CZ3005 Artificial Intelligence

Lab Assignment 1

Submitted by: Bryan Chatsirichai Daruenpat
Matriculation Number: U2022154D
Lab Group: BCT2

1. Introduction

This serves as the report for CZ3005 Artificial Intelligence Lab Assignment 1. This assignment consists of three tasks, which will be elaborated on in their respective sessions in the report. Each session consists of the output, an explanation of the approach used to tackle the task, explanation of why certain algorithms were used and not others.

The problem given is derived from the NYC instance which is a modified version of the 9th DIMACS implementation challenge. The graph given could be assumed to be a directed weighted graph with two different sets of weights for each edge, distance cost, and energy cost.

For this assignment, an assumption was made to find the shortest path for all tasks if applicable.
Task 1 is to solve a relaxed version of the NYC instance only considering distance cost. Therefore, equivalent to solving the shortest path problem. However, for tasks 2 and 3, given a certain energy constraint, one must try to find a path or shortest path possible if able from a given start node to a given end node. For both tasks 2 and 3, task 2 is to use uninformed search strategies while task 3 is to use informed search strategies respectively.
Both task 1 and task 2 will be done using uninformed search, **using only the information available in the problem definition.** While task 3 must be done using informed search strategies**, use problem-specific knowledge to guide the search.**

Uninformed Algorithms discussed in lectures.

Criterion	Breadth-first search (BFS)	Uniform-cost search (UCS)	Depth-first search (DFS)	Depth-limited search (DLS)	Iterative Deepening
Time	b^d	b^d	b^m	b^l	b^d
Space	b^d	b^d	bm	bl	bd
Optimal	Yes, when all steps cost equally	Yes	No	No	Yes, when all steps cost equally
Complete	Yes	Yes	No	Yes, if $l \geq d$	yes

Table 1. Uninformed search strategies

b: max branching factor of the search tree
d: depth of least-cost solution
m: maximum depth of state-space
l: limit bound depth

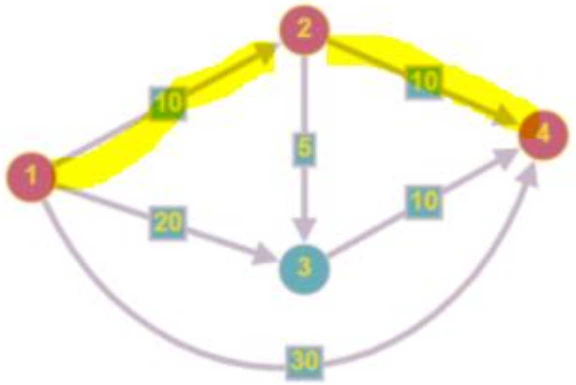
Table 1.1 Legends for Table 1

2. 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.**

The goal of task 1 is to find the shortest path from node 1 to target node 50.
From table 1 and 1.1,

- Breadth-first search is optimal if all steps are equal thus rejected as we are using a weighted graph to find the shortest distance.
- Depth-first not optimal and potentially not complete with infinite depth space or finite-depth spaces with loops therefore also reject.
- Similarly same for Depth-limited search not being optimal as well and even if it may be complete if $l \geq d$ but this NYC instance has too many nodes to find/guess an appropriate depth.
- Depth-limited search combines depth-first search's space-efficiency and breadth-first search's fast search (for nodes closer to root). Optimal if all steps are equal also.

Therefore, the best option to be use on a weighted graph is Uniform-cost search UCS / Dijkstra.



Example of UCS on a simple graph

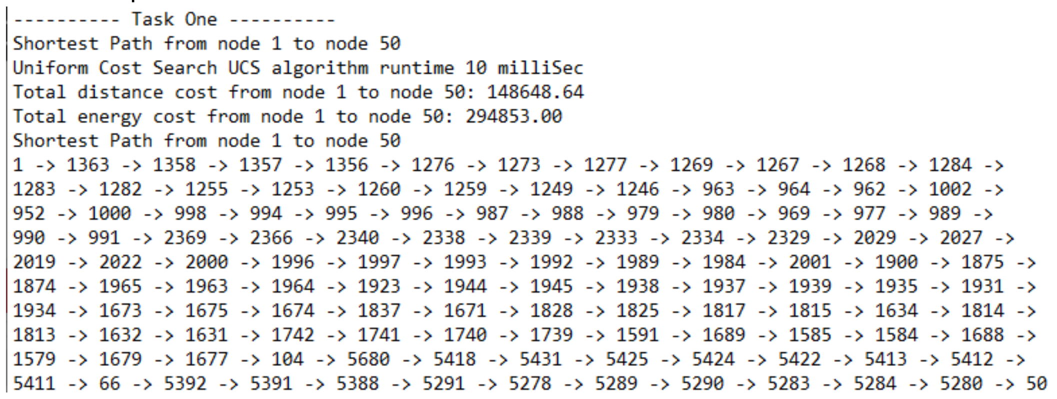
Explanation: Uniform-Cost Search is like Dijkstra's algorithm. Given a starting state visit the adjacent states and choose the least costly node then again choose the next least costly node from all un-visited and adjacent states of the visited states, in this way try to reach the goal node (once reach goal node, terminate). A priority queue is used to maintain which un-process node to visit next based on the shortest distance from the start node to the current node being processed.
In short,

1. Pick the min value node which is unprocessed.
2. Mark this node as processed.

3. Update all adjacent vertices if cost[u] + distCost[uv] < cost[v] else skip

In addition, an array or HashMap is used to keep track of which node has been visited.
Each node will have a reference to its ‘parent’ node to form the shortest path from start node to end node. Note: parent reference will be updated if a shorter path from start node to a node can be found.

Task 1 output:



```
---------- Task One ----------
Shortest Path from node 1 to node 50
Uniform Cost Search UCS algorithm runtime 10 milliSec
Total distance cost from node 1 to node 50: 148648.64
Total energy cost from node 1 to node 50: 294853.00
Shortest Path from node 1 to node 50
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
```

Overall Time complexity: $O(b^d)$,Space complexity: b^d

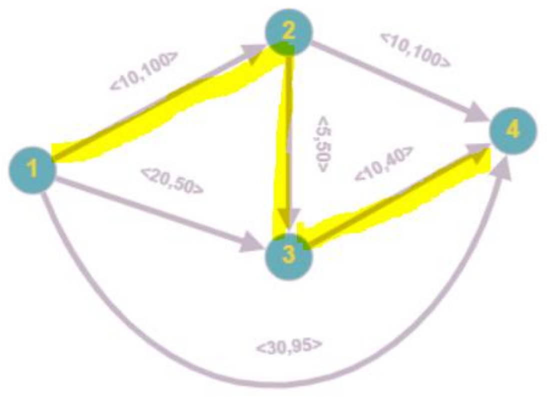
3. Task 2: **You will need to implement an uninformed search algorithm (e.g., the DFS, BFS, UCS) to solve the NYC instance.**

Task 2 could be trying to find a path (shortest if possible) with resource constraints, <distCost,energyCost>.Approach taken to tackle this task was to use a modified version of UCS. Same as standard UCS, priority queue is used to select which node that has a shortest distance from start node to it from the queue.But do not keep track of which node visited(relaxed) as could try again from visited node an alternate path, assuming allowed re-visit to a node, as could have better path-energy cost.

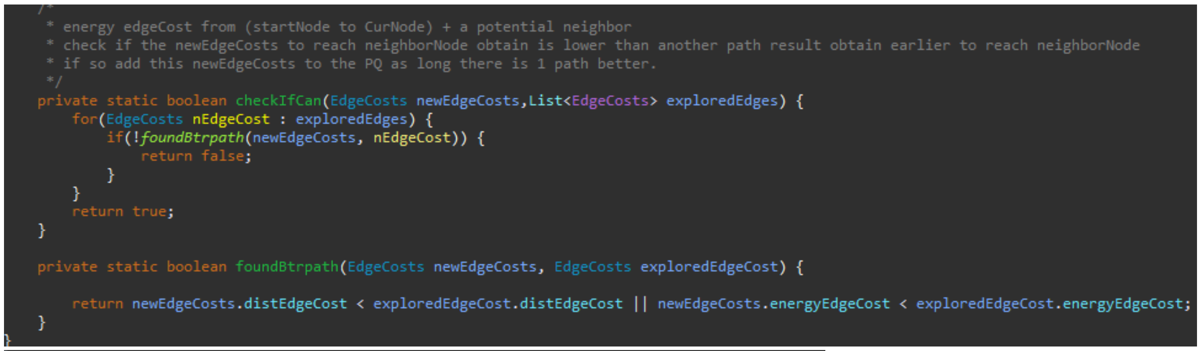
In short,

1. edgeCost from (startNode to CurNode) + a potential neighbor
2. check if the newEdgeCosts to reach neighborNode obtain is lower than another path result obtain earlier to reach neighborNode. (Note: either the distance cost is lower, or energy cost is lower)
3. If so, add this newEdgeCosts to the PQ if there is 1 path better.

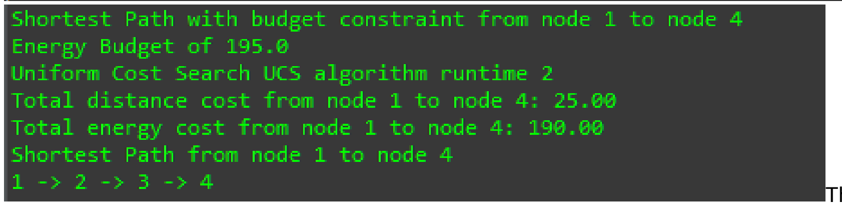
Therefore this modified UCS algorithm still tries to find shortest path, but shortest path may not be within energy budget thus we allow re-visit to already processed (relax) nodes. In addition each node will have a reference to its ‘parent’ node in order to form the shortest path from start node to end node. Note: parent reference will be updated if a shorter path and within energy budget from start node to a node can be found.



Example of modified UCS on a simple graph



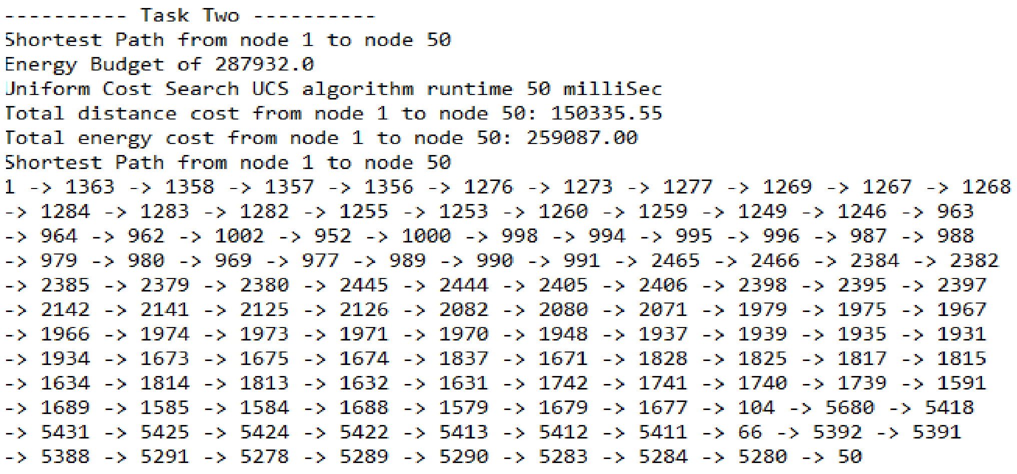
```java
    /*
     * energy edgeCost from (startNode to CurNode) + a potential neighbor
     * check if the newEdgeCosts to reach neighborNode obtain is lower than another path result obtain earlier to reach neighborNode
     * if so add this newEdgeCosts to the PQ as long there is 1 path better.
     */
    private static boolean checkIfCan(EdgeCosts newEdgeCosts,List<EdgeCosts> exploredEdges) {
        for(EdgeCosts nEdgeCost : exploredEdges) {
            if(!foundBtrpath(newEdgeCosts, nEdgeCost)) {
                return false;
            }
        }
        return true;
    }

    private static boolean foundBtrpath(EdgeCosts newEdgeCosts, EdgeCosts exploredEdgeCost) {

        return newEdgeCosts.distEdgeCost < exploredEdgeCost.distEdgeCost || newEdgeCosts.energyEdgeCost < exploredEdgeCost.energyEdgeCost;
    }
}
```



```
Shortest Path with budget constraint from node 1 to node 4
Energy Budget of 195.0
Uniform Cost Search UCS algorithm runtime 2
Total distance cost from node 1 to node 4: 25.00
Total energy cost from node 1 to node 4: 190.00
Shortest Path from node 1 to node 4
1 -> 2 -> 3 -> 4
```

The solution to the simplified graph.

Task 2 output:



```
---------- Task Two ----------
Shortest Path from node 1 to node 50
Energy Budget of 287932.0
Jniform Cost Search UCS algorithm runtime 50 milliSec
Total distance cost from node 1 to node 50: 150335.55
Total energy cost from node 1 to node 50: 259087.00
Shortest Path from node 1 to node 50
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
```

Overall Time complexity: $O(b^d)$,Space complexity: b^d, asymptotic time and space complexity will be the same as normal UCS, but since we allow repeated nodes to be visited the 'constant factor would be higher' thus taking longer time and more space, but overall complexity remains the same

4. 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.

Why use A* search (informed search).

To approximate the shortest path, an uninformed search may take a long time to find an optimal path but using informed search may take a shorter amount of time to find a good enough solution. Using problem-specific knowledge to guide the search, thus **'usually'** more efficient.

What is A* search :

Uniform-cost search

- *g(n):* cost to reach n (Past Experience)
- optimal and complete, but can be very inefficient

Greedy search

- *h(n):* cost from n to goal (Future Prediction)
- neither optimal nor complete, but cuts search space considerably

Combine Greedy search with Uniform-Cost search

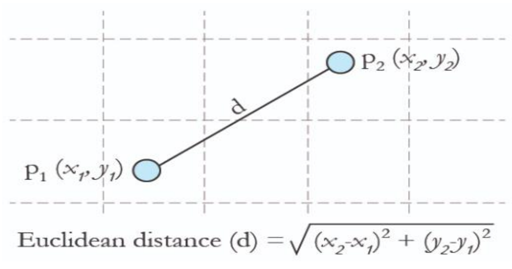
Evaluation function: f(n) = g(n) + h(n)

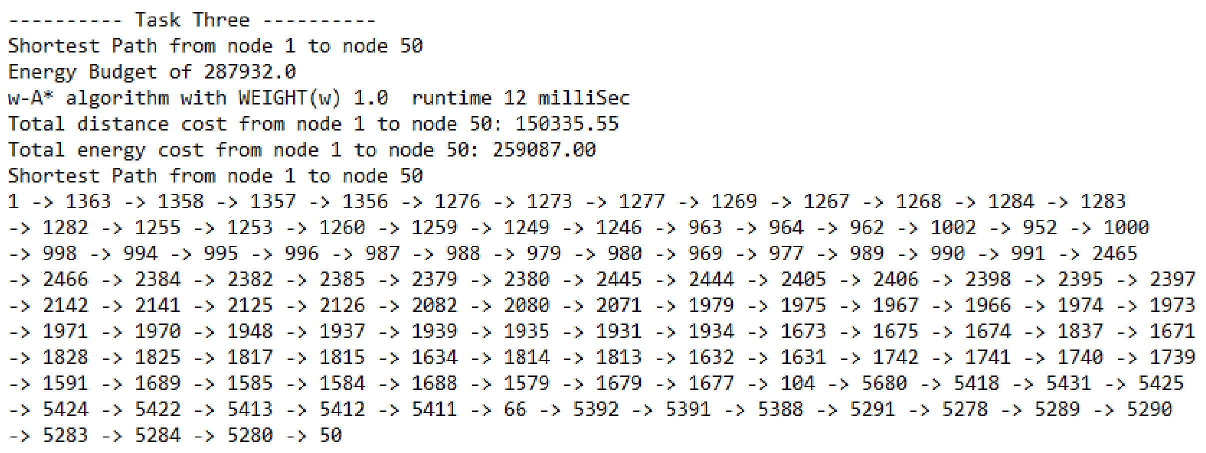
- *f (n):* estimated total cost of path through *n* to goal (Whole Life)
- If g = 0 a greedy search, If h = 0 a uniform-cost search
- Function A* Search(problem) returns solution, Return Best-First-Search(problem, g+h)

For task 3 implemented usin**g w-A* search algorithm** that place more emphasis on the heuristic function for any w >= 1 Evaluation **function: f(n) = g(n) + w * h(n)**.

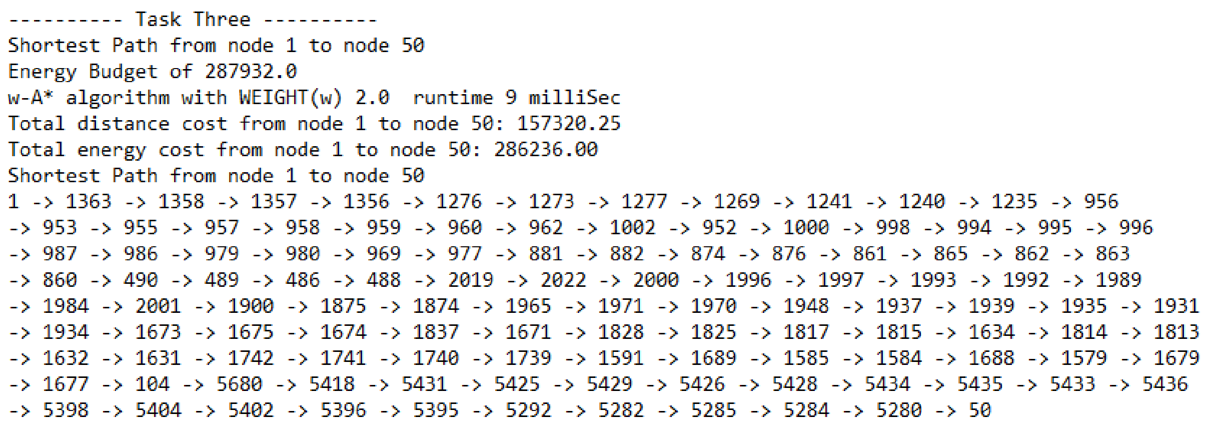
- Actual distance obtained g(n)
- Heuristic h(n) will be estimated using Euclidean distance (distance between the current cell and the goal cell using the distance formula).Note: (x,y) will be obtain using the Coordinates JSON file

h(n) = $\sqrt{(current\_Node.x - goal\_Node.x)^2 + (current\_Node.y - goal\_Node.y)^2}$

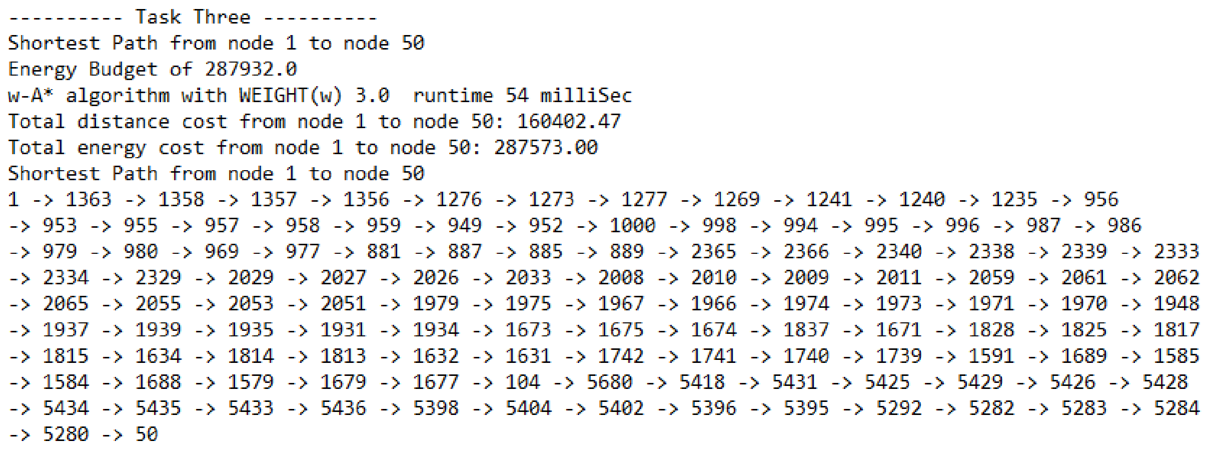




```
---------- Task Three ----------
Shortest Path from node 1 to node 50
Energy Budget of 287932.0
w-A* algorithm with WEIGHT(w) 1.0  runtime 12 milliSec
Total distance cost from node 1 to node 50: 150335.55
Total energy cost from node 1 to node 50: 259087.00
Shortest Path from node 1 to node 50
1 -> 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
```



```
---------- Task Three ----------
Shortest Path from node 1 to node 50
Energy Budget of 287932.0
w-A* algorithm with WEIGHT(w) 2.0  runtime 9 milliSec
Total distance cost from node 1 to node 50: 157320.25
Total energy cost from node 1 to node 50: 286236.00
Shortest Path from node 1 to node 50
1 -> 1363 -> 1358 -> 1357 -> 1356 -> 1276 -> 1273 -> 1277 -> 1269 -> 1241 -> 1240 -> 1235 -> 956
-> 953 -> 955 -> 957 -> 958 -> 959 -> 960 -> 962 -> 1002 -> 952 -> 1000 -> 998 -> 994 -> 995 -> 996
-> 987 -> 986 -> 979 -> 980 -> 969 -> 977 -> 881 -> 882 -> 874 -> 876 -> 861 -> 865 -> 862 -> 863
-> 860 -> 490 -> 489 -> 486 -> 488 -> 2019 -> 2022 -> 2000 -> 1996 -> 1997 -> 1993 -> 1992 -> 1989
-> 1984 -> 2001 -> 1900 -> 1875 -> 1874 -> 1965 -> 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 -> 5429 -> 5426 -> 5428 -> 5434 -> 5435 -> 5433 -> 5436
-> 5398 -> 5404 -> 5402 -> 5396 -> 5395 -> 5292 -> 5282 -> 5285 -> 5284 -> 5280 -> 50
```



```
---------- Task Three ----------
Shortest Path from node 1 to node 50
Energy Budget of 287932.0
w-A* algorithm with WEIGHT(w) 3.0  runtime 54 milliSec
Total distance cost from node 1 to node 50: 160402.47
Total energy cost from node 1 to node 50: 287573.00
Shortest Path from node 1 to node 50
1 -> 1363 -> 1358 -> 1357 -> 1356 -> 1276 -> 1273 -> 1277 -> 1269 -> 1241 -> 1240 -> 1235 -> 956
-> 953 -> 955 -> 957 -> 958 -> 959 -> 949 -> 952 -> 1000 -> 998 -> 994 -> 995 -> 996 -> 987 -> 986
-> 979 -> 980 -> 969 -> 977 -> 881 -> 887 -> 885 -> 889 -> 2365 -> 2366 -> 2340 -> 2338 -> 2339 -> 2333
-> 2334 -> 2329 -> 2029 -> 2027 -> 2026 -> 2033 -> 2008 -> 2010 -> 2009 -> 2011 -> 2059 -> 2061 -> 2062
-> 2065 -> 2055 -> 2053 -> 2051 -> 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 -> 5429 -> 5426 -> 5428
-> 5434 -> 5435 -> 5433 -> 5436 -> 5398 -> 5404 -> 5402 -> 5396 -> 5395 -> 5292 -> 5282 -> 5283 -> 5284
-> 5280 -> 50
```

For task 3, using the w-A* search, with w(weights) 1,2,3. When w = 1, we got similar results as obtained in task 2, same distance and energy cost from node 1 to node 50 but with a much faster time. However, when w increased to 2, the time for w-A* search decreased a bit but the cost for both distance and energy became worse. This is further confirmed when w = 3 which took the longest and had the highest distance and energy cost. As a result, for w-A* search algorithm to return the shortest path (runtime time), must use a heuristic that never overestimates the cost.
In conclusion A* search could potentially get a good enough solution (sometimes optimal) in a shorter amount of time compared to uninformed search algorithms. In this case, different weights could speed up or slow down the search time and affect the distance and energy cost obtain as different paths from the start node to the goal node are chosen. Thus, we need to be adjusted to get desired tradeoff between runtime time to distance and energy costs.

Task(s), Node 1 to Node 50	Runtime (Milli sec)	Distance cost	Energy cost
Task 1 (UCS)	10	148648.64	294853.00
Task 2 (Modified UCS)	51	150335.55	259087.00
Task 3 w-A*, w = 1	12	150335.55	259087.00
Task 3 w-A*, w = 2	9	157320.25	286236.00
Task 3 w-A*, w = 3	54	160402.47	287573.00

Task(s) result table

5. Conclusion

After doing this lab assignment, I have learned that there is no one 'best' search algorithm, it depends on the domain where the search is performed. Do we need to have an absolute optimal solution or a good enough solution when taking into consideration the time needed to carry out the runtime execution. For the w-A* search, with a good heuristic, significant savings are still possible compared to uninformed search, **sometimes** may get an optimal solution but mostly we will get a good enough solution for time trade-off, however, we can still get an un-optimal solution which is not considered good enough as well if we have a bad heuristic or have a heuristic that overestimates the cost. Therefore, by having a good heuristic it could be possible for A*(informed search) to get the optimal solution same as UCS (uninformed search) but with better efficiency, runtime execution time, etc.…

Basis of comparison	Uninformed search	Informed search
Basic knowledge	No use of knowledge	Uses knowledge to find the steps to the solution.
Efficiency	Efficiency is mediatory	Highly efficient as it consumes less time and cost. (In most cases)
Cost	Comparatively high	Low (In most cases)
Performance	Speed is slower than the informed search.	Finds the solution more quickly. (In most cases)

References

Ahle, T. (2012, January 19). *stackoverflow*. From https://stackoverflow.com/questions/8681648/shortest-paths-with-resource-constraints

Andrew1234. (2021, August 25). *geeksForgeeks*. From https://www.geeksforgeeks.org/uniform-cost-search-dijkstra-for-large-graphs/

GeeksforGeeks. (2022, February 6). *geeksforgeeks*. From https://www.geeksforgeeks.org/a-search-algorithm/

Vinita. (2019, July 3). *intellipaat*. From https://intellipaat.com/community/3654/what-is-the-difference-between-informed-and-uninformed-searches#:~:text=An%20uninformed%20search%20is%20a,current%20state%20to%20the%20goal.&text=Uses%20knowledge%20to%20find%20the%20steps%20to%20the%20solution

Williamfiset. (2020, February 3). *github*. From https://github.com/williamfiset/Algorithms/blob/master/src/main/java/com/williamfiset/algorithms/graphtheory/DijkstrasShortestPathAdjacencyList.java