Travelling Salesman Problem Using Genetic Algorithm

Achyut Agarwal-211IT003
Department of Information Technology
National Institute of Technology Karnataka
Surathkal, India
achyutagarwal.211it003@nitk.edu.in

Chirag Rajput-211IT018
Department of Information Technology
National Institute of Technology Karnataka
Surathkal, India
chiragrajput.211it018@nitk.edu.in

Garvit Goyal-211IT021
Department of Information Technology
National Institute of Technology Karnataka
Surathkal, India
garvitgoyal.211it021@nitk.edu.in

Khushi Gadling-211IT031
Department of Information Technology
National Institute of Technology Karnataka
Surathkal, India
khushigadling.211it031@nitk.edu.in

Siddharth Kelkar-211IT067
Department of Information Technology
National Institute of Technology Karnataka
Surathkal, India
siddharthkelkar.211it067@nitk.edu.in

***Abstract*—The Travelling Salesman Problem (TSP) is a well-known optimization problem in computer science and operations research, with applications in various fields such as logistics, transportation, and network routing. It involves finding the shortest possible route for a salesman to visit a given set of cities and return to the starting city, while visiting each city exactly once. The combinatorial nature of the problem makes it challenging to solve for large problem instances. This report focuses on solving the TSP using a Genetic Algorithm (GA) approach. Genetic Algorithms are inspired by the principles of natural evolution and employ a population-based search method to iteratively improve a set of candidate solutions. The genetic operators such as crossover and mutation are applied to the population, mimicking the natural processes of reproduction and genetic variation. The fitness of each individual in the population is evaluated based on the total distance travelled, and selection mechanisms guide the evolution of the population towards better solutions over generations.**

***Index Terms*—travelling salesman problem, genetic algorithm, combinatorial optimization, crossover, mutation**

I. Introduction

The Travelling Salesman Problem (TSP) is a widely studied and renowned problem in the field of combinatorial optimization. It poses a fundamental challenge of finding the shortest possible route for a salesman to visit a given set of cities and return to the starting city, while visiting each city exactly once. The problem's practical applications span across various domains, including logistics, transportation, network routing, and resource allocation.

The TSP's significance lies in its complexity and the computational difficulty associated with solving it optimally. As the number of cities increases, the number of possible routes grows exponentially, rendering an exhaustive search infeasible for all but small problem instances. The TSP is classified as an NP-hard problem, indicating that no known polynomial-time algorithm exists to solve it exactly.

The objective in solving the TSP is to minimize the total distance travelled by the salesman, which represents the cost or time required to complete the tour. This optimization problem has captivated researchers for decades, leading to the development of various approximation algorithms, heuristics, and metaheuristics to find near-optimal solutions efficiently.

The TSP's theoretical significance extends beyond its practical applications, as it serves as a benchmark problem for testing the performance of optimization algorithms and evaluating their efficiency and effectiveness. It challenges researchers to devise innovative strategies that balance exploration and exploitation of the solution space to achieve high-quality solutions within reasonable time constraints.

II. Common Solutions to the Problem

A. Brute Force

The brute force approach to solving the Travelling Salesman Problem (TSP) involves exhaustively evaluating all possible permutations of city visitation orders to find the shortest route. It systematically generates and evaluates every permutation of the cities and calculates the total distance for each permutation. The permutation with the minimum distance represents the optimal solution to the TSP.

The steps involved in the brute force solution to the TSP can be summarized as follows:

Generate all possible permutations: Enumerate all possible orders in which the cities can be visited, considering that each city should be visited exactly once.

Calculate the total distance for each permutation: For each generated permutation, compute the total distance required to travel the entire route, including the distance from the last city back to the starting city.

Find the permutation with the minimum distance: Compare the distances calculated for each permutation and identify the permutation that yields the minimum total distance.

Output the optimal solution: Once the permutation with the minimum distance is determined, output the corresponding order of city visitation as the optimal solution to the TSP.

The brute force approach guarantees finding the optimal solution to the TSP since it exhaustively evaluates all possible permutations. However, its major drawback is its computational complexity, which increases dramatically with the number of cities. As the number of cities grows, the number of permutations to evaluate becomes prohibitively large. For instance, with n cities, there are (n-1)! possible permutations to consider, resulting in an exponential time complexity.

Due to the combinatorial explosion of possible solutions, the brute force approach is only feasible for small problem instances with a limited number of cities. For larger instances, more efficient algorithms, such as heuristic methods or metaheuristics like Genetic Algorithms, are preferred to obtain near-optimal solutions within a reasonable amount of time.

B. Genetic Algorithms

The Genetic Algorithm (GA) as a computational intelligence method is a search technique used to solve the Travelling Salesman Problem (TSP). It is inspired by the principles of natural evolution and employs a population-based search method to iteratively improve a set of candidate solutions.

The steps involved in applying a Genetic Algorithm to the TSP can be summarized as follows:

1) Initialization: Generate an initial population of individuals, where each individual represents a potential solution to the TSP. The individuals are typically encoded as permutations of the cities, indicating the visitation order.

2) Fitness Evaluation: Evaluate the fitness of each individual in the population. In the TSP context, the fitness is typically defined as the inverse of the total distance travelled. It represents how well an individual performs in terms of minimizing the TSP objective.

3) Selection: Select a subset of individuals from the population for reproduction based on their fitness. The selection process is often guided by the concept of ”survival of the fittest,” where individuals with higher fitness have a higher chance of being selected.

4) Crossover: Apply crossover operators to the selected individuals to create offspring. Crossover involves exchanging genetic information between two parents to generate new solutions. In the TSP, crossover can be performed by inheriting segments of the visitation order from both parents.

5) Mutation: Introduce random changes to the offspring solutions through mutation operators. Mutation helps explore new regions of the solution space by introducing small alterations to the visitation order. In the TSP, mutation can involve swapping or randomly changing the order of a few cities.

6) Population Update: Combine the offspring solutions with the existing population, creating a new generation. This new generation replaces the previous population, forming the basis for the next iteration of the algorithm.

7) Termination Criterion: Repeat steps 2-6 for a certain number of iterations or until a termination criterion is met. The termination criterion can be a specific number of generations or reaching a satisfactory solution quality.

By iteratively applying the steps above, the Genetic Algorithm explores the solution space, favoring individuals with higher fitness values and gradually improving the population over generations. The algorithm's performance depends on various factors, such as the selection mechanisms, crossover and mutation operators, population size, and termination criteria. Appropriate choices for these components are crucial for balancing exploration and exploitation to achieve high-quality solutions efficiently.

The Genetic Algorithm approach to the TSP offers advantages over the brute force approach by providing a scalable and efficient way to approximate near-optimal solutions for larger problem instances. Although the solutions obtained by the Genetic Algorithm may not be guaranteed to be optimal, they often provide good solutions within reasonable time frames, making it a widely used technique for solving the TSP and other combinatorial optimization problems.

III. Solution Using Genetic Algorithm

1. Population Size – When the algorithm starts the initial number of random tours that are created is known as population size. When the population size is larger it will takes longer time to find the result. A small size of population increases the chance in the population that every tour will eventually look the same. This increases the chance that the best solution will not be found.

2. Neighborhood - This number of tours are randomly chosen for each generation from the population. The two best tours become the parents. The worst two tours are replaced by the children. When the group size is high it increases the chance for good tours will be selected as a parent. A large group size will cause the algorithm to run faster, but it might not find the best solution.

3. Mutation Percentage - The percentage that each child after crossover will undergo mutation .When a tour is mutated, one of the cities is randomly moved from one point in the tour to another.

4. Nearby Cities – As the genetic algorithm is a part of a greedy initial population, it will prefer to link cities that are nearer to each other to make the initial tours.

5. Nearby City Odds percentage - This is the percent chance that any one link in a random tour in the initial population will prefer to use a nearby city instead of a completely random city.

If the GA chooses to use a nearby city, then there is an equally random chance that it will be any one of the cities from the previous parameter.
6. Maximum Generations - How many crossovers are run before the algorithm is terminated.
First, create a group of many random tours in what is called a population. This algorithm uses a greedy initial population that gives preference to linking cities that are close to each other. Second, pick 2 of the better (shorter) tours parents in the population and combine them to make 2 new child tours. Hopefully, these children tour will be. A small percentage of the time, the child tours is mutated. This is done to prevent all tours in the population from looking identical. The new child tours are inserted into the population replacing two of the longer tours. The size of the population remains the same. New children tours are repeatedly created until the desired goal is reached. The accuracy of solution in TSP will depend upon factors such as: Speed and Population Size

IV. Methodology

The proposed methodology employs a simple Genetic Algorithm (GA) to address the problem at hand. The GA begins by generating an initial population of strings, referred to as the gene pool. Subsequently, three operators are applied in succession to create improved populations in successive generations. The reproduction operator copies strings to the next generation based on their objective function value. The crossover operator combines randomly selected pairs of strings, yielding new strings. The mutation operator introduces occasional random alterations to the value at specific positions within a string. Among these operators, crossover, in conjunction with reproduction, is considered the most influential in the GA search process. Mutation serves to diversify the search space and safeguard against the potential loss of genetic material resulting from reproduction and crossover. Consequently, a low probability is assigned to the application of mutation, while a high probability is assigned to crossover. The proposed GA methodology utilizes the key operators of reproduction, crossover, and mutation to iteratively improve the populations in successive generations.

The steps of the algorithm are outlined as follows:

1) Random Initialization: Generate the initial population of individual strings representing potential solutions to the TSP.

2) Fitness Assignment: Assign fitness values to each chromosome in the population using the fitness criteria measure F(x) = 1/x, where x represents the total cost of the string.

3) Crossover: Create a new offspring population by applying the crossover operator.

4) Mutation: Mutate the resulting offspring if necessary. Ensure that the fitness value of the offspring population is higher than that of the parents.

5) Iterative Improvement: Repeat steps 3 and 4 until an optimal solution to the TSP is obtained. Continuously generate new offspring populations through crossover and mutation while aiming for improved fitness values.

V. Result

A. Implementation of Brute Force Solution

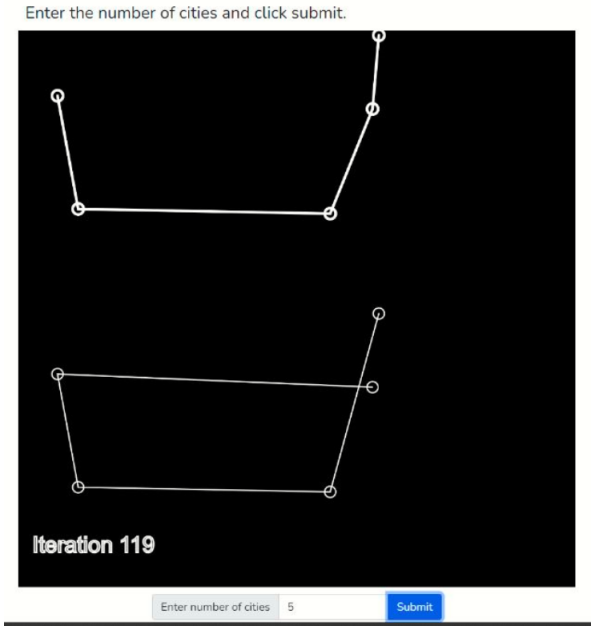

Fig. 1. Brute Force

B. Implementation of Genetic Algorithm Solution

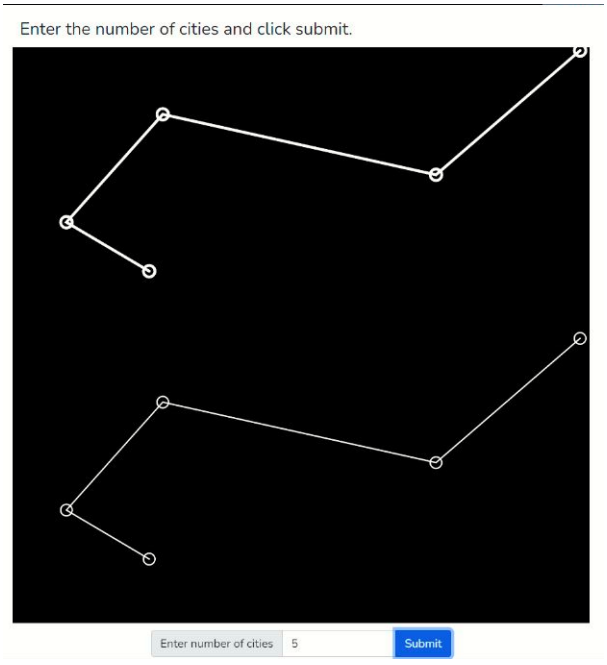

Fig. 2. Genetic Algorithm

The results obtained from comparing the genetic algorithm approach to the brute force method for solving the Traveling Salesman Problem (TSP) revealed distinct differences. The genetic algorithm exhibited considerably faster computation

times compared to the brute force approach. Additionally, while the brute force method guarantees an optimal solution by exhaustively evaluating all possible permutations, the genetic algorithm provided good-quality solutions within a reasonable timeframe, albeit without the guarantee of optimality. This trade-off between computation time and solution quality makes the genetic algorithm a practical choice for tackling large-scale TSP instances where the brute force method becomes computationally infeasible.

VI. Conclusion

Genetic algorithms have made significant contributions to solving the Traveling Salesman Problem (TSP), resulting in some of the best-known solutions to date. Our future work will primarily focus on the analysis of the survivor selector, aiming to generate further improvements. We have devised ideas for influencing the mutation factor based on fitness and exploring different crossover mechanisms.

In general, genetic algorithms have demonstrated their suitability for effectively addressing the challenges associated with the TSP. While no superior solution to the TSP has been found using genetic algorithms thus far, it is important to note that many of the existing best solutions have been discovered through genetic algorithm approaches.

Continued research endeavors will involve refining the performance of genetic algorithms by optimizing the survivor selector, implementing strategies to adapt the mutation factor based on fitness, and investigating various crossover techniques. These endeavors will contribute to the ongoing progress in solving the TSP using genetic algorithms.

In summary, genetic algorithms have proven to be valuable tools in solving the TSP. While they have yet to surpass the current best solutions, they continue to deliver effective and promising results. The exploration of alternative methodologies and advancements will pave the way for even better solutions in the future.

References

[1] Goldberg, Koza, Michalewicz and Beasley ”Potvin ,Jean-Yves ”new optimization technique for genetic algorithms” (Addison-Wesley, 1989)

[2] Naveen kumar, Karambir and Rajiv Kumar, “A Genetic Algorithm Approach To Study Travelling Salesman Problem”, Journal of Global Research in Computer Science, 2012, Vol. 3, No. (3).

[3] S.N.Sivanandam , S.N.Deepa “Introduction to Genetic Algorithms” Springer-Verlag Berlin Heidelberg 2008

[4] Saloni Gupta,Poonam Panwar “Solving Travelling Salesman Problem Using Genetic Algorithm” International Journal of Advanced Research in Computer Science and Software Engineering Volume 3, Issue 6, June 2013.