoritmo híbrido 2.2

Población	Función	Variables (v)	Mejor	Peor	Media	Moda	D. Típica
18	Rosenbrock	50	1.90E+00	4.85E+01	3.56E+01	1.90E+00	2.08E+01
		100	1.00E+00	9.80E+01	8.51E+01	1.00E+00	3.37E+01
	Foxholes	2	9.98E-01	2.98E+00	1.13E+00	9.98E-01	5.12E-01
	Trid10	10	-2.10E+02	-2.10E+02	-2.10E+02	-2.10E+02	2.64E-02
	Michalewicz	5	-4.65E+00	-2.74E+00	-4.12E+00	-4.65E+00	6.05E-01
30	Rosenbrock	50	5.36E-01	4.85E+01	4.50E+01	5.36E-01	1.23E+01
		100	9.78E+01	9.80E+01	9.79E+01	9.78E+01	1.10E-01
	Foxholes	2	9.98E-01	9.98E-01	9.98E-01	9.98E-01	4.89E-12
	Trid10	10	-2.10E+02	-2.10E+02	-2.10E+02	-2.10E+02	1.66E-02
	Michalewicz	5	-4.69E+00	-3.54E+00	-4.24E+00	-4.69E+00	4.54E-01

Tabla 6: Resultados del Algoritmo híbrido 2.2

Como se puede apreciar en la tabla 6, el algoritmo Rosenbrock para v=100 y NP=30 ha caído en un mínimo local. También presenta desviaciones típicas extremadamente elevadas exceptuando en el caso mencionado ya que todas las ejecuciones han caído en ese mínimo. Para v=50 y NP=30 si que obtiene un valor cercano a su optimo, pero si se observa el peor resultado, en todos los casos de esta función aparecen los mínimos locales. Por lo tanto, alguno de los experimentos ha caído en ese mínimo local.

La función Foxholes, pese a que sigue convergiendo correctamente sí que empeora el valor de su desviación típica frente a los valores obtenidos con anterioridad.

La función Trid10 y Michalewicz también presentan buena convergencia, llegan a su valor optimo con una desviación típica baja.

En resumen, la función Rosenbrock cae en óptimos locales pese a que algunas ejecuciones sí que consiguen escapar de esos valores.

6.2.7 Resultados del Algoritmo híbrido 3

Población	Función	Variables (v)	Mejor	Peor	Media	Moda	D. Típica
18	Rosenbrock	50	4.16E-01	4.65E+01	1.06E+01	4.16E-01	1.83E+01
		100	1.60E+00	9.77E+01	5.19E+01	1.60E+00	4.41E+01
	Foxholes	2	9.98E-01	2.98E+00	1.20E+00	9.98E-01	5.56E-01
	Trid10	10	-2.10E+02	-2.10E+02	-2.10E+02	-2.10E+02	3.22E-02
	Michalewicz	5	-4.65E+00	-2.95E+00	-4.29E+00	-4.65E+00	4.70E-01
30	Rosenbrock	50	2.65E-01	4.67E+01	2.30E+01	2.65E-01	2.24E+01

	100	3.14E+00	9.78E+01	5.72E+01	3.14E+00	4.46E+01
Foxholes	2	9.98E-01	9.98E-01	9.98E-01	9.98E-01	2.36E-12
Trid10	10	-2.10E+02	-2.10E+02	-2.10E+02	-2.10E+02	3.00E-02
Michalewicz	5	-4.68E+00	-3.10E+00	-4.00E+00	-4.68E+00	4.80E-01

Tabla 7: Resultados del Algoritmo híbrido 3

Al observar la tabla 7 se puede apreciar que la función Rosenbrock no consigue alcanzar su valor óptimo y viendo los peores resultados se puede observar que han vuelto a aparecer los mínimos locales. No obstante, algunas soluciones consiguieron escapar y llegar a un valor más cercano al valor óptimo. También muestra sensibilidad al aumento de variables ya que para v=100 muestra peores resultados.
La función Foxholes consigue resultados levemente mejores cuando NP=30. Consigue llegar a su óptimo para ambas poblaciones.

La función Trid10 también presenta buena convergencia y llega a su valor óptimo con una desviación típica se encuentra en valores cercanos a 0.

En cambio, la Michalewicz no consigue llegar a su valor óptimo, pero se queda considerablemente cerca y con una desviación típica cercana al 0.

6.2.8 Resultados del Modelo adaptativo

Los valores expresados en las columnas Ps SMA, Ps HHO, Ps HGS son el sumatorio de los pesos de cada subpoblación final obtenidos en cada uno de los 15 experimentos.

Población	Función	Variables (v)	Mejor	Peor	Media	Moda	D. Típica	Ps SMA	Ps HHO	Ps HGS
18	Rosenbrock	50	8.52E-02	4.63E+01	7.06E+00	8.52E-02	1.58E+01	159	86	25
		100	3.39E-01	9.80E+01	3.00E+01	3.39E-01	4.24E+01	151	84	35
	Foxholes	2	9.98E-01	9.98E-01	9.98E-01	9.98E-01	3.96E-13	164	91	15
	Trid10	10	-2.10E+02	-2.10E+02	-2.10E+02	-2.10E+02	1.41E-02	165	90	15
	Michalewicz	5	-4.69E+00	-3.50E+00	-4.03E+00	-4.69E+00	4.85E-01	138	85	47
30	Rosenbrock	50	1.79E-01	4.73E+01	2.26E+01	1.79E-01	2.31E+01	219	128	103
		100	1.32E+00	9.80E+01	4.96E+01	1.32E+00	4.67E+01	223	124	103
	Foxholes	2	9.98E-01	9.98E-01	9.98E-01	9.98E-01	2.12E-13	273	157	20
	Trid10	10	-2.10E+02	-2.10E+02	-2.10E+02	-2.10E+02	1.51E-02	245	150	55
	Michalewicz	5	-4.69E+00	-2.58E+00	-4.05E+00	-4.65E+00	6.20E-01	220	138	92

Tabla 8: Resultados del Modelo adaptativo

Los datos presentados en la tabla 8 muestran que la mayoría de las funciones han conseguido llegar a su optimo. Exceptuando de la función Rosenbrock. A pesar de ello ha conseguido acercarse a él. Aun así, observando el peor resultado conseguido se puede apreciar que sigue cayendo en algunas ejecuciones en óptimos locales. Lo cual explica la elevada desviación típica. En comparación con el operador HHO que es el que mejores resultados ha obtenido, este modelo se acerca bastante a su comportamiento exceptuando la función mencionada.

El resto de las funciones convergen correctamente llegando al óptimo global, con desviaciones típicas cercanas al 0. Especialmente en la función Foxholes.

Lo interesante de esta tabla, es que recoge la suma de subpoblaciones de cada ejecución. Por tanto, aquel que tenga un número mayor de ps tendrá mejor rendimiento. En este caso el modelo adaptativo ha considerado al SMA el mejor operador asignándole subpoblaciones de mayor tamaño.

Como se pudo predecir el operador HGS es el que peores resultados ha obtenido reduciendo su subpoblación al mínimo. Por otro lado, el SMA y el HGS han presentado resultados similares, exceptuando para la función Rosenbrock donde el SMA caía en mínimos locales y no convergía de la manera deseada. A pesar de ello ha sido el algoritmo con mayor peso para esta función y por eso se puede deducir basándose en los peores resultados obtenidos que ha caído en mínimos locales en algunas ejecuciones.

6.2.9 Resultados del Modelo adaptativo con reducción de población

Para la realización de estos experimentos se ha establecido el número mínimo de población en $NP_{min} = 6$.

Los valores expresados en las columnas Ps SMA, Ps HHO, Ps HGS son el sumatorio de los pesos de cada subpoblación final obtenidos en cada uno de los 15 experimentos.

Población	Función	Variables (v)	Mejor	Peor	Media	Moda	D. Típica	Ps SMA	Ps HHO	Ps HGS
18	Rosenbrock	50	9.14E-01	1.17E+01	4.28E+00	9.14E-01	3.42E+00	17	58	15
		100	2.26E-01	3.84E+01	1.35E+01	2.26E-01	1.30E+01	15	60	15
	Fox Holes	2	9.98E-01	9.98E-01	9.98E-01	9.98E-01	3.89E-11	31	41	18
	Trid10	10	2.10E+02	-2.09E+02	-2.10E+02	-2.10E+02	1.41E-01	20	53	17
	Michalewicz	5	-4.69E+00	-2.84E+00	-3.90E+00	-4.67E+00	5.84E-01	32	40	18
30	Rosenbrock	50	1.82E-01	2.13E+01	7.43E+00	1.82E-01	7.11E+00	16	57	17
		100	2.31E-01	5.30E+01	2.13E+01	2.31E-01	1.66E+01	16	57	17
	Foxholes	2	9.98E-01	9.98E-01	9.98E-01	9.98E-01	1.08E-10	31	40	19
	Trid10	10	-2.10E+02	-2.09E+02	-2.10E+02	-2.10E+02	1.85E-01	21	53	16
	Michalewicz	5	-4.69E+00	-2.69E+00	-3.88E+00	-4.52E+00	5.42E-01	37	37	16

Tabla 9: Resultados del Modelo adaptativo con reducción poblacional

Al observar el rendimiento de la primera función, se puede ver una mejora considerable, en primer lugar, porque no ha caído en óptimos locales y en segundo lugar porque su desviación típica es considerablemente inferior a la de la primera versión del modelo adaptativo. Aunque la desviación típica es relativamente elevada.

El resto de las funciones convergen correctamente, llegando a sus valores óptimos y con desviaciones típicas cercanas al 0. Especialmente la función Foxholes.

Observando el valor de los pesos de cada subpoblación, se concluye con que el operador que presenta una mayor convergencia es el HHO, seguido del SMA y por ultimo el HGS. Estos resultados presentan una mayor congruencia con los resultados obtenidos en cada uno de los operadores. Se pudo observar que el HGS ha sido el operador que peor converge notablemente con todas las funciones, fijándose en la asignación de pesos ya que esta implementa una estrategia para que el valor mínimo de la subpoblación no llegue a 0 y sea un valor mínimo en función al tamaño total de la población. Al tratarse de una población pequeña en este caso es 1. En el caso de la función Rosenbrock con pesos de 15 para el algoritmo HGS, se puede afirmar que el HGS ha recibido la asignación mínima de pesos durante todas las ejecuciones. Teniendo en cuenta que la función Rosenbrock solo convergía con el operador HHO y que con el resto de los operadores se quedaba atrapada en mínimos locales, era esperable que el operador que más peso obtuviera fuera el HHO dejando al resto considerablemente más atrás.

En el caso de la función Foxholes como se pudo observar en los resultados de los operadores mostraba una mayor convergencia con el algoritmo SMA, aunque con el algoritmo HHO también convergía correctamente. En este modelo adaptativo se ha dado un mayor peso al algoritmo HHO, aunque el algoritmo SMA también presenta un peso elevado quedando ligeramente detrás del HHO. En cambio, el algoritmo HGS vuelve a lograr unos pesos cercanos al mínimo.

La función Trid10 como se pudo apreciar en los resultados de los operadores, es una función que convergía de manera correcta y prácticamente similar tanto para el SMA como para el HHO, en cambio con el HGS no era capaz de llegar su valor óptimo. Y así se ve reflejado en los resultados de la tabla 10. Consiguiendo pesos muy cercanos al mínimo para el operador HGS y asignando una mayor cantidad de individuos al algoritmo HHO, quedando bastante alejado del SMA que quedaría en segundo puesto.

La función Michalewizc no es una función que haya tenido resultados significativamente diferentes entre el operador SMA y el operador HHO. En cambio, con el operador HGS no llegaba a converger en su valor óptimo. De igual manera que con las funciones anteriores esto se ve reflejado en los pesos consiguiendo el HGS un peso cercano al mínimo para esta función. En cambio, para los operadores SMA y HHO presentan valores elevados y cercanos entre ellos, consiguiendo el HGS un mayor peso para la población $NP = 18$ pero los mismos pesos para la población $NP = 30$.

Los resultados del modelo con reducción poblacional presentan una mayor congruencia frente a la primera versión del modelo adaptativo. Comparando la tabla 10 con el resto de las tablas es la que mejores resultados ha obtenido para todas las funciones. Si bien es cierto que el operador HHO presenta mejoras en alguna de las funciones frente a l modelo adaptativo con reducción de población, este también presenta mejoras en algunas funciones frente al HHO mejorando la convergencia como la función Foxholes.

Estos modelos adaptivos muestran que son capaces de dinámicamente elegir el mejor operador, además, se observa que es preferible trabajar con poblaciones que van disminuyendo para en la fase de explotación disponer de poblaciones no muy grandes, ya que así se mejora la eficiencia de la adaptabilidad.

Capítulo 7 Conclusiones y futuras propuestas

Es este apartado se describirán las conclusiones del proyecto y se propondrá posibles mejoras o adaptaciones para proyectos futuros.

7.1.- Conclusiones

Este proyecto ha consistido en el diseño de varios algoritmos combinatorios tanto estáticos como dinámicos, implementando tres algoritmos metaheurísticos de optimización: SMA (*Slime Mould Algorithm*), HHO (*Harris Hawk Optimization)* y HGS (*Hunger Games Search*). Los métodos dinámicos que se han desarrollado se adaptan al comportamiento de cada operador para cada función, intensificando aquellos operadores con mejores prestaciones.

Para evaluar su rendimiento se ha caracterizado tomando datos de cada uno de estos algoritmos de manera aislada y sus versiones combinatorias resolviendo 4 funciones de optimización. La función Rosenbrock, la funcion Foxholes, la función Trid10 y la función Michalewicz. Todas ellas funciones multimodales, exceptuando de la función Rosenbrock, la cual es una función unimodal. Sin embargo, según (Mia Jian, 2019) esta función puede presentar características de una función multimodal cuando sus dimensiones son superiores a 3 (en este caso 50 y 100).

De este análisis se puede concluir que los resultados de los operadores muestran muy poca convergencia con el operador HGS convirtiéndolo en el peor de ellos para estas funciones. Siendo el HHO el más equilibrado, seguido del SMA que muestra buena convergencia con todas las funciones exceptuando la función Rosenbrock y muy buena convergencia con la función Foxholes.

los métodos de hibridación no consiguen resultados equilibrados para todas las funciones como es el caso del modelo adaptativo sin reducción poblacional, en algunos de ellos caen en mínimos locales o presentan desviaciones típicas extremadamente elevadas. Siendo el algoritmo híbrido 3 el que mayor equilibrio presenta y en el que se basa el modelo de reparto de subpoblaciones de forma aleatoria para los modelos adaptativos.

El modelo adaptativo con reducción poblacional presenta resultados mejores y más congruentes frente a los resultados obtenidos para cada operador. La primera versión del modelo adaptativo no consigue pesos acordes a los resultados de cada operador, ya que la función Rosenbrock converge considerablemente mejor con el operador HHO.

Los pesos asignados a las subpoblaciones del modelo adaptativo sin reducción de población indican que el algoritmo SMA presenta una mejora en la fase de exploración, ya que al añadir la reducción de población se incrementa el enfoque en la fase de explotación, resultando en un tamaño final mayor de las subpoblaciones del HHO. En el caso de la función Foxholes muestra una ligera mejora en la primera versión del modelo adaptativo frente al modelo adaptivo con reducción de población, lo cual indica que el algoritmo es más efectivo para esta función durante la fase de exploración.

Se ha demostrado que la reducción de población ha cumplido las expectativas eliminando los individuos con peores resultados y centrándose en aquellos más prometedores, sin que el espacio de búsqueda se reduzca prematuramente, asegurando una buena exploración de este. Se ha evitado la caída en mínimos locales para todas las funciones, llegando todas a su valor optimo con desviaciones típicas cercanas a 0. Exceptuando de la función Rosenbrock que no consigue converger para ninguno de los modelos y operadores.

7.2.- Posibles mejoras

Los algoritmos combinatorios de este proyecto son independientes de los operadores. Por tanto, como línea futura se plantera analizar nuevos operadores incrementando o no el número total de ellos o sustituyéndolos por nuevos operadores

Visto los resultados de los modelos adaptativos donde se aprecia que existe una mejora del operador SMA para la fase de exploración y una mejora del HHO para la fase de explotación, planteamos analizar la fase de exploración y explotación de los operadores y combinarlos por fases para una línea futura de investigación.

Respecto al problema de convergencia que presenta la función Rosenbrock en (Mia Jian, 2019) se propone una adaptación para que esta función tenga una mayor convergencia con los algoritmos evolutivos. En ese artículo se propone una mejora adicional del algoritmo evolutivo, que combina la operación de cruce de múltiples estrategias y el ajuste dinámico de los parámetros del algoritmo. De modo que este tenga en cuenta tanto la búsqueda global como la búsqueda local. Mejorando la velocidad de convergencia y la precisión de la optimización en la función de Rosenbrock.

Bibliografía

Ali Asghar Heidari, S. M. (2019). Harris hawks optimization: Algorithm and applications. *Future Generation Computer Systems*, 851-854.

Karam M. Sallam, S. M. (2020). Multi-Operator Diferential Evolution Algorithm for Solving Real-Word Constrained Optimization Problems. *2020 IEEE Congress on Evolutionary Computation (CEC)* (pp. 1-8). Glasgow: IEEE.

Mia Jian, H. L. (2019, Noviembre). Research on Rosenbrock Function Optimization Problem Based on Improved Differential Evolution Algorithm. *Journal of Computer and Communications, 7*(11), 107-102. doi:10.4236/jcc.2019.711008

Rao, R. V. (2016). Jaya: A simple and new optimization algorithm for solving constrained and unconstrained optimization problems. *International Journal of Industrial Engineering Computations*, 20.

Shimin LI, H. C. (2020). Slime mould algorithm: A new method for stochastic optimization. *Future Generation Computer Systems*, 302-,304.

Son, P. V. (2023). Application of slime mold algorithm to optimize time, cost and quality in construction projects. *International Journal of Construction Management*, 375–1386.

Tero, A. (2010). Rules for Biologically Inspired Adaptive Network Design. *Science AAAS*, 439-443. doi:10.1126/science.1177894

Yutao Yang, H. C. (2021). Hunger games search: Visions, conception, implementation, deep analysis, perspectives, and towards performance shifts. *Expert System with Applications*, 4, 5.