Travelling Salesman Problem - CS325

Joshua Villwock Jaron Thatcher Ryan Phillips

March 14, 2014

Our Solution

Introduction

To prepare for solving the given test cases within the testing window, we wrote a suite of TSP-related functions/programs. These can be divided into four categories: analysis, solvers, optimizers, and a client-server system. These elements were then used together in a strategy that was actively run by all participants during the testing window.

Analysis

In order to determine how effective each approach was we generated and ran randomized test cases over a range of inputs. Matplotlib was then used to generate a number of plots including the city sets, particular routes of interest, route length vs. time, and route length vs. N.

Solvers

To get an initial solution, we wrote a couple of basic TSP-solvers. These included the following types of algorithms: exact, greedy, and minimum spanning tree.

Exact simply tried to every possible permutation on the city set and returned the one with the shortest route length. This wasnt practical for any of the given test set, but it is a good approach for a small set of cities (like less than 10). This algorithm has a runtime of O(n!), where n is the number of cities.

The greedy algorithm starts at a specified starting city and then picks the closest city to it as the next stop in its route. This new city is removed from the set of possible route destinations, and the process is repeated as long any unvisited cities remained. This algorithm has a runtime of O(n*n). A variation of this which gave consistently better results (at the expense of another factor of n).

The tsp solver began by constructing a graph representation of the cities by calculating the distances between every pairs of nodes. It then picked the smallest distance from the list, and added that edge and the two cities that connected to it to the graph. Next...

Optimizers

These functions took as input a set of cities and a valid route and would try making changes, and then continually test to see if any of the changes yielded a shorter route.

Reverse Optimization: Given a particular section length, this function reverses the order of every possible sub-route of this size.

City Swap Optimization: Takes 2 or more cities and swaps their positions in the route. This function had a couple of different flavors, including a randomized and a systematic method. The second flavor attempted every possible swap and the first just picked from a random subset of these possibilities.

City Transpose Optimization: For every city in the route, try shifting it to every other possible position in the route. There was also a variation of this function that performed this same operation with not just a single city, but with a subroute.

Exact Section Optimization: Given a particular subroute, it tries every possible permutation to see which is optimal. This uses the same approach as the exact TSP solver, so the same limits apply. We only tried it with subroute lengths less than ten.