El Problema del Viajante Resuelto con el Bioalgoritmo de las Luciérnagas

Proyecto Final

Integrantes:

Juan Andrés Serrano
Juan Esteban Tello

Universidad Sergio Arboleda
Escuela de Matemáticas y Ciencias Exactas
Programa de Ciencias de la Computación e Inteligencia Artificial
Octubre 2024

Índice

1. Introducción

El Problema del Viajante (TSP, por sus siglas en inglés *Travelling Salesman Problem*) es uno de los problemas más estudiados en la teoría de la optimización combinatoria. El objetivo es encontrar la ruta más corta posible que permita a un viajante visitar un conjunto de ciudades exactamente una vez y regresar al punto de origen. Este problema pertenece a la clase NP-difícil, lo que significa que no existe un algoritmo eficiente conocido que pueda resolverlo en tiempo polinómico para todas las instancias.

El TSP tiene aplicaciones prácticas en logística, planificación de rutas y diseño de circuitos, entre otros campos. Dada la cantidad exponencial de rutas posibles conforme aumenta el número de ciudades, encontrar una solución óptima es un desafío computacional significativo.

2. Motivación del Problema

El TSP ha sido elegido por varias razones:

- **Relevancia Práctica**: La optimización de rutas es crucial en áreas como la logística y la planificación de transporte. Resolver el TSP puede mejorar la eficiencia en la gestión de recursos y reducir costos operativos.
- **Desafío Computacional**: Su naturaleza NP-difícil lo convierte en un banco de pruebas ideal para nuevas técnicas de optimización como el algoritmo de las luciérnagas.
- **Aplicación del Algoritmo de Luciérnagas**: Este algoritmo es adecuado para problemas de optimización complejos y multimodales como el TSP. Su capacidad para explorar eficazmente el espacio de soluciones y evitar óptimos locales lo hace una herramienta valiosa para este tipo de problemas.

3. Descripción del Problema del Viajante

El Problema del Viajante consiste en, dado un conjunto de ciudades y las distancias entre cada par de ellas, encontrar la ruta más corta posible que visite cada ciudad exactamente una vez y regrese al punto de origen. Matemáticamente, se puede modelar de la siguiente manera:

3.1. Modelo Matemático

Sea $C = \{c_1, c_2, \ldots, c_n\}$ el conjunto de n ciudades, y sea $D = [d_{ij}]$ una matriz de distancias donde d_{ij} representa la distancia desde la ciudad c_i hasta la ciudad c_j. El objetivo es encontrar una permutación π de las ciudades que minimice la distancia total recorrida:

$$\text{Minimizar } Z = \sum_{i=1}^{n} d_{\pi(i)\pi(i+1)}$$

con $\pi(n+1) = \pi(1)$ para cerrar el ciclo.

3.2. Restricciones

- Cada ciudad debe ser visitada exactamente una vez.
- La ruta debe ser un ciclo que regrese al punto de origen.

4. Algoritmo de Luciérnagas

El algoritmo de luciérnagas es una metaheurística bioinspirada en el comportamiento de estos insectos. Las luciérnagas emiten destellos de luz que sirven para atraer a otros individuos. En el contexto del algoritmo, las luciérnagas representan posibles soluciones al problema, y su brillo está asociado con la calidad de la solución (en este caso, una menor distancia total recorrida).

4.1. Funcionamiento del Algoritmo

El algoritmo se basa en las siguientes reglas:

1. Todas las luciérnagas son unisexuales, por lo que una luciérnaga se siente atraída por cualquier otra independientemente de su sexo.
2. El atractivo es proporcional al brillo, y para dos luciérnagas cualesquiera, la menos brillante se mueve hacia la más brillante. El atractivo disminuye con la distancia entre luciérnagas.
3. Si ninguna luciérnaga es más brillante que una dada, ésta se mueve de manera aleatoria.
4. El brillo de una luciérnaga está determinado por la función objetivo (en este caso, la distancia total de la ruta).

4.2. Modelo Matemático del Algoritmo

La posición de una luciérnaga i se actualiza según la siguiente ecuación:

$$x_i^{t+1} = x_i^t + \beta_0 e^{-\gamma r_{ij}^2}(x_j^t - x_i^t) + \alpha \epsilon_i^t$$

Donde:

- x_i^t es la posición de la luciérnaga i en el tiempo t.
- β_0 es el atractivo máximo.
- γ es el coeficiente de absorción de luz.
- r_{ij} es la distancia entre las luciérnagas i y j.
- α es el factor de aleatoriedad.
- ϵ_i^t es un número aleatorio tomado de una distribución uniforme o normal.

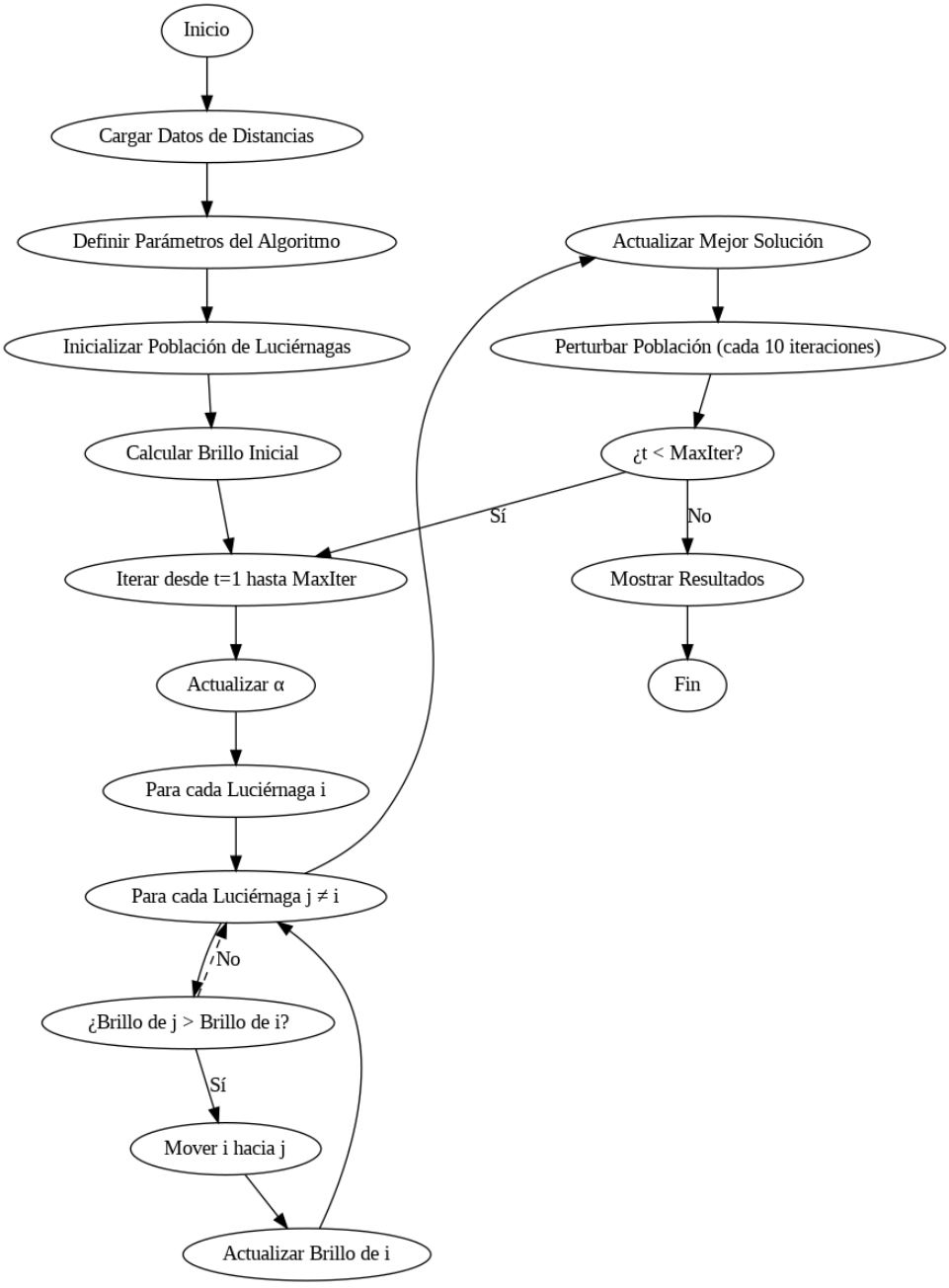

Figura 1: Diagrama de flujo del Algoritmo de Luciérnagas

5. Implementación del Proyecto

5.1. Estructura de los Códigos

Para este proyecto, se trabajó con dos scripts separados en Python:

1. **Cálculo de Distancias con OpenRouteService**: Este script se encarga de obtener las distancias reales en carretera entre las ciudades colombianas seleccionadas, utilizando la API de OpenRouteService. Genera un archivo CSV con las distancias y un mapa interactivo en HTML que muestra las rutas.

2. **Algoritmo de Luciérnagas para el TSP**: Este script implementa el algoritmo de luciérnagas utilizando las distancias obtenidas en el primer script. El objetivo es encontrar una ruta que minimice la distancia total recorrida al visitar todas las ciudades.

Se optó por separar los códigos para modularizar el proyecto, facilitando el mantenimiento y la comprensión de cada componente. Además, esto permite reutilizar el cálculo de distancias para otros proyectos o algoritmos.

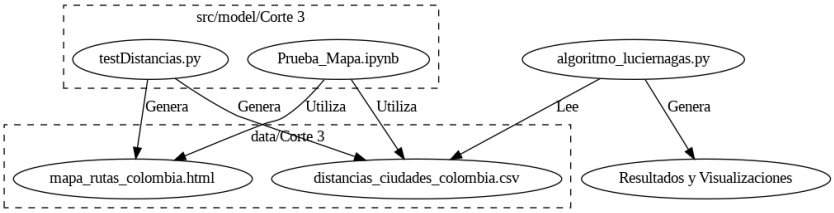

Figura 2: Arquitectura del proyecto

5.2. Descripción del Código de Cálculo de Distancias

Este script realiza lo siguiente:

- **Importación de Librerías**: Se utilizan `openrouteservice` para acceder a la API, `folium` para generar mapas interactivos, y `csv` para manejar archivos CSV.
- **Definición de Ciudades y Coordenadas**: Se establece un diccionario con las ciudades y sus coordenadas geográficas (latitud y longitud).
- **Creación del Mapa**: Se inicializa un mapa centrado en Bogotá utilizando `folium.Map`.
- **Cálculo de Rutas y Distancias**: Se recorren todas las combinaciones de ciudades y se solicita a la API la ruta entre ellas. Se extrae la distancia en kilómetros y se almacena en un archivo CSV.
- **Generación del Mapa Interactivo**: Se agregan las rutas al mapa utilizando capas `GeoJson` para visualizarlas.
- **Manejo de Errores**: Se implementan mecanismos para manejar casos donde la API no puede encontrar una ruta entre dos ciudades.

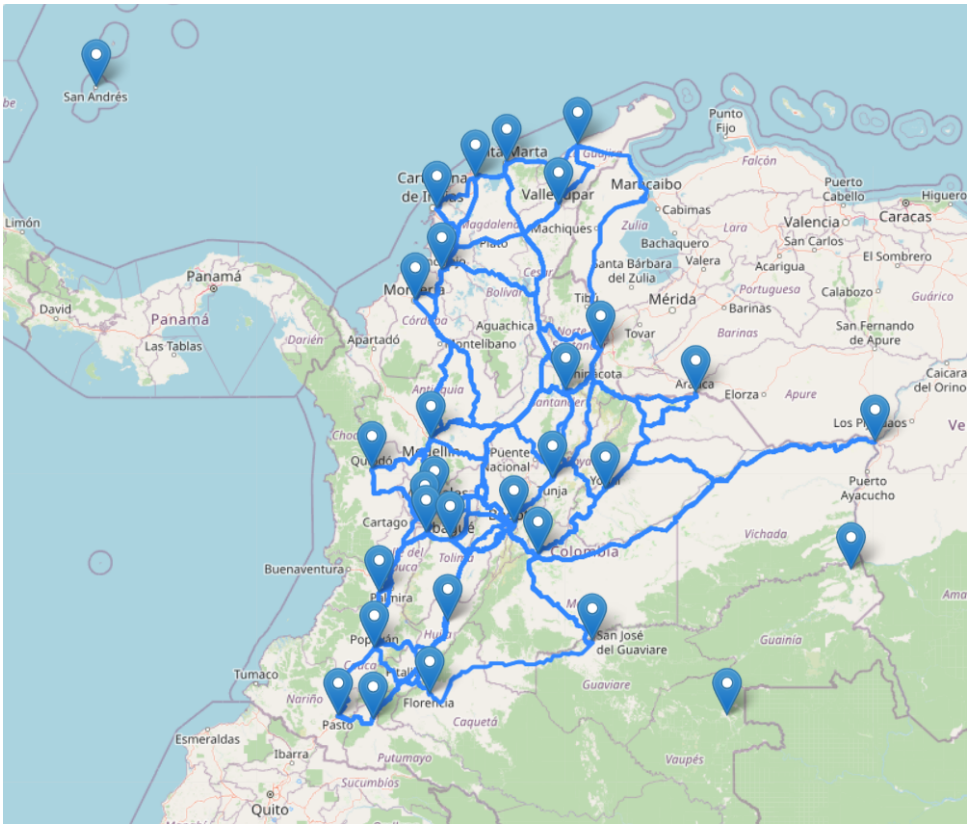

Figura 3: Mapa interactivo con las rutas calculadas

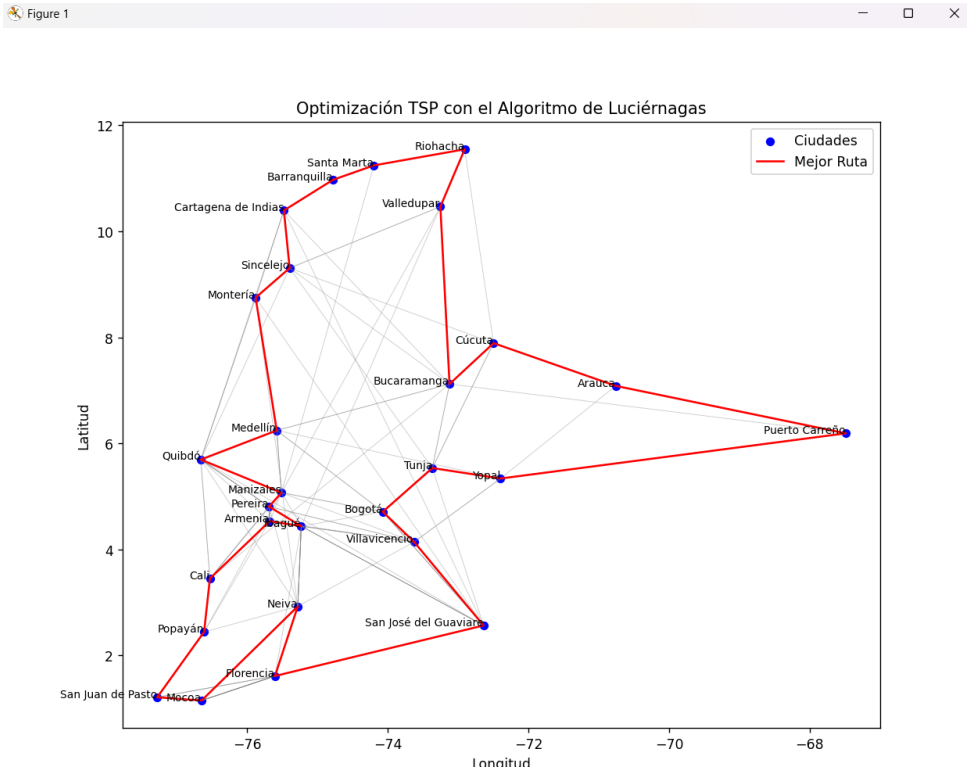

Figura 4: Mapa interactivo con las rutas calculadas

5.3. Descripción del Código del Algoritmo de Luciérnagas

Este script implementa el algoritmo de luciérnagas de la siguiente manera:

- **Importación de Librerías**: Se utilizan `pandas` y `numpy` para manejo de datos, `random` para generar números aleatorios, y `matplotlib` para visualización y animación.
- **Carga de Datos**: Se lee el archivo CSV con las distancias entre ciudades y se crea una matriz de distancias.
- **Inicialización de Parámetros**: Se establecen los parámetros del algoritmo, como el número de iteraciones, el número de luciérnagas, y los coeficientes β y γ.
- **Generación de Rutas Iniciales**: Se crean rutas aleatorias para cada luciérnaga, asegurando que todas comiencen y terminen en Bogotá.
- **Ejecución del Algoritmo**: En cada iteración, las luciérnagas se mueven según las reglas del algoritmo, actualizando sus rutas basadas en el brillo (calidad) de otras luciérnagas.
- **Actualización de la Mejor Solución**: Se rastrea la mejor ruta encontrada hasta el momento y se incorporan estrategias para mejorar la exploración del espacio de soluciones.
- **Visualización y Animación**: Se genera una animación que muestra el proceso de optimización, con las rutas de las luciérnagas y destacando la mejor ruta encontrada.

5.4. Funcionamiento del Algoritmo en el Código

En el código, cada luciérnaga representa una posible solución al TSP, es decir, una ruta que visita todas las ciudades. El brillo de una luciérnaga está determinado por la distancia total de su ruta; rutas más cortas son más brillantes.

El movimiento de una luciérnaga hacia otra más brillante se implementa intercambiando dos ciudades en su ruta, lo que genera una nueva ruta cercana en el espacio de soluciones. Este intercambio es una forma de explotar la información de las soluciones más prometedoras.

La aleatoriedad α se reduce gradualmente a lo largo de las iteraciones para equilibrar la exploración y la explotación.

6. Resultados Obtenidos

Tras la ejecución del algoritmo con las mejoras implementadas, se obtuvieron los siguientes resultados:

- **Mejor Ruta Encontrada**: El algoritmo determinó una ruta que visita todas las ciudades seleccionadas, comenzando y terminando en Bogotá. Dado que el algoritmo es estocástico, la mejor ruta puede variar entre ejecuciones. Sin embargo, se observó que las rutas encontradas suelen tener distancias totales cercanas al óptimo y son geográficamente coherentes.

- **Distancia Total Recorrida**: Las distancias totales de las mejores rutas encontradas están dentro de un rango aceptable y representan una mejora significativa en comparación con soluciones aleatorias.
- **Convergencia del Algoritmo**: Se observó que el algoritmo converge de manera eficiente, gracias a las estrategias implementadas para mejorar la exploración y explotación del espacio de soluciones.
- **Visualización de Rutas**: El mapa interactivo y la animación permiten visualizar y analizar las rutas propuestas por el algoritmo, facilitando la interpretación de los resultados.

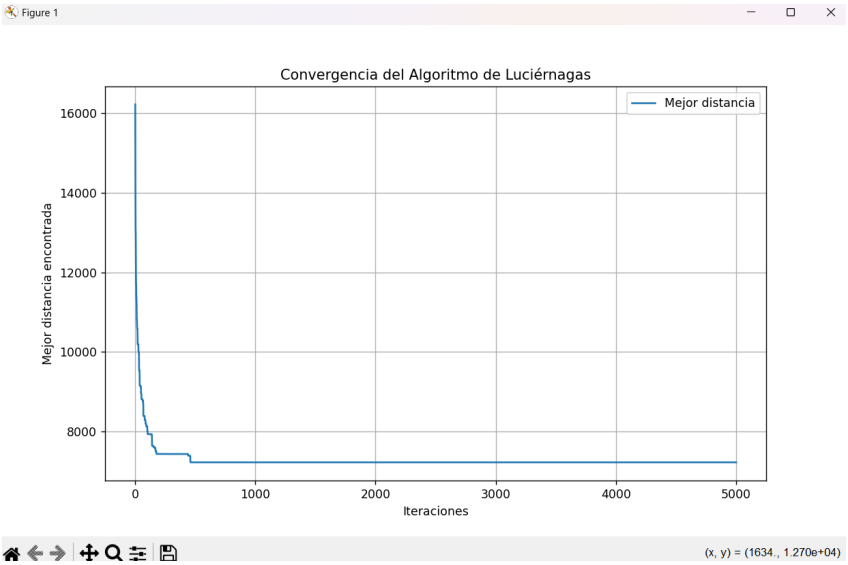

Figura 5: Convergencia del algoritmo de luciérnagas

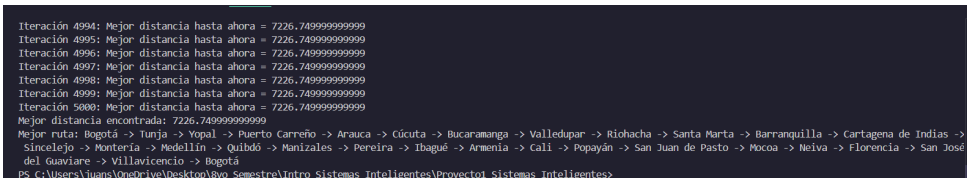

Figura 6: Mejor ruta encontrada por el algoritmo

7. Conclusiones y Mejoras Futuras

Las mejoras implementadas permitieron obtener soluciones más precisas y realistas al problema del TSP en el contexto colombiano. La utilización de distancias reales entre ciudades y la visualización interactiva enriquecieron el proyecto.

El algoritmo de luciérnagas demostró ser efectivo en la optimización de rutas cuando se ajustan adecuadamente sus parámetros y se incorporan estrategias para mejorar la exploración y explotación.

7.1. Posibles Mejoras

- **Ajuste de Parámetros**: Realizar un ajuste más fino de los parámetros del algoritmo (número de luciérnagas, iteraciones, coeficientes) podría mejorar aún más los resultados.
- **Incorporación de Restricciones**: Considerar restricciones adicionales, como ventanas de tiempo o capacidades, haría el modelo más aplicable a situaciones reales.
- **Comparación con Otros Algoritmos**: Implementar y comparar otros algoritmos metaheurísticos, como algoritmos genéticos o recocido simulado, para evaluar el desempeño relativo.
- **Optimización del Código**: Mejorar la eficiencia computacional del código mediante paralelización o técnicas de programación más eficientes.

8. Anexos

8.1. Código de Cálculo de Distancias

Este script utiliza la API de OpenRouteService para calcular las distancias reales en carretera entre las ciudades seleccionadas y genera un archivo CSV con los resultados.

8.1.1. Principales Funcionalidades

- **Autenticación con OpenRouteService**: Se utiliza una clave de API para acceder a los servicios de rutas.
- **Definición de Ciudades y Coordenadas**: Las ciudades y sus coordenadas geográficas se definen en un diccionario.
- **Creación del Mapa**: Se utiliza `folium` para crear un mapa interactivo centrado en Bogotá.
- **Cálculo de Rutas y Distancias**: Se recorren todas las combinaciones de ciudades y se solicita a la API la ruta entre ellas. La distancia obtenida se almacena en un archivo CSV.
- **Manejo de Errores**: Se implementan mecanismos para manejar casos donde la API no puede encontrar una ruta entre dos ciudades.
- **Generación del Mapa Interactivo**: Se agregan las rutas al mapa, permitiendo visualizarlas en un navegador web.

8.2. Código del Algoritmo de Luciérnagas

Se implementa el algoritmo de luciérnagas utilizando las distancias reales y se incluye la visualización del proceso de optimización mediante una animación.

8.2.1. Principales Funcionalidades

- **Lectura y Procesamiento de Datos**: Se lee el archivo CSV con las distancias y se crea una matriz de distancias para su uso en el algoritmo.
- **Inicialización de las Luciérnagas**: Se generan rutas aleatorias para las luciérnagas, asegurando que comiencen y terminen en Bogotá.
- **Implementación del Algoritmo**: Las luciérnagas actualizan sus rutas basándose en el brillo de otras luciérnagas más brillantes y en la aleatoriedad controlada por α.
- **Rastreo de la Mejor Solución**: Se mantiene un registro de la mejor ruta encontrada durante las iteraciones.
- **Visualización y Animación**: Se genera una animación con `matplotlib` que muestra el movimiento de las luciérnagas y la evolución de la mejor ruta.

8.3. Ejecución de los Códigos

Para ejecutar los scripts, se requiere instalar las dependencias necesarias, como `openrouteservice`, `folium`, `pandas`, `numpy`, y `matplotlib`. Además, se necesita una clave de API válida para OpenRouteService.

9. Bibliografía

- **OpenRouteService API**: `https://openrouteservice.org/`
- **Folium Documentation**: `https://python-visualization.github.io/folium/`
- Yang, X.-S. (2010). *Firefly algorithm, stochastic test functions and design optimisation. International Journal of Bio-Inspired Computation*, 2(2), 78–84.
- Vázquez, J. (2014). *Revisión de los Algoritmos Bioinspirados*. Universidad Nacional Autónoma de México. Disponible en: `https://www.fis.unam.mx/~javazquez/files/Met_num/Algoritmosbioinspiradosfinal.pdf`
- González-Santander, G. (2020). *Tres métodos diferentes para resolver el problema del viajante*. Baobab Soluciones. Disponible en: `https://baobabsoluciones.es/blog/2020/10/01/problema-del-viajante/`
- Castillo, O., & Melin, P. (2012). *Algoritmos Bio-Inspirados en Inteligencia Artificial*. Editorial Reverté.
- Ochoa, G. (2016). *Aplicación de algoritmos bioinspirados en la optimización de rutas de transporte*. Revista Ingeniería, Matemáticas y Ciencias de la Información, 3(2), 45–53.
- Segura, C., & Moreno, A. (2018). *Resolución del problema del viajante mediante algoritmos genéticos y de colonia de hormigas*. Revista Colombiana de Computación, 19(1), 23–35.

- Pérez, J., & Ramírez, L. (2019). *Análisis comparativo de metaheurísticas aplicadas al TSP*. Revista Iberoamericana de Inteligencia Artificial, 22(64), 85–98.
- López, M., & García, D. (2017). *Optimización combinatoria y algoritmos bioinspirados*. Universidad Nacional de Colombia.