University of Liège

Artificial Intelligence

Pacman - Search Agents

Master 1 in Data Science & Engineering

Authors :
Tom Crasset
Antoine Louis

Professors :
G. Louppe

Academic year 2018-2019

1 Formalizing the game

The game of Pacman can be formalized as a search problem, which consists of a state space, a goal test and a path cost.

1.1 State space

The state space defines the set of all states reachable from the initial state by any sequence of action. It consists of :

- the initial state of the agent
- the actions available to that agent, given a state s
- a transition model that returns a state s' that results from doing action a in state s

At this stage, Pacman is the only agent. A state is defined by the position of the agent and by a grid of boolean food indicators. So the initial state is defined by its initial position in the grid and by the initial grid of food dots present on the layout.

The actions are defined as the Pacman agent moving from its position to one position to the north, the south, the east or the west.

The transition model is defined as the position the Pacman agent arrives to after moving to it, and possibly after having eaten a food dot.

To summarize, depending of the action a available to our Pacman agent at a given state $s = \{(i, j),\ dot\ booleans\}$, the result will be :

- if going NORTH, st. s' = result(s,a) = $\{(i, j + 1),\ dot\ booleans'\}$
- if going SOUTH, st. s' = result(s,a) = $\{(i, j - 1),\ dot\ booleans'\}$
- if going EAST, st. s' = result(s,a) = $\{(i + 1, j),\ dot\ booleans'\}$
- if going WEST, st. s' = result(s,a) = $\{(i - 1, j),\ dot\ booleans'\}$

1.2 Goal test

The goal of this particular problem is to reach a goal state where all the food dots have been eaten. That means that a sequence of actions from the transition model has been made by our agent from the initial state of that agent to the goal state.

1.3 Path cost

The path cost corresponds to a sum of positive step cost which associates a numeric value to the action a in state s leading to state s'. Here, we assume that the cost of an action is the same in all states. So the path cost could correspond to the total number of actions executed to reach the goal, a better path corresponding to a minimum number of actions. The path cost is then closely linked to the score function which, for the moment, only depends on the time that Pacman takes to reach the goal, the number of eaten food dots and if it wins the game by reaching the goal :

$$score = -\#time\ steps + 10 * \#number\ of\ eaten\ foods + 500\ if\ winning\ end$$

2 Bar plots

After having implemented our Pacman agent with four different tree search algorithms (DFS, BFS, UCS, A*), some bar plots could be useful to analyze the performance of each of them in the different layouts in terms of total computation time, total number of expanded nodes and final score.

2.1 Computation time

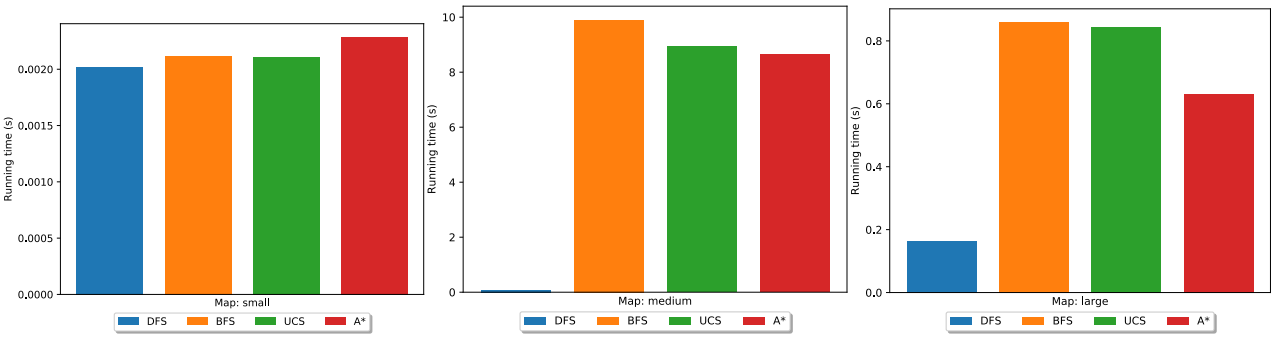

FIGURE 1 – Total computation times of the different agents on all maps

2.2 Expanded nodes

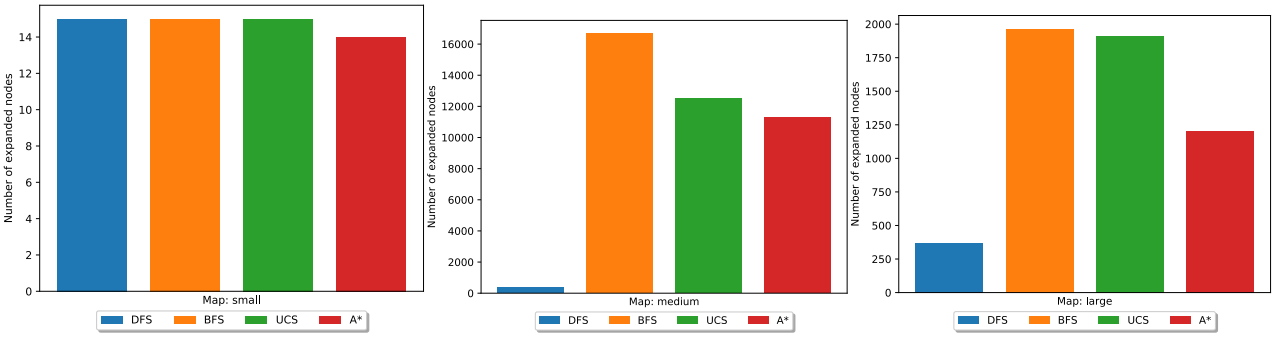

FIGURE 2 – Total number of expanded nodes of the different agents on all maps

2.3 Final score

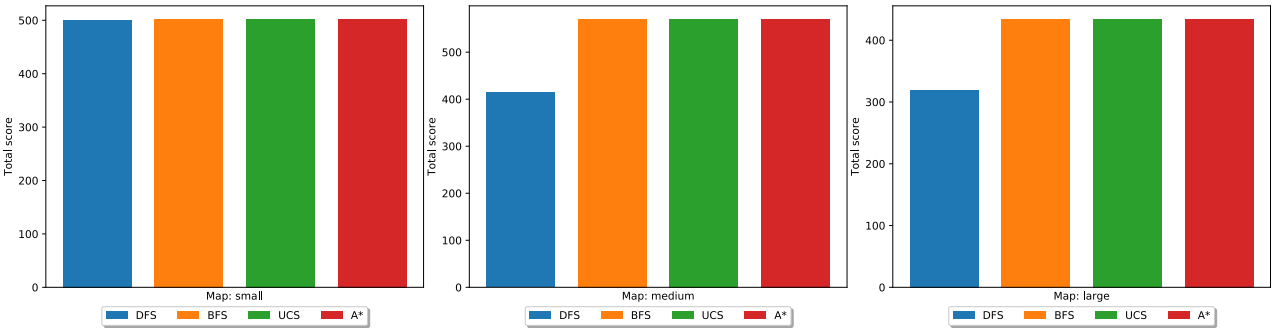

FIGURE 3 – Final score of the different agents on all maps

3 Performance analysis

The observations made from the Figure 1 show that the DFS algorithm was the fastest across all maps in comparison to other search algorithms, especially for the medium layout where it performed significantly faster. These fast computation times comes with a very small number of expanded nodes, particularly for the medium and the large layout. However, these speeds come on behalf of the score, which is also the worst across all layouts for that algorithm as can be seen on Figure 3. It is clearly not the optimal solution and it was to be expected, DFS finds the leftmost solution regardless of depth or cost.

The optimal solution was however achieved by several algorithms. As illustrated on Figure 3, this feat was accomplished across all maps by the BFS, the UCS and the A* algorithm. This was to be expected for each one of them. Indeed, BFS is optimal when the path cost is a non-decreasing function of the depth of the node, which is the case here, UCS expands the nodes in order of their optimal path cost and A* is simply an improvement of the UCS algorithm.

Out of these three algorithms, BFS is probably the less performing with the biggest number of expanded nodes resulting in the highest computation time. In contrast, one of them stands out : the A* algorithm. In comparison to the others, it had a faster computing time (Figure 1) as well as fewer number of nodes expanded (Figure 2) while still finding the optimal path for Pacman to take on each layout. For A* to be optimal, an admissible heuristic must be found, a heuristic h for which :

$$0 \leq h(n) \leq h^*(n)$$

where $h^*(n)$ is the true cost to a nearest goal. In the case of a 2D grid, the Manhattan distance seems to be one of the best choices. But the goal here is not to eat one particular food dot, but to eat all food dots. Different heuristics using the Manhattan distance could be implemented to this end :

- Sum of Manhattan distances : the idea consists of adding together the Manhattan distances from the current state to all food dots. Note that this heuristic is not admissible and thus doesn't guarantee an optimal path.
- Minimum of Manhattan distances : the idea consists of finding the Manhattan distances from the current state to all food dots and keeping the minimum one. This heuristic is admissible but it doesn't approach well the real cost.
- Maximum of Manhattan distances : the idea consists of finding the Manhattan distances from the current state to all food dots and keeping the maximum one. This heuristic is admissible and moreover, is a better approximation of the real cost than the previous heuristic.

For these reasons, the last heuristic seems to be a good choice, even if some better ones could be implemented. For example, the better admissible heuristic would be the one that computes the shortest path from the Pacman agent to all food dots in term of Manhattan distances. But finding this path would increase a lot the total computational time and that's why taking the maximum of the Manhattan distances is a good compromise.

Concerning the limitations of these search algorithms, they are clearly not appropriate for more complex layouts or layouts with a lot of foods. A sharp increase in computation time and total number of expanded nodes is clearly visible when passing from the small layout, where there is only one food, to the medium layout, where there are 13 of them. A noticeable increase also happens when passing from the small layout to the large one, where the number of foods isn't too different.