AI Mini Project Report-2

Name: S.Vishnu Teja **Roll No**: CS21B2037

Name: P.Veera Abhiram **Roll No**: CS21B2026

Travelling Salesman Problem using Genetic Algorithm

Solving Tsp using Genetic Algorithm(GA). Genetic Algorithms are adaptive heuristic search algorithms that belong to the larger part of evolutionary algorithms. They are based on the ideas of natural selection and genetics. They are commonly used to generate high-quality solutions for optimization problems and search problems.

Genetic algorithms simulate the process of natural selection which means those species who can adapt to changes in their environment are able to survive and reproduce and go to next generation. Simply, the “survival of the fittest” among individual of consecutive generation for solving a problem. Each generation consist of a population of individuals and each individual represents a point in search space and possible solution.

Foundation for Genetic Algorithms:

The basic genetic algorithm (GA) typically consists of two steps. The initial step is the selection of individuals for the purpose of generating the new generation, while the second phase involves manipulating the chosen individuals to create the following generation using procedures like as crossover and mutation.

The individuals that are chosen for mating are determined by the selection mechanism. The primary aim of selection strategy is that "the better an individual, the higher the possibility that it will be a parent". Generally, crossover and mutation explore the search space, whereas selection reduces the search area within the population by discarding poor solution.

Different selection strategies used in the GA process will have a significant impact on the way the algorithm performs.

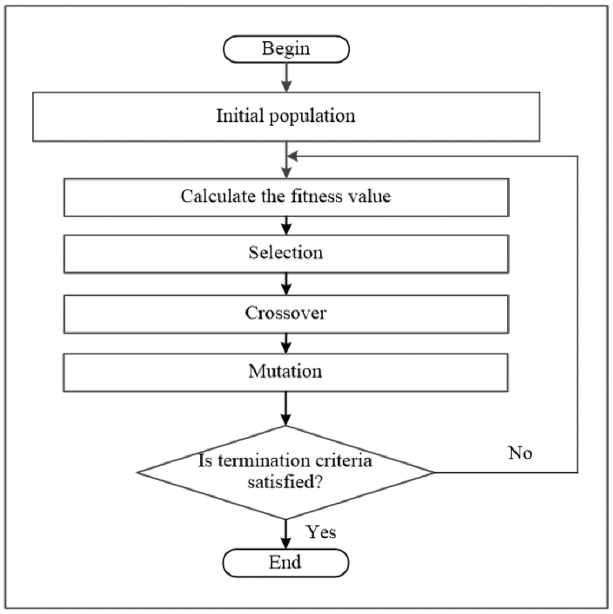
This Objective of this Report is intended to examine the performance of GA when using different selection strategy methods especially in solving the travelling salesman problem (TSP), a classic example for NP-Hard problem.

The performance of genetic algorithm is usually examined in terms of convergence rate and the number of generations to reach the optimal solution.

Like for example, Comparision between proportional roulette wheel with tournament selection, with tournament size equal to 6 at several general test functions concluded that tournament selection is more efficient in convergence than proportional roulette wheel selection.

Research concluded that tournament selection is preferred over rank-based selection because repeated tournament selection is faster than sorting according to ranks(based on fitness value).

The Basic procedure of GA for TSP is:


Begin
Initial population
Calculate the fitness value
Selection
Crossover
Mutation
Is termination criteria satisfied?
No
Yes
End

Fitness value: Fitness value/score is a measure of how good a particular solution to a problem fits the constraints of the problem. And Fitness function is used to Check the Quality of individual solution. The goal of GA is to go with individual with Better Fitness value.

Once the initial generation is produced, the algorithm evolves the generation using following operators.

The Operators Are Selection Operator, Crossover operator, Mutation Operator.

Selection Operator:

The Selection Operator selects Carefully selects better members (i.e fitness value is high) should have the high probability of mutate, but that poor individuals of the population still have a small probability of being selected, and this is important to ensure that the search process is global and does not simply converge to the nearest local optimum. Different methods have different strategies of methods of calculating selection probability, but all based on the "The Survival of the Fittest". Here We Will Discuss About two methods of selection strategies.

1.Roulette Wheel Selection:

In a proportional roulette wheel, each individual's selection corresponds to a particular region of the wheel, with a probability that is directly proportional to their fitness scores. The probability of choosing a parent can be equivalent to spinning a roulette wheel, with the size of each segment corresponding to the parent's fitness.
Those with the highest fitness are more likely to be selected, obviously.The total fitness values of the people make up the roulette wheel's circumference.

All segments have a chance, with a probability proportional to their width. By repeating this each time, the better individuals will be chosen more often than the loosy ones.Let f1, f2,…, fn be fitness values of individual 1,2,…, n.
Then, selection probability pi is

$$pi= fi/\Sigma\ (fi)$$

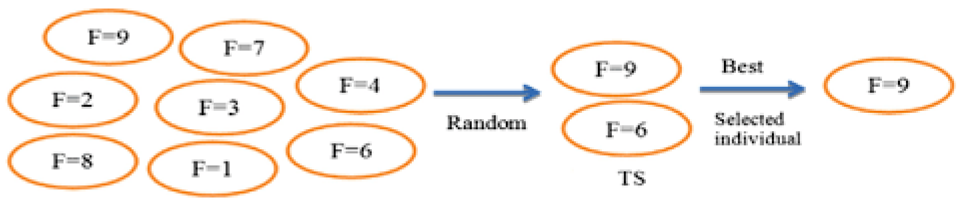
The advantage of proportional roulette wheel selection is that it gives a chance to all of them to be selected. And preserve the diversity among the population. Although individual with large fitness value gets more chance.

The Disadvantage of proportional roulette wheel is if the individual have the same fitness value then it will become very difficult for the population to select a better one. Since selection probabilities for fit and unfit individuals are very similar. The fitness function for minimization must be changed to a maximisation function, as in the case of TSP, making it challenging to apply this selection technique to minimization problems. Although this somewhat resolves the selection problem.

2.Tournament Based Selection:

Tournament selection is the most popular selection method in genetic algorithm due to its efficiency and simple implementation.
In tournament selection, n people are chosen at random from the greater population, and the individuals selected then compete with each other. The individual with highest fitness goes to the next generation population.
The idea of "tournament size" means the number of competitors in each tournament, which usually equals two individuals.

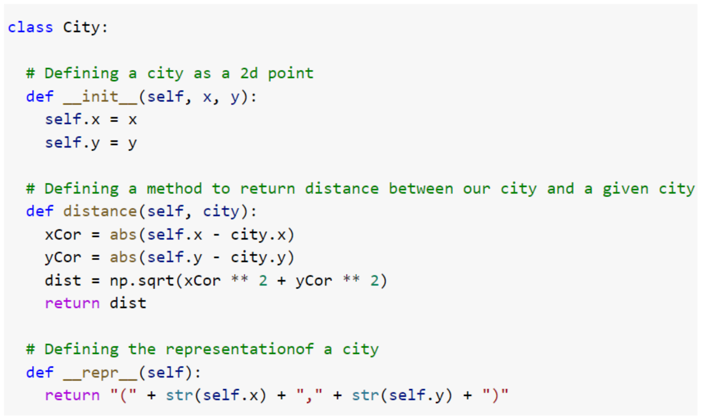
Also, the selection process used in tournaments allows all candidates to be chosen, maintaining diversity, even though it decreases convergence.



The advantage of tournament Selection is its efficient time complexity, especially if implemented in parallel, low chance domination of dominant individuals, and no requirement for fitness scaling or sorting.

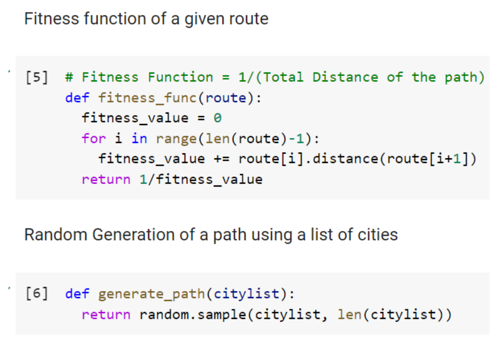
The Disadvantages of tournament Selection Weak individuals have a smaller chance to be selected if tournament size is large.

Coding Part:



```python
class City:

  # Defining a city as a 2d point
  def __init__(self, x, y):
    self.x = x
    self.y = y

  # Defining a method to return distance between our city and a given city
  def distance(self, city):
    xCor = abs(self.x - city.x)
    yCor = abs(self.y - city.y)
    dist = np.sqrt(xCor ** 2 + yCor ** 2)
    return dist

  # Defining the representationof a city
  def __repr__(self):
    return "(" + str(self.x) + "," + str(self.y) + ")"
```

In this part of code we create the necessary cities for TSP and calculated the distance between the cities.



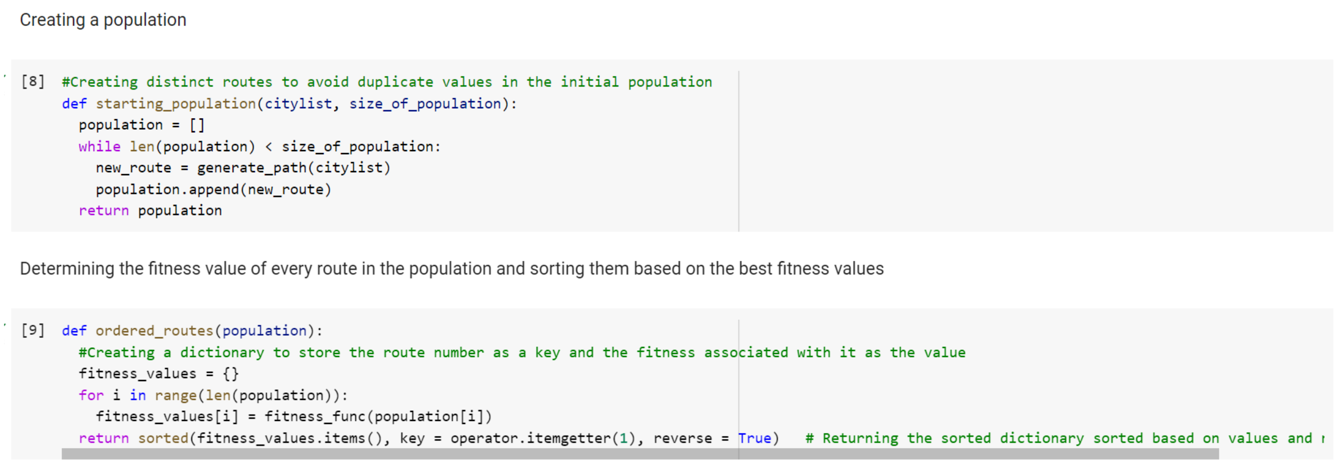
Fitness function of a given route

```python
[5]  # Fitness Function = 1/(Total Distance of the path)
     def fitness_func(route):
       fitness_value = 0
       for i in range(len(route)-1):
         fitness_value += route[i].distance(route[i+1])
       return 1/fitness_value
```

Random Generation of a path using a list of cities

```python
[6]  def generate_path(citylist):
       return random.sample(citylist, len(citylist))
```

As the main constraint is to find min distance between cities. The fitness value is defined as the distance between cities.Fitness function returns the value 1/ fitness_value. And below generatepath generate the between the list of cities.



Creating a population

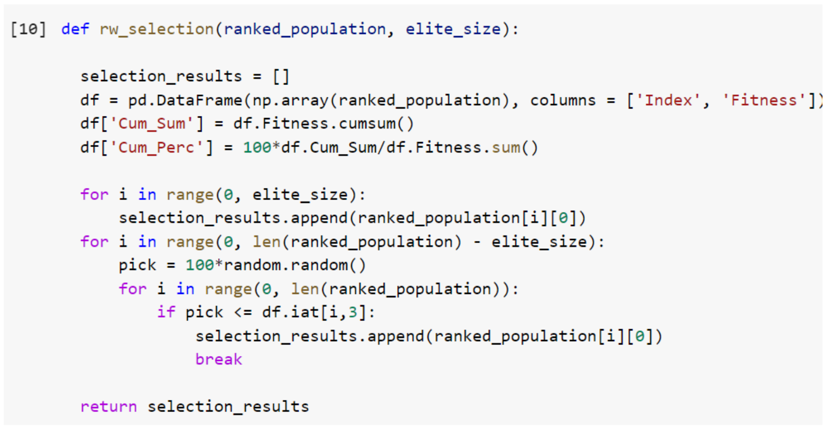
```python
[8]  #Creating distinct routes to avoid duplicate values in the initial population
     def starting_population(citylist, size_of_population):
       population = []
       while len(population) < size_of_population:
         new_route = generate_path(citylist)
         population.append(new_route)
       return population
```

Determining the fitness value of every route in the population and sorting them based on the best fitness values

```python
[9]  def ordered_routes(population):
       #Creating a dictionary to store the route number as a key and the fitness associated with it as the value
       fitness_values = {}
       for i in range(len(population)):
         fitness_values[i] = fitness_func(population[i])
       return sorted(fitness_values.items(), key = operator.itemgetter(1), reverse = True)    # Returning the sorted dictionary sorted based on values and
```

This is initialization of genetic Algorithm And above code is to generate the population necessary. And the second below function is to find fitness value for every individual in population and sort accordingly.

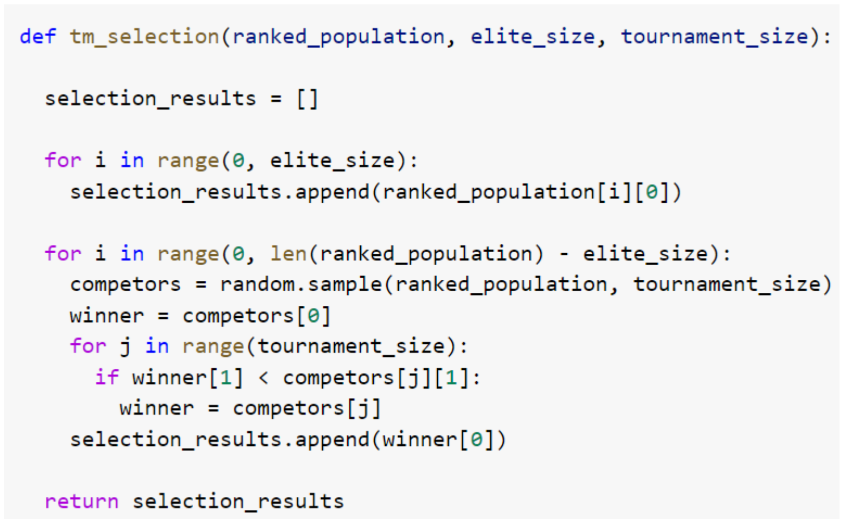
(a) Roulette Wheel Selection



```
[10] def rw_selection(ranked_population, elite_size):

       selection_results = []
       df = pd.DataFrame(np.array(ranked_population), columns = ['Index', 'Fitness']
       df['Cum_Sum'] = df.Fitness.cumsum()
       df['Cum_Perc'] = 100*df.Cum_Sum/df.Fitness.sum()

       for i in range(0, elite_size):
           selection_results.append(ranked_population[i][0])
       for i in range(0, len(ranked_population) - elite_size):
           pick = 100*random.random()
           for i in range(0, len(ranked_population)):
               if pick <= df.iat[i,3]:
                   selection_results.append(ranked_population[i][0])
                   break

       return selection_results
```

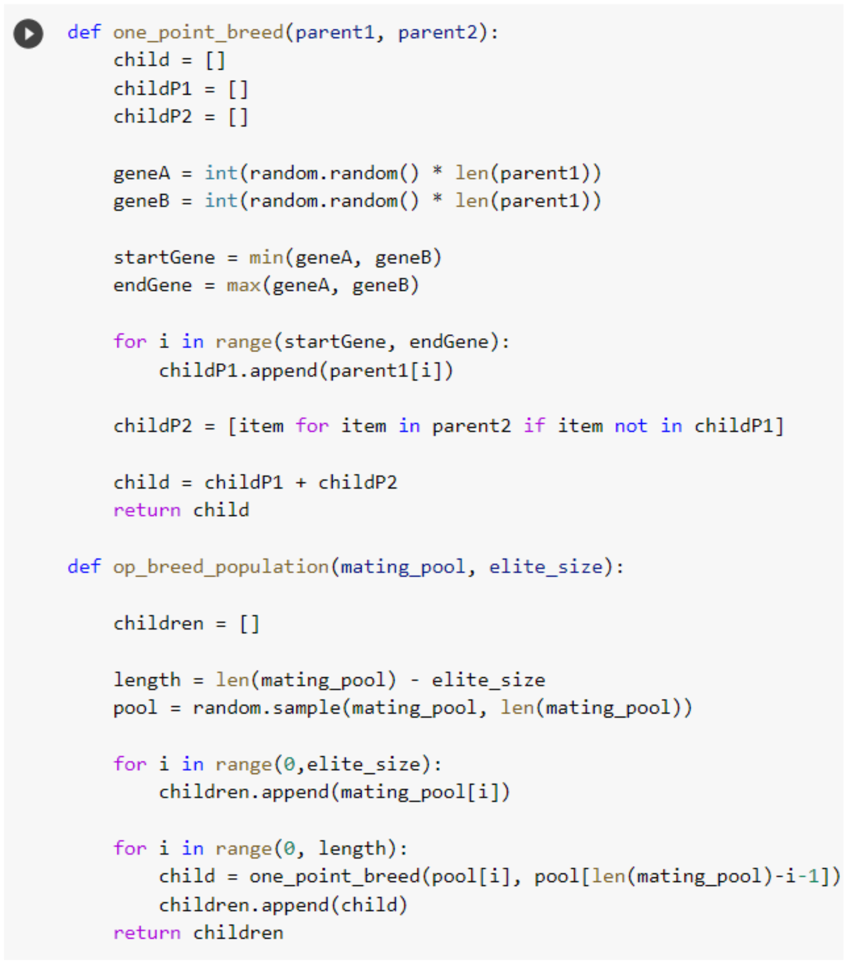
This is the part where the Roulette Wheel selection is applied and in this we created an array with selection results where the selected population that is who wins according to their fitness value is appended. And we also calculating the cumulative percentage by dividing(its fitness value)/(summation of all fitness values)*100. The more this percentage the more probability of selection. And we are selecting the required amount of individual by this and storing in the selection_results array.

Tournament based selection



```
def tm_selection(ranked_population, elite_size, tournament_size):

  selection_results = []

  for i in range(0, elite_size):
    selection_results.append(ranked_population[i][0])

  for i in range(0, len(ranked_population) - elite_size):
    competors = random.sample(ranked_population, tournament_size)
    winner = competors[0]
    for j in range(tournament_size):
      if winner[1] < competors[j][1]:
        winner = competors[j]
    selection_results.append(winner[0])

  return selection_results
```

This is the part where we will do the Tournament selection method. And this we create a tournament size of our choice. We will pick a random population of that size and there will be competition among them and the individual with highest fitness value is picked and likewise we will pick random again in the same method and winner are stored. we will do this until we get required number of individuals.

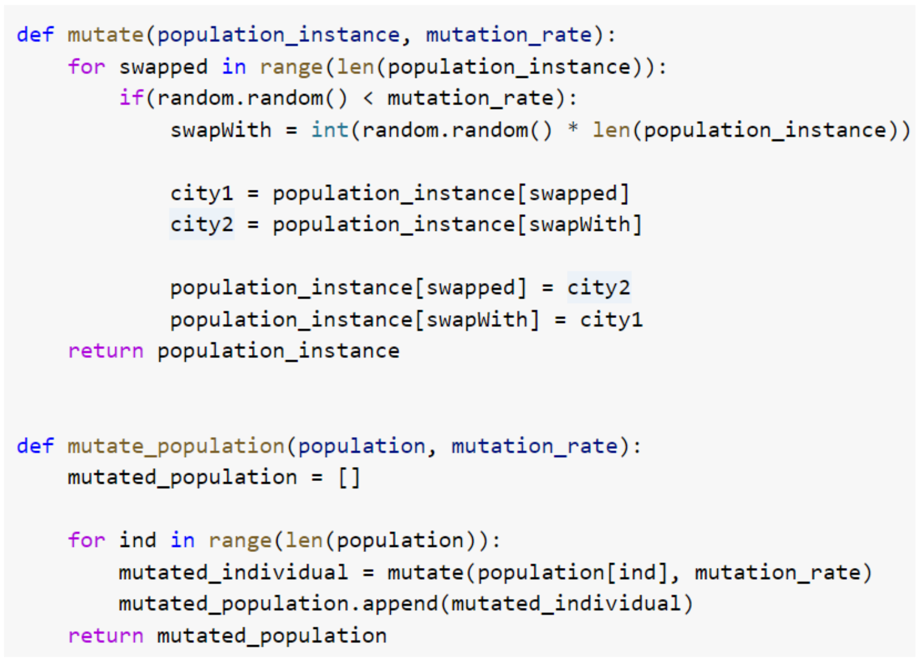


```
def one_point_breed(parent1, parent2):
    child = []
    childP1 = []
    childP2 = []

    geneA = int(random.random() * len(parent1))
    geneB = int(random.random() * len(parent1))

    startGene = min(geneA, geneB)
    endGene = max(geneA, geneB)

    for i in range(startGene, endGene):
        childP1.append(parent1[i])

    childP2 = [item for item in parent2 if item not in childP1]

    child = childP1 + childP2
    return child

def op_breed_population(mating_pool, elite_size):

    children = []

    length = len(mating_pool) - elite_size
    pool = random.sample(mating_pool, len(mating_pool))

    for i in range(0,elite_size):
        children.append(mating_pool[i])

    for i in range(0, length):
        child = one_point_breed(pool[i], pool[len(mating_pool)-i-1])
        children.append(child)
    return children
```

This is the part of Algo where crossover happens and in this case the above code is for single point crossover.

In this above part will generate the random number and at that point we will cut the parent and we will attach the 2nd part of 1st to 1st part of 2nd . And we will join them

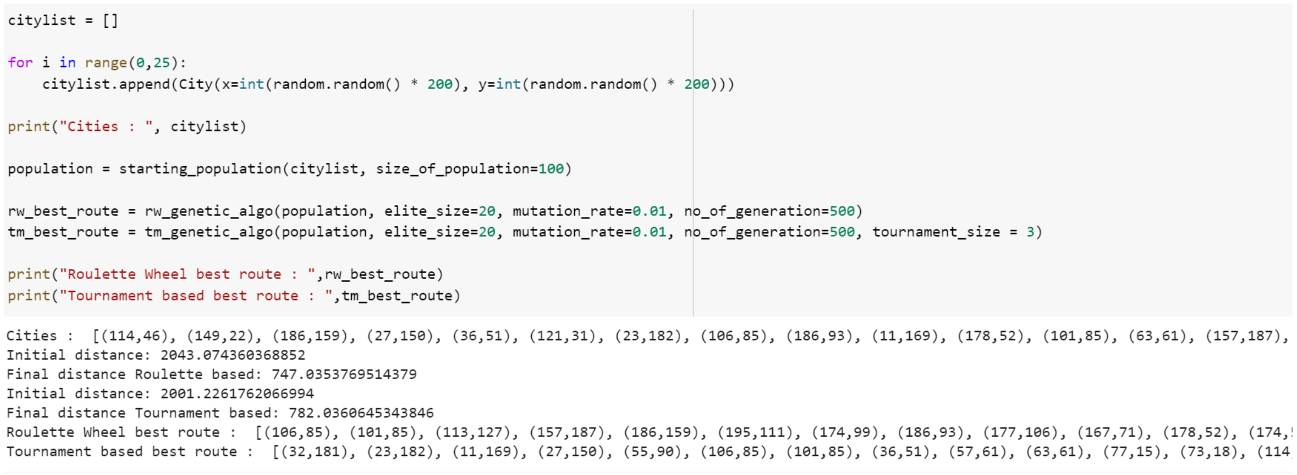
The below is for applying the same type of single point crossover to all the population except the elite members we selected at starting.this will do the all crossover of all the population.



```python
def mutate(population_instance, mutation_rate):
    for swapped in range(len(population_instance)):
        if(random.random() < mutation_rate):
            swapWith = int(random.random() * len(population_instance))

            city1 = population_instance[swapped]
            city2 = population_instance[swapWith]

            population_instance[swapped] = city2
            population_instance[swapWith] = city1
    return population_instance


def mutate_population(population, mutation_rate):
    mutated_population = []

    for ind in range(len(population)):
        mutated_individual = mutate(population[ind], mutation_rate)
        mutated_population.append(mutated_individual)
    return mutated_population
```

This is the part of Algo where mutation happens and the above code is for mutation

In this the above code part the mutation will takes place according to the mutation rate value and if at a city we will generate a random value and if that random value is less than our mutation rate, we will swap that city with any random city in the whole population. And in 2nd part of code mutation for all the population happens and the 2nd part of code will return the value of mutated_population.



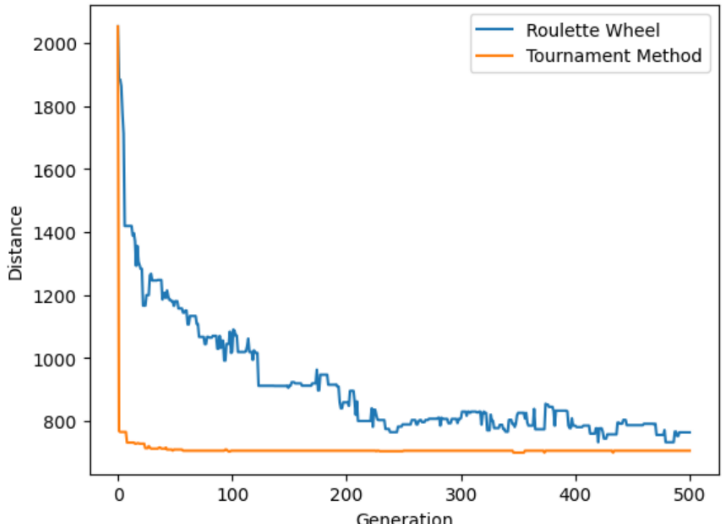
```python
citylist = []

for i in range(0,25):
    citylist.append(City(x=int(random.random() * 200), y=int(random.random() * 200)))

print("Cities : ", citylist)

population = starting_population(citylist, size_of_population=100)

rw_best_route = rw_genetic_algo(population, elite_size=20, mutation_rate=0.01, no_of_generation=500)
tm_best_route = tm_genetic_algo(population, elite_size=20, mutation_rate=0.01, no_of_generation=500, tournament_size = 3)

print("Roulette Wheel best route : ",rw_best_route)
print("Tournament based best route : ",tm_best_route)
```

```
Cities :  [(114,46), (149,22), (186,159), (27,150), (36,51), (121,31), (23,182), (106,85), (186,93), (11,169), (178,52), (101,85), (63,61), (157,187),
Initial distance: 2043.074360368852
Final distance Roulette based: 747.0353769514379
Initial distance: 2001.2261762066994
Final distance Tournament based: 782.0360645343846
Roulette Wheel best route :  [(106,85), (101,85), (113,127), (157,187), (186,159), (195,111), (174,99), (186,93), (177,106), (167,71), (178,52), (174,
Tournament based best route :  [(32,181), (23,182), (11,169), (27,150), (55,90), (106,85), (101,85), (36,51), (57,61), (63,61), (77,15), (73,18), (114
```

This is the Main function and for the code and it contains the value for the Distance in both roulette-wheel method and Tournament method.

It displays the city coordinates it generated and Initial distance and final distance based on the strategy. And it also displays the routes coordinates it is traversing while the Algo running whether it is roulette wheel route coordinates or Tournament based route coordinates.

```
geneticAlgorithmPlot(citylist, size_of_population=100, elite_size=20, mutation_rate=0.01, no_of_generation=500, tournament_size = 3)
```



This is plotting of Roulette wheel method and Tournament method for the 500 iterations with mutation rate of 0.01 and tournament size is 3 with population of 100. And the graph is plotted between the Distance vs Generation.

Conclusion:

From the above observations, we can clearly say tournament based method is a better method as compared to roulette wheel method.

Contributions:

We both discussed the Different methods of solving in genetic algorithm and worked on it together and discussed the way of writing the Algorithm of different methods and also the pattern of writing the Report based on that.

Team members:

CS21B2026-P.Veera Abhiram

CS21B2037-S.Vishnu Teja