Project1 COSC 6368 Fall 2017

Solving the Travelling Salesman Problems

---

1. Problem:

Travelling Salesman Problem is a NP-HARD problem.
Basically, we will be provided with a list of cities and the distances between each pair of cities.
We should find the shortest possible route that visits each city exactly once and returns to the origin city.
The shortest possible route is found based on the total distance travelled by following the cities in the route.
We need to know that in optimization problems like the travelling salesman problem, we need to find "an" optimal solution instead of "the" optimal solution, because there may be several solutions that achieve the same optimal value.

2. Problem Statement:

1) To design two search strategies that solve N-city Traveling Salesman Problems (TSP) with cities being numbered 0… N-1.
2) One of these search strategies (SIM) is quite simple.
3) The second strategy (SOPH) is more sophisticated.
4) To design a software system that implements SIM and SOPH and provides the capability to run the SIM and SOPH for 3 given cost functions.
5) Software has an interactive interface that allows user to select:
    - the search strategy (either SIM or SOPH)
    - the cost function (either c1 or c2 or c3)
    - the number of cities N for which the TSP should be solved
    - a maximal effort bound MEB (maximum number of states searched)
    - an integer random number (optional parameter; is only needed if your search strategy is randomized in which case the supplied random number is used as the seed for the random number generator you use in your implementation).

3. Implemented Strategies To Solve the Travelling Salesman Problem:

There are many a way to implement the travelling salesman problem. Out of which, I have selected two search strategies, one for simple and other for sophisticated.

a) Greedy Approach(Nearest Neighbor next):

A Greedy Algorithm "tries" to find an optimal solution by making a sequence of greedy choices. In each step, it makes the choice that looks best at the moment, according to some local criterion.

Travelling Salesman Problem specific greedy approach is as follows:

- Given 'n' cities and the distances between each pair of cities.
- Pick an arbitrary city from all the given cities by using some Random approach.
- Call this city as City 1.
- Now, find a city with the smallest distance from City 1.
- Call this new city as City 2.
- Find a city in the rest of the n-2 cities with the smallest distance from city2 and continue the process.
- Finally, we get the output of the tour as:
  City 1 ⟹ City 2 ⟹ City 3 ⟹ ……⟹ City n ⟹ City 1

Advantages:

This greedy approach is real quick. It gives the output path within seconds even for high value of n (number of cities).Analyzing the run time for **greedy algorithms** will generally be much easier than for other techniques.

Disadvantages:

Generally, for many difficult optimization problems, greedy algorithm fail to find an optimal solution, because the greedy choices are made according to some local criterion.

But our goal here is to find a globally optimum solution.

This algorithm is one in which the choices made in a globally optimal solution may not be locally optimum.

In this project, the implementation of Greedy approach is as follows:

I. Firstly, we are given 3 cost functions for which if we give two values (city numbers) as input, it will give us the distance between the two given cities.
II. Using one of the cost function for the specific test case, we construct a 2-D Array of size nXn which contains the distances between each pair of cities.
III. We will take the start city from the user and make it the first city in the output.

IV. Initialize the totalCost with zero.

V. Now, we will find out the column in which the minimum value with start city as a row number, i.e., find the city to which the distance from start city is minimum and get the city number which is the column number in the row.

VI. Add the distance between them to the totalCost and make the new city as start city and repeat V, VI till we get the n cities in the output.

VII. Now, we report the total cost and the path as output.

```
public static String bstFrstSrch(int[][] matrix, int rowNum, String path,int counter) {
    // TODO Auto-generated method stub
    if(counter==matrix[rowNum].length-1)
        return path;
    else{
        int min=getColNumOfMin(matrix[rowNum],rowNum,path);
        return bstFrstSrch(matrix, min, path+(min)+"\t",counter+1);
    }
}

private static int getColNumOfMin(int[] is,int rowNum,String path) {
    // TODO Auto-generated method stub
    int min=0, i;

    // to choose min
    for(i = 0 ; i < is.length ; i++){
        if(path.indexOf("\t"+(i)+"\t")==-1){
            min=i;
            break;
        }
    }
    // to find min
    for(i=0; i<is.length ;i++){
        if( i!=rowNum && path.indexOf("\t"+(i)+"\t")==-1 && is[i] < is[min] )
            min=i;
    }

    // to add the cost of next node into Netcost
    cost+=is[min];
    return min;
}
```

Fig.1: Greedy Algorithm Sample Code for Travelling Salesman Problem

```
enter simp for simple and soph for sophisticated
simp
enter the number of cities
30
enter the cost function: c1 or c2 or c3
c1
enter the meb value
200000
enter strt pt
2
path is
:2    0    1    3    10    17    24    23    9    16    15    8    22    29    28    7    14    21    20    6    13    27    26
5    12    19    18    4    11    25
cost is:466
```

Fig.2: Greedy Algorithm Output for Travelling Salesman Problem

The results from the simple greedy approach are presented in the Table 1 below.
In this table:

C1, C2, C3 ⟹ three distinct cost functions.
Seed ⟹ starting city in the tour.
No. Of Cities ⟹ Number of cities to be visited in the tour.

No. of cities	Seed	C1	C2	C3
10	I. 2 II. 5 III. 8	426 426 426	60 72 76	1476 1488 1464
30	I. 2 II. 11 III. 18	466 466 466	216 307 233	37056 37344 37344
60	I. 5 II. 26 III. 53	526 526 526	252 516 271	292238 293414 292334

Table 1: Greedy Approach Cost for different cost functions given Seed and number of cities

Here, if the number of cities, Seed and the Maximal Effort Bound are given by the user, then the corresponding costs form the values in other columns.

b) Genetic Algorithm:

Genetic Algorithm largely falls under the Evolutionary Computing or Evolutionary Algorithms.

These are mainly used to generate **high quality** solutions for **optimization problems** (like Travelling Salesman Problem in our case). This is achieved mainly by making use of the bio-inspired operators like **mutation, crossover** and **selection.**
Genetic algorithm goes as follows:

1) **[Start]** Generate random population of *n* chromosomes (suitable solutions for the problem)
2) **[Fitness]** Evaluate the fitness *f(x)* of each chromosome *x* in the population
3) **[New population]** Create a new population by repeating following steps until the new population is complete
    a. **[Selection]** Select two parent chromosomes from a population according to their fitness (the better fitness, the bigger chance to be selected)
    b. **[Crossover]** With a crossover probability cross over the parents to form new offspring (children). If no crossover was performed, offspring is the exact copy of parents.
    c. **[Mutation]** With a mutation probability mutate new offspring at each locus (position in chromosome).
    d. **[Accepting]** Place new offspring in the new population
4) **[Replace]** Use new generated population for a further run of the algorithm
5) **[Test]** If the end condition is satisfied, **stop**, and return the best solution in current population
6) **[Loop]** Go to step **2**

(https://courses.cs.washington.edu/courses/cse473/06sp/GeneticAlgDemo/gaintro.html)

Advantages:

Concepts are easy to understand.
Always an answer.
Answer gets better with time.
Inherently parallel and easily distributable.
Chances of getting optimal solution are more.

Limitations:

Population considered for the evolution should be moderate for the problem.
Crossover should be 80-95%.
Mutation rate should be 0.5-1%.
Method of selection should be appropriate.
Fitness function should be accurate.

In this project Genetic Algorithm is implemented as follows:

1) Generate random population of *n* tours.
2) Evaluate the fitness (1/tourDistance) of each tour in the population

3) Create a new population by repeating following steps until the new population is complete
    a. Select two tours as parents from the population according to their fitness using **Tournament Selection** process.
    b. Cross over the parents to form new tour as a child using **Greedy Crossover** algorithm. (ref:https://arxiv.org/ftp/arxiv/papers/1209/1209.5339.pdf)
    c. With a mutation probability of 1.5%, mutate the child which is generated using a simple swap operator.
    d. Place the child generated after mutation in the new population.
4) Use new generated population for a further run of the algorithm.
5) **If** the end condition is satisfied, i.e., the given MEB reached zero, then **stop**, and **return** the best solution in current population.
6) **Else** Go to step **2.**

```
public static AISophPop revisePopulation(AISophPop pop){

    AISophPop revisedPopulation = new AISophPop(pop.popSize(),false);
    revisedPopulation.setTour(pop.getFittestTourFromPop());
    for( int i=0;i<pop.popSize();i++){
        AISophTour parent1 = selectParent(pop);
        AISophTour parent2 = selectParent(pop);
        AISophTour child = crossover(parent1,parent2);
        revisedPopulation.setTour(child);
    }
    for( int i=0;i<pop.popSize();i++){
        mutation(revisedPopulation.getTourFromPop(i));
    }
    return revisedPopulation;
}
```

Fig. 3: Sample Code to Generate New Population By taking the original population

No. of cities	C1	C2	C3
10	426 426 426	56.0 56.0 56.0	818 818 818

30	475 477 476	170 205 215	7238 7240 7238
60	576 581 571	733 663 845	209996 210086 210358
120	1929 1984	50218 52229 43927	1762008 1767112 1762306

Table 2: Genetic Algorithm Cost for different cost functions given number of cities

4. Comparison of Simple (Greedy) and Sophisticated (Genetic) Algorithms

1) Now consider Table 1 and Table 2.
   From Table 1 and Table 2, take the row with 10 cities as input.
   Find Average values of each column of that row in each Table, which are represented in the Histogram's Data Table below.

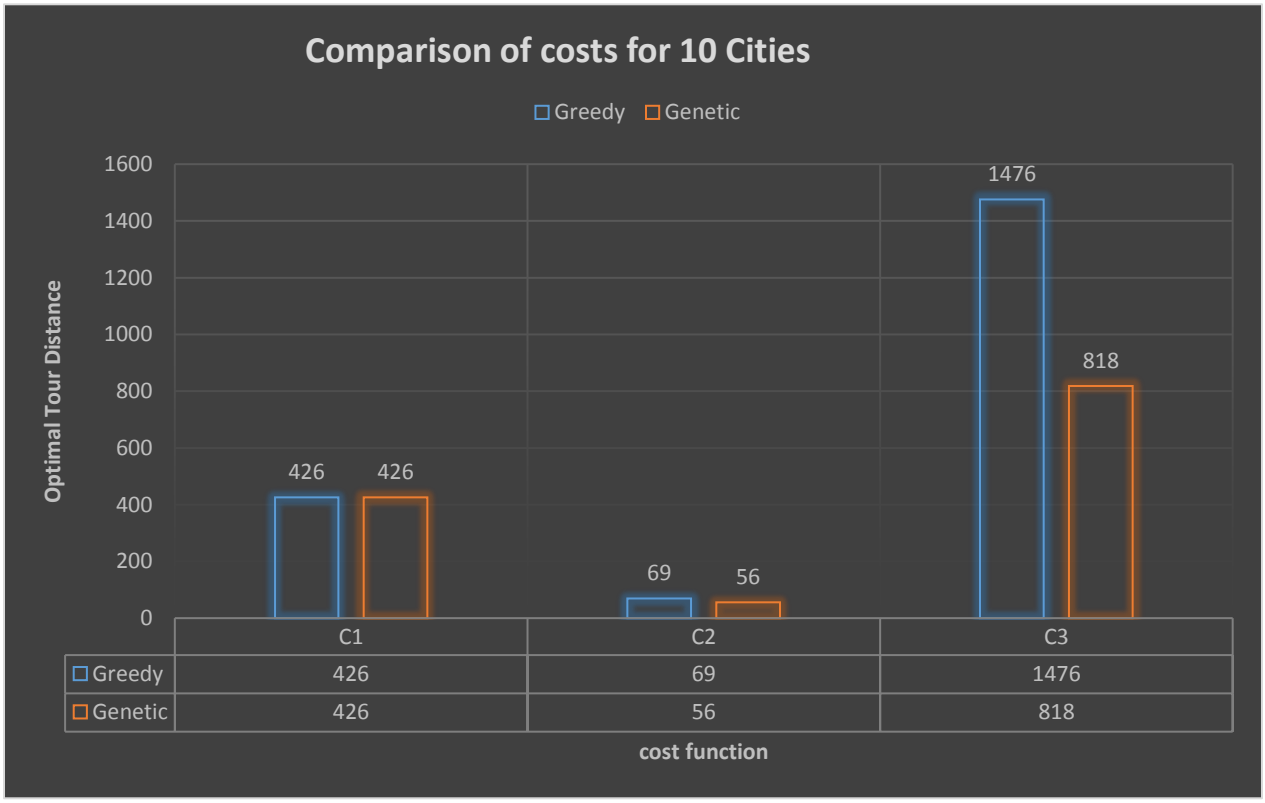

Fig. 5: Histogram that compares the Greedy and Genetic Algorithms' performance for 10 city tour

From this graph, we can conclude that the Genetic Algorithm's performance is:

i. Very high when compared to Greedy Approach for Cost Function C3.
ii. Equal to that of Greedy Approach for other Cost Functions.

2) Now consider Table 1 and Table 2.
From Table 1 and Table 2, take the row with 30 cities as input.
Find Average values of each column of that row in each Table, which are represented in the Histogram's Data Table below.

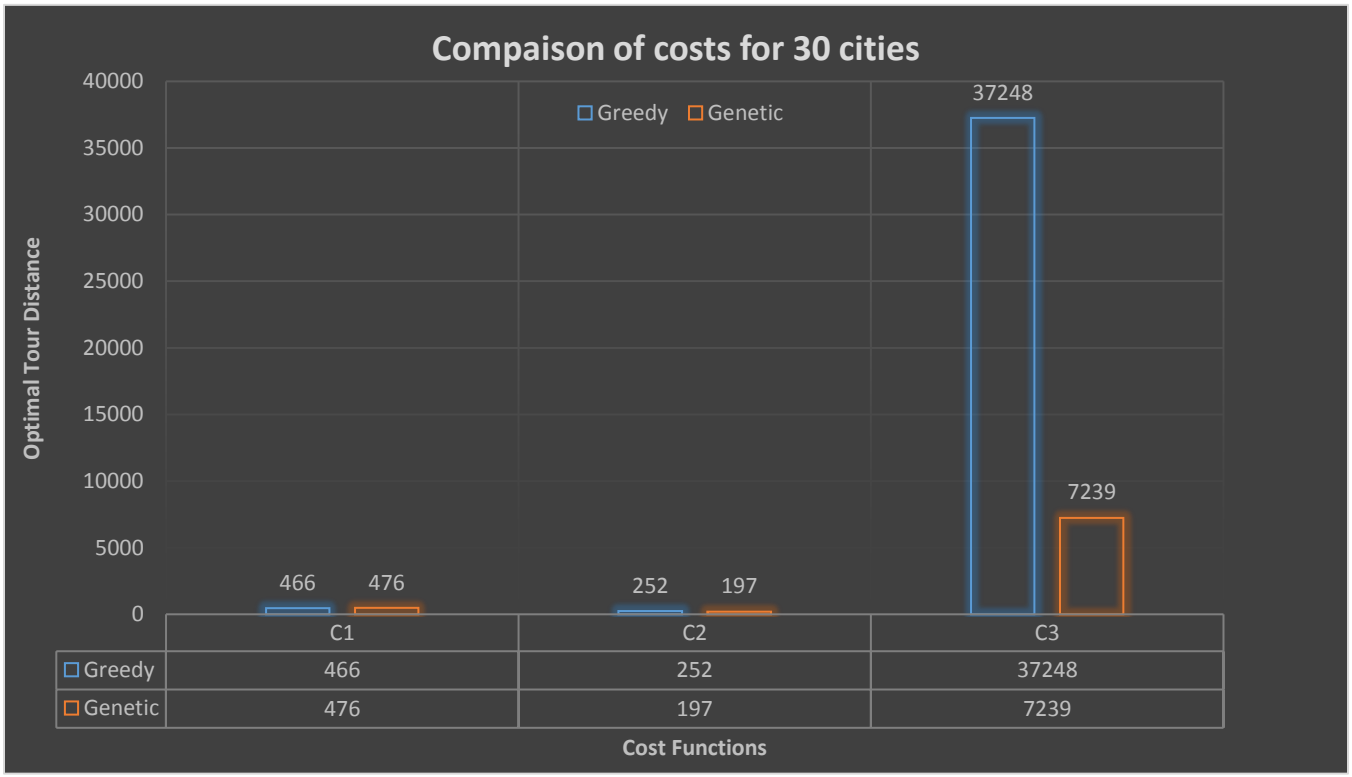

Fig. 6: Histogram that compares the Greedy and Genetic Algorithms' performance for 10 city tour

From this graph, we can conclude that the Genetic Algorithm's performance is:

i. Very high when compared to Greedy Approach for Cost Function C2 and C3.
ii. Slightly less to that of Greedy Approach for other Cost Functions.

3) Now consider Table 1 and Table 2.
From Table 1 and Table 2, take the row with 60 cities as input.
Find Average values of each column of that row in each Table, which are represented in the Histogram's Data Table below.

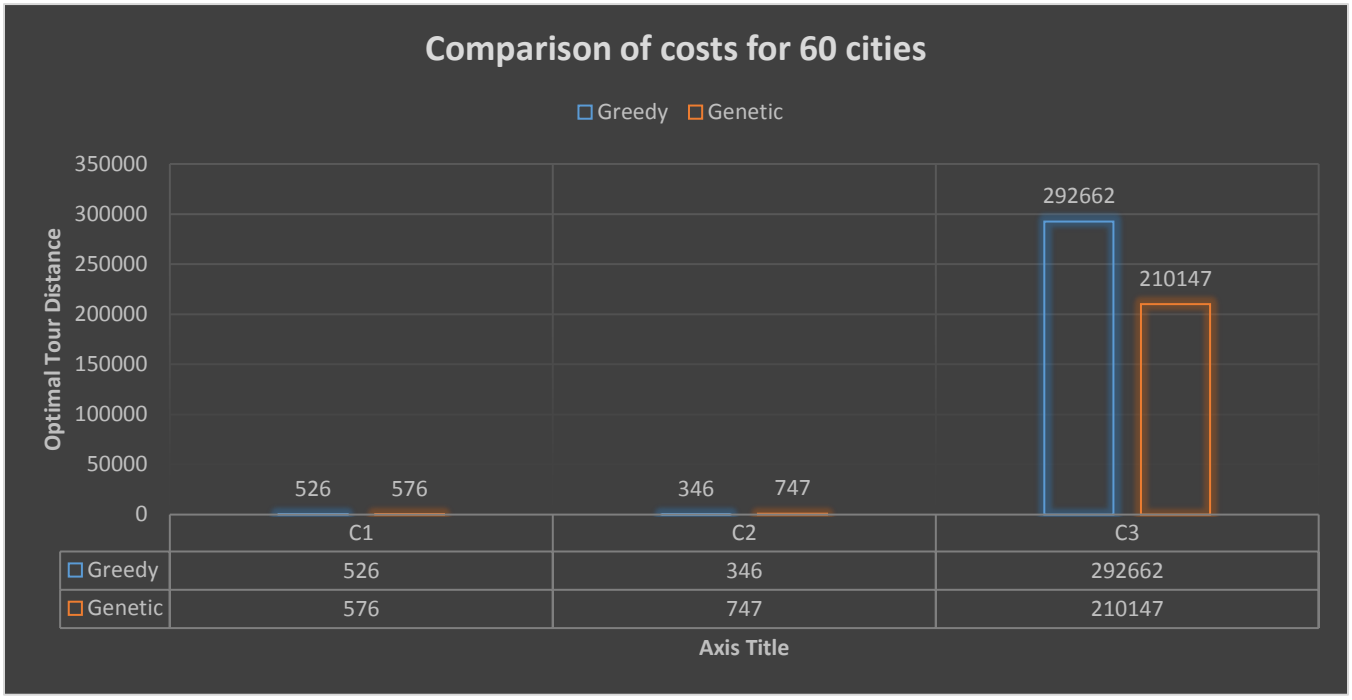

Fig. 6: Histogram that compares the Greedy and Genetic Algorithms' performance for 10 city tour

From this graph, we can conclude that the Genetic Algorithm's performance is:

i. High when compared to Greedy Approach for Cost Function C3.
ii. Slightly less to that of Greedy Approach for Cost Function C1.
iii. Very less to that of Greedy Approach for Cost Function C2.

5. Conclusion:

In this project, the Travelling Salesman Problem is described. Then, two of the most used algorithms to solve it are depicted. Each Algorithm is simulated using the Histograms and compared with each other. Three scenarios are used: 10,30 and 60 cities. It can be concluded that although Genetic Algorithm consumes more number of iterations to solve TSP, its result is closest to the optimum solution. The result is yet feasible with a reasonable number of iterations. On the other hand, the Greedy algorithm finds an alternative path much quickly, is not that efficient with increase in

number of cities. With Cost functions C1 and C3, Genetic algorithm works more efficiently than Greedy Approach. Therefore, it can be concluded that, in large number of nodes, the Greedy Algorithm does not converge to a valid solution when compared to Genetic Algorithm.