CPSC 470 – Artificial Intelligence
Problem Set #5 – Decision Trees and Genetic Algorithms
30 points
Due Friday April 2nd, 10:30 AM

Some reminders:

- **Grading contact:** Nick Georgiou (nicholas.georgiou@yale.edu) is the point of contact for initial questions about grading for this problem set.
- **Late assignments** are not accepted without a Dean's excuse.
- **Collaboration policy:** You are encouraged to discuss assignments with the course staff and with other students. However, you are required to implement and write any assignment on your own. This includes both pencil-and-paper and coding exercises. You are not permitted to copy, in whole or in part, any written assignment or program as part of this course. You are not to take code from any online repository or web source. You will not allow your own work to be copied. Homework assignments are your individual responsibility, and plagiarism will not be tolerated.
- **Students taking CPSC570:** There is no extra section for this assignment. Your assignment is the same as CPSC470.

Problem #1 : Decision Trees (6 Points)

Given the dataset below that consists of five examples (X_1 through X_5) for three Boolean attributes (A, B, and C) and a Boolean outcome, you are to assemble (by hand) an optimal decision tree for this data.

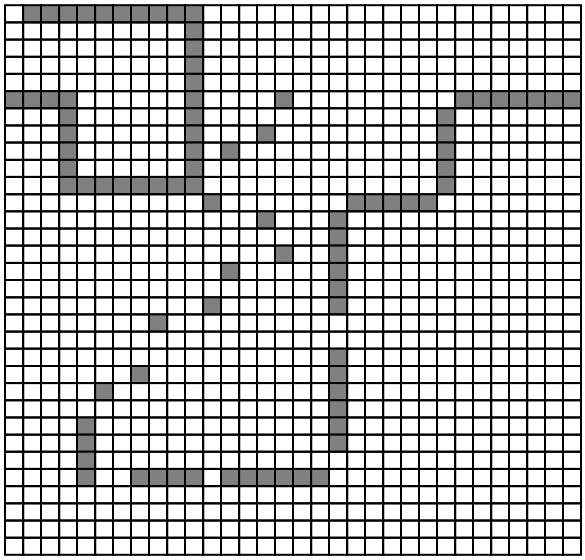
	A	**B**	**C**	**Outcome**
X_1	T	T	F	T
X_2	F	T	F	F
X_3	F	T	T	T
X_4	T	T	T	T
X_5	F	F	T	F

Your solution should show all calculations, starting with a comparison between using each of the three attributes (A, B, C) as the root of the tree. You should show the Remainder values for each comparison, and show the final decision tree that represents the optimal decision tree.

Please submit your solution to Gradescope. Your solution will be scored based on the accuracy of your calculations and the clarity of your explanations.

Problem #2: Genetic Algorithms (24 Points)

In this problem, we will evolve a controller for a simulated ant. Each ant must survive on its own in a world represented by a 2D grid of cells by following trails of food. Each cell in the world either has a piece of food or is empty and the cells wrap-around (so, moving up when in the top row leaves the ant in the bottom row of the grid). Shown below is an environment (called the "John Muir" trail) that consists of a 32 by 32 grid containing 89 food cells (shown in gray).



The ant's position at any point in time can be specified by a cell location and a heading (north, south, east, or west). The ant always starts in the cell in the upper left corner, facing right (east). At the beginning of each time-step it gets one bit of sensory information: whether there is food in the cell in front of the cell it currently occupies (i.e., the cell it would move to if it moved forward). At each time-step it has one of four possible actions. It can move forward one cell; rotate clockwise ninety degrees without changing cells; rotate counter-clockwise ninety degrees without changing cells; or do nothing. If an ant moves onto a food-cell, it consumes the food and the food disappears; when the ant leaves that cell, the cell is empty. The fitness of the ant is rated by counting how many food elements it consumes in 200 time-steps. (An ant that consumes 10 cells worth of food total in 200 time-steps receives a fitness score of 10.)

The controller for our ant will consist of 10 states. At each time step, the ant takes the following actions:

1. Read the sensor value.
2. The controller changes state based on the sensor value.
3. The ant takes an action indicated by the new state (which may result in a change in position).
4. If the ant is in a cell with food, the ant eats the food.

Each of the ten states has a unique identifier (a number ranging from 0 to 9) and the content of that state can be represented by three digits:

Digit #	Range	Meaning
1	1-4	The action that the ant takes upon entering this state, where 1 = move forward one cell 2 = rotate clockwise ninety degrees without changing cells 3 = rotate counter-clockwise ninety degrees without changing cells 4 = do nothing
2	0-9	If the ant is in this state and the sensor value is **false** (there is no food in the square ahead of it), then the ant will transition to the state with the unique identifier indicated by this digit.
3	0-9	If the ant is in this state and the sensor value is **true** (there is food in the square ahead of it), then the ant will transition to the state with the unique identifier indicated by this digit.

The ant begins its life with the controller in state 0. The entire genetic material for an ant thus consists of 10 states, each of which is represented by 3 digits, for a total of 30 digits.

Your task is to construct a genetic algorithm that attempts to build a better ant through evolution. Your algorithm should make use of multi-point crossover and mutation. In each generation, you should test the fitness of each ant (individually) on the Muir trail. Begin with an initial population of at least 10 ants and run your algorithm for at least 40 generations.

The starter code package provides you with the following files:

- geneticAlgorithm.py: the starter code.
    - It already implements the following functions for you (Please refer to the source for documentation):
        - ant_simulator: This is an ant simulator. It takes the food_map, map_size and ant_genes as parameters, and output the corresponding trial and the fitness.
        - get_map: It takes in the map file_name and return the food_map and map_size to be used in ant_simulator.
        - display_trials: It takes in the trial generated from ant_simulator, and the target_file name, and saves the trial in the target_file.
        - There is an example of how to use the functions implemented for you at the bottom.
    - You will need to implement the following functions (Please feel free to change the parameters and what to return to fit your implementation. The parameters mentioned above are just suggestions):
        - genetic_algorithm: This is the main function of the genetic algorithm. It takes in the population, the file name of the food_map, the maximum number of generations, the crossover probability, the mutation probability. It returns the maximum fitness in the last generation, the individual (gene) with the maximum fitness in the

last generation, the trial of the individual with the maximum fitness in the last generation, the overall statistics of all the generations (it is a list of [maximum fitness, minimum fitness, average fitness] of each generation), and the population in the last generation.
    - initialize_population: It takes in the number of population to be initialized and return the population (a list of individual/genes).
    - select: It selects the individuals as parents for the next generation.
    - crossover: It takes a population, and performs crossover based on the crossover probability, and returns the population generated.
    - mutation: It takes the genetic material and performs mutation based on mutation probability, and returns the resulting genetic material.
- Use the results generated from the genetic_algorithm function to output the figures and results needed to answer the questions required. You can use python's matplotlib library, which has already been installed in zoo. If you wish to install the library on your own computer, please feel free to search online, but we won't be able to provide extra help on the library installation. Please also feel free to generate graphs in other ways, or modify the any parts of the code you like as long as your code can generate the questions required below.
- muir.txt: One of the maps you will need.
- santefe.txt: Another map you will need.
- trial.txt: The trial generated if you run the starter code without modification.

You will need to make many design decisions on how to implement the algorithm and what parameter values to use**.** The main section of the starter code already provides some hints of what parameters to use. Please feel free to add more if necessary. **Your score on this problem will depend not only on the code that you write but also on how well you document your design decisions.**

Some notes:

- You should not be running evolutions that take more than 30 minutes of compute time. If you are, you are doing either something unnecessary (or more likely) incorrect.
- No part of your score for this portion is based on the success of your individual ants. GA's include a high degree of randomness… sometimes, you're just going to be unlucky.
- Your score will be based on the accuracy and completeness of the code that you submit, the detail and comprehensiveness of your documentation, and your answers to the required questions asked below.

With the write-up for your solution, you should answer the following questions (Please refer to the submission guidelines below for more information):

1) Fitness (3 points)

Question 1.1: What is the fitness score of most-fit individual in the first generation on the Muir trail?
Question 1.2: What was the fitness score of the most-fit individual in the last generation?
Question 1.3: Please plot the fitness score of the most-fit individual in each generation.

2) Please take a screenshot of the Path/trial of the most-fit individual in the final generation on the Muir trail. (2 points)

3) Trials comparison. (4 points)
Question 3.1: Please plot the individuals in the final generation and compare their performances on the Muir trail with their performances on the Santa Fe trail.
Question 3.2: Do individuals that do well on one trail tend to do well on the other, and why?

4) Your Implementation (15 points)

Question 4.1: Please provide a brief description of your genetic algorithm:
Question 4.2: Please document the parameters you chose (please feel free to add more parameters in the empty cells):

Size of the population	
Number of generation	
Crossover probability	
Mutation probability	

Question 4.3: Please submit your python file.

Submission Instructions:

- Please submit all files on Gradescope.
- For programming parts (Question 4.3), please submit to Problem Set 5 – Programming
- For solutions to other parts, please submit to Problem Set 5