Report LINGI2261: Assignment 2

Group N°160

Student1: Camille Caulier

Student2: Noah Meluzola

March 10, 2022

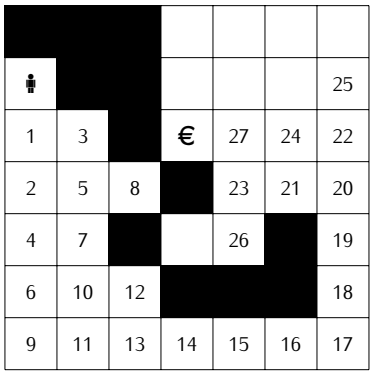
1 Search Algorithms and their relations (3 pts)

Consider the maze problems given on Figure 1. The goal is to find a path from 🚹 to € moving up, down, left or right. The black cells represent walls. This question must be answered by hand and doesn't require any programming.

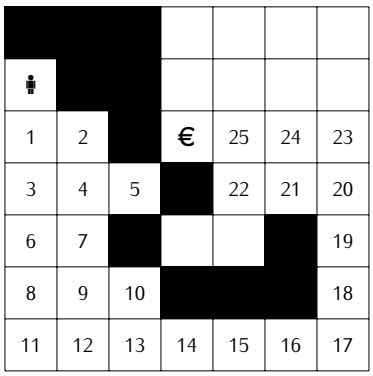
1. Give a consistent heuristic for this problem. Prove that it is consistent. Also prove that it is admissible. **(1 pt)**

A consistant heuristic for this problem would be using the manhattan distance (IE we take the distance of the direct line from the source to the goal). This heuristic is consistent since $h(n \rightarrow goal) \leq c(n, a, n') + h(n' \rightarrow goal)$ is always respected. Since we are only able to move up, right, down and left in increments of one unit the inequality is respected always due to the fact that h(n) will take the direct path, being able to choose any direction thus taking the "perfect path". From that we can deduce that $h(n \rightarrow n') \leq c(n, a, n')$ (IE for every action we take to go to a node, h(n) will be cheaper since it is able to choose any direction and distance) and through the triangle equality we have $h(n \rightarrow goal) \leq h(n \rightarrow n') + h(n' \rightarrow goal) \leq c(n, a, n') + h(n' \rightarrow goal)$

2. Show on the left maze the states (board positions) that are visited when performing a uniform-cost graph search, by writing the order numbers in the relevant cells. We assume that when different states in the fringe have the smallest value, the algorithm chooses the state with the smallest coordinate (i, j) ($(0, 0)$ being the bottom left position, i being the horizontal index and j the vertical one) using a lexicographical order. **(1 pt)**



3. Show on the right maze the board positions visited by A^* graph search with a manhattan distance heuristic (ignoring walls), by writing the order numbers in the relevant cells. A state is visited when it is selected in the fringe and expanded. When several states have the smallest path cost, they are visited in the same lexicographical order as the one used for uniform-cost graph search. **(1 pt)**



2 PageCollect problem (17 pts)

1. Model the PageCollect problem as a search problem; describe: **(2 pts)**
   - States
   - Initial state
   - Actions / Transition model
   - Goal test
   - Path cost function

State: the grid. In particular the postion of @ ,the presence of the papers on the grid (if they're taken or not) and X.

Initial state: the grid that we load at the beginning where we haven't moved yet.

Actions: move one cell up, right down or left. And also take the paper if we leave the cell with the paper or enter the cell with the X.

Goal test: if we're on the X and that there are no more papers left.

Path cost function: the number of cells traversed in order to reach the goal.

2. Give an upper bound on the number of different states for a PageCollect problem with a map of size $n \times m$, with k pages to collect. Justify your answer precisely. **(1 pt)**

(note: we consider that after the k pages we have to go to the door and that we are searching for the optimal path) Say our algorithm is very bad, for one page, we would have to explore n*m cells in worse case. Since our problem is to find the path from one page to another this means our path is a permutation of the order of pages that we have to take. This means that we would have a factor of k! and since the maximum cost to find the next page in worse case is n*m this would mean that the upper bound would be n*m*k!. We must also note that for eah permutation that we must remove $\sum^{k-1} i$ this is because if there are k pages in worse case wee need to visit m*n - (k-1) nodes since this affects every permutation we have to add a coefficient of k!. We also must add to our upper bound the cells that we would have to visit in order to get to our door which would me n*m in worse case once again. This finally gives us an upper bound of $n * m * (k!) - [(k!) \sum^{k-1} i] + n * m$.

3. Give an admissible heuristic for a PageCollect instance with k pages. Prove that it is admissible. What is its complexity ? **(2 pts)**

The heuristic used for our problem was as such
$h(n) = \sum^{no\_points} distance(p_i$, closest_point_to $p_i)$ - smallest_distance_between_points
Where p_i is either a paper, person or door. In layman terms for each point we find the distance to the closest point. Whilst we do that, we keep track of the smallest distance to remove it from the sum. This is to ensure admissibility since the goal of this heuristic is to have an approximation of the distance that we would have to travel and not removing a distance would mean that we would be adding an extra connection to the path approximation. To be more precise for n points we would have n connections instead of n-1. Our complexity is $\mathcal{O}(n^2)$. Since we we take the manhattan distance between the closest points and remove the smallest distance from the sum our heuristic can never overestimate since it will never make a path but instead "pair" points to their closest neighbour this allows us to have a rough estimate yet at the same time be admissible

5. **Implement** your solver. Extend the *Problem* class and implement the necessary methods and other class(es) if necessary. **(1 pt)**

6. **Experiment**, compare and analyze informed (*astar_search*) and uninformed (*breadth_first_graph_search*) graph search of aima-python3 on the 10 instances of PageCollect provided. Report in a table the time, the number of explored nodes and the number of steps to reach the solution. Are the number of explored nodes always smaller with *astar_search*? What about the computation time? Why? When no solution can be found by a strategy in a reasonable time (say **3 min**), indicate the reason (time-out and/or exceeded the memory). **(4 pts for the whole question)**

The number of nodes explored with *astar_search* is always smaller but the computation is not necessarily faster all the time. the reason why we don't explore as many nodes is due to the heuristic which tries to "guide" us to the goal state. On the other hand with our heuristic having a complexity of $\mathcal{O}(n^2)$ slows our code considerably and the tradeoff is somewhat argumentative (eg instance 5) this algorithm would be more appropriate if we were dealing with a transfer function that had a higher cost.
We can also note that as the problem increases in paper the frontier grows considerably and the heuristic does not help

	$A^\star$ Graph			BFS Graph		
Inst.	NS	T(s)	EN	NS	T(s)	EN
i01	19	0.0075	103	19	0.00413	194
i02	11	0.0011	19	11	0.001526	68
i03	37	0.015	223	37	0.00741	313
i04	26	0.0048	91	26	0.002586	129
i05		Timeout			Timeout	
i06	41	0.1824	884	41	0.1637594	2685
i07	21	0.0031	61	21	0.0023099	104
i08	70	9.1485	10030	70	2.4770708	11914
i09	21	0.0028775	57	21	0.002145	100
i10	11	0.0012173	21	11	0.0013893	64

NS: Number of steps — **T**: Time — **EN**: Explored nodes

6. **Submit** your program on INGInious, using the A^* algorithm with your best heuristic(s). Your file must be named *pagecollect.py*. You program must print to the standard output a solution to the PageCollect instance given in argument, satisfying the described output format. **(5 pts)**