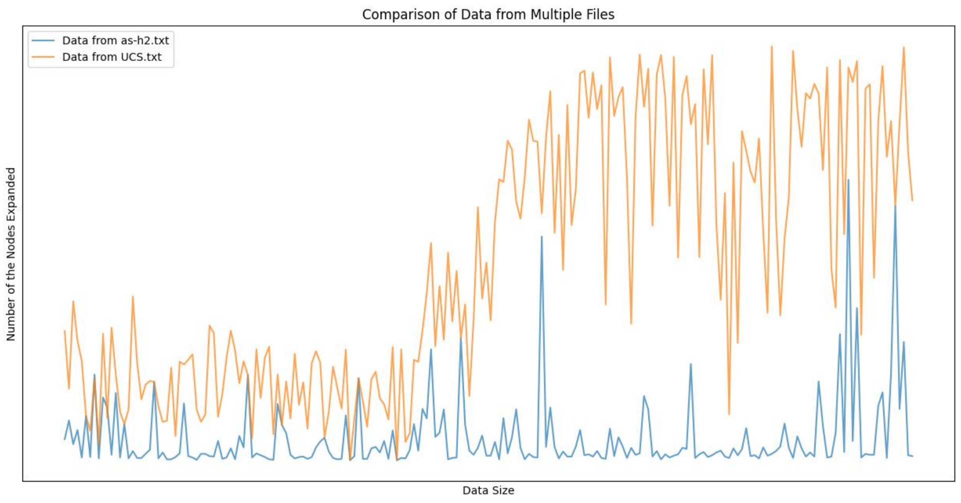
It was very hard to come up with an admissible solution. I come up with an almost admissible version but not able to come up with the proof of it being admissible.

I used a minimal spanning tree to form a graph and find the minimal cost of connecting all points, which reduces to reaching all pawns. However, there are three different graphs involved, making it challenging to implement the most optimal solution. Firstly, I checked whether there is a pawn reachable by the knight, as the knight has the lowest cost of action among all pieces. After eliminating one or two pawns, depending on the number of pawns adjacent to the knight, I applied the minimal spanning tree method using the bishop and the rook. Since there are two different graphs involved, with possibly intersecting points and different edges between the same two nodes, I first calculated the smallest cost of connecting all points using the rook, which has a lower cost of action than the bishop. The program then does the same for the bishop.

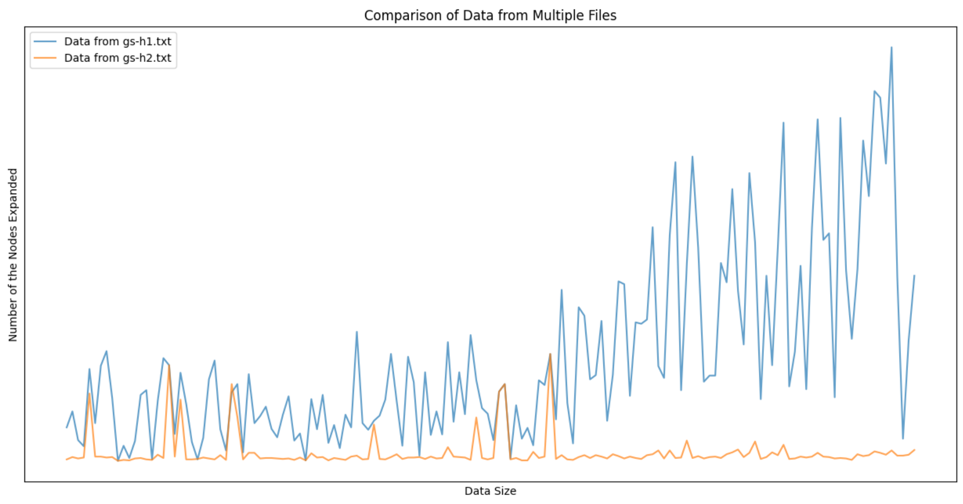
After obtaining all costs related to the different graphs, I combined them by using their mean. This approach proved to be very efficient, as it significantly outperformed heuristic1 and UCS, deviating from the optimal solution by only 0.4%. I ran the algorithm on different input sets of 300 tables, and about 40% of the time, my algorithm returned the optimal solution. In the other cases, I detected an average deviation of 0.4% from the optimal solution, which is acceptable given the dramatic speedup in the process.

Since I am unable to prove that my algorithm never overestimates the true cost, I cannot prove that it is admissible. However, because it uses hashing to avoid repeating states (loopy paths), it visits all possible nodes and finds the solution at any point if it exists

These are the plots related to performance comparison of my algorithm when used with a sample of size 200:

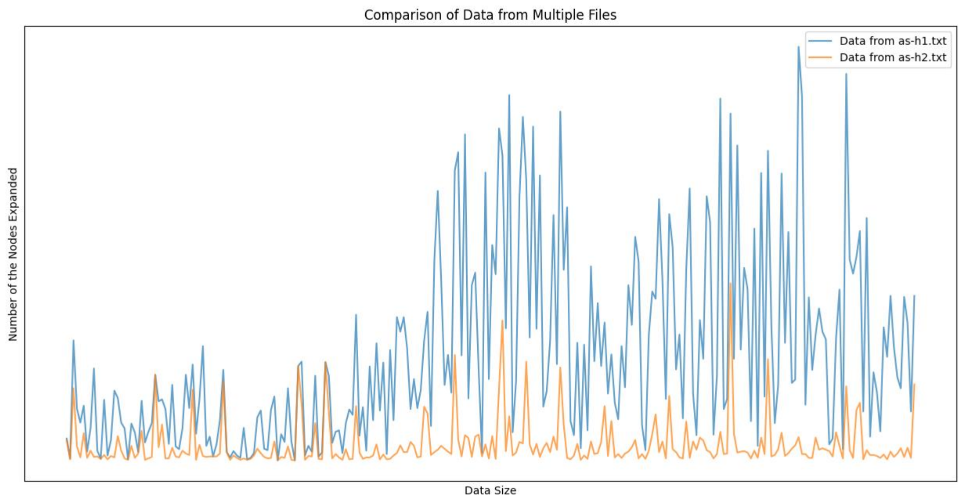


It completely outperforms UCS as the data size grows.

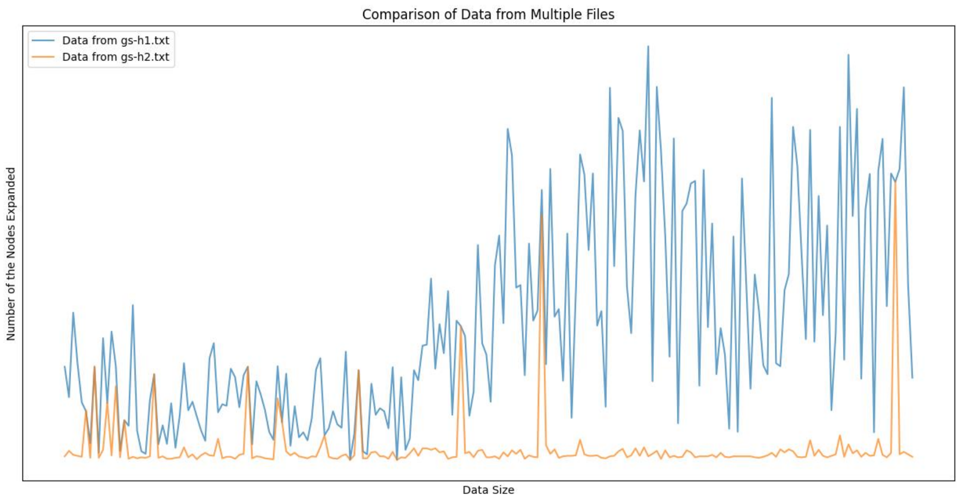


.

My heuristic algorithm completely outperforms heuristic1 when used with greedy search.



Here is a comparison of my algorithm used with A* search. It completely outperforms h1 and has a very slow growing rate as the input size grows larger.



Here is another comparison when used with a sample of size 250.My greedy search algorithm completely outperforms heuristic1.

There are some spikes in the graph, and these are the graphs that are not solvable because of the configuration of the obstacles. Then the algorithms visit all nodes and terminate.