DAT565 Assignment 8 (rule-based AI) – Group 43

Luca Modica - (5 hrs)
Hugo Alves Henriques E Silva - (5 hrs)
YenPo Lin - (5 hrs)

October 20, 2023

Problem 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 the goal is of length r.

Question a): what is the maximum number of breadth first search (BFS) iterations required to reach the solution in terms of d and r?

Assuming the we are using the uninformed BFS search algorithm, the number of iteration required rely on the maximum number of children that we can find in each level (let's define $l :=$ current level).

- At level $l = 0$ we only have the *root node*, so there is 1 node to explore and spend 1 iteration.
- At level $l = 1$, in worst cases, we will have d nodes, so d iteration will be needed. This because in BFS, every node in every level will be explored.
- At level $l = 2$, in worst case we will have to explore all of the children of the nodes in previous level. In other words, we will need d^2 iterations.

In general, the number of iterations needed can be expressed as the following summation:

$$1 + d + d^2 + ... + d^r = \sum_{l=0}^{r} d^l$$

where r can also express the number of levels that the algorithm explore in order to arrive to the goal state. Moreover, since the summation is a geometric series, the number of iteration can also be defined as:

$$\sum_{l=0}^{r} d^l = \frac{d^{r+1} - 1}{d - 1} \in O(d^{r+1})$$

where $O(d^{r+1})$ is the time complexity of the BFS, given the branching factor and the path length towards the solution of the related directed graph of possible states.

Question b): suppose the storing each node requires one unit of memory and the search algorithm stores each entire path as a list of nodes. Hence, storing 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?

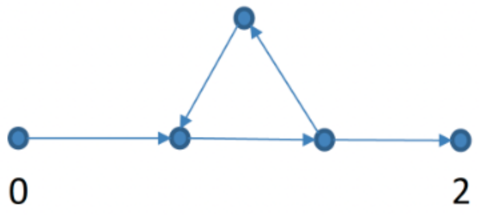
The case where the maximum amount of memory is required is when the algorithm reach the last level r, where the number of nodes to explore is d^{r+1} (including the root node in the first level). Thus, the maximum amount of memory required for BFS in terms of d and r will be:

$$d^{r+1} \in O(d^{r+1})$$

where $O(d^{r+1})$ is the space complexity of the BFS, given the branching factor and the path length towards the solution of the related directed graph of possible states.

Problem 2

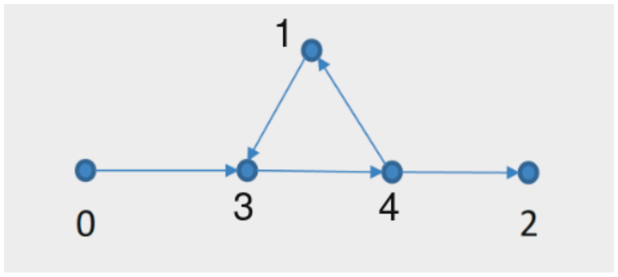
take the following graph where 0 and 2 are respectively the initial and the goal states. The nodes are to be labelled by 1, 3, and 4.

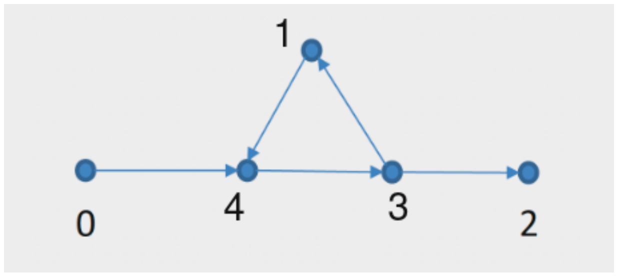


Suppose that we use the depth first search (DFS) method and, in the case of a tie, we choose the smaller label.

Question a) find all the labellings of these three nodes where DFS will never reach the goal.

The 2 possible labelings of the nodes that will nevee let the DFS reach the goal are the following.





In general, since in a case of a tie the algorithm will choose the smaller label, the approach is to put smaller numbers in alternative of the label in the goal state.

Question b): discuss how DFS should be modified to avoid this situation.

There are several modification that can be applied to the DFS algorithm in order to avoid the previous situation, especially related to how the search will solve a tie. 2 of them will be described.

- A first possible solution is to randomize the choice among the possible path alternatives. We can set, for example, a uniform probability of going in one of the possible nodes, for example:

$$P(d|s) = 1/n$$

  where d is the destination node, s the node from where we will move out and n the number of possible destination.

- Another approach can be to prioritize the exploration of the algorithm. In other words, in case of a tie we can let the DFS choose the node with less number of visits.

Problem 3

In the given scenario, *a publisher offers a system for teachers to create custom textbooks by selecting and combining content from various books in their catalog.* These books are associated with page numbers and topics they cover. The goal is to create a custom textbook that includes all the required topics, as defined in the "Topics" list, while minimizing the total number of pages used.

To achieve this goal, a node is defined as a combination of topics and books, and valid nodes should ensure that the selected books do not overlap in covering topics. Child nodes are created by selecting a topic, choosing a book covering that topic, adding it to the custom textbook, and updating the remaining topics.

The exercise revolves around designing an optimal custom textbook that covers all desired topics with the fewest pages. The starting point is the list of desired topics, and the endpoint is a custom textbook with no remaining topics.

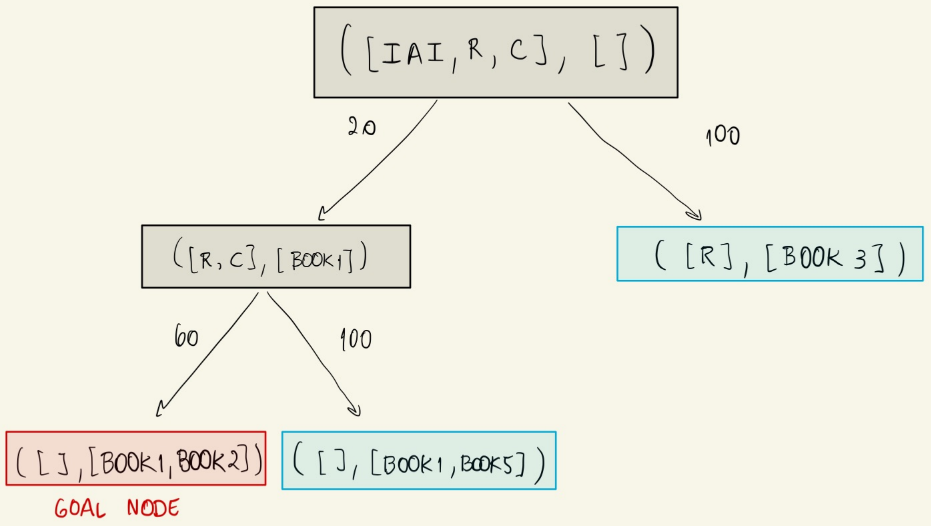
In particular the question **a)** is more focused on actually design and draw the tree search space, also specifying the goal node and the frontier state when the goal will be found. In the second question **b)**, instead, an admissible non-trivial heuristic function h is given.

Question a)

→ INITIAL STATE : (TOPICS , [])

= ([INTRODUCTION TO AI, REGRESSION, CLASSIFICATION] , [])

→ SELECT LEFTMOST TOPIC



LOWEST - COST - FIRST :

SOLUTION : ([], [BOOK1, BOOK2]) → GOAL NODE

TOTAL COST : 20 + 60 = 80

NODES AT THE FRONTIER :

([R], [BOOK 3])

([], [BOOK1, BOOK5])

Question b)

An heuristic function $h(n)$ can be considered *admissible* when it never overestimates the real cost to get to the goal state ($h(n) \leq h*(n)$). In this case, an admissible heuristic never overestimate the number of pages needed to cover the remaining topics. Moreover, the function should be *non-trivial*; $h(n) = 0$ for all n is considered trivial in this case.

A potential non-trivial admissible heuristics function can be based on pages-per-topic rate of a book, which covers one of the remaining topics. Taking as an example the node ([Regression, Classification], [Book 1]).

- Book 1 has already been selected, so it will not be considered anymore.
- Book 2 covers 2 topics in 60 pages, and both topics it covers are useful for us. So the average is 30 pages per topic.
- Book 3 covers 3 topics in 100 pages with one of the topics being useful for us, making the average 33.3 pages per topic.
- Book 4 covers 2 topics in 80 pages, but non of the topics are useful for us, so we will not consider it.
- Book 5 covers 2 topics in 100 pages, and both topics is covers are useful for us, making the average 50 pages per topic.

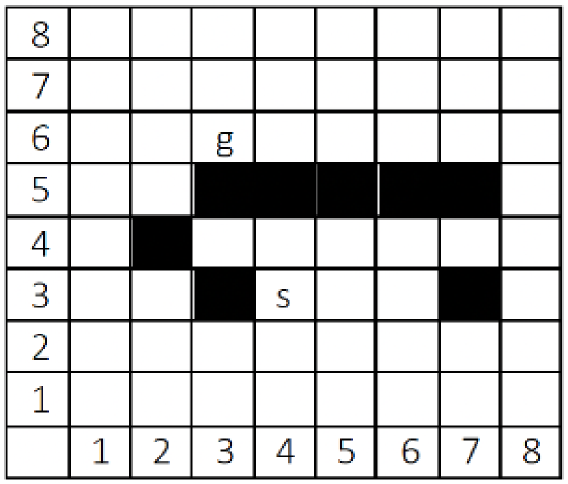
In general, given a textbook the heuristic function will be:

$$h(n) = \text{num\_pages} * \text{topics\_covered}.$$

In this scenario, among the available books, Book 1 gives us the lowest average pages per topic covered ($h(n) = 20/1 = 20$). Therefore the heuristic estimates that it will take us 30 pages to cover the 2 remaining topics, assuming we can find books as efficient as Book 2 to cover the rest of the topics.

Problem 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.

Question a)

In this application of the A* algorithm to a grid environment, the heuristic function ($h(x)$) will be the Manhattan distance between 2 cell. That is, given the coordinates of the starting i-th cell ($x_i y_i$) and destination j-th one ($x_j y_j$), their distance in th 2-dimension will be:

$$\text{Distance}(x_i y_i, x_j y_j) = |x_i - x_j| + |y_i - y_j|$$

Assuming that in the case of tie the algorithm prefers the path stored first, the following are the paths stored and selected in the first five iterations of the A* algorithm. Moreover, for each path, we report the already computed value of $f(n) = g(n) + h(n)$.

Iteration 1

In the first iteration, starting from the initial cell $s = 43$, we will expand the first 3 possible paths that we can take:

$$[< 43, 44 >_3, < 43, 42 >_6, < 43, 53 >_6].$$

Iteration 2

The path $< 43, 44 >_3$ has the minimum cost, so we selected it for this iteration and we expanded its relative possible paths:

$$[< 43, 44, 34 >_4, < 43, 44, 54 >_6, < 43, 42 >_6, < 43, 53 >_6].$$

Iteration 3

We would consider $< 43, 44, 34 >_4$ since it the minimum cost path, but it doesn't have any feasible neighbor to expand to:

- the cell 44 is already taken along that path
- the cells 33, 24 and 35 are forbidden areas.

For this reason, $< 43, 44, 34 >_4$ will be deleted. The remaining paths all have the same cost, thus we took the path $< 43, 42 >_6$ since it was the first stored among the 3.

$$[< 43, 42, 32 >_6, < 43, 42, 41 >_8, < 43, 42, 52 >_8, < 43, 44, 54 >_6, < 43, 53 >_6].$$

Iteration 4

We expanded $< 43, 53 >_6$ in the tie situation with minimum value $f(x) = 6$, since it's the earliest saved.

$$[< 43, 42, 32 >_6, < 43, 42, 41 >_8, < 43, 42, 52 >_8, < 43, 44, 54 >_6, < 43, 53, 54 >_6, \\ < 43, 53, 52 >_6, < 43, 53, 63 >_8].$$

Iteration 5

We have an identical tie situation, with the minimum value $f(x) = 6$. In this the earliest path stored by the A* algorithm was $< 43, 44, 54 >_6$, so we selected and expanded its related possible paths:

$$[< 43, 42, 32 >_6, < 43, 42, 41 >_8, < 43, 42, 52 >_8, < 43, 53, 54 >_6, < 43, 53, 52 >_6, < 43, 53, 63 >_8, \\ < 43, 44, 54, 53 >_8, < 43, 44, 54, 64 >_8].$$

Question b)

In this question, using the software in this link, we solve the presented grid problem using 2 different algorithm: **A* Search**, **Breadth-First Search (BFS)**, **Best-First Search**. For each of them It's described how they reached the solution, and discussed their efficiency in terms of the obtained numerical results and the situation/scenario.

1. **A Search***:

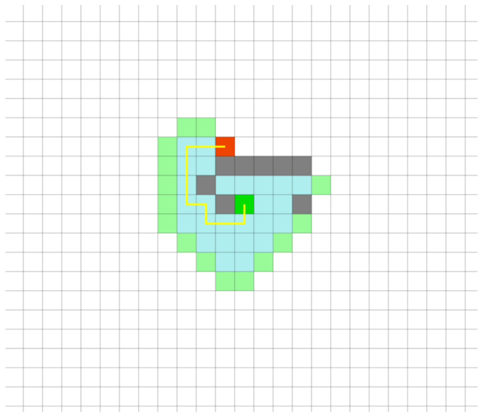


Figure 1: A* Search.

- **Path Length**: 10
- **Operations**: 71

A* evaluates the grid cell by combining the cost to reach the goal with the estimated cost from the starting cell to the goal. In other words, as mentioned in the previous question of the problem, the cost function is defined as follows:

$$f(n) = g(n) + h(n).$$

This ensures that the algorithm doesn't diverge too much from the goal. However, It also doesn't directly go to a more direct path if there is a better alternative. Based on the context of the problem and the number of iteration to reach the goal A* was quite efficient, while maintaining a good balance between exploration and a goal-oriented behavior.

2. **Breadth-First Search (BFS):**

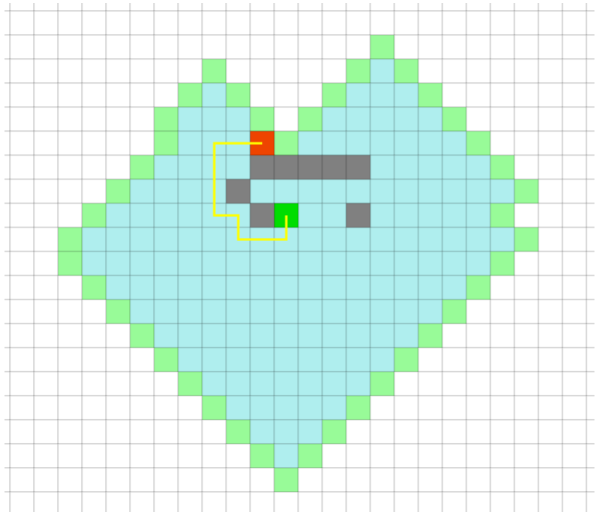


Figure 2: Breadth First Search.

- **Path Length**: 10
- **Operations**: 364

Compared to the other 2 algorithm, BFS is the only one that is an uninformative search. This means an heuristic function will not be used in order to reach the goal. In the case of the Breadth-First Search, It will explore all the neighboring grid cells at the present depth before moving on to other ones at the next depth level; this will ensure that every possible path is checked.

Even finding a path of the same length BFS took significantly more operations, compered to the other 2 informed (heuristic) search algorithms. Considering the context of a grid, this is mainly for a too much focus on the exploring (checking all possible paths), instead of being guided by a heuristic towards the goal.

3. **Best-First Search**:

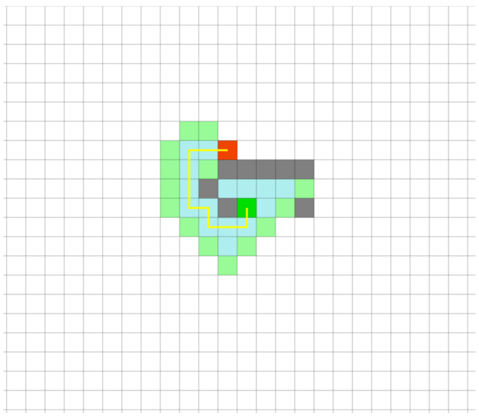


Figure 3: Best First Search.

- **Path Length**: 10
- **Operations**: 48

Best-First search (or greedy search) is an informed search algorithm that expands the node (consider the cell, in the problem context) that seems nearest to the goal, based only on a heuristic. In other words, the cost function in this case is only composed by the heuristic function:

$$f(x) = h(x).$$

This will make the algorithm mainly goal-oriented, with considering the actual distance traveled so far. Especially for this reason, in the scenario of the problem Best-First search was the most most operationally efficient in terms of iterations. However, a possible issue that can arise with the approach is that focusing only on the goal can sometimes lead sometimes to downfalls, especially in more complex environment with misleading paths.

Question c)

Using the same tool of the previous question, we now describe a particular situation where the BFS algorithm performs better than best-first search. The main idea is to take advantage of the main issue of the informed algorithm: its greediness. The best-first search focuses only on the most optimistic path (in this case, the shortest one to the goal); so, if we construct an environment with 2 possible paths to the goal, with one of those leading to a dead end, the best-first search will have to backtrack in order to choose the option, with a resulting longer path. The BFS algorithm, on the other hand, will always check for all possibilities systematically; in this way It will be less prone to misleading paths.

To further prove this difference, an example will be shown. The 2 algorithm will be used in the same environment, with few path towards the goal that bring to a dead end.

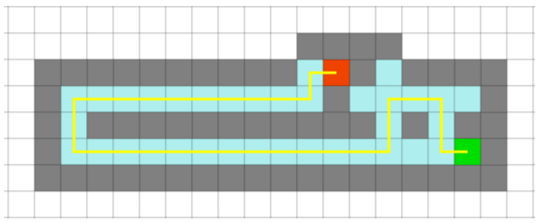


Figure 4: Best First Search on the environment with misleading path. Path length: 32.

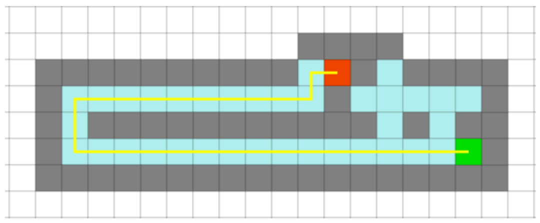


Figure 5: BFS on the environment with misleading path. Path length: 28.