MAvis Programming assignment: Uninformed Search

PART 1.2 - GRAPH-SEARCH, BFS AND DFS FRONTIERS

Latest Revision: Francesco Nespoli, Lorenzo Belgrano, Magnus Berg Sletfjerding & Sebastian Ray Mason

Previous contributors: Thomas Bolander, Mikkel Birkegaard Andersen, Andreas Garnæs, Lasse Dissing Hansen, Martin Holm Jensen, Mathias Kaas-Olsen

Group assignment for Elective Course: Artificial Intelligence.

Department of Computer Science
DIS Copenhagen

[Revised January 14, 2025]

Glossary

Below we provide a glossary of symbol/terms that you should follow throughout the assignments.

Symbol/term	Definition
`A`	Number of agents actively involved in the search problem or level.
`C`	Number of cells making up the map or environment in the search problem.
`B`	Number of boxes present in the search problem or level.
$\sum_{i=1}^{N}$ `<expression>`	The symbol (Σ) represents the sum. The expression directly below (Σ), ($i = 1$), is the lower limit (or lower bound) of the summation, indicating the starting index for (i). The expression directly above (Σ), (N), is the upper limit (or upper bound) of the summation, indicating the final index for (i) in the sum. The `<expression>` is the mathematical expression involving (i) that will be evaluated and summed from ($i = 1$) to ($i = N$). In this case, it is (N), indicating that this is an finite series you sum over. Summing over the integers up until first positive integer N: $$\sum_{i=1}^{N} i = 1 + 2 + ... + N$$
$\prod_{i=1}^{N}$ `<expression>`	The symbol Π represents the product. The term denotes a product where you multiply the values of `<expression>` for each integer value of i from the lower limit of 1 to the upper limit of N. An example of this product might be the product of the first N positive integers: $$\prod_{i=1}^{N} i = 1 \times 2 \times \ldots \times N$$

Exercise 1: implementing `Graph-Search`

In this exercise we consider multi-agent pathfinding problems in the `MAvis` hospital environment. Such an environment is thoroughly described in the `hospital_domain.pdf` file. You will be implementing the `Graph-Search` algorithm presented during the lectures yourself in order to be able to solve levels using different search strategies. The client already contains an implementation of the frontier for BFS via the `FrontierBFS class`. To complete this exercise you only need to modify the `graph_search.py` file.

(a) Implement the `Graph-Search` algorithm yourself in `graph_search.py`. Remember that by default the python client will use the BFS frontier implementation, which is already given

(b) Test out your `Graph-Search` implementation on the `SimpleDebug.lvl` and `TwoAgentsDebug.lvl` levels. Your goal should be to simply see the client correctly finding a solution

(c) Investigate in detail the way that BFS searches through `SimpleDebug.lvl` and `TwoAgentsDebug.lvl`: make use of `print` statements to check the content of the frontier at each iteration and enrich the output of your `Graph-Search` by invoking the `print_search_status` method to get additional info on the search status

(d) Craft a new level implementing the `HUMANVSMACHINE` level you encountered previously in Exercise 3 of the "Exercises 1.1: Search Problems" sheet

(e) In Exercise 1 of the "Exercises 1.2: Uninformed Search" sheet you have gone through a step-by-step execution of `Graph-Search` BFS on the `HUMANVSMACHINE` problem. Now run your implementation of `Graph-Search` using BFS on this problem:

  (1) Is the solution found by the algorithm the same you found by running the algorithm by hand?

(2) Is the total number of states generated the same you found by running BFS by hand?

(3) Compare, at each iteration, the states in the frontier by running BFS by hand and by running your implementation of `Graph-Search`: is there any difference?

Exercise 2: implementing the DFS frontier

In this exercise you will implement the DFS frontier and use it in your previously implemented `Graph-Search` algorithm.

(a) Using the already implemented BFS Frontier in `bfs.py` as inspiration, implement the DFS frontier

(b) Run `Graph-Search` using the DFS frontier on `SimpleDebug.lvl` and `TwoAgentsDebug.lvl`. Your goal should be to simply see the client correctly finding a solution

(c) Investigate in detail the way that DFS searches through `SimpleDebug.lvl` and `TwoAgentsDebug.lvl`: make use of `print` statements to check the content of the frontier at each iteration and check the status of the search through the output of the `print_search_status` method

(d) Consider the `ExampleMaze` level shown in Figure 2.1:

  (1) Craft a new level implementing the `ExampleMaze` level

  (2) During class you were shown a step-by-step execution of `Graph-Search` BFS on the `ExampleMaze` problem. Now run your implementation of `Graph-Search` using BFS: does the search behave as you expect (in terms of solution found, total number of states generated, etc.)?

  (3) Redo the previous point using DFS as a search strategy instead

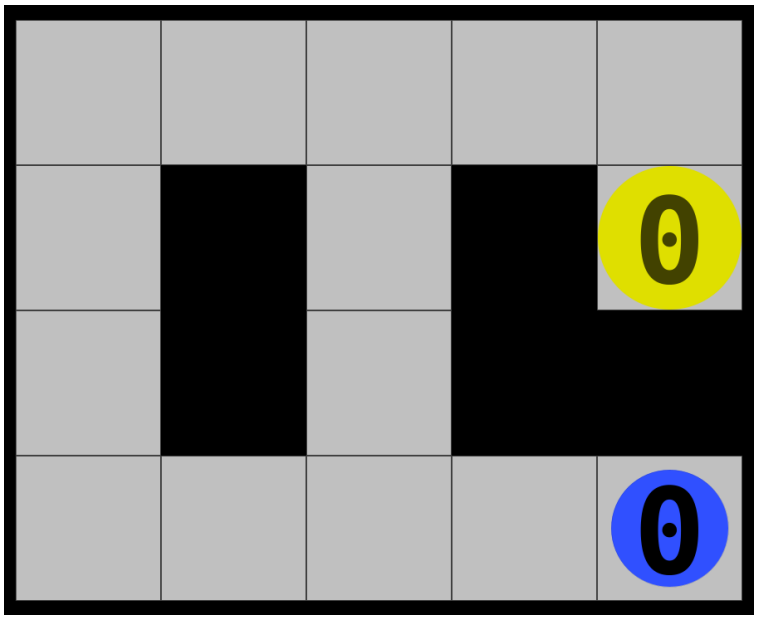

Figure 2.1: `ExampleMaze` level in the hospital domain