11/12/2022

METHODS TO SOLVE THE TRAVELING SALESMAN PROBLEM

Discrete Mathematics for Computing:
Group Project 3

Professor: MANUELE LEONELLI

Group 4: MAX HEILINGBRUNNER, PABLO OSTOS BOLLMANN, AIKATERINI ORLOVA, NICCOLO' MATTEO BORGATO, WENDY QUARSHIE, JOSEPH GUSS

INTRODUCTION

The Travelling Salesman Problem is one of the most common and classic problems in mathematics, optimization, and operations research.

The problem states that a salesman who is appointed to visit several different cities around the country wants to make his trip the shortest possible. He will have to visit each city once and will be able to come back to his home - the starting point - only after he completes all the visits. What is the order of trips that would allow him to minimize the time needed to complete the trip and get home the soonest?

The importance of the Travelling Salesman Problem relies on its use cases. If we were to generalize the prompt given it would be possible for us to think of many day-to-day scenarios where this problem applies and has to be solved. For example, a postal officer every day would love to find the best possible route to deliver all the packages given to him. A manufacturing machine needs to decide the order in which to perform a certain task to minimize energy and time expenses. Even if we were to project ourselves into the future, the traveling salesman will still be useful; for example, in logistics, it would be ideal to know the best possible route in which a drone should fly to maximize the area covered given its energy consumption. Given the importance and application of the salesman problem, we could think that the solution to it would be rather straightforward.

Differently, while the prompt seems simple and the problem important, the complexity of the solution can be surprising and, in some cases, could even be unfeasible to find. When taking into consideration problems with a small number of cities the optimal solution can be found with rather ease by making use of solutions such as the branch and bound methods for integer programming or complete linear programming. Differently, when the number of cities increases it is more common to search for an approximate solution using Heuristic algorithms or Christofides algorithms. This paper presents 3 different solutions to the Travelling Salesman Problem: Complete Linear Programming, Ant Colony System Optimisation, and Genetic Algorithms.

METHODS TO SOLVE THE TSP

COMPLETE LINEAR PROGRAMMING

The TSP can be formulated as a Complete Linear Programming model. Although there are many ways to solve this problem, in this section we will be covering the two most popular formulations, proposed by Miller, Tucker, and Zemlin (MTZ), and by Dantzig, Fulkerson, and Johnson (DFJ). Both approaches use the same mathematical notation:

$$I : set\ of\ nodes$$

$$x_{ij} = \begin{cases} 1 : the\ path\ goes\ from\ node\ i\ to\ node\ j \\ 0 : otherwise \end{cases}$$

$$u_i : a\ dummy\ variable$$

$$c_{ij} > 0 :\ the\ distance\ between\ node\ i\ and\ node\ j$$

Using this set of variables and parameters, we can formulate the objective function of the optimization problem, which aims to minimize the total cost of all travels:

$$min \sum_{i=1}^{n} \sum_{j \neq i, j=1}^{n} c_{ij} x_{ij}$$

MTZ FORMULATION

The complete linear programming model formulated by Miller, Tucker, and Zemlin is subject to the following constraints:

$$\begin{aligned}
&Binary\ integrality: \ x_{ij} \in \{0,1\} \\
&Number\ of\ cities\ visited\ at\ city\ i: \ u_i \in \mathbb{Z} \\
&Enter\ each\ city\ once: \ \textstyle\sum_{i \neq j, i=1}^{n} x_{ij} = 1 \qquad \forall j \\
&Exit\ each\ city\ once: \ \textstyle\sum_{j \neq i, j=1}^{n} x_{ij} = 1 \qquad \forall i \\
&Subtour\ breaking\ constraints: \\
&u_i - u_j + n x_{ij} \leq n-1 \qquad 2 \leq i \neq j \leq n \\
&0 \leq u_i \leq n-1 \qquad 2 \leq i \leq n
\end{aligned}$$

Where the first two constraints reinforce the binary requirement and the type of the variables. The next two constraints stipulate that only one city can be visited from city 'i' and that a given city is visited at exactly one stage of travel, respectively. The last two equations impose that only one route crosses all nodes (Hamilton circuit).

DFJ FORMULATION

The model formulated by Dantzig, Fulkerson, and Johnson takes into account the same variables x_{ij} and parameters c_{ij} and is subject to the following constraints:

$$\begin{aligned}
&Binary\ integrality: \ x_{ij} \in \{0,1\} \\
&Enter\ each\ city\ once: \ \textstyle\sum_{i \neq j, i=1}^{n} x_{ij} = 1 \qquad \forall j \\
&Exit\ each\ city\ once: \ \textstyle\sum_{j \neq i, j=1}^{n} x_{ij} = 1 \qquad \forall i \\
&Subtour\ breaking\ constraints: \\
&\textstyle\sum_{i \in S} \sum_{j \neq i, j \in S} x_{ij} \leq |S| - 1 \qquad \forall S \subsetneq \{1,2,...,n\},\ |S| \geq 2
\end{aligned}$$

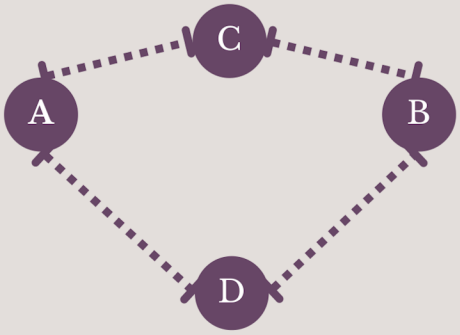
The first three constraints are the same as in the MTZ model, but the last equation implies there cannot be any sub-tours. In other words, the final solution is a unique route, not a combination of smaller sub-tours. In the worst case, we may need 2^n sub-tour breaking constraints, notice how the MTZ model has fewer constraints but is harder to solve.

Both these methods will be able to come up with the optimal solution to the problem. Nevertheless, since TSP is considered to belong to the class of non-deterministic polynomial time complete problems (NP-complete), its time complexity can be overwhelming. That is why, with the aim of obtaining "good" solutions in less time, a great effort has been made to solve this problem using other heuristic methods. This is where the Ant Colony Optimization and Genetic Algorithms come into play.

ANT COLONY OPTIMIZATION

The ant colony optimization method of solving the TSP is quite unique. This optimization technique gets its name from the real world of ants which mysteriously are among the most efficient and optimized living creatures on the planet. Ant colony optimization can be explained by the simple graph below.

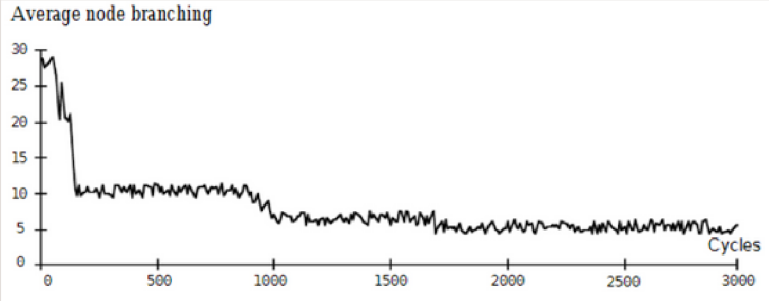
Take point **A** to be the ant colony, point **B** to be a food source, and points **C** and **D** to represent paths to the food source. If we have one ant go through the shorter path **C** and another ant through the longer path **D**, how will the next ants know which path is optimal? When ants find the food source and walk back to the colony they drop pheromones which only last a finite period of time before dissolving into the air. Therefore, as path **C** is shorter than path **D**, it will have a higher concentration of pheromones on it since the pheromones last a finite period of time. Other ants use this signal to then follow the shorter route. This simple process enables ants to have very efficient pathing and resource use. Both of which are conveniently useful for solving the TSP.



The framework for the application of the Ant System to the TSP is as follows. A solution is found by ants moving from city to city until a full tour is completed. In each iteration of the algorithm, every ant moves one city (node), and decisions to move from one node to another are done probabilistically where the probability is based on whether the city has been visited, how connected a city is, and how concentrated the pheromone trail is. Once an ant completes a tour, every edge that it took receives pheromones and the concentration of pheromones received depends on the distance the ant had to travel. Finally, the pheromones decay over time. A common addition to this framework is the introduction of “Elitist ants” to improve the performance of the model. Elitist ants serve the role of reinforcing edges that belong to the best solution. At each iteration, a number of elitist ants are added.

There are various implementations of this algorithm with slight modifications. The first is the introduction of a parameter to adjust the probability of an ant reinforcing an existing path or exploring a new path. This can be used to make the algorithm favor knowledge over exploration or vice versa. Another possible direction is to only allow the ant with the best tour to place pheromones on its path. This would lead to ants searching close to the best-found path.

Ant optimization has been observed to converge quickly to good solutions, however, tends to not converge to a single solution as it continues to produce new solutions that may or may not be better.



While there are certain algorithms specialized for the TSP that would perform better, ant optimization does a good job for a general solution as it can be applied to many complex and dynamic problems and can continue to improve instead of getting stuck in local optima.

GENETIC ALGORITHMS

Metaheuristic search algorithms called Genetic Algorithms are motivated by the mechanism that drives the evolution of life. The Genetic Algorithm is a search technique based on the theory of evolution and natural selection, for finding a solution to combinatorial optimization problems. The algorithm is created to mimic the natural selection process, or "survival of the fittest" of organisms, in order to carry out evolution. The terminology used when talking about the Genetic Algorithm is: gene, chromosome, population, and fitness. Typically, Genetic Algorithms have five steps, which are as follows:

1. Creating the initial population
2. Fitness assessment
3. Selection of the best gene
4. Crossover
5. Mutating to introduce variations

These methods can be used to implement solutions to various optimization challenges. The Travelling Salesman Problem is one such challenge. In order to solve the Travelling Salesman Problem with the Genetic Algorithm the cities are used as genes and a string created using these characters is referred to as a chromosome. A population is targeted using a fitness score that is equal to the combined path length of all the cities stated.

The fitness test that is applied to the population measures the distance between the two gene pools, identifying the chromosome with the shortest path (the cost of the path). The length of the path that the gene describes is how the fitness score is calculated. If needed, the chromosome will be modified to create new children using cross-over and mutate methods:

- *Crossover* – using part of the old solution and combining it with part of a new solution to create a new chromosome
- Mutate – swaps out solutions using random indexes

These methods are used to test the various combinations of potential solutions in order to come up with the most optimal one – the gene with the shortest journey. All the genes in the gene pool that are the fittest make it through the population test and onto the following iteration. The value of a cooling variable determines the number of iterations that need to be performed in order to determine the shortest

path. After a predetermined number of iterations, the cooling variable's value stops declining and reaches a threshold.

For the purpose of this task let's pretend there are 3 cities, and the salesman is in city 0 and he has to find the shortest route to go through all the cities and end up in city 0 again. For this problem, we would first have to create a permutation featuring an integer representation of a route between the 3 cities. The fitness calculation will let us know the total distance between all cities. Next is a selection stage that is done at random and selects an individual on the basis of the fitness level calculated prior to this stage. By switching around the genes or portions of the chromosomes, the crossover of two-parent strings generates offspring (new solutions). A genetic operator called crossover is used to change a chromosome's programming from one generation to the next. Two strings are randomly selected from the mating pool to produce superior offspring and crossed. On the other hand, mutation - the final stage of the problem, produces new solutions by flipping some of the digits in a string. Finally, the mutation is used to keep a population of Genetic Algorithm chromosomes genetically diverse from one generation to the next, in similarity with biological mutation.

After a number of successive generations the algorithm finds the approximately optimal solution, this is due to the fact that even if the algorithm provides an optimal solution, we cannot demonstrate or prove the optimality of the Genetic Algorithm unless there is a strong lower bound that closely fits the answer obtained.

CONCLUSION: REAL WORLD APPLICATIONS, DRAWBACKS, AND COMPARISON

Among the presented heuristic and metaheuristic algorithms, the Travelling Salesman Problem can also be solved with various other types of algorithms, such as branch and bound or Las Vegas algorithms. In reality, all of the previously mentioned algorithms can be applied:

1. Complete Linear Programming: Multi-Stop Routing in Apple Maps (small input)
2. Ant Colony Optimization: UPS Driver Delivery Route (high input)
3. Genetic Algorithm: Order Pick-up Route at Amazon Warehouse (medium input)

To compare the performance of the three presented algorithms, we are going to take into account multiple factors for benchmarking: the accuracy of the optimal solution, the time complexity (number of operations), the space complexity (amount of memory incorporated), the structural complexity of the algorithm, and the order of growth (Big-O notation). All three algorithms can be reflected in reality.

	Accurate Solution	Time Complexity	Space Complexity	Structural Complexity	Order of Growth (O)
Complete Linear Programming	●●●	●●●	●●○	●○○	●●●
Ant Colony Optimization	●●○	●○○	●●○	●●○	●○○
Genetic Algorithms	●●○	●●○	●●●	●●●	●●○

The Complete Linear Programming algorithm, either by DFJ or MTZ, is probably one of the most known algorithms for solving the Travelling Salesman Problem. However, depending on the input size, Linear Programming can become inefficient in reaching an optimal solution compared to other evolutionary methods such as Ant Colony Optimization or Genetic Algorithms.

For bigger input sizes, both methods, the Ant Colony Optimization and Genetic Algorithms are more suitable to accurately predict an optimal solution efficiently. However, both algorithms differ in their time, space, and structural complexity. For Genetic Algorithms, the size of the chromosome lengths is based on the number of nodes while the population of each child in each generation is based on the mutation and crossover. In contrast, the Ant Colony Optimization algorithm moves more quickly through the given nodes, and each population size is based on heuristics.

Hence, it can be concluded that the Ant Colony Optimization algorithm is more time-efficient, uses less memory, and consists of a lower structural complexity while both are able to find, in most cases, an optimal solution.

To highlight the structural complexity and better understand the logic of Genetic algorithms, this paper is going to compare a real-world implementation of code to the theoretical concept (see attachments). This comparison will not only simplify the algorithmic paradigm but also show how effective an implementation can be to find an optimal solution.