Soham Shah
Assignment 5 – Simulated Annealing
CSCI 3202
Rhonda Hoenigman

Assignment 5 Report

Purpose:

The goal of this project is to try and determine a way to district a neighborhood given the knowledge of each household's political affiliations. The reason for using a computer to solve this problem is to avoid the issue of Gerrymandering by trying to create districts which will result in representatives that closely resemble the popular vote. The restrictions are that all districts must have houses that are contiguous.

Procedure:

Fitness Function:

The fitness function that I use in our simulated annealing algorithm and which forms the base of new candidate solutions is modeled as follows:

First, I calculate the percentage of the popular vote that D receives:

$$percentage\ of\ popular\ vote\ for\ D = \frac{number\ of\ D's}{total\ population}$$

Then, I calculate the percentage of vote for D in the candidate solution:

$$candidate\ solution\ percentage\ for\ D = \frac{number\ of\ D's}{total\ population}$$

Finally, I take the percentage difference between the two to get fitness:

$$F = \frac{|popularVoteD - candidateD|}{popularVoteD}$$

Neighbor Detection:

I used a dictionary to store the districts with the value for each district being an array of tuples that represent the nodes in the district ad (x,y) coordinates. To figure out the neighbors, all that needed to be done was iterate through this array and then get the potential neighbors by calculating (+1,0), (-1,0), (0, +1), (0,-1) on each of the nodes in the district and then appending them to an array of potential neighbors. Diagonal neighbors are not considered contiguous and so are not part of the solution set.

Generating New Candidate Solutions

Generating new candidate solutions was done by adding an element of randomness in switching neighbors between the districts. Each iteration, the

algorithm would randomly swap x nodes in the districts. Then this new district map would be checked to see if it was valid. If so the new district map was compared to the previous on to see if it was more fit.

Data:

The data that was used was in the form of a text file with an NxN array of party affiliations. The data has to be a square array because that's how the algorithm was designed. Each node on the array either had an "R" or a "D" indicating the two different types of parties. Here's an example of what a sample test file would look like:

D R D R
D D R D
D D D R
D D R R

Results:

The Algorithm is able to efficiently find a solution that closely represents the popular vote each time. Each time it runs it is able to generate a new potential valid district thanks to the random element in solution generation. There are three variables that can be tuned to adjust how quickly the algorithm converges to a solution. That's T, Tmin, and k. The first is T which is the initial starting temperature. This is currently set to sys.intmax so that the algorithm runs for a good number of times. Then Tmin is when the algorithm should stop running. It's set to .000001. I found this to be a good number for running a good number of calculations without running into overflow issues. Finally k is how much the T value is going to decrease by each loop. It's currently set to 0.99. This means that T decreases by 1% each loop. Based on this, there are currently 5,721 search states being explored each time the algorithm is run. This is good because it keeps the program runtime low while also figuring out a good candidate solution each run. Overall, the solutions generated have been very good. It closely mirrors the popular vote while also keeping computation times down.