**Comparison of Running time for Greedy Search, Dijkstra Search and Brute Force Search about TSP**

Xin Zheng

[Computer Engineer and Computer Science]

Speed School of Engineering

University of Louisville, USA

[x0zhen07@louisville.edu](mailto:x0zhen07@louisville.edu)

1. **Introduction**

Greedy Search is an algorithm which makes a locally optimal choice in the hope that this choice will lead to a globally optimal solution[1]. It works for some problems, in which we can make sure the optimal solution of subproblem will yield to a global optimal solution. Meanwhile, the running time of Greedy search is quite fast comparing with some algorithms, such as Brute Force Search [2].Brute-force search consists of systematically enumerating all possible candidates for the solution and checking whether each candidate satisfies the problem's statement.

The algorithms relate to ‘Travelling Salesman Problem(TSP)’[3]. TSP asks the following question: "Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city once and returns to the origin city. In this paper, I will show the comparison of running time for Greedy Search, Dijkstra Search and Brute Force Search about TSP question.

Dijkstra Search[4] is an [algorithm](https://en.wikipedia.org/wiki/Algorithm) for finding the [shortest paths](https://en.wikipedia.org/wiki/Shortest_path_problem) between [nodes](https://en.wikipedia.org/wiki/Vertex_(graph_theory)) in a [graph](https://en.wikipedia.org/wiki/Graph_(abstract_data_type)). Here, I will just find the shortest path from the start city to the end city.

1. **Approach**

**2.1 Tool and software in this paper**

* Programming language: Python3.7
* Excel Sheet for mac: 16.161.

**2.2 Algorithms**

**2.2.1 Greedy Search**

Greedy Search will not guarantee to get the best solution for TSP, but normally it gives an ‘OK’ solution. Furthermore, this algorithm normally saves a large amount of running time. I will show the running result in the part 3.

To solve TSP with Greedy Search, how to pick the next city is the subproblem. If we can make sure every time, we pick the right city, then we can get an optimal solution for the whole trip. In the project I tried two different methods to find out the best solution for subproblem. Solution 1, find the shortest distance from the node which has been visited, to another nodes, which is not in the tour. Below is the code for this algorithm.

def greedySearch(graph,start):  
 vetex = []  
 vetex.extend(range(len(graph))) # create node list  
 #distance\_dic[start-1] = 0 # Assign distance for start node as 0  
 # Assign all other distance value as infinity  
  
 distance = 0 # make a flag for each node   
 for i in range(start, len(graph)):  
 color\_dic[i] = 'WHITE'  
  
 global visited  
 u = start -1  
 vetex.remove(u)  
 color\_dic[u]= 'BLACK'  
 print("this is vetex: " + str(vetex))  
 temp\_short = inf  
 visited.append(u) #append the first node to tour  
  
 while vetex: #solution of find out solution from node to node  
  
 for current\_neighbour in graph[u]:  
 if current\_neighbour != 0 and color\_dic[graph[u].index(current\_neighbour)] == 'WHITE' \  
 and current\_neighbour < temp\_short:  
 temp\_short = current\_neighbour  
 temp = graph[u].index(current\_neighbour)  
 print("this is temp short: ", temp\_short)  
 print("This is temp: ", temp)  
 distance += temp\_short  
 temp\_short = inf  
 visited.append(temp)   
 vetex.remove(temp)  
 color\_dic[u] = 'BLACK'  
 u = temp  
 distance+=graph[visited[-1]][0]  
 visited.append(start-1)  
 visited = np.array(visited)+1  
  
 return visited,distance

Meanwhile, I tried another method. Every time, when I pick the next node, I will calculate the distance from an edge, which have been visited, to a node, which is not in the tour. Then pick the node which has the shortest distance mentioned above. Below is the code to calculate the distance.

def distEdge\_node(edgeNode1,edgeNode2,node): #calculate distance between an edge to a node  
 x1 = cities[edgeNode1][0]  
 x2 = cities[edgeNode2][0]  
 x0 = cities[node][0]  
 y1 = cities[edgeNode1][1]  
 y2 = cities[edgeNode2][1]  
 y0 = cities[node][1]  
 d1 = abs((x2-x1)\*(y1-y0)-(x1-x0)\*(y2-y1))  
 d2 = ((x2-x1)\*\*2+(y2-y1)\*\*2)\*\*0.5  
 d = d1/d2  
 return d

**2.2.2 Brute force Search and Dijkstria Search.**

Since we have talk about these two algorithms in project1 and project2, I will not give more details about algorithm here.

**3. Result**

**3.1 Data**

In this project, there are two city lists, one is with 30 cities, the other one is with 40 cities.

The format is ‘.tsp’ which is a txt file. [Tap here for details about document format.](https://wwwproxy.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/tsp/)

Here is an example of list with 4 cities.

NAME: concorde4

TYPE: TSP

COMMENT: Generated by CCutil\_writetsplib

COMMENT: Write called for by Concorde GUI

DIMENSION: 4

EDGE\_WEIGHT\_TYPE: EUC\_2D

NODE\_COORD\_SECTION

1 87.951292 2.658162

2 33.466597 66.682943

3 91.778314 53.807184

4 20.526749 47.633290

**3.2 Result.**

**3.2.1 Running time chart.**

****

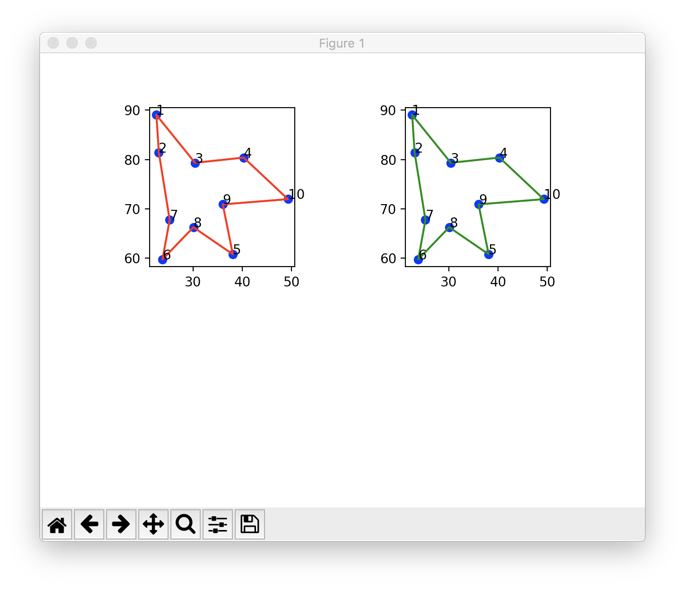
According to the chart above, we can see the running time of bot Greedy algorithms are the shortest. To find the short path of 10 cities, it takes 68 seconds with Brute Force algorithm. In another hand, to find a short path of 40 cities, it just takes around 1 second with Greedy algorithm. Thus, the Greedy algorithm bests Brute Force algorithm. Meanwhile, it takes around 1second to find a shortest path of 11 cities with Dijkstra Search. That is the same running time for Greedy Algorithm to search 40 cities. In this point of view, we get conclusion that, Greedy Algorithm runs fastest among the three algorithms.

Now let us compare the outputs of these algorithms.

**3.2.2 Output of three algorithms.**

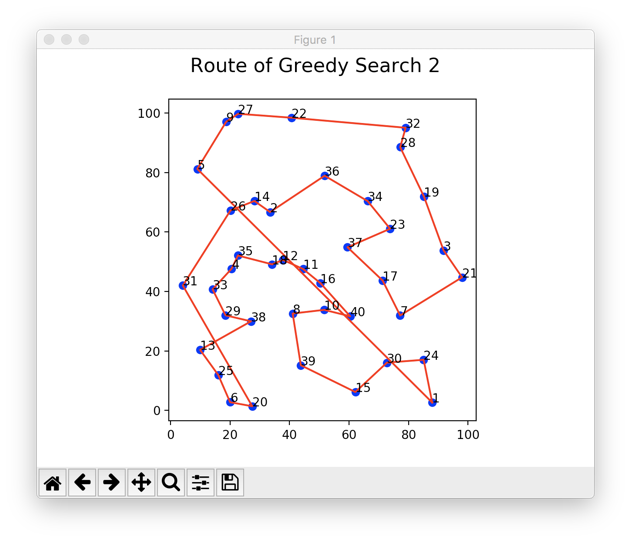
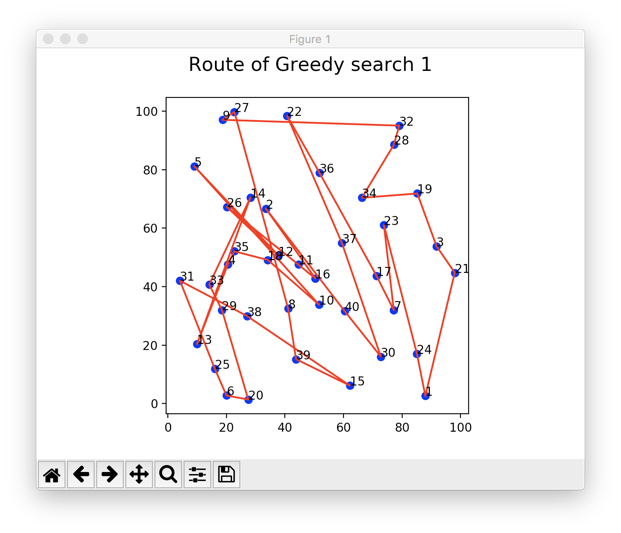
Below figure is the two routes of 10 cities from Brute Force Algorithm. That shows the solutions are pretty good.

Route 1

****

Route2 and Route 3 are the outputs for Greedy algorithm, ‘Greedy 1’ and ‘Greedy 2’, respectively.

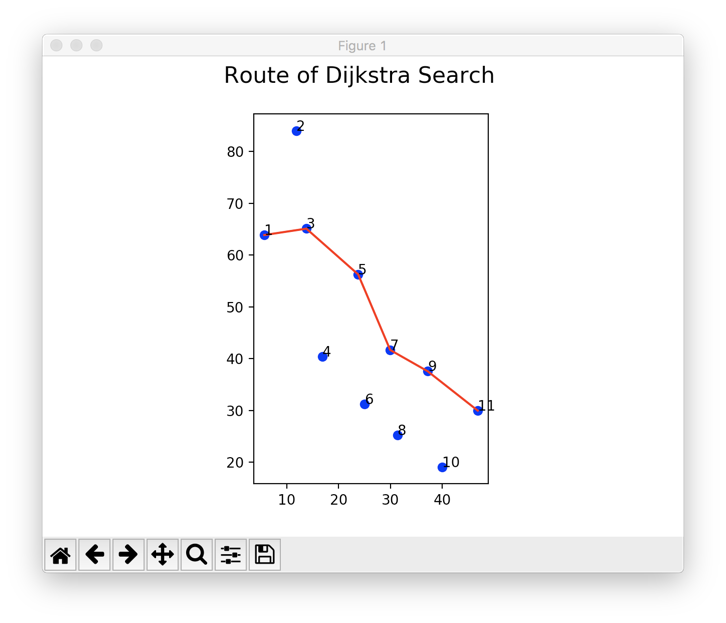
Route 2 Route 3

****

From route 2 and route 3, we can see, algorithm of Greedy 2 works better. According to the running time chart, the distance for Greedy 1 is 1139, and Greedy 2 is 684. This proves, algorithm of Greedy 1 works better. Even neither of them a very good solution.

Route 4 is the output of Dijkstra search, it shows the result of Dijkstra Algorithm is very good.

Route 4

****

**4. Discussion**

Base on above tests, we can see Brute Force Algorithm always gives a good solution, however, the running time increase fast with the increase of city number. Dijkstra Algorithm provides a good solution too, though the running time is not very good. Greedy algorithm supplies an OK solution in a short period. Thus, Greedy Algorithm is the best option when we visit a lot of cities, such as 20,30 or even 100.

**5. Conclusion**

From this project, we can make a conclusion that is sometimes we pick algorithms base on the balance of running time and solution. The algorithm with shorter running time is a better choice than find out the best solution with very long executing time for some cases.

**6. References**

1. Greedy Algorithm

<https://en.wikipedia.org/wiki/Greedy_algorithm>

1. Brute Force Algorithm

<https://en.wikipedia.org/wiki/Brute-force_search>

1. Travelling Salesman problem

<https://en.wikipedia.org/wiki/Travelling_salesman_problem>

1. Dijkstra Search

<https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm>