CSCI 3202 – Introduction to Artificial Intelligence
Instructor: Hoenigman
Assignment 5
Due: Friday, October 14, 4pm.

Can a Political Election be Rigged?

The drawing of political district boundaries is a contentious issue. It's been demonstrated repeatedly that boundaries can be drawn to give favor to one party, and that the same population divided up differently can result in a different outcome in an election. We can view a political election as a zero-sum game, where two parties (players), court the votes of individuals in swing districts or states, and there can be only one winner in the election. Both parties know which districts are contentious (they have perfect information), but they don't know how independent voters will ultimately vote, or what voter turnout will be on election day.

In this assignment, you're applying one of two algorithms - Simulated Annealing or a Genetic Algorithm, to partition a fictional state into fair districts populated by two political parties, Rabbits and Dragons. You need to design a fitness function that captures the necessary features in the system, and write a program that implements your algorithm and fitness function. The result of your program is a partitioning of the state into a set of districts.

There is a paper posted on Moodle called *A Simulated Annealing Approach to Police District Design* that describes a process similar to what you are asked to do in this assignment. The requirements on valid solutions are more rigorous than the requirements in this assignment, but the overall approach is the same. If you are stuck on how to begin this assignment, I recommend that you read that paper.

Input Data

There are two files on Moodle, one is called *smallState* and the other is called *largeState*. These files are the simulated census data that your program needs to use in your algorithm. The files each contain an *n x m* matrix of the predominant voting party in each block in the neighborhood. For example, the top left block of *smallState* has a value of "D", signifying that voters in that cell vote for the Dragon party. Your program will need to work on both files.

For *smallState*, your game should divide the neighborhood into eight districts of eight blocks each. For *largeState*, your game should divide the neighborhood into 10 districts of 10 blocks each.

All districts need to be contiguous blocks, meaning that every block needs to be connected to at least one block from the same district, either horizontally, vertically, or diagonally.

Algorithms

For this assignment, you need to implement either simulated annealing or a genetic algorithm. **You do not need to implement both.**

Simulated Annealing

Your simulated annealing (SA) implementation should follow the pseudocode provided in the lecture notes. Once the algorithm is implemented, explore the results using different values for k, T, and alpha.

Genetic Algorithm

Your genetic algorithm (GA) needs to handle:

- Defining the chromosome for the 2D matrix.
- Setting a crossover point on the chromosome.
- Fixing solutions that are invalid, or discarding them.
- Implementing a mutation probability and location.

Fitness function

Your SA or GA algorithm needs a fitness function. Several options for fitness functions have been presented in class, including functions that calculate the difference between the majority districts and functions that also consider the political party composition within the district.

Finding neighboring solutions

One of the biggest challenges of this assignment will be determining neighboring cells to existing districts. You will need an efficient method in your SA implementation for determining how districts can be modified to change the political makeup of each district while still satisfying the constraint that districts need to be contiguous. If you are implementing a GA, you will also need to identify valid and invalid districts.

Report

Along with your implementation, you need to submit a short report describing your algorithms, fitness function, and the results of your program.

Purpose:

Describe the purpose of the program you implemented and what you want to accomplish.

Procedure:

Your procedure section needs to include the following sub-sections:

Fitness function

Describe the fitness function and show a few examples of how the fitness function works, including specific inputs and outputs.

Neighbor detection
Describe the algorithm and data structures you used to detect cells at the edge of each district.

Generating new candidate solutions
Describe how you generated new solutions for evaluation in your SA or GA algorithms. Describe the neighbor detection algorithm and how you determined valid solutions for SA. For the GA, this means describing the crossover point and how you determined whether solutions were valid.

Data:
Describe the data you used in your program. What was the data format? What was the percentage of each political party in the data?

Results:
Describe the results of running your program. What were the results of using different values for the parameters in your search algorithm? Describe the solutions that your algorithm produced. How "good" were the solutions? If you run your algorithm multiple times, do you can the same solution each time? How many unique search states were explored?

Running your program and program output

The grader will run your program in python 2. Your main file should be called Assignment5.py and take one command line arguments - the name of the filename to use.

```
>>python Assignment5.py smallState.txt
>>python Assignment5.py largeState.txt
```

Program Output:
When the grader runs your program, it should produce a display of the following information.

```
Party division in population:
************************************
R: <% R>
D: <% D>
************************************

Number of districts with a majority for each party:
************************************
R: <number>
D: <number>
************************************
```

Locations assigned to each district:
****
District 1: (1,1), (1,2), (1,3)
District 2: (3,1), (3,2), (3,3)
.
.
.
****

****
Algorithm applied: <SA or GA>
****

****
Number of search states explored: <number of states>
****

Handling infeasible states:

Your program will need to handle infeasible states (states that violate the constraint that the districts be contiguous.)

As an example of an infeasible state, consider the following 9-block neighborhood that is labeled as Player-Block Index.

R-1	D-2	R-3
D-4	D-5	R-6
D-7	R-8	R-9

A feasible state of three districts is (1, 2, 3), (4, 5, 7), (6, 8, 9). All districts contain contiguous blocks and the same number of blocks. An infeasible state is (2, 4, 5), (1, 7, 8), (3, 6, 9) because the (1, 7, 8) district is not contiguous.

Extra credit:

There are two extra credit options. Each option is worth 10 additional points on the grade for this assignment.

Implement both SA and GA and compare the results.

You are only required to implement one algorithm. For extra credit, you could implement both algorithms and compare the results in your report. Both algorithms need to work to get credit.

Voter turnout or independent affiliation

So far, we have assumed that voter turnout will be 100% and all blocks were equivalent in regards to number of voters. However, in the real world this is not the case. Voter turnout is often influenced by voter registration campaigns and whether

an election will be a close race. In many districts, parties will target the small number of registered independent voters to gain a slim majority over their opponent.

Design a fitness function that considers voter turnout in assigning the district to a particular party. Clearly state your assumptions about voter turnout in the output of your program and in your report write-up.