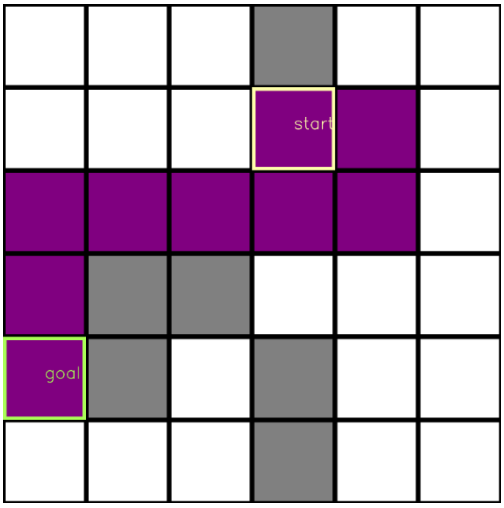
AERO 489/689 Foundations of Aerospace Autonomy

Fall 2023

Programming Assignment 1: Autonomous rover search

Context: Consider an autonomous rover in a scientific mission on a planetary surface such as Mars or Europa. Imagine that the rover has a 2D grid describing the map of its surroundings. Some cells in the grid are of special scientific interest. The goal of the agent is to autonomously find a path from its current location to one of those scientifically interesting areas. Some of the cells have rough terrain (e.g., high slope, large rocks, slippery conditions). These cells can be considered "obstacles" to be avoided at all costs. In this assignment, you will develop and benchmark some search algorithms for this rover search problem.

Along with this document, you have access to a script called gridworld.py on Canvas. This script contains the "main" function, which creates a grid world (predefined or random), defines a sequence of moves as a list of actions, and plots it. Your job is to update this script to substitute the predefined sequence with the output of a search algorithm. An example grid world and solution are shown below.



The cost of moving one cell in any direction is 1. In addition to finding a solution, your goal is to find the optimal solution that minimizes the total path cost.

Algorithms: Implement breadth-first search, depth-first search, and A* search. Grad students also implement iterative deepening and greedy best first.

Start with the basic implementation as shown in class and if you need to deviate from it for some reason, explain what the issue is and how you addressed it.

Specific questions:

1. **Data structures:** Create data structures that support the operations listed in the pseudocode discussed in class. You are encouraged to use Python classes to implement your data structures and the rest of your solution.

2. **Small grids**: Provide the solution and path cost for each algorithm for Grid 1, Grid 2, and a valid random grid of 10 x 10 with 20 walls to show that your algorithm is extensible. Compare the

answers and comment. Note: The gridworld.py includes plot_grid and plot_path utilities to help you show that your solution is correct.

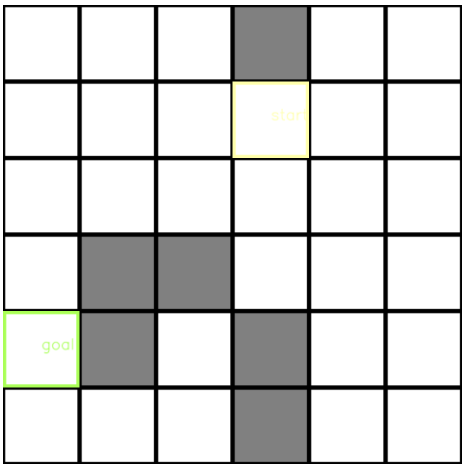
3. **Runtime for large grids**: Create 100 random 100x100 grids with 2000 walls. Run each algorithm for each of the 100 scenarios and compute the runtime, solution cost and number of nodes generated for each. Create a plot with one boxplot per algorithm comparing the runtime and number of nodes generated of the algorithm. Report the averages of the runtime, solution cost, and number of notes generated. Comment on the results.

Please submit your answers as a PDF showing your solutions and plots and submit your code as a Python file.

Notes:

- You will need to "pip install opencv-python" and "pip install matplotlib" to run the sample script. You may need to run "pip install upgrade –numpy" if you have trouble with opencv-python.
- The script generates the grid world as both a numpy array, a dict, and a JSON file. You may use any data structure you please as long as you do not modify the layout of the grid itself. The script is provided primarily for visualization, you may not want to use the data structures provided.
- You will want to create functions to replicate functions seen in the pseudocode, like a transition model function, etc.
- Obstacles in the gridworld are denoted with numpy "nan" values. The start is denoted with a 5 and the goal state is denoted with a 1.
- The location [0,0] is in the top left corner.

Grid 1:



Grid 2: