CP8326: Assignment 1

Due: October 3, 2024 at 9pm

Late Policy: Penalty for submitting the assignment late is 10% per day. However, if there are extenuating circumstances, please reach out and we will try to arrange for reasonable accommodations.
Total Marks: This assignment is worth a total of 55 marks, which represents 13% of the course grade.
Handing in this Assignment: You are only required to submit a report containing your answers to the various questions. This should be submitted as a PDF file on the D2L site. You do NOT need to submit your code. You can submit a new version of any file at any time, though the lateness penalty applies if you submit after the deadline. For the purposes of determining the lateness penalty, the submission time is considered to be the time of your latest submission.
Clarifications and Questions: Please use the discussion forum on the D2L site to ask questions as they come up. These will be monitored regularly. Clarifications will be made there, as needed.
Collaboration Policy: Students are encouraged to discuss the assignments amongst themselves. However, the reports must be done individually.

Introduction

The goal of this assignment is to give you experience running and experimenting with heuristic search code, as well as theoretically analyzing heuristic search algorithms. As such, you have been given a framework for running such experiments.

The questions below include both experimental and theoretical components. For the theoretical questions, feel free to use any of the theorems or lemmas we have already discussed in class.

Finally, note that different authors sometimes define the count of the number of node expansions slightly differently. For this assignment, we will count the number of node expansions as the number of times a node is taken off the open list, or (essentially) equivalently, the number of times the set of applicable actions is generated (see `num_get_action_calls` in the CSV output).

Code Information

Code for the assignment has been provided in an attached zip file. Documentation for the code can be found at `https://www.cs.torontomu.ca/~rick.valenzano/hsef_framework/index.html`. This will include information on how to compile and run the code, information about the architecture, full documentation of the code itself, and other tips and tricks.

1 A* Data Structures [10 marks]

For this question, you will be looking at the provided code to better understand how A* can be implemented.

(a) [**4 marks**] Consider the file `apps/grid_pathfinding_app.cpp` which shows how A* can be run to solve multiple grid pathfinding problems on a given map. Notice that the search engine is given a hash function, namely a `GridLocationHashFunction`. Find where this class is declared. You will notice that these object's have a function called `getHashValue` which generates an unsigned 32-bit integer for any given state (*ie.* a `GridLocation`). In 3-5 sentences, describe the calculation this method is performing. Make sure to describe why any two different map locations are guaranteed to be assigned assigned different hash values.

(b) [**2 marks**] The A* algorithm is implemented using a `BestFirstSearch`, which is defined in `engines/best_first_search/best_first_search.h`. You will see that this class has a list of nodes called `m_nodes`, which is a vector of nodes seen thus far in the search. It also has an open list called `m_open_list`. Find the class `m_open_list` belongs to. You will see that it stores a list of IDs for the nodes in `m_nodes`, ordered according to their evaluation. What data structure is used to maintain the ordering of the node ids? What is the runtime complexity of adding and removing items from such a structure?

(c) [**4 marks**] In addition to `m_nodes` and `m_open_list`, `BestFirstSearch` has a `NodeMap` called `m_node_map`. Note that`NodeMap` is actually an alias for a hash map (an `unordered_map` whose key is a hash value and whose value is an integer). For this question, look at lines 260 to 293, which shows the code for checking if a generated state is already in the open or closed list and then handles it accordingly. You should also take a look at the `getNodeID` method. In 3-6 sentences, describe how the hash function, `m_nodes`, `m_node_map`, and `m_open_list` is being used to detect if the generated child state is in the open list, closed list, or newly generated. HINT: `getNodeID` returns an `optional NodeID` value, meaning it sometimes returns a `NodeID`, and sometimes it returns `null`.

2 Tie-Breaking in Real Map [15 marks]

Recall that in class, we identified that tiebreaking can have a significant practical impact on the performance of A* on an artificial problem. In this question, you will test to see if this holds in a more practical setting. For this, you should consider the file `apps/grid_pathfinding_scenario_app.cpp`. This file shows the code for running a suite of experiments on a set of maps from a standard benchmark set. Note, that the experiments are run on 8-connected problems, meaning that the actions include `Northeast`, `Southeast`, `SouthWest`, and `NorthWest` in addition to the standard 4 directions.

You will now choose a map from `https://movingai.com/benchmarks/grids.html` to run your experiments on. Choose any map from either the "Commercial Game Benchmarks" or "Real-World Benchmarks" sets (do not pick an artificial benchmark). Get the corresponding scenario file for your chosen map. Now compare Hi-G, Low-G, and Default (ie. no) tie-breaking on the problems listed in the scenario file for the given map.

Please note the following:

- You should be able to run such experiments with only minor modifications to `apps/grid_pathfinding_scenario_app.cpp` provided you put the `.map` and `.scen` files in the correct locations.
- Once you generate the results in CSV format, you do not need to analyze the results in C++. You may load the results in whatever data analysis software you choose (ie. Excel, Google Sheets, Pandas, etc.).
- In your report, make sure to list the name of the map you tested on. You should also include the image of the map given on the benchmark website.
- Some of the scenario files contain a LOT of problems. You do not need to run on all of them, as it may take a while (especially in debug mode). Try to run on at least 500 problems. You may do so by simply deleting problems from the scenario files. However, make sure you run all three tiebreaking policies on the same set of problems.
- For each of Low-G, Hi-G, and Default, list the mean and median expansions over your test problems.
- For each of the 500+ test problems, determine which of Low-G, Hi-G, and Default solved the problem using the fewest expansions. You do not need to include the per-problem results in the report, but show what percentage of problems each method was fastest at.
- Briefly describe the conclusions of this experiment. Do they verify what we saw when testing on an empty grid in class?

3 Problem Difficulty [20 marks]

In this question, you will be investigating how different problem features can make a search problem more difficult than another when using A*. In particular, consider the following three features:

- The optimal solution cost of the problem (C^*).
- The heuristic value of the initial state ($h(s_I)$).
- The error of the initial state ($C^* - h(s_I)$).

(a) [**5 marks**] Before running any experiments, provide a hypothesis on how you think each of these features might be related to runtime (*ie.* number of expansions). Do you think all of them will impact the runtime? Which do you think has the greatest effect? What type of cross-correlation between any of these features would you expect?

(b) [**10 marks**] Create three scatter plots, one for each of the features considered above, which shows the relationship between these features and runtime when using A* and the octile heuristic on the same map you used in the previous question. In addition, calculate the correlation between each of these features and the runtime. In 5-10 sentences, discuss what these results say about the impact that these features have on problem difficulty, and how they correspond to your hypotheses in part (a). HINT: In order to calculate the heuristic value of each start state, notice that each element of the vector of `GridPathfindingScenario` objects contains the start and goal state of the corresponding problem. In addition, see Tutorial 03 for information on how to calculate the heuristic value of a specific state. Don't forget to change the goal state of the evaluator for each new problem.

(c) [**5 marks**] In 5-10 sentences, discuss how you might address any cross-correlation. How could you design an experiment (or focus your data analysis) to try to mitigate the impact that any one of these features has on runtime, so that you could examine the others in isolation?

4 Theoretical Analysis [10 marks]

Let α and β be two instances of A* that are identical aside from their tiebreaking policy. Assume that they are both using the same consistent heuristic h on a given problem, and they both find a solution to that problem. Prove that if a node n corresponding to some state s is expanded by α before a goal is found, while β does not expand any node corresponding to s before the goal is found, then $g * (n) + h(n) = C^*$.