Peter McPadden

AI Assignment 5

10/16/2016

Purpose: Gain practical experience implementing simulated annealing

Procedure:

Fitness Function: If districts are not contiguous or contain any number of nodes other than the allowed size (districts were allowed to be the length of each row) the graph was not scored. Otherwise the fitness function was very simple. Score = abs(1/(proportionOfRByPop – proportionOfRByDist)). For very similar proportions this would result in a large score, for very different proportions it would result in a small score.

Neighbor Detection: Cells at a district edge were detected by checking neighbors. If all neighbors were part of the same district then the cell was not an edge, otherwise it was.

Generating New Candidate Solutions: Every time a new solution was generated we started with a random edge node from district 1. That node would then be switched to the district of one of its external neighbors. So if district 2 was adjacent to our first node, that node would be switched to district 2. District 1 would have (assuming starting with 10 nodes per district) 9 nodes, district 2 would have 11. Then we would look at the edges of district 2. It could choose 1, and wouldn't involve much change, or it could pick something like 3 (which for our purposes will be adjacent to 1). The node in 2 then becomes part of 3, so now 2 has 10, and 3 has 11. Now we look at the edges of district 3, and set a district 3 node to district 1. Since we started at 1 and know it is the first, then we at least know that all districts now have 10 nodes. To check validity of this I wrote a modified Depth first search that picks a random node in each district, does DFS from there, and then does a count of nodes that were visited. If this number is different from the district size, it means the district is not contiguous. The validity check also double checks that all districts have the appropriate amount of nodes.

Data: The data used in the program were largeState.txt and smallState.txt. Small state was equally R and D, and the large state was 52 percent R and 48 percent D. The format for the data was made up of a square of R's and D's either 8x8 or 10x10 separated by spaces.

Results: Increasing T and decreasing Tmin resulted in more search states being explored, similarly increasing alpha resulted in more iterations. Changing k to higher values tended to

result in better numbers. When k was tiny I got 5 R districts and 3 D districts, at higher values I ended up with the closer to ideal 5 R and 4 D. This is because we end up raising e to the –(de/kT) and when k is bigger we are less likely to accept worse results. All of the solutions I investigated were valid, the districts did snake around quite a bit. The small state results were consistently 50/50 usually with 2 tied districts, and 3 R districts and 3 D districts. I have not seen the same output twice, but they solutions are all reasonably close to the population, and I've never seen something really out of proportion like D districts outnumbering R districts in the 30 or more times I've run this. With T max at 10000 and Tmin at .00001 and alpha at 0.95 it checks 405 different states in about 8 seconds, with alpha at .99 it goes through 2062 states in about 30 seconds when it doesn't run into recursion depth errors.