CS 256: Homework 4

Akriti Sethi

November 3, 2017

Abstract

In this assignment, we implement the Travelling Salesman Problem or the TSP. The TSP is a classic problem focused on optimization. The problem is described as follows: It describes a salesman who must visit all the cities and return back to the start city. In this case, the goal of the traveller is to keep the traversal cost and distance to the minimal. The problem is classified as a NP-hard problem. We implemented the TSP by three approaches: Firstly, we used the genetic algorithm to solve this problem. Secondly, we used the greedy approach to solve the problem. Thirdly, we implemented the A* algorithm to find an optimal path to the problem. In all the cases, we recorded the tour length and time required to complete the traversal of the salesman to all the cities. We ran the program on various benchmark datasets such as att48.tsp, pr1002.tsp, and so on. The results were tabulated and analysis was carried out on the basis of the table.

1 Introduction

This problem has been of interest to the computer scientists and mathematicians since a very long time. A problem x is set to be a NP-hard problem when there is a NP-complete problem Y, such that Y is reducible to X in polynomial time.TSP is one such problem. Firstly, we would implement the TSP by the genetic algorithm, the greedy approach and the A* algorithm. Next, we would run all the algorithms on the provided benchmark datasets such as att48.tsp, pr1002.tsp, nrw1379.tsp,fnl14461.tsp and so on. For each file and for every algorithm, we would tabulate the tour length, i.e., the distance travelled to traverse all the nodes and reach back to the start city, and also the time taken in doing so.

2 Theory

2.1 Travelling Salesman Problem

The TSP describes a salesman who must travel all n cities. His aim is to do so in a minimal time and with minimal cost. [**?**]

2.2 Genetic algorithm

Genetic algorithms are also known as evolutionary algorithms.The algorithm starts with a variety of solution sequences and performs a lot of random crossovers at each iteration. At every stage, the fitness function is calculated, until an optimal solution is reached. The algorithm always requires a genetic representation of the solution and a fitness function to generate the further crossovers. For our implementation we have used the pyevolve software.

2.3 Greedy algorithm

The greedy algorithm is an algorithmic paradigm that helps trace the path by taking the minimum distance available from that particular node. They usually fail to find a global optimal solution.

2.4 A* algorithm

This algorithm is widely used in path finding and graph traversal. In this algorithm, we find a start node and from there go in the direction of the lowest f(x) value. This f(x) is a summation of g(x) and h(x), where g(x) stands for the distance from the start and h(x) is the heuristics used to solve the problem. The solution of the problem depends a lot on the heuristics used.

3 Experiment

3.1 Fabrication

First, we started by running the genetic algorithm for the benchmark files. We started to organize the data in a table like format so that we could later use it for graph plotting and analysis. Second, we moved on to the greedy approach and recorded the tour length and time taken to visit all the nodes. Lastly, we implemented the A* algorithm. The benchmark files had too many records, hence we used a smaller data set to analyze our results for the A* algorithm.

3.2 Experimental set-up

To start our experiment we first ran all the 3 algorithms on the benchmark file: att48.tsp. The genetic algorithm took about 25.49 seconds to complete visiting the cities whereas for the same file, the greedy process is taking 0.010 seconds The att48.tsp has a total of 48 cities and the tour length we get for genetic algorithm is 44613.64 while that for greedy algorithm is 42982.06 In this case, greedy is proving out to be a better choice instead of genetic algorithm. As far as A* is considered for att48.tsp, it is taking a lot of time to run, hence we take some sample files and perform A* on that.

3.3 Computation Specifications

Processor: Intel(R)Core (TM)i5-2410M CPU@2.30GHz, 2.30GHz
Installed memory (RAM): 8.00 GB
System type: 64 bit operating system

Number of cities	File name	Genetic algorithm	Greedy Algorithm
		Time taken (sec)	Time taken (sec)
48	att48.tsp	25.499	0.109
1002	pr1002.tsp	1218.857	27.394
1379	nrw1379.tsp	1841.030	68.551
4461	fnl4461.tsp	6316.279	1280.121
1291	d1291.tsp	616.511	16.212
1060	u1060.tsp	470.292	9.420
1748	vm1748.tsp	859.240	42.365
2325	u2319.tsp	1272.371	94.931
1817	u1814.tsp	886.980	43.924
1323	rl1323.tsp	1534.115	19.401
1655	d1655.tsp	2662.046	124.443
1577	fl1577.tsp	2301.082	105.095
2103	d2103.tsp	2990.215	241.157
1084	vm1084.tsp	1572.211	39.691

Table 1: A table showing the time taken for each benchmark file.

Operating System used : Windows 7
Laptop used : Dell Inspiron

4 Table

I have tabulated the tour length and time taken for 14 benchmark files. Here is what the data looks like:

5 Results and interpretation

The above table shows that there is a significant drop in the time taken to complete the traversal from the implementation of genetic algorithm to greedy algorithm. The table in the excel file attached also highlights the tour length taken to complete the traversal of all the nodes is lesser for greedy as compared to genetic algorithm. The graph also reiterates the same fact. The greedy approach was to take the minimum distance from the respective node. The genetic algorithm randomizes and crossovers the obtained genome sequence and hence takes time to reach the goal state. As the number of cities increases, the computation time for both greedy and genetic algorithm increases. But in some case, example, in the benchmark file d1291.tsp, even though the number of cities has increased, the time taken for both the algorithms to be completed is less

Number of cities	File name	Genetic algorithm	Greedy Algorithm
		Tour length	Tour length
48	att48.tsp	44613.641	42982.076
1002	pr1002.tsp	4763977.01	316593.7983
1379	nrw1379.tsp	1110617.05	70565.289
4461	fnl4461.tsp	7294753.637	226062.738
1291	d1291.tsp	1380207.317	61535.163
1060	u1060.tsp	5006812.715	285612.224
1748	vm1748.tsp	12083215.07	417970.084
2325	u2319.tsp	5085123.353	274925.508
1817	u1814.tsp	1710207.002	71553.309
1323	rl1323.tsp	7739096.775	327537.0438
1655	d1655.tsp	1756139.941	76053.384
1577	fl1577.tsp	1090187.669	27798.68074
2103	d2103.tsp	2697923.256	92810.85475
1084	vm1084.tsp	6344315.932	302001.093

Table 2: A table showing the tour length for each benchmark file.

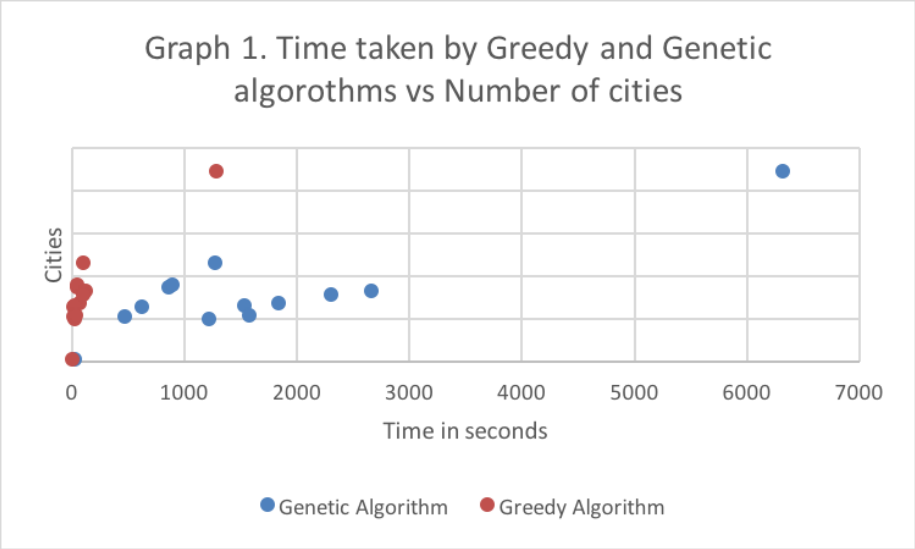

Figure 1: Graph showing the time taken by greedy and genetic algorithm against the number of cities

than the benchmark file pr1002.tsp. In the A* search algorithm, it was taking a very long time to run for the att48.tsp file, hence we could not output the result for the same. For A*, the probability of getting a faster solution is higher when we get all the minimum values of f(x) in the same branch and we need not go back to any of the previous nodes. Excluding A*, in my opinion, greedy algorithm works better than the genetic algorithm since the genetic algorithm is very dependent on the initial sequence and a lack of local search ability.

6 Discussion

Although there are some exceptions, but the performance of greedy appears to be better than that of genetic algorithm.

References

[1] *1.* www.wikipedia.com