Artificial Intelligence

Tutorial week 3 – Search

COMP9414

1 DFS and IDDFS

DFS (Depth-First Search) is a graph traversal algorithm where you explore as far as possible along each branch before backtracking. It uses a stack data structure to keep track of the visited nodes and the next nodes to visit.

IDDFS (Iterative Deepening Depth-First Search) is a variant of DFS where you perform DFS with increasing depth limits until the goal is found. This ensures the benefits of both depth-first and breadth-first search (BFS), with the memory efficiency of DFS and the completeness of BFS.

In this tutorial, you will develop Depth First Search (DFS) and Iterative Deepening Depth First Search (IDDFS) for a gridworld problem.

2 Setup

(a) The first step consists of downloading and uncompressing the provided code. This tutorial will use a graphical interface therefore you should use Anaconda (it will not run in Colab). Alternatively, you might use a CSE Linux workstation with VLAB.

(b) Be sure that both matplotlib and tkinter are installed as these packages are used by the gridworld. You can check if tkinter is properly installed by running `python -m tkinter` in your terminal.

(c) Open the gridworld using `python main.py`. You should see a screen as shown in Figure 1.

(d) The gridworld has already implemented some agents such as random, astar, greedy, q-learning, among others. You will also see DFS and IDDFS in the menu, however, these agents are not yet implemented.

(e) Using the mouse, you will be able to create more complex gridworlds including walls and goal states.

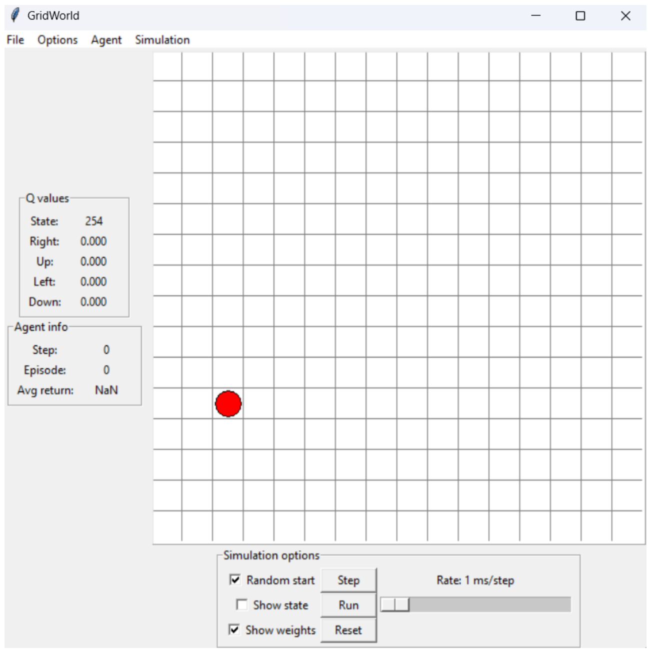

Figure 1: Main screen of the gridworld.

3 Experiments

(a) Familiarise yourself with the search agent and its output by running different instances, showing the states, and using random and fixed start positions.

(b) The gridworld has two newly added files: dfs.py and DFSAgent.py. Next, we will complete these two files by implementing both DFS and IDDFS agents filling TODO sections.

(c) The dfs.py file contains a function called find_path. This function takes six arguments as follows:

- **neighbour_fn:** is the successor function that returns a set of all the neighbouring nodes.
- **start:** is the start node.
- **goal:** is the goal node.
- **visited:** is the list of nodes that are already visited, explored along a path – cycle checking.
- **reachable:** is a function that takes a tile as an argument and returns false if there is a wall on top of the tile or if the tile is outside of the grid map.

- **depth:** is the depth that your function should avoid searching at any deeper depth

Implement the function that returns a path between two nodes as a list of nodes using depth-first search. If no path can be found, an empty list is returned. Modify only the commented section of the dfs.py file indicating TODO. Then DFSAgent class within the DFSAgent.py file utilises your function to create a DFS agent. Therefore, you can test it in the gridworld.

(d) Complete idDfsAgent class to create an IDDFS agent by implementing the start_agent method using IDDFS to find the path between self.start and self.goal nodes. Assign the path to the self.path variable. If no path can be found, an empty list is returned.