Kevin Rau

101289616

CSCI - 3202

Assignment 5

http://katrinaeg.com/simulated-annealing.html (I used this basic outline a lot with helping develop this code.

Fitness Function:

```
def acceptanceProbablity(new, old, t):
    ap = math.exp((old - new) / t)
    return ap
```

```
def fitnessFunction(districts):
    #k, your call here to pivot back to simulatedAnnealing
    d_majority = 0
    r_majority = 0
    equal = 0

    for i in range(len(districts)):
        d_count = 0
        r_count = 0
        for j in range(len(districts)):
            if districts[i][j] == "D":
                d_count = d_count + 1
            else:
                r_count = r_count + 1
        if d_count < r_count:
            r_majority = r_majority + 1
        elif d_count > r_count:
            d_majority = d_majority + 1
        else:
            equal = equal + 1
    return abs(d_majority - r_majority)
```

```
def anneal(solution):
    old_cost = cost(solution)
    T = 1.0
    T_min = 0.00001
    alpha = 0.9
    while T > T_min:
        i = 1
        while i <= 100:
            new_solution = neighbor(solution)
            new_cost = cost(new_solution)
            ap = acceptance_probability(old_cost, new_cost, T)
            if ap > random():
                solution = new_solution
                old_cost = new_cost
            i += 1
        T = T*alpha
    return solution, old_cost
```

The fitness function that I used takes in the values of the districts that are created by the ‘generateSolution()’ function. Above is a basic outline of what happens. Fitness is going to take in the individual districts and measure their majority. Returned value is the difference between D and R with 0 being the most optimal return value.

Neighbor Detection:

This doesn't work as well as I had hoped it would but what happens is the the text file is translated into a 2 d array that is then broken up into separate districts based on the size of the text file. The choices of what elements are added are based on the next consecutive element in the array. This however leads to some neighborhoods breaking the contiguous requirement when dropping into a new row.

Generating New Solutions:

This is done by calling the simulatedAnnealing portion of the code. Like how we had discussed in class with this function, we are going to make a call to 'generateSolution()', make a cost, and test that against an old cost of the previously picked solution. Solutions though were invalid sometimes and unfortunately this couldn't' be resolved in time.

Data:

Data was determined by the text file that was then translated into a 2-D array. Percentages form each political party seemed to average out to around 50% each and this did depend on the amount of iterations completed in simulatedAnnealing().

Results:

For one, the larger text file seemed to give results back that were less evenly spread as far as distribution goes for majorities in districts. Iterations also affected this, mostly because there are clearly less solutions to choose from. If I ran this multiple times there were different solutions and sometimes different percentages, but for the most part every solution was evenly distributed.