CS 486/686 Assignment 1 (121 marks in total)

Instructor: Alice Gao

Due Date: 6:00 pm on Tuesday, June 4, 2019

Instructions

- Submit the assignment in the Dropbox labeled Assignment 1 Submissions in the Assignment 1 folder on LEARN. No late assignment will be accepted. This assignment is to be done individually.
- For any programming question, you may use the language of your choice. 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)
  - Frederic Bouchard (frederic.bouchard@uwaterloo.ca)

  The TAs' office hours will be posted on the course website.
- Submit two files with the following naming conventions.
  - **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, a script to run your program, and a README.txt file with instructions to run the script. You may get a mark of 0 if we cannot run your program.

Learning goals

Informed 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.

Constraint Satisfaction Problem

- Formulate a given problem as a constraint satisfaction problem. Define the variables, the domains, and the constraints.
- Trace the execution of the backtracking search algorithm and the AC-3 arc consistency algorithm on a given CSP.

Local Search

- Formulate a given problem as a local search problem.
- Given a local search problem, verify whether a state is a local optimum or not.

1 The Hua Rong Dao sliding puzzle (104 marks)

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

A fellow undergraduate student implemented the A* search algorithm on this puzzle. It took this student 8 hours to write and debug a Python program that finds the optimal solution to the puzzle. Based on this, you may want to start working on this ASAP. =)

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 variants with ten pieces. The ten pieces are of four different kinds. There is exactly one two by two piece. There are five one by two pieces. Each such piece is either placed horizontally or vertically. In addition, there are four single-space pieces. There should be exactly two empty spaces after placing the ten pieces on the board.

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 puzzle.) In this configuration, one of the one by two pieces is placed horizontally and the other four pieces are placed vertically.

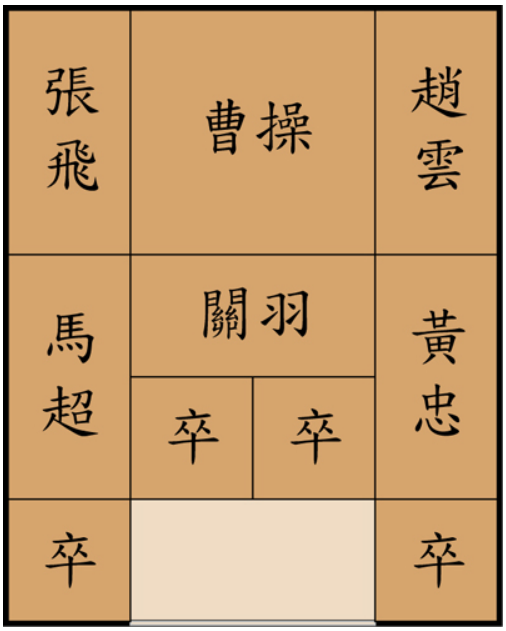

Figure 1: The classic configuration for Hua Rong Dao

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 either horizontally or vertically within the region until the two by two piece is right above the opening at the bottom. At this position, the two by two piece can slide down and "escape" the region 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 configuration 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 also 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 in the following way. 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. Also, 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 three search algorithms: A* search, Breadth-first search and Depth-first search. We will test the correctness of 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**. 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 (or the optimal solution), save it in a file using the following format. The names of the files are explained in the corresponding sections.

Each file should contain the following.

- First, print the initial configuration of the puzzle using the format below. All of the vertical pieces are represented by 3. All of the horizontal pieces are represented by 2. All of the single pieces are represented by 4.

  ```
  Initial state:
  3113
  3113
  3223
  ```

```
3443
4004
```

- Print the cost of the (optimal) solution using the following format.

```
Cost of the (optimal) solution: 116
```

- Print the number of states expanded and generated by the search algorithm.

```
Number of states expanded: 80000
Number of states generated: 90000
```

- Print the optimal solution using the following format.

```
(Optimal) solution:

0
3113
3113
3223
3443
4004

1
3113
3113
3223
3443
0404

2
3113
3113
3223
3443
0440
```

Please complete the following tasks:

1. Formulate the Hua Rong Dao sliding puzzle as a search problem. Make sure you define the states, the initial state, the goal state, the successor function, and the cost function.

   Note: There are multiple ways of representing each state of the puzzle because the pieces have different sizes. Be sure to provide enough details in your state definition so that it will be easy to translate it into code later on.

   Marking Scheme: (12 marks)

   - (4 marks) State definition
   - (1 mark) Initial state
   - (2 marks) Goal states
   - (4 marks) Action and successor function
   - (1 mark) Cost function

2. Draw the search tree of **Breadth-first search** until you have **expanded** 3 **nodes**. (Hint: draw.io will help you produce beautiful search trees in no time.)

   Choose an order of adding a state's successors to the frontier. Describe this order clearly and use it consistently.

   Label the nodes in order of expansion on the right side of each node. The first node you expand should be the initial state.

   If a state is already in the frontier, do not add the state to the frontier again.

   Assume that Breadth-first search prunes explored states. That is, when you expand a state by removing it from the frontier, if the state has been explored already, do nothing with this state and continue to expand another state. When you encounter an explored state, do not count it towards the states expanded.

   Marking Scheme: (9 marks)

   - (3 marks) Generated successors correctly.
   - (3 marks) The order of expansion and the order of adding nodes to the frontier are consistent.
   - (3 marks) Chose to expand the correct node and labeled expanded nodes correctly.

3. Draw the search tree of **Depth-first search** until you have **expanded** 4 **nodes**.

   When adding a state's successors to the frontier, use the same order as you did in part 2.

   Label the nodes in order of expansion on the right side of each node. The first node you expand should be the initial state.

   If a state is already in the frontier, do not add the state to the frontier again.

Assume that Depth-first search prunes explored states, using the same mechanism described in part 2.

Marking Scheme: (12 marks)

- (4 marks) Generated successors correctly.
- (4 marks) The order of expansion and the order of adding nodes to the frontier are consistent.
- (4 marks) Chose to expand the correct node and labeled expanded nodes correctly.

4. Propose an admissible heuristic function $h(n)$ and describe it in detail. Be sure to explain why it is admissible.

   Marking Scheme: (4 marks)

   - (2 marks) Define a heuristic function correctly.
   - (2 marks) Explain why the heuristic function is admissible.

5. Using the heuristic function you proposed in part 4, draw the search tree of **A* Search** until you have **expanded** 3 **nodes**.

   Label each node with its heuristic value $h(n)$ on the left side of the each node. Label the nodes in order of expansion on the right side of each node. The first node you expand should be the initial state.

   If two nodes on the frontier have the same $f(n)$ value (i.e. there is a tie), make sure that you describe how you determine which node to expand first (i.e. the tie breaking rule). Choose an order of expansion and use it consistently.

   If a state is already in the frontier, do not add the state to the frontier again.

   Assume that A* search prunes explored states, using the same mechanism described in part 2.

   Marking Scheme: (11 marks)

   - (3 marks) Generated successors correctly.
   - (3 marks) Use a consistent order to add nodes to the frontier.
   - (3 marks) Chose to expand the correct node and labeled the expanded nodes correctly.
   - (2 marks) The tie breaking rule is clearly specified and used correctly.

6. 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 helpers 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 the correctness of your implementation with two different puzzle configurations. The formats of the input files are specified in the Input Format section above. **Be sure to submit your entire program as well as clear instructions for running your program. If the TA cannot run your program, you will get very few marks for this part.**

   If your program finds the optimal solution to a puzzle, store the following output in the files **puzzle1sol_astar.txt** and **puzzle2sol_astar.txt** respectively. The format of each file is specified in the Output Format section above.

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

   (a) 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 1.

   (b) 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.

   (c) 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.

   (d) Implement a priority queue to store the frontier.

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

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

(g) Implement the function **get_cost(path)**: The function takes a path and returns the cost of the path. In other words, this function returns 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.

(h) 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 based on your heuristic function. In other words, this function returns the h value of the state.

(i) 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: (28 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.
- (24/24 marks) Your program finds the correct optimal solution for both puzzle configurations.

  (18/24 marks) Your program finds the correct optimal solution for one puzzle configuration only.

  (At most 12/24 marks) If your program does not find the correct optimal solution for either puzzle configuration, you can get at most 9 out of 20 marks. See the next bullet point for more details.
- If your program does not find the correct optimal solution, you can get at most 9 out of 20 marks. These 9 marks will be awarded below, based on the TA's discretion.

  (4 marks) The TA can read through and understand your implementation of the **is_goal(state)**, **get_cost(path)**, and **get_heuristic(state)** functions.

  (4 marks) The TA can read through and understand your implementation of the **get_successors(state)** function.

  (4 marks) The TA can read through and understand your implementation of the **a_star(initial_state)** function.

7. Write a program to solve the Hua Rong Dao sliding puzzle using Breadth-First Search.

   We will test the correctness of your implementation with two different puzzle configurations. The formats of the input files are specified in the Input Format section above. **Be sure to submit your entire program as well as clear instructions for running your program. If the TA cannot run your program, you will get very few marks for this part.**

   If your program finds a solution, store the following output in the files **puzzle1sol_bfs.txt** and **puzzle2sol_bfs.txt**. The format of each file is specified in the Output Format section above.

   If you like, follow the recommended steps below:

   (a) You should be able to re-use most of the data structures and helper functions from part 6, except the following.

   (b) Implement a FIFO queue to store the frontier.

   (c) Implement the function **bfs(initial_state)**: The function solves the Hua Rong Dao sliding puzzle using Breadth-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. The TA will run your program for up to 5 minutes.
   - (6 marks) Your program finds the correct optimal solution for both puzzle configurations.

     (4/6 marks) Your program finds the correct optimal solution for one puzzle configuration only.

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

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

8. Write a program to solve the Hua Rong Dao sliding puzzle using Depth-First Search.

   We will test the correctness of your implementation with two different puzzle configurations. The formats of the input files are specified in the **Input Format** section above. **Be sure to submit your entire program as well as clear instructions for running your program. If the TA cannot run your program, you will get very few marks for this part.**

If your program finds a solution, store the following output in the files **puzzle1sol_dfs.txt** and **puzzle2sol_dfs.txt**. The format of each file is specified in the **Output Format** section above.

If you like, follow the recommended steps below:

(a) You should be able to re-use most of the data structures and helper functions from part 6, except the following.

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

(c) 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. The TA will run your program for up to 5 minutes.
- (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 read through and 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.

9. Compare your results for A* and BFS search in terms of the solution found and the number of states expanded/generated. Discuss your observations in no more than two paragraphs.

   Marking Scheme: (4 marks)

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

10. Compare your results for A* and DFS search in terms of the solution found and the number of states expanded/generated. Discuss your observations in no more than two paragraphs.

    Marking Scheme: (4 marks)

    - (2 marks) Compare the solution found.

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

11. Which of the three algorithms would you choose to solve the Hua Rong Dao puzzle? Why?

    Marking Scheme: (4 marks)

    - (2 marks) Give an answer.
    - (2 marks) Give explanations.

2 Solving 4-Queens as a CSP (15 marks)

Recall that we have formulated the 4-queens problem as a CSP, as described below. In this problem, you will solve this CSP using backtracking search and the AC-3 arc-consistency algorithm with an initial assignment.

- Assume that exactly one queen is in each column. Given this, we only need to keep track of the row position of each queen.
- Variables: x_0, x_1, x_2, x_3 where x_i is the row position of the queen in column i, where $i \in \{0, 1, 2, 3\}$.
- Domains: $dom(x_i) = \{0, 1, 2, 3\}$ for all x_i.
- Constraints:

  No pair of queens are in the same row or diagonal.

  $(\forall i(\forall j((i \neq j) \rightarrow ((x_i \neq x_j) \wedge (|x_i - x_j| \neq |i - j|)))))$

  i.e., $((x_0 \neq x_1) \wedge (|x_0 - x_1| \neq 1) \wedge (x_0 \neq x_2) \wedge (|x_0 - x_2| \neq 2) \wedge (x_0 \neq x_3) \wedge (|x_0 - x_3| \neq 3) \wedge (x_1 \neq x_2) \wedge (|x_1 - x_2| \neq 1) \wedge (x_1 \neq x_3) \wedge (|x_1 - x_3| \neq 2) \wedge (x_2 \neq x_3) \wedge (|x_2 - x_3| \neq 1))$

Start with the initial assignment $x_0 = 1$. Trace the execution of the AC-3 arc consistency algorithm. We have provided the first few steps to get you started. Please **add 10 more steps** after the provided ones.

Please follow the conventions below when describing the steps.

- Always remove the first pair from the set of variable-constraint pairs.
- Always add pairs to the end of the list of variable-constraint pairs.
- If a step does not change the domain of the variable, write "No change to the domain".
- If a step does change the domain of the variable, describe the value removed, the variable-constraint pairs to be added back to the set, the resulting domains, and the resulting set of variable-constraint pairs.
- When describing the variable-constraint pairs to be added back to the set, please include all possible pairs even if a pair is already in the set.

Add 10 more steps after the provided steps below.

1. The starting domains:

   $dom(x_0) \in \{0,1,2,3\}$, $dom(x_1) \in \{0,1,2,3\}$, $dom(x_2) \in \{0,1,2,3\}$, and $dom(x_3) \in \{0,1,2,3\}$

   Assign $x_0 = 1$.

   The resulting domains:

   $x_0 = 1$, $dom(x_1) \in \{0,1,2,3\}$, $dom(x_2) \in \{0,1,2,3\}$, and $dom(x_3) \in \{0,1,2,3\}$

   The set of variable-constraint pairs:

   $(x_0, x_0 \neq x_1)$, $(x_1, x_0 \neq x_1)$, $(x_0, x_0 \neq x_2)$, $(x_2, x_0 \neq x_2)$, $(x_0, x_0 \neq x_3)$, $(x_3, x_0 \neq x_3)$, $(x_1, x_1 \neq x_2)$, $(x_2, x_1 \neq x_2)$, $(x_1, x_1 \neq x_3)$, $(x_3, x_1 \neq x_3)$, $(x_2, x_2 \neq x_3)$, $(x_3, x_2 \neq x_3)$, $(x_0, |x_0 - x_1| \neq 1)$, $(x_1, |x_0 - x_1| \neq 1)$, $(x_0, |x_0 - x_2| \neq 2)$, $(x_2, |x_0 - x_2| \neq 2)$, $(x_0, |x_0 - x_3| \neq 3)$, $(x_3, |x_0 - x_3| \neq 3)$, $(x_1, |x_1 - x_2| \neq 1)$, $(x_2, |x_1 - x_2| \neq 1)$, $(x_1, |x_1 - x_3| \neq 2)$, $(x_3, |x_1 - x_3| \neq 2)$, $(x_2, |x_2 - x_3| \neq 1)$, $(x_3, |x_2 - x_3| \neq 1)$.

2. Remove $(x_0, x_0 \neq x_1)$

   No change to the domain

3. Remove $(x_1, x_0 \neq x_1)$

   Remove 1 from $dom(x_1)$.

   Add back constraints:

   $(x_0, x_0 \neq x_1)$, $(x_2, x_1 \neq x_2)$, $(x_3, x_1 \neq x_3)$, $(x_0, |x_0 - x_1| \neq 1)$, $(x_2, |x_1 - x_2| \neq 1)$, $(x_3, |x_1 - x_3| \neq 2)$.

   Domains:

   $x_0 = 1$, $dom(x_1) \in \{0,2,3\}$, $dom(x_2) \in \{0,1,2,3\}$, and $dom(x_3) \in \{0,1,2,3\}$

   The set of variable-constraint pairs:

   $(x_0, x_0 \neq x_2)$, $(x_2, x_0 \neq x_2)$, $(x_0, x_0 \neq x_3)$, $(x_3, x_0 \neq x_3)$, $(x_1, x_1 \neq x_2)$, $(x_2, x_1 \neq x_2)$, $(x_1, x_1 \neq x_3)$, $(x_3, x_1 \neq x_3)$, $(x_2, x_2 \neq x_3)$, $(x_3, x_2 \neq x_3)$, $(x_0, |x_0 - x_1| \neq 1)$, $(x_1, |x_0 - x_1| \neq 1)$, $(x_0, |x_0 - x_2| \neq 2)$, $(x_2, |x_0 - x_2| \neq 2)$, $(x_0, |x_0 - x_3| \neq 3)$, $(x_3, |x_0 - x_3| \neq 3)$, $(x_1, |x_1 - x_2| \neq 1)$, $(x_2, |x_1 - x_2| \neq 1)$, $(x_1, |x_1 - x_3| \neq 2)$, $(x_3, |x_1 - x_3| \neq 2)$, $(x_2, |x_2 - x_3| \neq 1)$, $(x_3, |x_2 - x_3| \neq 1)$, and $(x_0, x_0 \neq x_1)$.

4. **Please add 10 more steps.**

Marking Scheme: (15 marks)

The TA will randomly choose and mark the correctness of 5 steps. Each step is worth 3 marks.

3 Solving 4-Queens as a Local Search Problem (12 marks)

In class, we formulated the 4-queens problem as a local search problem with two different neighbour relations. For this question, choose one neighbour relation for the formulation and answer the questions below.

1. Describe the formulation of the 4-queens problem as a local search problem. (You are allowed to copy the formulation from class.) Be sure to describe the neighbour relation clearly.

   Marking Scheme: (2 marks)

   - No marks for most of the problem formulation. We already discussed it in class.
   - (2 marks) Described the neighbour relation clearly.

2. Based on your chosen formulation, choose one state and write down all of its neighbours.

   Marking Scheme: (2 marks)

   - 2/2 marks if all the neighbours are specified correctly.
   - 1/2 marks at least half of the neighbours are specified correctly.
   - 0/2 marks less than half of the neighbours are specified correctly.

3. Give one state that is a local optimum but not a global optimum. Justify your answer.

   Marking Scheme: (4 marks)

   - (2 marks) Give a state with the required properties.
   - (2 marks) Provide correct justification.

4. Give one state that is neither a local optimum nor a global optimum. Justify your answer.

   Marking Scheme: (4 marks)

   - (2 marks) Give a state with the required properties.
   - (2 marks) Provide correct justification.