Artificial Intelligence Laboratory 2: A* Search Algorithm DT8042 HT22, Halmstad University

Santiago Plazas, sanpla22@student.hh.se, M.Sc. Embedded and Intelligent Systems
William Wahlberg, wahlbergwille@gmail.com, M.Sc. Computer Science

November 2022

Introduction

The objective of this lab is to start tinkering with search algorithms to determine the fitness of each algorithm in solving different tasks. For this laboratory we had two problem domains, Path planning: find the shortest path from the agent's current position to the goal. And, Poker game: find a sequence of bids for your agent to win more than 100 coins from the opponent within 4 hands.

Q1: What types of search algorithms you have learned from the lecture? Please briefly introduce them here.

Random

Random search is a strategy which expands nodes on the fringe randomly. All nodes on the fringe have an equal chance of being expanded.

DFS

Depth-first search is a strategy for exploration that always selects the deepest node on the fringe from the start node for expansion. By removing the deepest node and adding its children to the fringe, we are making the children deepest than any previous node. This follows a last-in, first-out (LIFO) stack strategy.

BFS

Breadth-first Search is a strategy which always prioritizes the nearby nodes first. This means the searched area will expand equally in all directions unless it encounters any obstacles. This follows a first-in, first-out stack strategy.

UCS

Uniform cost search is a strategy for exploration that always selects the lowest cost node on the fringe from the start node for expansion.

Greedy

Greedy search always expands the nodes on the fringe with the lowest heuristic. For example, in the case of searching a map, the heuristic is the distance to the end node. That means the algorithm would always expand the nodes nearest the end node in an attempt to head directly to it.

A*

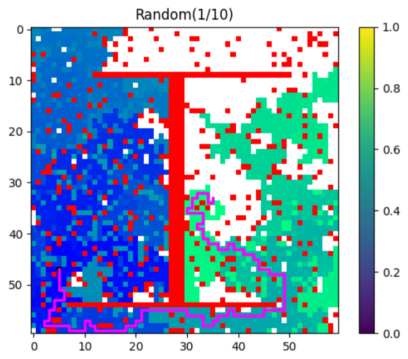
A* search is a strategy for exploration that always selects the node on the fringe with the lowest estimated total cost for expansion, where the total cost is the entire cost from the start node to the goal node. A* combines the total backward cost (real cost) used by UCS (Uniform cost search) with the estimated forward cost (heuristic value) used by greedy search by adding these two values, yielding an estimated total cost from start to goal.

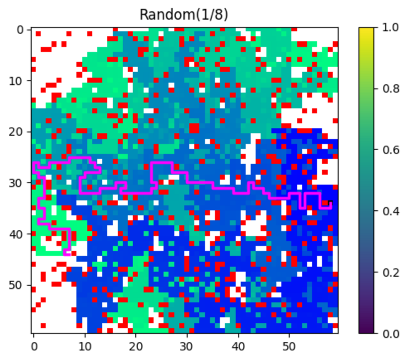
Task 1: Path Planning

Q2: Implement and apply uninformed search algorithms, e.g. random search, BFS, and DFS. Do they find the optimal path on both maps? How many nodes were expanded? Please include a few example plots of the grid maps with the evaluation values of each cell and the path found.

Random

Random search expands more nodes the further the optimal path is. Even though it expands randomly, it expands on average the same amount of nodes as the BSF. Random search does not find the optimal from the starting node to the end node. The found path is also random since it contains the nodes that were randomly expanded until it found the end node.



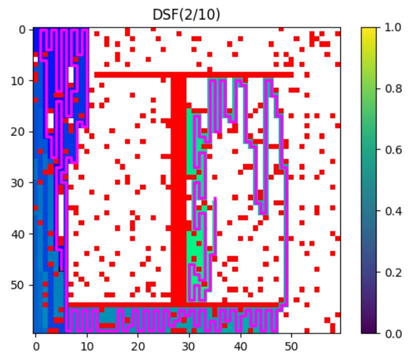


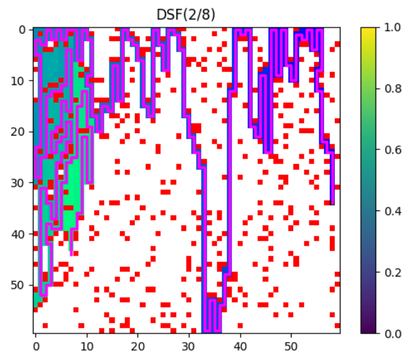
DFS

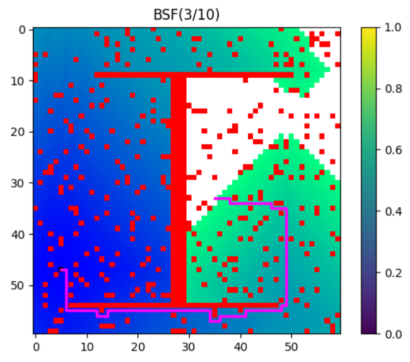
Depth-first search simply finds the ”rightmost” solution (in our implementation) in the search tree without caring for path costs, and so it is not optimal. This algorithm may end up exploring the entire search tree for any fitting solution. DFS adds b nodes (b is the branching factor) at each of m depth levels on the frontier, and it allows only one of the sub-trees of any of these children to be explored at any given point. So in the worst case, the frontier will contain b * m nodes.

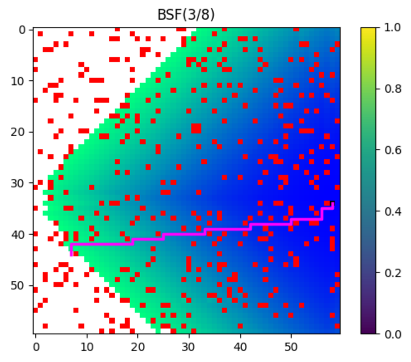
BFS

BFS is considered not to be optimal because it does not take into consideration any path costs when deciding how to expand the nodes. However, for this experiment, all the costs were equivalent which is a special case where BFS becomes UCS, which is guaranteed to be optimal. We used BFS to compare the optimality of all other search algorithms in the laboratory. BFS will try to add all the shallowest nodes to the fringe and in the worst case, will contain all the nodes in the level corresponding to the shallowest solution. So for the worst case, the fringe will contain b^s nodes if s is the level where you can find the shallowest solution.





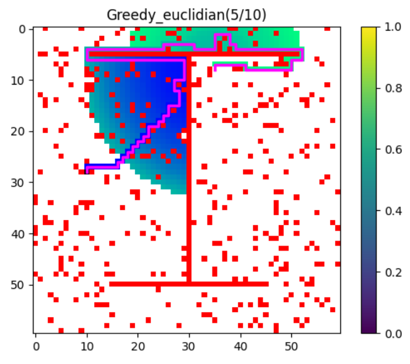


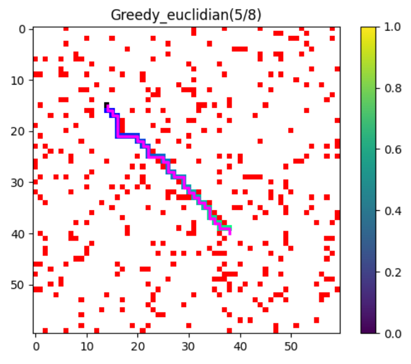


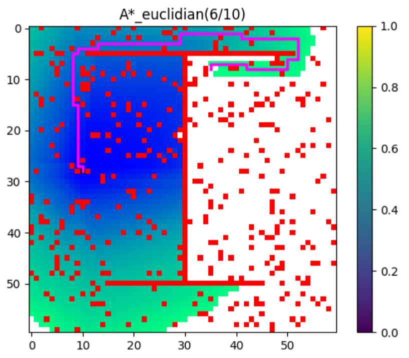
Q3: Implement and apply greedy and A* search algorithm. What heuristic function have you implemented? Do both algorithm found the optimal path on both maps? Please provide examples.

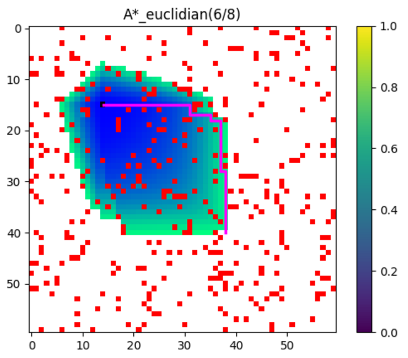
Greedy

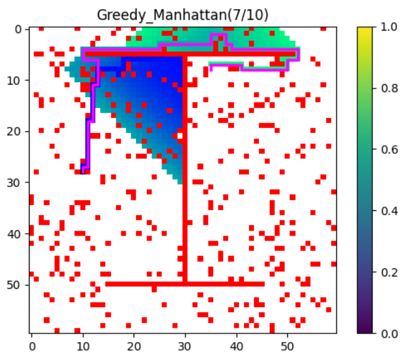
Greedy Search was implemented using both the euclidian heuristic and the manhattan heuristic. In this example, it found the optimal path on the map without the big obstacle using the euclidian heuristic. It did not find the optimal path on any other map using any of the heuristics. Greedy search tends to trap itself as it did on the large obstacle in the middle. Although it found the optimal path in one of the examples, finding the optimal path requires some luck regarding the map.

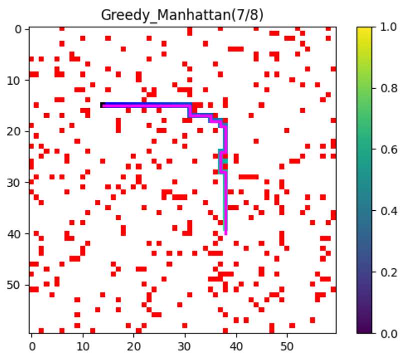








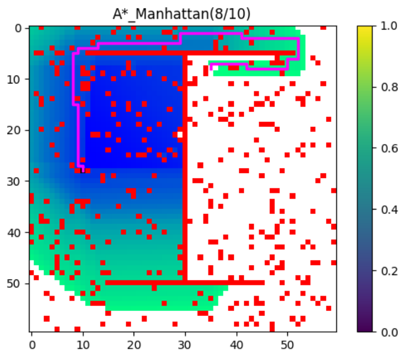


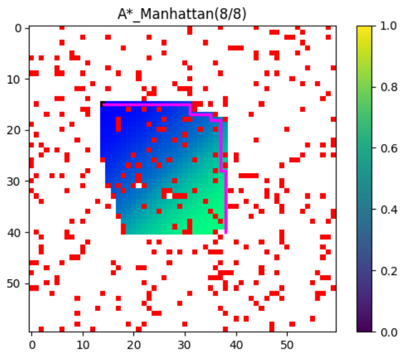


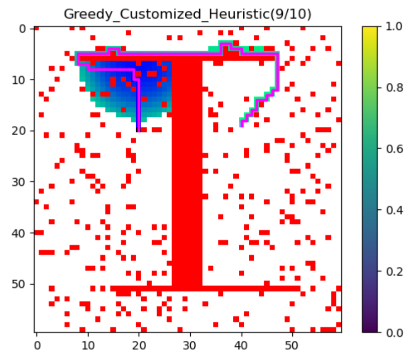
A*

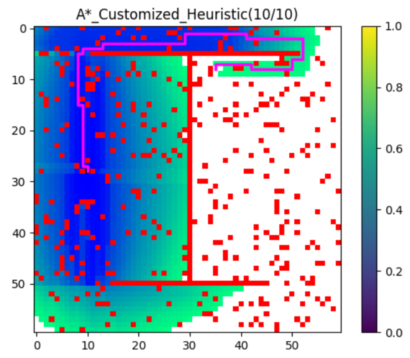
A* was implemented using the euclidian heuristic, manhattan heuristic and a custom heuristic. The shortest path was always found using the euclidian and manhattan heuristic. That is somewhat expected since A* is in some ways BSF but with improved calculation time.

We also implemented A* using a custom heuristic specifically for the map with an obstacle. The map was split into multiple sections with different heuristics. The strategy was to have the algorithm first find an opening in the obstacle and then aim towards the goal. The custom heuristic found a path to the end node but not the shortest path.









Q4: Compare different search algorithms. For each search algorithm, run the experiment multiple times (e.g. 20) and fill in the following table.

We ran the experiment 100 times for each algorithm and got the results shown in Tables 1 and 2. The mean path length we got for all the 100 maps using BFS corresponds to the mean shortest path for all the iterations. Hence, we can compare how close to this number is to the mean path length found by other algorithms, and this will tell us how close other algorithms were to the optimal path. From the table, we found that A* Manhattan and Euclidean found the optimal path all of the time. However, the custom A* did not find the optimal path all the time because the custom heuristic we designed is inadmissible. However, the path found by custom A* is much shorter than DFS or BFS and way more efficient with resources.

Search algorithms	Number of nodes expanded	Time consumed (ms)	Path length
Random	2032.9 ± 1117.84	1006.56 ± 960.08	86.27 ± 41.66
BFS	3033.91 ± 1682.9	972.81 ± 809.52	44.09 ± 20.27
DFS	1660.0 ± 1291.78	234.38 ± 264.44	830.03 ± 570.83
Greedy Euclidian	44.8 ± 20.77	1.56 ± 4.69	44.25 ± 20.25
A* Euclidian	1023.54 ± 1028.98	165.47 ± 273.31	44.09 ± 20.27
Greedy Manhattan	48.06 ± 23.04	1.09 ± 3.99	46.71 ± 21.99
A* Manhattan	692.34 ± 727.91	129.22 ± 235.81	44.09 ± 20.27

Table 1: Performance comparison (Path Planning, Map without rotated ”H”)

Task 2: Poker Bidding

Q5: Implement and apply random, BFS, DFS, and greedy search algorithms. What have you observed? Do all of them found a solution? How many nodes were expanded?

Surprisingly, the random algorithm performed better than any other search algorithm. And, explaining the apparently high success rate of the random agent is difficult. However it makes sense that an irrational player could beat a very rational player. The rational player will react to the actions of

Search algorithms	Number of nodes expanded	Time consumed (ms)	Path length
Random	2434.7 ± 988.88	1571.88 ± 1288.07	142.46 ± 70.56
BFS	3526.56 ± 1408.46	1273.83 ± 790.72	70.94 ± 33.15
DFS	1881.06 ± 1201.02	262.89 ± 258.03	832.26 ± 434.13
Greedy Euclidian	658.92 ± 665.59	74.45 ± 88.86	82.56 ± 45.72
A* Euclidian	1939.5 ± 1295.14	387.11 ± 367.84	70.94 ± 33.15
Greedy Manhattan	657.82 ± 701.61	107.27 ± 135.57	81.19 ± 42.22
A* Manhattan	1530.54 ± 1131.17	402.42 ± 442.3	70.94 ± 33.15
Custom Greedy	453.22 ± 288.66	35.0 ± 30.07	106.72 ± 23.58
Custom A*	2605.22 ± 965.76	567.03 ± 355.68	97.84 ± 18.61

Table 2: Performance comparison (Path Planning, Map with rotated ”H”)

the random agent and could end up making bad decisions. Moreover, we can think of the poker game as a maze that has to be navigated, and a random agent can free itself of a bad series of actions and take a new course.

Getting stuck in a bad series of actions is exactly the problem with the DFS agent, it will explore only one possibility to the deepest node, but if it is unlucky it can get stuck pretty easily exploring dead ends. This will make it hard for the algorithm to find a solution within the 20000 steps restriction.

On the other hand, the BFS agent will explore every possibility in the shallow levels of the game, and with the current 20000 step restriction it won't be able to advance deep enough to make progress and take the opponent's money. This is why DFS and BSF have under 5% success rate.

For the greedy agent, we used the least amount of times both players have bet as a heuristic. However, we saw that this algorithm also got stuck in the lower levels.

Finally, we implemented a custom heuristic as our estimation of distance to the goal state. We tried to guess the number of bids both players had to made so that the agent would win the game.

Q7: Briefly describe the heuristic function implemented. How many nodes were expanded with the proposed heuristic function? Is the solution optimal?

To come up with a custom heuristic, first, we calculated the average number of bids both players made by the time the agent was declared as the winner (we tested 20 cases). Then, the heuristic function would be the absolute value of the difference between the current number of bids by both players and this experimental value. So, the closer we are to the ”magic number”, the heuristic gets closer to zero, and the nodes will expand towards this result. However, getting the right number requires to run successful searches more than 20 times, which was not in the scope of this laboratory since for this environment, UCS was taking too long to compute .

This is not an optimal solution, but it is better at handling resources than the other algorithms we implemented, so if time and resources are a concern, then this algorithm might be good.

Q8: Compare different search algorithms.

We ran the search algorithms 20 times and limited the search to expand a maximum of 20000 nodes. In the case the algorithm expanded more than 20000 nodes we say that It could not find a solution.

Search algorithms	Number of nodes expanded	Number of hands	Number of biddings	success rate
Random	8684 ± 9327	3.42 ± 0.64	22.5 ± 3.28	60%
BFS	> 20002	...	...	0%
DFS	16005 ± 7994	2.0 ± 0.0	28.0 ± 1.0	5%
Greedy	> 20002	...	...	0%
Custom Greedy	13764 ± 8704	3.88 ± 0.33	30.0 ± 5.55	40%

Table 3: Performance comparison (Poker Bidding)

Conclusion

In this laboratory, we implemented uninformed and informed search algorithms. From this implementation, we saw that each search problem had a state space (the set of all states reachable from the initial state by any sequence of actions), a set of possible actions, a state transition function (a function that takes actions and returns a new state), an action cost, a start state, and a goal state. Each of our implementations dealt with these components differently, where the agents interacted with their unique environment. The agent function describes what the agent does in all circumstances and, for both environments, we described the completeness, optimality, frontier representation and, space complexity of every type of search.