CS 486/686 Assignment 1
(108 marks + 10 possible bonus marks)

Alice Gao

Due Date: 11:59 pm on Thursday, October 8, 2020

Instructions

- Submit the assignment in the A1 Dropbox on Learn. No late assignment will be accepted. This assignment is to be done individually.
- For any programming question, we will run your code in the CS-Teaching environment (linux.student.cs.uwaterloo.ca). You should make sure that (1) the language you use is available on this environment by default. (2) your code runs in this environment. We highly recommend using Python. If you don't know Python, it may be worthwhile to learn it for this course.
- Lead TAs:
  - Alister Liao (alister.liao@uwaterloo.ca)

  The TAs' office hours will be posted on MS Teams.
- Submit two files with the following names.
  - **writeup.pdf**
    - Include your name, email address and student ID in the writeup file.
    - If you hand-write your solutions, make sure your handwriting is legible and take good quality pictures. You may get a mark of 0 if we cannot read your handwriting.
  - **code.zip**
    - Include your program, and a script to run your program.
    - The TA will run your program using the following command in a terminal in the environment linux.student.cs.uwaterloo.ca.

      `bash a1.sh`

      You may get a mark of 0 if we cannot run your program.

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 Missionaries and Cannibals Problem (46 marks)

On the left side of the river, there are three missionaries, three cannibals, and a boat. The boat can hold one or two people. Every time the boat crosses the river, we count it as one boat trip. Our goal is to find a way to move all the people from the left side to the right side of the river using the smallest number of boat trips subject to the constraints below.

- The boat can only cross the river if there is at least one person in it.
- The missionaries must not be eaten by the cannibals.

  That is, at any moment in time, on either side of the river, if there is at least one missionary and the number of cannibals is greater than the number of missionaries, the missionaries will be eaten by the cannibals.

You will apply uninformed and informed search algorithms to solve this problem.

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.

In parts b and c, we will provide you with some other formulations. After that, you will have a chance to revise and improve your formulation in part d. It is important that you come up with your own formulation first before looking at the next two parts. This is a valuable learning opportunity and you don't want to miss it!

Marking Scheme:

(2 marks)

We will mark this part for completion only. As long as you provide a state definition and a successor function, you will get the 2 marks.

- (1 mark) for stating a state definition.
- (1 mark) for stating a successor function.

(b) Your friend Sydney came up with the following formulation. If Sydney's formulation is correct, applying a (optimal) search algorithm on this formulation will allow us to find a (optimal) solution. Unfortunately, Sydney's formulation has a problem, which significantly affects the performance of a search algorithm on this problem.

Describe the problem with Sydney's formulation in no more than 3 sentences. Then, discuss how this problem affects the performance of a search algorithm in no more than 2 paragraphs.

Hint: If you have trouble identifying the problem, draw the search graph and look at the states in the search graph. How many states are there? Why are these states in the graph?

Sydney's formulation:

- **State:** Each state is given by $M_1M_2M_3C_1C_2C_3B$. M_i and C_j are 1 if the missionary/cannibal is on the left bank and 0 otherwise. $B = d$ if the boat is on the left bank (departure side) and $B = a$ if the boat is on the right bank (arrival side).

  A state is valid if and only if it satisfies the following two constraints.

  If there is at least one missionary on the left bank, then the number of cannibals on the left bank must be less than or equal to the number of missionaries on the left bank.

  If there is at least one missionary on the right bank, then the number of cannibals on the right bank must be less than or equal to the number of missionaries on the right bank.
- **Initial state:** $111111d$.
- **Goal states:** Any state where every person is on the right bank is a goal state. $000000x$ where x can be a or d.
- **Successor function:** Consider a valid state. If there is at least one person on the same side as the boat, then we can generate a successor state by moving the boat with one or two people from one side of the river to the other side of the river. A successor state produced is valid as long as it satisfies the state definition.
- **Cost function:** The cost of each boat trip is one.

Marking Scheme:

(6 marks)

- (2 mark) Describe the problem with Sydney's formulation.
- (4 marks) Describe how the problem affects the performance of a search algorithm.

(c) Your other friend Cypress proposed a formulation that is similar to Sydney's. However, Sydney included the constraints in the state definition, whereas Cypress put the constraints in the successor function. See the state and the successor function in the two formulations below.

Sydney and Cypress argued for a while but couldn't figure out which choice is better. Could you help them settle the argument? Which location for the constraints is better? State your answer and justify your answer in no more than 5 sentences.

Sydney's formulation:

- **State:** Each state consists of $(M_1, M_2, M_3, C_1, C_2, C_3, B)$. M_i and C_j are 1 if the missionary/cannibal is on the left bank and 0 otherwise. $B = d$ if the boat is on the left bank (departure side) and $B = a$ if the boat is on the right bank (arrival side).

  A state is invalid if it violates either constraint below.

  - If there is at least one missionary on the left bank, then the number of cannibals on the left bank must be less than or equal to the number of missionaries on the left bank.
  - If there is at least one missionary on the right bank, then the number of cannibals on the right bank must be less than or equal to the number of missionaries on the right bank.

- **Successor function:** Given any valid state, we can generate a successor state by moving the boat with one or two people from one side of the river to the other side of the river.

Cypress's formulation:

- **State:** Each state consists of $(M_1, M_2, M_3, C_1, C_2, C_3, B)$. M_i and C_j are 1 if the missionary/cannibal is on the left bank and 0 otherwise. $B = d$ if the boat is on the left bank (departure side) and $B = a$ if the boat is on the right bank (arrival side).

- **Successor function:** Given any valid state, we can generate a successor state by moving the boat with one or two people from one side of the river to the other side of the river.

  A state does not have any successor if it violates either constraint below.

  - If there is at least one missionary on the left bank, then the number of cannibals on the left bank must be less than or equal to the number of missionaries on the left bank.
  - If there is at least one missionary on the right bank, then the number of cannibals on the right bank must be less than or equal to the number of missionaries on the right bank.

Marking Scheme:

(6 marks)

- (2 marks) Correct answer
- (4 marks) A reasonable justification

(d) Based on the previous parts, come up with the best formulation for this problem. Make sure you define the states, the initial state, the goal state, and the successor function. Assume the cost function is the same as the one in Sydney's formulation.

We will mark your formulation on its correctness and on its quality.

Marking Scheme:
(10 marks)

- (4 marks) The quality of your formulation.
- (2 marks) State definition.
- (2 marks) Constraints.
- (1 mark) Initial state and goal states.
- (1 mark) Successor function.

(e) Draw the search graph. The search graph should contain all of the states and all of the arcs based on your problem formulation in part d. Highlight the start state and the goal states in your graph.

If you search graph has more than 35 nodes in it, go back to part d and rethink your formulation.

Marking Scheme: (6 marks)

- (4 marks) includes all the states (and handles the constraints correctly).
- (2 marks) draws the arcs correctly (and handles the constraints correctly).

(f) If your goal is to find a solution quickly (while minimizing the number of nodes expanded), which **uninformed search algorithm** would you use to solve this problem?

You can choose among breadth first search, depth first search, iterative-deepening search, and lowest-cost-first search. You can also use any pruning strategy discussed in lectures. Since you already drew the search graph, you can assume that it is given.

State the algorithm of your choice and justify your answer in at most 5 sentences.

Marking Scheme:

(3 marks) Provide a good reason for choosing a particular uninformed algorithm.

(g) Propose an admissible heuristic function h. Describe the heuristic function in detail. Then, explain why the heuristic function is admissible. Some marks will be assigned to the quality of the heuristic function.

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

Hint: If you relax the problem by removing a constraint, then the optimal solution to the relaxed problem is guaranteed to be an admissible heuristic function. One way to justify why your heuristic function is admissible is to explain how you relaxed the problem and solved the relaxed problem optimally.

Marking Scheme: (10 marks)

- (6 marks) Describe the heuristic function.
- (2 marks) Describe why the heuristic function is admissible.
- (2 marks) The quality of your heuristic function.

(h) Your friend is considering using **the A* search algorithm** instead of using an uninformed search algorithm to solve the missionaries and cannibals problem. Would the A* search algorithm be a better choice than an uninformed search algorithm? Why or why not? State your answer and justify it in at most 5 sentences.

Tip: Do not trace the algorithm on the problem. We are not looking for the exact number of states expanded for either algorithm. Instead, formulate your answer by looking at the search graph and reasoning about the behaviour of the search algorithms.

Marking Scheme:

(3 marks)

- (3 marks) A good justification for your answer.

2 The Hua Rong Dao Sliding Puzzle (62 marks + 10 possible bonus marks)

The description of this question looks very long. Don't panic! We provided lots of details to help you progress through the question smoothly.

Please read the following collaboration policy carefully. This policy applies to this question only.

Collaboration policy:

For all the functions related to the search algorithm, the code you submit must be your own. These functions include is_goal, get_cost, get_heuristic, get_successors, a_star, and dfs.

For all other functions, you are allowed to consult any resources or collaborate with anyone you like. You may discuss the assignment questions verbally with others, but you should come away from these discussions with no written or electronic records. In your write-up, you must clearly cite any source you consult or any person you collaborate with. Do not post your code publicly on any website.

We take academic integrity very seriously. If you have questions, please ask on Piazza and consult the university academic integrity policies on the course outline.

Hua Rong Dao is a sliding puzzle that is popular throughout China. Check out the following page for some background story on the puzzle and an English description of the rules.

http://chinesepuzzles.org/huarong-pass-sliding-block-puzzle/

The board arrangement:

The puzzle has a rectangle board of width 4 and height 5. We will consider the variant with ten pieces. The ten pieces are of four different kinds. There is exactly one 2×2 piece. There are five 1×2 pieces. Each such piece is placed horizontally or vertically. In addition, there are four single-space pieces. After placing the ten pieces on the board, there should be exactly two empty spaces.

See an example of an initial configuration of the puzzle in Figure 1 below. (Don't worry about the Chinese characters. They are not important for understanding the rules of the puzzle.) In this configuration, one of the 1×2 pieces is placed horizontally and the other four 1×2 pieces are placed vertically.

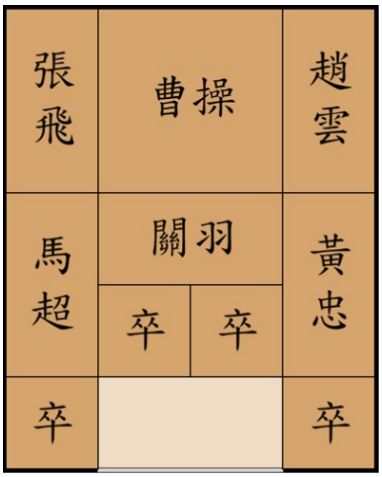

Figure 1: The classic configuration for Hua Rong Dao

The goal of the puzzle:

At the bottom center position of the region, there is an opening that is 2-space wide. That is the Hua Rong Dao/Pass. The goal of this puzzle is to move the pieces horizontally and/or vertically within the region until the 2×2 piece is right above the opening at the bottom. At this position, the 2×2 piece can slide down and "escape" through the Hua Rong Pass. You may only move a piece either horizontally or vertically if possible. No piece can be rotated.

Other initial configuration:

There are many other initial configurations for this puzzle. The Chinese wikipedia page for this puzzle shows you 32 initial configurations including the one in Figure 1. The Chinese writings on the page are not important. Feel free to use Google Translate if you are curious about what it says. The link below each puzzle configuration opens another page on which you can play the puzzle interactively and see the solution interactively.

Counting the number of moves:

An important note on counting the number of moves: We will count moves in a different way than the website does. The website above counts moves as follows. If a single piece moves multiple times, it is counted as one move rather than multiple moves. We will count moves differently. Every time a piece moves one space in any direction, we will count it as one move. For example, if I start by moving the single-space piece in the bottom left corner by two spaces to the right, the website will count this as one move whereas we will count it as two moves. As a result, for the initial configuration shown above, the optimal solution based on the website takes 81 steps, whereas the optimal solution based on our method of counting moves takes 116 steps. Make sure that your search problem formulation is consistent with our way of counting moves.

Input format:

You will implement two search algorithms: **A* search and Depth-first search**. We will test your implementations with two different puzzle configurations. The first configuration is the one in Figure 1 (which is also the first configuration on the Wikipedia page). The other configuration will be randomly chosen among the 31 other configurations on the Wikipedia page.

The two puzzle configurations will be provided in two files named **puzzle1.txt** and **puzzle2.txt**. You can hard-code these files names in your program. As an example, the content of the file **puzzle1.txt** is as follows. This is the initial configuration given in the picture above. Note that all the single-space pieces are denoted by 7.

```
2113
2113
4665
4775
7007
```

Output format:

Once your program finds a solution, save it in a file using the format described below. Be sure to **leave at least one empty line between separate states in your solution, but not between rows of the same state**.

For A* search, name the files **puzzle1sol_astar.txt** and **puzzle2sol_astar.txt**.

For DFS, name the files **puzzle1sol_dfs.txt** and **puzzle2sol_dfs.txt**.

Each file contains:

- The initial configuration of the puzzle.
- The cost of the solution.
- The number of states expanded by the search algorithm.
- The solution.

See an example of a solution file below. In this example, the solution is incomplete and the details are left out with the three dots at the end.

```
Initial state:
2113
2113
4665
4775
7007

Cost of the solution: 116

Number of states expanded: 80000

Solution:

0
2113
2113
4665
4775
7007

1
2113
2113
4665
4775
0707

2
2113
2113
4665
4775
0770

...
```

Running your program:

For parts c and d, your marks will depend on whether the TA can run your program to produce the expected output. The TA will run your program using the following command in a terminal in the CS teaching environment (linux.student.cs.uwaterloo.ca)

```
bash a1.sh
```

You must include the file **a1.sh** in **code.zip**. If you are using a compiled language, ensure that this script is able to compile and run your code. If you are using Python, this script will just run your code directly. The TA will run the script for at most 5 minutes. The script should produce the following output files:

1. Results of running A* search on **puzzle1.txt**, saved to **puzzle1sol_astar.txt**
2. Results of running A* search on **puzzle2.txt**, saved to **puzzle2sol_astar.txt**
3. Results of running DFS on **puzzle1.txt**, saved to **puzzle1sol_dfs.txt**
4. Results of running DFS on **puzzle2.txt**, saved to **puzzle2sol_dfs.txt**

Please complete the following tasks:

(a) To implement the program more smoothly, you should first think about the logics within each function. To help you to do this, please answer the following two questions briefly. We will give you the marks as long as you write down something reasonable.

- For each state, what information would you keep track of? How would you store the information? Which helper functions do you plan to implement to manipulate a state?
- How would you implement the successor function? Describe the steps at a high level. What are the different cases that you need to consider?

Note: There are multiple ways of representing each state of the puzzle because the pieces have different sizes. Be sure to provide enough details so that you can easily translate it into code later on.

Marking Scheme:
(4 marks)

- (2 marks) How would you store a state?
- (2 marks) How would you implement the successor function?

(b) Consider the heuristic function $h_0(n)$ described below. Consider the heuristic function $h_0(n)$ described below. For any puzzle configuration denoted by n, let $h_0(n)$ be the Manhattan distance between the location of the upper-left corner of the 2×2 piece in the configuration n and the location of the upper-left corner of the 2×2 piece in any goal state of the puzzle.

Is this heuristic function admissible? State your answer and justify your answer in at most 3 sentences.

Bonus question: Could you propose an admissible heuristic function that is better than $h_0(n)$? You will earn up to 10 bonus marks if you implement the heuristic function and show that it significantly reduces the number of nodes expanded. It does not count if you make a minor modification to $h_0(n)$, e.g. change the heuristic value of one state only.

Marking Scheme:

(2 marks) and (10 possible bonus marks)

- (1 mark) Correct answer
- (1 mark) Correct justification
- (10 bonus marks) Describe the heuristic function, implement it and show that it significantly reduces the number of nodes expanded.

(c) Write a program to solve the Hua Rong Dao sliding puzzle using **A* search**.

We have broken down this problem into several parts for you. This is to help you build your program incrementally and debug the individual helper functions before you test the entire program. You do not have to follow the steps below exactly. However, **if a step requires you to implement a function, make sure your function uses the provided name**. If your program does not run or does not produce the correct solution, you may receive part marks for implementing the individual helper functions correctly.

We will test your implementation with two different puzzle configurations. **Follow the instructions in "Running your program". If the TA cannot run your program, you will get very few marks for this part.**

If you like, follow the recommended steps below to implement your program.

(1) Based on the state definition in your problem formulation, implement a data structure to store a state.

Note: There are multiple ways of representing a state because the pieces have different sizes. If you discover that your state definition does not have enough details or you prefer a different state definition, please go back and revise your problem formulation in part a.

(2) Implement a function **read_puzzle(id)** which reads in an initial configuration of the puzzle from a file and store it as a state in your program. The id parameter can be used to specify which one of **puzzle1.txt** and **puzzle2.txt** the function is trying to read in.

(3) Implement a data structure to store a path. A path is a sequence of states.

When solving this puzzle, we care about finding a path to a goal state instead of simply reaching a goal state. We thought of two ways of handling this. Either approach (or a third approach that you come up with) will be acceptable.

- You may store the path to the current state directly in the frontier instead of only storing the current state.
- You may store only the current state in the frontier. However, you will need to store additional information in the current state. For example, you will need to store the parent state of the current state so that you can backtrack to recover the path at the end. You may also want to store the cost of the path to the current state, i.e. the g value.

(4) Implement a priority queue to store the frontier. You can use an existing implementation as long as you cite the source.

(5) Implement the function **is_goal(state)**: The function takes a state and returns true if and only if the state is a goal state.

(6) Implement the function **get_successors(state)**: The function takes a state and returns a list of its successor states.

(7) Implement the function **get_cost(path)**: The function takes a path and returns the cost of the path (the g value of the state). If you decide to store only the current state and not the path in the frontier, implement the function **get_cost(state)** instead.

(8) Implement the function **get_heuristic(state)**: The function takes a state and returns the heuristic estimate of the cost of the optimal path from the state to a goal state (the h value of the state).

(9) Implement the function **a_star(initial_state)**: The function solves the Hua Rong Dao sliding puzzle using A* search given the initial configuration of the puzzle.

Marking Scheme:

(40 marks)

- (2 marks) The program follows all the specifications (including function names, file names, and output format). These 2 marks are all or nothing.
- (2 marks) The TA can run your program to produce the specified output for a puzzle configuration. These 2 marks are all or nothing. The TA will run your program for up to 5 minutes.
- (36/36 marks) Your program finds the correct optimal solution for both puzzle configurations.

  (27/36 marks) Your program finds the correct optimal solution for one puzzle configuration only.

  (At most 18/36 marks) If your program does not find the correct optimal solution for either puzzle configuration, you can get at most 12 marks out of the 24 marks, based on the TA's discretion.

- (6 marks) Your implementations of the **is_goal(state)**, **get_cost(path)**, and **get_heuristic(state)** functions are readable and understandable.
- (7 marks) Your implementation of the **get_successors(state)** function is readable and understandable.
- (5 marks) Your implementation of the **a_star(initial_state)** function is readable and understandable.

(d) Write a program to solve the Hua Rong Dao sliding puzzle using **Depth-First Search**.

Similar to part c, we will test your implementation with two different puzzle configurations.

If you like, follow the recommended steps below:

(1) You should be able to re-use most of the data structures and helper functions from part c, except the following.

(2) Implement a LIFO stack to store the frontier.

(3) Implement the function **dfs(initial_state)**: The function solves the Hua Rong Dao sliding puzzle using Depth-First Search given the initial configuration of the puzzle.

Marking Scheme:

(8 marks)

- (2 marks) The TA can run your program to produce the specified output for a puzzle configuration. These 2 marks are all or nothing.
- (6 marks) Your program finds a solution for both puzzle configurations. The solutions found are similar to that of our implementation.

  (4/6 marks) Your program finds a solution for one puzzle configuration only. The solution found is similar to that of our implementation.

  (2/6 marks) Your program does not find a solution for either puzzle configuration. However, the TA can understand most of your program.

  (0/6 marks) Your program does not find a solution for either puzzle configuration. Your program is not readable to the TA.

(e) Compare and contrast your results for A* and DFS search. Which algorithm finds a better solution (you can define what it means for a solution to be "better")? Which algorithm expands fewer states? Discuss your observations in no more than two paragraphs.

Marking Scheme:

(4 marks)

- (2 marks) Compare the quality of the solutions found.
- (2 marks) Compare the number of states expanded/generated.

(f) Which of DFS and A* should we use to solve the Hua Rong Dao puzzle? Give one reason for choosing DFS over A*. Give one reason for choosing A* over DFS.

Marking Scheme:
(4 marks)

- (2 marks) Give a reason for choosing DFS. All or nothing.
- (2 marks) Give a reason for choosing A*. All or nothing.