Práctica 3

18 de agosto de 2023

Resumen

0.1. Introducción

El problema del agente viajero (TSP, Travelling salesman problem) es uno de los problemas de optimización combinatoria más conocidos y estudiados en la teoría de la optimización y la investigación de operaciones.
Este problema se plantea de la siguiente manera: Dado un conjunto de ciudades y las distancias entre cada par de ciudades, el objetivo es encontrar la ruta más corta que visite cada ciudad exactamente una vez y regrese a la ciudad de origen. El problema busca minimizar la distancia total recorrida por el agente viajero mientras cumple con las restricciones mencionadas.

El TSP se puede modelar y resolver utilizando conceptos y técnicas de teoría de grafos, en donde las ciudades y las distancias entre ellas se pueden representar como un grafo completo en el que cada ciudad es un nodo y cada arista entre dos nodos representa la distancia entre esas dos ciudades. Las ponderaciones de las aristas (distancias) son las que deben ser minimizadas en el TSP. Aqui entra otro concepto de la teoria de grafos, los ciclos Hamiltonianos.

Un ciclo Hamiltoniano en un grafo es un ciclo que visita cada nodo exactamente una vez. En el TSP, la solución óptima es un ciclo Hamiltoniano que también es de longitud mínima, es decir, la ruta más corta que visita todas las ciudades una vez y regresa a la ciudad de inicio.

La mayoría de los algoritmos de resolución del TSP utilizan estructuras de grafos y técnicas de teoría de grafos para buscar soluciones óptimas o aproximadas. En este ssentido, los métodos exactos y heurísticos a menudo explotan propiedades del grafo, como la búsqueda de árboles generadores mínimos (MST) o la búsqueda de rutas óptimas basadas en el algoritmo de Dijkstra.

Debido al gran costo computacional del TSP, es que se buscan soluciones de optimizacion, algunos de los cuales tienen un enfoque en el comportamiento de la naturaleza, como los algoritmos geneticos, busqueda de cuervos, colonia de hormigas, etc.

El objetivo de esta práctica es explorar cómo los conceptos de teoría de grafos pueden ser aplicados para resolver y evaluar el rendimiento de un algoritmo específico diseñado para abordar el TSP.

0.2. Algoritmos

El Problema del Agente Viajero es de tipo NP-duro, lo que significa que no se conoce un algoritmo eficiente para resolver instancias grandes en tiempo polinómico. Sin embargo, existen diversos métodos heurísticos y algoritmos de aproximación que se utilizan para abordar el problema y encontrar soluciones aceptables en un tiempo razonable. Uno de estos es el metodo de **Colonia de Hormigas**.

0.2.1. Algoritmo de Colonia de Hormigas

Es una técnica heurística inspirada en el comportamiento de las hormigas reales que encuentran caminos eficientes entre su colonia y fuentes de alimento.

- **Hormigas artificiales:** En el contexto de la colonia de hormigas, se utilizan "hormigas artificiales", que son agentes virtuales que se mueven por el espacio de soluciones en busca de rutas óptimas.
- **Feromonas:** Las hormigas reales dejan un rastro de feromonas en el camino que recorren. Las feromonas funcionan como señales químicas que otras hormigas pueden seguir. En el contexto del método de colonia de hormigas, las "feromonas artificiales"son utilizadas para marcar las soluciones y guiar la exploración.
- **Construcción de soluciones:** Las hormigas artificiales construyen soluciones iterativamente. Comienzan desde una ciudad inicial y eligen su próxima ciudad en función de la cantidad de feromonas en los caminos y una heurística que puede estar basada en la distancia entre ciudades.
- **Depósito de feromonas:** Después de construir una solución, una hormiga artifical deposita una cantidad de feromonas en cada arco (camino) de la solución, y esta cantidad puede depender de la calidad de la solución encontrada.
- **Evolución de las feromonas:** Con el tiempo, las feromonas en los caminos se evaporan gradualmente. Esto simula la tendencia natural de las feromonas a desaparecer con el tiempo en la vida real.
- **Explotación y exploración:** Las hormigas balancean entre la exploración de nuevas rutas (basadas en heurísticas) y la explotación de rutas con más feromonas (que son más prometedoras en función de la experiencia previa de la colonia).
- **Iteraciones y convergencia:** El proceso de construcción de soluciones, depósito de feromonas y evaporación se repite durante varias iteraciones. Con el tiempo, la colonia de hormigas tiende a converger hacia rutas más óptimas.

0.3. Implementación

La implementación se realizó en Python, a continuación se detallan las clases y metodos usados

0.3.1. Algoritmo de Colonia de Hormigas (ACO)

- **Método __init__:** Este es el constructor de la clase. Se inicializan los parámetros de la colonia de hormigas, como las distancias entre ciuda-

des, el número de hormigas, el número de mejores caminos, el número de iteraciones, la tasa de decaimiento, los valores alfa y beta.

- **Método run():** Este método ejecuta el algoritmo de colonia de hormigas. Itera a lo largo del número especificado de iteraciones, generando todos los caminos posibles, distribuyendo feromonas y actualizando el camino más corto global.
- **Método spread_pheronome():** Este método distribuye feromonas en los caminos según el número de mejores caminos especificados. Los caminos son ordenados por longitud y luego se distribuyen las feromonas en los primeros caminos.
- **Método gen_path_dist():** Este método calcula la distancia total de un camino dado.
- **Método gen_all_paths():** Este método genera todos los caminos para cada hormiga y calcula sus distancias.
- **Método gen_path():** Este método genera un camino para una hormiga específica.
- **Método pick_move():** Este método elige un movimiento para una hormiga en función de la probabilidad de feromonas y distancia.

0.3.2. Matriz TSP

Se utilizó una matriz cuadrada espejo como representación

$$Distancias = \begin{bmatrix} np.inf & 113 & 56 & 167 & 147 \\ 113 & np.inf & 137 & 142 & 98 \\ 56 & 137 & np.inf & 133 & 135 \\ 167 & 142 & 133 & np.inf & 58 \\ 147 & 98 & 135 & 58 & np.inf \end{bmatrix}$$

Donde:

- 0 = Grand Rapids
- 1 = Saginaw
- 2 = Kalamazoo
- 3 = Toledo
- 4 = Detroit

Para los experimentos con un mayor número de ciudades, se utilizaron matrices generadas aleatoriamente

El codigo fuente se puede encontrar en el siguiente repositorio

0.4. Resustados

0.5. Conclusiones