CS 351 - Project 2

Petros Blankenstein

Test cases:

Portland - Salem
Greedy Best First Search:
Portland -> Salem
Distance: 47.0
Time Taken: 0.00016045570373535156
Edges Evaluated: 4
Vertices Explored: 2

Dijkstra's Algorithm:
Portland -> Salem
Distance: 47.0
Time Taken: 8.916854858398438e-05
Edges Evaluated: 4
Vertices Explored: 2

A*:
Portland -> Salem
Distance: 47.0
Time Taken: 8.96453857421875e-05
Edges Evaluated: 4
Vertices Explored: 2

Portland - Eugene
Greedy Best First Search:
Portland -> Salem -> Eugene
Distance: 111.0
Time Taken: 0.00020885467529296875
Edges Evaluated: 7
Vertices Explored: 3

Dijkstra's Algorithm:
Portland -> Salem -> Eugene
Distance: 111.0
Time Taken: 0.0001266002655029297

Edges Evaluated: 17
Vertices Explored: 7

A*:
Portland -> Salem -> Eugene
Distance: 111.0
Time Taken: 0.0001125335693359375
Edges Evaluated: 7
Vertices Explored: 3

Portland - Ashland
Greedy Best First Search:
Portland -> Newport -> Florence -> Coos_Bay -> Roseburg -> Medford -> Ashland
Distance: 412.0
Time Taken: 0.00027108192443847656
Edges Evaluated: 18
Vertices Explored: 7

Dijkstra's Algorithm:
Portland -> Salem -> Eugene -> Roseburg -> Medford -> Ashland
Distance: 283.0
Time Taken: 0.0001616477966308594
Edges Evaluated: 51
Vertices Explored: 20

A*:
Portland -> Salem -> Eugene -> Roseburg -> Medford -> Ashland
Distance: 283.0
Time Taken: 0.00020694732666015625
Edges Evaluated: 23
Vertices Explored: 8

Portland - Burns
Greedy Best First Search:
Portland -> Hood_River -> The_Dalles -> Madras -> Redmond -> Bend -> Burns
Distance: 351.0
Time Taken: 0.00015735626220703125
Edges Evaluated: 19
Vertices Explored: 9

Dijkstra's Algorithm:
Portland -> Hood_River -> The_Dalles -> Madras -> Redmond -> Bend -> Burns
Distance: 351.0
Time Taken: 0.00029087066650390625
Edges Evaluated: 53
Vertices Explored: 22

A*:
Portland -> Hood_River -> The_Dalles -> Madras -> Redmond -> Bend -> Burns
Distance: 351.0
Time Taken: 0.00014901161193847656
Edges Evaluated: 52
Vertices Explored: 21

Portland - Medford
Greedy Best First Search:
Portland -> Hood_River -> The_Dalles -> Madras -> Redmond -> Bend -> Burns
Distance: 351.0
Time Taken: 0.00023674964904785156
Edges Evaluated: 16
Vertices Explored: 7

Dijkstra's Algorithm:
Portland -> Hood_River -> The_Dalles -> Madras -> Redmond -> Bend -> Burns
Distance: 351.0
Time Taken: 0.00014090538024902344
Edges Evaluated: 53
Vertices Explored: 22

A*:
Portland -> Hood_River -> The_Dalles -> Madras -> Redmond -> Bend -> Burns
Distance: 351.0
Time Taken: 0.00015950202941894531
Edges Evaluated: 27
Vertices Explored: 10

Dijkstra and A* consistently found the optimal path while GBFS had a less efficient path. Every time it had to choose between Roseburg and Crater Lake from Eugene, it chose Crater Lake, when Roseburg is the better option, giving it an unnecessary 28 mile detour. Though GBFS consistently explored/evaluated the least edges and vertices compared to Dijkstra and A*, except when going from Portland - Salem where they all evaluated 4 edges and explored 2 vertices (Portland and Salem).

GBFS is consistently the best option algorithm (in terms of exploring the least amount of vertices/edges) except when it has to choose between Roseburg and Crater Lake, or when going south from Portland (like to Ashland for example), it goes to Newport from Portland. Between A* and Dijkstra, A* explores less than Dijkstra but their time to run is similar. I would probably choose A* in a real world scenario because it always found the best solution and performed on par/slightly better than Dijkstra. I would use GBFS if I didn't need to travel very far (it performs perfectly in shorter cases). I would use Dijkstra if I needed to run it multiple times because it's more efficient when you run it again.

GBFS - time O(n log n), space O(n)

Dijkstra - time O(n log n), space O(n)

A* - time O(n log n), space O(n)

They are all the same in terms of time/space complexity. The map affects performance loglinearly. The more vertices and edges, the longer it will take to search through (especially for further vertices from the start). It will also use more space to store that data. The only reason to use GBFS is it searches through significantly less vertices and edges. I would like to see how the data scales on a really large graph.

Haversine is admissible for A* because it works. It calculates the distances based on the curvature of the Earth, and it gives the distances from the current vertex to the next one which is what A* needs. When the heuristic underperforms in GBFS it can pick a suboptimal route, such as going from Eugene to Crater Lake instead of Roseburg or going to Newport from Portland instead of Salem when trying to reach Astoria. It does this because Crater Lake is closer to the goal than Roseburg and it's only caring about getting closer to the goal, not how far it has to travel (same reason for Newport). I don't think any heuristic could help GBFS (I could be wrong). It fundamentally cannot know how far it is travelling/has to travel which means it will always make the mistake of going to Crater Lake or Newport instead of Roseburg or Salem because Crater Lake and Newport are closer to the goal. Maybe if it was calculating road distance to the goal instead of distance as the crow flies?