CS 486/686 Assignment 1 (123 marks)

Alice Gao

Due Date: 11:59 pm on Wednesday, October 6, 2021

Changes

- Q1b: Updated the two examples of the heuristic function.

Academic Integrity Statement

If your written submission on Learn does not include this academic integrity statement with your signature (typed name), we will deduct 5 marks from your final assignment mark.

I declare the following statements to be true:

- The work I submit here is entirely my own.
- I have not shared and will not share any of my code with anyone at any point.
- I have not posted and will not post my code on any public or private forum or website.
- I have not discussed and will not discuss the contents of this assessment with anyone at any point.
- I have not posted and will not post the contents of this assessment and its solutions on any public or private forum or website.
- I will not search for assessment solutions online.
- I am aware that misconduct related to assessments can result in significant penalties, possibly including failure in the course and suspension. This is covered in Policy 71: https://uwaterloo.ca/secretariat/policies-procedures-guidelines/policy-71.

By typing or writing my full legal name below, I confirm that I have read and understood the academic integrity statement above.

Instructions

- Submit any written solutions in a file named `writeup.pdf` to the A1 Dropbox on `Learn`. Submit any code to `Marmoset` at `https://marmoset.student.cs.uwaterloo.ca/`. No late assignment will be accepted. This assignment is to be done individually.
- I strongly encourage you to complete your write-up in Latex, using this source file. If you do, in your submission, please replace the author with your name and student number. Please also remove the due date, the Instructions section, and the Learning goals section. Thanks!
- Lead TAs:
  - Blake Vanberlo (bvanberlo@uwaterloo.ca)
  - Dake Zhang (dake.zhang@uwaterloo.ca)

  The TAs' office hours will be posted on MS Teams.

Learning goals

Uninformed and Heuristic Search

- Formulate a given problem as a heuristic search problem. Define the states, the initial state, the goal states, the action, the successor function, and the cost function.
- Trace the execution of Breadth-First Search and Depth-First Search.
- Define an admissible heuristic by solving a relaxed problem optimally. Explain why a heuristic is admissible.
- Trace the execution of the A* search algorithm with an admissible heuristic.
- Implement the A* search algorithm.

1 The Bridge and Torch Problem (25 marks + 1 bonus mark)

The Bridge and Torch problem is one version of a river crossing problem. The description below has been adapted from the Wikipedia page.

Four people would like to cross a bridge from left to right in the night. There is a narrow bridge, but it can only hold two people at a time. They have one torch. When any number of people are crossing the bridge, at least one person has to hold the torch. Person A can cross the bridge in 1 minute, B in 2 minutes, C in 5 minutes, and D in 8 minutes. When two people cross the bridge together, they must move at the slower person's pace.

The four people would like to cross the bridge from left to right in as little time as possible. Can you help them figure out a solution?

Please complete the following tasks.

(a) Formulate this problem as a search problem. Make sure you define the states, the initial state, the goal state, the successor function, and the cost function.

Marking Scheme: (8 marks)

- (2 marks) State definition.
- (1 mark) Initial state.
- (1 mark) Goal states.
- (2 marks) Successor function.
- (1 mark) Cost function.
- (1 mark) The quality of your formulation.

(b) Consider the heuristic function below. Explain why the heuristic function is admissible in no more than 2 paragraphs.

Hint: Construct a relaxed version of the original problem and show that for any state, the heuristic value is equal to the cost of the optimal solution to this relaxed problem.

For any state, the heuristic value is the number of minutes it takes for the slowest person on the left bank to cross the bridge.

Let me give you two examples. If state S_1 has all four people on the left and the torch is on the left, the heuristic value $h(S_1) = 8$. If state S_2 has A, B, C on the left bank and the torch is on the right, the heuristic value $h(S_2) = 5$.

Marking Scheme: (2 marks) A reasonable explanation

(c) Construct an admissible heuristic function h that dominates the one provided in part b. Describe your heuristic function in detail. Explain why your heuristic function is admissible. Also, explain why your heuristic function dominates the provided one in part b.

Hint: A heuristic function must specify a value for every state in the search graph.

Marking Scheme: (10 marks) + (1 bonus mark)

- (6 marks) Describe the heuristic function.
- (2 marks) Describe why the heuristic function is admissible.
- (2 marks) Describe why your heuristic function dominates the provided one.
- (1 bonus mark) Your heuristic function significantly improves upon the provided one. This is up to the TA's discretion.

(d) Execute the A* search algorithm on the problem using the provided heuristic function in part b. Do not perform any pruning. Draw the search tree until you have expanded 5 nodes. Expanding a node means removing the node from the frontier, generating its successors, and adding the successors to the frontier. In your search tree, label each node with $C + H = F$ where C is the cost of the path, H is the heuristic value, and F is the sum of the cost and the heuristic values.

Clearly specify the tie-breaking rule you are using. We suggest breaking ties by ordering the nodes in lexicographical order, but you can use any tie breaking rule as long as it is clearly specified in your solution.

We recommend using `https://app.diagrams.net/` to draw the search tree.

Marking Scheme: (5 marks) Correct search tree with 5 nodes expanded.

2 The Rush Hour Sliding Puzzle (98 marks)

In this programming question, you will solve the Rush Hour puzzle using the A* search and the depth-first search algorithms.

Take a look at an example of a Rush Hour puzzle below. The puzzle is on a 6 by 6 grid. We will number the rows as 0 to 5 from the top, and the columns as 0 to 5 from the left. In row 2, a horizontal car of length 2, called the goal car, is trying to escape through the exit on the right. There are horizontal and vertical cars of various lengths in the grid. A horizontal car can only move horizontally, whereas a vertical car can only move vertically. Each car may move more than one square in one step, but it cannot move over or through other cars. The goal is to move the cars around until the goal car reaches the exit, i.e. until the goal car is in the columns 4 and 5 in row 2.

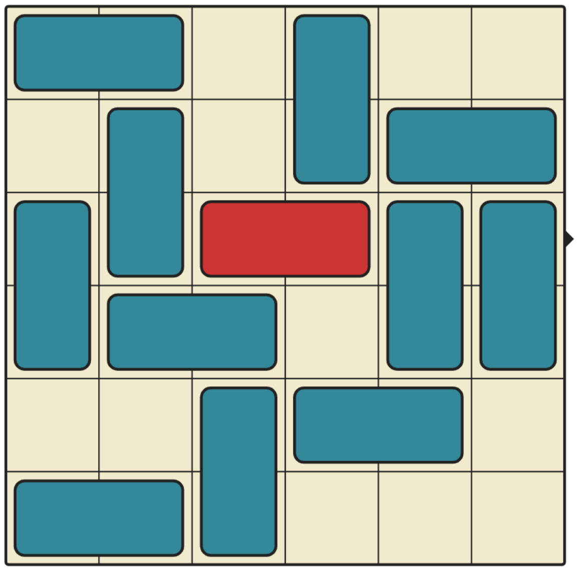

Figure 1: An example of a Rush Hour puzzle from https://www.michaelfogleman.com/rush/

You can make the following assumptions for this question.

- The puzzle must be on a 6 x 6 grid.
- The goal car is in row 2 and it has a length of 2.
- Besides the goal car, there is no other horizontal car in row 2.

Information on the Provided Code

We have provided three files `board.py`, `solve.py`, and `jams_posted.txt` on Learn. `board.py` contains code for handling input/output, representing states, etc. Your main task is to fill in the empty functions in `solve.py`.

Submit the `solve.py` to Marmoset only. We will use our version of `board.py` to test your code. Do not modify any provided function signatures in `solve.py`. Doing so will cause you to fail our tests. Feel free to add any new code to `solve.py`.

Input Format

The file `jams_posted.txt` contains 40 puzzles. You can use these puzzles to test your program. We will test your program using a different set of puzzles.

Below is an example of an input describing a puzzle.

```
example
6
1 2 h 2
2 0 v 2
4 0 h 2
3 1 v 3
4 1 v 2
4 3 h 2
.
```

- The first line assigns a name to the puzzle. In this case, the name is "example."
- The next line specifies the size of the grid. We only use 6 by 6 grid. So this number is always 6.
- The next line "`1 2 h 2`" gives a description of the goal car. The first two numbers $(1, 2)$ gives the (x, y) coordinates of the upper left corner of the car. The next letter "h" indicates that the car is horizontal ("v" would indicate that the car is vertical). The last number "2" indicates that the car has size 2.
- Each subsequent line, except the last line, describe a car in the puzzle, using the same format.
- The last line consists of a single period, indicating the end of the puzzle.

You can include multiple puzzles consecutively in the same file using the above format.

The Heuristic Functions for A* Search:

We have provided the implementation of the zero Heuristic function, which assigns a heuristic value of 0 to every state.

You must implement two other heuristic functions for A* search.

- Blocking Heuristic: The heuristic value for any goal state is zero. For any non-goal state, the heuristic value is one plus the number of cars blocking the path to the exit. For example, for the state in Figure 1, the blocking heuristic value is 3.
- Advanced Heuristic: Implement a third advanced heuristic of your choosing and invention. Your advanced heuristic should be consistent and should dominate the blocking heuristic.

Testing Your Program

Debugging and testing are essential skills for computer scientists. For this question, debugging your program may be especially challenging because of ties. Among "correct" implementations, the number of nodes expanded may vary widely due to how we handle the nodes with the same heuristic value on the frontier.

Please test your code using **Python 3.8.5**.

We rely on Python's hashing to generate a state's ID for breaking ties (see the Breaking Ties section below). However, Python' hashing function is not deterministic across different sessions. For example, you may get different hashing values of the same object for running your program multiple times. **Please set the environment variable PYTHONHASHSEED to 486 BEFORE running the Python script.** Note that setting the variable in the code/program will not work.

Implement **multi-path pruning for both DFS and A***. When there are multiple paths to a state, multi-path pruning explores the first path to the state and discards any subsequent path to the same state. Use an explored set to keep track of the states that have been expanded by the algorithm. When you **remove** a state from the frontier, check whether the state is in the explored set or not. If the state is in the explored set, then do nothing. Otherwise, add the state to the explored set and continue with the algorithm. Note that we perform pruning after we **remove** a state from the frontier, not before we **add** a state to the frontier.

DFS's behaviour depends on the order of adding a state's successors to the frontier. We will break ties by using the states' ID values. At each step, DFS will add the successors to the frontier in **decreasing** order of their IDs. In other words, DFS will expand the state with the smallest ID value among the successors.

A* search will also break ties using the states' ID values. Among several states with the same f value, A* will expand the state with the smallest ID value. If two states have the same ID value, A* will break ties using the states' parents — expanding the state whose parent has the smaller ID value.

Please complete the following tasks:

Submit your solutions to part (a) on Marmoset and submit your solutions to parts (b) and (c) on Learn.

(a) Complete the empty functions in `solve.py` and submit `solve.py` on Marmoset. Marmoset will evaluate your program for its correctness and efficiency.

For correctness, we have written unit tests for these functions: `get_path`, `is_goal`, `blocking_heuristic`, `get_successors`, `dfs`, `a_star`.

For each function, Marmoset provides one public test, which tests the function in a trivial scenario. There are also several secret tests. Before the deadline, you can only view the results of the public tests. After the deadline, Marmoset will run all the tests and calculate your marks.

Each test runs the function up to a predefined time limit. If the test passes if and only if the function terminates within the time limit and returns the expected result. Each test is all or nothing — there are no partial marks available.

Marking Scheme: (88 marks)

Unit tests on `get_path`, `is_goal`, `blocking_heuristic`, `get_successors`, `dfs`, and `a_star`.

- `get_path`: (1 public test + 2 secret tests) * 1 mark = 3 marks.
- `is_goal`: (1 public test + 4 secret tests) * 1 mark = 5 marks.
- `blocking_heuristic`: (1 public test + 9 secret tests) * 2 marks = 20 marks.
- `get_successors`: (1 public test + 9 secret tests) * 2 marks = 20 marks.
- `dfs`: (1 public test + 9 secret tests) * 2 marks = 20 marks.
- `a_star`: (1 public test + 9 secret tests) * 2 marks = 20 marks.

(b) Prove that the blocking heuristic is consistent using the definition of a consistent heuristic.

Marking Scheme: (3 marks) Proof is correct and easy to understand.

(c) Design and implement an advanced heuristic of your own invention. Your advanced heuristic should be consistent and dominate the blocking heuristic.

Prove that your advanced heuristic is consistent and dominates the blocking heuristic.

Implement your advanced heuristic. Show that A* search with the advanced heuristic expands fewer nodes than A* search with the blocking heuristic on all the 40 provided puzzles. (Make sure that you use the same PYTHONHASHSEED for all the program runs.)

Marking Scheme: (7 marks)

- (3 marks) Prove that your advanced heuristic is consistent.
- (2 marks) Prove that your advanced heuristic dominates the blocking heuristic.
- (2 marks) Show program output to support that the advanced heuristic dominates the blocking heuristic.