Omar Moustafa

900222400

February 21, 2023

CSCE 3601

Assignment 1

Introduction

Several search algorithms were explored to find the shortest path between a list of European cities utilizing a predefined road network. The objective was to implement and compare six different search algorithms: Depth First Search (DFS), Breadth First Search (BFS), Iterative Deepening Search (IDS), Uniform Cost Search (UCS), Greedy Best-First Search, as well as A Search*. After employing each algorithm, its output will be reviewed and evaluated based on the "Total Cost of the Path," which is the sum of the road distance, and the total "Number of Nodes Expanded," which measures the overall efficiency of the algorithm.

Test Case 1: DFS

When running Depth First Search (DFS) from Paris to Vienna, the algorithm expanded 10 nodes and found a path with a total cost of 3347 km. The path taken was the following:
Paris -> Madrid -> Genoa -> Rome -> Trieste -> Belgrade -> Budapest -> Vienna

This first result following the implementation of the DFS algorithm highlights one of its major weaknesses when it comes to pathfinding problems that involve weighted graphs. Since the DFS algorithm does not take path cost into consideration, it goes ahead and explores paths that are based on depth instead of efficiency. This is why it ended up finding a highly inefficient route which included traveling far south to cities, namely Madrid and Rome, before heading in the correct northeast direction to arrive in Vienna. This specific detour played a significant role in increasing the total travel cost as opposed to simply taking a much more efficient route that would have included a more direct route from Paris to Vienna.

With that said, the total number of nodes expanded was only 10 which reveals that the DFS algorithm found its path and solution quickly. However, the other side to that is that the total cost of the path strongly indicates that the solution was not optimal in the slightest. The DFS algorithm usually fails to find the shortest path because it tends to follow the first complete path it encounters, even if there are much better paths that exist and can be taken.

Test Case 2: BFS

When running Breadth First Search (BFS) from Paris to Vienna, the algorithm expanded 20 nodes and found a path with a total cost of 1307 km. The path taken was the following:

Paris -> Genoa -> Trieste -> Vienna

Unlike DFS, BFS works to explore all of the nodes at the current depth before moving any deeper. Therefore, it assures finding the shortest possible path when it comes to the number of edges, but it does not always or necessarily minimize the total distance in a weighted graph. In this particular case, BFS performed better than DFS, as it steered away from the unnecessary detour through far away cities like Madrid and Rome, which led to a much lower total path cost. But with a more direct path or route comes the trade-off of expanding more nodes, which it did as it expanded double the number of nodes that DFS did.

BFS is generally more relied on than DFS when it comes to finding solutions with a lesser number of steps, but it has to be said that it can be quite inefficient in graphs where edge weights vary a lot. Due to its lack of accounting for distances, it may exhibit paths with many short edges instead of choosing the most direct low-cost path out there.

Test Case 3: IDS

When running Iterative Deepening Search (IDS) from Paris to Vienna, the algorithm expanded 31 nodes and found a path with a total cost of 1195 km. The path taken was the following: Paris -> Brussel -> Amsterdam -> Munich -> Vienna

IDS is a combination of the two previously mentioned algorithms, DFS and BFS, and works to perform repeated depth-limited DFS runs with increasing depth limits. Similar to BFS, it ensures that it ends up finding the shortest path when it comes to the total number of edges while maintaining a lower memory footprint. With that said, due to it repeatedly exploring nodes at shallower depths before extending the search, it tends to end up with a greater number of nodes expanded in the BFS algorithm, which was revealed above as the IDS expanded a total of 31 nodes compared to DFS which only expanded 10 and BFS which only expanded 20.

Despite the increased number of nodes, the IDS algorithm implemented a more efficient path than BFS in terms of path cost. The total cost of 1195 is lower than the 1307 of BFS, hinting that the depth-first component of IDS helped steer away from the slightly more expensive paths and routes.

Test Case 4: UCS

When running Uniform Cost Search (UCS) from Paris to Vienna, the algorithm expanded 21 nodes and found a path with a total cost of 1195 km. The path taken was the following: Paris -> Brussel -> Amsterdam -> Munich -> Vienna

The UCS algorithm is designed in a way that leads it to always expand the least-cost node first which assures that it finds the optimal path or solution in a weighted graph. This leads it to be much more reliable than the three previous algorithms when it comes to looking for the optimal solution or shortest possible path. In this particular case, the output path of the UCS algorithm was the same as that of the IDS algorithm but it was able to utilize the path while only expanding 21 nodes versus the 31 of IDS. This, therefore, represents the major efficiency in avoiding unnecessary node expansions while at the same time also guaranteeing it provides the least-cost path.

Contrary to the BFS and IDS algorithms, which both work to expand nodes based on depth and not cost, the UCS algorithm works to evaluate paths based on their actual travel distance which prevents any detours or higher-cost routes. This allows for it to consistently succeed at finding the best possible path without any reliance on heuristics. With that being said, depending on the graph structure, the UCS algorithm could still expand a large number of nodes which would make it less efficient and more computationally expensive when dealing with larger networks and instances.

Test Case 5: Greedy Search

When running the Greedy Search from Paris to Vienna, the algorithm expanded 4 nodes and found a path with a total cost of 1307 km. The path taken was the following:
Paris -> Genoa -> Trieste -> Vienna

The Greedy Search algorithm chooses the next city to expand solely based on the heuristic function, which in this particular case is the Euclidean distance to the goal. Contrary to the UCS algorithm, the Greedy Search does not factor in the path cast that has been accumulated up until now. This leads to Greedy Search expanding a very small number of nodes, as stated above it only expanded a total of 4 nodes, making it clearly the most efficient algorithm when it comes to the number of nodes that were expanded. However, this efficiency comes at a cost or tradeoff, which is that the Greedy Search does not guarantee the shortest possible path.

The total path cost was 1307 which is higher than that of the UCS, which was only 1195, meaning that the route taken by the Greedy Search algorithm was not the most optimal one out there. Its priority was set on the cities that had appeared geographically closer to Vienna but it did not factor in the actual road distances from one city to another. Sometimes, this method can lead to longer and more expensive routes or paths as opposed to algorithms that work to take into consideration both the cost as well as heuristic information and not just one or the other.

Test Case 6: A* Search

When running the A* Search from Paris to Vienna, the algorithm expanded 9 nodes and found a path with a total cost of 1195 km. The path taken was the following: Paris -> Brussel -> Amsterdam -> Munich -> Vienna

The A* Search algorithm combines the strengths of the UCS and Greedy Search algorithms and it does so by factoring in both the actual path cost as well as the heuristic estimate to the goal. This specific balance allows for the A* algorithm to efficiently find the shortest possible route while also expanding a much lesser number of nodes than the UCS algorithm. Building on this, A* found the same optimal path as UCS but expanded only 9 nodes, compared to UCS's 21 nodes, making it the most efficient algorithm tested.

Since the A* Search algorithm sets its priorities on the sum of cost so far and estimated cost to goal, it avoids the risks of straight heuristic-based algorithms like the Greedy Search, which can lead to decent but not optimal paths or routes. The fact that A* produces the lowest-cost path while expanding fewer nodes than UCS strongly represents its effectiveness in efficiently solving such shortest-path problems.

Node Expansion: BFS vs DFS

- ***Case 1: BFS Expands Fewer Nodes than DFS***

  When trying to find a city pair for which breadth-first search expands fewer nodes than depth-first, it was found that from Amsterdam to Berlin, DFS expanded a total of 15 nodes and produced a path costing 4955, while BFS expanded only 10 nodes and also found a much cheaper path at only 806.

  DFS Path: Amsterdam -> Brussel -> Paris -> Madrid -> Genoa -> Rome -> Trieste -> Belgrade -> Budapest -> Vienna -> Munich -> Bern -> Berlin

  This shows that DFS followed a long and inefficient path which is due to how it worked to explore deeply into the search tree before backtracking to find a better one.

  BFS Path: Amsterdam -> Munich -> Vienna

  On the other hand, BFS explored cities level by level and very quickly found a significantly shorter route.

  This, therefore, establishes that the BFS algorithm is generally better than the DFS algorithm when the objective is to find the shortest and most efficient path in an unweighted graph. DFS blindly follows deep paths which tends to lead to expensive and highly inefficient detours whereas BFS systematically explores all of the possible paths at each depth level which assures that it ends up finding the shortest path in terms of the number of edges and hence, expanding a fewer number of nodes.

- ***Case 2: DFS Expands Fewer Nodes than BFS***

  When trying to find a city pair for which breadth-first search expands fewer nodes than depth-first, it was found that from Budapest to Belgrade, DFS only expanded 2 nodes whereas BFS expanded 5 nodes even though both algorithms found the same shortest path at 263 km.

  DFS Path: Budapest -> Belgrade

  BFS Path: Budapest -> Belgrade

  Analyzing the number of nodes expanded along with their respective paths, it can be interpreted that the DFS algorithm found the goal state almost instantly as the correct path was explored from early on. Whereas the BFS algorithm needed to expand multiple nodes before arriving at the goal state, as it systematically and automatically works to explore all of the possible paths and next moves at the same depth.

  Therefore, this demonstrates that when the goal state happens to be along the first deep path that is being explored, the DFS algorithm shines and outperforms the BFS algorithm by providing a more efficient and cheaper path. With that said, this particular efficiency is heavily reliant on the order in which DFS explores paths, making it not ideal and even unreliable when it comes to general shortest-path problems.

Impact of Heuristic Scaling on Greedy Search

To analyze and understand the impact of heuristic scaling, three different scaling factors (0.5, 1.0, and 1.5) were experimented with under the Greedy Search and A* Search algorithms.

When running Greedy Search also from Paris to Vienna with three different heuristic scaling factors, the algorithm **Expanded 4 Nodes** and **found the same path** with a **Total Cost of 1307** for all factors. The path was: Paris -> Genoa -> Trieste -> Vienna

When running A* Search also from Paris to Vienna with different heuristic scaling factors, all factors **gave the same path but different costs, and a different number of nodes expanded** as listed below. The path was: Paris -> Brussel -> Amsterdam -> Munich -> Vienna

- **Factor = 0.5:** ***Nodes Expanded:*** *17,* ***Total Cost:*** *1195*
- **Factor = 1:** ***Nodes Expanded:*** *9,* ***Total Cost:*** *1195*
- **Factor = 1.5:** ***Nodes Expanded:*** *5,* ***Total Cost:*** *1195*

This demonstrates that, unlike the A* Search algorithm, the Greedy Search algorithm is largely unaffected by heuristic scaling unless multiple paths have almost identical heuristic values. The path chosen was already heavily influenced by the heuristic, so increasing or decreasing its weight did not change the order in which cities were explored. Also unlike A*, Greedy Search solely depends on the heuristic to the extent that changing or adjusting the heuristic multiplier will not always lead to different results or outputs. If more complex maps are being dealt with, then perhaps the scaling and changing heuristic values would lead to different paths, results, and outputs. But in this particular case, the same output was obtained three different times with three different heuristic multipliers.

Conclusion

In conclusion, the outputs and results highlight the trade-offs between the six different search algorithms. With an excessively lengthy and expensive path, DFS delivered the worst result despite expanding the fewest nodes. Although IDS and BFS had discovered better routes, they continued to overlook road distances, producing less-than-ideal outcomes. UCS discovered the shortest path, although it added more nodes than required. The most computationally efficient method, Greedy Search, lost path quality because it didn't account for travel expenses.

After all of that, A* Search was the one that had emerged as the best-performing algorithm of the six as it perfectly balanced both efficiency and optimality. It expanded a lesser number of nodes than the UCS algorithm did while also maintaining the lowest cost route. This thoroughly demonstrates that integrating a well-designed heuristic along with path cost consideration leads to superior performance when it comes to solving shortest-path problems. Overall, the A* Search algorithm is the most efficient and suitable algorithm for this particular problem as it successfully achieved the best balance between the quality of the solution and the computational effort that was exerted.