UMass CMPSCI 383 (AI) HW1: Chapters 1-3

YOUR NAME HERE

Assigned: Sep 6 2017; Due: Sep 17 2017 @ Midnight

Abstract

Submit a (.zip) file to Moodle containing your latex (.tex) file, rendered pdf, and code. All written HW responses should be done in latex (use sharelatex.com or overleaf.com). All code should be written in Python and should run on the edlab machines ([yourumassid]@elnux[1,2,...].cs.umass.edu).

1 Search Problem Formulation

Consider an ambitious freshman who is interested in pursuing a double major in college. The college has a number of course requirements for completing a double major and this student would like to fulfill the requirements in as few classes as possible.

1.1 Formulation & Search Strategy

Frame this as a search problem. Give the initial state, goal test, available actions, transition model (a.k.a. successor function), and step cost function. A detailed description of these components in words or using pseudocode is acceptable.

Note that your choice of state space affects your set of available actions, which then affects your branching factor (b) and possible solution depth (d). Keep in mind when designing your search problem that many uniformed search algorithms require $\mathcal{O}(b^d)$ space or time complexity, so your formulation can have a significant impact on search. For example, introduce conditions or constraints that can reduce your action space (reduce b). Points will be deducted for naive, wasteful approaches.

Initial State **(1pt)**: YOUR ANSWER HERE.
At the beginning of the semester, the student has not taken any courses.

Goal Test **(1pt)**: YOUR ANSWER HERE.
The student takes minimum courses to full-fill the double major requirements

Available Actions **(6pts)**: YOUR ANSWER HERE.
Chooses courses and takes courses; Todo list on Spire; Finish prerequisite courses; Pay the tuition fee; Get rid of restriction; Make a schedule;

Transition Model **(1pt)**: YOUR ANSWER HERE.
The transition model of "Chooses courses and takes courses" is "Finish courses"

Step Cost **(1pt)**: YOUR ANSWER HERE.
Step cost:1

1.2 Repeated States

Will you need to consider repeated states when performing search? In other words, are there multiple ways to arrive at the same state? Explain. **(2pts)**

If there are multiple courses to take, ignore the course restrictions, if the student take different courses on each semester, eventually, the student will take 8 courses and graduate.
YOUR ANSWER HERE.

1.3 Complexity

As a toy example, consider a 4-year college that

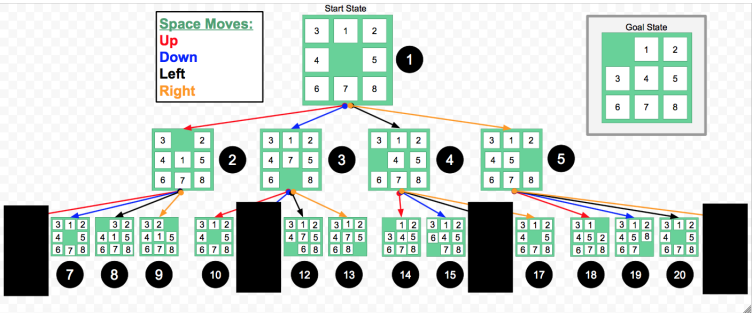

Figure 1: 8-puzzle Search Tree

- offers 10 distinct classes,
- requires a minimum of 8 classes for a double major,
- and requires students take exactly 1 class per semester.

Given your formulation,

- What is the maximum branching factor? Explain. **(2pts)** YOUR ANSWER HERE. The maximum branching factor is 10. The reason is that the student has 10 courses to choose in the first semester.
- What is the depth of the shallowest goal node? Explain. **(2pts)** YOUR ANSWER HERE. Eight. The student is required to take minimum 8 classes to graduate and 1 course per semester.

Consider the search strategies in the Big-O comparison table in Figure 3.21 of the book. Based on your branching factor and depth, is time or memory more of an issue in choosing a search strategy? **(5pts)** Which search algorithm would you use? **(5pts)** Which would you not use? **(4pts)**

YOUR ANSWER HERE.
1, time or memory is not a matter of this problem, because b,d,m and l are all not infinity numbers and less than 10.
2, I would like to choose "Iterative Deepening". The reason is this method uses the lowest time and space based on the strategy that I chose.
3,I would not to use "Breath First". It costs much more space than the other methods.

2 Search on the 8-puzzle

In the tasks below, you will demonstrate various search algorithms on the 8-puzzle domain and then implement A* search in python. To simplify the task, we are providing you with python source code that implements an 8-puzzle simulator as well as Node class. ***Assume nodes are always expanded to generate children in the order: 'Up', 'Down', 'Left', 'Right'.

2.1 Demonstrations

Draw the search tree to a depth of 2 for the 8-puzzle given the start state and `8Puzzle-SearchTree-Template` provided at HWs Public/HW1; make a copy and move the copy to your own Google Drive to edit it **(5pts)**.

See Figure 1.

List the order in which the nodes will be visited (up until the goal node) for

2.1.1 Breadth-first search (10pts),

YOUR ANSWER HERE.
1,2,3,4,5,7,8,9,10,12,13,14

2.1.2 Depth-limited search with depth limit 2 (10pts),

YOUR ANSWER HERE. 1,2,7,8,9,3,10,12,13,4,14

2.1.3 and Iterative deepening search (10pts).

YOUR ANSWER HERE. 1,2,7,8,9,3,10,12,13,4,14

2.2 A* Search

Consider A* with the Manhattan distance heuristic as described in Section 3.6 of the text. Using the same start state as above (Subsection 2.1), fill in the table below with $g(n)$, $h(n)$, and $f(n) = g(n) + h(n)$ and the nodes ordered according to when they were generated by A*. Remember, $g(n)$ is the path cost from start (root) to n, and $h(n)$ is the heuristic function **(5pts)**.

Node	g(n)	h(n)	f(n)
1	0	2	2
2	1	3	4
3	1	3	4
4	1	1	2
5	1	3	4
14	2	0	2
15	2	2	4
17	2	2	4

2.2.1 Implementation

Implement A* with the Manhattan distance heuristic in Python 3.5.2. As a sanity check for your code, make sure your implementation takes the correct steps for the search problem above (Subsection 2.1) **(5pts)**. As an additional check, make sure your implementation returns a solution with 26 steps for the start state given in Figure 3.28 of the text **(10pts)**.

Using the start state given in Figure 3.28 of the text, generate 100 random start states (use the `Puzzle.shuffle` method). For each start state, compute an upper bound for the effective branching factor and calculate the average branching factor over the 100 start states **(15pts)**. The branching factor should be strictly less than 1.48, which is approximately the branching factor for the *misplaced tiles* heuristic (h_1) on 26-step puzzles (see Figure 3.29).

We have provided template python code, `my_hw1.py`, with input-output signatures. Fill in the missing code and verify your implementation produces the correct outputs to achieve full credit (5+10+15=30pts). You will need to make use of the `Puzzle` class we have provided. You can read the doc string for the class with `help(Puzzle)`. Also, you may find the queue package helpful.