ECEN 521
11/10/15
Matthew Wyman
Matthew James
Connor Smith

Design Problem #3: Darius Demolishes Dissent

This design problem is similar to the Travelling Salesman Problem, which is a classical combinatorial optimization problem. Since TSP is NP-hard, there is no known algorithm that will find the optimal solution in polynomial time. In light of this, we decided to use a probabilistic approach based on simulated annealing to find solutions that are "close enough" to the optimal ones.

However, the problem as given is not exactly TSP. In the traditional form, TSP is modeled using a complete graph, whereas in this design problem the graph modeling the cost of movement between cities is quite sparse. This complicates the problem even more, since you cannot just permute the order of visiting cities arbitrarily, as this would likely generate one or more pairs of adjacent cities that do not have a direct path between them. To account for this, we designed a complex cost function that added a very high cost for including edges that weren't actually present in the given graph. This yielded terrible results, however, and eventually we decided to take the advice from Dr. Skiena to "design graphs, not algorithms."

We were able to reduce this problem to the TSP by turning the sparse graph into a dense intermediate graph. This intermediate graph contained the shortest distance between any two vertices, which was computed using the Floyd-Warshall algorithm. We then used this graph instead of the original with simulated annealing, and generated the order of unique cities that would be visited. After finding the best solution over many iterations, we converted this intermediate list into our final result by using the shortest path between each pair of cities in our list. These paths were retrieved from an auxiliary matrix that was computed when we ran the Floyd-Warshall algorithm.

We definitely found this problem to be the most difficult so far, since there is no way to brute-force the optimal solution in a reasonable amount of time. We spent the most time trying to come up with a complicated cost function to use with simulated annealing that would generate low-cost, accurate paths, but this approach was largely a failure. After running for nearly an hour on the smallest example input, the best solution we found was over 5x longer than the optimal one. Once we converted the graph to the intermediate representation as described above, however, we were able to solve the rest of the problem in very little time, and it yields much better results.

A table of our algorithm's performance is shown below. The execution time doesn't vary too much with the size of the input, since computing solutions is simply a matter of swapping two cities. The time to run does vary immensely with the number of iterations, however. Choosing a small number of iterations will allow relatively good solutions to be generated very quickly, but if waiting a couple minutes for a solution is acceptable, more optimized solutions can be generated. The table shows the results of running for 100, 1000, and 10000 iterations per temperature value. As can be seen, the execution time increases linearly with the number of iterations, but there are diminishing returns. Running for 100x more iterations (from 100 to 10,000) resulted in times within 10% of each other. Depending on the application, this added time may or may not be worth it.

	Iterations = 100		**Iterations = 1000**		**Iterations = 10000**	
	Time (s)	**Distance**	**Time (s)**	**Distance**	**Time (s)**	**Distance**
map8	0.847082	53	8.400635	53	84.48313	53
map10	0.93975	51	8.76748	54	87.759915	51
map13	0.943842	60	9.413797	58	94.032242	55
map16	0.987252	58	10.243369	58	98.689516	58
map19	1.047688	75	10.484279	75	104.415834	70
map22	1.098319	87	10.956893	86	109.360421	86
map25	1.163957	98	11.597512	98	116.70431	94