EXPLORING THE TRAVELLING SALESMAN PROBLEM

MATHEMATICS IN INTELLIGENT SYSTEMS

PROJECT WORK PRESENTED BY

ROHITH G [CB.EN.U4AIE19026]

ABSTRACT

The travelling salesman problem (TSP) is a widely talked about optimisation path finding problem. There are different types of this problem – the main ones being Search Algorithm and Optimisation Algorithm. The former is an NP Complete problem while the latter is an NP Hard problem because the most optimal solution for the most optimised path is hard to determine without traversing through all the paths possible. Hence we normally arrive at a sub optimal solution, but this is also what the real world requires most of the time.

In the genetic algorithm problem, due to natural selection, only the fittest survive from the given population. Some get mutated and die and some individuals survive with their fitness score through multiple generations. This is mapped and solved using this travelling salesman problem, finding (optimising) the fittest individual from the given population.

Also this problem can be reduced to being based on integer linear programming using some graph defining constraints. These are the topics presented in this work and explained upon.

INTRODUCTION

This work discusses the travelling salesman problem, one of the most used optimisation problems. The problem is discussed and the genetic algorithm is solved based on this method. The genetic algorithm is based off of Charles Darwin's famous quote, "Survival of the Fittest." This showcases how the individuals in a population decrease over time due to their fitness score and only the strongest ones survive. This is optimised using the TSP algorithm. But this is not the final solution because TSP – Optimisation basically is an NP Hard problem. We cannot arrive at the most optimal solution without trying out all the solutions possible. Hence we can only arrive at a suboptimal solution graph.

Then this idea of solving the travelling salesman problem is extended to Integer Linear Programming (ILP) where we define constraints for the graph using binary values, that specify the movement of the agent in and out of the nodes. Also, the GG TSP method is discussed that helps solve the sub-tour problem in a graph.

NON-DETERMINISTIC PROBLEMS (NP)

If it is simple to verify a problem's solutions, it is said to be in NP. A problem, on the other hand, is NP-hard if it is difficult to solve or discover a solution to. If a problem is both in NP and NP-hard, it is now called NP-complete. As a result, NP-completeness has two crucial, straightforward features. It's simple to verify, but difficult to find answers.

Based on the sort of TSP problem we're dealing with, there are two forms of NP difficulties.

TSP and TSP-OPT are distinguished by the fact that the former is a search problem while the latter is an optimization problem. The notion of verifying the solution of a search problem makes sense, and in the case of TSP, it is easy to do, therefore it is NP-complete. But we don’t know an efficient way to verify a solution for TSP-OPT, we cannot say that it is in NP, thus we cannot say that it is NP-complete.

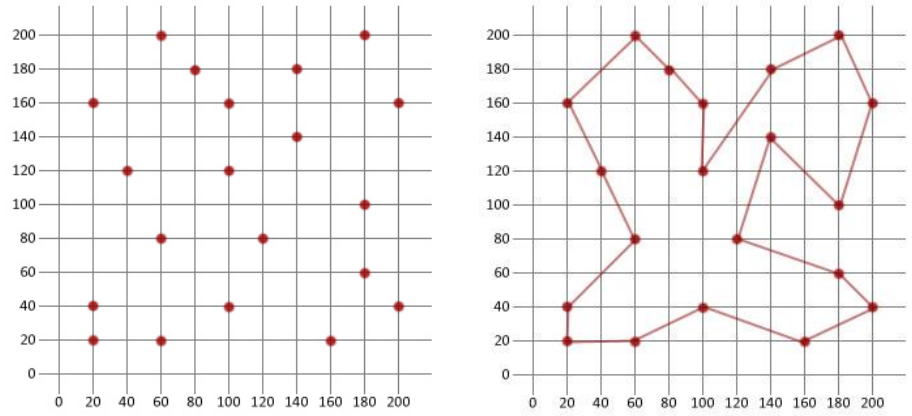
THE TRAVELLING SALESMAN PROBLEM

"Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the initial city?" asks the travelling salesman problem.

Even in its purest form, the TSP has various applications, including planning, logistics, and semiconductor manufacturing. It appears as a sub-problem in several domains, such as DNA sequencing, after being slightly adjusted. The notion city represents consumers, soldering spots, or DNA fragments in various applications, whereas the concept distance represents journey times or cost, or a similarity measure between DNA fragments.

The travelling salesman decision problem has a polynomial-time deterministic verifier. Given a connected graph G and a cost k and a candidate (a cycle that hits every vertex and costs less than or equal to k) C, we can quickly verify that C has the same number of vertices as G and that its cost is less than or equal to k. This is the decision version of the problem, TSP-SEARCH.

But we do not have any known polynomial-time verifier for the optimization version of the problem, TSP-OPTIMIZE. This is because, given a path, it is very difficult to verify that it truly is the lowest-possible cost without checking all other possible Hamiltonian cycles and making sure none of them cost less.



GENETIC ALGORITHMS

A genetic algorithm is a search heuristic based on Charles Darwin's natural selection hypothesis. This algorithm mimics natural selection, in which the fittest individuals are chosen for reproduction in order to create the following generation's children.

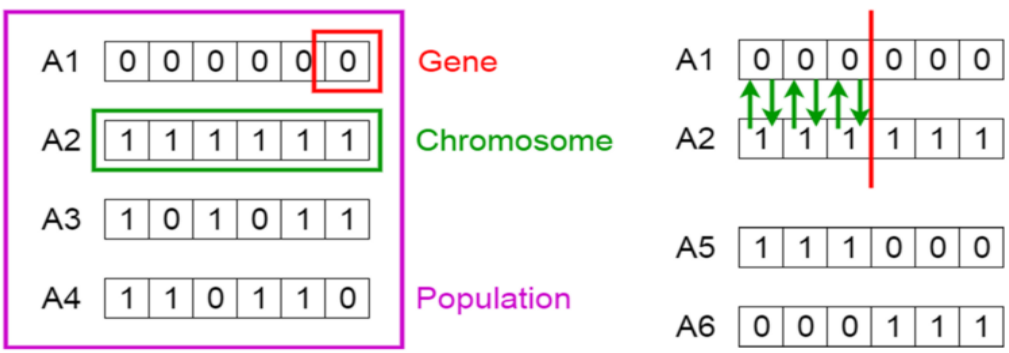
Natural selection begins with the selection of the most fit individuals in a population. They generate offspring who inherit the parents' qualities and are passed down to the next generation. If parents are physically active, their children will be fitter than they are and have a better chance of surviving. This procedure will continue to iterate until a generation of the fittest individuals is discovered.

This notion can be applied to a search problem. We consider a set of solutions for a problem and select the set of best ones out of them.

Five phases are considered in a genetic algorithm.
1. Initial population
2. Fitness function
3. Selection
4. Crossover
5. Mutation

The procedure begins with a group of individuals referred to as a Population. Each person is a potential solution to the problem at hand. Genes are a set of factors or traits that distinguish all individuals. Genes are linked together to form a Chromosome, which is the solution.

The set of genes of a person is represented by a string in terms of an alphabet in a genetic algorithm. We take the genes on a chromosome that we encode.



Fitness

The fitness function affects how fit an individual is, as well as their capacity to compete with others and ultimately survive. Each individual is given a fitness score. The fitness score determines the likelihood of an individual being chosen for reproduction.

Selection

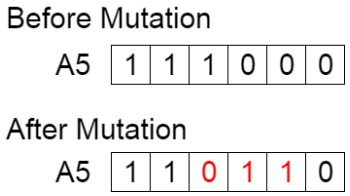
The selection phase selects the healthiest individuals and allows them to carry their genes down to future generations. Based on their fitness scores, two pairs of people (parents) are chosen. Individuals who are physically fit have a better probability of being chosen for reproduction.

Crossover

A genetic algorithm's most important phase is crossover. A crossover point is picked at random from within the DNA for each pair of parents to be mated. The kids are formed by exchanging the genes of the parents until they reach the crossover point. Finally, the population is replenished with these new offspring.

Mutation

Some of the genes in some new kids can be susceptible to a mutation with a low random frequency. This means that some of the bits in the bit string can be switched around. Mutation happens to maintain population variety and avoid premature convergence.



If the population has converged, that is, if no offspring are considerably different from the previous generation, the algorithm will terminate. The genetic algorithm is thus considered to have generated a set of solutions to our problem. The phases are repeated in order to develop individuals who are better than the previous generation in each new generation.

GENETIC ALGORITHM FOR THE TRAVELLING SALESMAN PROBLEM

Taking inspiration from natural selection, genetic algorithms (GA) are an amazing approach to solving search and optimization problems.

Here we create a population of candidates for the TSP solutions. **The fitness function is taken as the cost of the tour. Since a low-cost tour is necessary as a result**, this becomes a problem

of minimization. The cities are visited one by one on this tour until all of them have been visited once, and then we return to the first city to complete the cycle. Each tour is chosen based on fitness — the least expensive tours are the most fit.

This problem is started by redefining some definitions that was provided in the genetic algorithm section.

1. Gene: a city (represented as (x, y) coordinates)
2. Individual (aka "chromosome"): a single route satisfying the conditions above
3. Population: a collection of possible routes (i.e., collection of individuals)
4. Parents: two routes that are combined to create a new route
5. Mating pool: a collection of parents that are used to create our next population (thus creating the next generation of routes)
6. Fitness: a function that tells us how good each route is (in our case, how short the distance is)
7. Mutation: a way to introduce variation in our population by randomly swapping two cities in a route
8. Elitism: a way to carry the best individuals into the next generation

Creating the Genetic Algorithm Code:

First, a city class is built, which allows us to create and manage cities. These are the coordinates (x, y). Distance is calculated in the City class. The fitness class is then formed. Here, fitness is defined as the inverse of route distance. Because we want to reduce route mileage, a higher fitness score is preferable. Our first population has been produced. To accomplish so, we'll need a mechanism to develop a function that generates routes that meet the requirements. To construct an individual, we visit each city in a random order. Because we desire the complete population, an entire route algorithm is constructed. Everyone is now ranked depending on their fitness score.

The mating pool has now been chosen, and a fitness proportionate selection has been implemented. The probability of selection is determined by the fitness of each individual in comparison to the population. The use of elitism is another design component to consider. With elitism, the population's top performers will automatically pass down to the next generation, guaranteeing that the most successful people survive.

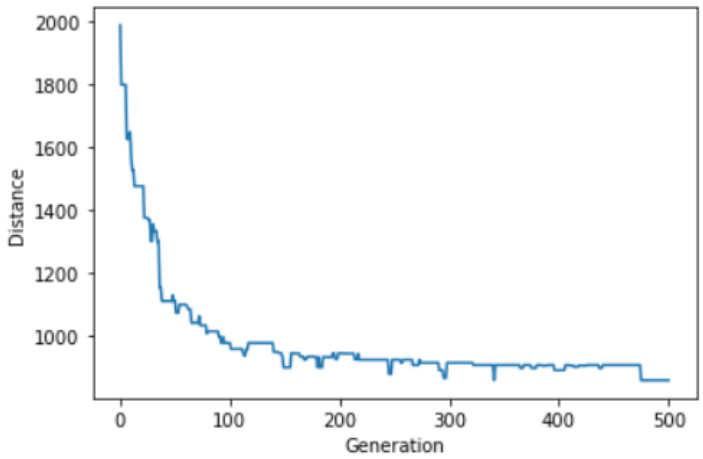
With our mating pool created, we can create the next generation in a process called crossover (aka "breeding"). If our individuals were strings of 0s and 1s and the two rules didn't apply, we could simply pick a crossover point and splice the two strings together to produce an offspring.

However, the TSP is unique in that we need to include all locations exactly one time. To abide by this rule, we can use a special breeding function called ordered crossover. In ordered crossover, we randomly select a subset of the first parent string and then fill the remainder of

the route with the genes from the second parent in the order in which they appear, without duplicating any genes in the selected subset from the first parent.

This is now generalized to create the offspring population. Elitism is used to retain the best routes from the current population. Then, the breed function is used to fill out the rest of the next generation. Now to simulate the mutation part of the algorithm swap mutation method is used. With specified low probability, two cities will swap places in our route.

This is now repeated for multiple generations until only the fittest ones survive.



The Implemented Code is found in the following link:
https://drive.google.com/file/d/1jYbsnmcKI93uZ8iquzfRDdrY8LAOkL2p/view?usp=sharing

EXTENDING TSP TO INTEGER LINEAR PROGRAMMING

The TSP can be formulated as an integer linear program. For the problem we see that it is based on undirected graphs. Hence we can develop constraints about how easily we can arrive at a solution. Suppose the agent is moving from node i to node j. Then the value for (i,j) can be updated to 1 and (j,i) will always be 0 because the same path shouldn't be visited again through the same edge.

Hence if we notice, in general when we formulate this problem, we notice that the following formula always holds true:

$$f(i,j)+f(j,i)\le 1$$

Hence following this we can formulate the following as the equation for the nodes that enter the current node:

$$For \text{ each node i}$$
$$\sum_{j:\ (j,i)\in G'} F(j,i)=1$$

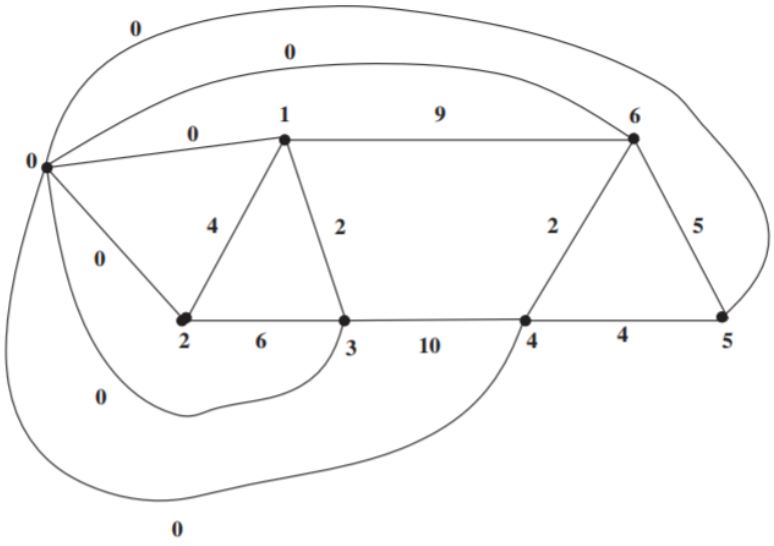
Also the equation defining the edge that is take to leave from the current node is given below:

$$For \text{ each node i}$$
$$\sum_{k:\ (i,k)\in G'} F(i,k)=1$$

Hence, overall, the ILP formulation that allows us to choose the proper minimum cost path entering and leaving a given node is as follows:

$$Minimize \sum_{undir\ edge(i,j)\in G'} \left[D(i,j)\left(F(i,j)+F(j,i)\right)\right]$$

Now this might look like this solves our problem but there arises the sub-tours that might occur. The starting and the ending nodes should be the same globally, but sub-tours are created when this is taken locally. Lets consider the given graph shown:



In this the constraints can be defined as

f(1,3) = 1
f(3,2) = 1
f(2,4) = 1
f(4,1) = 1

and again three more constraints defined by
f(5,7) = 1
f(7,6) = 1
f(6,5) = 1

In the graph, these two now form two sub-tours, which is undesirable. As a result, these equational limitations are required but not sufficient. As a result, we move on to the search type problem, in which the agent is given a fixed weight value that can be spent on costs while traversing each path, as long as it visits all nodes in the network and returns to the starting node.

Hence we resort to a complete and compact ILP formulation for the TSP on a given graph G. This abstract ILP formulation is called the GG TSP formulation, named after the two authors, B. Gavish and S. Graves. The GG formulation is called **complete** because, in contrast to an approach we will discuss later, its solution completely solves the instance of the TS Tour Problem it is based on. The GG formulation is considered compact because the number of variables and inequalities is relatively small – proportional to the number of edges of the graph G.

$$\texttt{Minimize} \sum_{\text{edge } (i,j) \text{ in } G'} D(i,j) \times F(i,j) + D(i,j) \times F(j,i),$$

such that
For each edge (i,j):

$$F(i,j) + F(j,i) \leq 1.$$

For each node j, from 0 to n:

$$\sum_{i \neq j} F(i,j) = 1,$$

$$\sum_{i \neq j} F(j,i) = 1.$$

For each node j from 1 to n:

$$b(0,j) = n \times F(0,j).$$

For each edge (i,j):

$$b(i,j) \leq n \times F(i,j),$$

$$b(j,i) \leq n \times F(j,i).$$

For each node j from 1 to n:

$$\sum_{\text{edge}(i,j)} b(i,j) - \sum_{\text{edge}(i,j)} b(j,i) = 1.$$

All the F variables are binary whereas all the b variables are

integrals with values between 1 and *n*. *b(j)* shows the number of packets/weight given to the agent aver it visits every node. Each *b(i,j)* variable is associated with a traversal from node i to node j through the edge *(i,j).*

VALIDATING THE GG TSP FORMULATION

Suppose, for a contradiction argument, that in a feasible solution to a concrete GG formulation, the values of the F variables specify two (or more) disjoint cycles in G. No node can be in more than one cycle, since every node is entered and exited exactly once. So, one of those cycles, say C, must not contain node 0. Now pick any node j on C. By inequalities defined for f(i,j), F(i, j) has value 1 for exactly one node i in G; and F(j, k) has value 1 for exactly one node k in G; and both nodes i and k are on C. Combined with the inequalities relating b(i,j) and f(i,j), it implies that b(h, j) has value 0 for any node h other than i; and that b(j, h) has value 0 for any node h other than k. Then, the last inequality defined imply that

b(i,j) = b(j,k) + 1
=> b(i,j) > b(j,k)

Consider a walk around cycle C that starts at node j and travels from node j to node k through edge (j, k). The values of the b variables must decrease for each consecutive edge on this walk, based on what we learned in the preceding paragraph and the fact that j was an arbitrary node on C. However, b(i, j) must be the smallest b value of any edge on C when the walk returns to node j. As a result, b(i, j) < b(j, k), contradicting the preceding conclusion that b(i, j) > b(j, k). As a result, no plausible GG TSP solution can define two or more cycles, hence the only viable GG solution is a TS Tour on G.

Finally, the b value can be defined as the weight dropped by the agent when it visits a node. In other words, the agent carrying n weights will not traverse an edge unless it is used to visit or exit a node. It's also the inequalities that tell us that the agent has one more weight value when it enters a node than when it leaves it. As a result, a TS tour must be defined by a set of b and F variables that meet inequalities in the GG formulation.

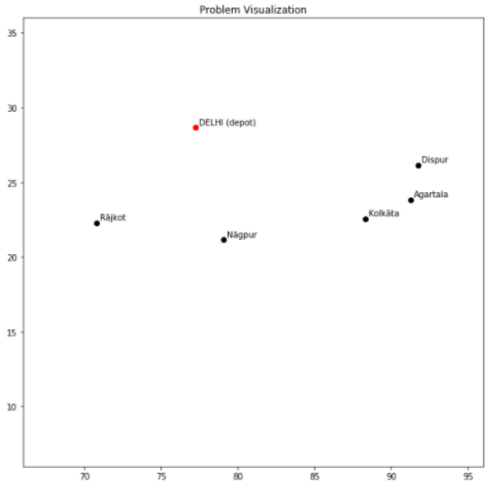
Then, the objective function enforces that the tour will be optimal, i.e., have the smallest cost possible.

FAMOUS FORMULATIONS FOR THE TSP PROBLEM USING ILP

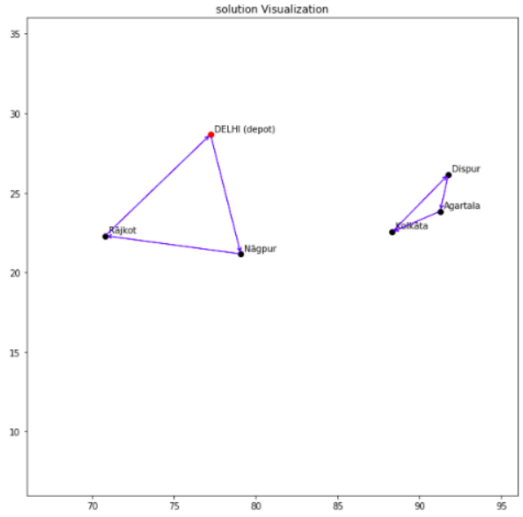
Let's take an example where we take six cities and we plot them on a graph and try to create a TS Tour for all of them avoiding the sub-tour problem. But based on the values given, the machine doesn’t understand much because the constraints given for a normal TSP problem isn’t sufficient. Hence we get sub-tours.

```
----------data-------------
      CITy        x        y
0    Delhi  77.2300  28.6600
1   Rājkot  70.7984  22.2969
2   Nāgpur  79.0831  21.1539
3  Kolkāta  88.3378  22.5411
4 Agartala  91.2667  23.8333
5   Dispur  91.7700  26.1500
----------------------------
```

The city dataset



Mapping the cities on the graph



Sub-tour problem

There are two methods present for sub tour elimination that we use. These were one of the basic ILP constraint formulations done before.

1. MTZ Method for subtour elimination

After the three authors, C. Miller, R. Tucker, and R. Zemlin, it is known as the MTZ formulation. The MTZ formulation employs U variables instead of b variables, with U(i) referring to the position (an integer from 1 to n + 1) of node I in the G tour. Because we added node 0 to G when generating graph G, the tour contains n + 1 nodes.

The interpretation is that ui < uj implies city i is visited before city j. Hence it is safe to say that values for ui equals to the number of edges present in the tour. Also based on ILP we can say

$u_j \geq u_i + 1$ if $x_{ij} = 1$.

But this is insufficient as it should also hold true when x(i,j) = 0, causing a contradiction. Hence it is reformulated as follows:

$u_j + (n-2) \geq u_i + (n-1)x_{ij}$ for all distinct $i, j \in \{2, \ldots, n\}$

The ui variables ensure that a single tour visits all cities by increasing by (at least) 1 for each step along the tour, with a drop allowed only when the tour passes through city 1. Every tour that does not pass through city 1 would break this limitation, hence the only way to satisfy it is for the tour that passes through city 1 to also pass through all other cities.

The MTZ formulation of TSP is thus the following integer linear programming problem:

$$\begin{aligned}
&\min \sum_{i=1}^{n} \sum_{j \neq i, j=1}^{n} c_{ij} x_{ij}: \\
&\qquad x_{ij} \in \{0,1\} && i, j = 1, \ldots, n; \\
&\qquad u_i \in \mathbf{Z} && i = 2, \ldots, n; \\
&\sum_{i=1, i \neq j}^{n} x_{ij} = 1 && j = 1, \ldots, n; \\
&\sum_{j=1, j \neq i}^{n} x_{ij} = 1 && i = 1, \ldots, n; \\
&u_i - u_j + (n-1)x_{ij} \leq n-2 && 2 \leq i \neq j \leq n; \\
&\qquad 1 \leq u_i \leq n-1 && 2 \leq i \leq n.
\end{aligned}$$

The first set of equalities stipulates that each city is reached from exactly one other city, while the second set stipulates that each city departs from precisely one other city. The final limitation requires that there be only one tour that covers all cities, rather than two or more disjointed excursions that merely cover all cities collectively. To demonstrate this, it is proven that every possible solution has only one closed sequence of cities, and that there are values for the dummy variables ui that meet the criteria for every single tour spanning all cities.

To prove that every feasible solution contains only one closed sequence of cities, it suffices to show that every subtour in a feasible solution passes through city 1 (noting that the equalities ensure there can only be one such tour). For if we sum all the inequalities corresponding to x(i,j) = 1 for any sub tour of k steps not passing through city 1 we get the following contradiction.

$(n-1)k \leq (n-2)k$

It now must be shown that for every single tour covering all cities, there are values for the dummy variables ui that satisfies all constraints. We now choose ui = t if city i is visited in step

t(i, t = 1,2,3,...,n). Then

ui – uj <= n-2

ui can't be greater than n.
uj can't be lesser than 2.
This satisfies all constraints for x(i.j) = 0
As for x(i,j) = 1 we have the following defined equation.

$$u_i - u_j + (n-1)x_{ij} = (t) - (t+1) + n - 1 = n - 2,$$

2. DFJ Method for subtour elimination

It is generally referred to as the DFJ formulation, after the three authors, Dantzig, Fulkerson and Johnson. Here we label the cities with numbers varying from 1 to n.
The DFJ formulation is as follows:

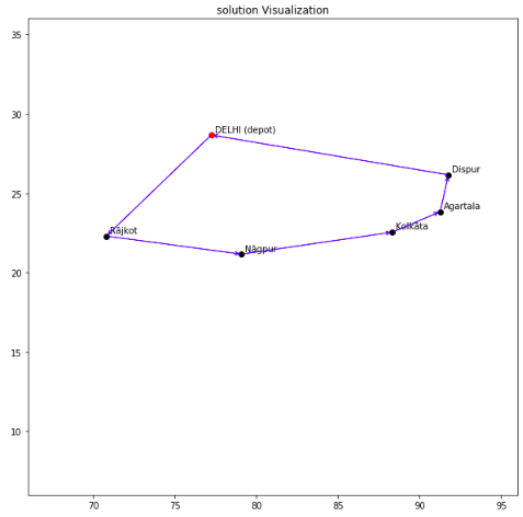
$$\min \sum_{i=1}^{n} \sum_{j \neq i, j=1}^{n} c_{ij} x_{ij}:$$

$$\sum_{i=1, i \neq j}^{n} x_{ij} = 1 \qquad j = 1, \ldots, n;$$

$$\sum_{j=1, j \neq i}^{n} x_{ij} = 1 \qquad i = 1, \ldots, n;$$

$$\sum_{i \in Q} \sum_{j \neq i, j \in Q} x_{ij} \leq |Q| - 1 \qquad \forall Q \subsetneq \{1, \ldots, n\}, |Q| \geq 2$$

The last constraint of the DFJ formulation ensures no proper subset Q can form a sub-tour, so the solution returned is a single tour and not the union of smaller tours.



Final output from MTZ and DFJ methods

Comparison of the models

MTZ's method imposes n^2 constraints (one for each pair (i,j) where i,j = [1..n]), whereas DFJ's method imposes subtour constraints for all feasible sets of locations, i.e. 2^n for n locations. As a result, MTZ's method adds a polynomial number of restrictions, whereas DFJ's method adds an exponential number.

The MTZ technique introduces n new decision variables (titi for i = [1..n]) in terms of decision variables. However, DFJ does not add any additional decision variables. To reach an optimal answer, MTZ's approach just needs to be solved once. While DFJ is usually implemented as a modified version that is iteratively solved (LP model has to be solved multiple times with new subtour constraints added every time).

There is no clear winner among the two. For some problems, DFJ gives solutions faster than MTZ and for some problems, MTZ is faster. But DFJ has an efficient branch and bound approach due to which it becomes more efficient than MTZ. Also, MTZ's formulation is weaker i.e the feasible region has the same integer points but includes more fractional points.

INFERENCE AND CONCLUSION

In this work the most famous NP Hard optimisation algorithm was discussed and worked on. Its variations were looked upon and explained. Then the genetic algorithm was fully explained and solved for specifying the fittest individuals using TSP. Further on this concept of obtaining the most optimal path, minimizing the cost was taken a step forward and explained using ILP formulations. ILP formulations look to solve the sub-tour problems by introducing more constraints in a manner that they don't contradict the other defined constraints. For these two famous methods were seen: the MTZ and the DFJ methods and their ILP formulations were given. Finally, they were compared on which one was the better formulation algorithm that helps us minimise the cost and find the most optimum path. This work can pave the way to more inspired path finding algorithms that can come up in the future that eliminates this NP Hard problem and make TSP – optimize, an NP-complete problem.

REFERENCES

[1] D. L. Applegate, R. E. Bixby, V. Chvatal, and W. J. Cook. The traveling salesman problem: a computational study. Princeton university press, 2006.

[2] J. Bruna, W. Zaremba, A. Szlam, and Y. Lecun. Spectral networks and locally connected networks on graphs. In the International Conference on Learning Representations, 2014.

[3] G. Agarwal and D. Kempe. Modularity-maximizing graph communities via mathematical programming. The European Physical Journal B, 66:409–418, 2008.

[4] R. Agarwala, D. L. Applegate, D.Maglott et al. A fast and scalable radiation hybrid map construction and integration strategy. Genome Research, 10:350–364, 2000.

[5] F. Alizadeh, R. M. Karp, D. Weisser, and G. Zweig. Physical mapping of chromosomes using unique probes. Journal of Computational Biology, 2:159–184, 1995.

[6] D. Applegate, R. Bixby, V. Chvatal, and W. Cook. The concorde TSP solver, 2003. Retrieved from www.math.uwaterloo.ca/tsp/concorde/index.html (last accessed January 24, 2019).