

Report LINGI2261: Assignment 2



Group N°26

Student1: 66732000 Manuelle Ndamtang

Student2: 39311400 Xavier Claude

March 11, 2021

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)

As heuristic function, we can use the Manhattan distance between the start point and the target. This function is consistent: First, by principle, the distance between a point and itself is always equal to 0. So $|x_g - x_g| + |y_g - y_g| = 0$ (g for goal). Second, the possible actions are moving up, down left or right. Whereas the cost will always increase by 1, the Manhattan distance will at most increase or decrease by 1. So the condition $h(N) \leq c(N, P) + h(P)$ is always verified. It is also admissible : With such 2D displacements, the Manhattan distance is the shortest path between a point and its target, therefore it IS the optimal cost to reach target. Therefore, we can be sure that, with obstacles, $h(N) \leq h^*(N)$ with h^* the optimal cost to reach the target.

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)

6	4	2	1			
8	7	5	3			
9						
10	12					
11	14		21			
13			19	22		
15	16	17	18	20	23	

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)

10	6	4	1	⚡		
7	5	3	2			
8						
9	11		€			
12	13		20			
14			19			
15	16	17	18			

2 Blocks planning problem (17 pts)

1. Model the Blocks planning problem as a search problem; describe: (2 pts)

- States
- Initial state
- Actions / Transition model
- Goal test
- Path cost function

- States: They are represented by 2D arrays containing blocks, walls and empty spaces where moving is possible.
- Initial state: 2D array with blocks located anywhere with at least one block that has to reach its target.
- Actions: the blocks can move left and right. If it has empty space below it, it falls until it reaches a wall "#" or a "@" symbol indicating that a block reached its goal. These two symbols can also prevent a block from moving left or right.
- Goal test: Check if every target position of the goal state provided is filled with "@".
- Path cost function: Every time a block moves left or right, the path cost is increased by 1.

2. Consider the following state for the a01 instance:

```
#####  
#           #  
#  c       #  
#  a       #  
####      #  
####      #  
##  b     #  
#####
```

According to the goal state, such a situation cannot lead to a solution. Can you find other similar situations (in general, not only on that specific instance) that leads to a deadlock? If so, describe two. (2 pts)

It was impossible to draw a grid in the answers frame, here is the description of two possible dead states:

- On instance a01: "b" needs "a" to be rightly positioned in order to pass the hole and reach its target "B". If "a" is not at the extreme right position and "b" still tries to pass, then b is definitely blocked. It is on the same line than its target with walls between them. As "b" cannot move up, it will never be able to reach "B".
- On instance a07: We have 3 "b" and 1 "a" for targets "A" and "B". If "a" goes right and falls, it will be below its target "A", which makes it impossible to reach it. Note that "b"s may fall below their target "B" as long as one "b" remains above, being able to reach "B".

These two particular problems have been implemented in our algorithm to discard dead states.

3. Why is it important to identify dead states? How are you going to take it into account in your solver? (2 pts)

Dead states are viewed by the algorithm as normal states that should be explored and expanded. As a result, they lead to useless and tremendous resources consumption. To avoid them we have to detect them and discard them (what we did), or give them a large heuristic value. In our algorithm, they are detected with what follows: A block is considered unavailable if it cannot reach its target anymore (fell below target or cannot reach it because of "@" or "#" on same height). Once all unavailable blocks have been discarded, we check whether there are enough available blocks (ex: "a"s) to fill corresponding free targets (here, "A"s) or not. If not, this is a dead state.

4. **Describe** a possible (non trivial) heuristic to reach a goal state. Is your heuristic admissible and/or consistent? Why ? (2 pts)

A possible heuristic could be the sum of horizontal distances between blocks and their target. It would be both admissible and consistent in this case (Manhattan distance is not admissible here). Without dead states, it would find the optimal path. But the execution time can become huge. We decided to tradeoff execution time and optimality with an overestimation of the path cost that takes into account : horizontal distances between blocks and goals, distance between blocks and "holes", and a factor increased when blocks are below any target. In the algorithm, it takes the (very empirical) form: $h = hor_dist \times 2.5 + dist_fall \times 1.5 + below_goal$ which is neither admissible nor consistent (it is an overestimation), but leads to the optimal solution for 9 problems out of 10.

5. **Implement** this problem. Extend the *Problem* class and implement the necessary methods and other class(es) if necessary. **Experiment**, compare and analyze informed (*astar_graph_search*) and uninformed (*breadth_first_graph_search*) graph searches of aima-python3 on the 10 instances of Blocks planning provided. Report in a table the time, the number of explored nodes, the number of remaining nodes in the queue and the number of steps to reach each solution. When no solution can be found by a strategy in a reasonable time (say **1 min**), indicate the reason (time-out and/or swap of the memory).

Are the number of explored nodes always smaller with *astar_graph_search*? What about the computation time? Why? (3 pts)

Inst.	A* Graph				BFS Graph			
	NS	T(s)	EN	RNQ	NS	T(s)	EN	RNQ
a01	12	0.0129	22	23	12	0.0264	244	20
a02	11	0.0031	12	1	11	0.0012	13	0
a03	5	0.0082	25	26	5	0.0295	219	102
a04	20	0.1059	403	13	20	0.0484	476	2
a05	48	0.3381	768	1009	/	TO	/	/
a06	21	0.1104	83	269	21	5.789	31541	137
a07	16	0.6221	713	1232	16	4.13	20680	2126
a08	23	0.3381	768	1009	18	6.8143	48959	13132
a09	25	4.1318	6260	3795	/	TO	/	/
a10	38	28.4709	66370	26957	38	46.5174	280038	401

NS: Number of steps — T: Time — EN: Explored nodes — RNQ: Remaining nodes in the queue

6. **Submit** your program on INGIInious, using the A^* algorithm with your best heuristic(s). Your program must print to the standard output given a time limit of 1 minute, a solution to the Blocks planning instance passed as parameter to it, satisfying the described output format. Your program will be evaluated on 11 instances, one of which is hidden. We expect you to solve at least 8 out of the 11. (6 pts)

BFS is way more demanding as the problem complexity increases (indeed, it tries every possibility). For the problems where blocks need to "work together" (support each other, free space), BFS number of explored nodes and execution time explode (or timeout). A^* has always less explored nodes, because the "best" node (in regards of the heuristic) is always popped from the PriorityQueue. As a conclusion, whereas performances for simple instances are quiet even for both algorithms, the positive effect of the heuristic for A^* becomes very clear when the complexity and the required interactions between blocks increase.