# TSP USING GENETIC ALGORITHM

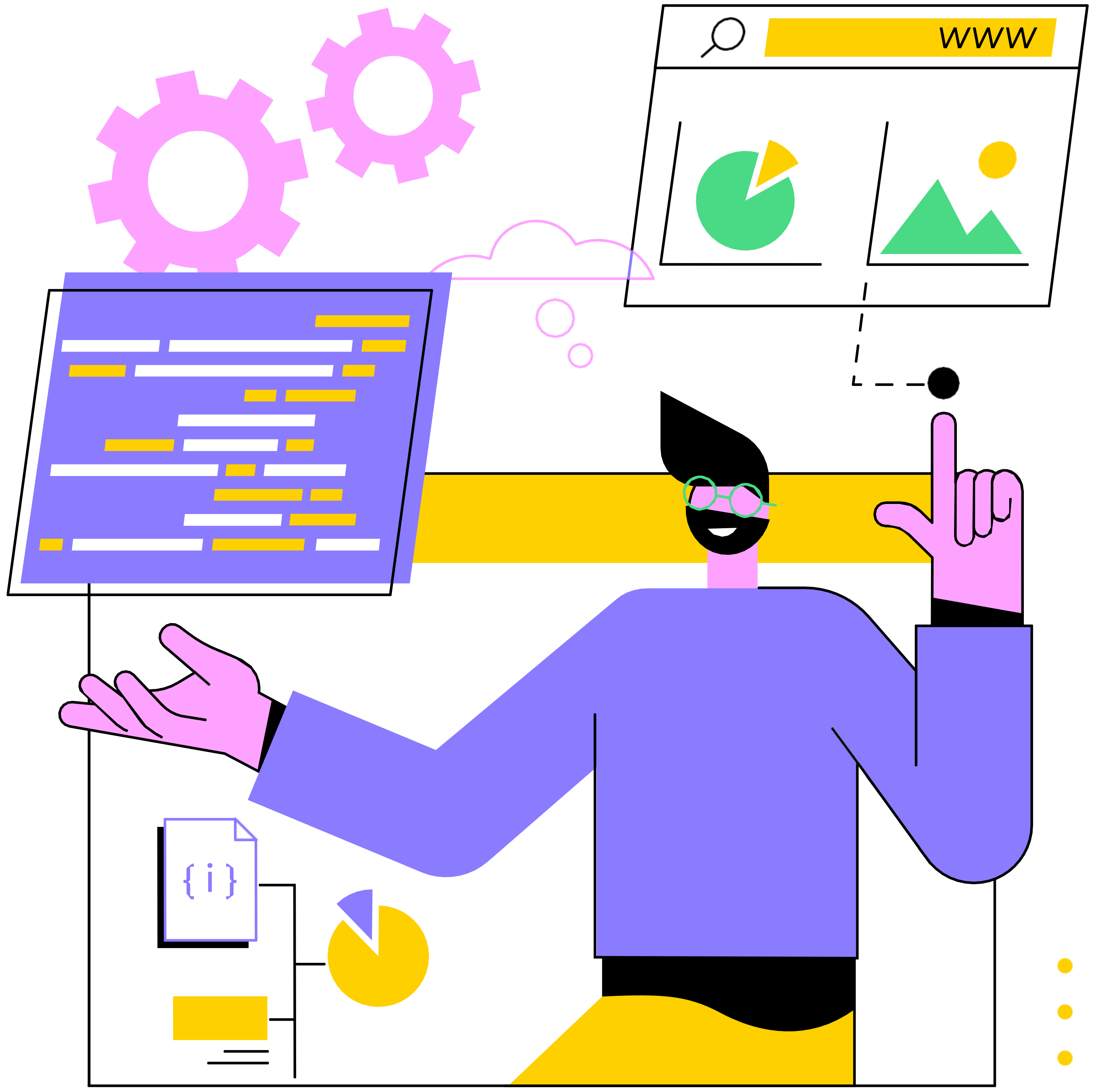
Achyut Agarwal (211IT003)

Khushi Gadling (211IT031)

Purushottam (211IT049)

Siddharth Kelkar (211IT067)

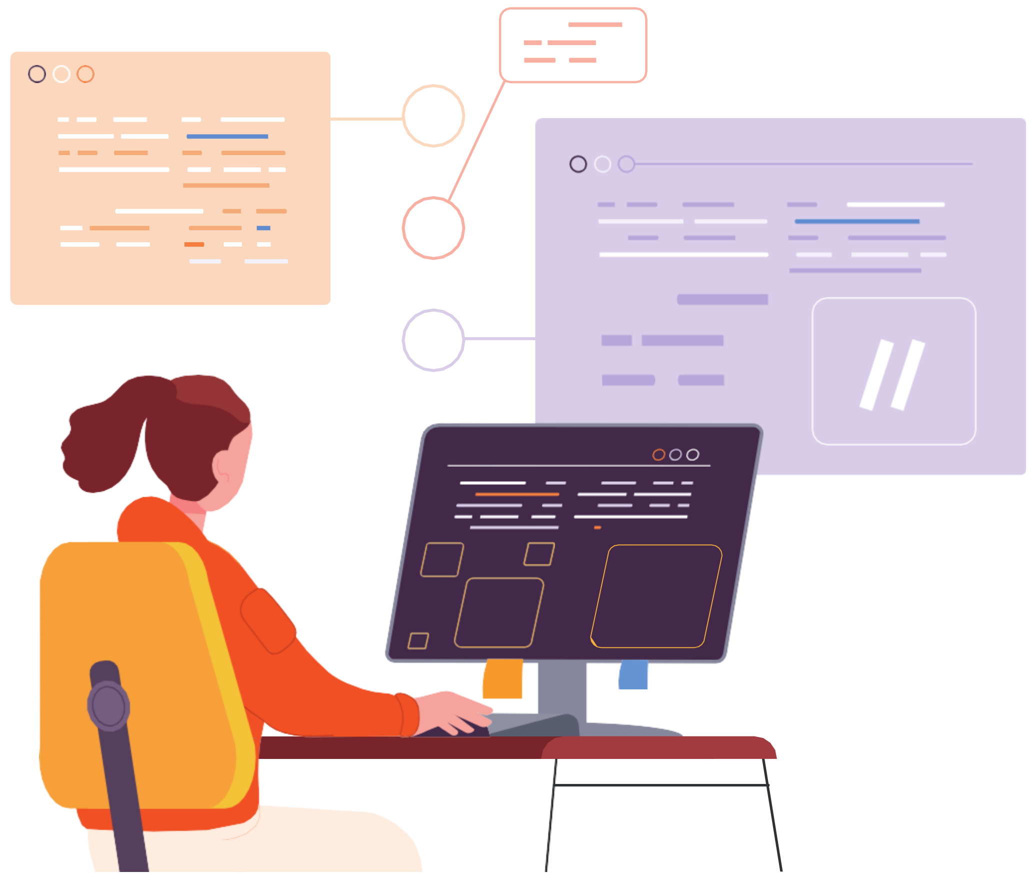
Garvit Goyal (211IT021)



## INTRO

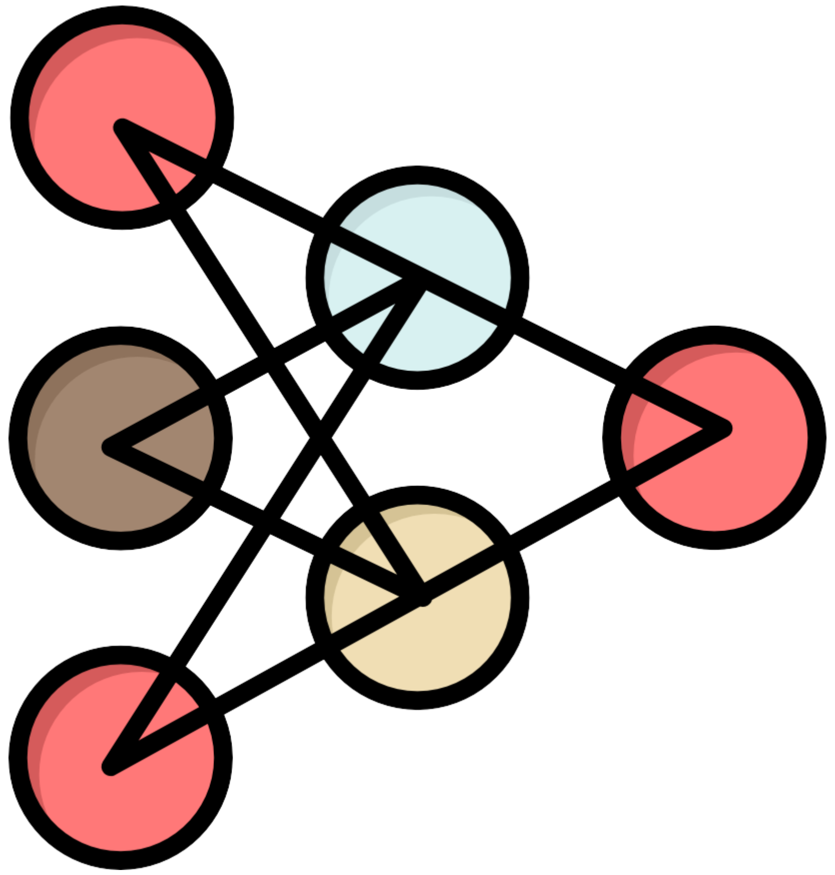
- The Traveling Salesman Problem involves finding the shortest possible route that a traveling salesman must take to visit a set of cities and return to the starting point while visiting each city exactly once.

- The TSP has been proven to be an NP-hard problem, meaning that finding an optimal solution becomes increasingly difficult as the number of cities increases and complexity grows exponentially.



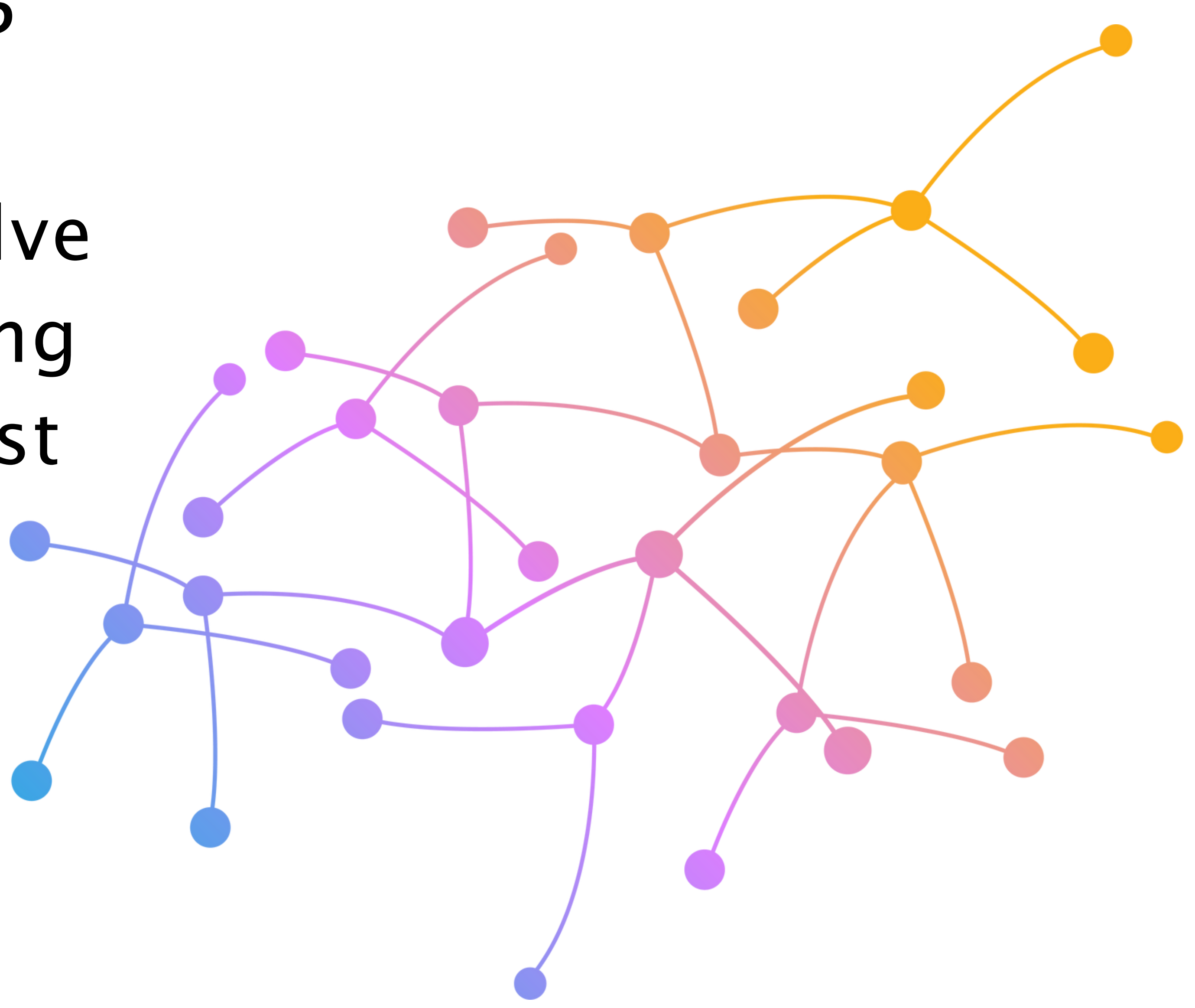
## INTRO

- Genetic Algorithms draw inspiration from natural evolution, using techniques such as selection, crossover, and mutation to evolve a population of candidate solutions over multiple generations.
- Genetic algorithms are based on the survival of the fittest and aim to optimize the solution.



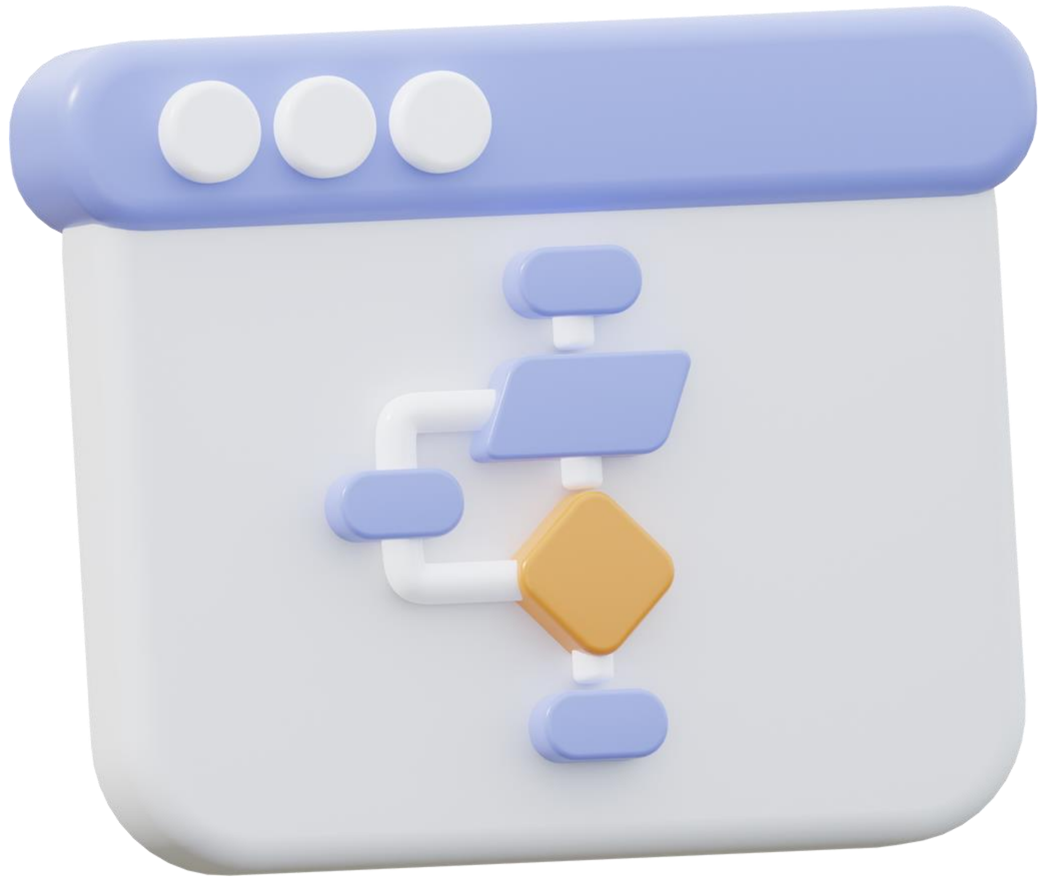
## PROBLEM STATEMENT AND AIM

- Our mini project's goal is to implement a Genetic Algorithm-based solution to the TSP and show its effectiveness in finding near-optimal solutions. Our algorithm aims to evolve a population of potential routes by leveraging evolutionary principles and selecting the best individuals to generate improved solutions iteratively.



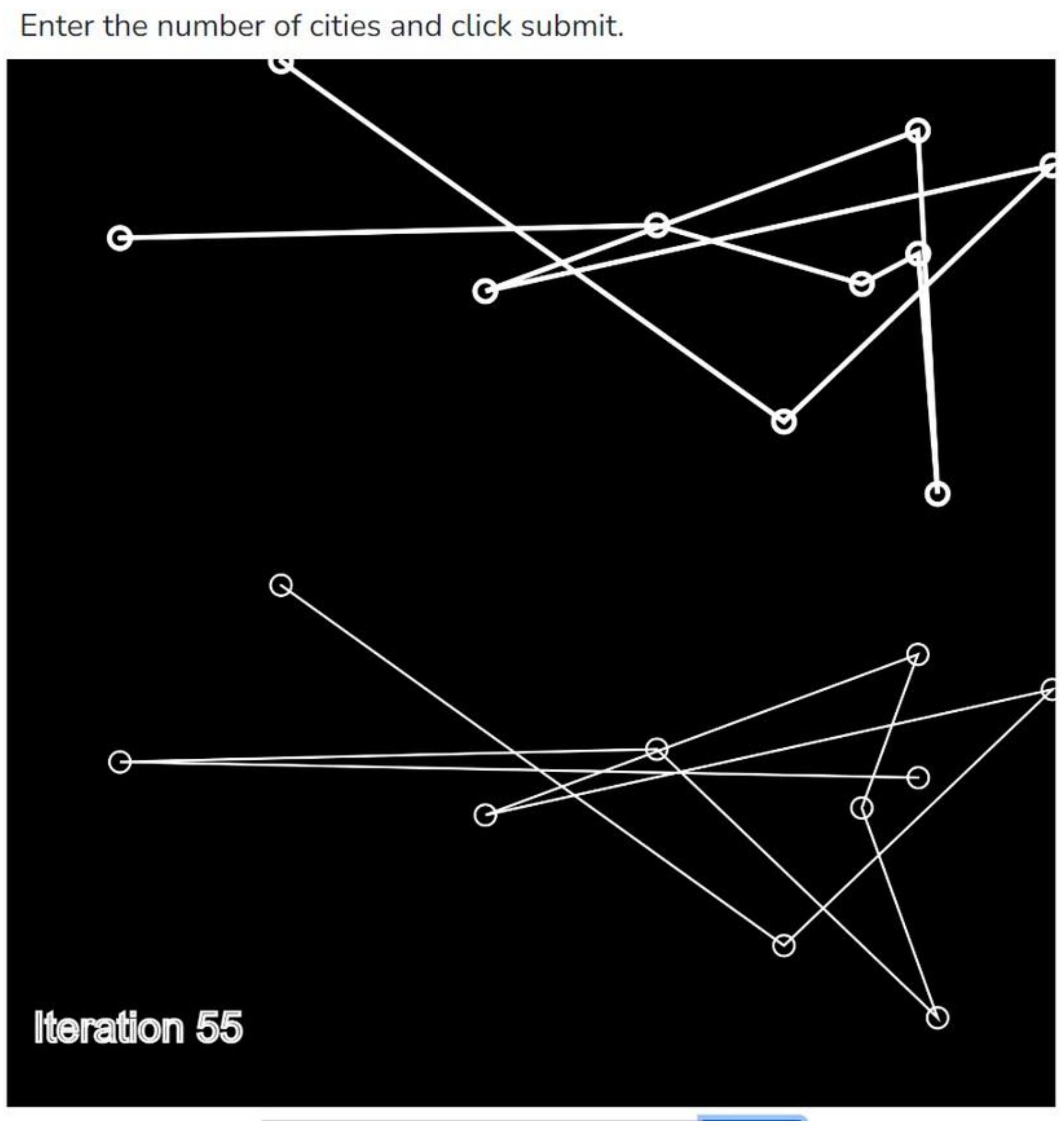
## IMPLEMENTATION

- We have written our working code in JS and used HTML CSS for displaying it.
- We implemented a web-based front to demonstrate the comparison between Brute force and Genetic algorithm-based solution.



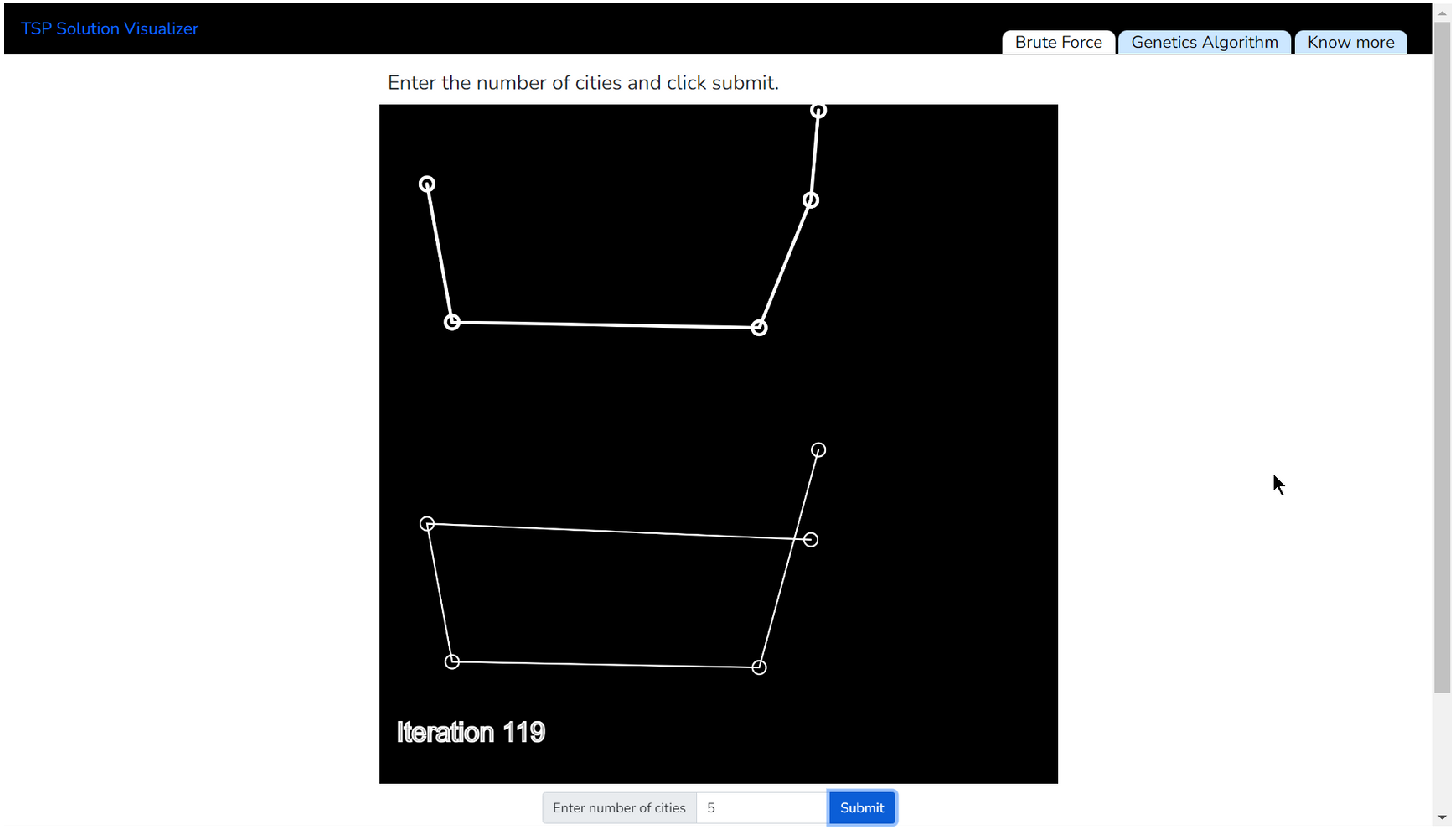
## BRUTE FORCE METHOD

- Check all available paths and iterate until done then select the path with the least total distance.
- We have used canvas and draw functions to randomly generate nodes and create vectors connecting them.


Enter the number of cities and click submit.

DSA PROJECT

Intro Problem Statement Implementation Result Conclusion



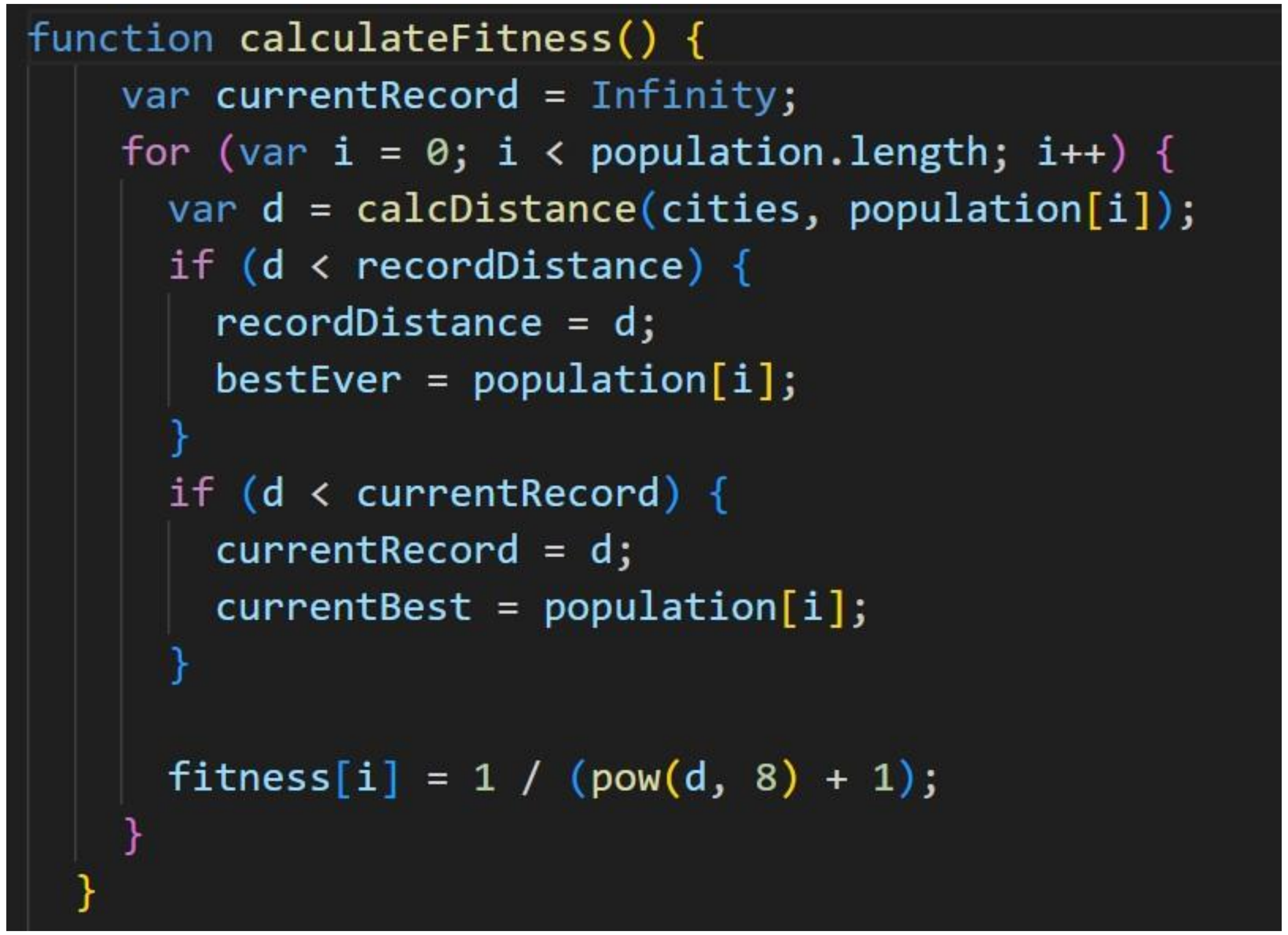
Enter the number of cities and click submit.

Brute force solution for 5 nodes

## SOME IMPORTANT STEPS

**Fitness Calculation-**

We calculate the fitness of a path by tracking the distance traveled and comparing it with the best route found so far.



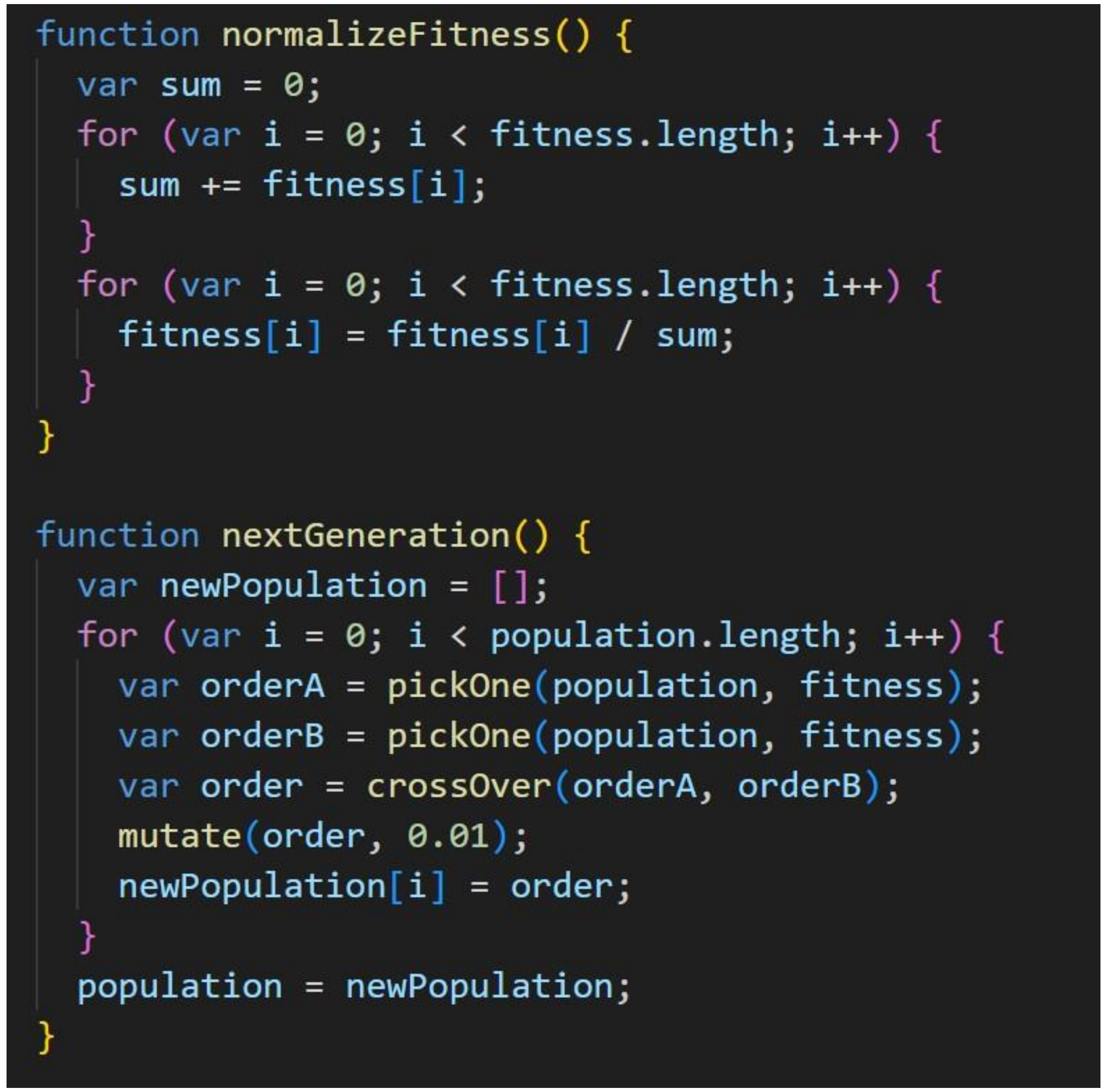
```
function calculateFitness() {
    var currentRecord = Infinity;
    for (var i = 0; i < population.length; i++) {
      var d = calcDistance(cities, population[i]);
      if (d < recordDistance) {
        recordDistance = d;
        bestEver = population[i];
      }
      if (d < currentRecord) {
        currentRecord = d;
        currentBest = population[i];
      }

      fitness[i] = 1 / (pow(d, 8) + 1);
    }
  }
```

## SOME IMPORTANT STEPS

**Fitness Normalization-**
We then normalize the fitness so they add to one.

**Find Next Generation-**
Create a new population by selecting parent routes based on their fitness, performing crossover



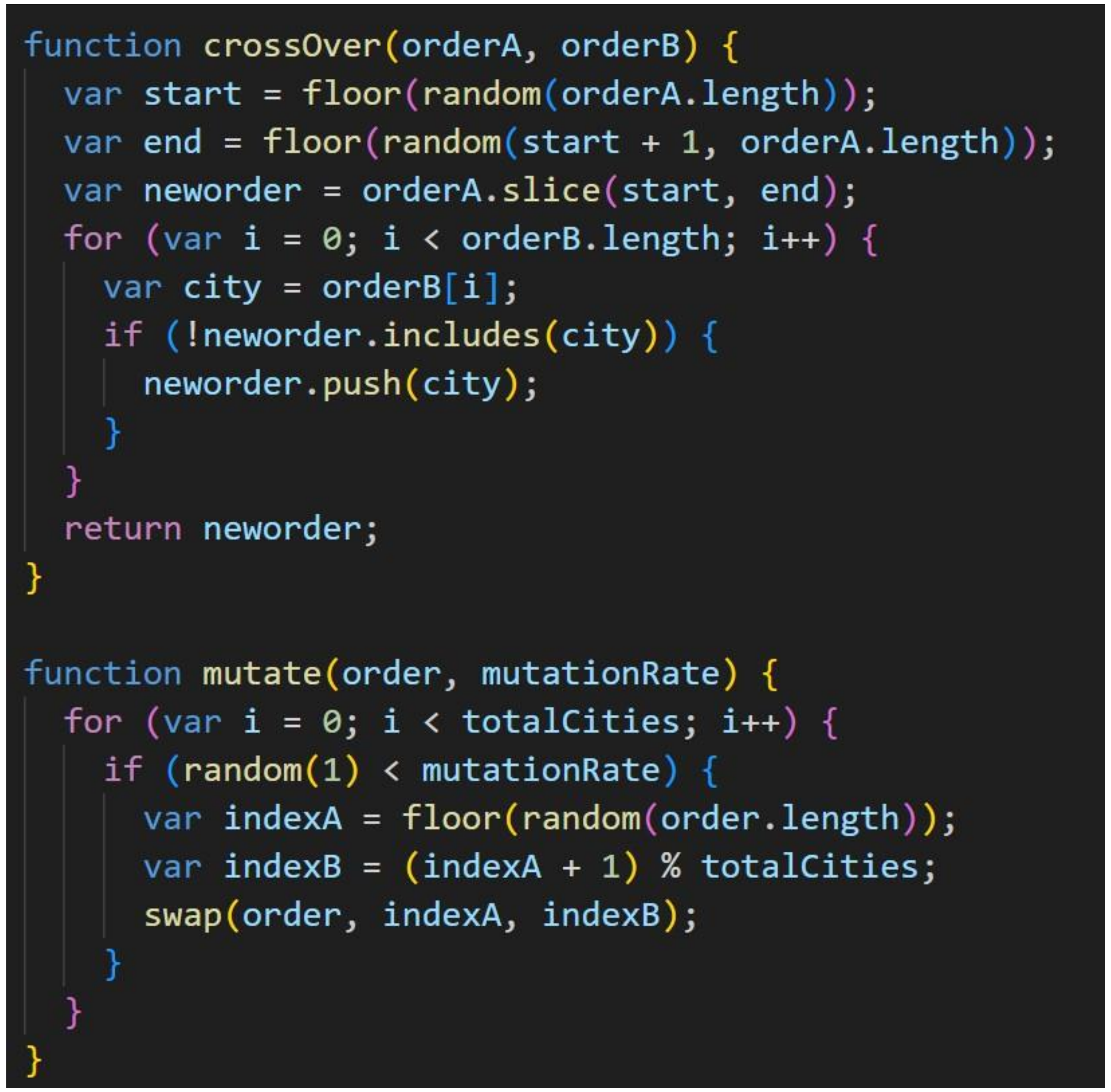
```
function normalizeFitness() {
  var sum = 0;
  for (var i = 0; i < fitness.length; i++) {
    sum += fitness[i];
  }
  for (var i = 0; i < fitness.length; i++) {
    fitness[i] = fitness[i] / sum;
  }
}

function nextGeneration() {
  var newPopulation = [];
  for (var i = 0; i < population.length; i++) {
    var orderA = pickOne(population, fitness);
    var orderB = pickOne(population, fitness);
    var order = crossOver(orderA, orderB);
    mutate(order, 0.01);
    newPopulation[i] = order;
  }
  population = newPopulation;
}
```

## SOME IMPORTANT STEPS

**Mutation-**
It introduces random changes (mutations) to a route

**Cross Over-**
**C**ombine two parent routes (orderA and orderB) to create a new offspring route. Picked randomly.

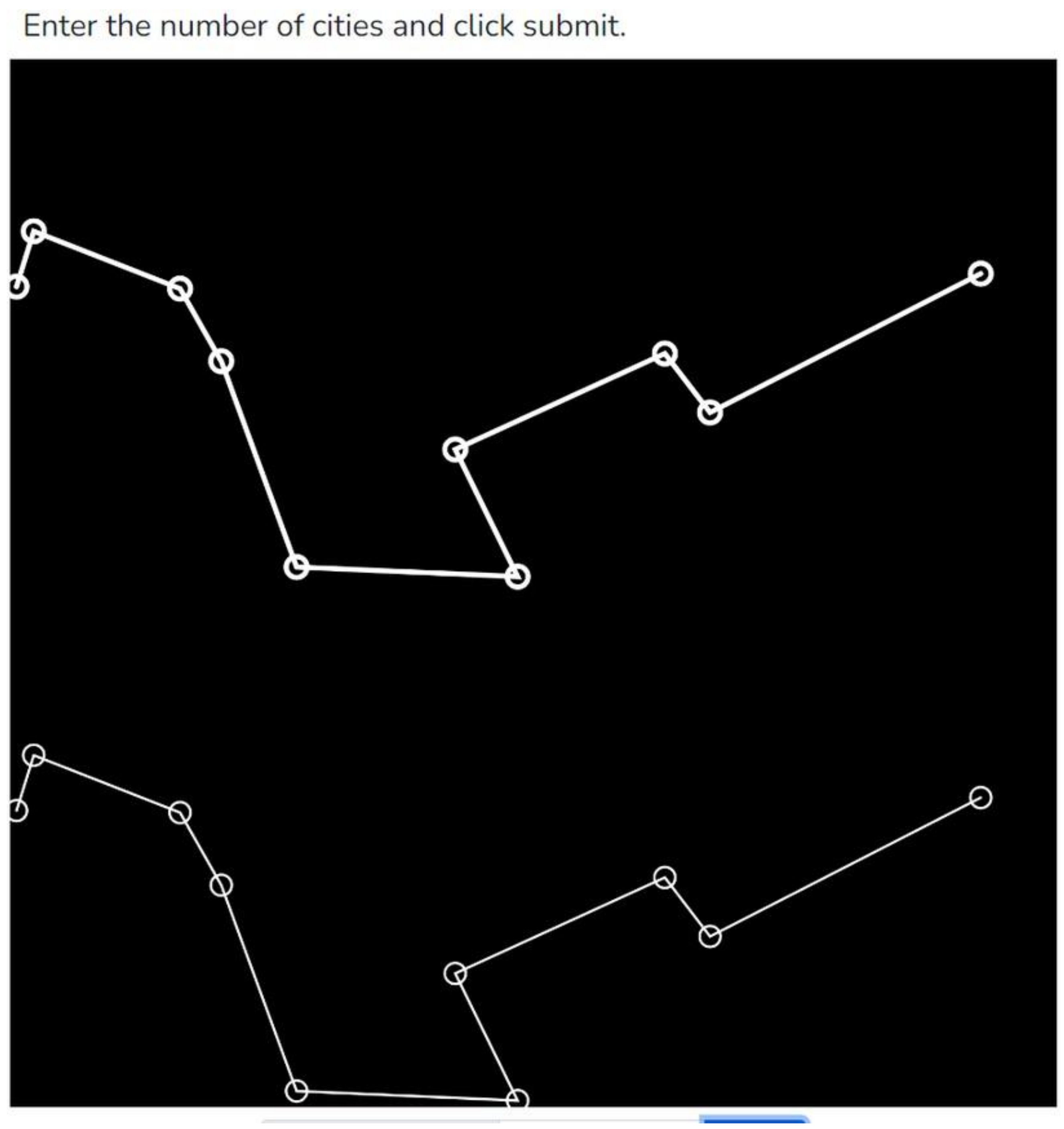


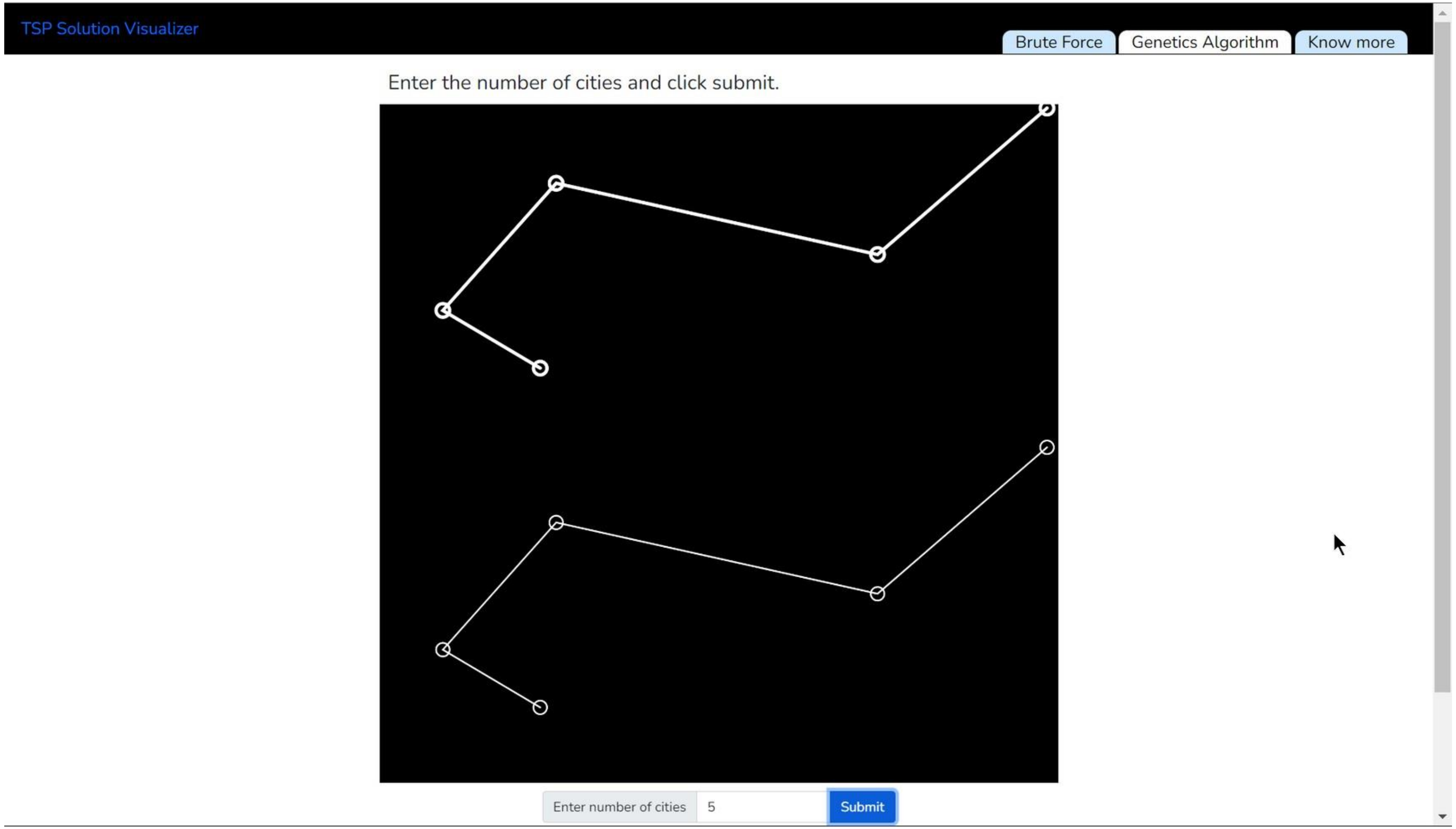
```
function crossOver(orderA, orderB) {
  var start = floor(random(orderA.length));
  var end = floor(random(start + 1, orderA.length));
  var neworder = orderA.slice(start, end);
  for (var i = 0; i < orderB.length; i++) {
    var city = orderB[i];
    if (!neworder.includes(city)) {
      neworder.push(city);
    }
  }
  return neworder;
}

function mutate(order, mutationRate) {
  for (var i = 0; i < totalCities; i++) {
    if (random(1) < mutationRate) {
      var indexA = floor(random(order.length));
      var indexB = (indexA + 1) % totalCities;
      swap(order, indexA, indexB);
    }
  }
}
```

## GENETIC ALGORITHM METHOD

- Randomly initialize the population of individual strings representing potential solutions to the TSP and create a matrix to represent the cost of paths between cities.
- Assign fitness to each chromosome in the population based on the total cost of the solution string using the fitness criteria $F(x) = 1/x$.
- Create new offspring populations by applying crossover operators on selected parent chromosomes.
- Optionally, introduce mutations to the offspring population to further diversify and explore the solution space.
- Repeat the crossover and mutation steps until an optimal solution is achieved for the TSP.


Enter the number of cities and click submit.

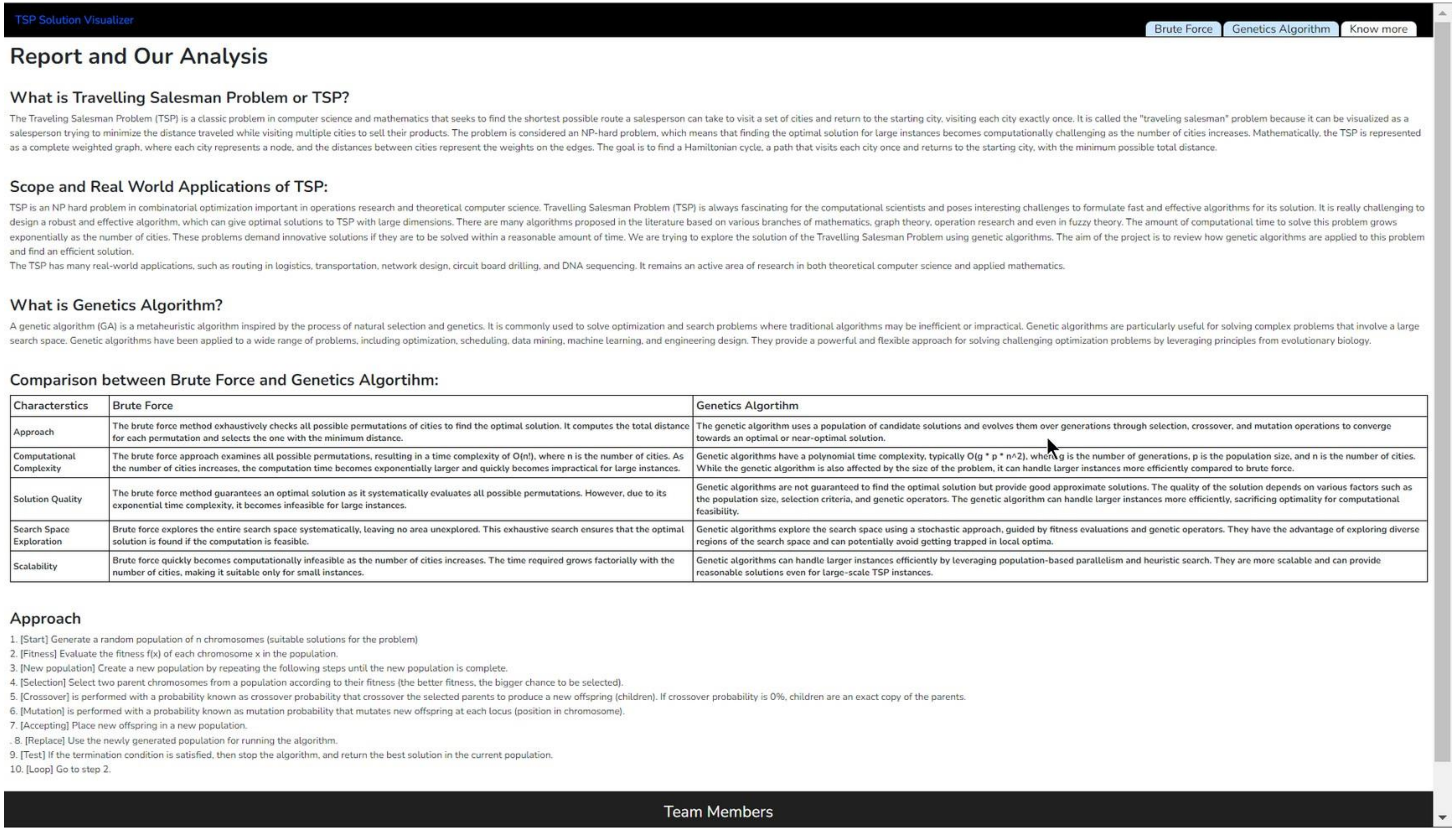


Enter the number of cities and click submit.

Genetic Algorithm Solution for 5 nodes

DSA PROJECT

Intro Problem Statement Implementation Result Conclusion



## Report and Our Analysis

### What is Travelling Salesman Problem or TSP?

The Traveling Salesman Problem (TSP) is a classic problem in computer science and mathematics that seeks to find the shortest possible route a salesperson can take to visit a set of cities and return to the starting city, visiting each city exactly once. It is called the "traveling salesman" problem because it can be visualized as a salesperson trying to minimize the distance traveled while visiting multiple cities to sell their products. The problem is considered an NP-hard problem, which means that finding the optimal solution for large instances becomes computationally challenging as the number of cities increases. Mathematically, the TSP is represented as a complete weighted graph, where each city represents a node, and the distances between cities represent the weights on the edges. The goal is to find a Hamiltonian cycle, a path that visits each city once and returns to the starting city, with the minimum possible total distance.

### Scope and Real World Applications of TSP:

TSP is an NP hard problem in combinatorial optimization important in operations research and theoretical computer science. Travelling Salesman Problem (TSP) is always fascinating for the computational scientists and poses interesting challenges to formulate fast and effective algorithms for its solution. It is really challenging to design a robust and effective algorithm, which can give optimal solutions to TSP with large dimensions. There are many algorithms proposed in the literature based on various branches of mathematics, graph theory, operation research and even in fuzzy theory. The amount of computational time to solve this problem grows exponentially as the number of cities. These problems demand innovative solutions if they are to be solved within a reasonable amount of time. We are trying to explore the solution of the Travelling Salesman Problem using genetic algorithms. The aim of the project is to review how genetic algorithms are applied to this problem and find an efficient solution.
The TSP has many real-world applications, such as routing in logistics, transportation, network design, circuit board drilling, and DNA sequencing. It remains an active area of research in both theoretical computer science and applied mathematics.

### What is Genetics Algorithm?

A genetic algorithm (GA) is a metaheuristic algorithm inspired by the process of natural selection and genetics. It is commonly used to solve optimization and search problems where traditional algorithms may be inefficient or impractical. Genetic algorithms are particularly useful for solving complex problems that involve a large search space. Genetic algorithms have been applied to a wide range of problems, including optimization, scheduling, data mining, machine learning, and engineering design. They provide a powerful and flexible approach for solving challenging optimization problems by leveraging principles from evolutionary biology.

### Comparison between Brute Force and Genetics Algortihm:

| Characterstics | Brute Force | Genetics Algortihm |
|---|---|---|
| Approach | The brute force method exhaustively checks all possible permutations of cities to find the optimal solution. It computes the total distance for each permutation and selects the one with the minimum distance. | The genetic algorithm uses a population of candidate solutions and evolves them over generations through selection, crossover, and mutation operations to converge towards an optimal or near-optimal solution. |
| Computational Complexity | The brute force approach examines all possible permutations, resulting in a time complexity of O(n!), where n is the number of cities. As the number of cities increases, the computation time becomes exponentially larger and quickly becomes impractical for large instances. | Genetic algorithms have a polynomial time complexity, typically O(g * p * n^2), where g is the number of generations, p is the population size, and n is the number of cities. While the genetic algorithm is also affected by the size of the problem, it can handle larger instances more efficiently compared to brute force. |
| Solution Quality | The brute force method guarantees an optimal solution as it systematically evaluates all possible permutations. However, due to its exponential time complexity, it becomes infeasible for large instances. | Genetic algorithms are not guaranteed to find the optimal solution but provide good approximate solutions. The quality of the solution depends on various factors such as the population size, selection criteria, and genetic operators. The genetic algorithm can handle larger instances more efficiently, sacrificing optimality for computational feasibility. |
| Search Space Exploration | Brute force explores the entire search space systematically, leaving no area unexplored. This exhaustive search ensures that the optimal solution is found if the computation is feasible. | Genetic algorithms explore the search space using a stochastic approach, guided by fitness evaluations and genetic operators. They have the advantage of exploring diverse regions of the search space and can potentially avoid getting trapped in local optima. |
| Scalability | Brute force quickly becomes computationally infeasible as the number of cities increases. The time required grows factorially with the number of cities, making it suitable only for small instances. | Genetic algorithms can handle larger instances efficiently by leveraging population-based parallelism and heuristic search. They are more scalable and can provide reasonable solutions even for large-scale TSP instances. |

### Approach

1. [Start] Generate a 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 the following steps until the new population is complete.
4. [Selection] Select two parent chromosomes from a population according to their fitness (the better fitness, the bigger chance to be selected).
5. [Crossover] is performed with a probability known as crossover probability that crossover the selected parents to produce a new offspring (children). If crossover probability is 0%, children are an exact copy of the parents.
6. [Mutation] is performed with a probability known as mutation probability that mutates new offspring at each locus (position in chromosome).
7. [Accepting] Place new offspring in a new population.
. 8. [Replace] Use the newly generated population for running the algorithm.
9. [Test] If the termination condition is satisfied, then stop the algorithm, and return the best solution in the current population.
10. [Loop] Go to step 2.

Analysis and comparison

## RESULT

- We have found out that Genetic Algorithms provide better results and are more efficient and fast as compared to the brute force approach

## CONCLUSION

- In summary, our study on the traveling salesman problem using a genetic algorithm has yielded **efficient and effective results** compared to the brute force approach.
- By carefully implementing crossover and mutation functions, we achieved optimal solutions in less time.
- Genetic algorithms have historically provided some of the best solutions to the TSP, and our future work will focus on improving the survivor selector to further enhance our algorithm's performance. Our findings underscore the potential of genetic algorithms for solving the TSP and highlight avenues for future research in this area.

# THANKYOU