Assignment 8 - Rule-based AI

Nathan HAUDOT (12 hours), Hugo MATH (13 hours)

January 5, 2022

1 The branching factor 'd' of a directed graph is the maximum number of children (outer degree) of a node in the graph. Suppose that the shortest path between the initial state and a goal is of length 'r'.

1.a What is the maximum number of Breadth First Search (BFS) iterations required to reach the solution in terms of 'd' and 'r'?

BFS stands for breadth first search, so the algorithm will mark the children nodes at each iteration and choose nodes from left to right in the same "row" of the graph.

So, if we assume the worst scenario which is having the maximum amount of child nodes at each node, we quickly see that each iteration is each node. Thus, the number of growing nodes is following a geometrical progression with the branching factor d as common ratio, and the first number of nodes as the scale factor a which is 1 in our case: $U_{n+1} = d * U_n$ where $U_0 = a$. We can also write this as $U_n = ad^n$. Thus, the sum of this progression is written like this : $\sum_{k=1}^{n-1} ad^k = a\frac{1-d^n}{1-d} = \frac{1-d^n}{1-d}$.

Finally, the variable **n** stands for the length of the graph. In our case it's **r**. Finally we have : $\frac{1-d^r}{1-d}$. If we assume that **r** is the number of edges between two nodes, then this formula needs to be written as follow : $\frac{1-d^{r-1}}{1-d}$.

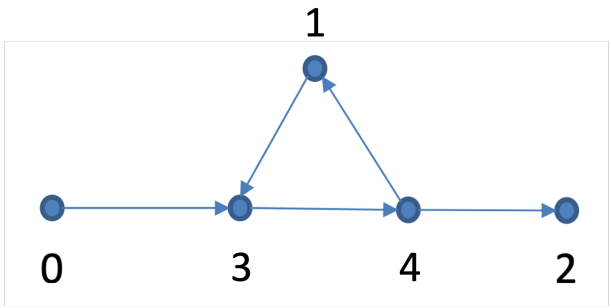
1.b Suppose that storing each node requires one unit of memory and the search algorithm stores each entire path as a string of nodes. A path with k nodes requires k units of memory. What is the maximum amount of memory required for BFS in terms of 'd' and 'r' ?

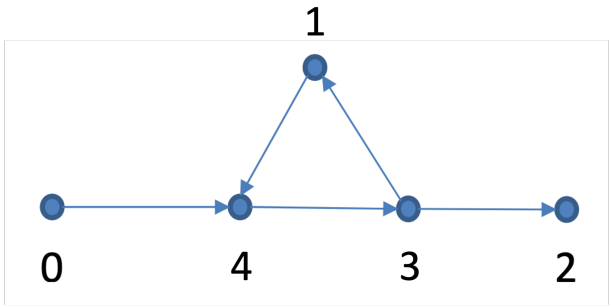
We can use a similar reasoning. Indeed, if we take the worst scenario, we take the maximum amount of child nodes in each node. Thus, storing paths of nodes means storing the geometric progression at step **r** with **k** unit for each nodes.

Then, we can reuse our geometric progression and write : $k_{max} = ad^r$. But **r** means again the length between the starting nodes and the last. So we shift the geometric progression to : $k_{max} = ad^{r-1} = d^{r-1}$ where **a** is the number of starting nodes.

2 Suppose that we use the Depth First Search (DFS) method and in the case of a tie, we chose the smaller label. Find all labelling of these three nodes, where DFS will never reach to the goal! Discuss how DFS should be modified to avoid this situation?

We can identify two cases in which the DFS method will never reach its goal. If we label the top node of the graph as "1", the algorithm will be stuck in an infinite loop. There are two labelings of these three nodes that cause problems :

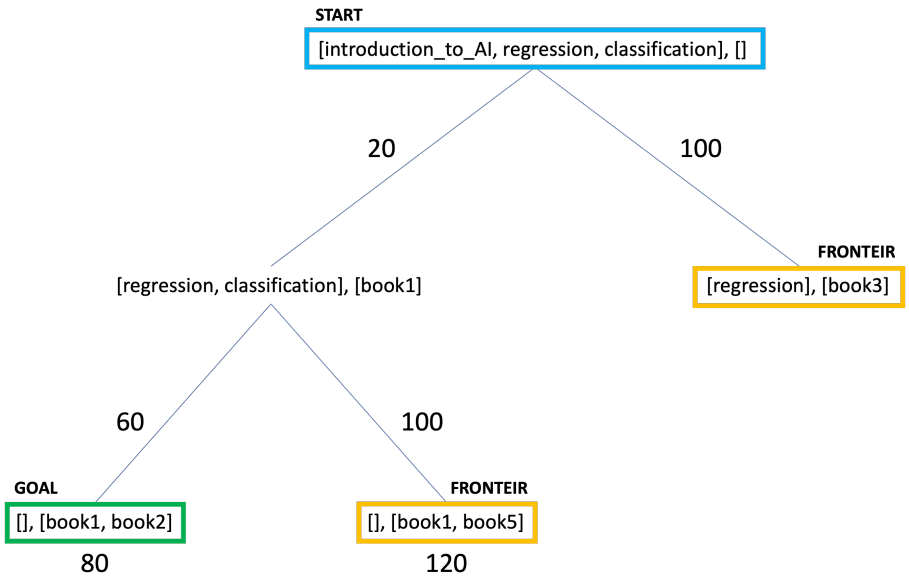




The first solution is to remember the previously visited nodes: we could then detect a repeating path pattern and, at a certain threshold, break the path choice to exit this loop (if possible). The second solution is to use a modified version of DFS called IDDFS. This algorithm uses DFS but with increasing depths (the search cannot go beyond this threshold), it is a way of doing "DFS" in the manner of "BFS".

3 A publisher allows teachers to "build" customised textbooks for their courses by concatenating the text from different books in their catalogue.

3.a Suppose a teacher requests a customised textbook that covers the topics [introduction_to_AI, regression, classification] and that the algorithm always selects the leftmost topic when generating child nodes of the current node. Draw (by hand) the search space as a tree expanded for a lowest-cost-first search, until the first solution is found. This should show all nodes expanded. Indicate which node is a goal node, and which node(s) are at the frontier when the goal is found.



The leftmost topic is "introduction_to_AI", it is treated first in the child nodes with books 1 and 3. Book 1 is the one that costs the least (20 pages), it is the one that we will prefer. The other node with book 3 is therefore a "frontier" node.

The last two topics "regression" and "classification" are both covered by books 2 and 5. Book 2 costs less (60 pages) than book 5 (100 pages), so we choose this one: this is our goal node. The other node with book 5 is therefore a "frontier" node.

3.b Give a non-trivial heuristic function h that is admissible. [h(n)=0 for all n is the trivial heuristic function.]

The heuristic value of a node in a graph tries to define the importance of the value of this node. The number of topics left to process in a node can tell us if we are progressing in the right direction or not: it will therefore be our heuristic value that will allow us to minimize the final cost.

4 Consider the problem of finding a path in the grid shown below from the position s to the position g. A piece can move on the grid horizontally or vertically, one square at a time. No step may be made into a forbidden shaded area. Each square is denoted by the xy coordinate. For example, s is 43 and g is 36. Consider the Manhattan distance as the heuristic. State and motivate any assumptions that you make.

4.a Write the first five paths selected by A* along with all stored paths at each selection, assuming that in the case of tie the algorithm prefers the path stored first.

Iteration 1

(x,y)	previous	dist from s	h(p)	dist from s + h(p)
(4,3)	(4,3)	0	4	4

Initialisation

Iteration 2 - (4,3)

(x,y)	previous	dist from s	h(p)	dist from s + h(p)
(4,4)	(4,3)	1	3	4
(5,3)	(4,3)	1	5	6
(4,2)	(4,3)	1	5	6

(4,4) smallest cost and heuristic : We choose this position

Iteration 3 - (4,4)

(x,y)	previous	dist from s	h(p)	dist from s + h(p)
(3,4)	(4,4)	2	2	4
(5,3)	(4,3)	1	5	6
(4,2)	(4,3)	1	5	6
(5,4)	(4,4)	2	4	6

(3,4) smallest cost and heuristic : We choose this position

Iteration 4 - (3,4)

(x,y)	previous	dist from s	h(p)	dist from s + h(p)
(4,2)	(4,3)	1	5	6
(5,3)	(4,3)	1	5	6
(5,4)	(4,4)	2	4	6

(5,4) smallest cost and heuristic : We choose this position (5,3) already visited

4.b Solve this problem using the software in http://qiao.github.io/PathFinding.js/visual/. Use Manhattan distance, no diagonal step and compare A*, BFS and Best-first search. Describe your observations. Explain how each of these methods reaches the solution. Discuss the efficiency of each of the methods for this situation/scenario.

Figure 1: A* - Best First Search - Breadth First Search

We found that Best-First-Search algorithm found the shortest path in 48 iterations whereas A* found it with 73 and Breadth-First-Search with 346.

Breadth-search algorithms (BFS, Dijktra..) don't work well on this type of graph because you have a lot of space and therefore a lot to discover, if you try to found the shorter way by breadth technique. Whereas algorithms like A* or Best-First-Search directly go to one direction by evaluating the neighbours at each iterations. This results in going to a straight line and then come back if there is frontier. This type of algorithm are much faster (0.5ms for A* and 0.8ms for Best-First-Search) but may not converge to the optimal path.

However, the graph problems where these algorithms are applied may modify a lot the results. In a maze for example, algorithm like A* might no be the most effective.

4.c Using a board like the board used in question 4a) or in http://qiao.github.io/PathFindi describe and draw a situation/scenario where Breadth-first search would find a shorter path to the goal compared to Greedy best-first search. Consider that a piece can move on the grid horizontally or vertically, but not diagonally. Explain why Breadth-first search finds a shorter path in this case.

We set this problem and run the two algorithms on it:

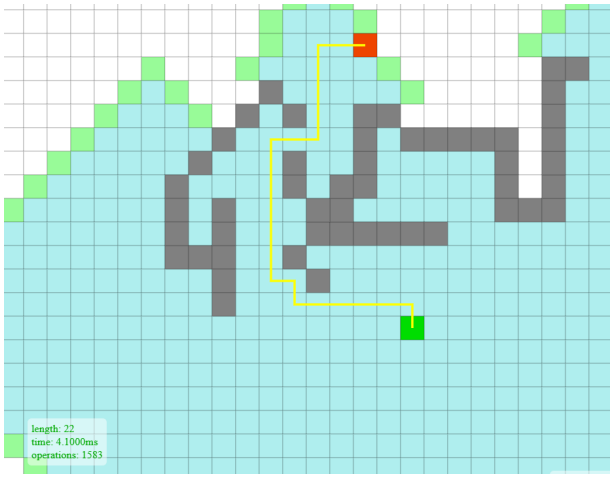


Figure 2: Breadth First Search

Breadth First Search algorithm find the solution in 1583 iterations with a path length of 22, whereas the Best First Search algorithm find it with a path length of 26 and 165 iterations.
Breadth First Search algorithm find a shortest because Best-First-Algorithm use an heuristic function called "Mannhatan" which directly goes and draw a straight line in the direction of the point to save the cost and then go along the frontiers. This result in finding a longer path than the Breadth First Search algorithm which "scan" the whole environment by sweeping the cases.

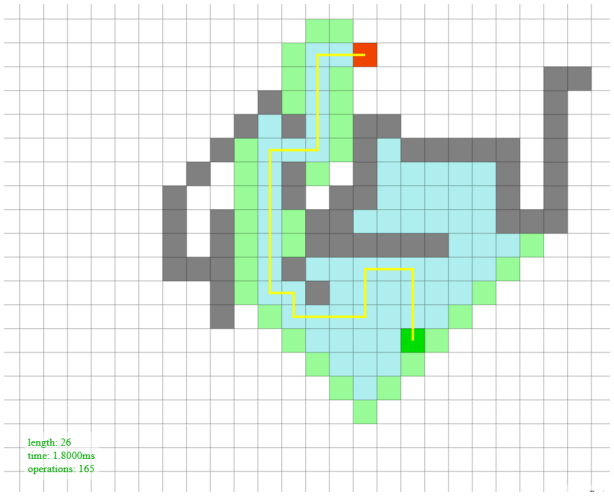


Figure 3: Best First Search

5 This question is about the relation of Markov decision processes in Assignment 5 and search algorithms.

5.a Discuss when and how the generic search problem can be described as a Markov decision process (MDP).

A generic search problem can be described as a Markov decision process by converting the problem environment to a space of states (defining, in part, the Markov chain) as well as actions between these states. These actions have a probability to move from one state to another. If we take the example of the problem in question 4, we would have a probability of 0 to go from the box (5,4) to the box (5,5) because the latter corresponds to a wall: this probability is not modifiable, it is part of the environment. The other actions have a probability between 0 and 1.

5.b When the search problem can be written as an MDP, what are the advantages and disadvantages of the value iteration algorithm over the A* algorithm?

The advantage of the value iteration algorithm over the A* algorithm is that it is able to find the shortest path, unlike the A* algorithm which finds a "convincing" short path. However, it has to perform enough iterations to find a decent optimal policy. Its disadvantage is therefore the computation time, where the A* algorithm is relatively fast in its execution.