CSE5AIF – Artificial Intelligence Fundamentals

2020 Assignment

Due 16th of September 2020

General Information

This assignment is to be done individually and contributes **50%** of your final mark for this subject. The submission date for the assignment is **16th of September 2020**. Submission details are provided below. Make sure that you follow the directions carefully and that the files are named exactly as specified in the instructions.

The assignment is to be done **individually**. This means that answers to questions, and code that you write must be your own. You must not collude with other students in any way, and you must not outsource your work to any third party. For information on plagiarism, see the La Trobe University policy on academic misconduct at http://www.latrobe.edu.au/students/learning/academic-integrity. Plagiarism is treated very seriously. Penalties will be applied and are strictly imposed.

Solving the Towers of Hanoi Problem using State Space Search

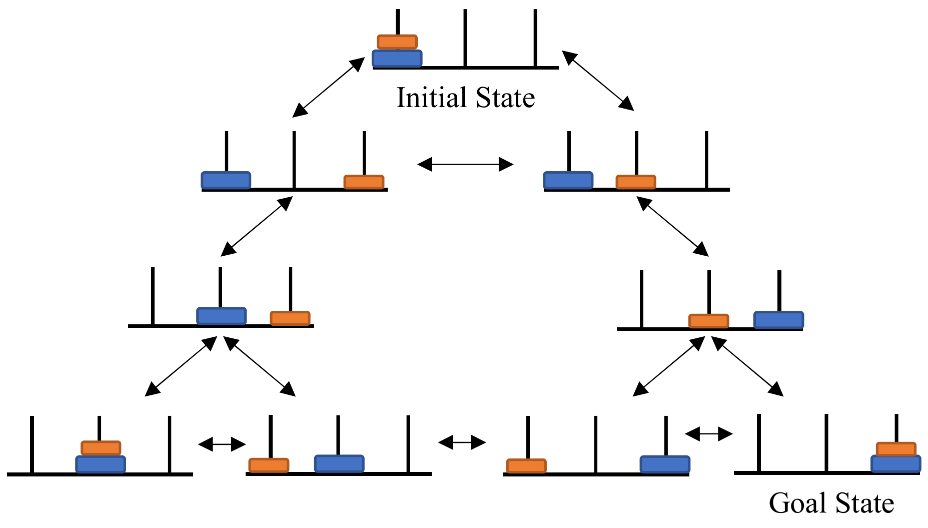
Among many classic AI problems is the Towers of Hanoi problem; one of many versions of this, forms the basis of this assignment and is told as follows:

> *In a monastery in the deepest parts of Tibet there are three crystal columns and 64 golden rings. The rings are different sizes and rest over the columns. At the beginning of time, all of the rings rested on the leftmost column, and since then the monks have toiled ceaselessly trying to perfectly transfer the rings to their resting place on the final column.*

The objective of this problem is to move the entire stack of rings from the first, to the last column, while obeying 3 simple rules:

1. Only one ring can be moved at time.
2. Each move consists of taking the upper ring from one of the stacks and placing it on the top of another stack or an empty pole.
3. A larger ring must not be placed on top of a smaller ring.

A state space representation of a 3 column, 2 ring, Towers of Hanoi problem is shown in the graph:



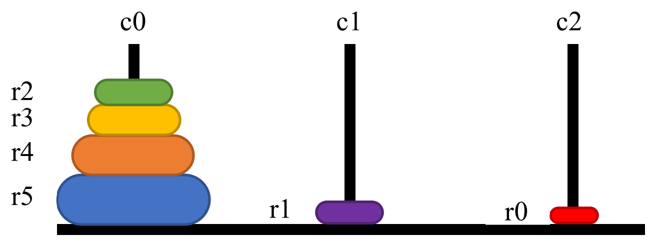
For this assignment, we will consider a Towers of Hanoi problem with 3 columns and 6 rings; although your final algorithm will most likely be capable of solving more complex versions. You will be expected to use the information given in this assignment to code two state space searching algorithms to achieve the objective state described above; an uninformed (Blind) search, and an informed (Heuristic) search.

Part 1 – Problem Analysis

Before writing the python code implementation of this problem, we must perform an analysis in order to conclude the best methodology.

Representing the Problem

How you choose to represent the states of any given problem can be critical in determining whether a solution can be found. Consider the following figure for our problem:



Scheme 1:

A state is represented by a list, where the list is of length 3 to represent the number of columns. Each index in this list contains the list of all rings stacked on this column; the stack order of each ring is represented by the rings index in each columns' inner list.

e.g. Given the problem state in the figure above, our scheme 1 representation is:

$$[\overbrace{[r5, r4, r3, r2]}^{c0}, \overbrace{[r1]}^{c1}, \overbrace{[r0]}^{c2}]$$

$$\text{Bottom} \longrightarrow \text{Top}$$

Scheme 2:

Another approach is to treat the rings individually, defining a rings class to store their size, current column, and index in that column.

Ring Class:

- Number/Size
- Column
- Column Index

Depending on how you decide to implement your future functions, scheme 2 may result in a cleaner, more readable implementation.

Questions:

1. What representation do you think will be best to implement? Why?

2. What is the optimal number of moves for a Towers of Hanoi problem with *n* columns and *m* rings?

3. Write pseudocode to perform the node expansion for this problem. (i.e. in the 8-Puzzle we have the left, right, up, down movement functions)

4. Sketch the first four levels (i.e., root node plus next three levels) of the breadth-first-search search tree. Make sure that you indicate the state corresponding to each node, and the operators that have been applied to move from one state to another. Do not include illegal states, or states that have already been visited.

5. As discussed in Lab 3, the heuristic function to a problem varies based on application. For the 8-Puzzle problem, we utilized a count of all misplaced tiles; for the Shortest Path Problem we calculated the distance to the end node. What heuristic function can be used for the Towers of Hanoi Problem? (hint: the end goal is to have all rings on the final column)

6. As in question 4, Sketch the first four levels (i.e., root node plus next three levels) of the A* search tree.

Part 2 – Coding

As previously discussed, your Towers of Hanoi problem must be capable of solving for **N Columns and R Rings**. The user must enter N and R.

Your final code must implement both Breadth-First Search and A* Search solutions. Similar to your work solving the 8-Puzzle with BFS and A*, both solving algorithms must be implemented into the same file.

The output of your program must be a count of the total number of steps taken to reach the solution; and a path list, showing the rings positions at each level of the search. You need to try different N and R and then create a table (see below) to compare the results between these two algorithms. The compression can be based on the total number of steps taken to reach the solitons and time. You need to run each algorithm 5 times and report the best, worst and average.

E.g.:

```
[5, 4, 3, 2, 1], [0], [0]
[5, 4, 3, 2], [1], [0]
[5, 4, 3], [1], [2]
…
[0], [0], [5, 4, 3, 2, 1, 0]
```

N	R	BFS		A*	
		Steps	Time	Steps	Time
3	6				
..	..				
..	..				
..	..				

You have been supplied with a skeleton Jupyter Notebooks file:

{student_id}_assignment.ipynb

Rename this file to your student id.

Open this file utilizing the Anaconda navigator (refer to the lab material if further instruction on Anaconda is needed).

Uninformed Search

Write Python code which follows your pseudocode and uses the representation you have selected in Part 1 to find a solution using **Breadth-First Search**. Ensure you follow all three rules and output the correct solution.

Informed Search

In the same **{student_id}_assignment.ipynb** file, implement the **A* search** algorithm to solve the problem.

The code in your **{student_id}_assignment.ipynb** file must contain**:**

- A function containing your Breadth-First Search implementation.
- A function containing your A* Search implementation.
- A correct representation of the problem state, indicating all variables required to solve the problem.
- The ability to switch between both search algorithms without major modification to any part of the code.

As well as correctly solving the problem, your code must display good programming style:

- appropriate function and parameter naming;
- appropriate and consistent documentation;
- appropriate use of Classes;
- appropriate and consistent indentation that reflects logical structure of the code.

Marking Criteria

Your solution will be marked according to the following criteria:

- Correct answers to questions from Part 1.
- Python code the finds the correct solution using Breadth-First Search.
- Python code the finds the correct solution using A* Search.

Submission

Submission is electronic only and must be submitted to LMS. You must submit the following 2 files as a **single ZIP file named your STUDENT ID**:

- A word document or pdf, named **Part1** containing your answers to questions in part 1
- A file **aif_assignment.ipynb** that contains all of your python code.

You must include your **name** and **student ID** in both of these files.