UNIVERSIDAD DE GUADALAJARA

Centro universitario de ciencias exactas e ingenierías

Práctica 3. Búsqueda del camino más corto usando el algoritmo de optimización por colonia de hormigas (ACO).

SEMINARIO DE SOLUCION DE PROBLEMAS DE INTELIGENCIA ARTIFICIAL I

Alumno: Carlos Eduardo Borja Soltero

Código: 218292319

Sección: D05

Fecha: 06/Marzo/25

Introducción.

El Algoritmo de Optimización por Colonia de Hormigas (ACO) es una técnica metaheurística inspirada en el comportamiento colectivo de las hormigas reales cuando buscan alimento. Fue desarrollado por Marco Dorigo en 1992 y se ha convertido en un enfoque ampliamente utilizado para resolver problemas de optimización combinatoria, como la búsqueda del camino más corto en un grafo o el problema del viajante de comercio (TSP). Su éxito radica en la forma en que simula la comunicación indirecta entre las hormigas mediante feromonas, permitiendo que encuentren rutas óptimas de manera distribuida y eficiente.

Las hormigas en la naturaleza no tienen visión, pero pueden encontrar el camino más corto entre su nido y una fuente de alimento gracias a la liberación de feromonas en el suelo. Cuando una hormiga encuentra un camino exitoso, refuerza la cantidad de feromonas en ese trayecto, haciendo que otras hormigas lo sigan con mayor probabilidad. Con el tiempo, los caminos más largos o menos eficientes van perdiendo feromonas debido a la evaporación, favoreciendo la consolidación del mejor trayecto disponible. El ACO imita este comportamiento para explorar soluciones óptimas en problemas de optimización.

El algoritmo comienza con la representación del problema como un grafo, donde los nodos representan puntos de decisión y las aristas corresponden a las conexiones entre estos nodos, con un peso asociado que representa la distancia, costo o tiempo. Inicialmente, todas las aristas tienen un nivel bajo de **feromonas**. Luego, se generan múltiples agentes (hormigas virtuales) que exploran diferentes caminos en el grafo, guiándose por una regla probabilística basada en la cantidad de feromonas en las aristas y un valor heurístico que prioriza caminos más cortos. Esta regla de selección de caminos está determinada por la siguiente ecuación:

$$P_{ij} = \frac{(\tau_{ij})^{\alpha} \cdot (\eta_{ij})^{\beta}}{\sum_{k \in \text{permitidos}} (\tau_{ik})^{\alpha} \cdot (\eta_{ik})^{\beta}}$$

1. Definición de la matriz de adyacencia.

```python
graph = {
    'A': {'B': 2, 'C': 4},
    'B': {'A': 2, 'C': 1, 'D': 7},
    'C': {'A': 4, 'B': 1, 'E': 3},
    'D': {'B': 7, 'E': 2, 'F': 5},
    'E': {'C': 3, 'D': 2, 'F': 6},
    'F': {'D': 5, 'E': 6}
}
```

La imagen representa un grafo ponderado no dirigido utilizando una lista de adyacencia en Python, una estructura eficiente donde cada nodo está asociado a un diccionario de vecinos con los pesos de sus aristas. Esta representación es ideal para grafos dispersos ya que utiliza menos memoria que una matriz de adyacencia y permite una recorrido eficiente sobre los nodos conectados.

La lista de adyacencia del código es óptima para representar redes de transporte, rutas de vehículos y problemas de optimización, ya que permite explorar rutas de manera eficiente sin almacenar datos innecesarios.

2.- Generar la población inicial.

hormigas.py

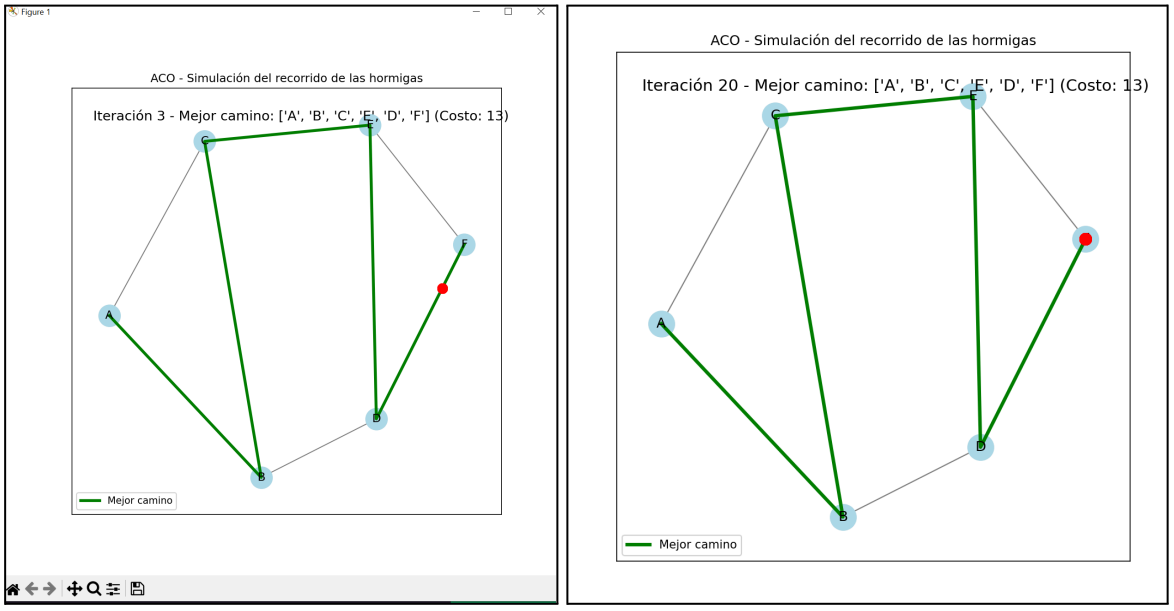
```python
# Parámetros del ACO
alpha = 1.0      # Influencia de la feromona
beta = 2.0       # Influencia del valor heurístico (1/distancia)
rho = 0.5        # Tasa de evaporación
Q = 100          # Constante para depósito de feromonas
num_ants = 10    # Número de hormigas por iteración
num_iterations = 20  # Número de iteraciones

# Inicialización de feromonas: todas las aristas parten de 0.1
pheromones = {i: {j: 0.1 for j in graph[i]} for i in graph}

# Valor heurístico: inversa de la distancia
heuristic = {i: {j: 1.0/graph[i][j] for j in graph[i]} for i in graph}
```

La población inicial consiste en un número determinado de hormigas, cada una comenzando su recorrido desde el nodo de inicio (por ejemplo, A en el problema del camino más corto).

Además, se deben establecer parámetros clave del ACO de forma arbitraria o experimental, como:

Número de hormigas (m m): Define cuántas soluciones se generan en cada iteración.
Influencia de las feromonas (α α): Controla cuánto influyen las feromonas en la elección de caminos.
Influencia de la heurística (β β): Determina la importancia de la distancia en la selección del siguiente nodo.
Tasa de evaporación (ϱ ρ): Regula la desaparición de feromonas con el tiempo.
Cantidad de feromona depositada (Q Q): Define cuánto refuerzo recibe un camino exitoso.

7. Resultados de la Ejecución del Programa.



1. Iteración actual

La simulación está en la Iteración 3, lo que indica que se han realizado tres ciclos completos en los que las hormigas han explorado diferentes caminos.

En cada iteración, las hormigas recorren el grafo, explorando rutas y depositando feromonas en los caminos utilizados.

2. Mejor camino encontrado

El mejor camino actual en la iteración 3 es ['A', 'B', 'C', 'E', 'D', 'F'] con un costo total de 13.

Este camino ha sido identificado como el más eficiente hasta el momento basándose en la acumulación de feromonas y la distancia total.

7. Conclusión y resumen de los resultados.

Conclusión Personal

En mi opinión, creo que el algoritmo de optimización por colonia de hormigas es muy interesante, ya que basándonos en un proceso de la vida real, de la naturaleza, que es, como las hormigas buscan la mejor ruta para encontrar la comida y llevarla a su nido, y el tema de como utilizan las feromonas para guiar a las demás hormigas, considero que es algo muy interesante, y que haya algoritmos que se basen en estos procesos de la vida real, es algo muy interesante.

8. Referencias Bibliográficas.

- Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press. Disponible en línea
- Chollet, F. (2021). Deep Learning with Python (2nd ed.). Manning Publications.
- Ruder, S. (2016). An overview of gradient descent optimization algorithms. https://arxiv.org/abs/1609.04747
- Nielsen, M. A. (2015). Neural Networks and Deep Learning. http://neuralnetworksanddeeplearning.com/
- Geron, A. (2019). Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow. O'Reilly Media.
- Bishop, C. M. (2006). Pattern Recognition and Machine Learning. Springer.
- Aggarwal, C. C. (2018). Neural Networks and Deep Learning: A Textbook. Springer.
- Raschka, S., & Mirjalili, V. (2019). Python Machine Learning. Packt Publishing.
- Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press.
- Géron, A. (2019). Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow (2nd ed.). O'Reilly Media.