**Project 1 – Brute Force**

Zachary Shumate

[CECS]

Speed School of Engineering

University of Louisville, USA

znshum01@louisville.edu

1. **Introduction** (What did you do in this project and why?)

This project tasked me with solving the Traveling Salesperson Problem for a set of cities whose coordinates are provided via brute force. This required finding the lowest costing route that goes to each city and returns to the starting city. The brute force method for finding this cycle is to check every permutation of cycle and compare the route costs until the lowest is determined.

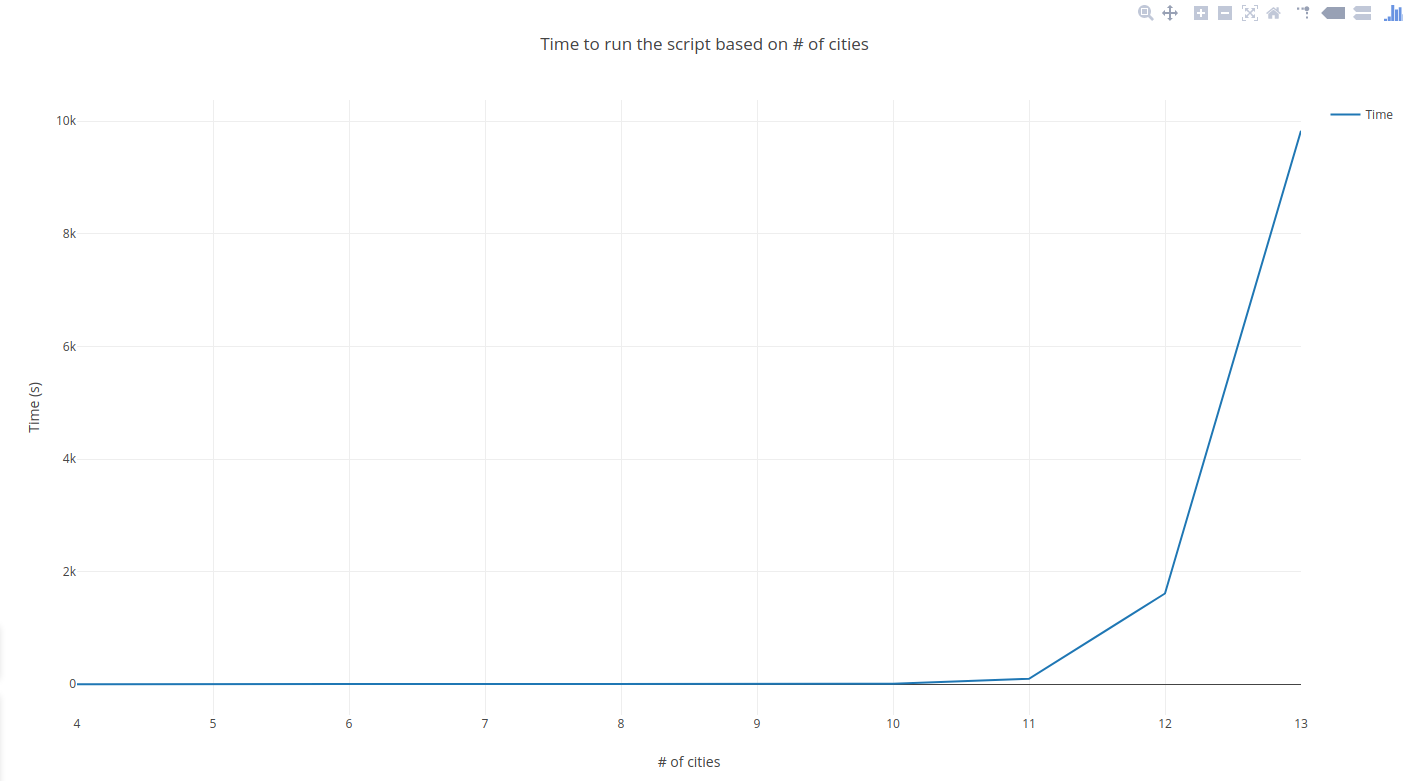
1. **Approach** (Describe algorithm you are using for this project)

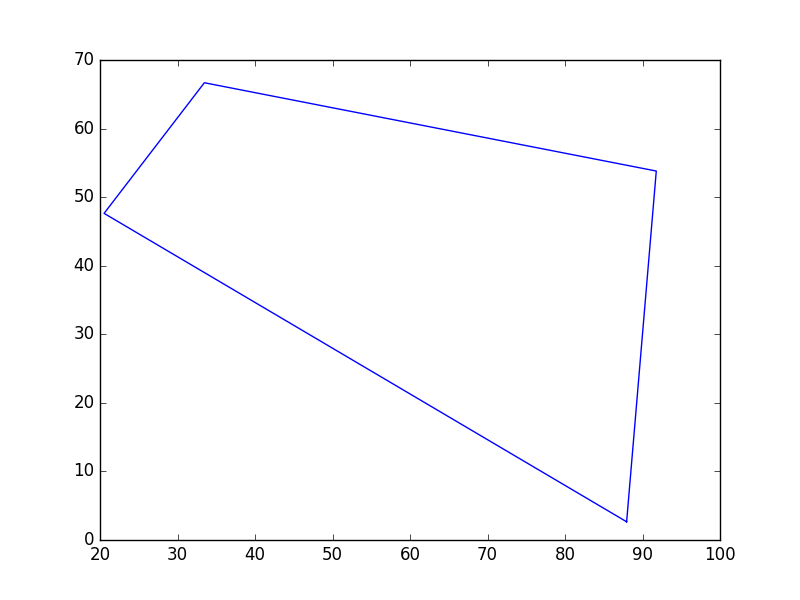
My Python algorithm takes a file with the cities and coordinates and then puts them into a list. After removing the first city to remove unnecessary computation cost, I then generated every permutation of cities in the list using the itertools Python library. Looping though this list, I calculate the total distance traveled in a cycle and compare it to the shortest distance calculated thus far. After every permutation has been tested, my output contain only the minimum costing cycle.

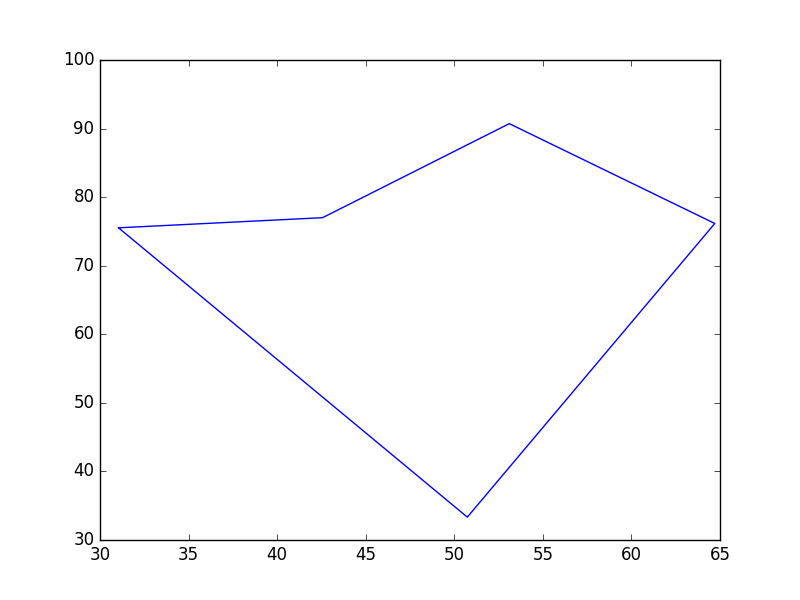
1. **Results** (How well did the algorithm perform?)
   1. **Data** (Describe the data you used.)

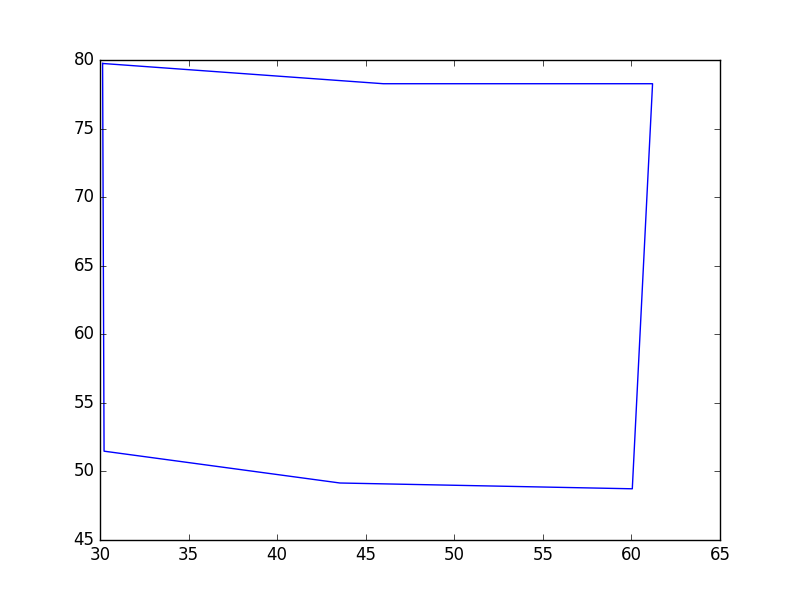
I used a variety of input files containing four to fourteen cities with random x and y coordinates. Most of these files contain city coordinates unique to the file with some exceptions in the files for thirteen and fourteen cities.

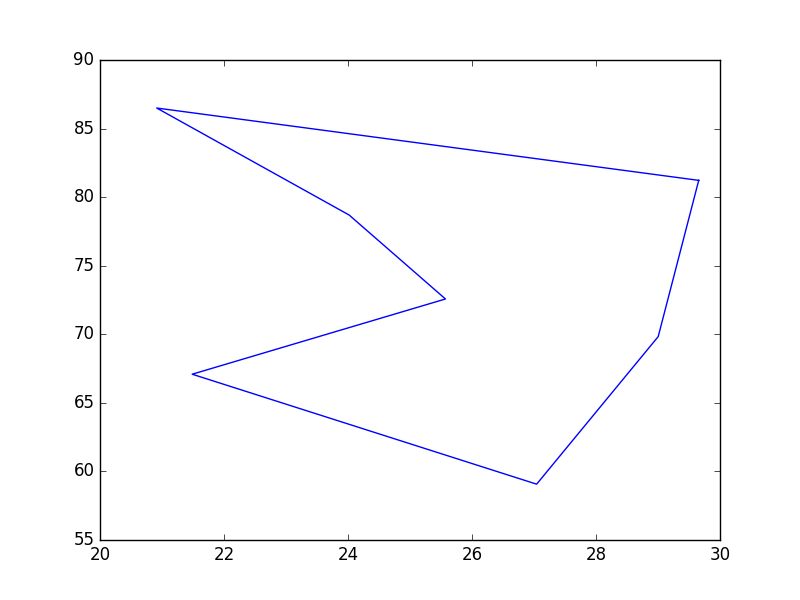
* 1. **Results** (Numerical results and any figures or tables.)

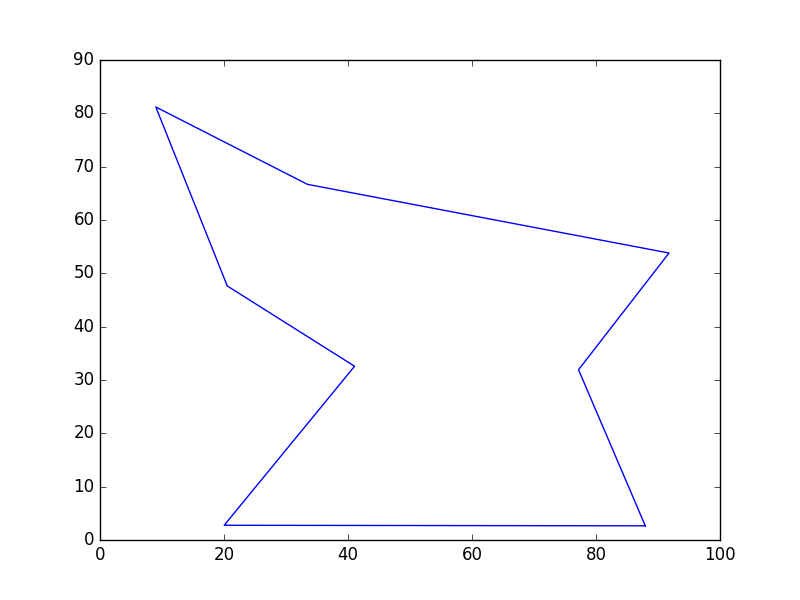


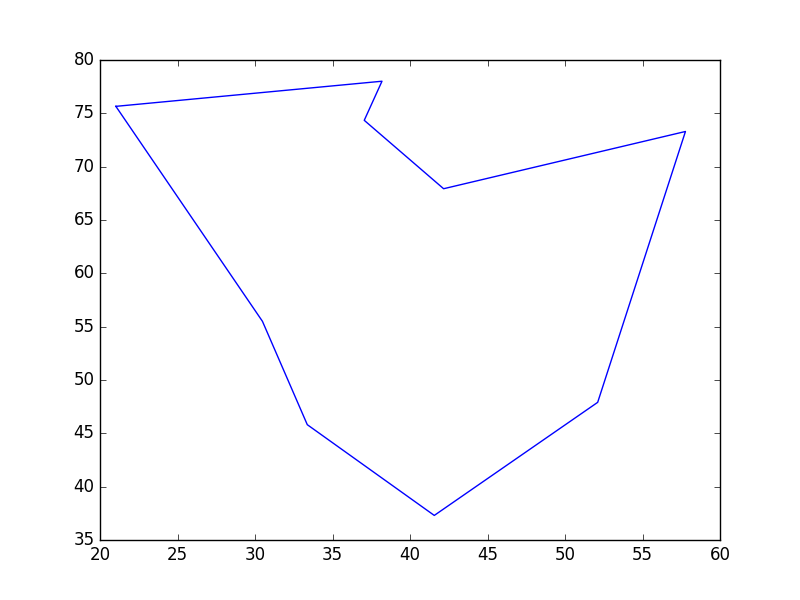
  
4 Cities

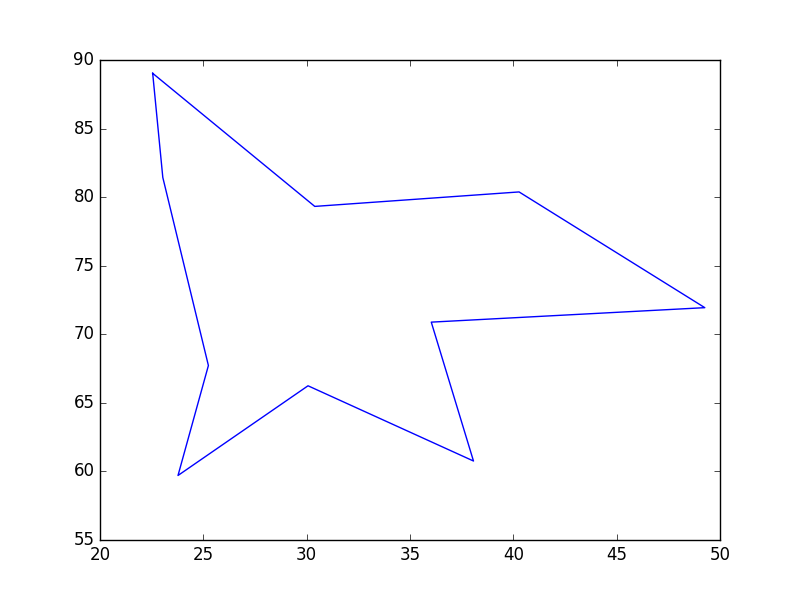
  
5 Cities

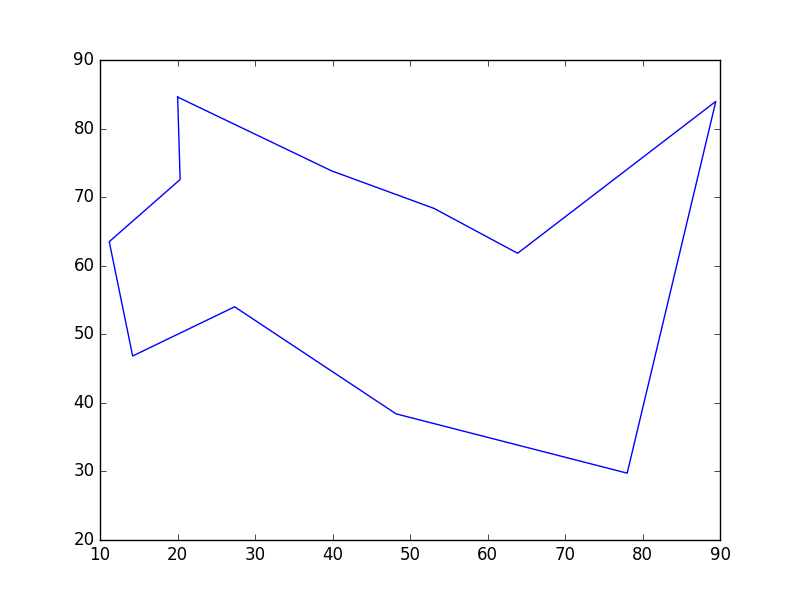
  
6 Cities

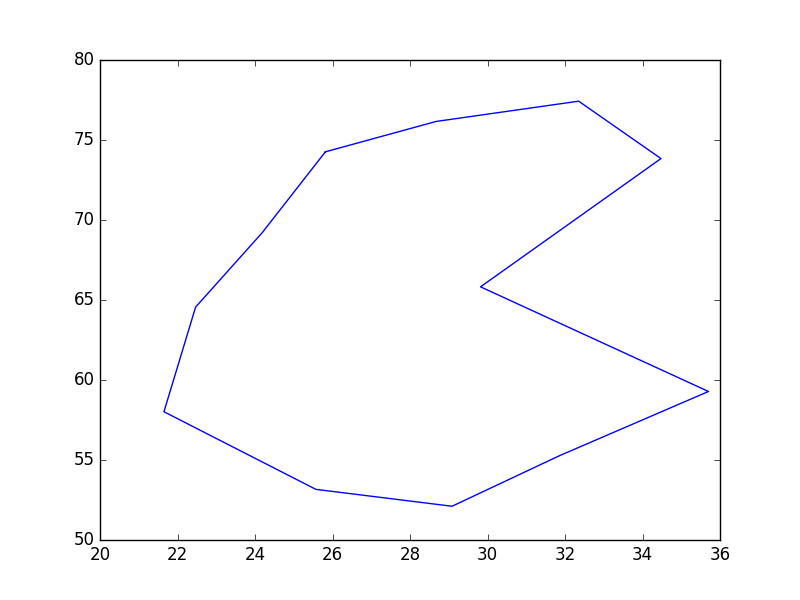
  
7 Cities

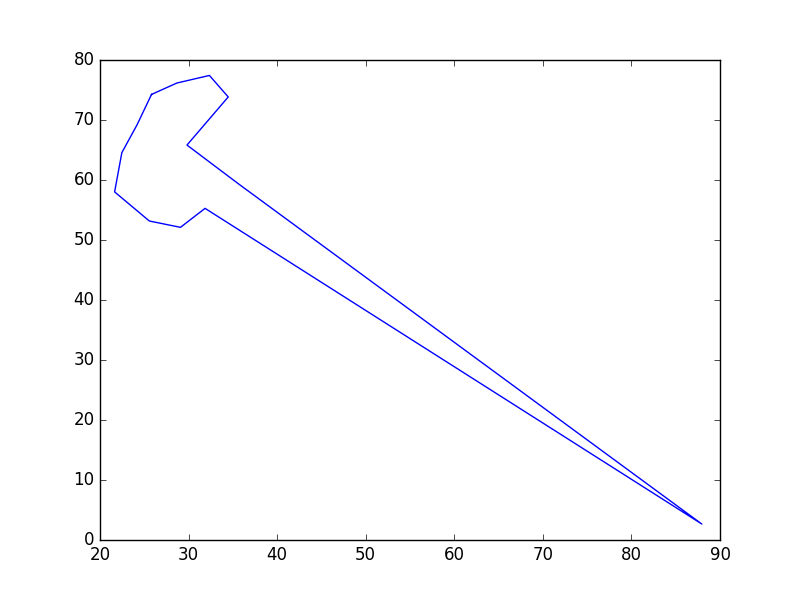
  
8 Cities

  
9 Cities

  
10 Cities

  
11 Cities

  
12 Cities

  
13 Cities

I used the provided files for testing 4 to 12 cities. The file for 13 cities is the same as 12 with an additional city.

**4 cities**: **'path'**: [{'y': '2.658162', 'x': '87.951292', 'num': '1'}, {'y': '47.633290', 'x': '20.526749', 'num': '4'}, {'y': '66.682943', 'x': '33.466597', 'num': '2'}, {'y': '53.807184', 'x': '91.778314', 'num': '3'}], **'distance'**: 215.08553303209044

**5 cities**: **'path'**: [{'y': '75.527426', 'x': '31.045752', 'num': '1'}, {'y': '33.333333', 'x': '50.735294', 'num': '4'}, {'y': '76.160338', 'x': '64.705882', 'num': '3'}, {'y': '90.717300', 'x': '53.104575', 'num': '5'}, {'y': '77.004219', 'x': '42.565359', 'num': '2'}], **'distance'**: 139.1335417499496

**6 cities**: '**path**': [{'y': '79.746835', 'x': '30.147059', 'num': '1'}, {'y': '51.476793', 'x': '30.228758', 'num': '6'}, {'y': '49.156118', 'x': '43.545752', 'num': '5'}, {'y': '48.734177', 'x': '60.049020', 'num': '4'}, {'y': '78.270042', 'x': '61.192810', 'num': '3'}, {'y': '78.270042', 'x': '45.996732', 'num': '2'}], '**distance**': 118.96891407553862

**7 cities**: '**path**': [{'y': '81.223629', 'x': '29.656863', 'num': '1'}, {'y': '86.497890', 'x': '20.915033', 'num': '4'}, {'y': '78.691983', 'x': '24.019608', 'num': '5'}, {'y': '72.573840', 'x': '25.571895', 'num': '6'}, {'y': '67.088608', 'x': '21.486928', 'num': '3'}, {'y': '59.071730', 'x': '27.042484', 'num': '7'}, {'y': '69.831224', 'x': '29.003268', 'num': '2'}], '**distance**': 63.863031874767636

**8 cities**: '**path**': [{'y': '2.658162', 'x': '87.951292', 'num': '1'}, {'y': '31.922361', 'x': '77.181310', 'num': '7'}, {'y': '53.807184', 'x': '91.778314', 'num': '3'}, {'y': '66.682943', 'x': '33.466597', 'num': '2'}, {'y': '81.185339', 'x': '9.006012', 'num': '5'}, {'y': '47.633290', 'x': '20.526749', 'num': '4'}, {'y': '32.578509', 'x': '41.059603', 'num': '8'}, {'y': '2.761925', 'x': '20.032350', 'num': '6'}], '**distance**': 310.98207974423167

**9 cities**: '**path**': [{'y': '75.654450', 'x': '21.024096', 'num': '1'}, {'y': '78.010471', 'x': '38.192771', 'num': '8'}, {'y': '74.345550', 'x': '37.048193', 'num': '4'}, {'y': '67.931937', 'x': '42.168675', 'num': '9'}, {'y': '73.298429', 'x': '57.771084', 'num': '2'}, {'y': '47.905759', 'x': '52.108434', 'num': '5'}, {'y': '37.303665', 'x': '41.566265', 'num': '3'}, {'y': '45.811518', 'x': '33.373494', 'num': '6'}, {'y': '55.497382', 'x': '30.481928', 'num': '7'}], '**distance**': 131.02836613987677

**10 cities**: '**path**': [{'y': '89.029536', 'x': '22.549020', 'num': '1'}, {'y': '81.434599', 'x': '23.039216', 'num': '2'}, {'y': '67.721519', 'x': '25.245098', 'num': '7'}, {'y': '59.704641', 'x': '23.774510', 'num': '6'}, {'y': '66.244726', 'x': '30.065359', 'num': '8'}, {'y': '60.759494', 'x': '38.071895', 'num': '5'}, {'y': '70.886076', 'x': '36.029412', 'num': '9'}, {'y': '71.940928', 'x': '49.264706', 'num': '10'}, {'y': '80.379747', 'x': '40.277778', 'num': '4'}, {'y': '79.324895', 'x': '30.392157', 'num': '3'}], '**distance**': 106.78582021866472

**11 cities**: '**path**': [{'y': '84.599156', 'x': '20.016340', 'num': '1'}, {'y': '73.839662', 'x': '39.869281', 'num': '6'}, {'y': '68.354430', 'x': '53.104575', 'num': '10'}, {'y': '61.814346', 'x': '63.888889', 'num': '11'}, {'y': '83.966245', 'x': '89.460784', 'num': '8'}, {'y': '29.746835', 'x': '78.022876', 'num': '9'}, {'y': '38.396624', 'x': '48.202614', 'num': '7'}, {'y': '54.008439', 'x': '27.369281', 'num': '5'}, {'y': '46.835443', 'x': '14.215686', 'num': '3'}, {'y': '63.502110', 'x': '11.192810', 'num': '4'}, {'y': '72.573840', 'x': '20.343137', 'num': '2'}], '**distance**': 252.6844344550543

**12 cities**: '**path**': [{'y': '74.261603', 'x': '25.816993', 'num': '1'}, {'y': '76.160338', 'x': '28.676471', 'num': '8'}, {'y': '77.426160', 'x': '32.352941', 'num': '2'}, {'y': '73.839662', 'x': '34.477124', 'num': '3'}, {'y': '65.822785', 'x': '29.820261', 'num': '12'}, {'y': '59.282700', 'x': '35.702614', 'num': '4'}, {'y': '55.274262', 'x': '31.862745', 'num': '9'}, {'y': '52.109705', 'x': '29.084967', 'num': '5'}, {'y': '53.164557', 'x': '25.571895', 'num': '10'}, {'y': '58.016878', 'x': '21.650327', 'num': '6'}, {'y': '64.556962', 'x': '22.467320', 'num': '7'}, {'y': '69.198312', 'x': '24.183007', 'num': '11'}], '**distance**': 66.08484401133855

**13 cities**: '**path**': [{'y': '74.261603', 'x': '25.816993', 'num': '1'}, {'y': '76.160338', 'x': '28.676471', 'num': '8'}, {'y': '77.426160', 'x': '32.352941', 'num': '2'}, {'y': '73.839662', 'x': '34.477124', 'num': '3'}, {'y': '65.822785', 'x': '29.820261', 'num': '12'}, {'y': '59.282700', 'x': '35.702614', 'num': '4'}, {'y': '2.658162', 'x': '87.951292', 'num': '13'}, {'y': '55.274262', 'x': '31.862745', 'num': '9'}, {'y': '52.109705', 'x': '29.084967', 'num': '5'}, {'y': '53.164557', 'x': '25.571895', 'num': '10'}, {'y': '58.016878', 'x': '21.650327', 'num': '6'}, {'y': '64.556962', 'x': '22.467320', 'num': '7'}, {'y': '69.198312', 'x': '24.183007', 'num': '11'}], '**distance**': 214.48611904211143

1. **Discussion** (Talk about the results you got and answer any specific questions mentioned in the assignment.)

My algorithm was able to handle finding the Hamiltonian path for a file with up to 13 cities; on the file with 14 cities, the program crashed. For each instance that was able to run, I was able to confirm that it was the shortest path for that set of data. I also took notice of how the times that each run took was exponentially long from the previous times the program was run. Files with 9 cities or less all took under one second to complete, but those with more cities saw a clear exponential explosion in time.

1. **References** (If you used any sources in addition to lectures please include them here.)

My code used an external library Matplotlib to generate graphs for the shortest cycle. The code I used was generic code similar to example that were provided. To run my code, this library needs to be installed; instructions for this can be found at <https://matplotlib.org/users/installing.html>