INFO8006: Project 1 - Report

Mohamed-Anass Gallass - s174796
Yassine Maziane - s184384

October 11, 2020

1 Problem statement

a. The **set of possible states** consists in the pacman position (x,y) and a boolean matrix representing the positions of the dots in the map. The initial state correponds to the start position of the pacman.

b. The **set of legal actions** is defined by 4 potential actions: North, South, East, West. The legality of each action must respect the border of the map.

c. The **transition model** gives the state reached after the pacman has executed an action. The position is updated and a dot or a capsule may have been eaten.

d. The **goal test** tells us whether all dots have been eaten

e. The **step cost** $state_i$ corresponds to the cost related to a step. The step cost at the initial state is equal to 0. If pacman eats a capsule after an action, the step cost is increased by 6, otherwise the step cost is increased by 1. We chose to increase the step cost of 6 after eating a capsule because in order to eat one, pacman must make a move with a cost of 1 and eat the capsule with a cost of 5.

$$state_i = \begin{cases} 0 & \text{if i = 0 (initial state)} \\ state_{i-1} + 6 & \text{if a capsule has bean eaten in state i} \\ state_{i-1} + 1 & \text{if no capsule has been eaten in state i} \end{cases}$$

2 Implementation

a. The problem in the implementation of DFS is due to the fact that a state is defined only as the position of the pacman. This had the consequence that the pacman could never pass twice through the same position which is a big issue. Indeed, it would not have been able to get a food dot located in a dead-end for example. To remedy this, we add the boolean matrix representing the positions of the dots in the map in the definition of a state.

b. *Leave empty.*

c. Our heuristic $h(n)$ is defined as follow:

$$h(n) = \max_{\forall i \in foodList(n)} \{M_d(P_{pacman}(n), foodlist(n)[i])\}$$

where $foodList$ is a list containing all the positions of the dots at state n, M_d the Manhattan Distance between two positions, $P_{pacman}(n)$ the position of the pacman at state n, $foodList(n)[i]$ the position of the dot i at state n.

Our cost function $g(n)$ is defined as follow:

$$g(n) = \begin{cases} 0 & \text{if n = 0 (initial state)} \\ g(n-1)+6 & \text{if a capsule has bean eaten in state i} \\ g(n-1)+1 & \text{if no capsule has been eaten in state i} \end{cases}$$

A^* is optimal because our heuristic will always underestimate the path cost to the furthest food dot. So, our heuristic is admissible.

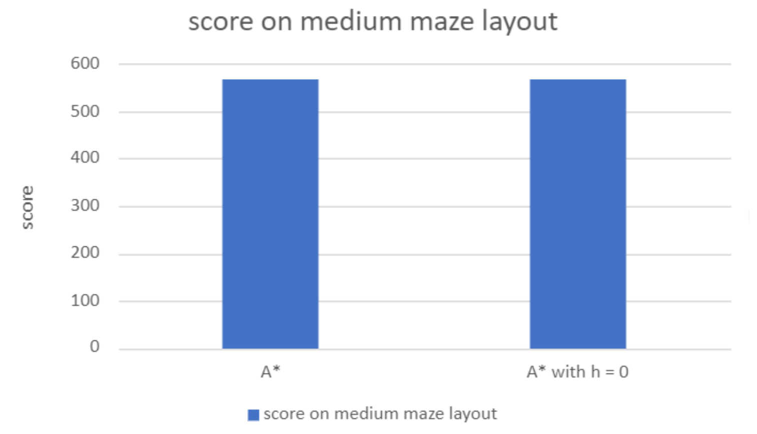
d. $g(n)$ preserves completeness because $g(n) \geq 0 \forall n$. With $h(n) = 0$, A^* becomes an Uniform-Cost Seach (UCS) algorithm and the optimatlity is preserved because UCS expands nodes in order of their optimal path cost.

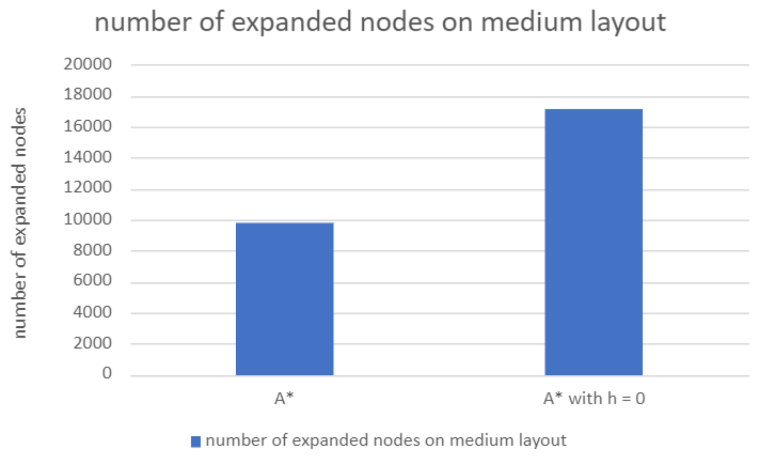
e. *Leave empty.*

f. To transform A^* into BFS, we choose $h(n)$ and $g(n)$ constant. So, the priority queue becomes a FIFO queue because there is no longer priority between the nodes.

3 Experiment 1

a. Our results in graphical form are as follows:





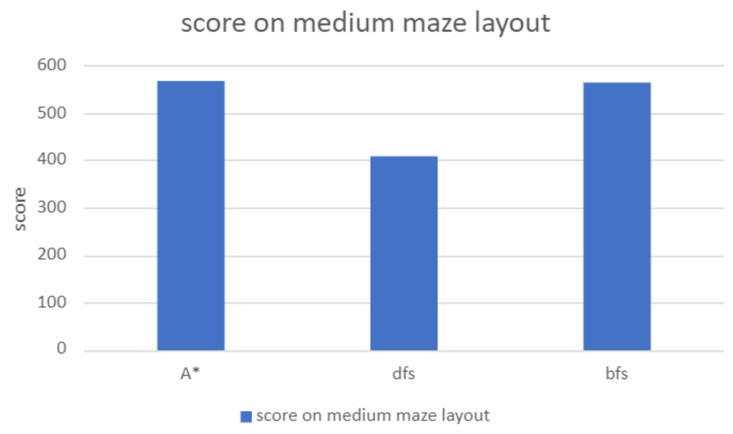
b. Both versions of A* manage to score 568 in the medium maze layout however, A* with an identically null heuristic needs to expand many more nodes.

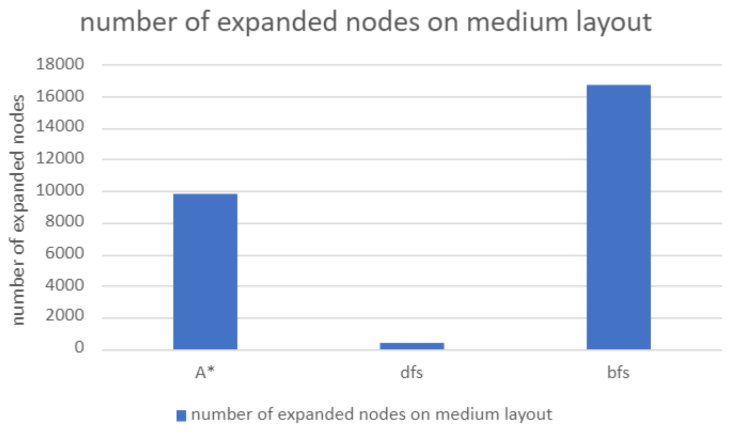
c. In both cases, the heuristic is admissible, therefore the solutions returned will be optimal solutions but as it can be seen above, the better the heuristic, the less the nodes have to be expanded. Our heuristic computes the manhattan distance between pacman's current position and the furthest food dot.The heuristic will always underestimate the path cost to the furthest food dot. The heuristic will orient the search and expand less nodes since ours is strictly positive.

d. A* with a null heuristic corresponds to the Uniform Cost Search (UCS) algorithm.

4 Experiment 2

a.





b. On the medium maze layout, A* gets a score of 568 by expanding 9770 nodes while dfs only gets a score of 409 by expanding only 361 nodes. Bfs manages to get a score of 565 but it does so by expanding 17198 nodes!

c. Dfs is a very straightforward solution, it explores the nodes as deep as possible following an order convention and stops as soon as it finds a valid solution not caring about the cost of the path leading to that solution. Furthermore, neither bfs nor dfs cares about the capsules. Bfs will find the shallowest solution not caring about its cost however some paths may be deeper but with a lesser cost. A* guarantees the optimal path and leads to the best score given an admissible heuristic. Notice that heuristics play a strong role in limitating the number of expanded nodes.