Department of Computer Engineering

Bu-Ali Sina University

Evolutionary Computing Course

Assignment 1: Solving TSP with an Evolutionary Algorithm

By:

Ali Nafisi

Course Professor:

Professor Hassan Khotanlou

Fall 2023

TABLE OF CONTENTS

1 INTRODUCTION

The Traveling Salesman Problem (TSP) is a classical optimization challenge with widespread applications in various fields, ranging from logistics and transportation to network design and manufacturing. The core objective of the TSP is to determine the most efficient route that visits a set of cities exactly once, minimizing the total distance traveled. As the number of cities increases, the complexity of finding an optimal solution grows exponentially, making the TSP a well-known NP-hard problem.

In light of the computational challenges posed by the TSP, this project aims to explore the application of Evolutionary Algorithms (EAs) as a promising approach for solving this intricate problem. Evolutionary Algorithms draw inspiration from the principles of natural selection and genetics to iteratively evolve a population of candidate solutions. This iterative process mimics the survival-of-the-fittest mechanism, gradually converging towards optimal or near-optimal solutions.

The utilization of Evolutionary Algorithms in solving the TSP introduces a dynamic and adaptive methodology for generating and refining solutions. Unlike traditional algorithms that may struggle with the combinatorial explosion of possible solutions, EAs offer a robust framework to explore the solution space efficiently. Through the iterative evolution of potential solutions, Evolutionary Algorithms can navigate the vast search space of the TSP, discovering high-quality solutions that approach or achieve optimality.

This project delves into the theoretical foundations of the TSP, outlining the inherent challenges and complexities associated with finding optimal solutions. Subsequently, it investigates the principles of Evolutionary Algorithms and their suitability for addressing combinatorial optimization problems such as the TSP. By combining the TSP's real-world significance with the adaptive power of Evolutionary Algorithms, this research endeavors to contribute to the development of effective and scalable solutions for this challenging problem.

2 METHODOLOGY

2.1 Representation

In tackling the Traveling Salesman Problem (TSP), it becomes imperative to devise a systematic representation for the cities and the entire route. An effective approach involves assigning each city a unique integer, encapsulating the entire route within an array of integers, with the array's length denoted as 'n' (representing the total number of cities). However, the TSP imposes a crucial condition, necessitating the visitation of each city exactly once. To adhere to this constraint, our array must consist of non-repeating integers, each ranging from 0 to n-1. Among the available representations, the Permutation array emerges as the optimal choice, seamlessly meeting all the specified conditions of the TSP problem.

2.2 Crossover Methods

In the context of almost every evolutionary algorithm, the fundamental requirement involves establishing a population of individuals. With each successive generation, a crucial step entails generating a set number of offspring derived from selected parents within the existing population. Among the various methods employed for generating offspring, one prominent approach is the crossover method.

2.2.1 Partially Mapped Crossover (PMX)

Partially Mapped Crossover (PMX)[1] was introduced by Goldberg and Lingle as a recombination method for the Traveling Salesman Problem (TSP). It has become widely adopted for problems involving adjacency. Various versions of PMX have been proposed, and for this explanation, we follow Whitley's definition. The process involves selecting two random crossover points. The segment between these points from the first parent is copied to create the first offspring. Subsequently, elements in the same segment of the second parent that haven't been copied are identified. For each of these elements, the corresponding element from the first parent is found in the offspring. The element from the second parent is then placed in the position occupied by its counterpart from the first parent. If that position is already filled, a similar swap is performed to maintain consistency. After addressing the crossover segment, the remaining positions in the offspring are filled with elements from the second parent. The second child is created in a similar manner with the roles of the parents reversed. Figure 1 shows the how the PMX crossover works.

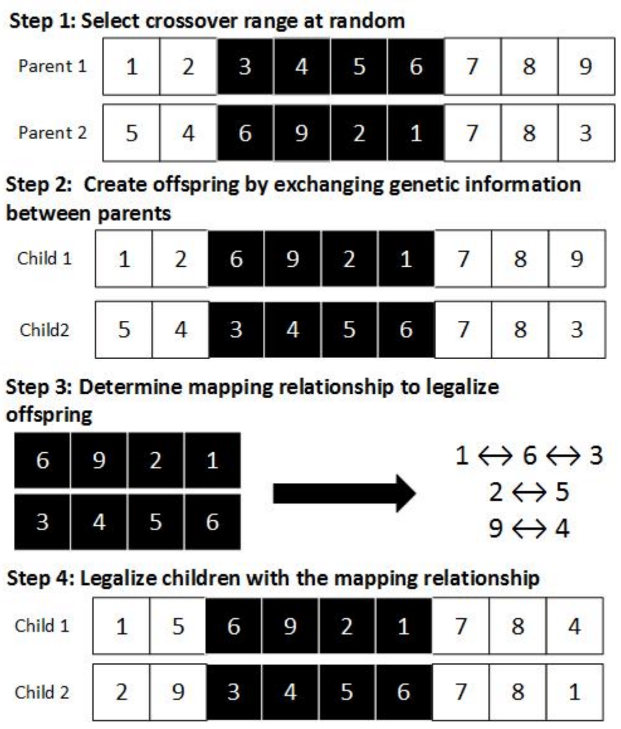

Step 1: Select crossover range at random
Parent 1
1 2 3 4 5 6 7 8 9
Parent 2
5 4 6 9 2 1 7 8 3
Step 2: Create offspring by exchanging genetic information between parents
Child 1
1 2 6 9 2 1 7 8 9
Child2
5 4 3 4 5 6 7 8 3
Step 3: Determine mapping relationship to legalize offspring
6 9 2 1
3 4 5 6
1 ↔ 6 ↔ 3
2 ↔ 5
9 ↔ 4
Step 4: Legalize children with the mapping relationship
Child 1
1 5 6 9 2 1 7 8 4
Child 2
2 9 3 4 5 6 7 8 1

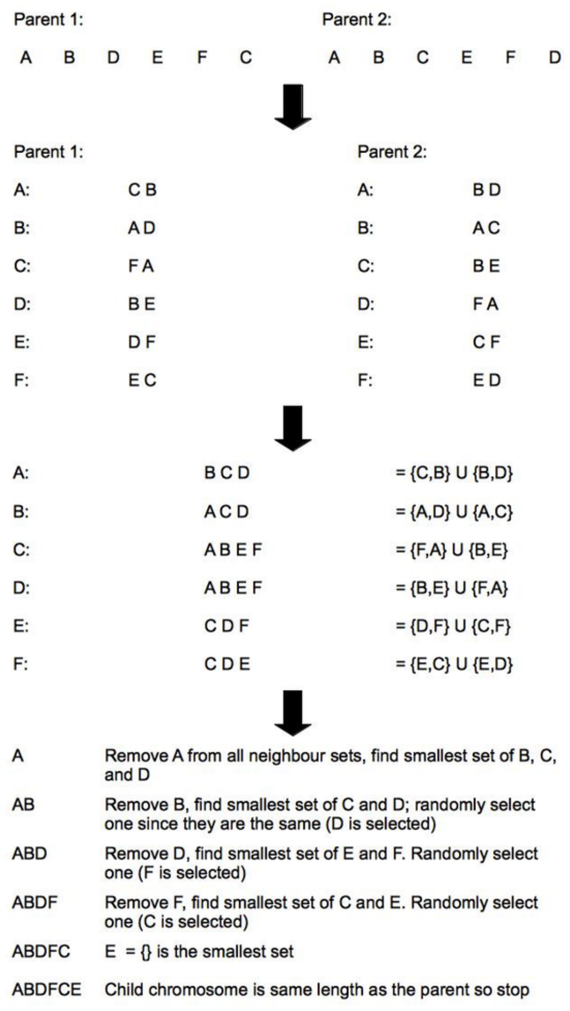
Figure 1: PMX crossover[2]

2.2.2 Edge Crossover

Edge crossover[1] is a method that aims to generate offspring using edges present in one or both parents. The edge-3 crossover, as outlined by Whitley, prioritizes the preservation of common edges. To achieve this, an edge table is created, detailing the connections of each element in the parents. A '+' in the table signifies the presence of an edge in both parents. The process involves randomly selecting an initial element for the offspring, setting it as the current element, and then eliminating references to it in the table. The next element is chosen based on the presence of a common edge or, if none exists, by selecting the entry with the shortest list (randomly breaking ties). If the list becomes empty, the other end of the offspring is considered for extension; otherwise, a new element is randomly chosen. Figure 2 shows the how the Edge crossover works.

2.3 Mutation Methods

Another approach for generating offspring involves the process of mutation. In the subsequent sections, we will elucidate two distinct methods employed for mutation.



Parent 1:

A B D E F C

Parent 2:

A B C E F D

Parent 1:		Parent 2:	
A:	C B	A:	B D
B:	A D	B:	A C
C:	F A	C:	B E
D:	B E	D:	F A
E:	D F	E:	C F
F:	E C	F:	E D

A:	B C D	= {C,B} U {B,D}
B:	A C D	= {A,D} U {A,C}
C:	A B E F	= {F,A} U {B,E}
D:	A B E F	= {B,E} U {F,A}
E:	C D F	= {D,F} U {C,F}
F:	C D E	= {E,C} U {E,D}

A	Remove A from all neighbour sets, find smallest set of B, C, and D
AB	Remove B, find smallest set of C and D; randomly select one since they are the same (D is selected)
ABD	Remove D, find smallest set of E and F. Randomly select one (F is selected)
ABDF	Remove F, find smallest set of C and E. Randomly select one (C is selected)
ABDFC	E = {} is the smallest set
ABDFCE	Child chromosome is same length as the parent so stop

Figure 2: Edge crossover[3]

2.3.1 Swap Mutation

Swap Mutation is a genetic algorithm operator used to introduce diversity in offspring generation. In this mutation method, two randomly selected elements within an individual's chromosome are exchanged, leading to a new configuration. This process helps explore different combinations of genetic material, promoting the exploration of the solution space. Swap Mutation is particularly effective in preventing premature convergence by disrupting existing orderings and allowing the genetic algorithm to explore alternative solutions. By swapping elements, the algorithm can potentially discover novel and beneficial arrangements, contributing to the overall effectiveness of the optimization process. Figure 3 shows the how the swap mutation works.

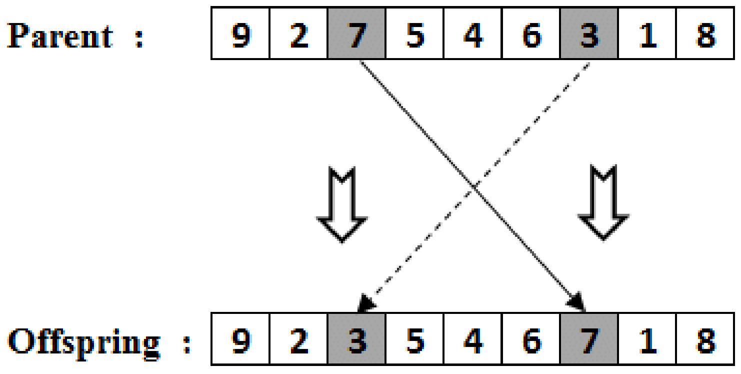


Figure 3: Swap mutation[4]

2.3.2 Insert Mutation

Insert Mutation[1] is a genetic algorithm operator characterized by the repositioning of two randomly chosen alleles within an individual's chromosome. The process involves selecting two alleles at random, and then moving the second allele to a position adjacent to the first one, thereby shuffling the remaining alleles to accommodate the change. This genetic operation introduces variation by altering the sequence of alleles along the chromosome. The insertion of the second allele in proximity to the first disrupts the existing genetic order, promoting the exploration of different genetic configurations. This method is particularly effective in preventing premature convergence by creating diverse genetic arrangements, contributing to the genetic algorithm's ability to discover alternative and potentially improved solutions in optimization tasks. Figure 4 shows the how the insert mutation works.

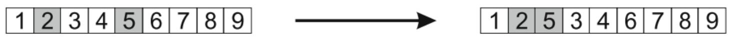


Figure 4: Insert mutation[1]

2.4 Parent and Survivor Selection Methods

In evolutionary algorithms, we first pick parents and create offspring from them. After that, we need to choose which μ individuals will continue in the process. There are different ways to make these selections, and in the next sections, we'll explain two methods for doing this.

2.4.1 Fitness-Proportionate Selection (FPS)

Fitness-Proportionate Selection (FPS), also known as roulette wheel selection, is a method used in evolutionary algorithms to choose individuals for reproduction based on their fitness. The idea is to give individuals with higher fitness a higher chance of being selected as parents. The probability of an individual being chosen is directly proportional to its fitness relative to the total fitness of the population. It's like spinning a roulette wheel, where each individual's "slice" of the wheel corresponds to its likelihood of being selected. This ensures that individuals with better fitness have a greater influence on the next generation, mimicking the natural selection process where more fit individuals are more likely to pass on their genes to the next generation. Equation 1 shows how to calculate selection probabilities for each individual using the FPS method.

$$P_{FPS}(i) = \frac{fitness(i)}{\sum_{j=1}^{\mu} fitness(j)} \tag{1}$$

2.4.2 Linear Ranking Selection

Linear Ranking Selection is a technique used in evolutionary algorithms to select individuals for reproduction based on their relative fitness within the population. The individuals are ranked according to their fitness, and each is assigned a selection probability based on its rank. In this method, a linear transformation is applied to the ranks, often in the form of a linear equation. Individuals with higher fitness ranks are assigned higher selection probabilities, promoting the likelihood of their inclusion in the next generation. This ranking process allows for a balance between selecting individuals with the best fitness and maintaining diversity in the population. Linear Ranking Selection is particularly useful in scenarios where explicit fitness values may vary widely, providing a way to emphasize the importance of relative fitness without being overly sensitive to extreme values. Equation 2 shows how to calculate selection probabilities for each individual using the linear Ranking method.

$$P_{linear-rank}(i) = \frac{2-s}{\mu} + \frac{2i(s-1)}{\mu(\mu-1)} \tag{2}$$

3 EXPERIMENTAL RESULTS

3.1 Dataset

In this project, the Traveling Salesman Problem (TSP) dataset consists of 127 records, each representing the x and y coordinates of a city. The dataset is organized into three columns: "NODE," "X," and "Y." The "NODE" column serves as the identifier for each city, ranging from 1 to 127. The "X" and "Y" columns contain the Cartesian coordinates of the cities, where "X" represents the horizontal position and "Y" represents the vertical position. These coordinates define the spatial distribution of the cities in the TSP instance. The dataset provides a comprehensive set of geographical locations for cities, forming the basis for the TSP algorithm implementation in the project. Table 1 shows the first 5 records of the dataset.

Table 1: The first 5 records of the dataset.

NODE	X	Y
1	9860	14152
2	9396	14616
3	11252	14848
4	11020	13456
5	9512	15776

To facilitate the solution of the Traveling Salesman Problem (TSP), I calculated an $n \times n$ matrix named "distance." This matrix, where n represents the number of cities in the dataset (127 in this case), captures the Euclidean distance between each pair of cities. The Euclidean distance $distance[i][j]$ between city i and city j was computed using the coordinates provided in the original dataset. Specifically, I employed the standard Euclidean distance formula, which is $\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$. The resulting distance matrix encapsulates the spatial relationships and proximity between all cities in the dataset, serving as a crucial input for the TSP algorithm implemented in the project.

3.2 Evaluation Metric

To evaluate each individual, we require a function that assigns a fitness value to each individual. We seek an individual that maximizes this function. Therefore, we calculate the inverse of the total distances along the individual's path, and this becomes our fitness function. Equation 3 illustrates the process for calculating the fitness function.

$$fitness(chromosome) = \frac{1}{\sum_{i=1}^{n-1} distance[chromosome[i]][chromosome[i+1]]} \tag{3}$$

3.3 Result

We executed the algorithm using eight distinct combinations of selection, crossover, and mutation methods, each utilizing a population size of **100** with a **10%** probability of mutation. The total distances achieved by the best individual for each method combination are presented in Table 2 (Individuals with lower total distances have higher fitness). Notably, the algorithm demonstrated superior performance with linear ranking selection compared to FPS, PMX crossover outperformed Edge crossover, and insert mutation proved to be more effective than swap mutation.

Table 2: Results from **3,000** iterations using a population size of **100** and diverse selections, crossovers, and mutations.

Selection method	**Linear ranking (S=1.5)**				**FPS**			
Crossover method	**PMX**		**Edge**		**PMX**		**Edge**	
Mutation method	**Swap**	**Insert**	**Swap**	**Insert**	**Swap**	**Insert**	**Swap**	**Insert**
Total distances (1/fitness)	199810	193842	240012	248311	573769	572466	546658	544642

To ensure the continual improvement of the best individual across generations and avoid overlooking potentially superior solutions, we chart the fitness of the top individual per generation for each method combination. The plots depicting this progression can be observed in Figure 5. The plots unmistakably reveal that employing linear ranking selection leads to a consistent increase in fitness across generations. In contrast, when utilizing fitness-proportionate selection (FPS), there is a noticeable lack of substantial improvement over the generations.

Table 2 reveals that the optimal combination of methods involves employing linear ranking for selection, PMX for crossover, and the insert method for mutation. Our initial algorithm run had **3000** iterations and a population size of **100**. Seeking an enhanced solution for the TSP problem, we conducted a more extensive run with **3000** iterations and an increased population size of **1000** using the identified optimal method combination. The results demonstrated a notable improvement compared to the 100-population size run. As depicted in Figure 6, the total distance of the best individual reached an impressive **141412**.

This is the chromosome of the best solution: [101, 100, 79, 78, 25, 11, 19, 107, 14, 105, 113, 5, 23, 22, 3, 21, 18, 20, 16, 76, 17, 71, 7, 8, 10, 119, 2, 99, 63, 57, 90, 60, 61, 89, 115, 59, 58, 66, 72, 73, 67, 75, 77, 116, 83, 80, 74, 68, 69, 70, 109, 84, 85, 86, 87, 108, 95, 118, 62, 125, 81, 82, 97, 96, 122, 94, 27, 121, 31, 28, 32, 24, 37, 40, 35, 36, 50, 49, 4, 51, 123, 64, 98, 91, 88, 103, 124, 112, 65, 54, 46, 52, 117, 47, 93, 92, 126, 106, 110, 111, 45, 48, 55, 120, 34, 42, 33, 38, 41, 39, 43, 102, 44, 53, 56, 1, 0, 15, 13, 29, 26, 30, 104, 6, 9, 114, 12].

To assess the efficacy of our solution, establishing upper and lower bounds for total distances is crucial. The initialization of a random population yields a total distance around **633,407**, serving as our upper bound. For the lower bound, we consider that each route must pass through all cities, leading to a sorted summation of the first $n-1$ distances above the main diagonal of the

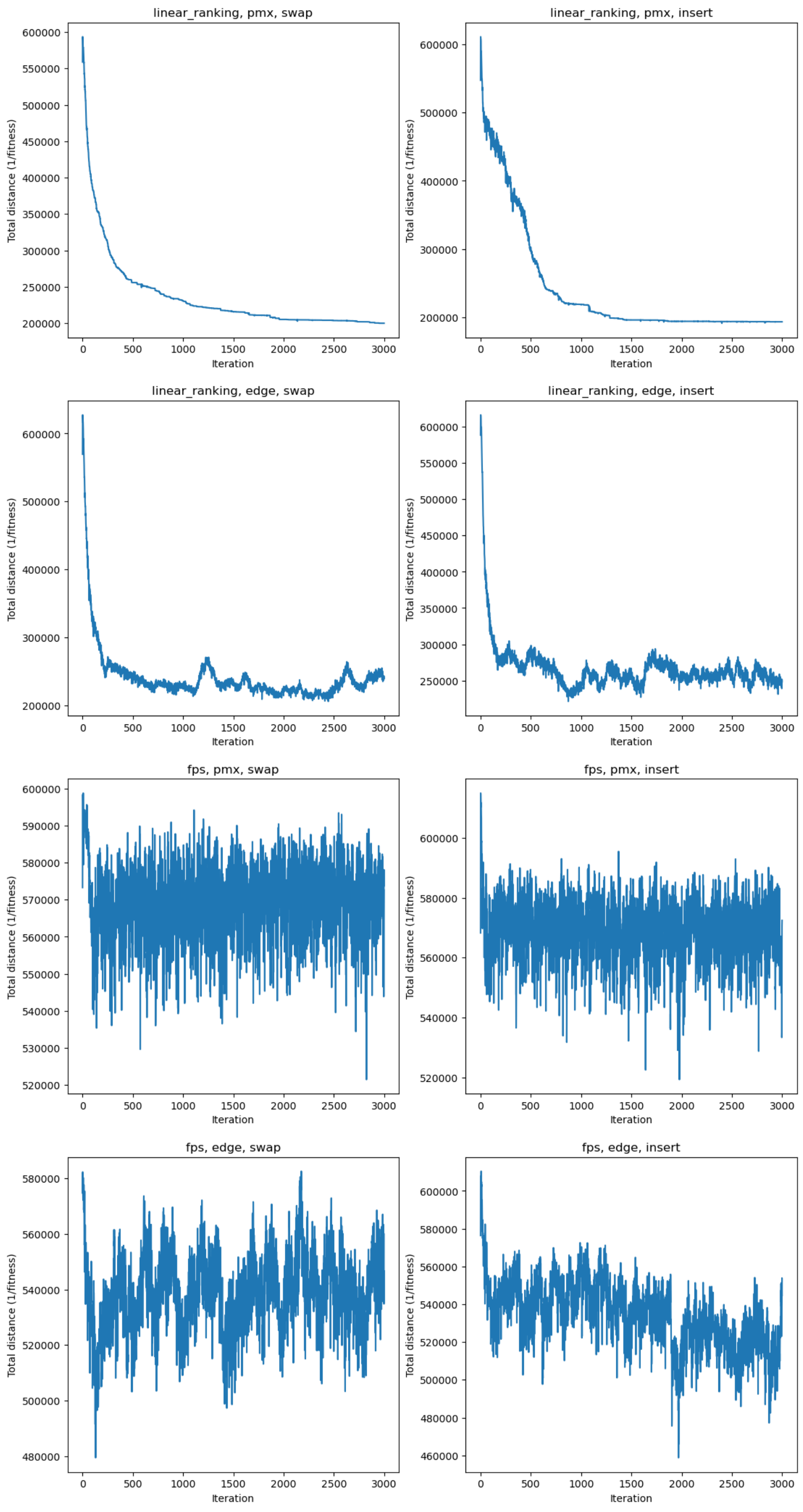


Figure 5: Plots showing total distances of the best individual per generation with varied selections, crossovers, and mutations.

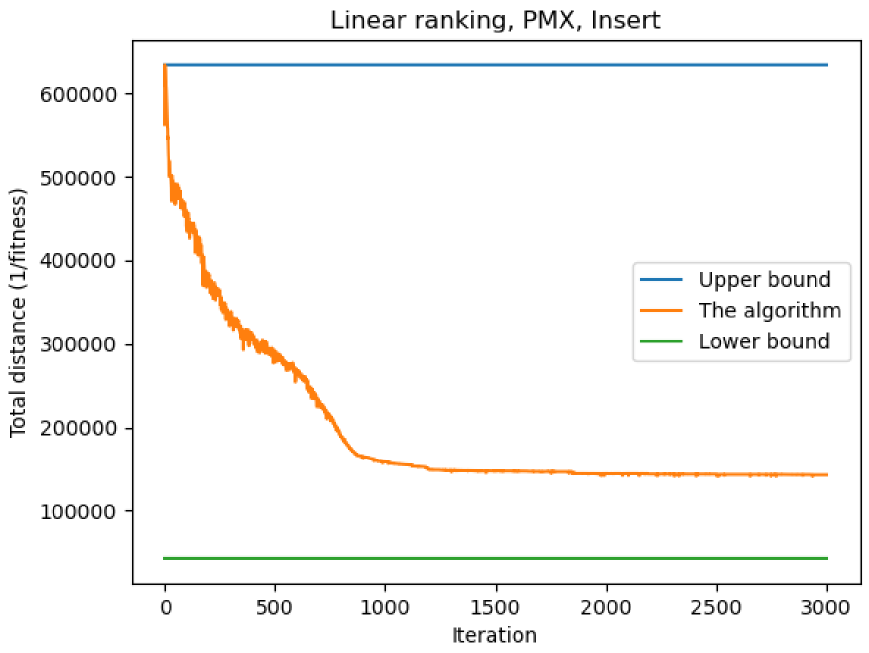


Figure 6: Plot depicting total distances of the top individual per generation using the optimal method combination.

$distance$ matrix, resulting in a lower bound of **42,908**. It is important to note that the optimal solution's total distance unequivocally surpasses this lower bound. Figure 6 visually depicts the algorithm's progress, initially close to the upper bound and progressively converging towards the lower bound in each generation. By the final generation, the algorithm significantly narrows the gap with the lower bound, showcasing substantial improvement over the upper bound.

4 CONCLUSION

In summary, our project set out to tackle the Traveling Salesman Problem (TSP) using an evolutionary algorithm, aiming to determine optimal choices for representation, crossover, mutation, and selection. Through experimentation, we utilized permutation arrays to represent TSP solutions, comparing edge crossover with partially mapped crossover and determining the latter as a more effective approach. In the realm of mutations, we evaluated swap and insert methods, ultimately identifying insert mutation as better suited for this problem. Similarly, our exploration of selection strategies involved testing linear ranking and fitness-proportionate selection, with linear ranking emerging as the more effective choice. In the culmination of these efforts, we successfully arrived at a good solution for the TSP problem.

REFERENCES

[1] A. E. Eiben and J. E. Smith, *Introduction to evolutionary computing*. Springer, 2015.

[2] B. Desjardins, R. Falcon, R. Abielmona, and E. Petriu, “Planning robust sensor relocation trajectories for a mobile robot with evolutionary multi-objective optimization,” *Computational Intelligence in Wireless Sensor Networks: Recent Advances and Future Challenges*, pp. 179–210, 2017.

[3] B. Ferriman and C. Obimbo, “Solving for the rc4 stream cipher state register using a genetic algorithm,” *International Journal of Advanced Computer Science and Applications*, vol. 5, no. 5, 2014.

[4] A. L. EL IDRISSI, C. Tajani, and M. SABBANE, “Analyzing the performance of mutation operators to solve the fixed charge transportation problem,”