

DESARROLLO DE UNA HERRAMIENTA PARA LA APLICACIÓN DE ALGORITMOS DE ENJAMBRES COMO TÉCNICA DE CUANTIFICACIÓN DE COLOR

Trabajo Fin de Grado

Icono

Descripción generada automáticamente con confianza media



Autor:

Ángel Casado Nogueras

Tutores:

María Luisa Pérez Delgado Jesús Ángel Román Gallego

Junio 2025

Índice

[1. Motivaciones 5](#_Toc199358850)

[2. Introducción 6](#_Toc199358851)

[2.1 Representación de imágenes en color 6](#_Toc199358852)

[2.2 Los modelos de color 7](#_Toc199358853)

[3. El problema de la cuantificación de color 7](#_Toc199358854)

[3.1 Definición del problema 7](#_Toc199358855)

[3.2 Aplicaciones prácticas 9](#_Toc199358856)

[4. Los algoritmos de enjambres. 10](#_Toc199358857)

[4.1 Características generales. 10](#_Toc199358858)

[4.2 Algoritmo de optimización por enjambre de partículas (PSO) 11](#_Toc199358859)

[4.2.1 Introducción 11](#_Toc199358860)

[4.2.3 Variables 11](#_Toc199358861)

[4.2.4 Definición del algoritmo 12](#_Toc199358862)

[4.2.5 Parámetros PSO 14](#_Toc199358863)

[4.3 Luciérnagas (Firefly) 15](#_Toc199358864)

[4.3.1 Introducción 15](#_Toc199358865)

[4.3.3 Variables 15](#_Toc199358866)

[4.3.4 Definición del algoritmo 15](#_Toc199358867)

[4.4 Lobos / Grey Wolf optimization (GWO) 17](#_Toc199358868)

[4.4.1 Introducción 17](#_Toc199358869)

[4.4.2 Fundamentos 17](#_Toc199358870)

[4.4.3 Definición del algoritmo 19](#_Toc199358871)

[4.4.4 Variables 21](#_Toc199358872)

[4.5 El algoritmo de Ballenas (Whale Swarm Algorithm - WSA) 22](#_Toc199358873)

[4.5.1 Introducción 22](#_Toc199358874)

[4.5.2 Fundamentos 22](#_Toc199358875)

[4.5.3 Definición del algoritmo 23](#_Toc199358876)

[4.6 Abejas – Algoritmo Colonia de Abejas Artificiales (ABC) 25](#_Toc199358877)

[4.6.1 Introducción 25](#_Toc199358878)

[4.6.3- Variables 25](#_Toc199358879)

[4.6.4- Fundamentos del algoritmo 26](#_Toc199358880)

[5. Tecnologías empleadas 30](#_Toc199358881)

[5.1 Python 30](#_Toc199358882)

[5.2 Numphy 30](#_Toc199358883)

[5.3 OpenCv 31](#_Toc199358884)

[5.4 Scikit-learn 31](#_Toc199358885)

[5.5 Scikit-image 31](#_Toc199358886)

[5.6 Entornos virtuales 31](#_Toc199358887)

[5.7 Control de versiones 32](#_Toc199358888)

[5.8 Visual Studio Code 32](#_Toc199358889)

[5.9 Otros 32](#_Toc199358890)

[6. Pruebas 33](#_Toc199358891)

[6.1 MSE 33](#_Toc199358892)

[6.2 MAE 34](#_Toc199358893)

[6.3 SSIM 34](#_Toc199358894)

[6.4 UQI 34](#_Toc199358895)

[6.5 Tablas de resultados 34](#_Toc199358896)

[7. Desarrollo del programa 35](#_Toc199358897)

[5.1 Archivo main o ejecutor 35](#_Toc199358898)

[5.2 Algoritmos 39](#_Toc199358899)

[5.2.1 PSO 39](#_Toc199358900)

[5.2.2 Luciérnagas 43](#_Toc199358901)

[5.2.3 Lobos 48](#_Toc199358902)

[5.2.4 Abejas 52](#_Toc199358903)

[5.3 Funciones 61](#_Toc199358904)

[5.3.1 pintaImagen 61](#_Toc199358905)

[5.3.2 preparaImagen 62](#_Toc199358906)

[5.3.3 generaCuantizada 63](#_Toc199358907)

[5.3.4 getMse 64](#_Toc199358908)

[5.3.5 getMae 64](#_Toc199358909)

[5.3.6 getSsim 64](#_Toc199358910)

[5.3.7 getMssim 65](#_Toc199358911)

[8. Trabajo futuro 66](#_Toc199358912)

[9. Conclusiones 67](#_Toc199358913)

[10.Bibliografía 67](#_Toc199358914)

**Índice de figuras**

Ilustración 1 Imagen Inicial 9

Ilustración 2 Imagen resultante con 32 colores 10

Ilustración 3 Pirámide 25

Ilustración 4 Lobos Grises 29

Ilustración 5 Logo de python 45

Ilustración 6 Numphy 46

Ilustración 7 GitHub 48

Ilustración 8 Visual Studio Code 49

Ilustración 9 JSON de argumentos 49

Ilustración 10 Argumentos 51

Ilustración 11 Obtener ruta de la imagen 52

Ilustración 13 Comprobaciones 53

Ilustración 12 Cuerpo del programa 53

Ilustración 14 Mapeo de funciones 54

Ilustración 15 Comienzo PSO 55

Ilustración 16 Constructor PSO 55

Ilustración 17 Inicialización PSO 56

Ilustración 18 Constructor de Intelligence 56

Ilustración 19 Comienzo bucle PSO 57

Ilustración 20 Método de Intelligence para paso a lista 58

Ilustración 21 Actualización mejor solución particular PSO 59

Ilustración 22 Actualización mejor solución global PSO 59

Ilustración 23 Fin PSO 60

Ilustración 24 Constructor Luciérnagas 61

Ilustración 25 Inicio luciérnagas 62

Ilustración 26 Comienzo del bucle luciérnagas 62

Ilustración 27 Movimientos de luciernagas 63

Ilustración 28 Cálculos Luciérnagas 64

Ilustración 29 Actualización posición y fitness luciérnagas 64

Ilustración 30 Actualización mejor solución global luciérnagas 65

Ilustración 31 Función mover luciérnaga 65

Ilustración 32 Inicialización Lobos 66

Ilustración 33 Get mejores lobos 67

Ilustración 34 Inicio bucle Lobos 67

Ilustración 35 Calculo de aleatorios, A y C 68

Ilustración 36 Cálculo de D 68

Ilustración 37 Cálculo de X para cada lobo 69

Ilustración 38 Final bucle lobos 69

Ilustración 39 Constructor abejas 70

Ilustración 40 inicialización abejas 70

Ilustración 41División de individuos 71

Ilustración 42 Inicio del bucle abejas 72

Ilustración 43 Nuevos individuos 72

Ilustración 44 función para nuevos individuos 72

Ilustración 45 función vecinos 73

Ilustración 46 Comprobación abejas 73

Ilustración 47 Fin bucle abejas 73

Ilustración 48 Constructor ballenas 74

Ilustración 49 inicialización ballenas 75

Ilustración 50 Inicio del bucle ballenas 75

Ilustración 51 movimiento de ballenas 76

Ilustración 52 función buscar mejor y más cercana ballena 77

Ilustración 53 calcular distancia entre individuos 77

Ilustración 54 Fin bucle ballenas 78

Ilustración 55 pintaImagen 79

Ilustración 56 preparaImagen 80

Ilustración 57 generaCuantizada 81

Ilustración 58 función MSE 82

Ilustración 59 getMae 82

Ilustración 60 getSsim 82

Ilustración 61 getMssim 83

# 1. Motivaciones

El desarrollo de este trabajo de fin de grado nace del creciente interés en las técnicas de optimización inspiradas por la naturaleza, en particular los algoritmos de enjambre, que han demostrado ser métodos bastante eficaces y versátiles en la solución de problemas complejos en diversas áreas.

También ha sido motivado por la asignatura de Sistemas Inteligentes con la cual personalmente disfruté mucho cursándola. En ella pudimos probar otro lenguaje de programación como Python, comprobar las diferencias que existen entre C y Python, e implementar código con un objetivo algo más visual que en el resto de la carrera.

# 2. Introducción

La cuantificación de color en imágenes es un problema complejo en estos tiempos en los que nos comunicamos frecuentemente con el uso de imágenes.

El objetivo principal es reducir el número de colores de una imagen a un conjunto más pequeño de colores representativos, a los que se llama **paleta cuantizada** de colores, con la menor pérdida de calidad posible. La imagen resultante, que se denomina **imagen cuantizada**, debe ser reconocible y lo más parecida posible visualmente a la original, pero con una representación mucho más simple y eficiente.

Puede parecer un proceso simple a primera vista, pero representa varios desafíos y requiere del uso de técnicas avanzadas, como algoritmos de enjambres, para obtener buenos resultados usando un tiempo de cómputo razonable. Algunas de las razones que explican por qué la cuantificación de color en imágenes es un problema complejo son:

* **Gran volumen de datos:** Ya que las imágenes que se procesan pueden contener miles de pixeles y cada píxel está representado por varios datos, procesar todos estos datos de manera eficiente y efectiva es un gran desafío.
* **Preservación de detalles importantes:** para la cuantificación de color es esencial preservar los detalles importantes de la imagen, como bordes y características distintivas. La simplificación excesiva puede llevar a la pérdida de información crítica y degradar la calidad visual de la imagen.
* **Selección de colores representativos:** Elegir los colores que representarán la imagen cuantizada de manera precisa es una tarea compleja. La selección de estos colores debe hacerse de manera inteligente para garantizar que la imagen cuantizada sea lo más fiel posible a la original.
* **Mantenimiento de la apariencia visual:** La reducción de color debe lograr un equilibrio entre la reducción del número de colores propiamente dicha y la retención de la apariencia visual general de la imagen. Los cambios agresivos en los colores pueden hacer que la imagen resultante sea irreconocible o menos atractiva.
* **Optimización de la eficiencia:** Procesar imágenes en tiempo real o en aplicaciones donde la velocidad es crucial (como la transmisión de video) requiere algoritmos de reducción de color eficientes que puedan realizar cálculos rápidos y precisos.
* **Evaluación de la calidad:** Medir la calidad de la imagen resultante es subjetivo y puede depender de la percepción humana. Evaluar y comparar algoritmos en términos de calidad de reducción de color es un aspecto importante de la investigación en este campo.

## 2.1 Representación de imágenes en color

La representación de imágenes en color en un ordenador es un proceso fundamental que implica codificar y almacenar la información de color de una imagen para luego procesarla, visualizarla y manipularla. Para lograr esto, los ordenadores utilizan modelos de color, siendo el modelo RGB (Red, Green, Blue) uno de los más comunes.

En el modelo RGB, cada píxel de una imagen se representa como una combinación de tres canales de color: rojo, verde y azul. Cada canal puede tomar valores entre 0 y 255 que representan la intensidad de ese color en un píxel específico.

La combinación de estos tres canales de color en cada píxel permite crear una gran cantidad de colores y tonalidades (16.777.216 colores posibles), lo que nos permite representar imágenes en color de alta calidad. La información de color de cada píxel se almacena en una matriz tridimensional, donde las dos primeras dimensiones representan la posición espacial del píxel en la imagen y la tercera dimensión almacena los valores de intensidad de rojo, verde y azul.

Esta representación de color en RGB es la base para todas las operaciones de procesamiento de imágenes en color, incluida la reducción de color. Durante este proceso, el objetivo es reducir el número de colores de la imagen, manteniendo al mismo tiempo la apariencia visual deseada. Los algoritmos de enjambres nos ayudan a “escoger” los colores adecuados para preservar la calidad en la imagen reducida.

## 2.2 Los modelos de color

Ya que se ha hablado de cómo un ordenador representa las imágenes en color utilizando el modelo de color RGB, me parece importante hablar brevemente de otros modelos que existen.

Uno de los más simples es la **escala de grises** que tiene un solo canal, en el que representa valores de grises del blanco al negro. Pueden tener una profundidad de 8 o de 16 bits por píxel.

Otro de ellos es el **indexado** en el que las imágenes tienen una gama de colores reducida para que sus archivos sean más pequeños. Las imágenes indexadas por lo general se suelen usar en páginas web o en gráficos con pocos colores. Solo cuentan con un canal de 8 bits.

Otros dos modelos bastantes extendidos hoy en día son:

1. **CMYK:** Es un modelo de color de cuatro canales. Los valores representan el color cian, el magenta, el amarillo y el negro. Al igual que el RGB puede tener 8 o 16 bits por canal. Este modelo compone los colores de manera sustractiva, es decir, parte del blanco y va restando colores usando sus complementarios.

Este modelo es el que usan habitualmente las impresoras. A la hora de retocar las fotos en el ordenador nos servirá de gran ayuda para ver si los colores de la imagen que hemos trabajado en RGB serán los mismos que los impresos en CMYK ya que a veces no coinciden. Esto ocurre porque hay algunos colores RGB que no pueden conseguirse en CMYK.

1. **LAB:** En este modelo existen 3 canales, L, A y B. L representa la información tonal o luminancia (intensidad de la luz en la imagen) y A y B la información de color o crominancia (cómo los colores se combinan y varían en una imagen), siendo A la que añade los colores en la línea del rojo y el verde y B la que añade el contenido en amarillo o azul.[1]

# 3. El problema de la cuantificación de color

## 3.1 Definición del problema

Una imagen digital se forma con una cantidad de píxeles representados usando un modelo de color concreto. Cuando este modelo de color es el RGB, como ya se ha indicado, cada píxel se representa con 3 números enteros entre el 0 y el 255 por lo cual este espacio de color nos permite usar 2563 colores diferentes.

Como se ha dicho anteriormente la cuantificación de color trata de reducir este número de colores evitando la pérdida de información visual. Para conseguirlo realiza dos operaciones:

1. **Selección de colores representativos:** Esta operación implica la elección de un conjunto limitado de colores representativos que serán utilizados para reemplazar los colores originales de la imagen. Para seleccionar estos colores, se aplican algoritmos de agrupación de colores, uno de ellos es el algoritmo K-means, que agrupa los colores similares en clusters y utiliza los centroides de estos clusters como colores representativos. Este conjunto de colores representativos lo llamamos **paleta cuantizada**. Esta operación es esencial para simplificar la representación de color de la imagen y reducir el número de colores utilizados.
2. **Asignación de colores representativos a los píxeles:** Una vez que se han seleccionado los colores representativos, se lleva a cabo la operación de asignación de colores. En esta etapa, cada píxel en la imagen original se asigna al color representativo más cercano en términos de distancia de color seleccionado en la anterior operación. Esto implica calcular la diferencia de color entre el color original del píxel y los colores representativos de la paleta cuantizada y seleccionar el color representativo que minimice esta diferencia. Sin embargo, existen otros métodos de asignación de colores, y este es solo uno de ellos. Dependiendo del algoritmo o el objetivo, se podrían emplear técnicas alternativas para asignar los colores representativos a los píxeles. Esta operación es fundamental para reemplazar los colores originales por sus equivalentes representativos y, de esta manera, lograr la reducción de color en la imagen.

A continuación, en la Ilustración 1 se muestra una imagen del paquete CQ100 [18] y en la Ilustración 2 se muestra la imagen cuantizada resultante con 32 colores.



Ilustración 1 Imagen Inicial tomada del paquete CQ100 [18]

Ilustración 2 Imagen resultante con 32 colores

## 3.2 Aplicaciones prácticas

La cuantificación de color tiene diversas aplicaciones prácticas en varios campos, tales como:

* **Procesamiento de imágenes y fotografía:** Se utiliza para reducir el número de colores en una imagen digital, lo que es útil para la compresión de imágenes y la optimización para dispositivos y medios con capacidad de color limitada.
* **Diseño gráfico y web:** Ayuda a los diseñadores a crear paletas de colores coherentes y limitadas para mantener la consistencia y mejorar la estética en diseños y sitios web.
* **Reconocimiento de patrones e identificación de objetos:** la cuantificación de color puede ayudar a simplificar la identificación y clasificación de objetos al reducir la complejidad de las imágenes, facilitando el reconocimiento de patrones.
* **Impresión y fabricación de textiles:** En la industria de la impresión y en la fabricación de textiles, la cuantificación de color es crucial para asegurar que los colores utilizados en los diseños sean reproducibles y consistentes en diferentes lotes de producción.
* **Cartografía y análisis geoespacial:** Se utiliza para simplificar la representación visual de mapas y datos geoespaciales, ayudando a resaltar características importantes y mejorar la legibilidad.
* **Análisis médico y de imágenes biomédicas:** En el campo de la medicina, la cuantificación de color se aplica en el análisis de imágenes biomédicas para mejorar la visualización y el diagnóstico de ciertas condiciones al resaltar variaciones sutiles en los tejidos.
* **Análisis de calidad y control en la industria alimentaria:** Se usa para evaluar la calidad y la madurez de alimentos basándose en su color, lo cual es especialmente útil en líneas de producción automatizadas.
* **Arte y restauración:** En el arte digital y la restauración de obras, la cuantificación de color puede ayudar a analizar y replicar los colores utilizados por los artistas originales, así como a detectar alteraciones o daños.
* **Seguridad y vigilancia:** La cuantificación de color puede mejorar el rendimiento de los sistemas de vigilancia al reducir la complejidad de las imágenes, facilitando la detección de movimientos o actividades sospechosas.
* **Investigación científica y análisis de datos:** En diversas áreas científicas, la cuantificación de color puede ser útil para visualizar y analizar datos, especialmente en representaciones gráficas de información compleja.

# 4. Los algoritmos de enjambres.

## 4.1 Características generales.

Los algoritmos de enjambres o inteligencia de enjambres es la disciplina que se ocupa de los sistemas naturales y artificiales compuestos por muchos individuos que se coordinan mediante el control descentralizado y la autoorganización. Se trata de un subcampo de la inteligencia artificial que se centra en los comportamientos colectivos que resultan de las interacciones de los individuos entre sí y con su entorno o medio ambiente al igual que lo hacen los enjambres naturales.

Algunos de los sistemas estudiados por la inteligencia del enjambre son las colonias de hormigas y termitas, los bancos de peces, las bandadas de pájaros o manadas de animales terrestres.

Ciertos artefactos humanos también caen en el dominio de la inteligencia de enjambre. En particular, algunos sistemas de múltiples robots, y también algunos programas de computadora que están escritos para abordar problemas de optimización y análisis de datos.[2]

Algunas propiedades de estos algoritmos son:

1. **La autonomía**: Cada agente (o partícula) en el enjambre opera de manera individual, tomando decisiones basadas en su información local y, posiblemente, en alguna información global disponible para el enjambre.
2. **Descentralización**: No existe un control centralizado que dirija las acciones de los agentes; en su lugar, el comportamiento global emerge de las interacciones locales entre los agentes y entre los agentes y su entorno.
3. **Distribución**: Los algoritmos de enjambres suelen ser inherentemente distribuidos, lo que los hace robustos y escalables, ya que el fallo de una partícula tiene un impacto limitado en el enjambre en su totalidad.
4. **Autoorganización**: Las interacciones entre las partículas y la aplicación de reglas sencillas dan lugar a un comportamiento colectivo "inteligente" y autoorganizado. Este comportamiento es emergente, es decir, surge de las acciones de las partículas individuales.
5. **Interacciones simples**: Las reglas que rigen las interacciones entre las partículas son simples, pero pueden dar lugar a comportamientos colectivos complejos y a la solución de problemas complejos.
6. **Retroalimentación positiva y negativa**: Los mecanismos de retroalimentación, tanto positiva como negativa, son muy importantes para el desarrollo y la estabilización de los patrones de comportamiento del enjambre. La retroalimentación positiva promueve la formación de estructuras o caminos para hallar una mejor solución mientras que la retroalimentación negativa ayuda a prevenir la saturación de datos o una mala solución rápida.
7. **Exploración y explotación**: Los algoritmos de enjambres equilibran entre la exploración del espacio de búsqueda para descubrir nuevas soluciones y la explotación de las mejores soluciones encontradas para afinarlas.
8. **Adaptabilidad**: La inteligencia de enjambre es altamente adaptable a cambios en el entorno o en los parámetros del problema, lo que permite al enjambre encontrar nuevas soluciones cuando las condiciones cambian.
9. **Robustez**: Los enjambres son robustos frente a fallos y variaciones, ya que la pérdida de algunas partículas generalmente no impide que el enjambre en su conjunto continúe funcionando eficazmente.

Algo que comparten todos los algoritmos de optimización es que realizan su búsqueda en un determinado espacio de búsqueda, en el caso de la cuantificación de color se puede definir claramente al estar determinado por el número de colores que se desea obtener en la paleta cuantizada, y cada color está representado por un vector de tres dimensiones que corresponden a las componentes RGB.

Por lo tanto, si se desea reducir la imagen a una paleta de colores, el espacio de búsqueda sería: dimensiones.

## 4.2 Algoritmo de optimización por enjambre de partículas (PSO)

### 4.2.1 Introducción

La **Optimización por Enjambres de Partículas** (conocida como **PSO**, por sus siglas en inglés, **Particle Swarm Optimization**) es una técnica de optimización/búsqueda.

Este algoritmo fue descrito alrededor de 1995 por Kennedy y Eberhart, y se inspira en el comportamiento de los enjambres de insectos en la naturaleza.

En este algoritmo se trabaja con una función objetivo que queremos optimizar, denotada comúnmente como f(x1…, xn), esta función devuelve una medida de lo buena que es una solución en cierto punto del espacio de búsqueda.

El objetivo de este algoritmo es encontrar los valores óptimos de las variables x1, x2, …, xn para maximizar (o minimizar) el valor de la función f. Esta función se denomina **función fitness.**[3]

### 4.2.3 Variables

* Se considera un conjunto de **I** individuos llamados en este algoritmo **partículas**. Cada una de las **I** partículas tendrá asociado un estado que se irá modificando con el tiempo. Estas partículas tendrán:
  + Posición (xi): representa el estado de la partícula, es una posible solución al problema.
  + Velocidad (vi): determina la actualización de la posición.
  + Mejor posición personal (bi): esta variable representa la mejor solución encontrada por la partícula durante la ejecución.
* La solución al problema viene dada por la mejor posición encontrada por el enjambre la cual llamamos la mejor posición global, representada como: g.

En este algoritmo las partículas (individuos) se mueven por el espacio de solución del problema guiadas por ellas mismas y por el conjunto de todo el enjambre.

Este movimiento hace variar su posición, velocidad y mejor posición personal.

### 4.2.4 Definición del algoritmo

**PSEUDOCODIGO**

Inicializar la población de individuos

**REPETIR**

Evaluar el fitness de cada individuo.

Actualizar la mejor solución personal de cada individuo.

Actualizar la mejor solución global.

Actualizar la velocidad y posición de cada individuo. ([Ec.1](#ec1pso), [Ec.2](#ec2pso))

**HASTA** (condición de parada)

**Inicializar la población de individuos**

Al principio del algoritmo se inicializan las variables de cada individuo su **posición** y su **velocidad.**

Su posición será un valor aleatorio dentro del espacio de búsqueda y su velocidad será un valor aleatorio entre [vmin , vmax ], que son dos parámetros del algoritmo que se definen a priori.

**Evaluar el fitness de cada individuo**

Para calcular este valor se aplica la función objetivo del problema a la posición de la partícula. En el caso de la cuantificación de color se utilizarán funciones tales como MSE (error cuadrático medio), MAE (Error absoluto medio), SSIM (Índice de similitud estructural) o MS-SSIM (Índice de similitud estructural multiescala) (a partir de ahora función objetivo).

Este valor llamado fitness determina la calidad de la solución cuya posición representa.

**Actualizar la mejor solución personal de cada individuo.**

Si al calcular el fitness un individuo encuentra una mejor solución personal (es decir, encuentra una mejor posición lo que resulta en un mejor fitness) que la que tuviese almacenada para ella hasta el momento, actualiza la mejor solución con el valor de ésta.

Se da un nuevo valor a bi

**Actualizar la mejor solución global.**

Con la solución global ocurre lo mismo que con la mejor solución personal de cada individuo: si se ha encontrado una solución que mejora a la que hubiese almacenada como mejor solución global, se guarda como nueva mejor solución global.

Se da un nuevo valor a g.

**Actualizar la velocidad y posición de cada individuo.**

En este paso se realizan dos procedimientos:

1. *Actualizar velocidad de cada individuo.*

Se calcula una nueva velocidad de cada individuo i:

Ec.1

Cuyos parámetros son:

**e1, e2:** valores aleatorios en [0, 1].

**w:** inercia.

**f1:** parámetro cognitivo.

**f2:** parámetro social.

Esta actualización incluye tres componentes que son:

-Velocidad anterior:

Donde w (inercia) debe seleccionarse atendiendo a que si es grande se realizará una exploración global del espacio de solución, si es pequeña será exploración local. Se aconseja que este parámetro decrezca con las iteraciones.

-La distancia a la mejor posición personal:

En esta componente encontramos dos parámetros:

**e1:** será un valor aleatorio entre [0, 1].

**f1:** el parámetro cognitivo que determina la importancia que se da a la experiencia propia de cada individuo.

-La distancia a la mejor solución global:

En esta componente tenemos los parámetros:

**e2:** será un valor aleatorio entre [0, 1].

**f2:** el parámetro social que determina la importancia que se da a la experiencia del enjambre.

Con el valor calculado de la velocidad, se debe ajustar al intervalo válido de la velocidad, es decir, nuestro nuevo valor: deberá estar entre .

Para ello:

-

-

1. *Actualizar posición de cada individuo.*

Se calcula la nueva posición de cada individuo i.

Ec.2

Este nuevo valor para la posición de cada individuo depende de la actual posición y de la nueva velocidad que se acaba de calcular.

### 4.2.5 Parámetros PSO

Como se ha hablado antes en el PSO existen varios parámetros principales que influyen en el comportamiento y en la eficacia del algoritmo, son: el coeficiente de **inercia**, y los parámetros **cognitivo** y **social**.

**Inercia:**

Este coeficiente afecta a la velocidad a la que los individuos se mueven en el espacio de búsqueda. Ayuda a controlar los resultados de la velocidad anterior del individuo sobre su velocidad actual. Un valor alto ayuda a que el individuo mantenga su dirección y velocidad anterior, lo que se traduce en una mejor exploración global. Por otro lado, un valor bajo hace que el individuo pierda velocidad por lo que ayuda a una exploración más local y detallada.

**Parámetro cognitivo:**

También conocido como parámetro personal o local, determina el grado en el que cada individuo se ve influido por sí mismo. Influye en cuánto se moverá un individuo hacia la mejor posición encontrada individualmente. Un valor alto de este componente dará a cada individuo una mayor motivación para seguir su propio camio hacia lo que percibe como una mejor solución, dando así lugar a una mayor variedad de soluciones exploradas por el enjambre.

**Parámetro social:**

Determina el grado de influencia de la mejor posición encontrada por todo el enjambre sobre el movimiento de cada individuo. Valores más altos de este componente resultan en una mayor convergencia del enjambre hacia las áreas del espacio de búsqueda que parecen más prometedoras según la experiencia del enjambre, es decir, los individuos preferirán moverse hacia la mejor solución encontrada por el conjunto de individuos.

## Luciérnagas (Firefly)

### 4.3.1 Introducción

El algoritmo firefly, es un algoritmo metaheurístico de optimización, inspirado en el comportamiento del centelleo de las luciérnagas [4]. El propósito primario de una luciérnaga es generar destellos de luz para actuar como sistema de señal para atraer a otras luciérnagas. Xin-She Yang formuló este algoritmo con las siguientes premisas:

1. Todas las luciérnagas son "unisexuales", de modo que cualquier luciérnaga individual será atraída por todas las demás.
2. La atracción es proporcional a su brillo, y para cualquier par de luciérnagas, la menos brillante será atraída por (y por lo tanto se desplazará hacia) la más brillante; aun así, la intensidad (el brillo aparente) decrece cuando aumenta la distancia entre ambas;
3. Si no hay luciérnagas más brillantes que una dada, esta se mueve aleatoriamente.

El brillo es asociado con los valores de una función objetivo.

### 4.3.3 Variables

En este algoritmo se considera un conjunto de I individuos (luciérnagas). Cada uno de estos individuos representa una posible solución al problema. La calidad de esa solución dependerá del **brillo** que esté asociado a cada individuo.

Posición →

Brillo →

En este contexto el **brillo** de cada individuo será el fitness de este algoritmo, es decir, el individuo que posea el mayor brillo será la mejor solución al problema. Como en este algoritmo los individuos se mueven hacia el individuo con un mayor brillo, todos ellos se acercan a la mejor solución, menos el individuo más brillante.

La **atracción** entre los individuos disminuye con la distancia debido a la absorción de la luz. Este valor normalmente va cambiando para representar la disminución (visibilidad) del brillo con la distancia.

### 4.3.4 Definición del algoritmo

**PSEUDOCODIGO**

Generar la población inicial de luciérnagas

**REPETIR**

Mover cada luciérnaga hacia las más brillantes ([Ec.3](#ec1Luciernagas) [Ec.4](#ec2Luciernagas))

Mover la luciérnaga más brillante [Ec.5](#ec3Luciernagas)

Actualizar el brillo de las luciérnagas

Ordenarlas por brillo y encontrar la mejor

**HASTA** (condición de parada)

**Generar la población inicial de luciérnagas**

En este primer paso repartimos los individuos aleatoriamente en el espacio de búsqueda y calculamos el brillo inicial de cada uno.

Por cada iteración:

**Mover cada luciérnaga hacia las más brillantes**

Cada luciérnaga (excepto la más brillante) mueve su posición hacia la más brillante, debido a esto se calcula una nueva posición para cada luciérnaga, en función de su posición previa y de la atracción ejercida por las otras luciérnagas.

Un individuo **I** se mueve hacia otro individuo **k**, que es más brillante que él, siguiendo esta ecuación:

Ec.3

Donde:

-> es la siguiente posición del individuo i

-> es la posición actual del individuo i

-> es el atractivo del individuo k sobre el i

-> un valor aleatorio

Donde β(rik) se calcula mediante:

Ec.4

-> atractivo a distancia cero

->coeficiente de absorción de la luz del medio

rij -> distancia entre los individuos i y j

**Mover la luciérnaga más brillante**

El individuo más brillante no se ve atraído por ningún otro, por lo que se mueve aleatoriamente.

Su nueva posición será:

Ec.5

**Actualizar el brillo de las luciérnagas**

En este paso se calcula el fitness (brillo) de cada individuo mediante la función objetivo del problema.

**Ordenarlas por brillo y encontrar la mejor**

En este último paso se ordenan los individuos por brillo y se determina el mejor (el que más brillo tenga). Para ello se utilizan diversos algoritmos de ordenación comunes (esta elección dependerá del número de individuos, la complejidad del algoritmo, y si el conjunto de datos esta parcialmente ordenado).

## 4.4 Lobos / Grey Wolf optimization (GWO)

### 4.4.1 Introducción

Este algoritmo de optimización metaheurística está basado en el comportamiento social y de caza de los lobos grises. Fue desarrollado por **Seyedeli Mirjalili** en 2014, simula tanto la estructura social de estos lobos como sus tácticas de caza para resolver complejos problemas de optimización. Estos lobos son conocidos por su comportamiento de caza estratégico y su jerarquía social, estos componentes se incluyen en este algoritmo para resolver estos problemas.

En esta jerarquía social, los individuos se clasifican en varias categorías: Alfas, Betas, Deltas y Omegas. Los Alfas son los que lideran la manada, estos toman las decisiones sobre la caza, el movimiento del grupo y su descanso. Los Betas, que son los colideres, ayudan a los Alfas en la toma de decisiones y en la comunicación con el resto de los individuos de la manada. Los Deltas ocupan un rango inferior, estos se ocupan de cuidar o vigilar. Por último, los Omegas están en la base de la pirámide jerárquica, estos siguen las decisiones de los miembros de rango superior sin cuestionar sus decisiones.

Este algoritmo utiliza esta jerarquía para modelar la cooperación entre los lobos en la resolución de problemas de optimización. Los mejores candidatos que son los Alfas lideran la búsqueda, estos son seguidos por los Betas y los Deltas, mientras que los Omegas siguen las direcciones establecidas por los miembros superiores. Durante el proceso de optimización, todos los lobos, que son todas las posibles soluciones, van ajustando sus posiciones respecto a los lobos de mejor desempeño en su entorno (espacio de búsqueda).

Este enfoque le permite al GWO explorar eficazmente el espacio de búsqueda y explotar las áreas más prometedoras que se acerquen a las mejores soluciones encontradas. Posee un gran equilibrio entre la exploración y la explotación de las soluciones, y una gran capacidad para no quedarse estancado en mínimos locales, el GWO ha demostrado ser efectivo en una gran variedad de aplicaciones prácticas, desde la ingeniería hasta la ciencia de datos. Desde sus inicios, este algoritmo ha generado un gran interés en la comunidad de investigación por su robustez y rendimiento superior en comparación con otras técnicas de optimización.[5]

### 4.4.2 Fundamentos

El GWO introduce I lobos en la manda, desde aquí llamados I individuos.

-Cada individuo representa una posible solución al problema.

-La calidad o fitness de esa solución se calcula mediante la función objetivo del problema.

Esta jerarquía se representa de la siguiente manera:

Los tres mejores lobos de la manada estarán en los tres niveles superiores de la jerarquía, es decir, en cada uno de estos tres niveles habrá un solo lobo.

-El mejor lobo (mejor solución) se denota como α. ->

-El segundo mejor lobo se denota como β. ->

-El tercer mejor lobo se denota como δ. ->

-El resto de los lobos se consideran lobos ω. ->

Ilustración 3 Pirámide

El proceso de optimización está representado por la operación de caza, en esta actividad los lobos de los tres primeros niveles guían, mientras que el resto siguen a éstos.

Este algoritmo posee varios coeficientes sociales (a, A y C).

El parámetro a o **factor de exploración** es el que guía el proceso de búsqueda. Este parámetro varía linealmente desde 2 hacia 0 con las iteraciones y se utiliza para calcular dos vectores aleatorios (A y C) que serán los condicionantes de la exploración/explotación del algoritmo.

Estos vectores al depender de a también disminuirán con las iteraciones. Estos se calculan de la siguiente manera:

Ec.6 Ec.7

Donde:

: son vectores aleatorios en [0, 1].

: es el parámetro del algoritmo (factor de exploración)

Cuando el valor absoluto de estos vectores es mayor que uno se entiende que los individuos divergen respecto de la presa, se dice que están en fase de exploración.

Al converger a la presa, este valor será menor que uno, en este caso están en fase de explotación.[6]

### 4.4.3 Definición del algoritmo

En el GWO la búsqueda de la presa, es decir, la solución del problema se inicia con la creación de una población de I individuos al azar, igual que en el resto de los algoritmos.

Después de esto se realiza un proceso iterativo en el cual:

1. Los lobos: alfa, beta y delta, estiman la posición probable de la presa.
2. Cada lobo del grupo actualiza su distancia respecto de la presa.
3. Se decrementa el parámetro a.

**PSEUDOCODIGO**

Inicializar la población de lobos

Calcular el fitness de los lobos

Determinar los lobos alfa, beta y delta

**REPETIR**

Actualizar la posición de cada lobo [Ec.8](#ec3Lobos)

Ajustar el parámetro a [Ec.9](#ec10Lobos)

Calcular el fitness de los lobos

Determinar los lobos alfa, beta y delta

**HASTA** (condición de parada)

En el primer paso, **inicializar la población de individuos**, se escogen unos valores aleatorios del espacio de solución para la posición de cada individuo Xi

Aparte de esto también se inicializa el parámetro a, este factor de exploración comienza con el valor de 2.

En el segundo paso, se calcula el **fitness** de cada individuo, esto se hace en base al valor de la función objetivo en Xi

Lo siguiente es decidir los individuos alfa, beta y delta que serán los individuos con mejor fitness.

Xα, Xβ y Xδ

En el bucle, el primer paso será actualizar la posición de cada individuo. Esta nueva posición será:

Ec.8

Ec.4 Ec.7

Ec.5 Ec.8

Ec.6 Ec.9

En donde:

T: es la iteración actual.

Ai, Ci (i=1, 2, 3): vectores de coeficientes, calculados usando la [Ec6](#ec1Lobos) y la [Ec7](#ec2Lobos).

En el siguiente paso, se **actualiza el parámetro** del algoritmo.

El factor de exploración (a) se reduce linealmente en el intervalo [2,0] al avanzar las iteraciones, según:

Ec.9

En donde:

: es la iteración actual del algoritmo.

: es el número máximo de iteraciones del algoritmo.[7]

Para **calcular el fitness de cada individuo** se aplica la función objetivo del problema.

Por último, para **encontrar los mejores individuos** (alfa, beta y delta) se toman los 3 individuos con mejor fitness.

### 4.4.4 Variables

Como en cualquiera de estos algoritmos tenemos **el tamaño de la población** de individuos (I), en este caso es el número de lobos que existen en la población. Un mayor número de individuos puede mejorar la exploración del espacio de búsqueda, pero a su vez aumenta el costo computacional.

Ilustración 4 Lobos Grises – Referencia: yocambioelmundo.org

En el GWO existen 3 parámetros o coeficientes sociales los cuales ya se han mencionado que son a, A y C.

* a: Disminuye linealmente de 2 a 0 a lo largo de las generaciones y se utiliza para calcular los coeficientes A y C.
* A: Calculado de la forma ([Ec6](#ec1Lobos)), es un vector aleatorio cuyos valores están entre el [0,1]. Este coeficiente afecta a la amplitud del movimiento del individuo (lobo) hacia o desde la presa.
* C: Calculado de la forma ([Ec7](#ec2Lobos)), este coeficiente influye en el reconocimiento y la localización de la presa, proporcionando un elemento de aleatoriedad en el acercamiento de los lobos a la presa.

Jerarquía de Lobos (Alfa, Beta, Delta)

* Alfa (α): El lobo líder, que posee la mejor solución encontrada hasta el momento.
* Beta (β): El segundo mejor lobo, apoya al Alfa en la toma de decisiones y mantiene la disciplina dentro del grupo.
* Delta (δ): El tercero en la jerarquía, juega un rol subordinado pero esencial, facilitando la comunicación entre el Alfa y Beta con el resto de la manada.
* Omega (ω): El resto de los lobos, siguen las direcciones de los lobos con rango superior.

De la misma manera que el resto de los algoritmos, el GWO también intenta optimizar una función, llamada función objetivo que es la que determina la calidad de cada solución.

Por último, los criterios de terminación del problema, que son las condiciones bajo las cuales el algoritmo debe finalizar, ya pueden ser una solución concreta, el número máximo de iteraciones o una limitación del tiempo de ejecución.

## 4.5 El algoritmo de Ballenas (Whale Swarm Algorithm - WSA)

### Introducción

El WSA parte de la idea del WOA (Whale Optimization Algorithm), este algoritmo inspirado en el comportamiento de caza de las ballenas jorobadas fue desarrollado por Seyedali Mirjalili en 2016.

El WOA simula la técnica de burbujeo empleada por estos animales, es una estrategia predadora que utiliza la creación de burbujas en círculos o redes alrededor de su presa antes de atraparla. Este comportamiento es interesante desde un punto de vista de optimización, ya que engloba tanto los elementos de exploración como de explotación, que son las claves para la eficacia en la búsqueda de soluciones optimas en espacios complejos y multidimensionales.

Este algoritmo posee una estructura simple pero potente que permite resolver una amplia gama de problemas de optimización en ingeniería, informática y más allá. La adaptabilidad y eficiencia del WOA se deben a su capacidad para imitar la dinámica natural de estas ballenas, ajustando sus posiciones en el espacio de búsqueda de manera que maximiza las posibilidades de localizar y converger hacia un óptimo global.

### 4.5.2 Fundamentos

El WOA se basa en el comportamiento de caza de las ballenas jorobadas, particularmente en la técnica conocida como "burbujeo", donde las ballenas crean burbujas en círculos concéntricos alrededor de su presa. Esta técnica no solo confina a la presa, sino que también reduce sus vías de escape, facilitando a la ballena capturarla eficientemente.

Modelo Matemático del WOA

El WOA utiliza dos mecanismos principales basados en este comportamiento: la encerrona por burbujeo y los movimientos en espiral hacia la presa. La elección entre estos dos comportamientos se modela mediante un enfoque probabilístico que depende de la proximidad de la solución candidata al mejor candidato actual (la presa).

Encerrona por Burbujeo:

En este enfoque, las posiciones de las ballenas se ajustan según la posición del mejor candidato actual. Donde la posición se mueve hacia el mejor candidato o se aleja ligeramente de él basándose en un coeficiente aleatorio.

Movimiento en Espiral:

Este comportamiento simula el movimiento en espiral de las ballenas alrededor de su presa. Se utiliza una ecuación helicoidal para actualizar la posición de la ballena en dirección a la presa.

Características del WOA

-Exploración y Explotación:

El WOA equilibra eficazmente la exploración (buscando en nuevas áreas del espacio de búsqueda) y la explotación (intensificando la búsqueda cerca del óptimo conocido). La alternancia entre el comportamiento de encerrona por burbujeo y el movimiento en espiral permite este equilibrio, haciendo que el WOA sea robusto en encontrar soluciones globales.

-Flexibilidad y Aplicabilidad:

El algoritmo se ha aplicado en una variedad de problemas, desde optimización de funciones continuas hasta problemas de diseño industrial, demostrando su versatilidad y efectividad.

-Simplicidad de Implementación:

A pesar de su eficacia, el WOA es relativamente simple de implementar comparado con otros algoritmos metaheurísticos, lo que lo hace accesible para investigadores y profesionales.[8]

### 4.5.3 Definición del algoritmo

En este algoritmo se comienza por inicializar una población de I individuos, seguido por la evaluación de estos individuos basada en la función objetivo y se actualizan sus posiciones.

**PSEUDOCODIGO**

Inicializar la población de individuos.

**REPETIR**

PARA i+1 HASTA número de individuos HACER

Se actualiza a, A, C y p. [Ec.10](#ec1woa) [Ec.11](#ec2woa) [Ec.12](#ec3woa)

SI

SI

Se actualiza la posición respecto a la mejor solución. [Ec.13](#ec4woa) [Ec.14](#ec14)

SI NO

Se selecciona un individuo aleatorio y se actualiza su posición. [Ec.15](#ec6woa) [Ec.16](#ec7woa)

FIN SI

SI NO

Se mueve el individuo en un camino en espiral alrededor de la mejor solución. [Ec.17](#ec8woa)

FIN SI

Se actualiza la mejor solución global.

FIN-PARA

**HASTA** (condición de parada)

**Inicialización de la población**

Para ello se inicializan todos los individuos con posiciones aleatorias en el espacio de búsqueda permitido, después se calcula el fitness de cada individuo según la función objetivo y por último se calcula la mejor solución (individuo con mejor fitness).

También en este paso se inicializan los parámetros del algoritmo, el parámetro comienza con un valor de 2 y el como 1.

Después de este primer paso se actualizan los parámetros de este algoritmo para cada individuo siguiendo las siguientes ecuaciones.

**Actualización de** :

Ec.10

Donde:

es la iteración actual.

es el número total de iteraciones permitidas.

**Actualización de A**

Ec.11

Donde:

es el parámetro anteriormente calculado

r es un vector aleatorio en [0,1]

**Actualización de C**

Ec.12

Donde:

r es un vector aleatorio en [0,1]

**Se actualiza la posición respecto a la mejor solución:**

Se calcula el vector D

Ec.13

Donde:

C es el parámetro calculado anteriormente mediante la Ec.12.

X\* es la posición del mejor individuo

X es la posición del individuo que se está evaluando

Con D se calcula la nueva posición del individuo

Ec.14

**Se selecciona un individuo aleatorio y se actualiza su posición**, para ello se aplica la ecuación:

Para la cual se aplican esta ecuación para calcular

Ec.15

Ec.16

Donde:

es un individuo seleccionado aleatoriamente.

**Se mueve el individuo en un camino en espiral alrededor de la mejor solución** mediante esta ecuación que simula el movimiento en espiral hacia la presa:

Ec.17

Dónde:

es un parámetro que usa este algoritmo para definir el movimiento en espiral

es un número aleatorio en el intervalo [-1,1]

En la Ec.17 se utiliza la siguiente ecuación para calcular

## 4.6 Abejas – Algoritmo Colonia de Abejas Artificiales (ABC)

### 4.6.1 Introducción

El Algoritmo de Colonia de Abejas Artificiales (Artificial Bee Colony, ABC) es una técnica de optimización metaheurística que fue desarrollada por Karaboga en 2005. Inspirado por el comportamiento forrajero de las abejas, el ABC simula las dinámicas de búsqueda de comida de una colmena de abejas para encontrar soluciones óptimas en problemas complejos. Este algoritmo se basa en el modelo de división del trabajo y el reclutamiento mutuo entre abejas, características que permiten una exploración eficiente del espacio de soluciones.

Este proceso iterativo de exploración y explotación sigue hasta alcanzar una condición de terminación. Este algoritmo ha demostrado ser efectivo en problemas de optimización numérica y ha sido aplicado en muchos campos como la ingeniería, la investigación operativa y la inteligencia artificial. La simplicidad de este algoritmo sumada a su potente capacidad de optimización ha ayudado en gran parte a su popularidad y gran aceptación en la comunidad científica.[9], [10]

### 4.6.3- Variables

En el algoritmo ABC se define un conjunto de **F fuentes de alimento**, donde cada fuente representa una posible **solución al problema de optimización**. La posición de cada fuente está asociada a una solución y se considera como una variable clave en la búsqueda de soluciones óptimas.

* **Posición de la fuente**: Representa una solución factible dentro del espacio de búsqueda.
* **Néctar o calidad de la fuente**: La calidad o **fitness** de una fuente es una medida de su valor en términos de la función objetivo, es decir, cuanto mayor es la calidad de la fuente, mejor es la solución que representa.
* **Agotamiento de la fuente**: A medida que se explota una fuente, su **valor de calidad puede no mejorar**. Esto se modela a través de una variable que cuenta los intentos fallidos para mejorar dicha fuente. Si esta variable supera un cierto umbral, la fuente se considera agotada y se reemplaza por una nueva fuente aleatoria.

En el ABC, existen tres tipos de abejas que interactúan con estas fuentes de alimento:

* **Abejas exploradoras**: Se encargan de buscar nuevas fuentes de manera aleatoria, explorando áreas no visitadas previamente.
* **Abejas empleadas**: Se asignan a fuentes específicas y exploran en sus alrededores en busca de mejores soluciones.
* **Abejas observadoras**: Evalúan la información proporcionada por las abejas empleadas y seleccionan las fuentes más prometedoras para explotarlas.

Esta interacción entre los diferentes tipos de abejas y las fuentes de alimento permite al algoritmo buscar, explotar y reemplazar soluciones de manera eficiente a lo largo de las iteraciones.

### 4.6.4- Fundamentos del algoritmo

El algoritmo ABC sigue una estructura cíclica, donde diferentes tipos de abejas realizan operaciones sobre las fuentes de alimento, buscando mejorar la solución óptima. A continuación, se presenta el pseudocódigo simplificado que resume las principales etapas del algoritmo:

Inicializar el conjunto de fuentes de alimento

**REPETIR**

Operaciones de abejas empleadas [Ec.19](#ec2Abejas)

Operaciones de abejas observadoras [Ec.20](#ec3Abejas)

Operaciones de abejas exploradoras

Actualizar la mejor solución hasta el momento

**HASTA QUE** (condición de parada)

**Inicialización de las fuentes de alimento**

El proceso de inicialización consiste en seleccionar F fuentes de alimento dentro del espacio de búsqueda. A cada fuente se le asigna una posición aleatoria en dicho espacio, de acuerdo con la siguiente fórmula:

Ec.18

Dónde:

: número aleatorio en el intervalo [0, 1]

y : límites inferior y superior de la dimensión j del espacio de búsqueda.

: posición de la fuente i en la dimensión j.

Además, a cada fuente se le asigna una variable llamada **limit**, que inicialmente comienza con el valor 0. Esta variable se utiliza para definir cuántos intentos se han realizado para mejorar la fuente sin éxito. Si este valor supera un valor previamente definido, la fuente es reemplazada.

**Operaciones de abejas empleadas.**

En el algoritmo ABC, cada **abeja empleada** está asignada a una fuente de alimento, cuya posición en el espacio de búsqueda representa una solución candidata al problema de optimización. Estas abejas exploran la vecindad de la fuente asignada en busca de nuevas soluciones mejores, basándose en la calidad de la fuente (valor de fitness). Si encuentran una fuente de mayor calidad, se mueven hacia ella; de lo contrario, incrementan el valor del contador **limit**, que indica cuántos intentos han hecho sin éxito. El pseudocódigo de este tipo de abejas sería:

**Pseudocódigo de las Abejas Empleadas:**

**Para cada abeja empleada** i:

Obtener una **fuente candidata** vi​ próxima a la fuente actual xi

**Si** fitness(vi) < fitness(xi):

Moverse a la **nueva fuente** vi​.

**Sino**:

limit = limit + 1

Fin-si

Fin-para

Estas abejas para elegir la fuente candidata próxima, también llamada fuente vecina (vi), siguen esta ecuación:

Ec. 19

Donde:

: Fuente vecina elegida al azar con k ≠ i (Fuente distinta a la actual)

: número aleatorio en [-1,1]

En cada iteración, las abejas empleadas generan una nueva fuente candidata vi​, desplazándose desde la posición xi ​ hacia la fuente vecina xk​, ajustada por el valor aleatorio αi​. Si esta nueva fuente mejora la solución actual, la abeja se moverá hacia la fuente candidata.

**Operaciones de abejas observadoras.**

Las **abejas observadoras** eligen las fuentes de alimento que están siendo explotadas por las abejas empleadas, basándose en la calidad o **fitness** de dichas fuentes. La selección de la fuente se realiza en función de una **probabilidad** que refleja la calidad relativa de cada fuente en comparación con el resto. Una vez que una abeja observadora ha seleccionado una fuente, procede de manera similar a una abeja empleada, buscando una solución en la vecindad de la fuente seleccionada.

**Pseudocódigo de las Abejas Observadoras:**

**Para cada abeja observadora** i:

Elegir una fuente de alimento xi para explotar, en base a una probabilidad **pi** ([Ec. 20](#ec3Abejas))

Obtener una fuente candidata vi próxima a la fuente actual xi.

Si fitness(vi) < fitness(xi):

Moverse a la nueva fuente vi.

Sino

limiti=limiti+1

Fin-si

Fin-para

**Cálculo de la Probabilidad de Selección**

La selección de la fuente de alimento por parte de las abejas observadoras se realiza utilizando una **probabilidad de selección** ​, que está determinada por la calidad (fitness) de cada fuente en comparación con el resto de las fuentes. Esta probabilidad se calcula con la siguiente fórmula:

Ec.20

: valor de fitness de la fuente i-ésima

: contador con valores de 1 y el número de fuentes.

Las abejas observadoras eligen fuentes de alimento de manera probabilística, de acuerdo con la calidad relativa de cada fuente. Fuentes con un mayor valor de fitness tienen una probabilidad más alta de ser seleccionadas.

Después de seleccionar una fuente, esta abeja busca una fuente candidata vi cercana utilizando la [Ec. 19](#ec2Abejas)

Si la fuente candidata mejora la solución actual, la abeja observadora se mueve hacia ella, de lo contrario, incrementa el contador limiti​ asociado a la fuente actual.

**Operaciones de abejas exploradoras.**

Cuando una fuente de alimento se agota, es decir, no se ha podido mejorar después de un número determinado de intentos, la fuente se abandona y es reemplazada por una nueva fuente generada aleatoriamente. En esta fase, la abeja empleada asignada a esa fuente agotada se convierte en una abeja exploradora y busca una nueva fuente de alimento en el espacio de búsqueda. Esta nueva fuente reemplaza a la fuente agotada.

**Para cada abeja empleada i**:

**Si** limiti ​ (número de intentos fallidos) supera el valor máximo permitido:

Generar una **nueva fuente de alimento** vi​, asignando una nueva posición aleatoria en el espacio de búsqueda. [Ec.18](#ec1Abejas)

Reemplazar la fuente actual xi​ por la nueva fuente vi​.

limiti =0

Fin-si

**Fin-para**

**Mecanismo de Exploración**

* La **abeja empleada** asociada a la fuente agotada se convierte en **abeja exploradora**, y su objetivo es encontrar una **nueva fuente** aleatoria en el espacio de búsqueda.
* Una vez que se encuentra una nueva fuente vi​, la abeja empleada olvida la fuente agotada xi​ y comienza a explotar la nueva fuente vi​.
* El valor del contador limiti​​, que se utiliza para rastrear el número de intentos fallidos de mejora, se reinicializa a 0.

# 5. Tecnologías empleadas

En este apartado se describirán las tecnologías que se han utilizado para el desarrollo de este proyecto.

## 5.1 Python

Lenguaje de alto nivel elegido para la implementación de los algoritmos de optimización por enjambre y procesamiento de imágenes debido a su facilidad de uso, su rica colección de bibliotecas científicas, y su soporte para la creación de entornos virtuales.

Fue creado a finales de los años ochenta por Guido van Rossum en Stichting Mathematisch Centrum (CWI),​ en Países Bajos. Fue creado como sucesor del lenguaje ABC, capaz de manejar excepciones e interactuar con el sistema operativo Amoeba.[11], [12]

Es un lenguaje interpretado y de propósito general, conocido por su simpleza y legibilidad, lo que lo convierte en una opción popular tanto para principiantes como para desarrolladores experimentados. Se utiliza en una gran variedad de ámbitos, desde el desarrollo web hasta ciencia de datos, inteligencia artificial, blockchain …

Ilustración 5 Logo de Python (https://es.m.wikipedia.org/)

## 5.2 Numphy

NumPy es una librería de Python especializada en el cálculo numérico y el análisis de datos, especialmente para un gran volumen de datos.

Incorpora una nueva clase de objetos llamados arrays que permite representar colecciones de datos de un mismo tipo en varias dimensiones, y funciones muy eficientes para su manipulación.

La ventaja de Numpy frente a las listas predefinidas en Python es que el procesamiento de los arrays se realiza mucho más rápido (hasta 50 veces más) que las listas, lo cual la hace ideal para el procesamiento de vectores y matrices de grandes dimensiones.[13]

Ilustración 6 Numpy (https://numpy.org/)

## 5.3 OpenCv

OpenCV es una librería de computación visual para el procesamiento de imágenes en Python. Esta biblioteca proporciona herramientas para realizar operaciones de procesamiento de imágenes, como el filtrado, la detección de bordes, el reconocimiento de características, el seguimiento de objetos, etc. Estas herramientas nos permiten desarrollar aplicaciones de visión artificial, como el reconocimiento facial, el seguimiento de objetos, etc. [14]

Esta librería ha sido muy útil en este proyecto debido a que permite el procesamiento y manipulación de imágenes. Es fundamental para la cuantificación de color en imágenes.

## 5.4 Scikit-learn

Es una biblioteca de Python que proporciona acceso a versiones eficaces de muchos algoritmos comunes. También proporciona una API propia y estandarizada. Por tanto, una de las grandes ventajas de Scikit-Learn es que una vez que se entiende el uso básico y su sintaxis para un tipo de modelo, cambiar a un nuevo modelo o algoritmo es muy sencillo. La biblioteca no solo permite hacer el modelado, sino que también puede garantizar los pasos de preprocesamiento.

La gran variedad de algoritmos y utilidades de Scikit-learn la convierten en la herramienta básica para empezar a programar y estructurar los sistemas de análisis datos y modelado estadístico. Los algoritmos de Scikit-Learn se combinan y depuran con otras estructuras de datos y aplicaciones externas como Pandas o PyBrain.

La ventaja de la programación en Python, y Scikit-Learn en concreto, es la variedad de módulos y algoritmos que facilitan el aprendizaje y trabajo del científico de datos en las primeras fases de su desarrollo.[15]

OJO: SECCIONES MAL NUMERADAS. P.E. TIENES DOS VECES 4.4 Y 4.5

## 5.5 Scikit-image

Esta es otra librería de procesamiento de imágenes de código abierto que está diseñada para operar con las bibliotecas numéricas y científicas de Python como Numpy o SciPy.

El proyecto scikit-image comenzó como scikits.image, de Stéfan van der Walt. Su nombre proviene de la idea de que es un "SciKit" (SciPy Toolkit), una extensión de terceros desarrollada y distribuida por separado para SciPy. El código base original fue posteriormente reescrito en profundidad por otros desarrolladores. De los diversos scikits, scikit-image y scikit-learn fueron descritos como "bien mantenidos y populares" en noviembre de 2012. [16]

## 5.6 Entornos virtuales

AQUÍ TIENES QUE CONTAR ALGO MÁS

Esta característica me ha permitido aislar las dependencias del proyecto, asegurando que las versiones correctas de las librerías se utilicen sin conflictos con otros proyectos.

Una vez que se tiene el entorno creado hay que activarlo para poder ejecutar el programa, esto se hace con source entorno/bin/activate

## 5.7 Control de versiones

Para el desarrollo del proyecto se ha utilizado en todo momento la herramienta de GitHub lo que me ha permitido tener un control visual de los cambios que realizaba en el código y poder probar en diferentes entornos de trabajo con rapidez.

Ilustración 7 GitHub (https://1000logos.net)

## 5.8 Visual Studio Code

Ha sido el editor de código que he usado en todo el proyecto, permitiéndome la integración de los entornos virtuales de Python y la depuración del código.

Icono

Descripción generada automáticamenteTambién me facilitó una manera de pasarle diferentes argumentos requeridos por el programa desde el propio editor formando un JSON con todos ellos.

Ilustración 8 Visual Studio Code (https://techriders.tajamar.es/)

## 5.9 Otros

Aparte de todo lo anterior se realizó un script en bash para automatizar el proceso de ejecución de los algoritmos y la generación de resultados. Dicho script lanza el programa con diferentes argumentos y vuelca los resultados en diferentes txt para recuperar los datos para formar los resultados. En estas sucesivas llamadas al programa se cambia el algoritmo que se usa, la función con la que se calcula el fitness y también las imágenes que Captura de pantalla de computadora

Descripción generada automáticamenteprocesa.

Ilustración 9 JSON de argumentos

LA IMAGEN 9 TENDRÍA QUE IR EN LA SECCIÓN 4.6, Y PONER EN ESTA EL SCRIPT

# 6. Pruebas

En este apartado se exponen cómo se han realizado las pruebas de la herramienta desarrollada.

Se parte de que la valoración de la imagen resultante es objetiva, es decir se calcula un valor de error basado en una fórmula matemática.

Para esta herramienta se usan índices que utilizan una imagen de referencia (Full-reference indices) ya que se tiene la imagen original, aunque también existen índices que solo usan la foto resultante.

Los índices utilizados se describen a continuación.

TIENES QUE NUMERA LAS ECUACIONES DE LAS SUBSECCIONES SIGUIENTES

## 6.1 MSE

El MSE o error cuadrático medio mide el promedio de los errores al cuadrado, es decir, la diferencia entre el estimador y lo que se está estimando. Su ecuación es la siguiente:

Dónde:

es el número total de datos (pixeles, en el caso de imágenes).

es el valor real del dato .

​ es el valor predicho para el dato .

## 6.2 MAE

El MAE o error absoluto medio es una medida de la diferencia entre dos variables continuas. A continuación, se muestra su ecuación:

Dónde:

es el número total de datos (pixeles, en el caso de imágenes).

es el valor real del dato .

​ es el valor predicho para el dato .

## 6.3 SSIM

El SSIM es una métrica diseñada para medir la similitud entre dos imágenes. A diferencia de otras métricas como el MSE o el PSNR, que simplemente cuantifican las diferencias absolutas entre los valores de píxeles, el SSIM se basa en el modo en que los humanos perciben las diferencias estructurales entre las imágenes.

Su ecuación es la siguiente:

Dónde:

y ​ son las medias de las imágenes e , respectivamente.

​ y ​ son las varianzas de e .

es la covarianza entre e .

​ y son constantes para evitar divisiones por cero.

MARISA – PAR DE REVISAR AQUÍ (26/Mayo/2025)

## 6.4 UQI

DESARROLLAR

## 6.5 Tablas de resultados

Cuando acabe abejas rellenar este punto (Poner también las características del ordenador)

# 7. Desarrollo del programa

DEBO DESARROLLAR AUN ESTE PUNTO, QUITARE TODO Y HARE INCAPIÉ SOLO EN EL DESARROLLO DEL SCRIPT QUE GENERA LOS RESUTADOS, APARTE DE COMO SE EJECUTA LA HERRAMIENTA, DESCRIBIRE EL ARCHIVO EJECUTOR Y EL SCRIPT

Para el desarrollo como se ha mencionado antes se realizó un programa en Python que realiza las operaciones de los algoritmos de enjambre y genera imágenes cuantizadas con las que calcula un error con 4 índices de error diferentes (MAE, MSE, SSIM, MS-SIM), este programa se llama con diferentes argumentos que se explicarán en este apartado.

Por otro lado, también se realizó un script para la automatización de la obtención de resultados que se encarga de llamar al programa repetidas veces para ejecutarlo con todos los algoritmos, todas las funciones y 4 paletas de colores diferentes, estas 4 paletas son de 32 colores, de 64, de 128 y de 256.

## 5.1 Archivo main o ejecutor

Este archivo es el principal del programa encargado de recibir los argumentos que se le pasen y de elegir o “mapear” estos argumentos con el algoritmo que toque junto con la función que toque.

Para recibir estos argumentos se ha utilizado la librería por defecto de Python llamada argparse.

Pantalla de computadora con fondo negro

Descripción generada automáticamente con confianza baja

Ilustración 10 Argumentos

En la ilustración 10 se ve como se añaden los argumentos de tipo cadena o de tipo entero, estos argumentos por orden son:

* Imagen: imagen que se va a procesar en el algoritmo.
* NumeroColores: el número de colores de la nueva imagen o la paleta que se va a usar.
* Algoritmo: algoritmo que usará el programa para la cuantificación de color. Estos pueden ser: PSO, FA, BA, GWO, ABA que respectivamente son: pso, luciérnagas, ballenas, lobos y abejas.
* Función: función que calculará el fitness de cada iteración del algoritmo, las opciones son: MSE, MAE, SSIM, MSSIM.
* Iteraciones: número entero que representa el número de iteraciones que realizara el algoritmo escogido.
* Individuos: número entero que representa la población de individuos que realizarán el algoritmo.
* PintaImagen: este argumento booleano solo lo usé para mis pruebas, si viene a True el programa mostrará por pantalla las dos imágenes, tanto la original como la resultante, me fue útil al principio para comprobar visualmente que la imagen resultante realmente estaba modificada.

Ilustración 11 Obtener ruta de la imagen

En la ilustración 11 se ve la línea que obtiene la ruta de la imagen que se le ha pasado. Se ha usado el módulo os para obtenerla de manera que funcione bien en cualquier sistema operativo. Esta operación se realiza debido a que las imágenes que se procesan en el algoritmo están en una carpeta images que está ubicada al mismo nivel que el archivo ejecutor, si estuviesen al mismo nivel que ejecutor no haría falta, pero esto sería más caótico. args.imagen es la forma de obtener la imagen que se le ha pasado como argumento.

Texto

Descripción generada automáticamentePantalla de computadora con letras

Descripción generada automáticamente con confianza mediaEn la ilustración 12 se ven algunas comprobaciones necesarias por si no se pudiese encontrar el archivo en la carpeta images y la comprobación de que se puede leer o abrir la imagen pasada.

Ilustración 12 Comprobaciones

Ilustración 13 Cuerpo del programa

En la ilustración 13 se muestra el cuerpo del programa, donde se evalúa que algoritmo ha pedido el usuario y se llama a la función que mapea que función va a usar el propio algoritmo.

Texto

Descripción generada automáticamente con confianza media

Ilustración 14 Mapeo de funciones

En la ilustración 14 se ve una de las funciones que mapean que función va a usar el algoritmo y llama al algoritmo que se haya pedido con la función que se necesite.

Todas estas operaciones constituyen la primera parte del programa, donde se reciben argumentos, se comprueban y se “escoge” el algoritmo a usar junto a la función seleccionada.

## 5.2 Algoritmos

A continuación, se explica el desarrollo de los algoritmos utilizados.

### 5.2.1 PSO

Texto

Descripción generada automáticamente

Ilustración 15 Comienzo PSO

En la ilustración 15 se muestran los imports de este algoritmo y dos variables globales que definen el comportamiento de la velocidad máxima y mínima para cada individuo del algoritmo en este caso 4 y -4.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Ilustración 16 Constructor PSO

En la ilustración 16 comienza la clase que define el PSO y su constructor, sus parámetros son los siguientes:

* n: número de individuos.
* function: función que se aplica en el algoritmo.
* lb: límite inferior del espacio de búsqueda.
* ub: límite superior del espacio de búsqueda.
* dimension: dimensión del espacio de solución (r).
* iteration: número de iteraciones.
* numeroColores: número de colores de la nueva imagen.
* pintor: booleano que se usa para saber si pintamos imagen al final o no (Es el argumento pintaImagen del programa).
* w: parámetro inercia.
* c1: parámetro cognitivo.
* c2: paramero social.
* imagen: ruta de la imagen a procesar por el algoritmo.

Texto

Descripción generada automáticamente

Ilustración 17 Inicialización PSO

En la anterior ilustración se ve como se inicializa cada uno de los parámetros.

Gracias al constructor de intelligence se inicializan posiciones, mejores posiciones y mejor fitness, todo a 0 en un principio (Ilustración 18)

También se generan números aleatorios gracias a la librería numpy de manera uniforme y aleatoria, esto da las primeras posiciones de cada individuo.

Se inicializa el vector de velocidad a 0 y las mejores posiciones de los individuos que son las primeras que obtuvimos antes.

Se calcula el fitness actual de todos los individuos y se inicializa el mejor fitness con esos valores.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamentePor último se inicializa la mejor posición global llamando a nuestra función objetivo (MSE, MAE, SSIM o MSSIM) para cada una de las posiciones de cada individuo, el resultado de esta función es el fitness de cada individuo por lo que con argmin() obtenemos el menor resultado, es decir, la mejor posición de todos los individuos.

Ilustración 18 Constructor de Intelligence

Texto

Descripción generada automáticamente

Ilustración 19 Comienzo bucle PSO

En la ilustración 19 comienza el bucle del PSO, es decir, las instrucciones que va a realizar el algoritmo en cada una de las iteraciones.

Comienza por calcular r1 y r2 llamando a numpy para obtener una matriz de números aleatorios entre 0 y 1 (np.random.rand()) de tamaño (n, numeroColores, dimension), donde para cada individuo n se generan numeroColores valores aleatorios en un espacio de dimension dimensiones, es decir, si tenemos 10 individuos y una paleta de colores de 64 el resultado sería una matriz de tamaño (10, 64, 3) y sus valores serian números aleatorios entre 0 y 1.

En la línea 62 se calcula la velocidad siguiendo la fórmula de la velocidad del PSO definida en la página 19, después se ajusta esta velocidad con clip() a los valores permitidos (entre -4 y 4).

Se ajusta la posición de las partículas sumando la velocidad, se acota esta posición a los límites del espacio permitidos.

Una captura de pantalla de un celular con texto e imagen

Descripción generada automáticamente con confianza media

Ilustración 21 Actualización mejor solución particular PSO

En la ilustración 21 se muestra la actualización de los valores del fitness actual, ejecutando la función objetivo para todos los individuos.

Después se muestra un bucle que recorre todos los individuos que tenga el algoritmo en el cual se comprueba si el fitness actual es mejor que los mejores valores de fitness guardados hasta el momento, si esto se cumple se actualiza la mejor posición del individuo y el mejor fitness del individuo.

Ilustración 22 Actualización mejor solución global PSO

Texto

Descripción generada automáticamentePor último, según como se muestra en la ilustración 22 se actualiza la mejor solución global cogiendo el índice del individuo cuyo fitness sea menor gracias a la librería Numphy y su función argmin()

Captura de pantalla de un celular

Descripción generada automáticamenteSe actualiza el mejor fitness hallado y se imprime.

Ilustración 23 Fin PSO

En esta ilustración se muestra el final del PSO en el que se guarda la mejor solución global obtenida por el algoritmo, y se genera una imagen cuantizada con esa mejor solución (mejor paleta de colores) para luego pintarla. Esta última generación de cuantizada no hace falta si no se pinta.

### 5.2.2 Luciérnagas

De igual manera que en el anterior algoritmo se comienza con el constructor.



Ilustración 24 Constructor Luciérnagas

En esta ilustración se muestra el constructor cuyos parámetros son:

* n: número de partículas
* function: función a optimizar
* lb: límite inferior del espacio (0 para imágenes)
* ub: límite superior del espacio (255 para imágenes)
* dimension: dimensiones del espacio
* iteration: número de iteraciones
* numeroColores: número de colores de la nueva imagen
* pintor: booleano que se usa para saber si pintamos imagen al final o no.
* csi: atracción mutua
* psi: Coeficiente de absorción de la luz del medio
* alpha0: valor inicial del parámetro aleatorio alpha
* alpha1: valor final del parámetro aleatorio alpha
* norm0: primer parámetro para una distribución normal (Gaussiana)
* norm1: segundo parámetro para una distribución normal (Gaussiana)
* imagen: ruta de la imagen a procesar por el algoritmo

Captura de pantalla con la imagen de una pantalla

Descripción generada automáticamente con confianza media

Ilustración 25 Inicio luciérnagas

Se continúa inicializando intelligence, la población de individuos y las mejores posiciones de cada individuo de la misma manera que en el anterior algoritmo.

Por último, se calculan todos los fitness, fitnessP representa los mejores valores de fitness (mejores posiciones) que se han encontrado y fitnessA el fitness actual, como se esta inicializando no importa que no sea del todo correcto ya que en un principio tanto la mejor posición como el mejor fitness coincide con la primera posición y el primer fitness hallado.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Ilustración 26 Comienzo del bucle luciérnagas

En esta última ilustración se comienza el bucle del algoritmo, y se calcula alpha que será necesario para la función de mover individuos.

Alpha = valor inicial del parámetro alpha + (valor final del parámetro alpha) \* exponcial(-numero iteracion).

Texto

Descripción generada automáticamenteEn esta ilustración se muestran dos bucles anidados que recorren cada individuo y por cada individuo se recorre de nuevo cada individuo en el algoritmo por lo que se compara cada individuo con todos y cada uno de los individuos. Se hace para que si el fitness actual de un individuo i es mayor el que el de un individuo j se mueve este individuo i hacia el individuo j ya que el individuo j es más atractivo, si no cumple esa condición se mueve aleatoriamente por el espacio.

Ilustración 27 Movimientos de luciernagas

Estas operaciones se realizan para mover cada luciérnaga a las que son más brillantes que ellas para realizar una explotación global.

Texto

Descripción generada automáticamenteEn esta ilustración se ve como se acota la posición al rango permitido y se transforma a lista (de igual manera que en el algoritmo anterior).

Ilustración 28 Cálculos Luciérnagas

Con estos pasos concluye el movimiento de luciérnagas por lo que se puede empezar a calcular el fitness actual (fitnessA) de manera igual que en el PSO, para cada individuo se llama a la función objetivo para obtener el fitness.

Ilustración 29 Actualización posición y fitness luciérnagas

Texto

Descripción generada automáticamente

En la ilustración 29 se muestra cómo se actualiza tanto la mejor solución particular de cada individuo y su mejor fitness.

Texto

Descripción generada automáticamenteTambién se actualiza la mejor solución global y el mejor fitness, por último, se imprime este valor.

Ilustración 30 Actualización mejor solución global luciérnagas

Captura de pantalla de computadora

Descripción generada automáticamente

En esta última ilustración se muestra cómo se mueven las luciérnagas siguiendo las fórmulas que están en los comentarios.

Ilustración 31 Función mover luciérnaga

Para el cálculo de la distancia (r) se utiliza la biblioteca numpy, en este caso se calcula la distancia euclidiana que es la distancia recta entre dos puntos en el espacio.

Esta función devuelve las nuevas posiciones de cada individuo.

### 5.2.3 Lobos

Ilustración 32 Inicialización Lobos

Captura de pantalla de computadora

Descripción generada automáticamente

Como en el resto de los algoritmos se empieza con el constructor del algoritmo e inicializando la población.

En este algoritmo los parámetros son:

* n: numero de individuos
* function: función del algoritmo
* lb: limite inferior del espacio de búsqueda
* ub: limite superior del espacio de búsqueda
* dimension: dimensión del espacio
* iteration: número de iteraciones

Después de inicializar intelligence y la población, se calculan los valores de fitness actual de cada individuo y se buscan los mejores lobos (alfa beta y delta ABD), alpha representa el individuo con mejor posición por lo cual se asigna Gbest (mejor posición global) con esta posición.

Texto

Descripción generada automáticamente

Ilustración 33 Get mejores lobos

Esta función retorna los tres mejores individuos de todos los individuos del algoritmo y el fitness del mejor individuo. Para ello primero inicializa result como una lista vacía, después se calcula el fitness de cada individuo y se guarda en una lista junto con su índice. Después se ordena esta lista en orden ascendente.

Texto

Descripción generada automáticamentePor último, se recorren los tres mejores individuos para añadirlos a result, se selecciona de la lista de individuos el individuo cuyo índice coincide con el índice almacenado en la lista fitness y se añade a la lista result el fitness del mejor individuo para recuperarlo facilmente.

Ilustración 34 Inicio bucle Lobos

En esta ilustración se muestra el inicio del bucle del algoritmo y el cálculo del parámetro a siguiendo esta ecuación:

Texto

Descripción generada automáticamente

Ilustración 35 Calculo de aleatorios, A y C

En esta ilustración se muestra el cálculo de números aleatorios para hallar A y C de cada individuo siguiendo las ecuaciones 1 y 2 de la página 31.

Texto

Descripción generada automáticamente

Ilustración 36 Cálculo de D

Se calculan las D de cada individuo alfa beta y delta siguiendo las ecuaciones de la página 33.

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

Ilustración 37 Cálculo de X para cada lobo

Texto

Descripción generada automáticamenteEn esta ilustración se muestra el cálculo de X para cada individuo siguiendo las ecuaciones de la página 33.

Ilustración 38 Final bucle lobos

Se ajustan las posiciones de los individuos siguiendo la ecuación de la página 33.

Se acotan las nuevas posiciones de cada individuo a los límites permitidos.

Llamando de nuevo a la función getABD() se obtienen los tres mejores individuos y el fitness del alfa.

Después se comprueba si el fitness actual del mejor individuo es mejor que el mejor fitness almacenado, en caso verdadero se actualiza la mejor posición global con la posición del individuo alfa y se actualiza el mejor fitness.

Por último, se guarda el mejor fitness y se imprime.

### 5.2.4 Abejas

Ilustración 39 Constructor abejas

Aquí se muestra el constructor de abejas, estos son sus parámetros:

* n: número de individuos
* function: función
* lb: límite inferior del espacio
* ub: límite superior del espacio
* dimension: dimensión del espacio
* Pantalla de computadora con letras

  Descripción generada automáticamente con confianza mediaiteration: número de iteraciones

Ilustración 40 inicialización abejas

En esta ilustración se realiza la inicialización del algoritmo, como en el resto se inicia intelligence, la población de individuos y el fitness.

Se busca la mejor solución cogiendo la posición del individuo cuyo índice es el índice del menor valor almacenado en la lista de los mejores fitness y se inicializa Gbest con el valor de Pbest.

Texto

Descripción generada automáticamente

Ilustración 41División de individuos

Se dividen los individuos en varios grupos, si el número de individuos es igual o menor a 10 se dividen en 4 grupos el primero que será aproximadamente la mitad y tres grupos de 1.

Si es mayor a 10 se dividen en 4, el primero es un 10% de los individuos, el segundo son 5 individuos, el tercero que será (número de individuos - individuos en el primer grupo \* individuos en el segundo grupo – individuos en el primer grupo) / 2, y un cuarto grupo de dos individuos.

Esta división se hace para que el algoritmo pueda explorar diferentes áreas del espacio de búsqueda de manera más eficiente. Los grupos grandes tienden a centrarse más en la explotación de soluciones mientras que los más pequeños se enfocan en la exploración.

Pantalla de computadora con letras

Descripción generada automáticamente con confianza mediaSe comienza el bucle del algoritmo calculando el fitness de cada individuo y almacenándolo en una lista.

Ilustración 42 Inicio del bucle abejas

Después se ordenan de menor a mayor y se almacenan sus índices en sorted\_indices gracias a argsort de Numphy.

Best\_indices representa los índices de los individuos con los mejores valores de fitness en la población. Estos son los agentes que se consideran las fuentes de alimento más ricas, y son explotados intensamente por las abejas.

Escala de tiempo

Descripción generada automáticamenteSelected\_indices representa los índices de otro conjunto de individuos en la población. Estas no son las mejores soluciones, pero siguen siendo importantes, y se consideran como fuentes de alimento adicionales que las abejas observadoras pueden explorar.

Ilustración 43 Nuevos individuos

Se generan nuevos individuos gracias a \_\_new() y se obtiene el número de individuos nuevos.

Texto

Descripción generada automáticamente

Ilustración 44 función para nuevos individuos

Texto

Descripción generada automáticamenteEsta función genera nuevos individuos para cada individuo en l moviéndose a posiciones vecinas.

Ilustración 45 función vecinos

Genera un vecino aleatorio para un individuo who ajustando su posición a los límites.

Texto

Descripción generada automáticamente

Ilustración 46 Comprobación abejas

Captura de pantalla de computadora

Descripción generada automáticamente con confianza mediaSe comprueba si n-m es positivo antes de generar nuevos individuos y se ajustan sus posiciones a los límites de búsqueda.

Ilustración 47 Fin bucle abejas

En esta ilustración se muestra el final del bucle del algoritmo en el que se ajustan las posiciones a los límites del espacio de búsqueda de los individuos.

Se actualiza el fitness actual aplicando la función objetivo a cada individuo.

Se actualiza la mejor solución personal de cada individuo.

Se actualiza la mejor solución global si se cumple la condición.

Por último, se guarda el valor del mejor fitness y se imprime.

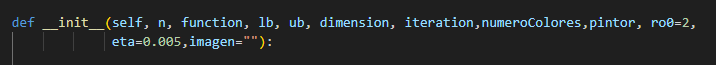
5.2.5 Ballenas

Ilustración 48 Constructor ballenas

En esta ilustración se muestra el constructor del algoritmo cuyos parámetros son:

* n: número de ballenas
* function: función a optimizar
* lb: límite inferior del espacio
* ub: límite superior del espacio
* dimension: dimensión del espacio
* iteration: número de iteraciones
* numeroColores: número de colores de la nueva imagen
* pintor: booleano que se usa para saber si pintamos imagen al final o no.
* ro0: intensidad de ultrasonido en la fuente de origen
* eta: probabilidad de distorsión de mensaje a largas distancias
* imagen: ruta de la imagen a procesar por el algoritmo

Texto

Descripción generada automáticamenteEn esta ilustración se muestra la inicialización del algoritmo de igual manera que el resto.

Ilustración 49 inicialización ballenas

Texto

Descripción generada automáticamenteSe inicializan los valores del fitness actual y del mejor fitness.

Ilustración 50 Inicio del bucle ballenas

Se inicia el bucle del algoritmo.

Una captura de pantalla de un celular

Descripción generada automáticamente con confianza mediaEn esta ilustración se muestra cómo se mueven cada uno de los individuos. Para cada individuo se busca un individuo mejor que él mismo y que sea más cercano.

Ilustración 51 movimiento de ballenas

Si se encuentra se mueve el individuo siguiendo esta ecuación:

Texto

Descripción generada automáticamente

Ilustración 52 función buscar mejor y más cercana ballena

Esta ilustración muestra cómo se busca un individuo mejor y más cercano.

Se recorren todos los individuos y para cada uno se compara el fitness del individuo desde el que se busca (u) y el fitness del individuo que toque en el bucle, si es menor se comprueba la distancia.

Se calcula la distancia entre individuos y si esta distancia es menor que la anterior que se encontró se almacena para comparar esta última con las siguientes iteraciones del bucle.

Al finalizar el bucle se devuelve el individuo con un mejor fitness que el del individuo que se evalúa y que sea más cercano.

Imagen de la pantalla de un video juego

Descripción generada automáticamente con confianza bajaEn esta ilustración se muestra cómo se halla la distancia entre individuos usando la librería de numphy.

Ilustración 53 calcular distancia entre individuos

Texto

Descripción generada automáticamentePor último, se mueven los individuos, se acotan las posiciones al espacio de solución, se actualiza el fitness actual de cada uno, se actualiza la mejor solución de cada individuo comparando el fitness actual con el mejor fitness encontrado, si el actual es mejor que el mejor almacenado se actualiza la mejor posición particular y su mejor fitness.

Ilustración 54 Fin bucle ballenas

Se actualiza la mejor solución global y se guarda el mejor fitness para imprimirlo.

## 5.3 Funciones

En este apartado se documentan las funciones que se han utilizado en cada uno de los algoritmos.

### Texto Descripción generada automáticamente5.3.1 pintaImagen

Ilustración 55 pintaImagen

Esta última ilustración muestra como el programa pinta las imágenes resultantes del algoritmo.

Primero se obtiene la ruta del directorio actual y se sube un nivel para estar en el directorio raíz, esto es debido a que este archivo está en un nivel diferente al de la carpeta que contiene las imágenes.

Luego se accede a la carpeta de imágenes.

Se forma el nombre a la imagen resultante que se va a almacenar en formato ALGORITMO\_NUMCOLORES\_IMAGEN.

Se guarda la imagen cuantizada, se hace una comprobación para en caso de que no se haya podido guardar de error y no intente leer la imagen.

Después se leen las dos imágenes, la original y la cuantizada.

La comprobación de si es un jpg la redimensiona es porque al hacer las pruebas y pasarle una imagen personal hecha con un teléfono, el programa me pintaba las imágenes en un formato muy grande, se salía de la pantalla.

Si se el parámetro de pintar la imagen se ha pasado a true pinta las dos imágenes, después el programa espera a que se pulse cualquier tecla, al pulsarla se destruyen las ventanas y termina.

Este método se llama al final de cada algoritmo.

### Texto Descripción generada automáticamente5.3.2 preparaImagen

Ilustración 56 preparaImagen

Esta ilustración muestra el método que se llama para preparar las imágenes para operar con ellas, se encarga de leer la imagen, redimensionarla si es jpg y redimensionar la imagen a una matriz 2D de pixeles y luego se devuelve la matriz junto con la imagen.

### Texto Descripción generada automáticamente5.3.3 generaCuantizada

Ilustración 57 generaCuantizada

Esta ilustración muestra el método encargado de generar una imagen cuantizada.

Primero llama a preparaImagen para preparar la imagen para su uso.

Se eliminan los pixeles duplicados, esa línea es debida a que a raíz de actualizar la librería sklearn se experimentaron errores que se corrigieron con esto.

Se ajustan el número de clústeres al número de pixeles únicos si es menor que tam\_paleta que es el tamaño de la paleta que se pide o el número de colores.

Se aplica el Kmeans de la librería sklearn a los pixeles únicos para encontrar los clústeres.

Se guardan los valores de las etiquetas de cada pixel y se obtienen los centroides para luego reemplazar cada pixel en la imagen original por el color del clúster al que pertenece.

Por último, se devuelve la imagen cuantizada.

### 5.3.4 getMse

Pantalla de computadora con letras

Descripción generada automáticamente con confianza media

Ilustración 58 función MSE

En esta última ilustración se muestra cómo se calcula el mse, se prepara la imagen, se genera la cuantizada y se llama a mean\_mean\_squared\_error de la librería sklearn para calcular el error cuadrático medio

### Captura de pantalla con la imagen de una pantalla Descripción generada automáticamente5.3.5 getMae

Ilustración 59 getMae

En esta ilustración se muestra cómo se calcula el mae, se prepara la imagen, se genera la imagen cuantizada, se redimensiona y se calcula el mae gracias a numphy. Devuelve el error absoluto medio.

### 5.3.6 getSsim

Texto

Descripción generada automáticamente

Ilustración 60 getSsim

En esta ilustración se muestra cómo se calcula el ssim, igual que en las anteriores se prepara la imagen y se genera la imagen cuantizada, luego se redimensiona la imagen a una matriz de dos dimensiones para luego calcular el ssim gracias a la librería skimage

Como esta librería devuelve el índice de similitud en el que 0 no se parecen y 1 si se parecen, invierto el valor al restar 1 menos lo que devuelve ssim para que se ajuste a los algoritmos.

### 5.3.7 getMssim

Texto

Descripción generada automáticamente

Ilustración 61 getMssim

Esta función calcula el mssim de la misma manera que el ssim, la diferencia son los parámetros que se mandan a ssim lo que en vez de devolver el índice de similitud estructural (ssim) devuelve el índice de similitud multi escalar (mssim).

Como en la función anterior se invierte el índice para ajustarse a los algoritmos.

# 8. Trabajo futuro

Existen varias áreas en las que este trabajo podría expandirse y mejorar, algunos de ellos son:

* **Optimización de Algoritmos:** Aunque los algoritmos de enjambre como PSO, GWO, y ABA han sido aplicados exitosamente, se podrían explorar versiones avanzadas de estos algoritmos, como versiones híbridas o adaptativas que ajusten sus parámetros dinámicamente durante la ejecución. Esto podría mejorar la convergencia y reducir el tiempo de procesamiento.
* **Ampliación de la Biblioteca de Algoritmos:** Actualmente, la herramienta incluye un conjunto de algoritmos específicos. Una dirección interesante sería la incorporación de nuevos algoritmos de optimización basados en la naturaleza, como el algoritmo de murciélagos, el algoritmo de colonias de hormigas o variantes de redes neuronales evolutivas, ampliando las opciones para la cuantificación de color.
* **Paralelización y Uso de GPU:** A medida que las imágenes procesadas son de mayor tamaño o mayor resolución, los tiempos de cómputo aumentan considerablemente. La paralelización del código o el uso de procesamiento en GPU podría reducir de manera significativa el tiempo de ejecución de los algoritmos, haciéndolos más eficientes en escenarios de grandes volúmenes de datos.
* **Evaluación de Nuevas Métricas:** Las métricas de calidad utilizadas, como el MSE, MAE, SSIM y MS-SSIM, ofrecen una visión del rendimiento, pero podrían incluirse nuevas métricas que tomen en cuenta aspectos específicos de la percepción humana del color. Métricas más sofisticadas que evalúen la fidelidad del color y las características visuales serían un buen complemento.

# 9. Conclusiones

El desarrollo de esta herramienta para la aplicación de algoritmos de enjambre en la cuantificación de color ha demostrado la efectividad y versatilidad de estas técnicas bioinspiradas en el ámbito del procesamiento de imágenes. A lo largo del proyecto, se han implementado y probado diversos algoritmos de optimización como PSO (Optimización por Enjambre de Partículas), GWO (Optimización por Lobos Grises), FA (Luciérnagas), y ABA (Abejas Artificiales). Estos métodos han mostrado su capacidad para reducir el número de colores en una imagen manteniendo un alto nivel de calidad visual, lo que los convierte en herramientas útiles para aplicaciones de compresión de imágenes, procesamiento gráfico y diseño visual.

La principal ventaja de los algoritmos de enjambre reside en su capacidad para explorar eficientemente el espacio de búsqueda y encontrar soluciones óptimas en problemas complejos, como la cuantificación de color. Además, estos algoritmos son altamente flexibles, permitiendo su ajuste y personalización para adaptarse a diferentes escenarios y requisitos de precisión.

En resumen, este trabajo ha evidenciado el potencial de los algoritmos de enjambre en el ámbito del procesamiento de imágenes, y la herramienta desarrollada servirá como una base sólida para futuras investigaciones y aplicaciones en este campo.

# 10.Bibliografía

[1] «Los modos de color de la imagen digital». Accedido: 28 de enero de 2024. [En línea]. Disponible en: https://www.fotonostra.com/fotografia/modoscolor.htm

[2] «Inteligencia de enjambre e inteligencia artificial - Fundación Aquae». Accedido: 5 de febrero de 2024. [En línea]. Disponible en: https://www.fundacionaquae.org/la-inteligencia-enjambre-y-la-inteligencia-artificial/

[3] «PSO: Optimización por enjambres de partículas». Accedido: 21 de abril de 2024. [En línea]. Disponible en: https://www.cs.us.es/~fsancho/Blog/posts/PSO.md

[4] «Algoritmo firefly - Wikipedia, la enciclopedia libre». Accedido: 5 de mayo de 2024. [En línea]. Disponible en: https://es.wikipedia.org/wiki/Algoritmo\_firefly

[5] S. Mirjalili, S. M. Mirjalili, y A. Lewis, «Grey Wolf Optimizer», *Advances in Engineering Software*, vol. 69, pp. 46-61, mar. 2014, doi: 10.1016/J.ADVENGSOFT.2013.12.007.

[6] «GWO». Accedido: 20 de mayo de 2024. [En línea]. Disponible en: https://seyedalimirjalili.com/gwo

[7] H. Faris, I. Aljarah, M. A. Al-Betar, y S. Mirjalili, «Grey wolf optimizer: a review of recent variants and applications», *Neural Comput Appl*, vol. 30, n.o 2, pp. 413-435, jul. 2018, doi: 10.1007/S00521-017-3272-5/TABLES/4.

[8] J. Nasiri y F. M. Khiyabani, «A whale optimization algorithm (WOA) approach for clustering», *Cogent Math Stat*, vol. 5, n.o 1, p. 1483565, ene. 2018, doi: 10.1080/25742558.2018.1483565.

[9] «AN IDEA BASED ON HONEY BEE SWARM FOR NUMERICAL OPTIMIZATION».

[10] «Algoritmo colonia de abejas artificiales - Wikipedia, la enciclopedia libre». Accedido: 4 de agosto de 2024. [En línea]. Disponible en: https://es.wikipedia.org/wiki/Algoritmo\_colonia\_de\_abejas\_artificiales

[11] *Why was Python created in the first place?* General Python FAQ. Accedido: 4 de septiembre de 2024. [En línea]. Disponible en: http://docs.python.org/faq/general#why-was-python-created-in-the-first-place

[12] «artima - The Making of Python», *www.artima.com*, Accedido: 4 de septiembre de 2024. [En línea]. Disponible en: https://www.artima.com/articles/the-making-of-python

[13] «La librería Numpy | Aprende con Alf». Accedido: 4 de septiembre de 2024. [En línea]. Disponible en: https://aprendeconalf.es/docencia/python/manual/numpy/

[14] «Procesamiento de imágenes con OpenCV en Python». Accedido: 4 de septiembre de 2024. [En línea]. Disponible en: https://imaginaformacion.com/tutoriales/opencv-en-python

[15] «Scikit-Learn, herramienta básica para el Data Science en Python». Accedido: 4 de septiembre de 2024. [En línea]. Disponible en: https://www.master-data-scientist.com/scikit-learn-data-science/

[16] S. Van Der Walt *et al.*, «scikit-image: image processing in Python», *PeerJ*, vol. 2:e453, n.o 1, p. e453, jul. 2014, doi: 10.7717/peerj.453.