Faculty of Applied Science
University of Liege
October 18, 2018

Introduction to Artificial Intelligence

Part I Search Agent

October 2018

Derroitte Natan
Testouri Mehdi

Prof. G.Louppe

1 Formalisation of the search problem

A search problem is defined by several components : a state representation, the initial state, the actions, a transition model, a goal test, a path cost and a solution. These components are described in this section.

1. **State representation**
   The state s is represented by a tuple formed by the position of Pacman and the food matrix (matrix giving the positions of the remaining foods). More formally :
   $$s = ((i, j), fm)$$
   where i is the position of Pacman along x, j the position along y and fm is the food matrix.

2. **Initial state**
   The initial state $s_0 = ((i_0, j_0), fm_0)$ is simply the initial position of Pacman and the initial food matrix.

3. **Actions**
   The set of actions $actions(s)$ are all the allowed moves for the given Pacman state s. The possible allowed actions are 'North', 'South', 'East' and 'West'.

4. **Transition model**
   The transition model $result(s, a)$ is the following, assuming an initial state $s = ((i, j), fm)$.
   - $s' = result(s, North) = ((i, j + 1), fm')$
   - $s' = result(s, South) = ((i, j - 1), fm')$
   - $s' = result(s, East) = ((i + 1, j), fm')$
   - $s' = result(s, West) = ((i - 1, j), fm')$

   Where fm' is the food matrix modified if the new position of Pacman contained a food.

5. **Goal test**
   The goal test simply evaluates if the current food matrix fm corresponding to the current state s doesn't contain any food which means that Pacman has eaten all of them.

6. **Path cost**
   Since every move are equivalent in term of distance, the cost of moving from one node to another is constant and set to 1.
   $$c(s, a, s') = 1 \quad \forall a, s, s'$$

7. **Solution and optimal solution**
   A solution $path = \{a_0, a_1, .., a_n\}$ is a series of actions leading to a goal state s_g from an initial state s_0. An optimal solution is a solution that minimises the cost $c(s_0, path, s_g)$ i.e the cost of the path used to go from s_0 to s_g.

2 Algorithms Comparison

2.1 Observations

In order to obtain a complete comparison, it is necessary to take into account 3 main aspects: the **computation time**, the final **score** and the number of **explored nodes**.
In the following figures, we make these comparisons for the different layouts available : *small*, *medium* and *large*.

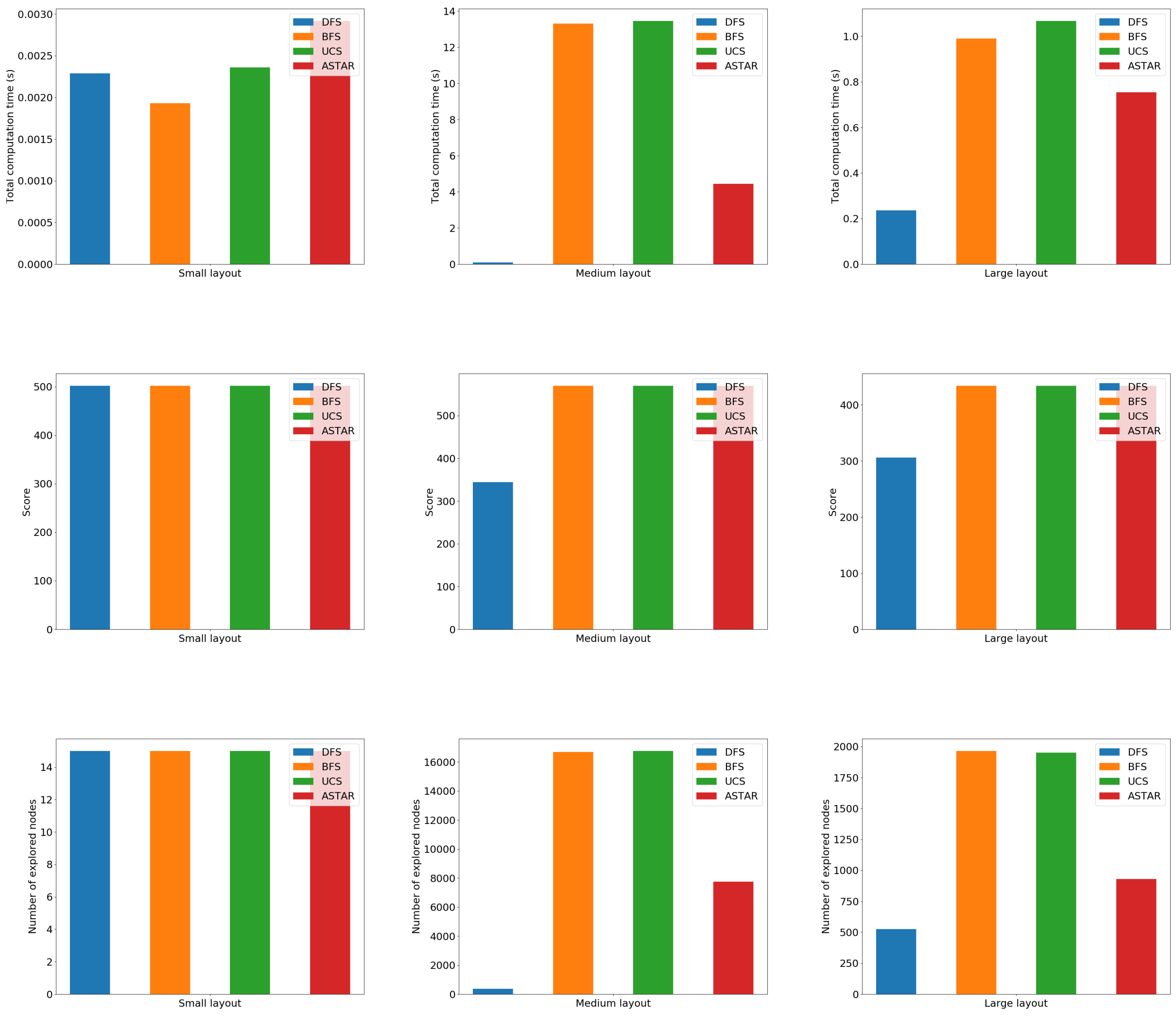

Figure 1: Comparison of the score for the different maps. From up to down : Computation time, Scores and Number of explored nodes. From left to right : small, medium and large layout

2.2 Invariance of results on the small map

The first thing we can observe is that the small map shows similar results, both in terms of score and number of nodes explored. The execution time shows visual variations but looking at the time scale, we conclude the same thing.

It seems that the number of possible states of this map does not allow us to see any real difference between the algorithms.

2.3 Time and explored nodes variation with the number of food and the size of the map

The first thing that jumps to our eyes is that the medium map requires a much longer computation time than for the other two maps. This is true for all the algorithms tested except DFS, which also see its calculation time increase, but much more slightly.

In order to understand these results, we must understand that the larger and deeper the state tree we have to treat, the more likely it is that we will have to explore more nodes to find a solution.

Two factors can affect the size of the tree: the position and the food matrix since these two variables are included in the states of our problem.

The number of possible positions increases linearly with the size of the map.
However, the number of food matrix possible corresponds to 2^n where n is the number of possible food. We see that this matrix number increases exponentially with n.

Therefore it is possible for us to understand why the number of nodes explored increases only slightly between the small and large map: although the map is growing, the number of foods only increases lightly. In the case of the map medium, the number of foods is increasing sharply and the result is a much larger number of nodes explored. The execution time is related to these results: the more nodes we have to explore, the longer it will take us to get a result.

2.4 Score variation with respect with the algorithm

The first thing that can be observed is the DFS algorithm. Although faster, both in terms of number of nodes explored and execution time, the score is lower than in the case of other algorithms.
It simply means that this algorithm finds a solution quickly but that it is not optimal.

In our problem, the shallowest solution is the optimal solution. Indeed, the depth of a node corresponds to the size of the path to access this point. Knowing this, it appears that the BFS algorithm will necessarily find an optimal solution. His score is therefore optimal.

Not having fixed a specific weight for certain directions or positions, each move has an equivalent unit cost. This explains why the UCS algorithm is equivalent to BFS, both in score and number of nodes explored and in time. The only difference that could come to light is that BFS explores the nodes from "left to right" while UCS does not have any guidance on how it should treat nodes that have the same cost.

Finally, we see that A* also finds the optimal score. However, thanks to its heuristic, the algorithm no longer uses unit costs to move from one position to another, which allows it to find the optimal solution in fewer nodes explored and therefore less time.

3 Implementation

3.1 Hash

In the implementation, the states are represented by a tuple containing the current position of Pacman and the current food matrix. However, the element stored in the *discovered* list are hashes of their corresponding states. This has been done for performance reasons, indeed, comparing two integers is much faster than comparing two tuples and since those operations of comparison are executed at every node exploration, it saves computation time (about 10 to 20 seconds on the used machines).

3.2 A* Heuristic

We first thought of using Manathan distance. Working in a grid where the only possible movements are horizontal or vertical, this one seems particularly appropriate to us as an evaluation of the distance.
Considering a situation where only one food is present, we want to minimise this distance.
In the less trivial case where many foods are present, we have decided to minimise all Manathan distances between the Pacman position and the different foods. In this way, the algorithm tends to favours the path that leads to an area were the remaining foods are closer to Pacman.

As seen above this heuristic reduced the number of node explored and thus make the algorithm faster.

4 Possible improvements

4.1 Alternative approach

An alternative approach can be to use subgoals and a super goal. For example, the subgoals can be to find a food and the super goal to find all food.
In this case, the tree search is done for one food at a time. Pacman therefore goes to the found food and then another search is done for the next food until there is no food left.
In this case, there is no assurance that the solution is optimal, however, the search is significantly faster.

/	BFS^*	BFS
Computation time (s)	0.034	13.314
Scores	562	570
Explored Nodes	132	16688

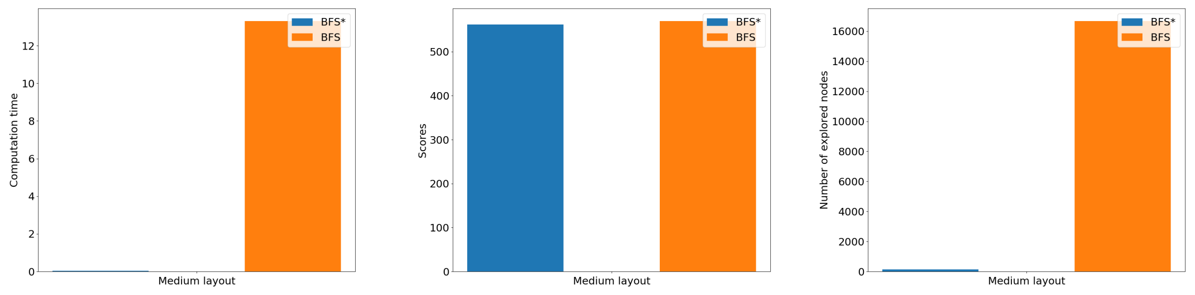

Figure 2: Comparaision for BFS algorithm BFS* which correspond to the modification of BFS as introduced in this section. From left to right : computation time, scores, number of explored nodes

4.2 Alternative heuristic

An other heuristic could be used in the A* algorithm to try to optimise it in terms of score and computation time.

4.3 Improve performance

Even if the hash has allowed to improve the performance, the search of the solution is still slow (around 5s), especially if there is a lot of food. Thus, some improvements could be done to improve performance.