University of Alberta
Computing Science Department
CMPUT 366 - Fall 2021

Assignment 1
Due date: September 24
12 marks

Intelligent Systems (CMPUT 366)

Submission Instructions

Submit on eClass your code as a zip file and the answers to the questions of the assignment as a pdf. The pdf must be submitted as a separate file so we can more easily visualize it on eClass for marking.

Overview

In this assignment you will implement Dijkstra's algorithm and A* for solving pathfinding problems on video game maps. We will consider a grid environment where each action in the four cardinal direction (north, south, east, and west) has the cost of 1.0 and each action in one of the four diagonal directions has the cost of 1.5. Each search problem is defined by a video game map, a start location and a goal location. The assignment package available on eClass includes a large number of maps from `movingai.com`, but you will use a single map in our experiments; feel free to explore other maps if you want.

Most of the code you need is already implemented in the assignment package. Please spend some time reading the code before starting to implement what is asked. The starter code includes: a function for reading the maps and creating state objects, a transition function (given a state it returns the set of neighbors in the state space), and a function for plotting different results about the algorithms you will implement. The package also includes a set of test cases with the code needed to verify whether your implementation is correct (see `main.py` for details).

How to Run Starter Code

Follow the steps below to run the starter code (instructions are for Mac and Linux).

- Install Python 3.
- It is usually a good idea to create a virtual environment to install the libraries needed for the assignment. The virtual environment step is optional.
  - `virtualenv -p python3 venv`
  - `source venv/bin/activate`
  - When you are done working with the virtual environment you can deactivate it by typing `deactivate`.
- Run `pip install -r requirements.txt` to install the libraries specified in requirements.txt.

You are now ready to run the starter code by typing: `python3 main.py --testinstances`. Copy and paste might not work properly because `--testinstances` could be pasted as `-testinstances`.

If everything goes as expected, you should see several messages as shown below. These messages are running a set of test cases on your code. Naturally, if you haven't implemented the search algorithms, then all test cases will return with a "mismatch." You will not see any of these mismatch messages once you have correctly implemented what is being asked. If you're curious to see how the test cases are implemented, see `main.py`.

```
There is a mismatch in the solution cost found by Dijkstra
and what was expected for the problem:

Start state:  [108, 26]
Goal state:  [105, 67]
Solution cost encountered:  -1
Solution cost expected:  42.5

There is a mismatch in the solution cost found by A*
and what was expected for the problem:

Start state:  [108, 26]
Goal state:  [105, 67]
Solution cost encountered:  -1
Solution cost expected:  42.5
```

Implement Dijkstra's Algorithm (3 Marks)

Implement Dijkstra's algorithm in method `search` of the class `Dijkstra` in file `algorithms.py`. The implementation must be correct, i.e., it must find an optimal solution for the search problems, if they have a solution. The implementation also must be efficient, i.e., it should use the correct data structures to implement the OPEN and CLOSED lists. You can test the correctness of your implementation of Dijkstra's algorithm by running `python3 main.py --testinstances`.

Implement A* Algorithm (3 Marks)

Implement A* algorithm in method `search` of the class `AStar` in file `algorithms.py`. You will need a heuristic function to guide the A* search. In this assignment we will use the Octile distance, which is a version of the Manhattan distance function we have seen in class that accounts for diagonal moves. Let Δx and Δy be the differences in distance in the x-axis and in the y-axis, respectively, between the evaluated state s and the goal state. The Octile distance between s and the goal is

$$h(s) = \max(\Delta x, \Delta y) + 0.5 \times \min(\Delta x, \Delta y)\,. \tag{1}$$

This heuristic function must be implemented in method `h_value` of class `AStar`. The Octile distance is consistent and admissible. Since the heuristic is consistent, you do not have to implement the re-expansion of nodes we discussed in class. The A* implementation must be correct, i.e., it must find an optimal solution for the search problems, if they have a solution. The implementation must be efficient, i.e., it should use the correct data structures to implement the OPEN and CLOSED lists. You can test the correctness of your implementation by running `python3 main.py --testinstances`.

Analyzing the Scatter Plots

Once you have implemented both Dijkstra's algorithm and A*, run the code with the "plots" option enabled: `python3 main.py --testinstances --plots`. Your program will generate three scatter plots: `nodes_expanded.png`, `running_time.png`, and `solution_cost.png`. Each point in the scatter plot represents a search problem and one of the axis represents Dijkstra's algorithm and the other A*.

Explain the Results (3 Marks)

Explain the three plots that were generated. You should explain the relation between the running time with number of nodes expanded of each algorithm: are the plots identical or only similar? Why? You must also explain the results shown in the scatter plot for the solution cost. Please include the plots in your answer.

Multiplicative Factor (3 Marks)

Next, you will change your implementation of the heuristic function used with A*. Instead of returning the heuristic value $h(s)$ for a state s as described in Equation 1, the function will return $2 \times h(s)$—we are inflating the h-value by 2. Generate all three plots and add them to your answer: running time, nodes expanded, and solution cost. Similarly to what was done in the previous question, explain the results you obtained by relating all three plots with the inflated heuristic function with the three plots from the previous question. What are the differences between this set of plots and the previous set of plots? Explain your answer.