Farid Gahramanov

Report on Implementation of DFS, UCS, and A* Algorithms

Introduction :

In this project the main goal was to use several tree-search algorithms and heuristic functions to discover optimal or nearly optimal solutions for the routing problem.

Depth First Search (DFS) :

Implementation - >

1. It starts with a depote node and stack is used to manage the path.
2. After, nodes are marked as visited once they're included in a path.
3. As the path grows, the cumulative load is checked to ensure it doesn't exceed the capacity.
4. If a path returns to the depot and visits all store nodes, it is returned as the solution.
5. Else, it will explore furthermore, by using neighboring nodes.

Here is the Pseucode for better understanding :

Pseudo-Code:

```
1.Initialize a stack with the depot node and a set to track visited nodes.
2. While the stack is not empty:
        ->Pop the latest path and get the current node.
        ->Calculate cumulative load along the path.
        ->If the path is complete (returns to the depot with all store nodes visited), return it.
        ->Otherwise, explore neighboring nodes that haven't been visited and meet load
          constraints.
3. Return the last valid path or an empty solution.
```

Uniform Cost Search (UCS) :

The main point of the UCS is to explore path’s based on the total cost by always choosing the lowest one.

Implementation - >

1. A priority queue is used to keep track of paths by their cost.
2. The algorithm avoids revisiting nodes and checks that the total load doesn't go over capacity.
3. If a path goes back to the depot and visits all store nodes, it gives the outcome.

Pseucode :

1. Add the depot node to the priority queue and keep track of the lowest cost to each node.
2. While the queue has path’s left :

   ->Get the path with the lowest cost.

   ->If the path completes by returning to the depot and visiting all stores, return it.

   ->If not, keep exploring nearby nodes that don't exceed capacity.

3. Return the last path or return nothing at all.

Astar (A*) :

Heuristic Function: The heuristic is the estimated cost from a node to the depot.

Implementation - >

1. A queue keeps track of paths based on their total cost, including the heuristic.
2. If a path goes back to the depot and visits all store nodes, then comes the outcome.
3. The algorithm tries not to exceed the capacity and revisitation.

Pseucode :

1. Add the depot node to the queue, by also tracking the lowest costs and heuristics.
2. While the queue has paths:
   a. Get the path with the lowest cost.

    b. If the path completes by returning to the depot and visiting all nodes, return the outcome.

    c. If not, keep exploring neighboring nodes, by checking it does not exceed the capacity and get nodes reviseted.

3. Return the last valid path or nothing at all.

Runtime:

DFS has longer runtimes because of deeper exploration before backtracking. However, UCS and A* have shorter runtimes because they use cost-based exploration.

Memory Usage:

DFS uses less memory because it doesn't maintain a complete tree structure.

UCS and A* require more memory due to their priority queues and heuristic calculations.