Ciencias de la computación II

Universidad Distrital Francisco José de Caldas


UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

# Inteligencia y sinergia de enjambre: colonia de hormigas para el problema del viajante

Andrés Felipe Vanegas Bogotá

Sergio Andres Sanabria Castillo

Carlos Andres Sinuco Murcia

Profesor:

Rafael Antonio Acosta Rodriguez

2025

# Introducción:

El objetivo de este taller es abordar el Problema del Viajante (TSP, por sus siglas en inglés) utilizando el algoritmo de Optimización por Colonia de Hormigas (ACO). El TSP es un problema fundamental en la optimización combinatoria que consiste en encontrar la ruta más corta que visite cada ciudad exactamente una vez y regrese al punto de partida. Este problema tiene aplicaciones en diversas áreas, como logística, planificación de rutas y optimización de recursos.

Como ingeniero de inteligencia artificial en una empresa de logística, nuestro objetivo es diseñar la ruta más eficiente para la entrega de productos en múltiples ciudades). El algoritmo ACO, inspirado en el comportamiento de búsqueda de alimento de las hormigas, ofrece una estrategia prometedora para abordar eficazmente los desafíos que plantea el TSP.

**Librerías utilizadas**

En este proyecto, se utilizaron las siguientes librerías de Python:

1. **numpy:**
    - **Descripción:** numpy es una librería útil para la computación científica, proporciona soporte para arrays y matrices multidimensionales, junto con una gran colección de funciones matemáticas para operar sobre estas estructuras).
    - **Uso en el proyecto:** Se utiliza para manejar operaciones matriciales, como la creación de la matriz de distancias entre ciudades y la matriz de feromonas).

2. **matplotlib.pyplot:**
    - **Descripción:** es una librería para la generación de gráficos en Python. pyplot es un módulo de matplotlib que proporciona una interfaz similar a MATLAB para crear visualizaciones).
    - **Uso en el proyecto:** Se utiliza para graficar la ruta óptima encontrada por el algoritmo ACO en un espacio 3D.

3. **mpl_toolkits.mplot3d:**
    - **Descripción:** Este módulo de matplotlib permite la creación de gráficos en tres dimensiones (3D).
    - **Uso en el proyecto:** Se utiliza para visualizar la ruta óptima en 3D, incluyendo la latitud, longitud y altitud de las ciudades.

# Representación del problema:

El diagrama de flujo proporciona una ilustración detallada de la metodología empleada por el algoritmo de Optimización por Colonia de Hormigas (ACO) para abordar el desafío de optimizar las rutas de entrega de productos. Al aprovechar estratégicamente la inteligencia colectiva inspirada en el comportamiento de búsqueda de alimento de las hormigas, ACO busca identificar el camino más eficiente para la entrega de mercancías, minimizando las distancias de viaje y el tiempo.

Este proceso implica la exploración iterativa de diversas rutas, guiada por la interacción entre hormigas artificiales y rastros de feromonas. A través de iteraciones y refinamientos continuos, ACO converge progresivamente hacia una solución óptima, garantizando la entrega puntual y rentable de los productos a sus destinos previstos.

# Paso a paso:

1. Inicializa una lista vacía para almacenar las coordenadas de las ciudades.
2. Utiliza un bucle for para iterar number_cities veces.
3. Genera coordenadas aleatorias para cada ciudad usando np.random.rand(3).
4. Asigna las coordenadas generadas a la variable city.
5. Agrega la ciudad generada a la lista usando append().
6. Devuelve la lista de ciudades una vez que se hayan generado las coordenadas para todas las ciudades
7. Además, define una función llamada calculate_distance:
8. La función recibe dos arreglos de NumPy, point_1 y point_2, que representan las coordenadas de dos puntos en un espacio multidimensional.
9. Calcula la distancia euclidiana entre estos dos puntos.
10. La función devuelve la distancia calculada como un valor de tipo float.
11. La distancia euclidiana se calcula utilizando la función np.linalg.norm(), que computa la norma euclidiana (magnitud) del vector que representa la diferencia entre point_1 y point_2.

12. number_cities = len(cities): Determina el número de ciudades almacenadas en la lista cities y lo asigna a la variable number_cities. Esto es fundamental para definir el tamaño de la matriz de feromonas en la siguiente línea.
13. pheromone = np.ones((number_cities, number_cities)): Crea una matriz inicial de feromonas llena de unos (según la recomendación del profesor). La matriz tiene dimensiones de number_cities por number_cities, representando cada posible conexión entre ciudades. Cada entrada en la matriz indica la cantidad de feromonas

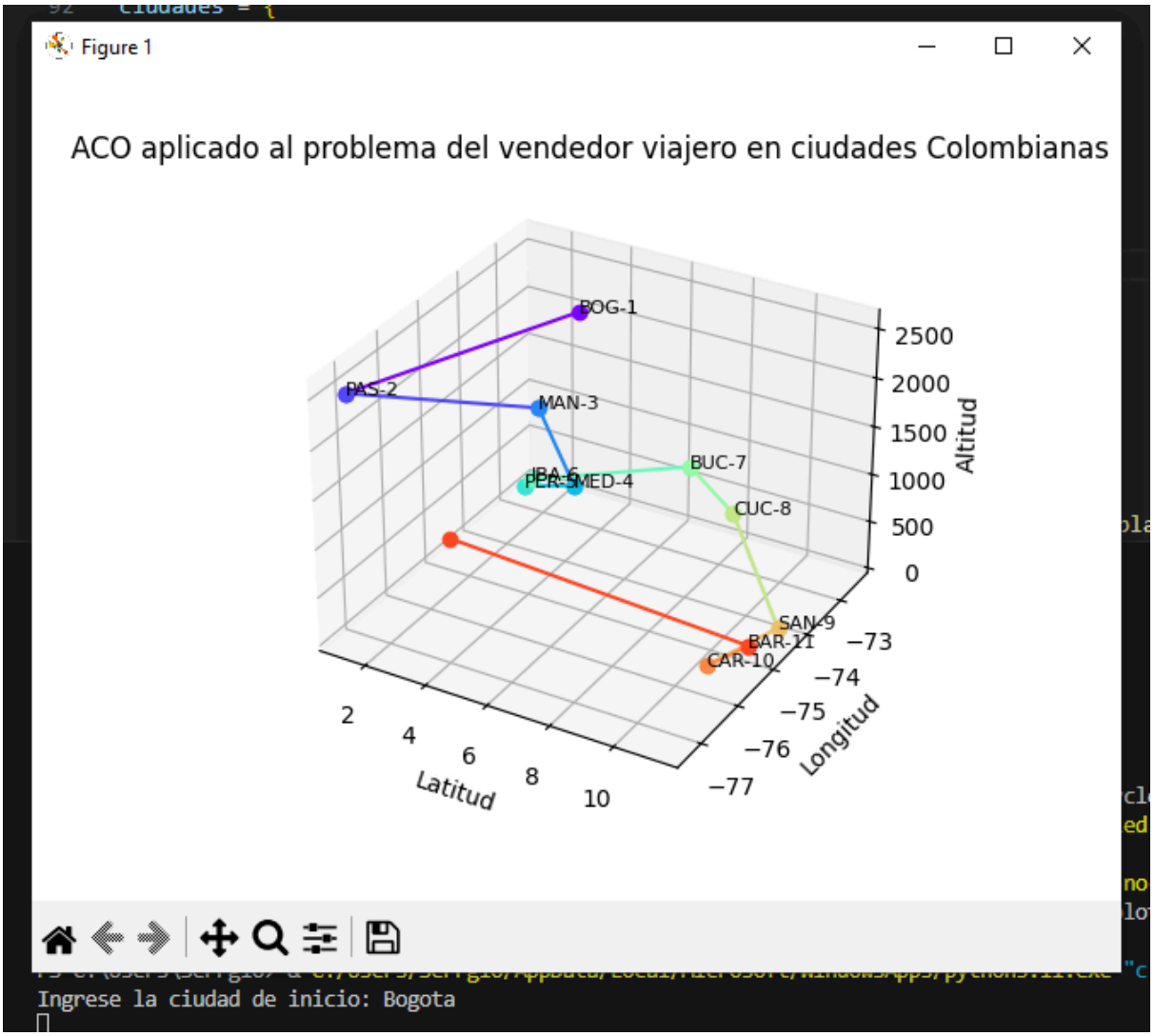
presente en la conexión entre dos ciudades. Inicializar con unos garantiza que todas las conexiones comiencen con la misma cantidad inicial de feromonas

14. La fórmula utilizada para calcular la probabilidad de moverse a una ciudad, basada en los niveles de feromonas, la distancia, alpha y beta, está lista para su implementación dentro del bucle for. Según lo especificado por el instructor, la fórmula incorpora los siguientes pasos:
    - Utilizar la matriz de feromonas con los índices i y j.
    - Multiplicar por alpha.
    - Incorporar la distancia entre ciudades obtenida de la función calculate_distance.
    - Multiplicar el resultado por beta
15. La normalización se calcula mediante probabilities /= probabilities.sum(), asegurando que la suma de las probabilidades sea 1.
16. En esta sección, he optado por proporcionar comentarios más detallados sobre los parámetros, generar la lista de ciudades y ejecutar el algoritmo de Optimización por Colonia de Hormigas (ACO). El algoritmo recibe como entrada el conjunto de ciudades junto con varios parámetros que regulan su comportamiento.

    La función devuelve dos valores:

    - **best_path**: Representa la ruta óptima descubierta por el algoritmo. Consiste en una secuencia ordenada de ciudades que indica el camino más corto encontrado por las hormigas.
    - **best_path_length**: Representa la distancia total del mejor camino determinado. Refleja la distancia total recorrida por las hormigas en la solución más óptima encontrada.

# Ejemplo de salida:




# Análisis y Conclusiones:

Basándonos en los resultados obtenidos tras ejecutar el algoritmo varias veces y modificar los parámetros, podemos deducir que:

- A medida que aumenta el número de ciudades, la longitud total del camino también tiende a aumentar, lo que indica que el problema se vuelve más complejo y la optimización se vuelve más difícil con un mayor número de ciudades.
- Estos resultados sugieren que el número de hormigas también influye en la longitud total del camino más corto encontrado por el algoritmo ACO. En este caso, un aumento en el número de hormigas parece llevar a una disminución en la longitud total del camino más corto hasta cierto punto, tras el cual el efecto se estabiliza o se revierte ligeramente. Esto puede indicar que existe un número óptimo de hormigas que permite una mejor exploración y explotación del espacio de búsqueda en el algoritmo, lo que me recuerda a un concepto de economía: la "Ley de los rendimientos marginales decrecientes".

- Un aumento en la tasa de evaporación conduce a una mayor evaporación de feromonas, lo que puede resultar en una menor cantidad de feromonas en los bordes y, por lo tanto, un aumento en la longitud total del camino más corto. Por otro lado, una disminución en la tasa de evaporación puede generar una mayor acumulación de feromonas en los bordes y, en consecuencia, una reducción en la longitud total del camino más corto.