CIS579 Artificial Intelligence

Four Knights Problem

Project Report by Kyrylo Krysko

CONTENTS

1 INTRODUCTION

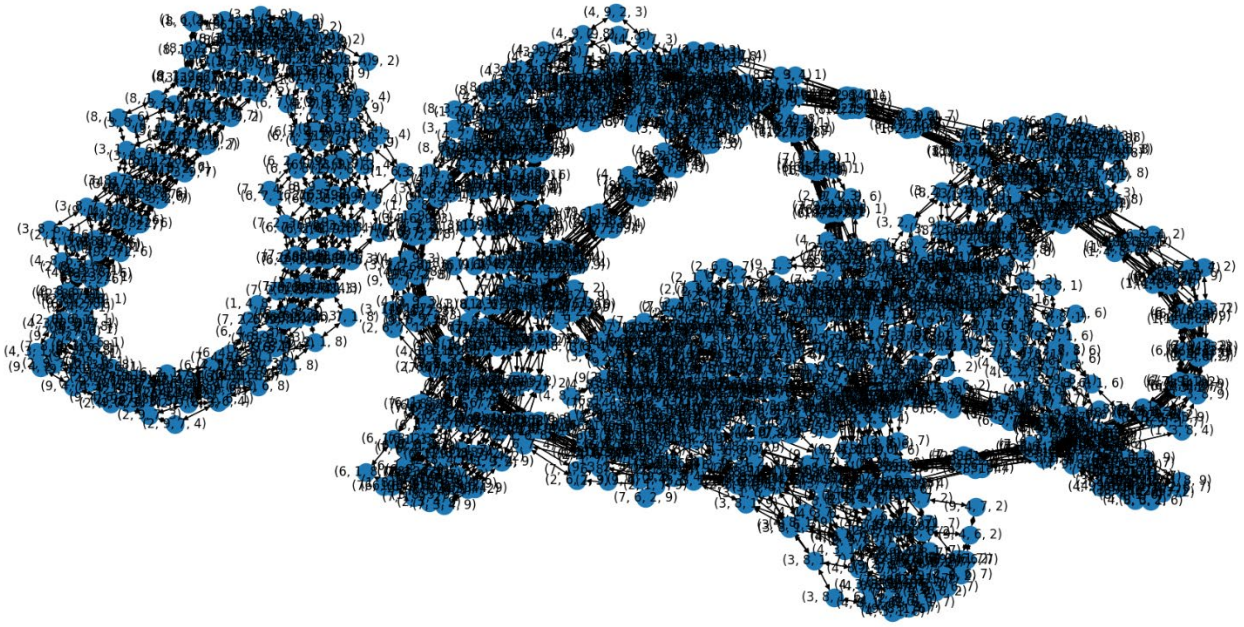
This is the Python program that solves the Four Knight problem using graph data structure and different graph search algorithms. The goal is to compare the effectiveness of A* and Branch and Bound search algorithms.

2 IMPLEMENTATION

We can build a model of the Four Knight problem as a graph where each node is a tuple of 4 values, which are the current position of each knights on the board (assuming we numbered all board cells from 1 to 9). E.g. the initial position of the problem described as (1, 3, 7, 9) and the solution is (9, 7, 3, 1).

Next we need to build a graph of all possible knight moves where staring node will be (1, 3, 7, 9). This task was done automatically, first (build_all_states function) we created a list of tuples, all possible positions of knights on the board (keeping in mind that two knights can't be in the same cell and they can't hit the center cell number 5). As a result we get 1680 nodes that were used to create graph with edges. Then we add resulting tuples to a graph and add edges (build_solution_graph function).

Here is a graph visualization to understand the scale of the problem:

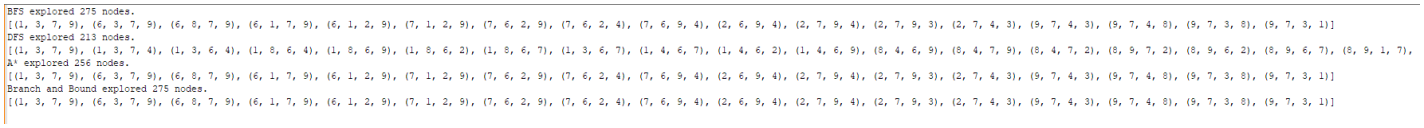


Having graph in place we can find a solution by finding the path from (1, 3, 7, 9) node to (9, 7, 3, 1) node. There could be multiple possible valid solutions but we are interested in the most optimal, which means we need to find the SHORTEST path, which is well known computer science problem. The evolution of finding the shortest path started from basic Breadth First Search algorithm making it smarter. The latest best algorithm is an A* search.

To determine the algorithm efficiency I decided to calculate the number of visited nodes to find a solution. In addition I tested Depth First Search and Breadth First Search algorithms to get better understanding of results due to a straight forward nature of both of them.

3 RESULTS

The result of a program execution:



Search Algorithm	Visited Nodes	Number of Moves
BFS	275	17
DFS	213	49
B&B	275	17
A*	256	17

The DFS found result with less number of visited nodes but the solution is far away from optimal, 49 steps instead of 17 (best solution). BFS founds the optimal solution with the same efficiency as B&B. I think this behavior of BFS and DFS can be dramatically different if we use more complex problems, looks like here we might hit the best case scenario due to the nature of programmatically generated data. But even in this case this is a good illustration on how those two algorithm working, looking layer after layer vs going deep on each branch.

As for the original assignment task, we can see that based on this particular case A* is 7% more efficient than B&B, which does not seems that impressive, but again, 17 hundreds nodes is a very simple case comparably to real world scenarios of finding the shortest route on the map.

I was trying to change A* heuristic algorithm (implementing manhattan_distance, euclidean_distance, chebyshev_distance), the results was completely the same which is not a surprise for such relatively simple case.

Then I try to provide different goal values:

Search Algorithm	Goal	Visited Nodes	Number of Moves
BFS	(4, 9, 6, 2)	127	8
DFS	(4, 9, 6, 2)	43	38
B&B	(4, 9, 6, 2)	127	8
A*	(4, 9, 6, 2)	72	8
BFS	(1, 8, 6, 9)	24	3
DFS	(1, 8, 6, 9)	5	4
B&B	(1, 8, 6, 9)	24	3
A*	(1, 8, 6, 9)	4	3
BFS	(8, 9, 1, 6)	149	9
DFS	(8, 9, 1, 6)	19	19
B&B	(8, 9, 1, 6)	149	9
A*	(8, 9, 1, 6)	93	9

The A* was 44%, 84%, 38% more efficient than B&B in those cases which looks more promising.

4 CONCLUSION

This was a very interesting project that helped to get experience in creating a mathematical models of common problems, feel the difference between core search algorithms, understand how small logical improvement can results in dramatic boost of efficiency.

As a possible improvements I would suggest to increase the data complexity by exploring bigger boards (4x4, 5x5 etc) as well as building graph for another similar problems to potentially examine heuristics differences.