CSC 584 Building Game AI
Homework 3

hbharat2

Question 1:
The given task involves creating weighted digraphs for subsequent use in the following parts of the assignment. Creating the first graph involves airplane routes from major routes in the USA. The airport list[1] and the corresponding visualization[2] are in the appendix. The first graph simulates a typical grid type setting and everything is connected to just a few more components and those components carry forward all the other connections to the other nodes. This helps observe properties such as fringe and fill in action. The second graph is created using a python program submitted as a part of the files. Some key metrics that can be extracted from the large graph are {'Number of vertices': 20000, 'Number of edges': 84048, 'Number of connected components': 1, 'Average degree': 8.4048}. The graph was created using the networkx module in python and a rand function for the weights.

Question 2:
The given task involves implementing Dijkstra's and A star search algorithms on the given graphs. Let us go over some key finds we have uncovered with the data presented in the appendix. Overall results point to the fact that A Start is faster, considering heuristic calculations are optimized.
Key Find 1: The shortest path and shortest path costs are consistent in both Dijkstras;' and A star showing accuracy in both the algorithms, a basic requirement for further analysis.[4][5][6][7]
Key Find 2: Runtime grows almost linearly for Dijkstras'[4][5] and seems reasonable for 20k nodes[7][8] in the second graph. The fringe and fill increase considerably pointing towards the amount of states needed to be stored.
Key Find 3: A* processes significantly fewer nodes than Dijkstra on the large graph (608 [8] vs 1216)[5] , a clear efficiency gain! Fringe and fill still grow, but slower than in Dijkstra. This is because the graph is connected and sparse and there are many candidate nodes to manage. This is reflected by the number of updates needed to be done to the priority queue.
Key Find 4: For the small graph - Fringe and fill still grow, but slower than in Dijkstra. (218 [5] Dijkstra vs 269 [8] A star) but when we come to the larger graph A* is dramatically more efficient in nodes processed(1648 Dijkstra[5] vs 234 A star [7][8]), but it's slower in execution time (7ms [5] vs 2712 ms [9]). This could be possible due to reasons I think being the heuristic calculation overhead at each point. It could also be due to the priority queue behaviour because the queue may contain fewer nodes, but deeper heaps or more frequent updates could slow down operations.

Although the fringe size was similar in both algorithms, A* was more focused in the way it explored nodes, thanks to the guidance from its heuristic. Overall, these results show that A* can scale better in larger graphs when using a well-designed heuristic. However, its actual performance may still be affected by how much time it takes to calculate the heuristic and how the algorithm is implemented.

Question 3:+
The given task involves implementing 2 different heuristics for the A star search algorithm. The 2 heuristics I have used in this scenario are:
Minimum Outgoing Edge Heuristic (Greedy) : $h(n) = min\,(v\,\varepsilon\,E)\;edge\;weight$, which takes the local minimum outgoing edge from every node and applies it as the heuristic value at that node.
Landmark based heuristic :
$h(n) = |dist(landmark -> goal) - dist\,(landmark -> node)|$ where landmarks are randomly selected nodes acting as a cluster representative. Carefully choosing said landmark

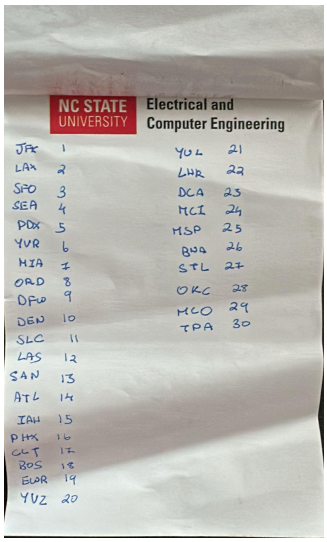
node will help make all computations simpler but since it takes a lot of preprocessing I have selected the landmark nodes at random, considering it is a densely populated graph and that the choice of landmark has a very minimal probability of being a edge node or a sparse node with lesser than average degree of connections to other nodes. Let us go over some finds. We immediately notice that both heuristics give us the same answer thus proving the basic qualifications to start comparing them are valid. (They are the same as the Dijkstra' paths too). Landmark heuristics need well-distributed and connected landmarks. We get 5 overestimations in the small graph implying the section is poor in smaller graphs with partial disconnections.
We have calculated a focus ratio which is calculated as the $focus\ ratio\ =\ number\ of\ path\ nodes\ expanded\ /\ total\ expanded\ nodes$. This helps us get an idea of any wasteful exploration done and whether the heuristics guide the search optimally. The ratio is almost 33 to 38% [7] for the small graph suggesting decent coverage and decent cutting down of wasteful explorations. Whereas in the larger graph it is very minimal (less than 5%) [8][9] suggesting that both the heuristics need to be improved and landmark selection needs to be improved. When we compare both the heuristics we notice that the minimum outgoing edge which was our safe and admissible heuristic performs equal to landmark based heuristic in terms of the focus ratio and better in terms of number of nodes processed and runtime.
We also notice that the Landmark based heuristic is improved and behaves much better at scale in the bigger graph with no overestimation which is a very good sign !! While absolute performance worsens (runtime, nodes processed), this is expected because the graph is ~300x larger. The fact that A* with a landmark that doesn't fail and still finds the path optimally and has zero overestimation means that the landmark based heuristic scales better than they begin. We could say it scales in safety but not sharpness considering the perfect admissibility but lesser ratio of nodes on path vs the total nodes explored for the A star search.

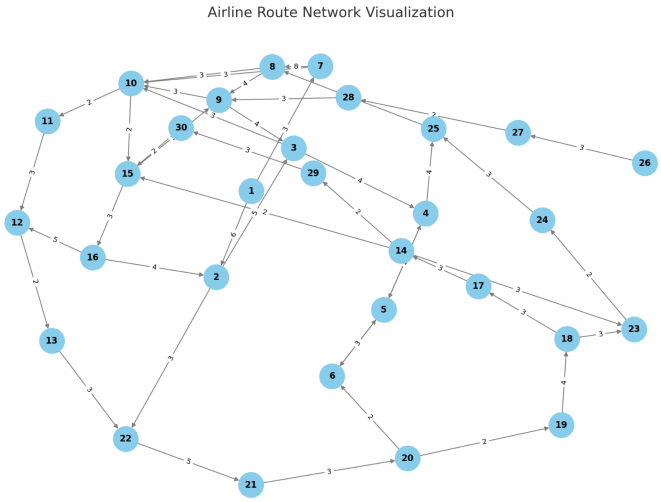
Question 4:
The given task involves pathfinding and path following algorithms rendered in SFML along with A star search. The given task was quite tricky and the implementation I have chosen is as below. The window is divided into a 40 x 30 grid where each of the grid cells can be considered a node in the graph. Each cell size is approximately 20 x 20 pixels which I thought was enough of a resolution to set the way considering the boids size. Once the path is returned from A start the path is quantized to help guide the boids movement.The A star algorithm applied here helps find the shortest path from the boid to the click destination.
The algorithm taught in class has been implemented where a target is selected which is further along the path than the mapped point by some distance. This helped give the boid some human-like behaviour as it would anticipatorily slow down before a turn without me giving any explicit directions in the program to do so.
Along with the obvious process of tweaking the parameters (especially my velocity) a major problem encountered was the lack of initial velocity or acceleration to get the boid moving as the next node was very nearby and thus the amount of acceleration to be put and neglected cancelled out each other, also the fact that the radii of satisfaction and deceleration coincided. This had to be tackled by taking the next node and not the immediate node in the path and then appropriately tweaking the parameters and giving a look ahead kind of feeling.
Raw steering outputs created jittery, unnatural movement with the boid. Low-pass filtering of angular velocity dramatically improved orientation stability.
One inference I have made from implementing this algorithm is the trade-off between precise path following and smooth fluid like movement of the boid which is appealing to the human eye. In all practical applications I feel like fluid-like movement even when there is a little straying of path is more preferable than behaviours that might involve sharp twists and turns like precise path following.
Small changes to maxVelocity or maxAcceleration dramatically affected movement quality.

The relationship between linear and angular parameters was critical - mismatched values caused erratic behavior and this was fixed with insights from the previous homeworks relationship and ratios developed between the parameters.
Satisfaction radius needed careful tuning - too small of a radius caused oscillation and too large of a radius caused corner-cutting and abrupt / erratic behaviour.Deceleration radius needed to be proportional to maximum velocity.
A test was also carried out by tweaking the values of max velocity and max Acceleration to higher numbers. One observation made when doing these was that if we increase these to high numbers, the boid is bound to collide with the obstacles as there is not enough braking power set. The other alternative is to implement obstacle avoidance, but a finely dense graph might also solve this issue .

APPENDIX

[1][2]





30 nodes
67 edges

[3]

```
hari@Haricharan:~/Game_AI/hw3$ ./dijkstra
Enter the source node: 1
Shortest distances from node 1:
Node 29 : 31.000000
Node 14 : 29.000000
Node 2 : 6.000000
Node 15 : 8.000000
Node 3 : 11.000000
Node 5 : 18.000000
Node 7 : 3.000000
Node 8 : 11.000000
Node 9 : 11.000000
Node 10 : 6.000000
Node 11 : 8.000000
Node 12 : 10.000000
Node 4 : 15.000000
Node 30 : 34.000000
Node 1 : 0.000000
Node 16 : 11.000000
Node 13 : 12.000000
Node 22 : 9.000000
Node 21 : 14.000000
Node 20 : 17.000000
Node 19 : 19.000000
Node 18 : 23.000000
Node 17 : 26.000000
Node 23 : 26.000000
Node 24 : 28.000000
Node 25 : 19.000000
Node 26 : INF
Node 27 : INF
Node 28 : INF
Node 6 : 19.000000
```

[4]

```
hari@Haricharan:~/Game_AI/hw3$ ./dijkstra_avg
Enter the source node: 1
Enter the goal node: 17
Shortest distances from node 1:
Shortest distance from node 1 to node 17 is 26

Average shortest path distance from node 1 to all reachable nodes: 16.3077

Algorithm Metrics:
Maximum Fringe Size: 6
Total Fill (distance updates): 28
Total Execution Time: 0 ms
Nodes Processed: 29
```

[5]

```
hari@Haricharan:~/Game_AI/hw3$ ./dijkstra_avg
Enter the source node: 0
Enter the goal node: 13626
Shortest distances from node 0:
Shortest distance from node 0 to node 13626 is 292

Average shortest path distance from node 0 to all reachable nodes: 147.491

Algorithm Metrics:
Maximum Fringe Size: 218
Total Fill (distance updates): 1215
Total Execution Time: 7 ms
Nodes Processed: 1216
hari@Haricharan:~/Game_AI/hw3$
```

[6]

```
Using Outgoing Edge Minimum Heuristic:

Algorithm runtime: 1 ms
Nodes processed: 24
Maximum fringe size: 6
Fill count (unique nodes visited): 22
Shortest path: 1 -> 2 -> 22 -> 21 -> 20 -> 19 -> 18 -> 17 -> GOAL
hari@Haricharan:~/Game_AI/hw3$
```

[7]

```
hari@Haricharan:~/Game_AI/hw3$ ./a_star
Enter start node: 1
Enter goal node: 17

Using Landmark-Based Heuristic:
Heuristic overestimates actual cost! Not admissible for node 1 with h = inf and actual = 26
Heuristic overestimates actual cost! Not admissible for node 7 with h = inf and actual = 29
Heuristic overestimates actual cost! Not admissible for node 21 with h = 20 and actual = 12
Heuristic overestimates actual cost! Not admissible for node 20 with h = 23 and actual = 9
Heuristic overestimates actual cost! Not admissible for node 19 with h = 25 and actual = 7

Algorithm runtime: 1 ms
Nodes processed: 21
Maximum fringe size: 7
Fill count (unique nodes visited): 18
Focus ratio (path nodes / expanded nodes): 0.380952
Overestimation count: 5
Average overestimation: inf
Shortest path: 1 -> 2 -> 22 -> 21 -> 20 -> 19 -> 18 -> 17 -> GOAL

Using Outgoing Edge Minimum Heuristic:

Algorithm runtime: 1 ms
Nodes processed: 24
Maximum fringe size: 6
Fill count (unique nodes visited): 22
Focus ratio (path nodes / expanded nodes): 0.333333
Overestimation count: 0
Shortest path: 1 -> 2 -> 22 -> 21 -> 20 -> 19 -> 18 -> 17 -> GOAL
hari@Haricharan:~/Game_AI/hw3$
```

[8]

```
hari@Haricharan:~/Game_AI/hw3$ ./a_star
Enter start node: 0
Enter goal node: 13626

Using Landmark-Based Heuristic:

Algorithm runtime: 2283 ms
Nodes processed: 718
Maximum fringe size: 239
Fill count (unique nodes visited): 713
Focus ratio (path nodes / expanded nodes): 0.0111421
Overestimation count: 0
Shortest path: 0 -> 5659 -> 7050 -> 9304 -> 9997 -> 10121 -> 12600 -> 13626 -> GOAL

Using Outgoing Edge Minimum Heuristic:

Algorithm runtime: 2148 ms
Nodes processed: 608
Maximum fringe size: 269
Fill count (unique nodes visited): 607
Focus ratio (path nodes / expanded nodes): 0.0131579
Overestimation count: 0
Shortest path: 0 -> 5659 -> 7050 -> 9304 -> 9997 -> 10121 -> 12600 -> 13626 -> GOAL
hari@Haricharan:~/Game_AI/hw3$
```

[9]

```
hari@Haricharan:~/Game_AI/hw3$ ./a_star
Enter start node: 0
Enter goal node: 16460

Using Landmark-Based Heuristic:

Algorithm runtime: 2712 ms
Nodes processed: 925
Maximum fringe size: 248
Fill count (unique nodes visited): 914
Focus ratio (path nodes / expanded nodes): 0.00972973
Overestimation count: 0
Shortest path: 0 -> 5659 -> 7050 -> 9304 -> 9997 -> 10121 -> 12600 -> 13626 -> 16460 -> GOAL

Using Outgoing Edge Minimum Heuristic:

Algorithm runtime: 2630 ms
Nodes processed: 955
Maximum fringe size: 269
Fill count (unique nodes visited): 948
Focus ratio (path nodes / expanded nodes): 0.00942408
Overestimation count: 0
Shortest path: 0 -> 5659 -> 7050 -> 9304 -> 9997 -> 10121 -> 12600 -> 13626 -> 16460 -> GOAL
hari@Haricharan:~/Game_AI/hw3$
```

Parameters table 1:

Parameter	Values
Max Velocity	40
Max Acceleration	100
Vel/ Acceleration	0.4
Time to target Velocity	0.3
Radius of Satisfaction	8.0
Radius of Deceleration	30.0

[10]