CP8326: Assignment 2

Due: October 17, 2024, 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 71 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 further your understanding of A*, WA*, and GBFS. Recall that you should 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).

1 Search Data Structures [6 marks]

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

(a) **[3 marks]** Consider the set of unique f-costs that that will be encountered during search. For example, if the search only generates nodes n_1, n_2, and n_3 with $f(n_1) = 1$, $f(n_2) = 1$, and $f(n_3) = 0$, then it would encounter 2 unique f-costs (1 and 0).
Now suppose you are trying to solve a unit-cost problem (*ie.* where all actions have a cost of 1) on a state-space such that the goal can be reached from any state along a path with a cost of at most 100. If your heuristic is consistent, what is the maximum number of unique f-costs that your search may encounter. Justify your answer in no more than 6 sentences. HINT: First identify what is the maximum f-cost of any node you encounter, then consider how you can use that number to find the maximum number of unique f-costs.

(b) **[3 marks]** Recall the data structure used to sort the open list in the HSEF code base. It was a good choice as a general solution that is effective for a variety of state-spaces because it has strong worst-case guarantees on the time needed to add and remove elements. However, suppose we were in the situation described in part (a) of this question. Describe in 6-10 sentences how you could define a more efficient data structure for maintaining the open list that takes advantage of this problem's properties. Justify why it would be better in this instance.

2 Non-Unit Cost Domains [30 marks]

For this component of the assignment, you will be experimenting with two variants of the sliding tile domain:

- The *unit-cost* variant where all actions cost 1
- The *inverse* variant where moving tile t has a cost of $1/t$

Just as in the regular sliding tile puzzle, we can create versions of the Manhattan distance for the inverse variant. For each tile, we simply count the distance that every tile is from its goal location, and multiply it by the cost of moving that tile (*ie.* the distance-to-go of every tile is weighted by the cost of moving that tile). The resulting heuristic is still admissible.

To run experiments using the sliding tile puzzle, you will have to run A* on different versions of the problem. You may choose to start with `apps/sliding_tile_app.cpp`, which contains code for running IDA* on 100 instances of this problem. Either create a new app, or modify it to run A*. Note that both the transitions and heuristic objects take in a parameter indicating which variant to use. This will allow you to quickly change between variants. In addition, recall from Assignment 1 that you need a hash function for best-first search. You will be able to find one in `environments/sliding_tile_puzzle/sliding_tile_puzzle_hash_function`. Technically, this is just an alias for a `PermutationHashFunction`, which is a hash function for any problem that consists of an ordering of the numbers from 0 to $n-1$.

You should now complete the following tasks.

(a) **[20 marks]** Run A* on the two sliding tile puzzle variants while using Hi-G tiebreaking. Save this data in a file (csv, txt, it is up to you), and perform the following analysis on it using whatever charting software you prefer (Excel, matplotlib, etc.). The goal of this analysis is to understand the relative difficulty of solving these problem variants.

- Create a table that compares the average and median number of node expansions of A* on these puzzle variants. Recall that A* will solve all problems for all variants optimally. Comment on the relative difficulty of the different problems.
- Create a scatter plot that compares the runtime of A* on each of the 100 problems for each variant. The x-coordinate should be the number of node expansions needed to solve the problem under the unit-cost variant, and the y-coordinate should be the number of node expansions needed to solve the problem under the inverse variant. For clarity, you may want to use a logarithmic scale for the axes. You should also include a line following y=x to help clarify when a problem is harder under one variant than the other. Figure 2 in this paper: `https://www.ijcai.org/proceedings/2021/0557.pdf` provides an example of such a chart. Note that keeping the axes symmetric also makes it easier to read.
- Perform a similar analysis as on Assignment 1, where you consider the correlation between C^*, $h(n_{init})$, and $C^* - h(n_{init})$ and the runtime (in terms of expansions). Comment on if the results are similar to what you saw on the pathfinding problems, and why or why not that may be.

- Similarly measure the correlation between the *length* of the solutions found and runtime. Note that this is unnecessary for the unit cost variant since C^* is also the plan length.
- Create a histogram that compares the distribution of the number of optimal plan lengths (NOT costs) between the two variants. Note that the objective here is to understand how the optimal plan lengths compare between the problems

(b) **[5 marks]** Comment on what these tables and charts suggest about these variants and the relative difficulty of using A* to solve them. Refer to the charts as needed to make your case. What do you think are the major factors driving the differences you see?

(c) **[10 marks]** How would you further investigate the hypotheses in the last question? In particular, how could you use the data already generated to evaluate this hypothesis? Describe your pproach, perform the analysis, and discuss the results. Identify any assumptions made, or how you could improve the analysis with additional experiments. You do not need to perform additional experiments beyond those performed in part (a), just discuss what additional information could help to provide further evidence of your claims.

3 Re-Expansions in WA*[25 marks]

You will now evaluate the impact of re-expansions in WA*. To do so, consider the following:

- To run WA* instead of A*, you need to use the weighted f-cost evaluation function instead of the standard f-cost one. You will find the former in `core/engines/engine_components/eval_functions/weighted_f_cost_evaluator.h`. The constructor takes in a weight as its parameter. After construction, you should be able to use this in a similar way as the A* evaluation function.

- To enable and disable re-openings, you need to change the value of the variable `m_use_reopened` of the `BestFirstSearchParams` that is passed to the engine.

Below, we refer to a WA* that does reopen nodes as rWA* and a WA* that does not reopen nodes as nrWA*. You can use whatever tie-breaking strategy you see fit with WA*, but document which you choose and be consistent across all experiments[1].

(a) [**10 marks**] Run rWA* and nrWA* each with weights 2, 5, and 10, on the 100 unit-cost 3x4-puzzle problems and create two tables — one for each of rWA* and nrWA* — that shows the average and median number of node expansions, and the average and median solution costs. In roughly 5 sentences, describe how the methods compare in terms of both number of node expansions and solution quality.

(b) [**10 marks**] Run rWA* and nrWA* each with weights 2, 5, and 10, on pathfinding problems. Select any scenario file as before, and run on at least 500 problems. Make 2 tables — one for each of rWA* and nrWA* — that shows the average and median number of node expansions, and the average and median solution costs. In no more than 5 sentences, describe how the methods compare in terms of both number of node expansions and solution quality.

(c) [**5 mark**] Contrast the relative performance of rWA* and nrWA* on the 3x4-puzzle vs the pathfinding problem. What trends are consistent between the two domains? What trends are different? What difference about the domains do you think accounts for the differences. Answer in 5-10 sentences.

[1] Tie-breaking has an effect on WA*, and you will explore the topic with respect to GBFS in the next question. In any case, being consistent in your selection is most important here. Feel free to explore this topic on your own if you are interested, but you do not have to submit any commentary on the topic.

4 GBFS and Tiebreaking [10 marks]

Recall that GBFS is a best-first search that uses the heuristic alone to define the priority of a node. Ties may still occur in this case. In this question, you will analyze the impact of different approaches to handling ties in GBFS.

(a) **[2 marks]** Before running any experiments, hypothesize what the relative impact of HI-G, Low-G, and Default tie-breaking will have on GBFS. Briefly discuss and justify your answer

(b) **[8 marks]** Design an experiment on one of the given problem sets to test your hypothesis. Use charts/tables/diagrams as you see necessary. Note, GBFS should be set to not reopen nodes, as doing so may complicate the results.