Implementar el algoritmos

(A*, MST , TSP)

Profesor: Angel Augusto Agudelo Zapata

Andres Felipe Gordillo Guerrero

1088348241

a.gordillo@utp.edu.co

Inteligencia Artificial

Universidad tecnológica de pereira

2024 - Semestre II

Proyecto de Documentación: Implementación de Algoritmos de Búsqueda

1. Introducción

Este proyecto implica la creación e implementación de búsquedas de rutas sobre un grafo utilizando algoritmos con varias heurísticas en Python. Usando el algoritmo A* y tres heurísticas diferentes, nuestro objetivo es encontrar el camino más óptimo entre dos nodos y analizar las diferencias entre ellas. Además, se realizará un seguimiento de los caminos explorados, mostrándoles gráficamente.

2. Objetivos del Proyecto

El objetivo principal del proyecto es comparar tres heurísticas de búsqueda dentro de un paquete gráfico que permita visualizar la exploración de las rutas. La salida de los resultados se presentará a través de una secuencia de instrucciones en la consola y gráficamente, facilitando la comparación de la eficiencia y el costo de las diferentes heurísticas.

3. Descripción de las Heurísticas

Las heurísticas utilizadas en este proyecto son:

- **Heurística Euclidiana**: Esta heurística calcula la distancia rectilínea entre el nodo actual y el nodo objetivo, ignorando cualquier obstáculo en el medio.
- **Heurística de Manhattan**: Esta heurística mide la distancia en un plano, considerando que solo se puede mover horizontal y verticalmente.
- **Heurística de Árbol de Cobertura Mínima (MST)**: Usando MST, esta heurística estima el costo restante desde el nodo actual hasta el nodo objetivo.

Cada una de estas heurísticas tiene características particulares, y este proyecto ayudará a comprender cómo funcionan y cuáles son sus diferencias.

4. Parámetros Necesarios

Bibliotecas

Se requieren las siguientes bibliotecas de Python para este proyecto:

- **networkx**: Para la creación y manipulación de grafos.
- **matplotlib**: Para la visualización gráfica de grafos y rutas.

Estas bibliotecas se pueden instalar usando las siguientes instrucciones:

```
pip install matplotlib networkx
```

Estructura del Código

El código se divide en varias secciones que abarcan la inicialización del grafo, la implementación de las heurísticas, la ejecución de la búsqueda A*, y la visualización gráfica de los resultados.

5.1. Inicialización del Grafo

Se genera un grafo en el que cada nodo representa un punto en el espacio, y las aristas representan rutas posibles entre estos nodos. Utilizamos la librería `networkx` para definir el grafo y asignar pesos a las aristas.

5.2. Implementación de Heurísticas

Se definen tres funciones, cada una correspondiente a una heurística. Estas funciones calculan la distancia desde un nodo dado hasta el objetivo y devuelven el valor a utilizar en el algoritmo A*.

Heurística Euclidiana:

```
def euclidean_heuristic(node, goal):
    x1, y1 = positions[node]
    x2, y2 = positions[goal]
    return ((x2 - x1) ** 2 + (y2 - y1) ** 2) ** 0.5
```

Heurística de Manhattan:

```
def manhattan_heuristic(node, goal):
    x1, y1 = positions[node]
    x2, y2 = positions[goal]
    return abs(x1 - x2) + abs(y1 - y2)
```

Heurística de Árbol de Expansión Mínima (MST):

```
def mst_heuristic(node, goal, graph):
    subgraph_nodes = list(nx.shortest_path(graph, node, goal))
    mst = nx.minimum_spanning_tree(graph.subgraph(subgraph_nodes))
    return mst.size(weight="weight")
```

5.3. Implementación del Algoritmo A*

Se utiliza una implementación personalizada del algoritmo A* para buscar en el grafo. La función toma como parámetros el grafo, el nodo inicial, el nodo objetivo y la función heurística.

```
def a_star(graph, start, goal, heuristic_func):
    ...
    return path, total_cost, end_time - start_time
```

5.4. Gráfica

Utilizamos `matplotlib` y `networkx` para generar los gráficos. Se crea una figura para cada heurística, mostrando el camino encontrado, el costo y el tiempo de ejecución.

```
def plot_graph(graph, positions, path, title):

    plt.figure(figsize=(6, 6))

    ...

    plt.title(title)

    plt.show()
```

Ejecución del Código

El código principal ejecuta el algoritmo A* tres veces, una vez por cada heurística, y muestra los resultados en la consola y en un gráfico.

```
if __name__ == "__main__":

    start, goal = 0, len(graph.nodes) - 1

    for heuristic_name, heuristic_func in heuristics.items():

        path, total_cost, exec_time = a_star(graph, start, goal,
heuristic_func)

        print(f"{heuristic_name} - Cost: {total_cost}, Time:
{exec_time:.4f}s")

        plot_graph(graph, positions, path, f"{heuristic_name} -
Cost: {total_cost}, Time: {exec_time:.4f}s")
```

Resultados

- **Consola**: muestra el costo y el tiempo de ejecución de cada heurística.
- **Gráfico**: genera tres figuras, cada una titulada con el costo y el tiempo de ejecución de cada heurística, mostrando el camino encontrado.

Datos para Ingresar

Viajes entre ciudades en Colombia
Datos proporcionados por Google. La ciudad de **Pereira** es el punto de inicio y se representará el peso como las horas de camino entre cada ciudad.

Ejemplo de Distancias en Horas:

- **Pereira - Bogotá**:
  - Peso: 7h
- **Pereira - Cali**:
  - Peso: 3h
- **Pereira - Medellín**:
  - Peso: 4h
- **Medellín - Bogotá**:
  - Peso: 8h
- **Medellín - Cali**:
  - Peso: 7h
- **Cali - Bogotá**:
  - Peso: 9h

Iteraciones

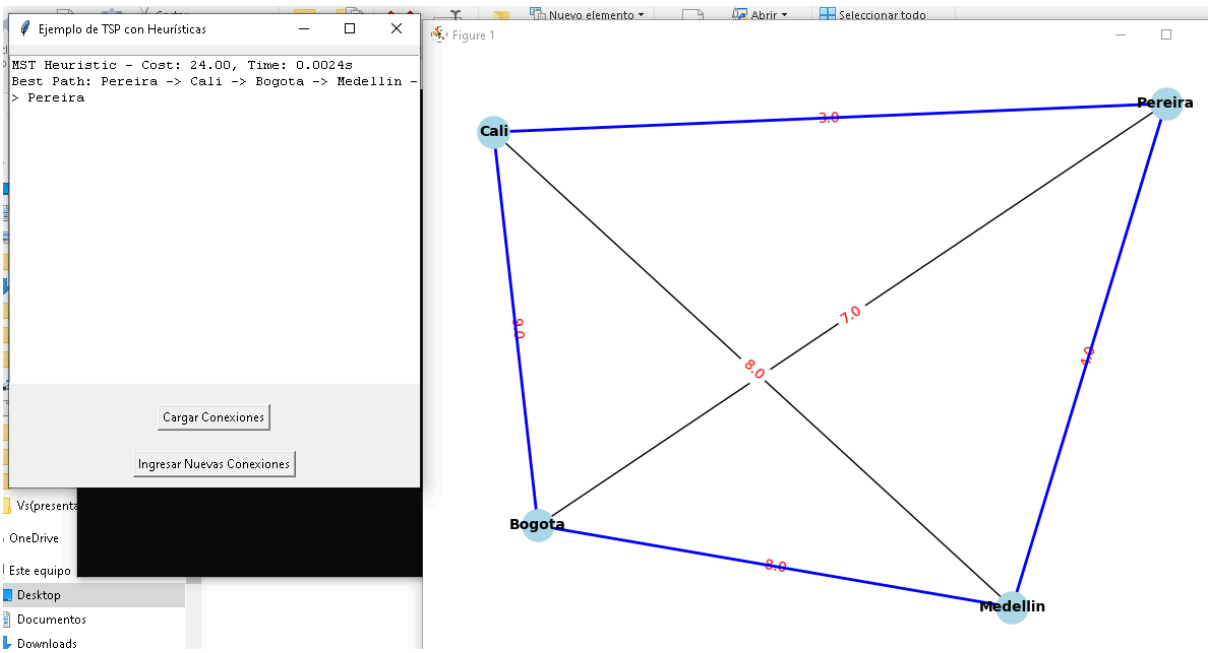
Se utilizará un vector de **Costo Total (CT)** con 3 opciones variables: **Bogotá (B)**, **Cali (C)** y **Medellín (M)**.

Iteración 1: Desde Pereira

Ciudad	Costo Total (CT)	Elección	Total Horas
Pereira	CT (B = 7, C = 3, M = 4) += 14	- C(3)	= 25h
Cali	CT (P = 3, M = 7, B = 9) += 19	- M(7)	
Medellín	CT (P = 4, B = 8, C = 7) += 19	- B(8)	
Bogotá	CT (P = 7, M = 8, C = 9) += 24	- P(7)	

Iteración 2: Desde Pereira

Ciudad	Costo Total (CT)	Elección	Total Horas
Pereira	CT (B = 7, C = 3, M = 4) += 14	- C(3)	= 24h
Cali	CT (P = 3, M = 7, B = 9) += 19	- B(9)	
Bogotá	CT (P = 7, M = 8, C = 9) += 24	- M(8)	
Medellín	CT (P = 4, B = 8, C = 7) += 19	- P(4)	



Ejemplo 2: Viajes entre Ciudades

Nodos de conexión:

1. Pereira
2. Medellín
3. Cartagena
4. Cali
5. Bogotá
6. Barranquilla
7. Popayán
8. Pasto
9. Bucaramanga
10. Neiva

Distancias en Horas

- **Bogotá**
    - Medellín: 8h
    - Cartagena: 18h
    - Cali: 9h
    - Pereira: 7h
    - Barranquilla: 17h
    - Popayán: 11h
    - Pasto: 17h
    - Bucaramanga: 8h
    - Neiva: 6h
- **Medellín**
    - Cartagena: 11h
    - Cali: 7h
    - Pereira: 4h
    - Barranquilla: 13h
    - Popayán: 9h
    - Pasto: 15h
    - Bucaramanga: 6h
    - Neiva: 9h
- **Cartagena**
    - Cali: 18h
    - Pereira: 15h
    - Barranquilla: 2h
    - Popayán: 21h
    - Pasto: 27h
    - Bucaramanga: 11h
    - Neiva: 18h
- **Cali**
    - Pereira: 3h
    - Barranquilla: 20h
    - Popayán: 3h
    - Pasto: 7h
    - Bucaramanga: 13h
    - Neiva: 8h
- **Pereira**
    - Barranquilla: 17h
    - Popayán: 6h
    - Pasto: 11h
    - Bucaramanga: 10h
    - Neiva: 5h
- **Barranquilla**
    - Popayán: 23h
    - Pasto: 29h
    - Bucaramanga: 11h
    - Neiva: 18h

- **Popayán**
    - Pasto: 5h
    - Bucaramanga: 15h
    - Neiva: 5h
- **Pasto**
    - Bucaramanga: 20h
    - Neiva: 11h
- **Bucaramanga**
    - Neiva: 10h

Conclusión

Este proyecto permite observar cómo diferentes heurísticas afectan el rendimiento del algoritmo A* en la búsqueda de rutas. Cada heurística tiene ventajas según el contexto y el tipo de grafo. La gráfica facilita el análisis visual de los caminos encontrados, mientras que la salida en consola proporciona métricas numéricas del desempeño de cada heurística.