CMPT 310 Assignment 1

October 4, 2017

100 points total, worth 10% of the course grade. Turn in on CourSys. Submit a compressed directory (`.zip` or `.tar.gz`) with your solutions. Code should be submitted as a single `.py` file, one file per question. Free-answer questions should be submitted as PDF file. You may write your solution using LaTeX, Microsoft Word (print to PDF, do not submit a `.doc` file), or write on paper and scan it. If you write on paper, please use a scanner or a scanner app (such as TinyScanner for iOS) that removes the background; do not submit a photo.

You are encouraged to work in a group. Feel free to discuss solution strategies and check each other's work. However, you must write all the text and code you submit on your own. Joint submissions are not allowed, nor is copying someone elses text or code. Plagiarism is not okay, and will be taken very seriously. If you're not sure whether something is okay, please ask.

If you are having trouble, please take advantage of office hours and the discussion forum. If you see a question from another student on the discussion forum that you think you know the answer to, please respond. (Do your best to lead your fellow student to a solution, rather than simply giving the solution directly.)

1 PEAS model

(10 points) For each of the following problems, develop a PEAS model and characterize the environment along the dimensions: observable, deterministic, episodic, static, discrete, single-agent.

1. Google Maps search for directions using public transit in Vancouver.
2. The game show Jeopardy, as played by IBM's Watson.

2 Problem formulation

(10 points) Give a complete problem formulation for each of the following. Choose a formulation that is precise enough to be implemented.

1. Using only four colors, you have to color a planar map in such a way that no two adjacent regions have the same color.

2. You have three jugs, measuring 12 gallons, 8 gallons, and 3 gallons, and a water faucet. You can fill the jugs up or empty them out from on to another or onto the ground. You need to measure out exactly one gallon.

3 Searching using composite actions

(10 points) An action such as *Go(Sibiu)* really consists of a long sequence of finer-grained actions: turn on the car, release the brake, accelerate forward, etc. Having composite actions of this kind reduces the number of steps in a solution sequence, thereby reducing the search time. Suppose we take this to the logical extreme, by making super-composite actions out of every possible sequence of *Go* actions. Then every problem instance is solved by a single super-composite action, such as *Go(Sibiu)Go(Rimnicu Vilcea)Go(Pitesti)Go(Bucharest)*. Explain how search would work in this formulation. Is this a practical approach for speeding up problem solving?

4 Lights Out puzzle

(60 points total) In this question we will consider the Lights Out puzzle. In this puzzle, there is an $n \times m$ board of cells, each of which is On or Off. The puzzle starts out with some cells On and some Off. In each move, the player chooses one cell; that cell and all adjacent cells are toggled (i.e. On turns to Off, and Off turns to On). Cells count as adjacent horizontally or vertically, but not diagonally. In other words, targeting a cell in the center of the board toggles a total of five cells; a move on the side toggles four; and a

move in the corner toggles three. The goal of the game is to change all cells to the Off position in as few moves as possible.

You can try playing Lights Out yourself at:
`http://www.neok12.com/games/lights-out/lights-out.htm`

4.1

(5 points) Create an implementation of the Lights Out puzzle. It should support the following functions:

- `create_puzzle(rows, cols)`: Create a Lights Out puzzle with all cells initialized to off.
- `perform_move(puzzle, row, col)`: Toggle the appropriate cells corresponding to a move at position (row,col).
- `scramble(puzzle)`: Scramble the puzzle by, on each cell, randomly with 50% probability, calling `perform_move()` . This ensures that the puzzle is solvable, which might not be the case if cells are flipped individually.
- `is_solved(puzzle)`: Returns True or False depending on whether all cells are off.

4.2

(10 points) Does the order of moves for a Lights Out puzzle matter? Can an optimal solution ever involve running `perform_move()` twice on the same cell? Explain why or why not.

Write a function `next_cell(rows,cols,row,col)` that takes the (row,col) coordinates of a cell on a rows $\times$ cols grid and outputs the (row,col) coordinates of the next cell in left-right top-bottom order. For example, in a 5×5, zero-indexed grid, `next_cell(5,5,0,0)` = (0,1) and `next_cell(5,5,3,4)` = (4,0). (Any other order is fine, too.)

4.3

(10 points) Come up with a nontrivial admissible heuristic for this puzzle. Describe your heuristic and explain why it is admissible.

4.4

(30 points) Implement three functions to solve the Lights Out puzzle, using each of the following algorithms respectively: Depth-First, Breadth-First, A*. A* should use the heuristic you designed above. Each function should take a puzzle as input and return a list of moves that solves the puzzle. In addition, each function should output the number of steps it took to find the solution, measured by the number of times the search algorithm used `perform_move()` to find the next board state. (You only need to output a single solution (i.e. one list of moves), not all possible solutions.)

Hint: Your implementations of Depth-First and Breadth-First search should have worst-case running time of $O(2^{nm})$, where n and m are the numbers of rows and columns respectively. You may want to make use of `next_cell()`.

4.5

(5 points) Create 100 random puzzles with 4 rows and 4 columns, and run each of the three algorithms on each puzzle. For each algorithm, report (1) the total length of the 100 solutions, (2) the total number of steps taken by the search algorithm to generate all the solutions, and (3) the wallclock time taken by the algorithm.

Hint: This should take less than 5 minutes to run.

4.6

(Challenge problem: +10 points extra credit). Come up with your own algorithms for searching for a solution. You may use A* with any heuristic you like, rely on precomputed sub-problems, or use any other algorithm you can think of. Run your algorithm on 100 $n \times n$ puzzles and report moves, search steps and wallclock time, as above. We will post a leaderboard on the course web page with the top solutions for $n = 5$ and $n = 8$, and award extra credit for solutions that perform particularly well on either measure.

If you decide to use an algorithm that does not run `perform_move()` directly as part of its search, it may not be obvious how to measure how many steps it takes. If that is the case, use your judgement and measure the number of steps in whatever way is most appropriate. In particular, every "step" should take $O(1)$ time: the running time of a step should not depend on the number of rows and columns of the puzzle.

If you like, you can do any amount of pre-computation, such as storing the results of puzzles with partial solutions. However, your algorithm must do the pre-computation before seeing any of the 100 test puzzles.

There will likely be some variation in results due to random chance; however, this will likely be small because of the large number of puzzles. You are allowed to repeat the 100-puzzle process as many times as you like and report the best results.

We are counting on everyone to report their results faithfully. This is a friendly competition; please do not cheat and report different numbers than you got. We may verify specific results if we suspect the numbers are mis-reported.

5 Feedback

(5 points) Give one short piece of feedback about the course so far. What have you found most interesting? Is there a topic that you had trouble understanding? Are there any changes that could improve the value of the course to you?

6 Time

(5 points) How many hours did you spend on this assignment?