OPTIMIZATION FORMULATIONS OF TRAVELLING SALESMAN PROBLEM

Done By:

Deepthi Sudharsan (CB.EN.U4AIE19022)

Sajja Tarun Teja (CB.EN.U4AIE19055)

Sannidhi Sri Sai Hanuma (CB.EN.U4AIE19057)

Vijai Simmon S (CB.EN.U4AIE19068)



B.TECH CSE-AI, CEN DEPARTMENT, AMRITA SCHOOL OF ENGINEERING

# ACKNOWLEDGEMENT

This project would not have been possible without the support and guidance of Mr. Sachin Kumar S, who has provided us with sufficient knowledge and guidelines even during these unprecedented and trying times.

We would also like to thank our Computer Science and Engineering (Artificial Intelligence) department for giving us this opportunity to nurture and hone our skills.

Furthermore, we would like to thank the Amrita Vishwa Vidyapeetham management for providing ample resources to avail our project needs and a platform for online lectures during this lockdown.

TABLE OF CONTENTS

[ACKNOWLEDGEMENT 2](#_Toc57832720)

[ABSTRACT 5](#_Toc57832721)

[TRAVELLING SALESMAN PROBLEM 6](#_Toc57832722)

* [TSP USING NAIVE AND DYNAMIC PROGRAMMING 6](#_Toc57832723)

[PYTHON COMPUTATION AND OUTPUT 9](#_Toc57832724)

* [TRAVELLING SALESMAN PROBLEM USING BACKTRACKING 11](#_Toc57832725)

[PYTHON COMPUTATION AND OUTPUT 12](#_Toc57832726)

[TRAVELLING SALESMAN PROBLEM IN BIOINFORMATICS 14](#_Toc57832727)

* [ANT COLONY OPTIMIZATION (ACO) 15](#_Toc57832728)
* [GENETIC ALGORITHM (GA) 16](#_Toc57832729)

[TRAVELLING SALESMAN PROBLEM USING GENETIC ALGORITHM 16](#_Toc57832730)

* [FITNESS FUNCTION AND SELECTION 17](#_Toc57832731)
* [CROSSOVER AND MUTATION 18](#_Toc57832732)

[SINGLE POINT CROSSOVER 18](#_Toc57832733)

[SWAP MUTATION 19](#_Toc57832734)

* [NEW POPULATION (NEXT GENERATION) 20](#_Toc57832735)

[PYTHON COMPUTATION AND OUTPUT 21](#_Toc57832736)

[ADVANTAGES OF TSP USING GA 26](#_Toc57832737)

[DISADVANTAGES OF TSP USING GA 26](#_Toc57832738)

[TRAVELLING SALESMAN PROBLEM USING ILP IN GENOMICS 26](#_Toc57832739)

* [INTEGER LINEAR PROGRAMMING 26](#_Toc57832740)

[TSP USING ILP FOR GENETIC ALGORITHM 31](#_Toc57832742)

[PYTHON COMPUTATION AND OUTPUT 31](#_Toc57832741)

[TSP USING ILP FOR STRING RECONSTRUCTION 31](#_Toc57832742)

[PYTHON COMPUTATION AND OUTPUT 31](#_Toc57832741)

[CHALLENGES 33](#_Toc57832743)

[INDIVIDUAL CONTRIBUTIONS: 34](#_Toc57832744)

[RESOURCES 35](#_Toc57832745)

# ABSTRACT

**Optimization** is finding the best/optimum output by maximizing or minimizing a given function such that it also satisfies certain constraints. Suppose, we are given the area of a lawn and we also know that it’s perimeter is as small as possible and we have to find the dimensions of the lawn. This is an example of optimization.

One of the most famous NP-hard optimization problems is the **Travelling Salesman Problem**.The shortest optimum distance/cost-efficient path a salesman can take given the cities he has to visit exactly once before returning to the starting point is the most basic visualization of this problem.

The Travelling Salesman Problem (TSP) has been able to influence and enhance many other algorithms and fields. One of the many applications the TSP has these days is under the field of bioinformatics; TSP can be applied in genomics, in genome assembly, in DNA synthesis etc. The underlying principles in these applications remain the same. Given a genome, using the TSP approach we can follow the path through the genome and find the shortest genetic distance.

*Keywords*: Travelling Salesman Problem (TSP), Genetic Algorithm, Optimization

OPTIMIZATION FORMULATIONS OF TRAVELLING SALESMAN PROBLEM

# TRAVELLING SALESMAN PROBLEM

The Travelling Salesman problem (TSP) can be approached using different methods. To name a few methods, the problem can be approached using Integer Linear Programming, Backtracking, Branch and bound, Naïve and Dynamic method of approach, etc. To begin with the basic understanding of TSP, let us look at the concept and computation of TSP using Naive method of approach and Backtracking.

# TSP USING NAIVE AND DYNAMIC PROGRAMMING

Problem is to find the shortest possible route/distance that visits every city exactly once and returns to the starting point when the cities and distance between the cities are given.

Here, the Hamiltonian Tour exists to find the shortest route because the graph is complete and many such tours exist, the problem is to find a Hamiltonian cycle with minimum weight.

Naive Approach:

1. Consider the first city as the starting and ending point.
2. Generate all permutations (n-1)! of all cities.
3. Calculate the minimum cost of every permutation.
4. Result as the permutation with the minimum cost.

Formula used:

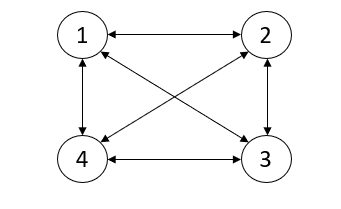


i -----------> starting vertex

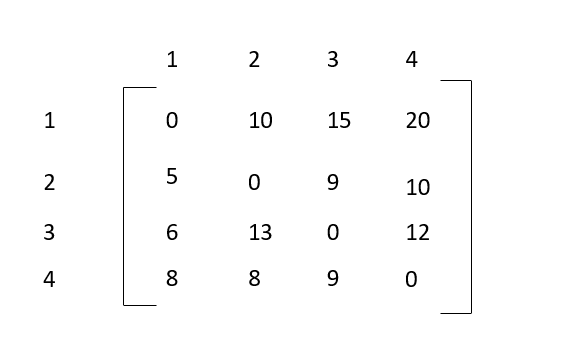
s-----------> set of vertices

-------> cost of edge from starting vertx

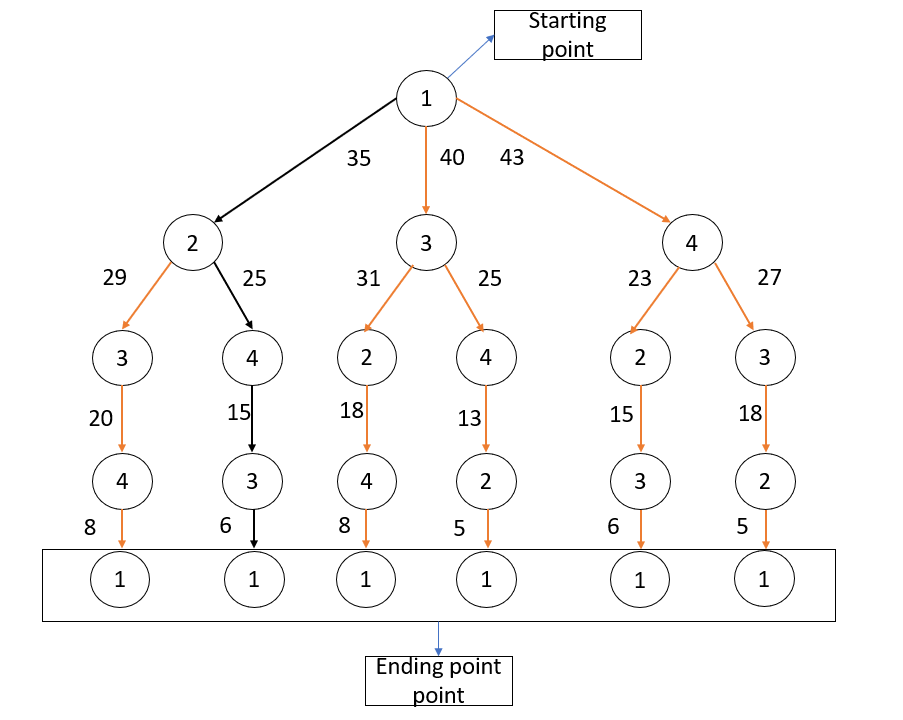
For example:



Weighted directed graph

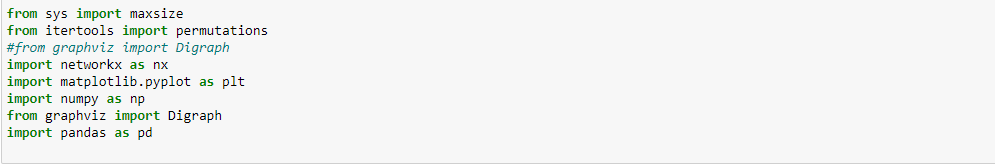


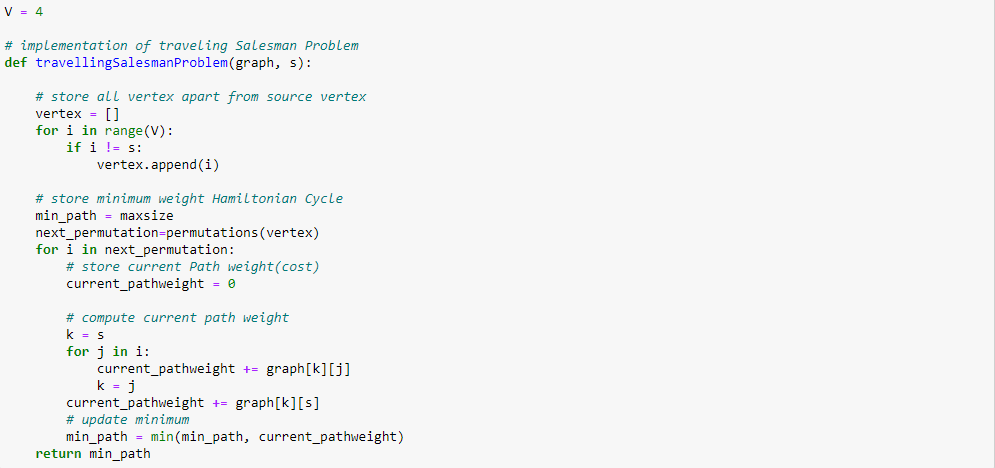
Adjacency Matrix for the graph

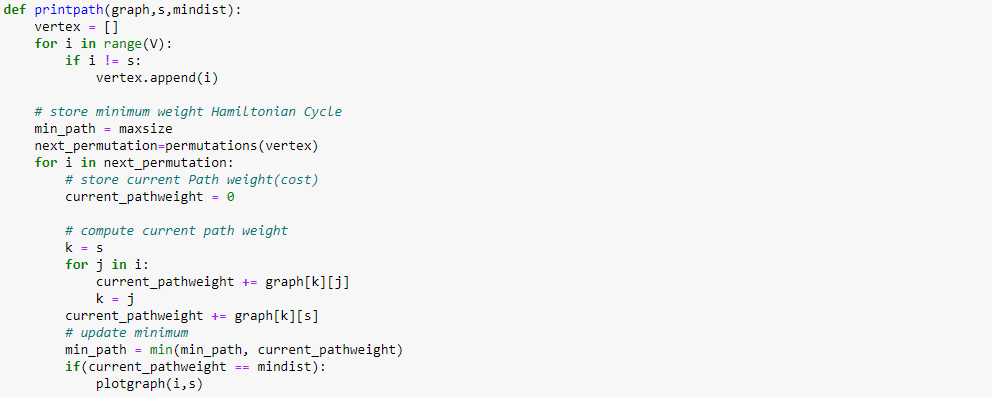


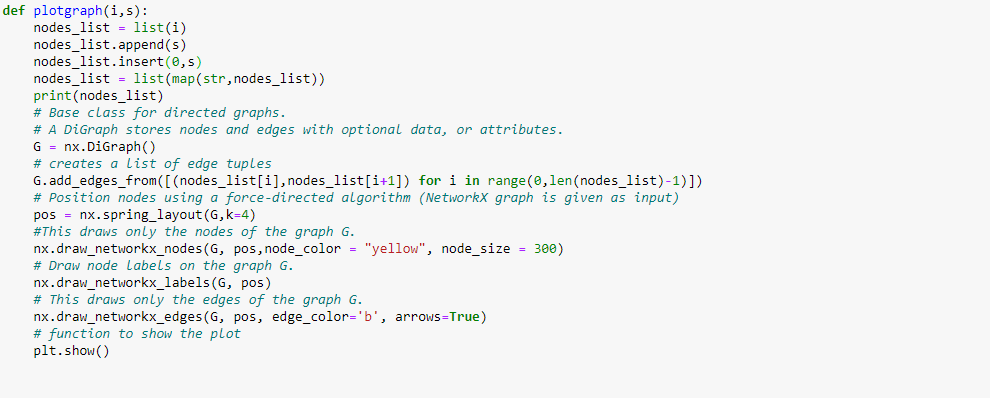
From the above graph, node 1 as the starting point and the ending point and by using a naive and dynamic method the edges with black color is the shortest route that visits every node exactly with minimum cost as 35.

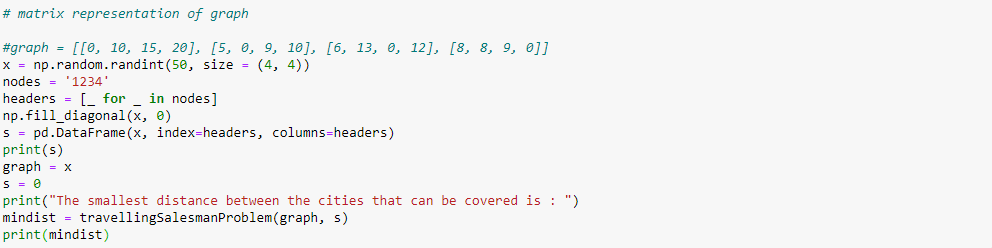
# PYTHON COMPUTATION AND OUTPUT





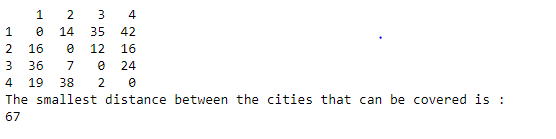


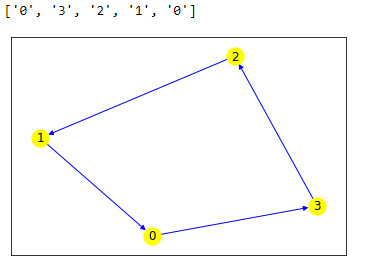






Output:





# TRAVELLING SALESMAN PROBLEM USING BACKTRACKING

Back tracking is an algorithmic method of resolving problems repeatedly by trying to create a solution, one piece at a time, removing those solutions that fail to satisfy the problem problems at any given time (at a time, here, referring to the past and reaching any level of the search tree). For example, consider the problem of solving Sudoku, we try to fill in the digits one by one. Whenever we find that the current digit cannot lead to a solution, we remove it (which is backtrack) and try the next digit. This is better than the naive approach (creating all the digital combinations and trying all the combinations one by one) as it discards a set of permissions whenever it backtracks.

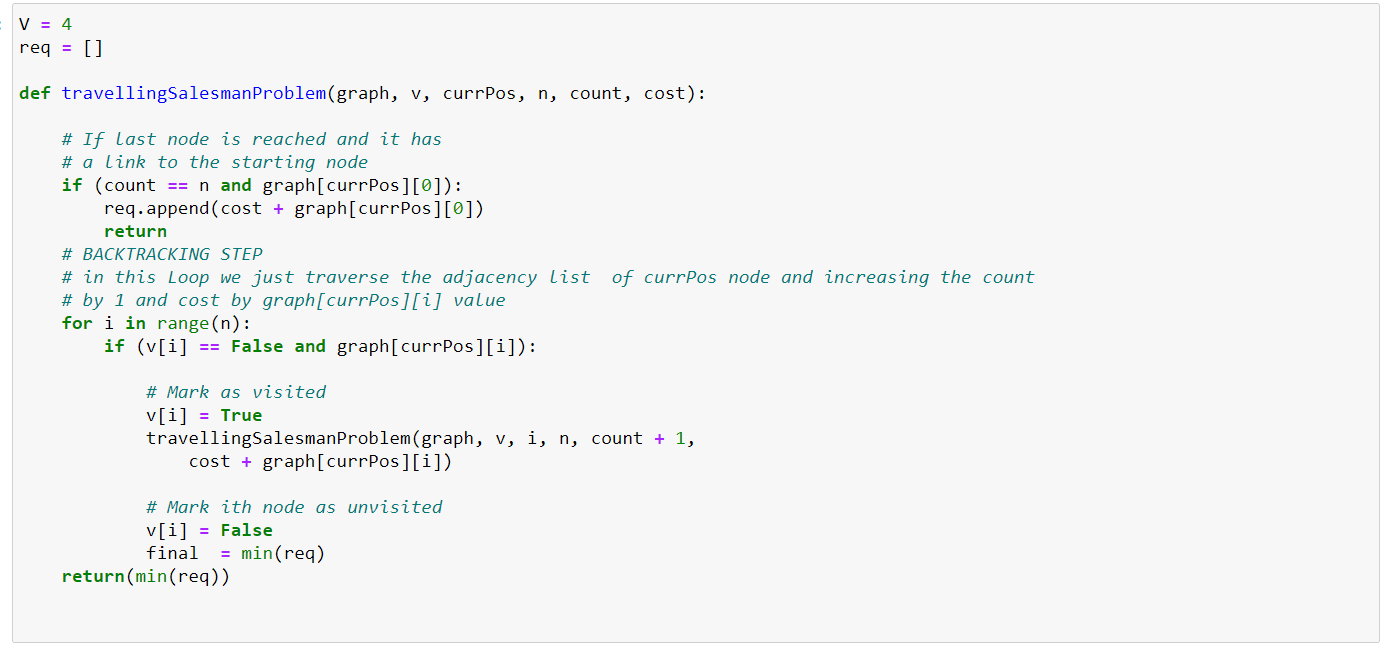
**Approach:**

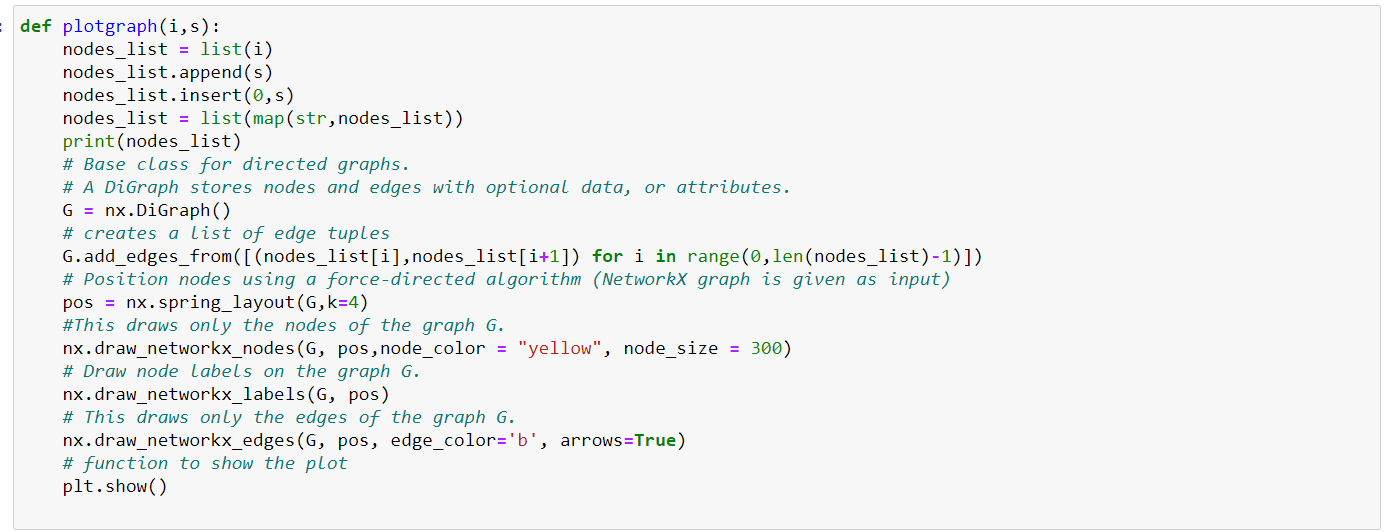
 In this post, implementation of a simple solution is discussed.

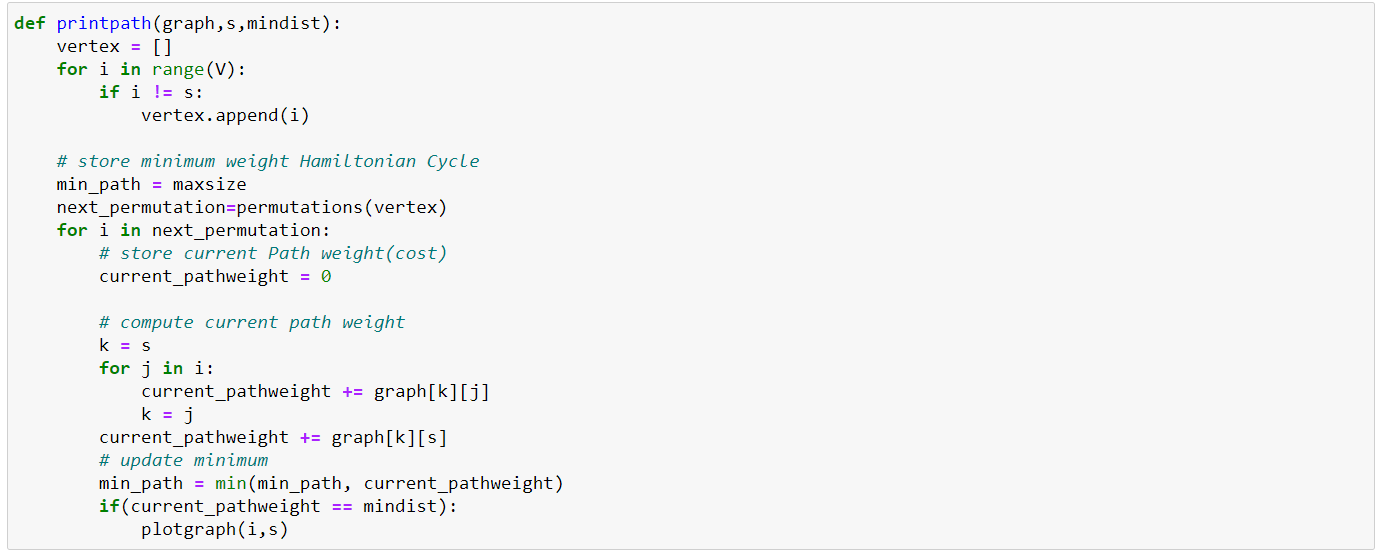
* Let’s Consider city 1 (which is 0th node) as the starting and ending point. where the route is cyclic, we can consider any point as starting point.
* So, we Start traversing from the source (starting node) to its adjacent nodes in DFS manner.
* Calculate cost of every traversal and keep track of minimum cost and keep on updating the value of minimum cost stored value.
* Finally return the minimum cost.

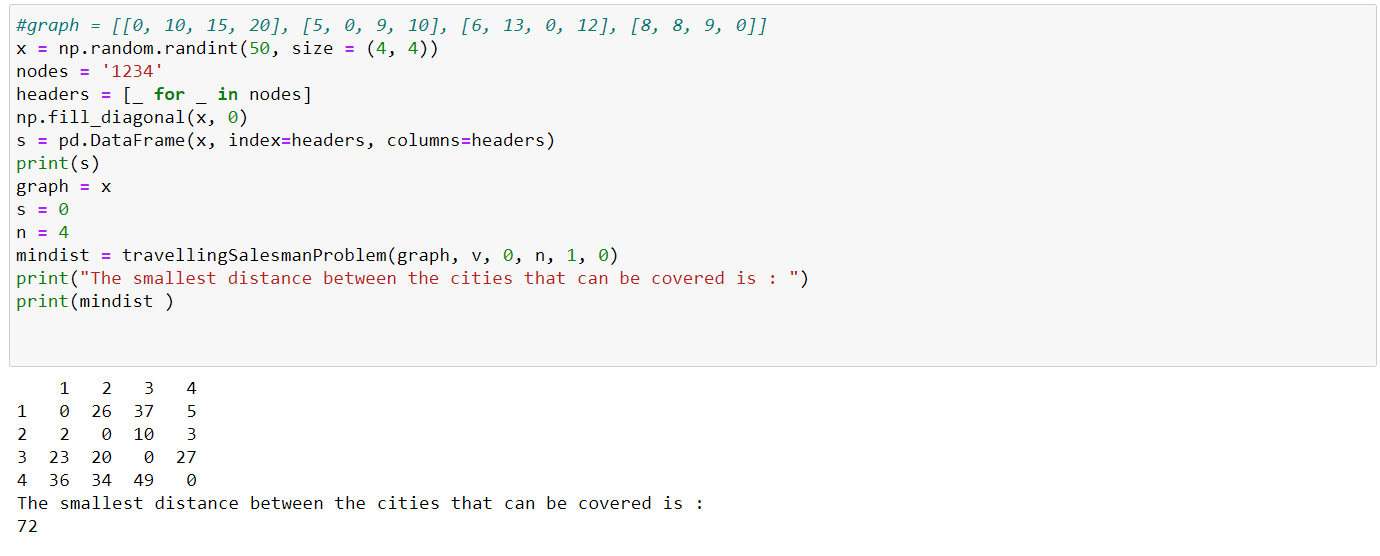
# PYTHON COMPUTATION AND OUTPUT

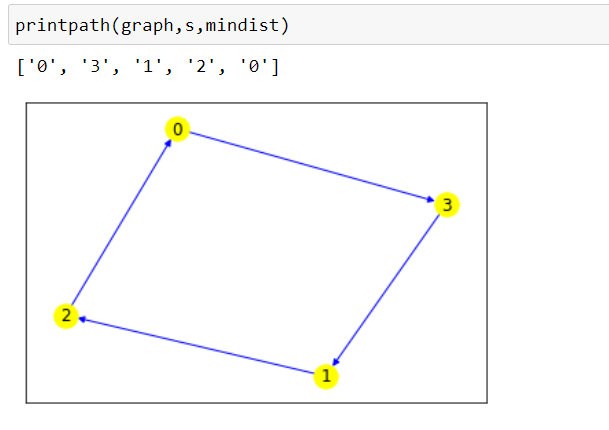












# TRAVELLING SALESMAN PROBLEM IN BIOINFORMATICS

There are many heuristic methods and algorithms for solving optimization problems. Some of the popular ones in bioinformatics that are solvable using the TSP approach are Ant Colony Optimization (ACO) Problem and Genetic Algorithm (GA) problem. This project focuses only on TSP using GA.

## ANT COLONY OPTIMIZATION (ACO)

The main idea behind ACO is to simulate the real behavior of ants. The ant colonies function based on the indirect communication with the help pheromones, which ants excrete, to find the most optimum path that has been found by other ants that leads them to their food source from their hill. Pheromones are chemical substances which attract these insects to search for food.

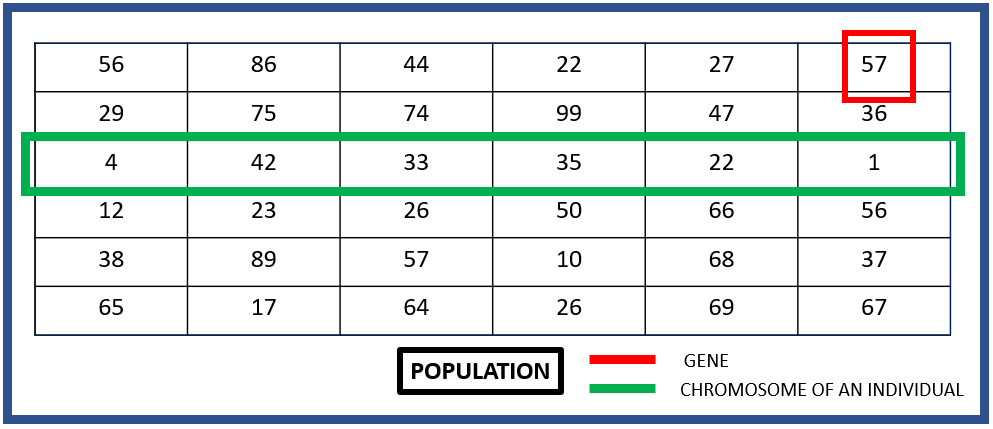
The quantity of pheromones depends on the attractiveness of the path. The use of a more attractive route ensures that the ant exudes more pheromones on its way back and so that the attractive path is traceable for the other ants. One of the important characteristics of pheromones is that over a period, it evaporates, and the power of the pheromones are more. So, the main criteria of the Ant colony optimization algorithm are the movement of ants. In some simulations that were conducted, it is known that ants can find the shortest and best route to a food source from their anthill. Ants do not move continuously; they move in jumps, which means that, after some time, they will always be in another graph node. The absolved path is saved in the ant’s memory so that the created cycles are detected in their memory. In the next tour, the ant decides based on the pheromones’ power. Pheromones on shortest edges are stronger because of the property of pheromone evaporation, due to this fact that the ant goes across these edges faster.

## GENETIC ALGORITHM (GA)

Inspired by Charles Darwin’s theory of evolution, came the genetic algorithm concept. It is based on the concept of “survival of the fittest”. Genetic algorithm reflects the process of natural selection based on the fitness of individuals, and the selected fit individuals give rise to the next generation population.

# TRAVELLING SALESMAN PROBLEM USING GENETIC ALGORITHM

Let us consider an initial population of 6 individuals.



In our python computation, we have created this initial population using ‘random.sample’ to generate unique values in the chromosome of each individual.

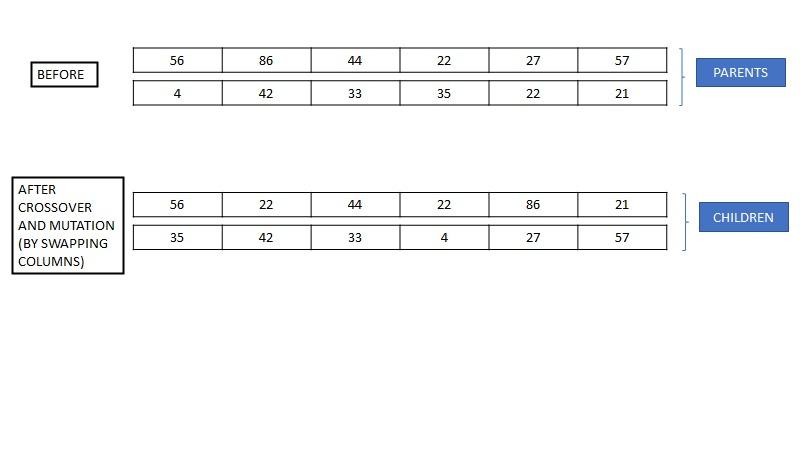
These unique values in the chromosomes are called alleles ( A gene is one element position of a chromosome. Allele is the value a gene takes for a particular chromosome).

## FITNESS FUNCTION AND SELECTION

Given this population, we must select the individuals based on their fitness. So, for the fitness calculation, we must define the fitness function, so in our case we defined the fitness function as the row sum. If the row sum is minimum, we claim that the fitness of that individual is maximum. And the (Population Size)/2 number of fittest individuals are taken to crossover and mutate. Here, we take 6/2 = 3 fittest individuals as parents to crossover. Now the number of selected individuals, = 3 (here), so the number of ways we can choose two parents out of these 3 individuals that have been selected are,.

## CROSSOVER AND MUTATION

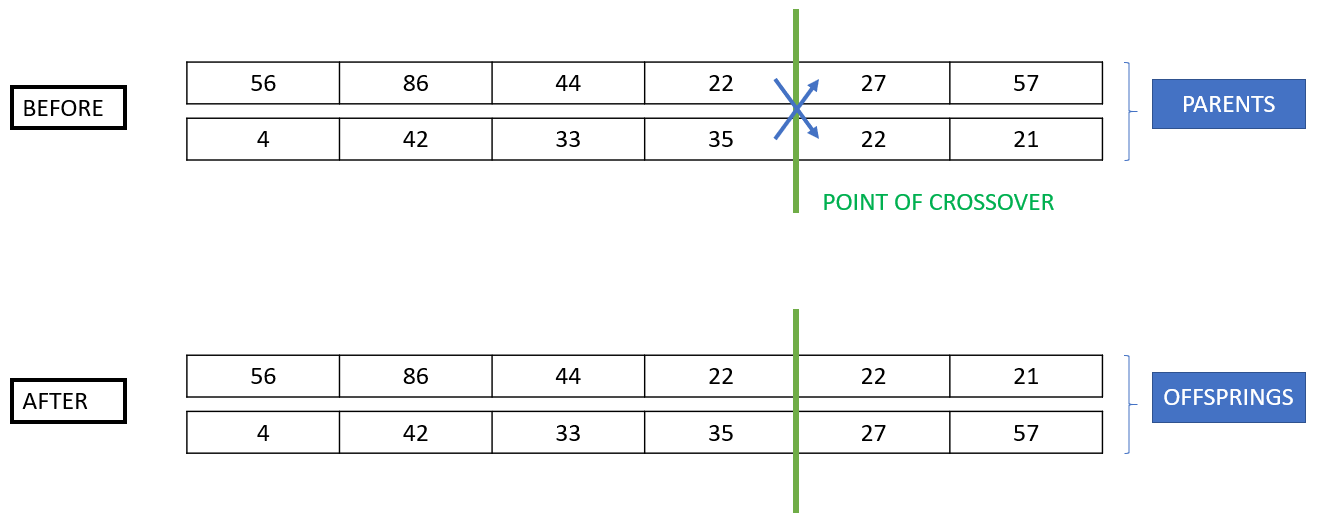
Now that we have the different combinations of parents that can be taken for crossover, we will cross over the chromosomes of the parents which will give rise to 2 offspring, and the chromosomes of these 2 new offspring from each parent pair are mutated. In our python computation, we are using single point crossover and swap mutation.



### SINGLE POINT CROSSOVER

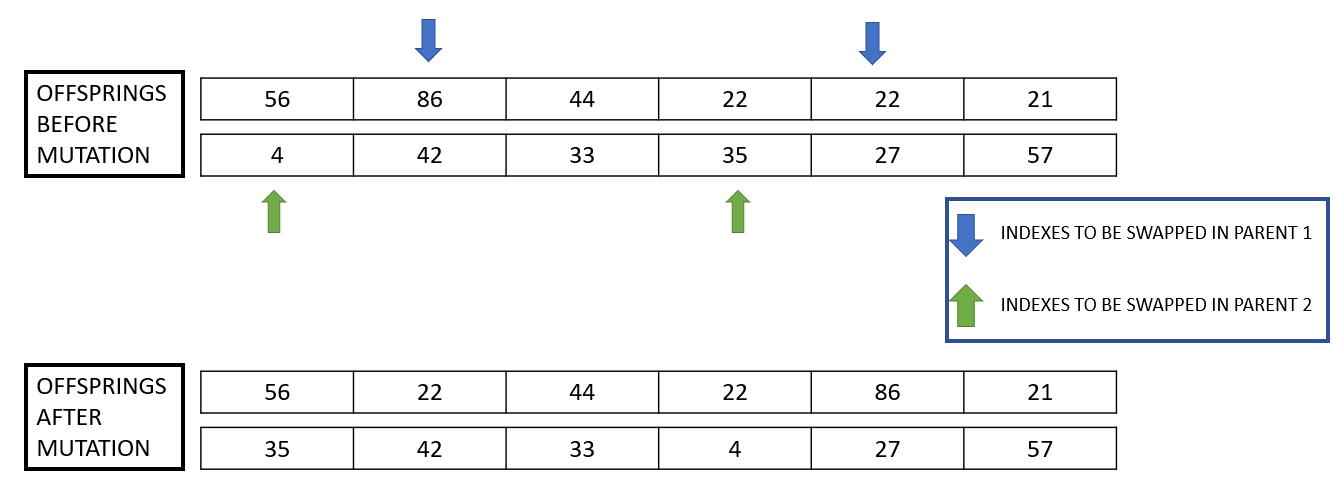
In a single point crossover, a point or index is chosen and the genes after that point of index of the two parents are swapped to give rise to a new individual. In our python computation, we have randomly generated this point or index and then we performed the crossover.

Let us now consider 2 parents from the initial population that we have at hand to see how the crossover works.



### SWAP MUTATION

The alteration in the genome of an organism is called mutation. To bring about this alteration in the nucleotide sequence of the genome, we must alter and tweak the offspring’s chromosomes. Now that we have the offspring of the 2 parents, we will perform swap mutation (as used in our python computation) to bring about this alteration. In swap mutation, two indices are taken and the values in those indices are swapped to create the mutation.



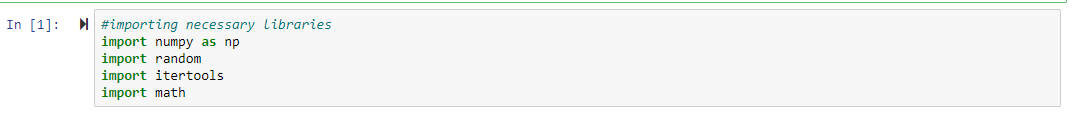
### NEW POPULATION (NEXT GENERATION)

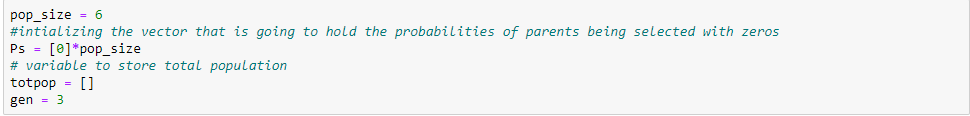
The offspring that we get from each combination of parents will constitute the new population and for the total number of generations we want, the new population that we get in each iteration will be fed as the initial population to the fitness calculation function, selection and so on and the process continues.

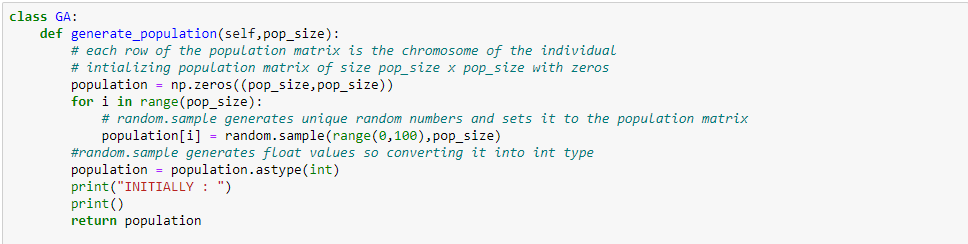
One special case that we come across during this process is when the population size is odd. When we have an odd number of individuals in the population, during the selection process, the (Population Size)/2 is not a whole number, so what we do is we round it off to the highest closest whole number and we move forward. Since we are rounding it off to the closest highest number, while taking and getting 2 offspring from each combination, we get one extra offspring than the population size, so now to remove one extra offspring what we do is that we compare the fitness of all the offspring that we got and we remove the offspring with the least fitness of them all. This way, our new population size becomes the same as the old population size / required population size.

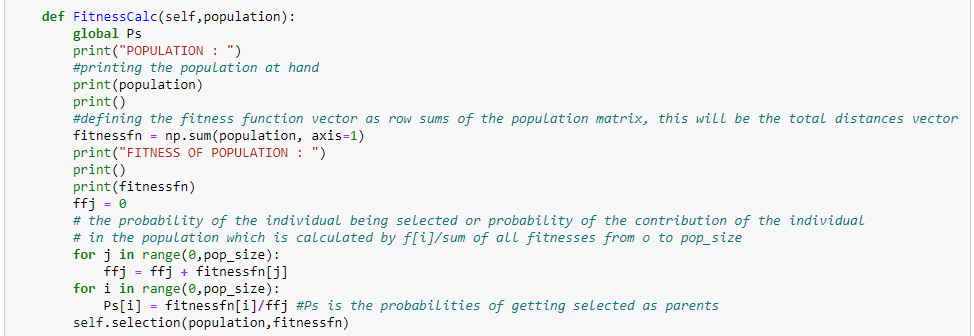
### PYTHON COMPUTATION AND OUTPUT

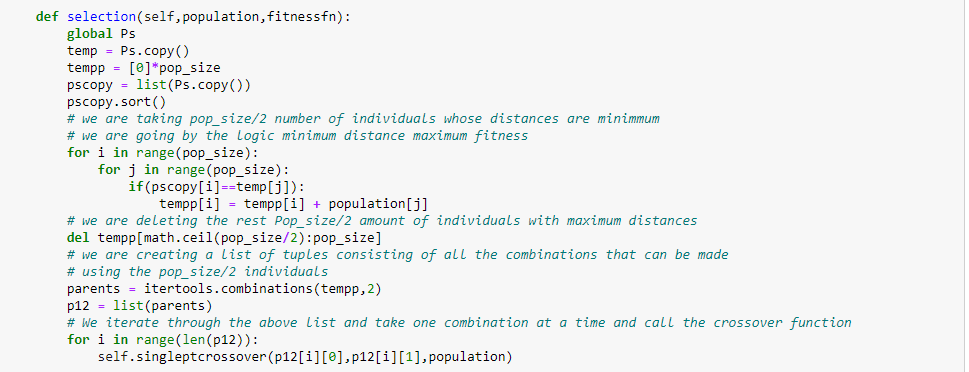
Let us look at this approach with the help of its python computation.

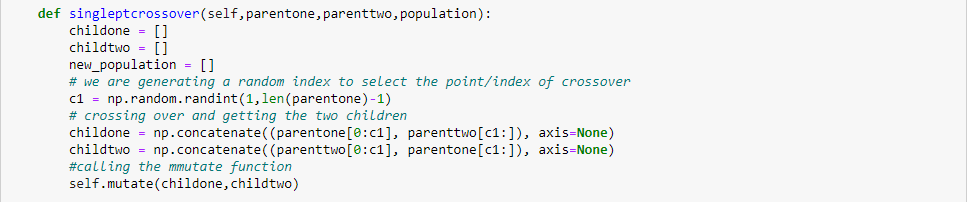


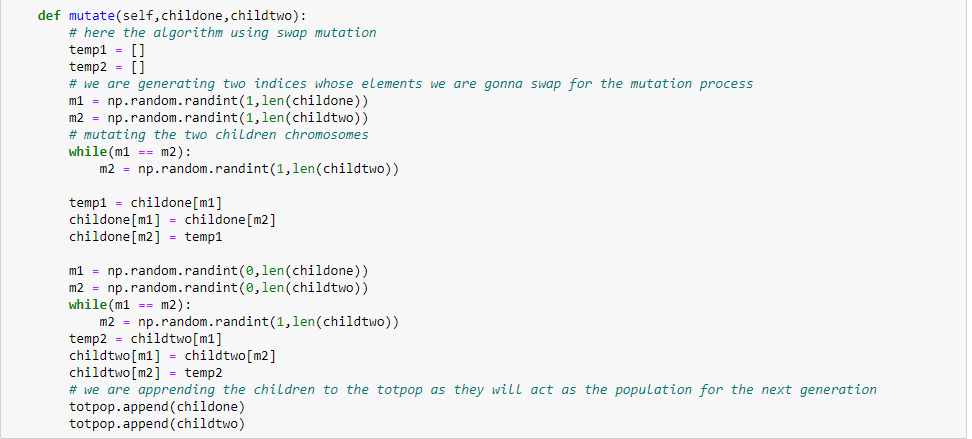




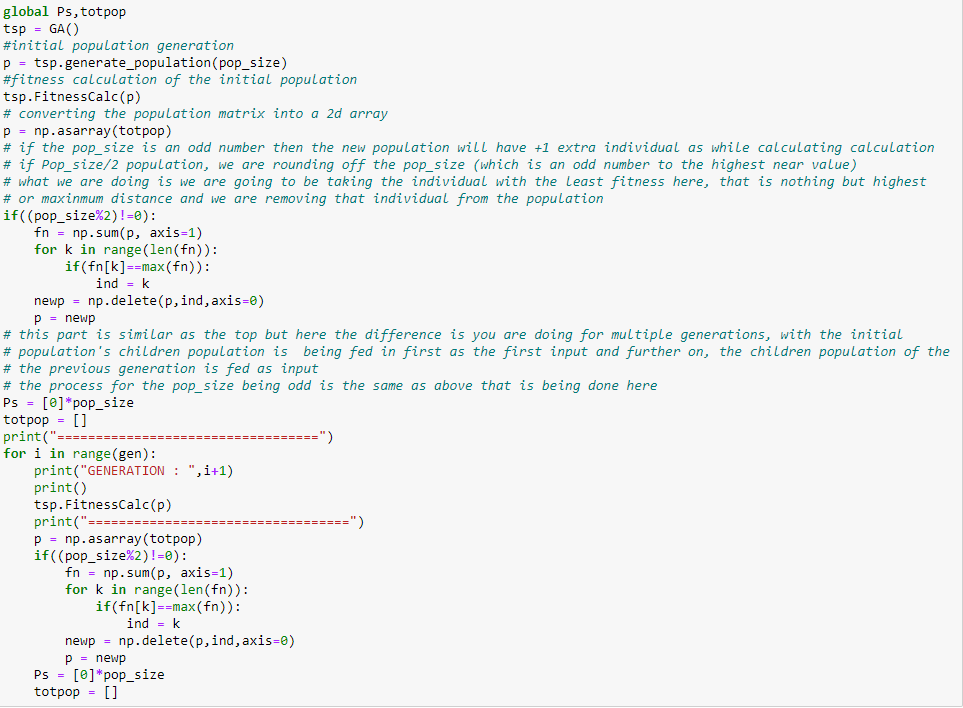




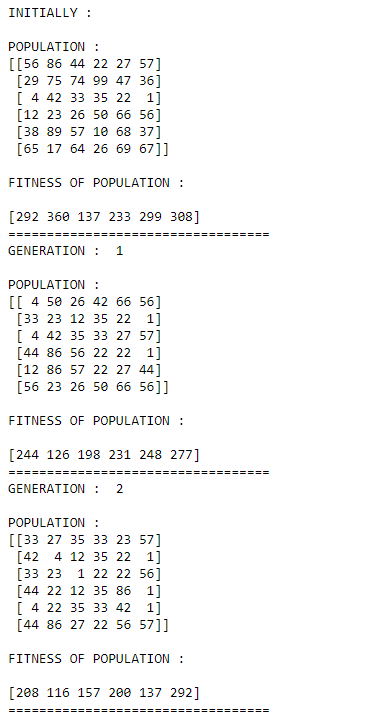


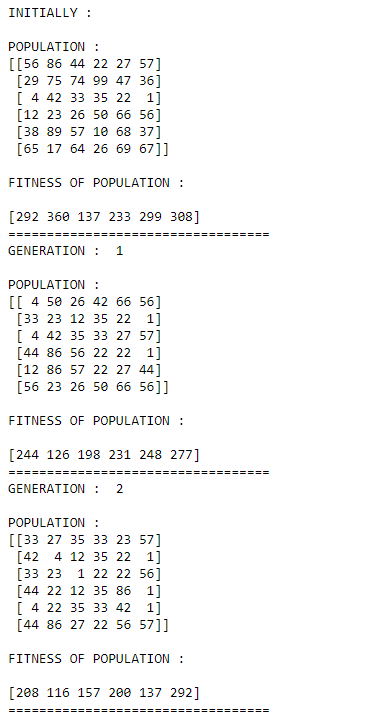


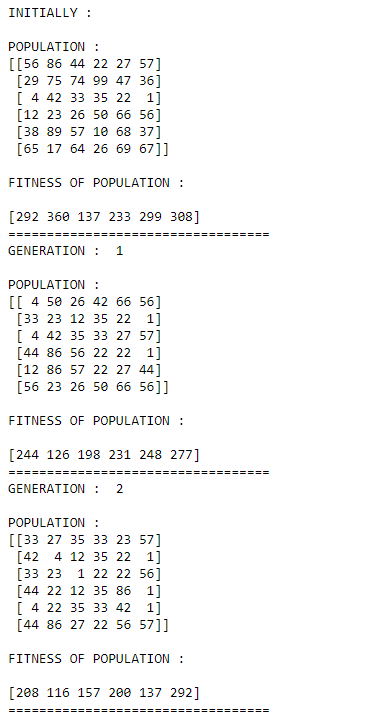
In the above snippets, we have imported the necessary libraries, made the necessary global initialization and we have created the various function definitions. Now we will look at the main function/driver code and how we have dealt with the special case that we discussed in the previous pages.

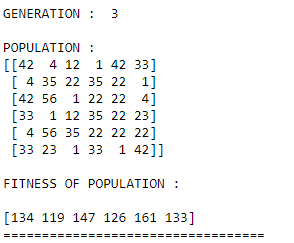


OUTPUT:









## ADVANTAGES OF TSP USING GA

1. GA works well on any problem be it continuous of discrete in nature.
2. The division of the TSP problem is made simpler in GA compared to other algorithms.
3. Multi-objective optimization is supported by GA.
4. Sometimes deserving sub-solutions or partial solutions have a high chance of being a part of the final output.

## DISADVANTAGES OF TSP USING GA

1. Computationally, this method is quite expensive.
2. The space size often sees an exponential increase due to the large number of elements that are being exposed to mutation.
3. Worst case time complexity

# TRAVELLING SALESMAN PROBLEM USING ILP IN GENOMICS

## INTEGER LINEAR PROGRAMMING

Integer Linear Programming (ILP) problems involve the maximization/minimization of a given linear function with respect to a set of linear constraints and all the variables are restricted to take on integer values. We will be dealing with the minimization of a linear cost/distance function given the set of constraints like all the cities should be visited exactly once before returning to the starting city etc. when we take into consideration our TSP using ILP problem. The concept of TSP is used in many integrated systems and applications, the ILP formulation of it reduced the complexity of the problem to a simple optimization problem with constraints.

The method of ILP we chose to implement is known as Miller-Tucker-Zemlin formulation.

The Objective Function:



Subject to the constraints:



Where:

xij - 1, if a route exists from i to j

0, otherwise

cij - Distance matrix between all i, j pairs

n - The total number of nodes

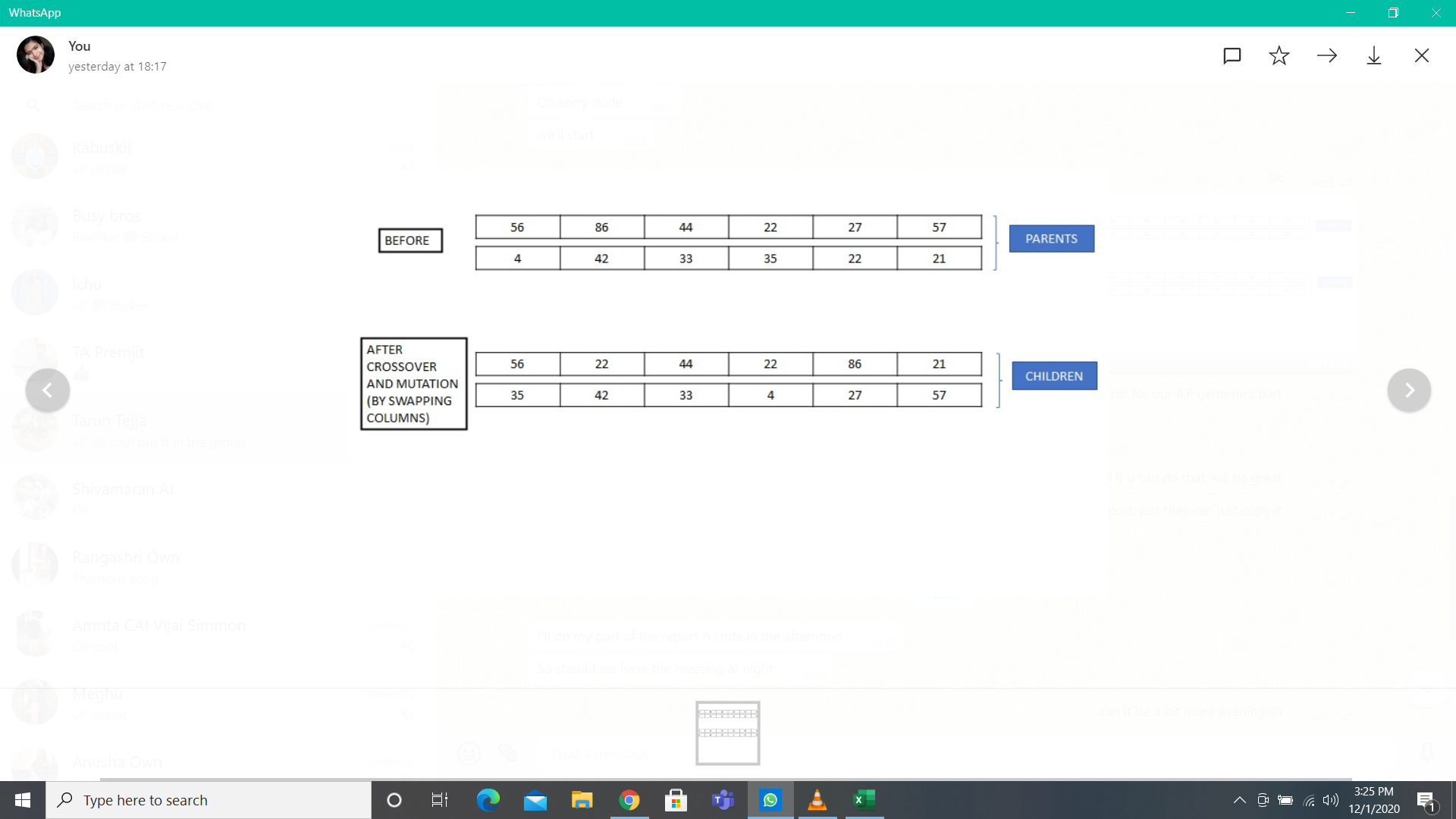
i - Each potential Origin Node

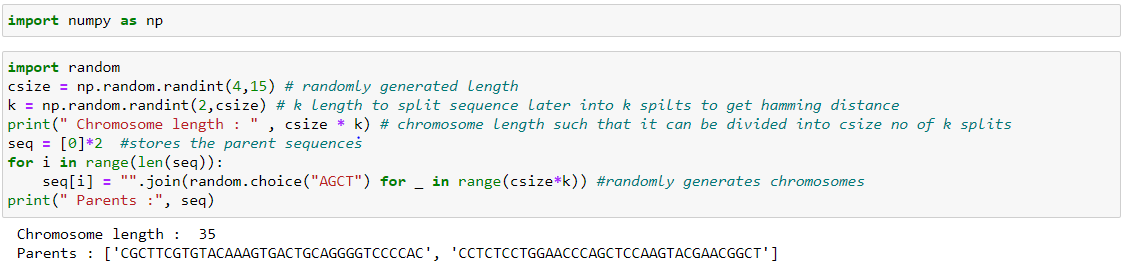
j - Each potential Destination Node

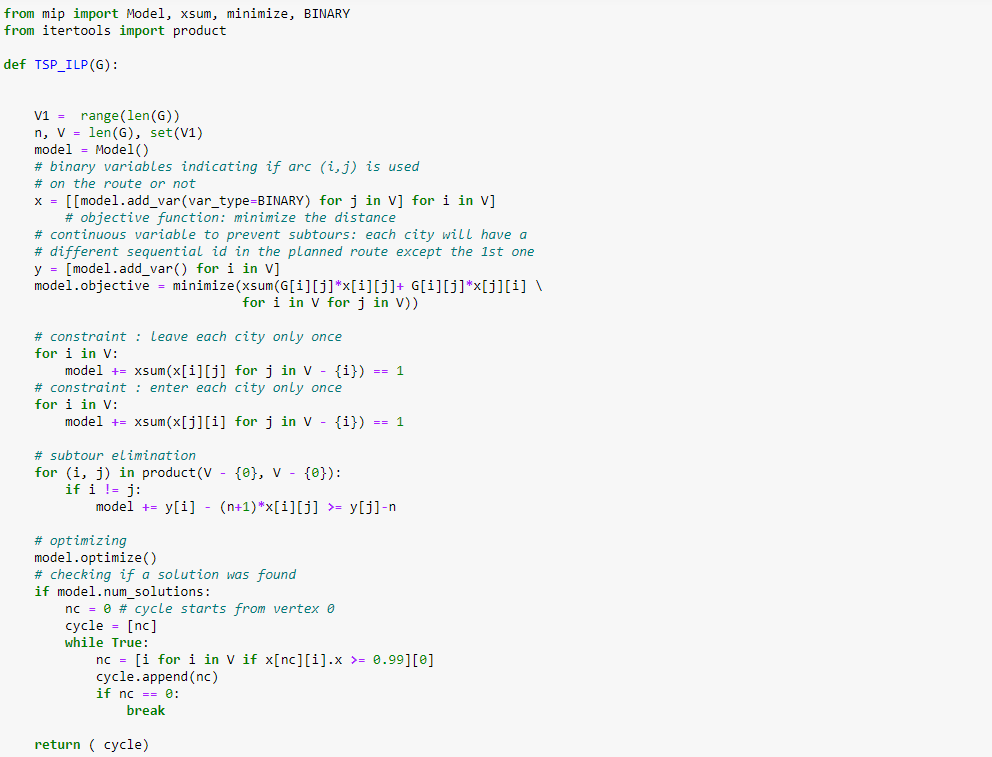
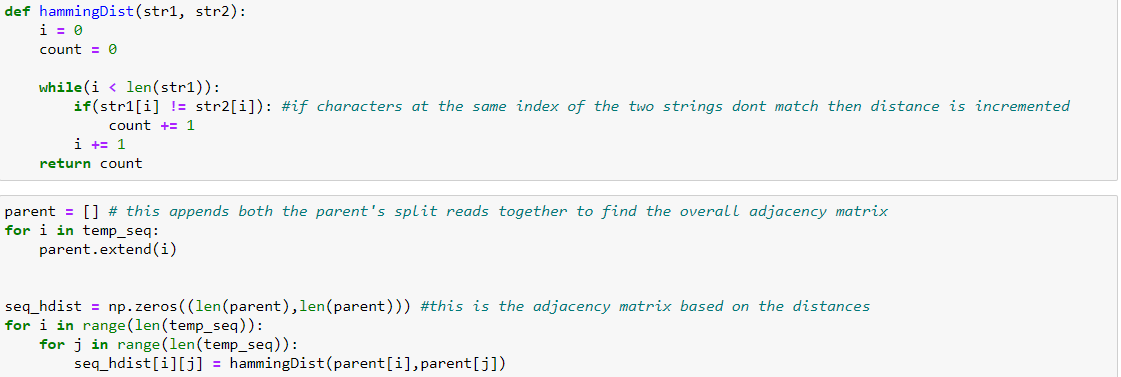
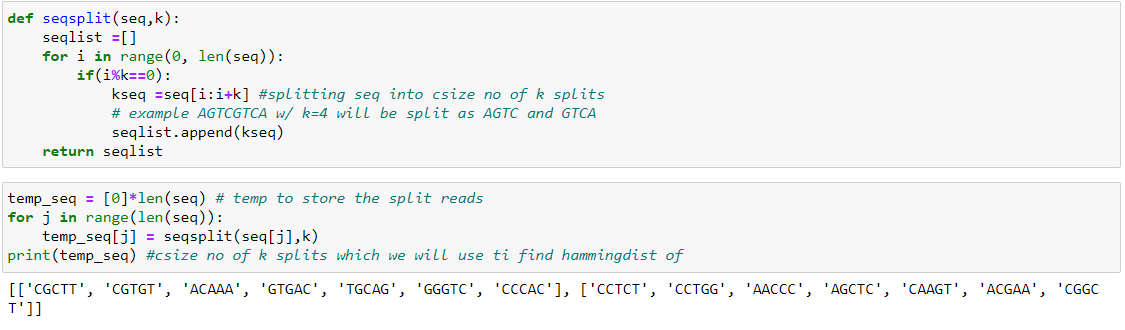
ui - Continuous, non-negative real numbers\

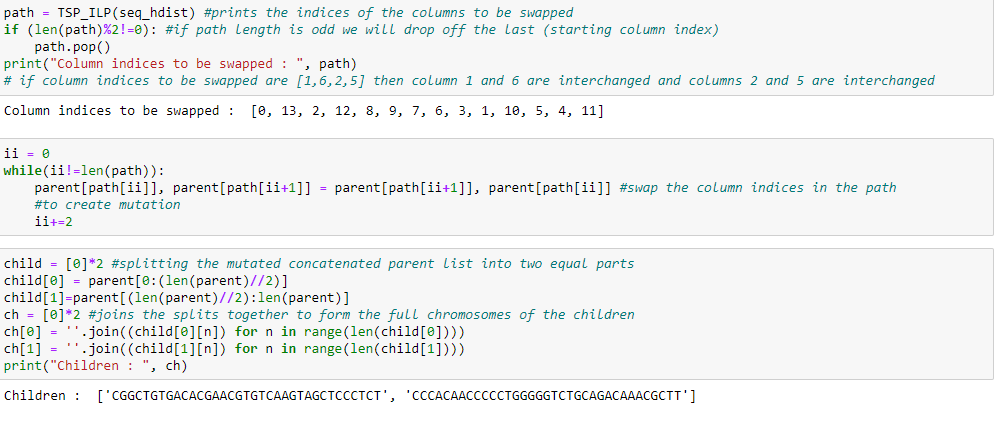
p - Number of allowed visits per node

Let us look at the implementation of TSP using ILP in genomics.

  **PYTHON COMPUTATION AND OUTPUT**







In the code we are using another concept called Hamming Distance. Hamming Distance, for our case, can be defined as the difference in the bases among two given kmers of equal k size at corresponding positions.

This implementation of TSP using ILP, is used to get the mutated off spring chromosome from a couple of parent chromosomes. We generate two random chromosomes of a random length and split them into kmer with a random k value. Then we find the hamming distances of each kmer of one parent with the kmers of the other parent. Thus found distances are put into a matrix to form the adjacency matrix for our TSP problem.

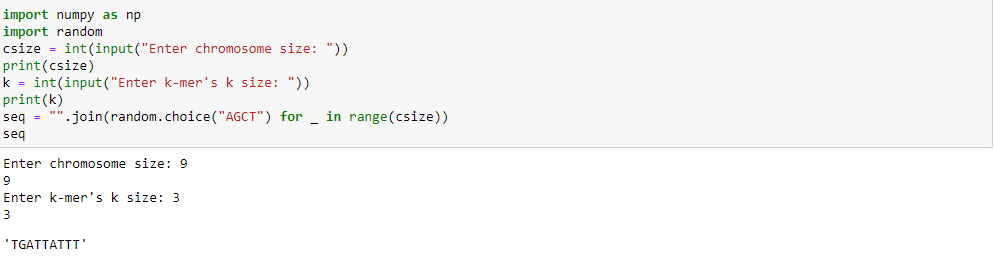
We are using the Mixed Integer Linear Programming (mip) package for the optimization functions in the TSP.

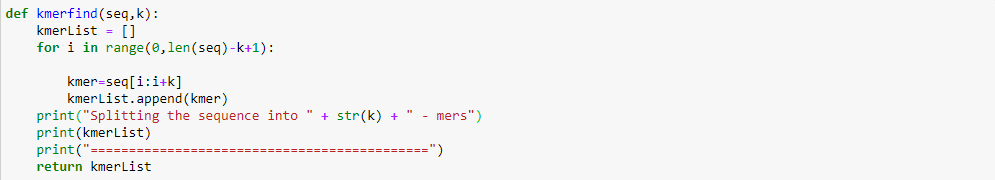
Then we pass the adjacency matrix containing Hamming Distances, into the TSP function. The minimum path, which is the output of the TSP function is used as a map for the mutation in the Genetic Algorithm.

Once the Swapping is done on the parent chromosomes, we split them and combine them in a crossed fashion. Therefore, at the end of the Algorithm we get two children chromosomes with the same size as that of the parent chromosomes and these are the mutated products of their parents.

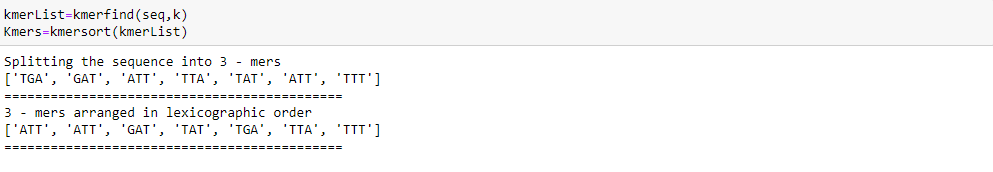
# PYTHON COMPUTATION AND OUTPUT

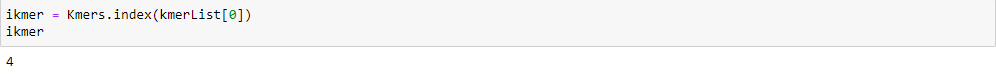
# (ILP RECONSTRUCT)

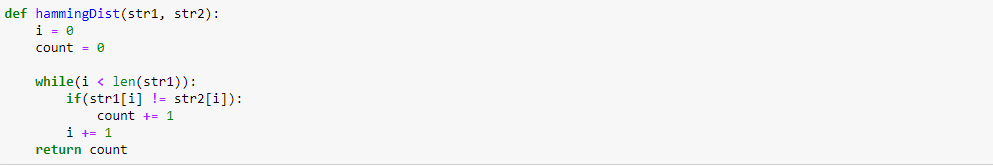


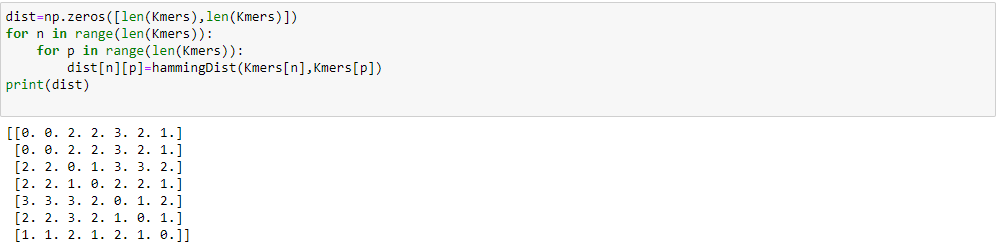


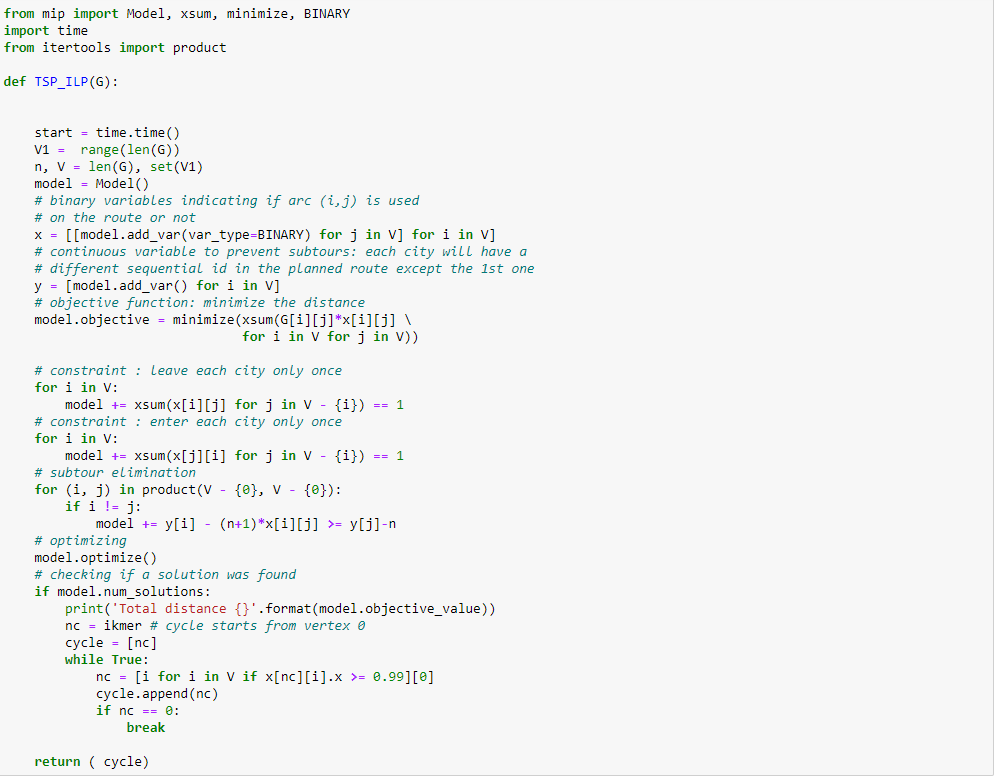


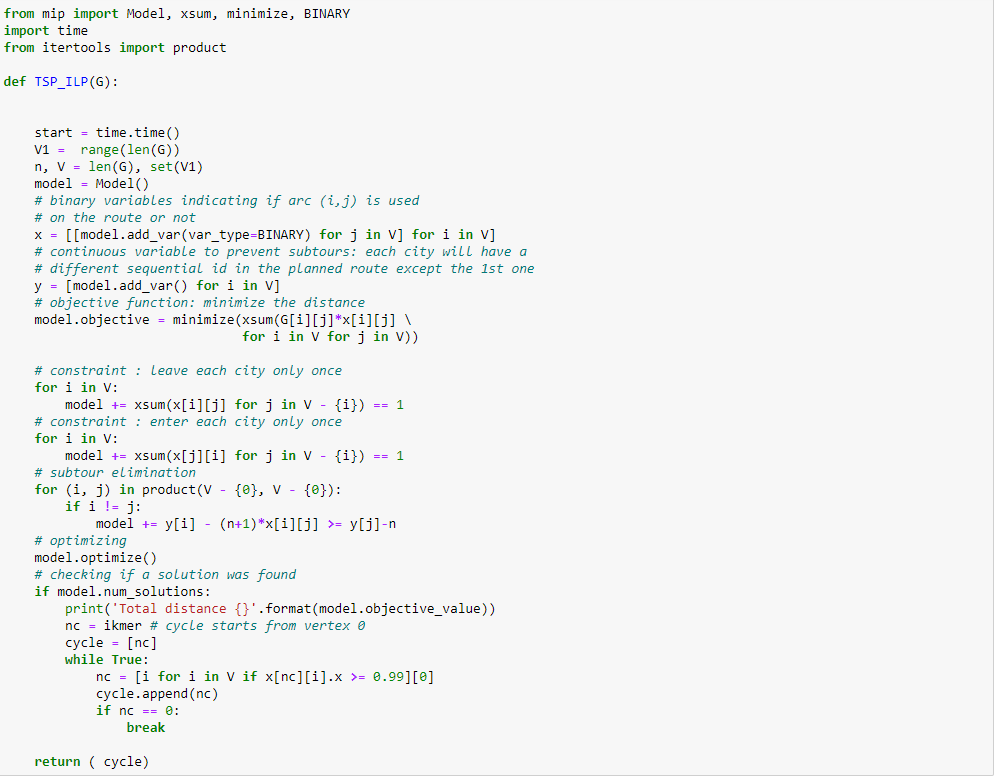


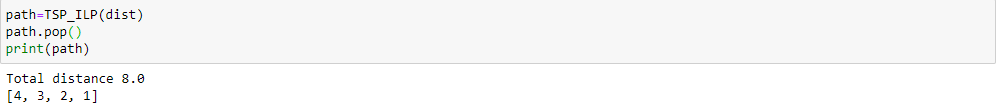












# CHALLENGES

• We can't be sure that the initial input sequence has minimum overall hamming distance, so we can't expect to get the original sequence back after reconstruction

• This logic works better only for a very large k value

• Initial kmer or it's index needs to be fed as input to the model

• Unique kmers are disregarded or placed in a wrong position

• Since this model does not take into consideration the prefix-suffix overlap, the final path cannot be reconstructed together into a perfect chromosome

• Since hamming distance adjacency matrix can sometimes be way too sparse, the size of the final chromosome

tends to be smaller than the initial chromosome as some kmers' distances are dropped during the ILP formulation

• There is also the challenge of K-mer duplicates in the given sequence

# INDIVIDUAL CONTRIBUTIONS:

The documentation and compilation of work was done by everyone.

TSP using Backtracking - Hanuma

TSP using Naive method of approach - Hanuma and Vijai

TSP using GA - Deepthi

TSP using ILP in genomics - Tarun and Deepthi

TSP using ILP for string reconstruction - Vijai and Tarun

# RESOURCES

1. Integer Linear Programming in Computational and Systems Biology - Dan Gusfield
2. https://www.youtube.com/watch?v=XaXsJJh-Q5Y
3. https://www.youtube.com/watch?v=nRJSFtscnbA
4. https://www.hindawi.com/journals/cin/2017/7430125/
5. https://www.hindawi.com/journals/tswj/2014/178621/
6. <https://medium.com/@becmjo/genetic-algorithms-and-the-travelling-salesman-problem-d10d1daf96a1>
7. <https://www.youtube.com/watch?v=InVJWW_NzFY>
8. https://www.geeksforgeeks.org/traveling-salesman-problem-using-genetic-algorithm/
9. https://medium.com/swlh/travelling-salesman-problem-tsp-with-python-1d7a0839bd7c