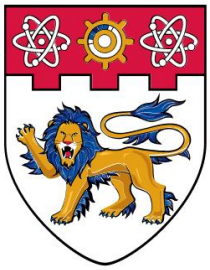

NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

CZ3005 Artificial Intelligence

Lab Assignment 1

Submitted by: Low Yu Benedict

Matriculation Number: U1821762E

Lab Group: FS2

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. Also, some exploratory analysis on the background was performed and this will be covered in the Background section.

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 tasks. Task 1 seeks to only accomplish the shortest path problem which is well within the scope of our syllabus. However, with the added constraint of energy budget we get a more challenging scope that seemingly goes beyond our syllabus.

In short, Tasks 2 and 3 can be reframed as finding the k-th shortest path, as we want the shortest path that meets the requirement for max budget of energy. This approach requires significant refactoring of the algorithms taught in our syllabus such that we are always looking for the ‘next shortest’ path (i.e. a suboptimal path, or the next best optimal path). For this assignment the Yen’s Algorithm is used for finding the k-th shortest path.

A zipped file containing source codes is also included. Running main.py will run all three tasks, although it may take several minutes (~10 minutes max) for tasks 2 and 3. There should not be any external libraries used and as such the script can be run in any Python 3.x environment. Finally, the EDA ipynb in the directory ./notebook/ contains all findings that are outlined in the Background section.

2. Background

All content in this section was derived using the EDA ipynb in the directory ./notebook/. As mentioned, the dataset comes from a modified road ***network*** of New York City.

Naturally, we can expect a road network to possess cycles. This was confirmed in the EDA ipynb Chapter One. This is important to note as we will have to explicitly avoid cycles within our search algorithm to avoid searching endlessly.

Also, the name of vertices has been determined to be unique. As the assignment sheet mentions there are 264346 unique nodes, Chapter Two of our EDA ipynb confirms that there are 264346 unique keys.

There is also no entry for costs of a node looping back to itself. This is found in Chapter Three of the EDA notebook.

Finally, 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.

3. 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)

The goal of task 1 is to find the shortest path from source node 1 to target node 50. This is a rather simple question and Uniform Cost Search (UCS) was used to find the shortest path.

```
Shortest path:
1->1363->1358->1357->1356->1276->1273->1277->1269->1267->1268->1284->1283->1282->125
5->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->23
34->2329->2029->2027->2019->2022->2000->1996->1997->1993->1992->1989->1984->2001->19
00->1875->1874->1965->1963->1964->1923->1944->1945->1938->1937->1939->1935->1931->19
34->1673->1675->1674->1837->1671->1828->1825->1817->1815->1634->1814->1813->1632->16
31->1742->1741->1740->1739->1591->1689->1585->1584->1688->1579->1679->1677->104->568
0->5418->5431->5425->5424->5422->5413->5412->5411->66->5392->5391->5388->5291->5278-
>5289->5290->5283->5284->5280->50.
Shortest distance: 148648.63722140007.
Total energy cost: N/A.
```

4. 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 both Tasks 2 and 3 the question can be reframed as finding the k-th shortest path between nodes 1 and 50. This is because we want to find the shortest path

between source and target nodes while maintaining an energy budget of less than or equal to 287,932.

Yen's algorithm was used because of the following reasons:

1. Weights of edges are non-negative
2. We can use any fastest path algorithm with Yen's Algorithm (UCS and A-Star)

The idea behind the Yen's algorithm is that the k-th shortest path will generally have a similar path, or contain similar subsets of paths as the shortest k-1th shortest path. As such, by going through the k-1th shortest path and iteratively removing edges to spur nodes (i.e. a node that we want to search for a new shortest path from) we can possibly find the next shortest path to the target node that is different from previously found shortest paths [1].

The time complexity of Yen's algorithm is dependent on the algorithm used for finding the fastest path. In our case we used Uniform Cost Search (UCS) for task 2 which has a time complexity of $O(b^{\frac{c}{e}})$ where *c* is the cost (in our case the distance), *b* is the branching factor of our network and *e* is the minimum cost per edge [2].

With the complexity of UCS determined, the time complexity of Yen's algorithm would be $O(KN(b^{\frac{c}{e}}))$. The *K* implies the number of times Yen's algorithm was run, while *N* is the worst case for the length of spur paths created (paths that stretch from a spur node to the target node) [1]. This time complexity however does not take into account the searching of lists of replica shortest paths. As we used lists in python the time complexity of searching a list of paths would be $O(NM)$where *N* denotes the number of currently found paths, and *M* denotes the length of each shortest path found.

In short, running Yen's algorithm can take a very long time especially for our use case where the K value cannot be predetermined and our only viable termination point would be when we've found a shortest path that meets the requirements

The implementation of the Yen's algorithm works by maintaining two lists, A which stores the shortest path from source to target node and list B that stores the candidate paths for each iteration in the algorithm.

We first find the shortest path using UCS (or A star for Task 3), then we iteratively go through each node in the shortest path and replace the edge that came before it. We then find the shortest path with this updated network graph and store it in list B.

Finally, we parse through list B to find the shortest path and append it to list A. These steps are then repeated for k number of times.

For our case we limit k to 5 in the interest of time, although the actual termination point would've been when we find a shortest path that meets the energy budget requirements - this could take a very long time as the Yen's algorithm has an exponential time complexity.

For Task 2 we manage to find a shortest path that meets the total energy cost requirements.

```
Shortest path:
1->1363->1358->1357->1356->1276->1273->1277->1269->1267->1268->1284->1283->1282->125
5->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->23
34->2329->2029->2027->2019->2022->2000->1996->1997->1993->1992->1989->1984->2001->19
00->1875->1874->1965->1963->1964->1923->1944->1945->1938->1937->1939->1935->1931->19
34->1673->1675->1674->1837->1671->1828->1825->1817->1815->1634->1814->1813->1632->16
31->1742->1741->1740->1739->1591->1689->1585->1584->1688->1579->1679->1677->104->568
0->5418->5431->5425->5429->5426->5428->5434->5435->5433->5436->5398->5404->5402->539
6->5395->5292->5282->5283->5284->5280->50.
Shortest distance: 150784.60722193593.
Total energy cost: 287931
```

5. 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)

Similar to Task 2, Task 3 requires us to find the k-th shortest path but with the restriction of using the A* search algorithm. The A* search algorithm combines both the UCS approach of exploring using the cost of adjacent nodes, as well as information outside of just the cost.

In our 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.

```
Shortest path:
1->1363->1358->1355->1274->1143->1264->1239->1237->1236->1228->1227->941->936->935->
923->930->919->927->925->924->906->901->755->677->663->660->659->366->365->364->349-
>347->345->342->320->343->297->294->295->196->202->181->182->184->215->216->217->96-
>98->221->232->222->237->236->4814->4808->4806->4804->4787->4788->4789->5149->5150->
```

```
5180->5195->5196->5197->5199->5166->5165->5163->5162->5169->5170->5172->5174->5273->
5266->5265->5264->5255->5267->5269->5257->5268->50.
Shortest distance: 224198.03681434665.
Total energy cost: 452855.
```

We can also observe that the shortest path returned by the A* search algorithm is inferior to the shortest path determined via UCS (assuming we use the given distance as the only metric).

Running the Yen's algorithm with the A* Search yielded no results within reasonable runtime and as such can be concluded to be infeasible. It is probable that the poor performance of our A* search alongside the Yen's algorithm might be because of the heuristic function used. Euclidean distance may not be the most suitable for pathfinding on a graph as two nodes may be relatively near in distance but on opposite ends of a graph.

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.

6. Conclusion

Through this project we learn that there is no one 'best' search algorithm and it all ultimately depends on the domain where the search is performed. While I started off this project expecting the A* search to outperform UCS, the results were contrary to my expectations.

It is clear that domain knowledge is important when performing search algorithms. For example, with a basic understanding of the problem we might wrongly assume that two nodes close in proximity would be relatively close on a graph, when in fact they may be very far in a graph if for example a train track separates two buildings.

It is probable that if a different heuristic function is used for the A* Search it can surpass the efficiency of UCS.

7. References

[1] En.wikipedia.org. 2021. *Yen's algorithm - Wikipedia*. [online] Available at: <https://en.wikipedia.org/wiki/Yen%27s_algorithm#Time_complexity> [Accessed 14 October 2021].

[2] Stack Overflow. 2021. *Uniform Cost Search and its Time/Space Complexity*. [online] Available at: <https://stackoverflow.com/questions/44145334/uniform-cost-search-and-its-time-space-complexity> [Accessed 15 October 2021].