ROUTING ALGORITHM FOR OCEAN SHIPPING AND URBAN DELIVERIES

Carlos Filipe Oliveira Sanches Pinto up202107694

David dos Santos Ferreira up202006302

João Maria Correia Rebelo up202107209

DATA

- **1.Dataset Types**:

Small Toy Graphs: Three small datasets representing simplified delivery points in Porto. These are used to validate our algorithms and establish a benchmark for comparing heuristic and exact solutions.

Medium-Sized Graphs: Twelve fully connected graphs ranging from 25 to 900 nodes. These datasets allow us to test and optimize our heuristics before applying them to larger, more complex graphs.

Real-World Graphs: Three large graphs derived from actual urban delivery and ocean shipping data. These include:

Graph 1: 1K nodes and ~500K edges

Graph 2: 5K nodes and ~4M edges

Graph 3: 10K nodes and ~10M edges

READING THE GIVEN DATASET

- **1-Initialization**:

The file reader initializes by opening the specified dataset file in CSV format. This file contains either node information (with geographic coordinates) or edge information (with distances between nodes).

- **2-File Parsing**:

Nodes File: The reader processes the nodes file to extract each node's identifier and its corresponding geographic coordinates (latitude and longitude). This information is stored in a data structure that maps node identifiers to their coordinates.

Edges File: The reader processes the edges file to extract pairs of node identifiers and the distance between them. Each edge entry is stored in a list or a graph data structure representing the network of routes.

- **3-Data Structures**:

The parsed data is stored in appropriate data structures:

  - **Graph Representation**: Nodes and edges are organized into a graph, where nodes are vertices, and edges are weighted connections between these vertices.
  - **Adjacency List**: An adjacency list is created for efficient traversal and algorithmic operations. Each node points to a list of neighboring nodes along with the corresponding edge weights.
  - **The use of unordered maps**: Made not only the parsing efficient but also the node search because it has a search of O(1) given the hash key

4.1. BACKTRACKING ALGORITHM

- To tackle the Travelling Salesperson Problem (TSP), we implemented a backtracking algorithm designed to find the optimal route in a graph, starting and ending at the node labeled with the zero-identifier. The algorithm operates by recursively exploring all possible paths, marking nodes as visited or unvisited as it progresses. For each path, it calculates the total distance traveled and updates the shortest path found whenever a shorter complete tour is identified. This exhaustive search ensures that we find the optimal solution, though at the cost of high computational complexity.

4.2. TRIANGULAR APPROXIMATION HEURISTIC

In our implementation of the 2-approximation algorithm for the Travelling Salesperson Problem (TSP), we employed the Minimum Spanning Tree (MST) calculation using Prim's algorithm and followed it with a preorder traversal. Here's why and how we used these methods:

1.Minimum Spanning Tree (MST) Calculation using Prim's Algorithm:

Why Prim's Algorithm: We chose Prim's algorithm because it is efficient for dense graphs and ensures the construction of an MST with a time complexity of O(V^2) for adjacency matrix representation or O(E + V log V) with a priority queue. This makes it suitable for our dataset, where we need to ensure all nodes are connected with the minimum total edge weight.

Purpose: The MST represents a subset of the edges that connects all vertices together without any cycles and with the minimum possible total edge weight. This is a crucial step because the MST provides a framework that approximates the optimal TSP tour without redundant paths.

2 . Preorder Traversal of the MST:

Why Preorder Traversal: Preorder traversal is used to generate a tour of the nodes in the MST. This method ensures that we visit each node exactly once before returning to the starting node, mimicking the behavior of a depth-first search (DFS).

Purpose: The preorder traversal converts the MST into a Hamiltonian circuit, visiting nodes in the order they are first encountered. This approach avoids revisiting nodes unnecessarily and helps in creating a feasible tour that approximates the solution to the TSP.

The time complexity of Prim's algorithm is O(V^2).

WHY THIS APPROACH WORKS

The combination of MST calculation and preorder traversal works effectively for the following reasons:

Efficiency: Prim's algorithm efficiently constructs the MST, ensuring that we have a minimal total edge weight to work with. This reduces the overall distance we need to cover in the tour.

Approximation Quality: By converting the MST to a TSP tour using preorder traversal, we ensure that the solution is within a factor of 2 of the optimal solution. This is because the total weight of the MST is a lower bound on the TSP tour, and the preorder traversal essentially doubles back at most once over any edge, leading to a 2-approximation.

Scalability: This approach is computationally feasible for larger graphs, making it suitable for both small and large datasets. The simplicity of preorder traversal ensures that we can quickly generate a tour after constructing the MST.

4.3. OTHER HEURISTICS

- **Algorithm Design**: The Nearest Neighbor heuristic starts at the node labeled with the zero-identifier and repeatedly visits the nearest unvisited node until all nodes have been visited, finally returning to the starting node. This method is straightforward and prioritizes minimizing the immediate distance traveled at each step.

- **Initialization**: Begin at the starting node (node 0).
  - **Finding Nearest Neighbor**: At each step, select the closest unvisited node.
  - **Marking as Visited**: Mark the selected node as visited and move to it.
  - **Repeat**: Repeat the process until all nodes are visited.
  - **Return to Start**: Return to the starting node to complete the tour.

  The Nearest Neighbor heuristic is particularly useful for large datasets due to its O(V**2) time complexity, making it significantly faster than exact algorithms like backtracking.

WHY THIS APPROACH WORKS?

- **Local Optimization**: By always selecting the nearest unvisited node, the heuristic ensures that each step is locally optimal. This tends to keep the total tour length relatively short, although it may not be globally optimal.
- **Simplicity and Speed**: The algorithm is simple to implement and fast to execute, making it suitable for large datasets where more complex algorithms would be computationally prohibitive.
- **Practicality**: In many real-world scenarios, having a quick, near-optimal solution is more practical than spending excessive computational resources to find the absolute optimal solution.

4.4. TSP IN THE REAL WORLD

Objective: The algorithm aims to provide a feasible solution to the TSP, even when the graph is not fully connected. It starts from an arbitrary node provided by the user and attempts to visit all nodes, returning to the origin if possible.

Feasibility Check: If no valid tour exists that visits all nodes and returns to the starting point, the algorithm outputs a message indicating the infeasibility.

- **1-Input Handling**: The user provides the starting node for the TSP tour. The algorithm must dynamically adjust to any starting point.
- **2-Graph Representation**: The graph is represented using adjacency lists or matrices, incorporating only the edges provided in the dataset.

3-Pathfinding and Connectivity Check:

- The algorithm first checks if all nodes are reachable from the starting node using a graph traversal method like Depth-First Search (DFS) or Breadth-First Search (BFS).
- If any node is found to be unreachable, the algorithm concludes that a complete tour is not possible and outputs an appropriate message.

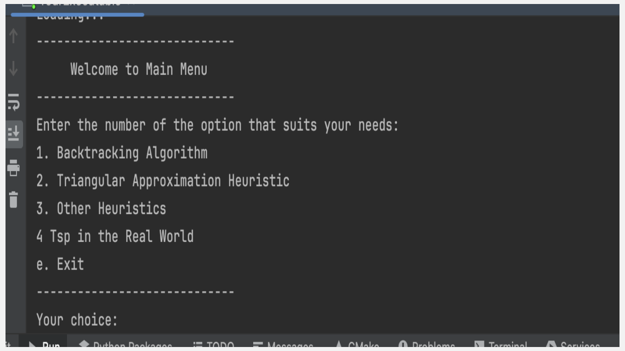
4-Tour Construction:

- If all nodes are reachable, the algorithm constructs a tour. It uses heuristic methods, such as Nearest Neighbor, to approximate a solution.
- The algorithm prioritizes finding a feasible path that covers all nodes and returns to the start.

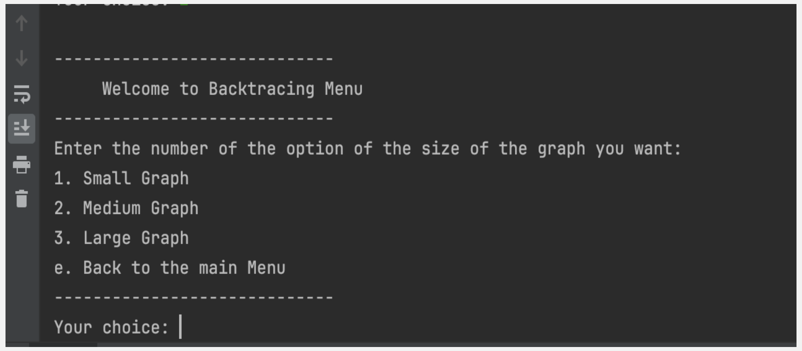
WHY THIS APPROACH WORKS?

- **Handling Partial Connectivity**: By focusing on checking connectivity first, the algorithm ensures that it only attempts to construct a tour when it is feasible, saving computational resources.
- **Heuristic Flexibility**: Using heuristics like Nearest Neighbor allows the algorithm to adapt to various graph structures and sizes, providing a good balance between solution quality and computational efficiency.
- **Real-World Applicability**: This method aligns well with real-world scenarios where some routes may not exist, and the graph might be sparse. It reflects practical constraints and provides solutions that are operationally viable

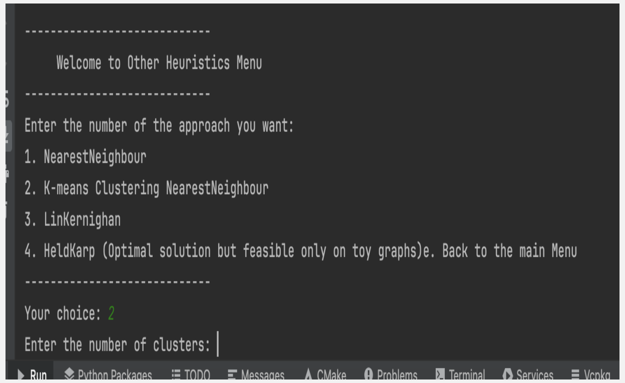
USER INTERFACE



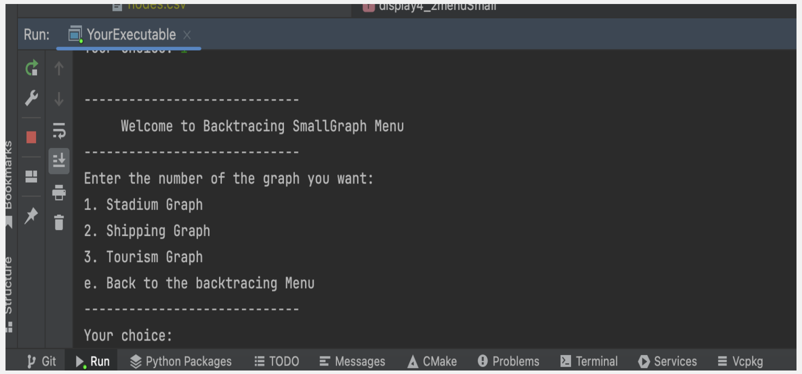
```
----------------------------
     Welcome to Main Menu
----------------------------
Enter the number of the option that suits your needs:
1. Backtracking Algorithm
2. Triangular Approximation Heuristic
3. Other Heuristics
4 Tsp in the Real World
e. Exit
----------------------------
Your choice:
```



```
----------------------------
     Welcome to Backtracing Menu
----------------------------
Enter the number of the option of the size of the graph you want:
1. Small Graph
2. Medium Graph
3. Large Graph
e. Back to the main Menu
----------------------------
Your choice:
```



```
----------------------------
     Welcome to Other Heuristics Menu
----------------------------
Enter the number of the approach you want:
1. NearestNeighbour
2. K-means Clustering NearestNeighbour
3. LinKernighan
4. HeldKarp (Optimal solution but feasible only on toy graphs)e. Back to the main Menu
----------------------------
Your choice: 2
Enter the number of clusters:
```



```
----------------------------
     Welcome to Backtracing SmallGraph Menu
----------------------------
Enter the number of the graph you want:
1. Stadium Graph
2. Shipping Graph
3. Tourism Graph
e. Back to the backtracing Menu
----------------------------
Your choice:
```

FUNCTIONALITY TO HIGHLIGHT

- In our project, we implemented 3 more heuristics to tackle the Travelling Salesperson Problem (TSP) across various scenarios
- **K-means Clustering Nearest Neighbor**: This method enhances the Nearest Neighbor approach by first clustering the nodes using the K-means algorithm. Within each cluster, the Nearest Neighbor heuristic is applied. This reduces the problem size for each sub-problem, potentially leading to better overall tour lengths compared to using Nearest Neighbor alone, the con is it requires a fine tunning with the k-means clustering number of clusters O(V**2).
- **Lin-Kernighan**: A more sophisticated heuristic that iteratively refines a given tour by making a series of local optimizations. The Lin-Kernighan heuristic is well-regarded for its ability to produce high-quality solutions, often approaching optimality, through extensive search and optimization techniques, the problem is it´s computational cost making it hardly scalable O(V**3).
- **Held-Karp**: This algorithm provides an optimal solution to the TSP but is computationally feasible only for small, toy graphs due to its exponential time complexity. It serves as a benchmark for evaluating the performance and accuracy of our heuristic approaches on smaller datasets O((V**2)*2**V).

CONCLUSION

- In conclusion, this project provided a comprehensive exploration of the Travelling Salesperson Problem (TSP) through the implementation and analysis of various algorithms and heuristics. By developing solutions such as the Nearest Neighbor, K-means Clustering Nearest Neighbor, Lin-Kernighan, and Held-Karp algorithms, we were able to address the complexities and challenges posed by both small and large graphs. Our work demonstrated the trade-offs between computational efficiency and solution optimality, highlighting the practicality of heuristic methods in real-world applications where exact solutions are computationally prohibitive.