NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

CZ3005 Artificial Intelligence Assignment 1

Cheong Jun Hao (U1921202F)
Chia Jia Tian (U1921632K)
Quek Lin Hui (U1921918F)

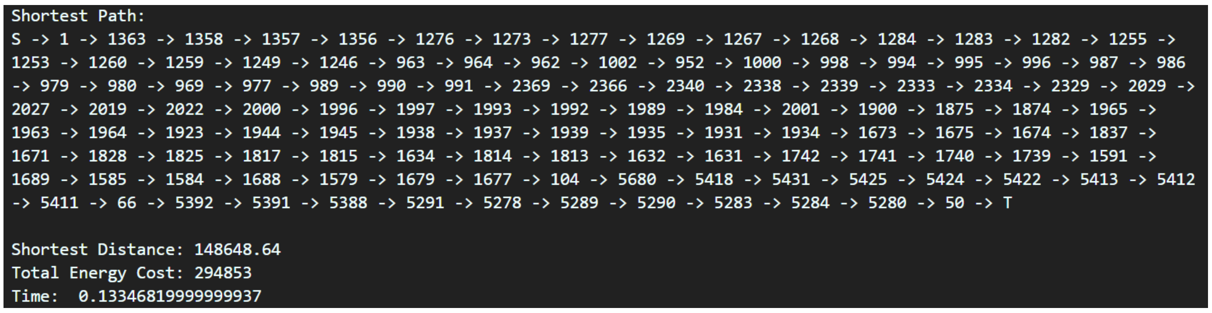
Task 1

Approach

To find the shortest path to the goal node, we chose uniform-cost search (UCS) as our algorithm of choice. The graph is undirected and weighted without negative weights. UCS choses which node on the frontier to expand based on their path cost from the start node, with the node with the lowest past cost being expanded first. When a node is expanded, it is removed from the frontier and its neighbouring nodes will be added. This is implemented using a priority queue, giving the maximum priority to the node with the lowest cumulative cost. This would give the optimal solution since at every state, the path with the least cost is chosen to be expanded. This cycle of expansion continues until the current node of highest priority in the priority queue is the end node and the function returns the path from the start to the end node which would have the lowest cost.

We noted that since UCS does not take into account the number of steps taken, it may repeatedly visit the same node in the graph which leads to an infinite loop and prevents the end node from ever being reached. To counter this problem, we made use of a set to take note of the nodes that have been expanded before and avoid revisiting the same node again.

Output



```
Shortest Path:
S -> 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 -> 5424 -> 5422 -> 5413 -> 5412
-> 5411 -> 66 -> 5392 -> 5391 -> 5388 -> 5291 -> 5278 -> 5289 -> 5290 -> 5283 -> 5284 -> 5280 -> 50 -> T

Shortest Distance: 148648.64
Total Energy Cost: 294853
Time:  0.13346819999999937
```

Figure 1. Output from the relaxed UCS

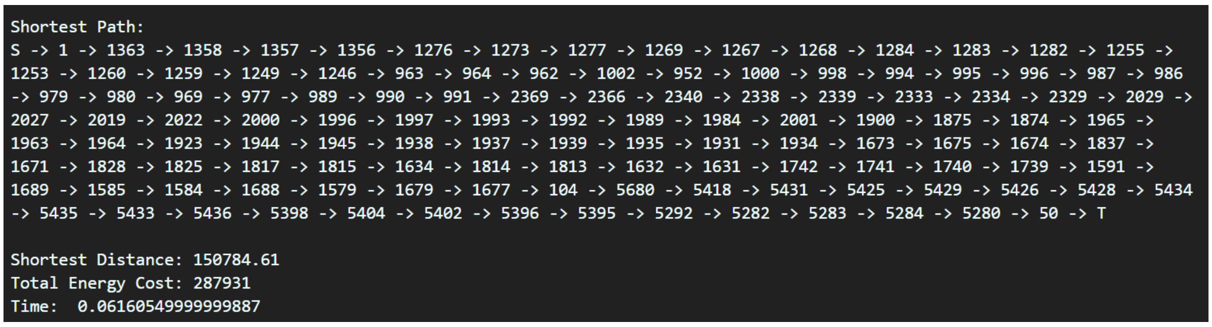
Task 2

Approach

To complete task 2, we performed a uniform-cost search as used in Task 1 but altered with an energy cost check. In this iteration of uniform cost search, both the energy and distance is accumulated, a check is done to ensure that the energy cost that will be incurred on the path from the start node up till the current node is lower than the energy constraint of 287932 before adding it to the frontier by putting it into the priority queue. From there, the node that has the shortest accumulated path cost will be expanded until the end node is reached.

To avoid redundant expansion, nodes which require more energy than the energy budget will not be considered for expansion. Additionally, If the current node does not have any neighbour node that it can move to without exceeding the energy constraint, the current node will also not be added to the set of visited nodes yet. This allows for paths that are longer but have lower energy cost to be explored and expanded as well to prevent the situation where no path is returned as the current node is prevented from being in other paths that satisfy the energy constraint. Thus, this guarantees a better chance of a shortest possible path that satisfies the energy constraint from being returned at all if it exists. Lastly, similar to task 1, the problem of infinite loop still exists and hence is resolved in the same way.

<u>Output</u>



```
Shortest Path:
S -> 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 -> T

Shortest Distance: 150784.61
Total Energy Cost: 287931
Time:  0.06160549999999887
```

Figure 2. Output from the constrained UCS

The shortest path returned in this task, seen in Figure 2, satisfies the energy constraint but is longer than the path returned in Task 1.

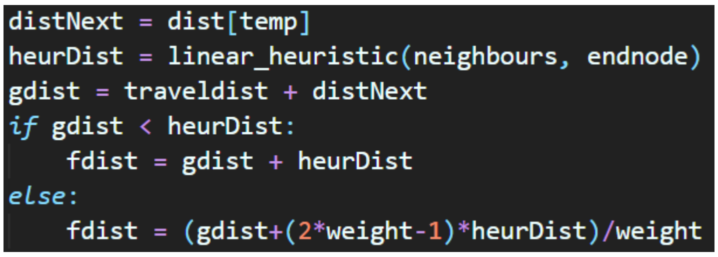
Task 3

<u>Approach</u>

To develop an A* search algorithm, similar to the approach in UCS, the frontier is stored in a priority queue and the node with the highest priority, in other words the smallest f(n) value, that has not been visited before will be expanded. The f(n) is the sum of the actual path cost incurred to reach the current node from the start node and the heuristic value which is the predicted cost to reach the goal.

Given that there were no obstacles in the path, we concluded that the euclidean distance would be a suitable heuristic, where the heuristic function will return the node's straight line distance to the end node calculated using their coordinate values. Having a shorter euclidean distance to the goal node would suggest that the node would be closer to the goal node and thus taking a bigger step towards the goal node and avoid short paths moving away from the goal node as well.

To further optimise our A* search, we decided to add weights to the actual path cost and heuristic cost. In Necessary and Sufficient Conditions for Avoiding Reopenings in Best First Suboptimal Search with General Bounding Functions (Chen & Sturvetant, 2021), two approaches are presented: one which decreases the weight when approaching the goal and one which increases the weight when approaching the goal. In the paper, they found that increasing the weight produced better results. Hence, our group decided to implement a weighted A* search algorithm, as seen in Figure 3, that increases the weight when nearing the goal node based on the paper.



```
distNext = dist[temp]
heurDist = linear_heuristic(neighbours, endnode)
gdist = traveldist + distNext
if gdist < heurDist:
    fdist = gdist + heurDist
else:
    fdist = (gdist+(2*weight-1)*heurDist)/weight
```

Figure 3. Weighted A* Function

Since we have chosen to use a weighted A* search algorithm, it is essential to use an appropriate weight to produce the optimal results. In our project, we have defined the optimal results as having the shortest run time while still producing the correct shortest path. To determine the optimal weight value, we wrote a function to test a range of possible weight values and plotted their corresponding runtime on a graph.

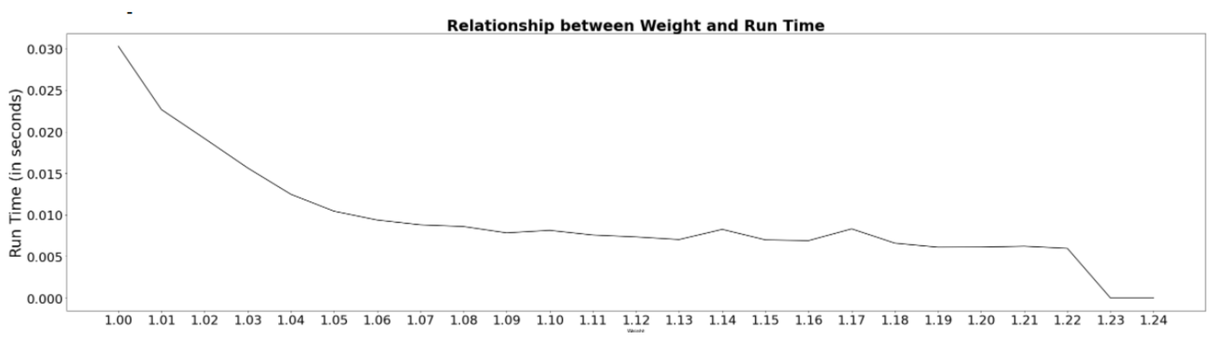
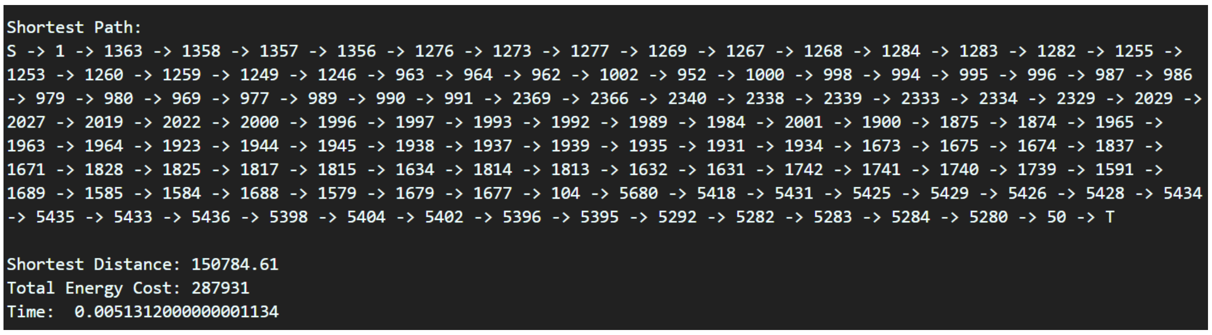


Figure 4. Graph of run time of code against weight used in the algorithm

From the Figure 4 above, we observe that the algorithm generally performs better as the weight increases and would achieve the shortest runtime when the weight is set at 1.22. We also observed that the algorithm would run indefinitely if the weight is raised beyond 1.22 and hence we set the runtime to 0 should the algorithm run for more than a second although no output is returned. The heuristic weight should be selected such that the heuristic is still an underestimate of the actual distance of the route. This is because A* is only complete and optimal on graphs that are locally finite where the heuristics are admissible and monotonic (Mayefsky, Anene, & Sirota, 2003). Therefore, we concluded that the optimal weight is 1.22 for our data set.

<u>Output</u>



```
Shortest Path:
S -> 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 -> T

Shortest Distance: 150784.61
Total Energy Cost: 287931
Time:  0.005131200000001134
```

Figure 5. Output from weighted A* search

The output from the search in Figure 5 is exactly the same as the output from the constrained UCS as seen in Figure 2 as this is the optimal path given the energy constraint since A* and UCS are both optimal. Additionally, the run time for A* is approximately 10 times shorter than that of the UCS which shows that the heuristic function we have chosen is good which leads to the significant time saving.

Conclusion

Through this project, in addition to learning to implement multiple search algorithms, we also gained experience in trying to optimise our algorithm. We were able to better understand the impact of weight of the heuristic function on the performance of the search. For example, we learnt that according to *When does weighted A* fail?,* "the weight degrades performance share a common trait: the true distance from a node to a goal, defined as d*(n), correlates very poorly with h(n)", in our case, h(n) has a strong correlation with the true distance from the node to a goal and therefore increasing the weight helped our algorithm run faster. However, we understand that this might not be the case in other scenarios. We also gained a greater sense of appreciation for search algorithms in solving the problems of our daily lives like in this case the shortest distance from node X to node Y in the New York City instance.

Contribution

Member	**Contributions**
Cheong Jun Hao	Q1, Q2, Q3, Report
Chia Jia Tian	Q1, Q2, Q3, Report
Quek Lin Hui	Q1, Q2, Q3, Report

References

Chen, J., & Sturtevant, N. R. (2021). Necessary and Sufficient Conditions for Avoiding Reopenings in Best First Suboptimal Search with General Bounding Functions. Retrieved October 5, 2021, from https://webdocs.cs.ualberta.ca/~nathanst/papers/chen2021general.pdf.

Wilt, C., & Rum, W. (2012). When does weighted A* fail? - computer science. Retrieved October 5, 2021, from https://www.cs.unh.edu/~ruml/papers/wted-astar-socs-12.pdf.

Mayefsky, E., Anene, F., & Sirota, M. (2003). *ALGORITHMS - A**. Algorithms. Retrieved October 5, 2021, from https://cs.stanford.edu/people/eroberts/courses/soco/projects/2003-04/intelligent-search/astar.html.