CS 486/686 Assignment 1 (122 marks)

Blake VanBerlo

Due Date: 11:59 pm on Wednesday, February 2, 2021

Changes

- v1.1: Marking scheme for Q1d
- v1.2: Marking scheme for Q2c

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`. If your written submission on Learn does not contain **one** file named `writeup.pdf`, we will deduct 5 marks from your final assignment mark.
- 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

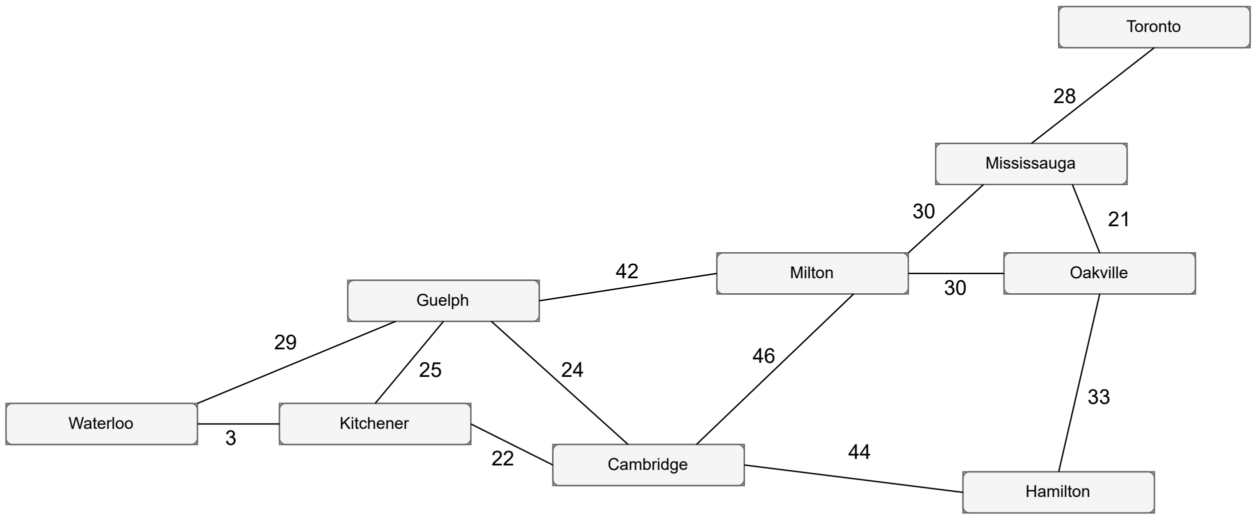
Uninformed and Heuristic Search

- 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 Shortest Route to Waterloo (22 marks)

Suppose that you want to drive to Waterloo from your family home in Toronto, since classes are returning to in-person delivery. You are conscious of your carbon footprint; therefore, you are seeking the shortest path from your house to Waterloo.

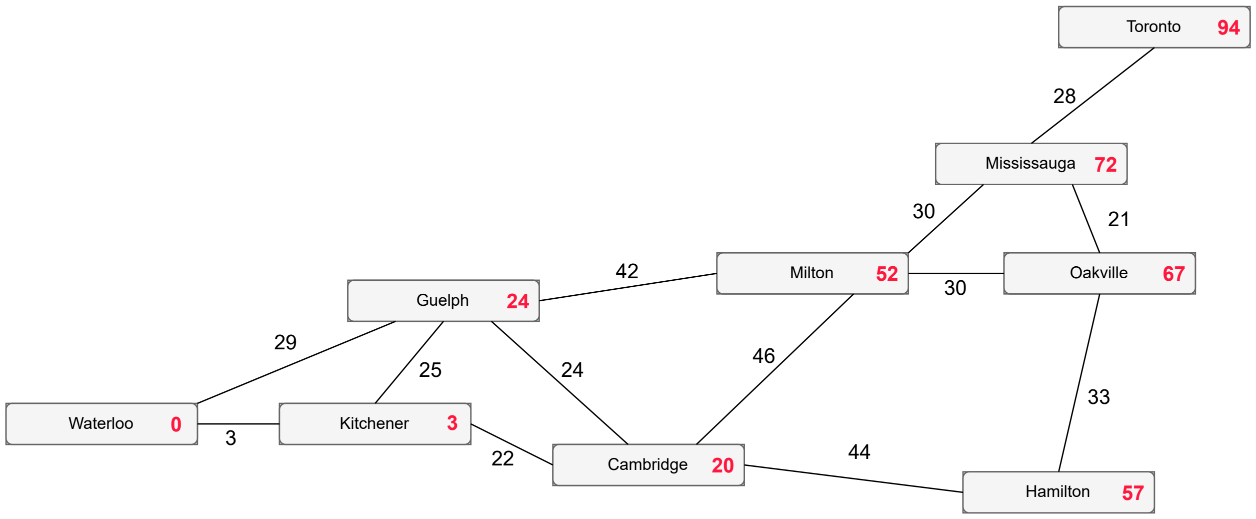
Below is a graph indicating the driving distances between various cities surrounding Toronto and Waterloo. All distances are given in kilometres.



You would like to apply A* search to identify the shortest route to Waterloo from your house in Toronto. Below are the components of the search problem formulation:

- **States:** Each city is the state. We will identify each state by the first 3 letters of its name. For example, the "Guelph" node is denoted as `Gue`.
- **Initial state:** `Tor`
- **Goal state:** `Wat`
- **Successor function:** State B is a successor of state A if and only if there exists an edge on the above graph connecting city B and city A.
- **Cost function:** The cost of an edge connecting two states is the distance between the cities that the states correspond to.

Your decide to use Euclidean distance as a heuristic function. That is, h is the Euclidean distance from a city to Waterloo (in kilometres). That is, if (x_C, y_C) is city C's longitude and latitude coordinates, $h(C) = \sqrt{(x_C - x_{Wat})^2 + (y_C - y_{Wat})^2}$. On the diagram below, the red text number to each city indicates its Euclidean distance to Waterloo.



Please complete the following tasks.

(a) In no more than one paragraph, describe why the Euclidean distance to the destination is an admissible heuristic function.

Marking Scheme: (4 marks)

- (4 marks) A reasonable explanation

(b) In no more than one paragraph, describe why Euclidean distance to the destination is a consistent heuristic function.

Marking Scheme: (4 marks)

- (4 marks) A reasonable explanation

(c) Prove that a heuristic function that is consistent is also admissible.

Hint: try a direct proof. Apply the definition of a consistent heuristic across an optimal path from some arbitrary node to a goal.

Marking Scheme: (6 marks)

- (4 marks) Correct proof
- (2 marks) The proof is clear, succinct, and easy to understand

(d) Execute the A* search algorithm on the problem using the Euclidean distance heuristic function as described above. Do not perform any pruning. Add nodes to the frontier in alphabetical order. Remember to stop if you expand the goal state.

When drawing nodes, remember to abbreviate cities by writing the first 3 letters. For example, label a "Waterloo" node as `Wat`. Annotate each node in the following format: $C + H = F$ 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 indicate which nodes you expanded and in what order. You do not need to write out the frontier, but the tree must show all paths expanded after removing a node from the frontier.

Break any F-value ties using alphabetical order. For example, "Milton" precedes "Mississauga" and should be expanded first if the F values for both nodes are the same.

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

Marking Scheme: (8 marks) Correct search tree with all nodes correctly expanded in order.

2 The Rush Hour Sliding Puzzle (100 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.

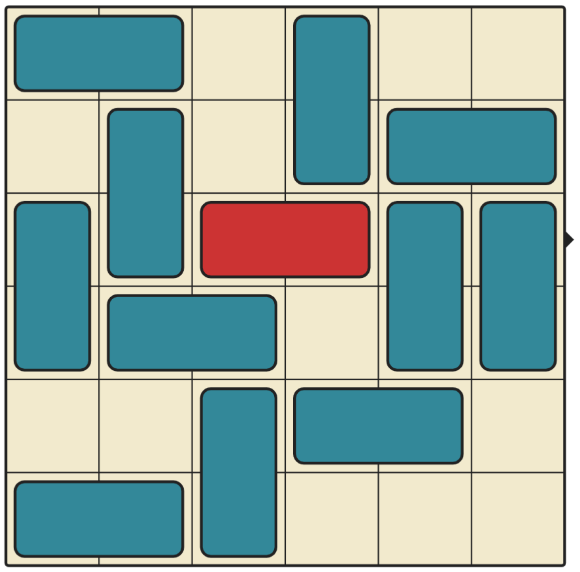


Figure 1: An example of a Rush Hour puzzle from https://www.michaelfogleman.com/rush/

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: (4 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.

Describe your advanced heuristic *in 6 sentences or less.*

Prove that your advanced heuristic dominates the provided heuristic. Be as succinct as possible.

Implement your advanced heuristic. Show program output to support 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: (8 marks)

- (3 marks) Heuristic description is clear and is no more than 6 sentences.
- (3 marks) Correct proof. Proof is clear and succinct.
- (2 marks) Program output supports that the advanced heuristic dominates the blocking heuristic.