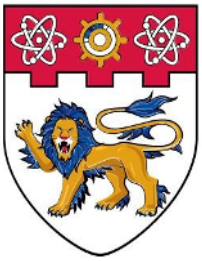


NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

CZ3005 Artificial Intelligence

Lab Assignment 1

Submitted By:

Lee Su Bin

1. Introduction

This document serves as the report submission for CZ3005 Artificial Intelligence Lab Assignment 1. The assignment itself entails three tasks, with each task being elaborated in their respective sections.

The problem given is derived from the NYC instance which is a modified version of the 9th DIMACS implementation challenge. In short, the graph given is a directed weighted graph with two different sets of weights for each edge.

For this lab, it is assumed that the shortest path is expected for all the tasks.

We are given four json files and we use these files as such:

1. G.json: Contains the adjacency matrix of the graph.

2. Coord.json: Contains the coordinates of each node. Considering these coordinates are in a two-dimensional space, we will use Euclidean Distance between any node and the target node (node 50 in our case) as the h(n) value in our A* Search for Task 3.

3. Cost.json: Contains cost between two nodes (i.e. a weighted edge). This is used for the energy cost constraint for Task 2 and Task 3.

4. Dist.json: Distance between two connected nodes. This distance serves as the g(n) value in our A* Search.

2. Tasks

Task 1

You will need to solve a relaxed version of the NYC instance where we do not have the energy constraint. You can use any algorithm we discussed in the lectures. Note that this is equivalent to solving the shortest path problem. (30 marks)

Task 1 is simply figuring out the shortest path from one node to another. Shortest path is the combination of several shortest distances, which uses dynamic programming. Since distance is always a positive value, Dijkstra's algorithm is a reasonable choice for this task. It is a shortest path finding algorithm used to calculate shortest path from one particular node to another, mostly used in real-life problems like this task. It uses previous information of the shortest path to obtain a new one.

There are several ways to implement Dijkstra's algorith*m*. First way is the simple linear search using 'for loop', which has a time complexity of O(n^2). Another way is using priority queue, which has the time complexity of O(Nlog2N), which is smaller than O(N^2). Therefore, we used the priority queue to solve this task.

Priority queue used in this algorithm, returns the minimum value, which means nodes with smaller value have higher priority regardless of the order of the entry. To implement this, we imported a heapq *module* which provides a heap made of tree structure. Priority queue is used to select the node with shortest distance among the nodes that have not been visited. Also, it is used to store the closest node. So, you compare the distance of the node taken out of the queue with that of the existing distance. If the new distance is smaller, you modify the distance table. By repeating this, you can attain the shortest distance.

```
Shortest path: 1->1363->1358->1357->1356->1276->1273->1277->1269->1267->1268->1284->1283->1282->1255->1253->1260->1259->1249->1246->963->964->962->1002->952->1000->998->994->995->996->987->988->979->980->969->977->989->990->991->2369->2366->2340->2338->2339->2333->2334->2329->2029->2027->2019->2022->2000->1996->1997->1993->1992->1989->1984->2001->1900->1875->1874->1965->1963->1964->1923->1944->1945->1938->1937->1939->1935->1931->1934->1673->1675->1674->1837->1671->1828->1825->1817->1815->1634->1814->1813->1632->1631->1742->1741->1740->1739->1591->1689->1585->1584->1688->1579->1679->1677->104->5680->5418->5431->5425->5424->5422->5413->5412->5411->66->5392->5391->5388->5291->5278->5289->5290->5283->5284->5280->50
Shortest distance: 148648.63722140007
```

Task 2

You will need to implement an uninformed search algorithm (e.g., the DFS,

BFS, UCS) to solve the NYC instance. (30 marks)

For Task 2, Uniform Cost Search is the choice of approach. Other than having a distance between nodes, we also have to take into account the energy cost to move from one node to the other. As such, in addition to finding a shortest path to the goal node, we now have to find the shortest path without exceeding the energy constraint.

The approach is to make use of UCS to traverse to other nodes starting from the root node until the goal node where the path taken has an energy cost below the constraint. Each node traversed is stored in a priority queue where they are ranked by their distances from the root node.

On each iteration of UCS, the node that is currently closest to the root node will be dequeued and checked if it is the goal node. This ensures that only the node that is closest to the root node is considered every single time. If the node is not the goal node, its neighbouring nodes will be enqueued into the priority queue where they will be checked later.

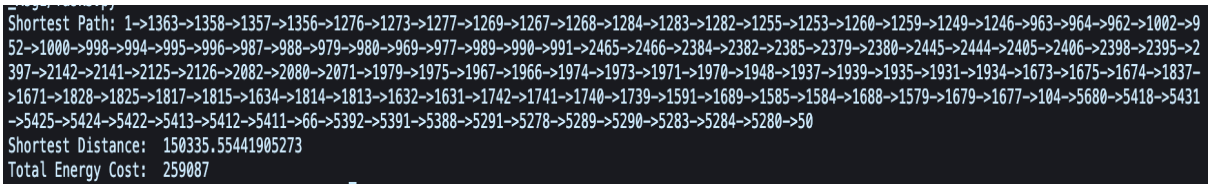
A very practical trick to reduce search time is to check before enqueuing any node into the priority queue whether the energy constraint will be exceeded. This can be done easily by computing and storing the energy cost from the root node to each subsequent node that was searched by UCS.

```
Shortest path: 1->1363->1358->1357->1356->1276->1273->1277->1269->1267->1268->1284->1283->1282->1255->1253->1260->1259->1249->1246->963->964->962->1002->952->1000->998->994->995->996->987->986->979->980->969->977->989->990->991->2369->2366->2340->2338->2339->2333->2334->2329->2029->2027->2019->2022->2000->1996->1997->1993->1992->1989->1984->2001->1900->1875->1874->1965->1963->1964->1923->1944->1945->1938->1937->1939->1935->1931->1934->1673->1675->1674->1837->1671->1828->1825->1817->1815->1634->1814->1813->1632->1631->1742->1741->1740->1739->1591->1689->1585->1584->1688->1579->1679->1677->104->5680->5418->5431->5425->5429->5426->5428->5434->5435->5433->5436->5398->5404->5402->5396->5395->5292->5282->5283->5284->5280->50
Shortest distance: 150785
Total energy cost: 287931
```

Task 3

You will need to develop an A* search algorithm to solve the NYC instance. The key is to develop a suitable heuristic function for the A* search algorithm in this setting. (40 marks)

Task 3 is quite similar to task 2, where it requires us to find k-th shortest path using A* search algorithm. A* search combines both UCS approach of exploring using the cost of adjacent nodes, and the information outside of just the cost. In this case, we were given the coordinates of each node. As the problem is based on the New York City map we decided to use the euclidean distance between any node and the target node as our heuristic function. Running the A* search we can find the shortest path as shown below.



A* search is actually a modification of *Dijkstra's algorithm for finding a single source shortest path*. The only difference is that A* search uses a heuristic function that *guides* the search, pruning unnecessary search tree branches. A good heuristic function should be designed in such a way that its value becomes monotonic along any search path. It should not *overestimate* the actual distance between two states. As euclidean distance is the least possible distance between two points, there's no chance of overestimation on distance between two points.

An incorrect heuristic function restricts the algorithm from finding the shortest path. As the Cost and Distance can be assumed to be somewhat correlated the inferior heuristic function used might find paths with a higher distance and thus never find a path that can meet the energy budget constraint in reasonable time.

3. Conclusion

Through this project, we learned that different searches serve different purposes. There isn’t a ‘best’ search algorithm, but it ultimately depends on the domain where the search is performed. With the right heuristic function used for A* search, it is possible to surpass the efficiency of UCS.

4. References

- https://www.geeksforgeeks.org/a-search-algorithm/
- https://towardsdatascience.com/a-star-a-search-algorithm-eb495fb156bb#:~:text=A%2Dstar%20(also%20referred%20to,s%20to%20find%20the%20solution.
- https://stackoverflow.com/questions/8793411/aa-star-search-algorithm-for-shortest-path
- https://arxiv.org/pdf/1107.0041.pdf
- https://www.geeksforgeeks.org/heap-queue-or-heapq-in-python/
- https://www.geeksforgeeks.org/python-program-for-dijkstras-shortest-path-algorithm-greedy-algo-7/
- https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-using-priority_queue-stl/
- https://www.educative.io/edpresso/what-is-uniform-cost-search