Paul Laliberte

10/16/16

CSCI3202

Assignment5 Write-up

**Purpose:**

The algorithm I implemented was the genetic algorithm. The purpose of the genetic algorithm is to find the optimal (or optimize the) solution. In this assignment I created boundaries for political redistricting. The algorithm successfully creates ``fair’’ boundaries based off a fitness function (grade function in file).

**Procedure:**

* Fitness Function: For my fitness function I took into account the number of R’s and D’s in each district in comparison with the total percent of population of R and D (-1,+1 if fell between 10% in pos. or neg. direction) and also if districts were contiguous (+50 for a contiguous district). Our goal was to get as high of a score as possible (in comparison with trying to converge to 0), i.e. the most fit is the highest score. Using the highest score allowed for the fitness calculations to be more relaxed, in that I did not have to worry about negative values, or many values converging close to zero. An example would be given a district of a population, A = [(1,1),(3,2),(4,5),(2,1)], where A can be represented as R’s and D’s, A = [D, D, R, D]. We have 3 D’s and 1 R, hence precent\_R = ¼ and percent D = ¾. Suppose that the population was 75% D and 25% R. Then we would get a +1 for each D and R percent. The points are not contiguous, so they do not get the added +50. I then added up all of the scores (2, in this case). This is the score for a given solution. I gave such a large added score for contiguous solution because a solution that is not contiguous is essentially not an actual solution. We iterate through all populations and record the fitness scores of each.
* Neighbors: We start off with a gene, and randomly pick a random point in the gene. For finding neighbors I used a ``find\_neighbors’’ function. This function returned all of the possible neighbors for the random point. Then I would fill in my solution with the given possible neighbors. For example, say we have point (1,1). Then the function returns an array of [(0,0),(1,0),(0,1), … , (1,1)]. We then compare the elements of the neighbor array with our initial ``gene.’’ This is a safety precaution to make sure we do not add a point that is not possible, i.e. (0,0). We remove the neighbors and the random point from the gene array, pick another point, and continue for the specified number of districts. After this we will still have some elements in the gene. Hence, we iterate through the gene, find neighbors of each iteration, and append the point when we have found a neighbor in one of the districts (this ensures contiguous solutions). We now have a complete solution.
* Generating New Candidate Solutions: For the crossover point in my GA, I took the two highest graded solutions and made the first solution into a gene, we call it sG (using gene\_function). Then we picked a random spot in sG and sliced out 3 points. We then inserted these 3 points into the smallest district of solution two (trying to counter-balancing have a district with the majority of points), after deleting the 3 points first. We now have a new solution, and hopefully one that leads to higher scores as we continue to evolve. We also had a mutate function that has a manually set probability of running.

**Data:** The data was in the format of R or D nodes. I used an adjacency matrix to represent the data, with a dictionary (hash table) to keep track of which values were D’s and which were R’s.

For the first smallState set the percentage was 50% R and 50% D. For the largeState the percentage was 52% R and 48% D.

**Results:** The results for the most part were accurate. The smaller state text produced some generally good solutions, where as the larger state produced good solutions also, but once and awhile there would be a district that is boxed off, i.e. a district with only two points. Running my algorithm multiple times will produce different solutions, but running it enough there are likely to be duplicate solutions eventually, or at least duplicate districts. To build on my algorithm would be to create a more precise fitness function (grade\_function), that would rigorously analysis solutions, and separate the truly excellent ones from the weaker, and then evolve (crossover).