CS 170: Introduction to AI
Project 1 – Search Algorithms

Assigned: April 9, 2025
Due: April, 20, 2025

Overview

In this assignment, you will implement a series of search algorithms to navigate an agent through a 2D grid-based environment. The objective is to explore different strategies for path-finding using the following algorithms that we covered in class:

- Breadth-First Search (BFS)
- Depth-First Search (DFS)
- Uniform Cost Search (UCS)
- A* Search

The grid environment allows interactive visualization of algorithm behavior, aiding in comprehension and debugging.

Environment Description

The state space is a two-dimensional grid, where the agent must find a path from the start cell (marked **S**) to the goal cell (marked **G**). Obstacles and high-cost regions are embedded in the grid. Gray blocks represent obstacles (non-traversable), and the colored blocks indicate high-cost regions. The grid cells are 4-connected, meaning that the agent is only allowed to take NSEW actions. In all given pathfinding problems, the cost of moving from one cell s to an adjacent one s' when taking an action a is exressed a function of the landing cell only s', with cells in the lightly colored regions having a higher cost than the white cells.

Code Requirements

After downloading the code, you should be able to display the grid world and generate different maps through the GUI. Run the code with:

```
python search.py
```

You are required to modify `search.py` to implement BFS, DFS, UCS, and A* algorithms. Each algorithm should be implemented as a graph search strategy, where no state is expanded more than once following the pseudocodes covered in class. You should test all four algorithms with the default map and two random maps.

Breadth-First Search (10 points)

Modify the `breadth_first_search` function. Implement BFS by changing the open list to a FIFO queue.

Depth-First Search (10 points)

Modify the `depth_first_search` function. Implement DFS using a stack for the open list.

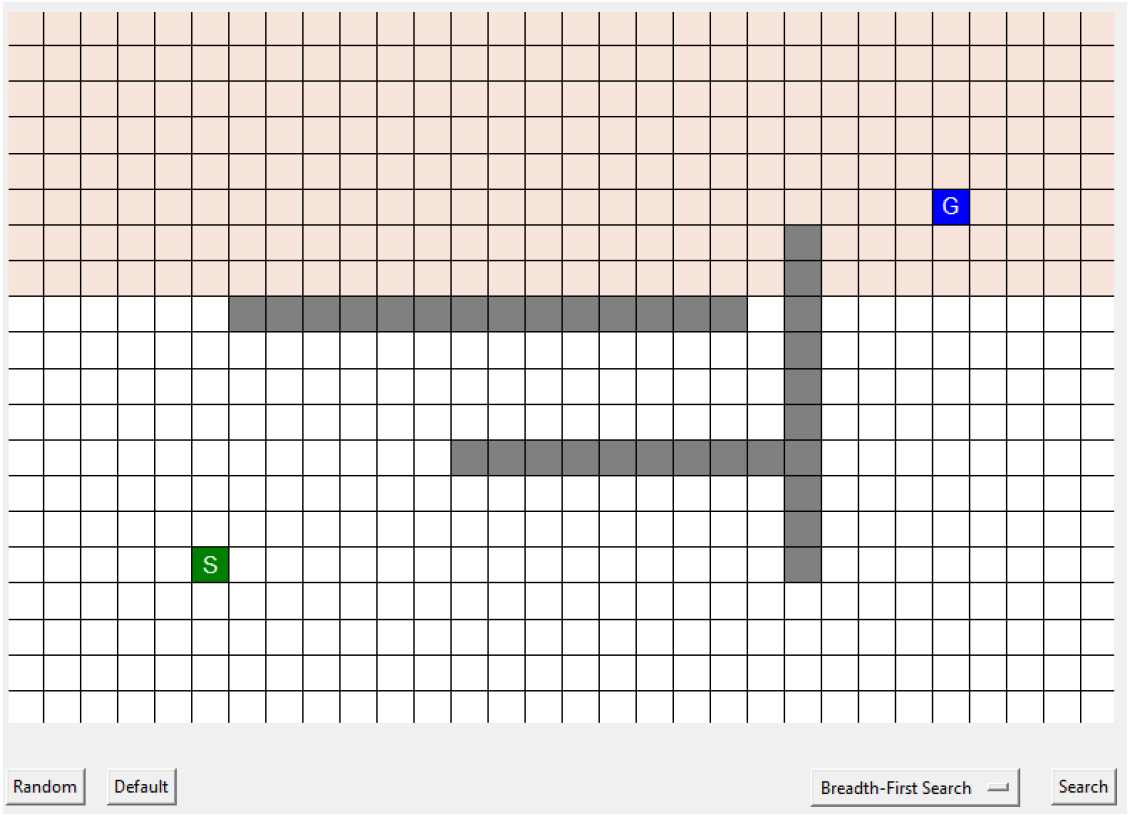

Figure 1: Grid world environment: Start (S), Goal (G), gray cells are obstacles, light colored cells represent high-cost regions.

Uniform Cost Search (25 points)

Modify the `uniform_cost_search` function. UCS should expand the lowest cumulative cost node next. Implement it with a priority queue.

A* Search (25 points)

Modify `astar_search`. A* combines UCS with a heuristic (Manhattan distance). Implement this heuristic in the `heuristic` function. A* should prioritize nodes based on their path cost (g-value) and their heuristic value.

Important Notes

- **Please read carefully all the comments in the `search.py` file**. It's unlikely that you can start implementing the different algorithms without getting a better idea of how the code works and what is expected from each function.
- **You must use the provided data structures (Stack, Queue, PriorityQueue, Set).**
- **While implementing all algorithms, refer to the pseudocodes provided in class.** Do not refer to external pseudocodes.
- In the visualization, the open list is white; closed list (expanded nodes) is red.

Report (30 points)

Along with your code, you need to submit a written PDF format that includes the following components:

1. **Results (20 points)**:
   Include screenshots for each of the four algorithms on the default map and two randomly generated maps using the GUI. Along with the screenshots, report the total path cost (shown in the terminal) and your observations.

2. **A* on Uniform Cost Grid (5 points)**
   Compare the performance of the A* path on the default map to the one that you would have obtained if all states are equally costly (uniform cost grid). Will A produce the same path, or are there differences? Briefly explain your findings. If the paths are different, can you sketch an optimal path with justification?

3. **BFS vs. UCS on Uniform Cost Grid (5 points)**
   Assuming a uniform cost grid, will the BFS generated path on the default map have the same cost to the one obtained by Uniform Cost search? Support your explanation with reasoning.

Submission

Please submit your report and code via `Gradescope`. If you are a team, please include the names of both members in your report.

Help

If you get stuck, do not hesitate to contact us for help and stop by during the discussion and office hours. We also encourage you to post questions and initiate discussions on `Slack`.