Project 1: Shortest Path Problem

Group Members

- Janosch Baltensperger (janosch.baltensperger.7118@student.uu.se)
- Domenic Fürer (domenic-luca.furer.2478@student.uu.se)
- Marion Dübendorfer (marion-francine.dubendorfer.0571@student.uu.se)

Instructions

We recommend to use a Python version > 3.7. There is a dependency of `matplotlib`, which can be installed with `pip install -r requirements.txt`. To run the script, use `python node_search.py` and then follow the required inputs.

Algorithm Main Components

In order to model the nodes, we create a dataclass called `Node`. It has the following fields:

- `x` , `y`: x- and y-coordinate of the node
- `state`: index of the node in the adjacency matrix
- `cost`: the cost to reach the node
- `path`: the path from the starting node to the given node

The algorithm consists of multiple functions:

- A `main()` function serves as the entry point of the program and handles data source parsing and user input handling. It then initialises the A* search algorithm.
- A function called `astar_search_graph()` that takes as input the adjacency matrix, the starting node, the goal node, and the weight parameter. The function then performs the weighted A* graph search. Inside the `astar_search_graph` function, we make use of two helper functions called `expand()` and `select_best_node()`.
- Please note that we have experimented with both graph and tree search. We realised, that with the tree search and a non-existent path to a goal node (as example: start 0 to goal 99), the program keeps running forever. We therefore adapted the A* search algorithm to the graph search implementation that keeps track of `visited` nodes in a reached array. Thereby, the algorithm does not keep returning to visited nodes and loops forever.
- In order to find the next node for evaluating its children, we implement `select_best_node()` that examines all nodes currently in the frontier based on a weighted A* evaluation function. This function is composed of heuristic costs (rectilinear distance to goal node) and the path costs given by the adjacency matrix: `f = w * node.cost + (1 - w)* heuristic_cost`
- The `expand()` function takes as input the adjacency matrix, and the current node the A* algorithm is considering. The function calculates the possible child nodes that can be generated from the current node. For each potential child node, the function also calculates the cost of reaching that child node as well as the path that leads to that node. The function then returns a list of children along with their configuration (state, path, and cost).

Configurations and Results

Starting node	Ending node	Weight w	# of Nodes generated	Length of path	Sequence of nodes on path
0	19	0.5	47	13	[0, 10, 11, 12, 13, 14, 24, 25, 15, 16, 17, 18, 19]
0	19	0	33	13	[0, 1, 2, 12, 13, 14, 24, 25, 15, 16, 17, 18, 19]
11	97	0.5	52	15	[11, 21, 31, 32, 33, 34, 44, 45, 46, 56, 66, 76, 77, 87, 97]
11	97	0.25	52	15	[11, 21, 31, 32, 33, 34, 44, 45, 46, 56, 66, 76, 77, 87, 97]
40	49	0.5	83	16	[40, 41, 51, 52, 42, 43, 44, 45, 46, 47, 37, 38, 28, 29, 39, 49]
49	40	0.5	54	14	[49, 59, 58, 48, 38, 37, 36, 35, 34, 33, 32, 31, 30, 40]
0	99	0.5	959	-	Failure, no path found
99	0	0.5	52	19	[99, 98, 97, 87, 77, 76, 75, 65, 64, 54, 44, 34, 33, 23, 22, 21, 20, 10, 0]
99	0	0.25	52	19	[99, 98, 97, 87, 77, 76, 75, 65, 64, 54, 44, 34, 33, 23, 22, 21, 20, 10, 0]