Finding a Shortest Path with An Energy Budget 21S2 CZ3005 Assignment 1

Team members: Anagha S. Ani, Chin Kin Mun, Lau Chen Yi Wynne

---

1. Task approach and output explanation

For tasks 2 and 3, the energy budget is set to be 287932. For all the 3 tasks, the starting and ending nodes are set to be '1' and '50', respectively.

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)

Approach:

Implemented Dijkstra's algorithm to find the shortest path.

Created a class Node that will contain:

(1) distance: the shortest distance from the predefined source node (*S*) to the current node, initialized to -1

(2) previous_node: the current node's previous node, initialized to 'NA'

(3) visited: whether the current node has already been visited, initialized to False

To reduce unnecessary running time, the Dijkstra's algorithm will stop running once the current node is the target node (*T*) as the Dijkstra's algorithm would continue to expand all the nodes to find the shortest path of all nodes from the source node (*S*) if we do not limit it to stop running once the target node (*T*) is reached.

To extract the shortest path taken from the predefined source node (*S*) to the target node (*T*), we start from the target node (*T*) and go through all the nodes it previously visited in the dijkstra's algorithm, by using `nodes[current_node].previous_node`.

To extract the shortest distance, we simply take the shortest distance from the predefined source node (*S*) to the current node, where the current node is the target node (*T*), using `nodes[end].distance`.

Output:

```
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->20
19->2022->2000->1996->1997->1993->1992->1989->1984->2001->1900->1875->187
4->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)

Approach:

Implemented Uniform Cost Search (UCS), using a priority queue which orders elements in terms of distance in ascending order. The priority queue is initialized to contain the start node. Every child of each node in the queue is expanded and updated with the new cost and distance. These values are calculated by adding the distance/cost of the parent node and the distance/cost from the parent to the child. The energy budget is imposed as a constraint such that when the cost exceeds the budget, that particular path is not considered.

The class Node was created to contain:

(1) distance: the shortest distance from the source S to the current node

(2) previous_node: the node explored just before the current node. This is used to derive the path once the search is complete

(3) cost: the energy or cost required to get from the source S to the current node

Additionally, a dictionary dictExplored was maintained to keep track of all the nodes that have been explored, along with the parent node that led to each node. For every child of each item in the priority queue which is not yet in dictExplored, the cost is evaluated against the budget. If the cost is within the budget, the child is added into the queue so it's subsequent children can be explored as well.

A variable numNodes is used to keep track of the number of nodes expanded in order to evaluate the efficiency of the algorithm. This value is incremented every time a node is visited.

When the end node is reached, the loop is broken and the search is complete.

Output:

```
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->9
98->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->19
97->1993->1992->1989->1984->2001->1900->1875->1874->1965->1963->1964->1923-
>1944->1945->1938->1937->1939->1935->1931->1934->1673->1675->1674->1837->16
71->1828->1825->1817->1815->1634->1814->1813->1632->1631->1742->1741->1740-
>1739->1591->1689->1585->1584->1688->1579->1679->1677->104->5680->5418->543
1->5425->5429->5426->5428->5434->5435->5433->5436->5398->5404->5402->5396->
5395->5292->5282->5283->5284->5280->50

Shortest distance: 150784.60722193593

Total energy cost: 287931

Nodes Expanded: 5438
```

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)

Approach:

Implemented A* search algorithm with the heuristic function which adds the distance traveled previously and the displacement between the current node to the goal node.

Created a class called Node.

Node class Contains:

1. Distance traveled from the start
2. Displacement from the node to the goal node
3. Energy cost since the start node
4. Previous node visited to track the pathing
5. Whether it has been visited
6. Heuristic value
7. Neighbors of the node

To update the node, the conditions are that the node is (unvisited or heuristic value is lower than the (distance traveled + displacement)) and energy cost is lower than the budget.

A dictionary of nodes is created to initialize all the nodes before starting the algorithm to access the nodes to update according to the previous paragraph when needed to.

A stack is created to keep track of which node we are evaluating and to insert child nodes and pop next nodes. The stack will be initialized with the starting node before starting the algorithm. Algorithm will pop a node from the stack with the lowest heuristic and evaluate that node.

After popping a node, it will push all the neighbor nodes of that node into the stack, treat that node as the current node, and update the current node according to the condition.

If the current node is the goal node, it stops and prints all the output.

Output:

```
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->9
98->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->19
97->1993->1992->1989->1984->2001->1900->1875->1874->1965->1963->1964->1923-
>1944->1945->1938->1937->1939->1935->1931->1934->1673->1675->1674->1837->16
71->1828->1825->1817->1815->1634->1814->1813->1632->1631->1742->1741->1740-
>1739->1591->1689->1585->1584->1688->1579->1679->1677->104->5680->5418->543
1->5425->5429->5426->5428->5434->5435->5433->5436->5398->5404->5402->5396->
5395->5292->5282->5283->5284->5280->50.

Shortest distance: 150784.60722193593.

Total energy cost: 287931.

Nodes Expanded: 1901
```

2. Contributions

Task 1	Lau Chen Yi Wynne
Task 2	Anagha S. Ani
Task 3	Chin Kin Mun

3. Conclusions

Dijkstra's algorithm looks for the shortest path of every node from the source node while the Uniform Cost search looks for the shortest path between the source and the target node only. Hence the number of nodes expanded by the Uniform Cost search would be smaller than that of the Dijkstra's algorithm, if we do not limit the Dijkstra's algorithm to stop running once it reaches the target node.

The Uniform Cost search and the A* search algorithm have the same output as they both find the shortest distance from the source to the target node within an energy budget of 287932. Since the A* search evaluates each node by the sum of the displacement between the current node and the goal node, and the distance traveled, and the UCS expands and evaluate the next node with the lowest distance traveled, A* search will be able to filter out more impossible paths. Hence the A* search algorithm expands less nodes than the UCS as our results show, A* search expands 1901 nodes whereas UCS expands 5438. This makes the A* search more efficient compared to the UCS algorithm when finding the shortest distance with an energy budget .

Given that there is no energy budget, Djikstra's algorithm will expand all 264346 nodes, while UCS will expand 5647 nodes and A* Search will expand 1559 nodes. From these results, regardless of whether there is an energy budget, the Djikstra algorithm is the least efficient one, followed by UCS, and then the A* Search.

Hence, the A* search algorithm is the most efficient algorithm to search, followed by the Uniform Cost search and then the Dijkstra's algorithm.

4. References

Task 3: Math library from python is used: https://docs.python.org/3/library/math.html