



## **CZ3005 Lab 1 - Finding a Shortest Path with An Energy Budget**

### **SSP2 Group0**

#### **TABLE OF CONTENTS**

Tasks	2 - 4
Contribution	4 - 5
Conclusion	5
References	5

#### **GROUP MEMBERS**

Ma Xinyi - U2022639C

Zhang Chi - U2022109B

Zhang Meng'ao - U2023060L

# 1. Tasks

\*All of our codes are written in Python

## 1.1 Task1

### Approach:

We applied the Uniform Cost Search (UCS) algorithm and to achieve higher efficiency we implemented the priority queue using binary heaps.

Among all the searching algorithms that we had learnt, for example, Depth First Search (DFS) algorithm, Breadth First Search (BFS) algorithm, Iterative Deepening algorithm and Dijkstra's algorithm, we chose UCS for the following reasons.

First, we want to find the shortest path in terms of edge distance from the source to the destination which is an optimal problem, therefore DFS and Depth-Limited algorithm were not our ideal choices as they cannot guarantee optimality.

Second, BFS and Iterative Deepening algorithm are efficient in finding the path with the least number of edges from the source to the destination, but the graph given to us is weighted and we focus on finding the path with lowest edge distance which is not what the two algorithms target at, so these two algorithms are not our best choices.

Last, Dijkstra's algorithm is able to find the shortest distance path but it uses more memory space than UCS as it puts all vertices into the priority queue at the beginning which is very costly for a large graph as in our case.

Based on all the reasons mentioned above, we chose to apply UCS to this task. By running UCS, we calculated the shortest path as follows with the distance of 148648.637221 and energy cost of 294853.

### 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 -> 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:

148648.637221

Total energy cost:

294853

## 1.2 Task2

### Approach:

We used a variant of UCS named “constrainedUCS” in our codes since we need to keep all paths that have not already exceeded the budget. In the priority queue, instead of keeping a node's name only, we keep each node's state in a particular path: previous node's state which is like a pointer pointing to the previous node in a linked list, distance traveled, energy cost, and name. Moreover, we create a dictionary named “weights” to store (distance traveled, energy cost) sets of all the possible states for each explored node. For example, A->B, C->B are two past states for B, then “weights” need to store two different tuples (distance traveled, energy cost) for B which are obtained from different paths respectively.

When expanding a node “cur” dequeued from the priority queue, namely visiting its neighbors, at first we calculate the distance and energy cost of this neighboring node “v” assuming that we would take “cur->v”. Then we have two cases to discuss.

For the first case, “v” has been explored before and has  $\geq 1$  (distance, energy cost) record in its past states set. If its distance and energy are higher than any state's distance and energy cost of “v”'s past states that have been stored in “weights”, we can confirm that we will not adopt this path since there is a better one already. Otherwise, we will add “cur->v” state into the priority queue and also add its (distance, energy cost) into “weights”.

For the second case, “v” hasn't been explored and doesn't have any (distance, energy cost) record in its past states set. Then we can directly add “cur->v” state into the priority queue and add its first (distance, energy cost) tuple into “weights”.

By using this algorithm, we can keep all the possible paths and avoid losing any information since the original UCS only keeps one node's name and if we dequeue the node from the priority queue, we will lose this node and cannot access it from any other path at all. That's why we choose to store a node's state which carries all the information as well as the reference to the previous node's state.

By running the “constrainedUCS”, we calculated the shortest path as follows with the distance of 150335.55441905273 and energy cost of 259087

### 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 -> 2465 -> 2466 -> 2384 -> 2382 -> 2385  
-> 2379 -> 2380 -> 2445 -> 2444 -> 2405 -> 2406 -> 2398 -> 2395 -> 2397 -> 2142  
-> 2141 -> 2125 -> 2126 -> 2082 -> 2080 -> 2071 -> 1979 -> 1975 -> 1967 -> 1966  
-> 1974 -> 1973 -> 1971 -> 1970 -> 1948 -> 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:

150335.55441905273

Total energy cost:

259087

### 1.3 Task3

#### Approach:

We used the coordinates to calculate the rough distance between a node and the goal, which will be the heuristic function for our A\* algorithm. Then we use the past cost plus estimated future cost, i.e. heuristic function, as the evaluation function "f". Following a similar strategy in the constrained UCS in task 2, we use a tuple to store the information of the node in a particular path. The only difference is that we add the "f" value into the tuple and push the tuple into the priority queue, considering "f" value as priority instead of distance traveled. Now the tuple contains one more element - the "f" value.

As a result, we get the constrained optimal path given as follows, which is identical with the output in task 2.

#### Output:

Shortest path:



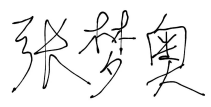
1 -> 1363 -> 1358 -> 1357 -> 1356 -> 1276 -> 1273 -> 1277 -> 1269 -> 1267 -> 1268  
-> 1284 -> 1283 -> 1282 -> 1255 -> 1253 -> 1260 -> 1259 -> 1249 -> 1246 -> 963 ->  
964 -> 962 -> 1002 -> 952 -> 1000 -> 998 -> 994 -> 995 -> 996 -> 987 -> 988 -> 979  
-> 980 -> 969 -> 977 -> 989 -> 990 -> 991 -> 2465 -> 2466 -> 2384 -> 2382 -> 2385  
-> 2379 -> 2380 -> 2445 -> 2444 -> 2405 -> 2406 -> 2398 -> 2395 -> 2397 -> 2142  
-> 2141 -> 2125 -> 2126 -> 2082 -> 2080 -> 2071 -> 1979 -> 1975 -> 1967 -> 1966  
-> 1974 -> 1973 -> 1971 -> 1970 -> 1948 -> 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:  
150335.55441905273

Total energy cost:  
259087

## 2. Contribution

Name	Individual Contribution to Lab 1	Percentage of Contribution	Signature
Ma Xinyi	Analysis of the problem and discuss the solutions. Contribute in writing the source code and final report.	33.3%	
Zhang Chi	Analysis of the problem and discuss the solutions. Contribute in writing the source code and final report.	33.3%	
Zhang Mengao	Analysis of the problem and discuss the solutions. Contribute in writing the source code and final report.	33.3%	

## 3. Conclusion

We have gained a deeper understanding of UCS and A\* algorithm, We have learned how to design data structures and how to modify the algorithms that we learned before to fulfill our different requirements, namely solving the shortest path problem with a budget constraint.

## 4. References

Wu, G. (2021, August 25). *Comparison between uniform-cost search and Dijkstra's algorithm*. Baeldung on Computer Science. Retrieved March 5, 2022, from <https://www.baeldung.com/cs/uniform-cost-search-vs-dijkstras>