I.J. Intelligent Systems and Applications, **2025, 3, 15-33**
Published Online on June 8, 2025 by MECS Press (http://www.mecs-press.org/)
DOI: 10.5815/ijisa.2025.03.02

Solving Traveling Salesman Problem Through Genetic Algorithm with Clustering

Md. Azizur Rahman*
Mathematics Discipline, Science, Engineering and Technology School, Khulna University, Khulna, 9208, Bangladesh
E-mail: mdazizur@math.ku.ac.bd
ORCID iD: https://orcid.org/0000-0003-3950-9528
*Corresponding author

Kazi Mohammad Nazib
Mathematics Discipline, Science, Engineering and Technology School, Khulna University, Khulna, 9208, Bangladesh
E-mail: kazinazib1999@gmail.com
ORCID iD: https://orcid.org/0009-0001-5956-3313

Md. Rafsan Islam
Department of Mathematics, Hamdard University Bangladesh, Gazaria, Munshiganj, 1510, Bangladesh
E-mail: rafsan@hamdarduniversity.edu.bd
ORCID iD: https://orcid.org/0009-0003-1536-216X

Lasker Ershad Ali
Mathematics Discipline, Science, Engineering and Technology School, Khulna University, Khulna, 9208, Bangladesh
E-mail: ershad@math.ku.ac.bd
ORCID iD: https://orcid.org/0000-0003-4987-9493

Received: 11 November 2024; Revised: 28 January 2025; Accepted: 05 February 2025; Published: 08 June 2025

Abstract: The Traveling Salesman Problem (TSP) is a well-known NP-hard combinatorial optimization problem, commonly studied in computer science and operations research. Due to its complexity and broad applicability, various algorithms have been designed and developed from the viewpoint of intelligent search. In this paper, we propose a two-stage method based on the clustering concept integrated with an intelligent search technique. In the first stage, a set of clustering techniques - fuzzy c-means (FCM), k-means (KM), and k-mediods (KMD) - are employed independently to generate feasible routes for the TSP. These routes are then optimized in the second stage using an improved Genetic Algorithm (IGA). Actually, we enhance the traditional Genetic Algorithm (GA) through an advanced selection strategy, a new position-based heuristic crossover, and a supervised mutation mechanism (FIB). This IGA is implemented to the feasible routes generated in the clustering stage to search the optimized route. The overall solution approach results in three distinct pathways: FCM+IGA, KM+IGA, and KMD+IGA. Simulation results with 47 benchmark TSP datasets demonstrate that the proposed FCM+IGA performs better than both KM+IGA and KMD+IGA. Moreover, FCM+IGA outperforms other clustering-based algorithms and several state-of-the-art methods in terms of solution quality.

Index Terms: Traveling Salesperson Problem (TSP), Genetic Algorithm (GA), Fuzzy C-means, K-means, K-medoids, Local Search Mechanism.

1. Introduction

The Traveling Salesman Problem (TSP) is one of the most well-known nondeterministic polynomial hard (NP-hard) problems in computer science and operations research, with no quick solution. In the 18th century, it was investigated by the Irish mathematician Sir William Rowan Hamilton and the British mathematician Thomas Penyngton Kirkman. The TSP has diverse applications across various sectors, including logistics and supply chain management, circuit board design, DNA sequencing, computer wiring, network design, production planning, manufacturing, and more. The TSP involves determining the most efficient and shortest way for a traveler to visit all locations exactly once and return to the starting point at the end of the journey. It resembles the Hamiltonian circuit, where each node has exactly two edges: the nodes represent different destinations, and the edges correspond to the travel routes of the

salesman. While understanding the TSP is straightforward, the complexity of the problem makes it difficult to find the optimal solution. For an n cities problem, there are $(n-1)!/2$ feasible solutions [1]. For example, if there are 10 cities, there will be 181,440 possible solutions, whereas, the number of possible solutions exceeds 40 billion in case of 15 cities problem. Therefore, searching for the shortest path from all these solutions is highly challenging.

There are several approaches to solving the TSP. Finding the optimal solution from all feasible options is known as the brute-force approach [2]. Other well-known methods in this category include Branch and Bound [3], Cutting Planes [4], and others. These are considered exact methods but take too long to solve the TSP and are suitable only for small instances. Lin-Kernighan Heuristics [5] and Match Twice and Stitch [6] are heuristic approaches used to search for solutions to the TSP. While these methods provide solutions in a reasonable amount of time, the solutions may not be optimal. Moreover, they are not ideal for general TSP datasets. Bio-geography migration algorithms [7] or nature-inspired meta-heuristic algorithms [8] are quite well-known for solving the TSP. Local Search Algorithms [9] and Meta-heuristic Algorithms [10] are popular nowadays. Methods like the Zero Suffix Method [11], Genetic Algorithm [12], ant colony optimization [13], bee colony optimization [14], Particle swarm optimization algorithm [15], Greedy Heuristic [16] and List based Simulated Annealing Algorithm [17] are familiar meta-heuristic approaches. These methods have gained popularity because they often provide acceptable solutions within a reasonable time. However, they frequently suffer from premature convergence and may yield non-deterministic poor solutions.

Recently, hybrid or mixed algorithms have been designed and improved for the TSP. These algorithms are the Genetic Algorithm with Learning Automata [18], Simulated Annealing Algorithm with three neighbor generation [19], adaptive simulated annealing algorithm with greedy search [20], Improved Simulated Annealing Algorithm with Sequential Access Restrictions [21], Intelligent Route Construction Algorithm [22], Simulated Annealing and Gene-Expression [23], List based Simulated algorithm with crossover [24], Genetic simulated annealing ant colony system with particle swarm optimization [25]. Moreover, machine learning-based algorithms like neural networks [26] have also been introduced to solve the TSP. Actually, meta-heuristic techniques do not guarantee an optimal solution to the TSP problem. Besides that, the parameters involved in the process significantly influence the quality of the solution. In addition, the generation of the initial solution greatly impacts the quality of the solution in many meta-heuristic algorithms. Typically, the initial population in common meta-heuristic algorithms is generated randomly. As a result, establishing a decent initial solution is critical.

On the other hand, clustering plays a significant role in solving the TSP by breaking it down into smaller, more manageable sub-problems. Rather than treating the entire set of cities as one unit, clustering divides them into groups based on proximity or other metric, such as geographic regions or density. This division allows TSP solutions to be applied within each cluster, reducing computational complexity and enabling faster, more efficient solutions. Once optimal or near-optimal paths are determined for each cluster, they can be combined to form a global solution, typically using techniques to minimize inter-cluster travel. Clustering thus enhances scalability, improves solution accuracy in large-scale problems, and reduces the overall time required for computation. Motivated by this, we propose a two-stage approach for solving the symmetric TSP using a Genetic Algorithm (GA) combined with clustering concept. First, the TSP data is clustered utilizing three clustering techniques: fuzzy c-means (FCM), k-means (KM), and k-medoids (KMD). These clusters are then arranged side by side to generate the initial populations. An improved Genetic Algorithm (IGA) is subsequently applied to these initial populations to find the optimal TSP solution. In fact, the GA is enhanced through local search mechanism based on first-improvement manner during the mutation phase. Additionally, a position-based heuristic crossover approach is introduced and applied to generate new populations. Moreover, order crossover, heuristic crossover, and position-based heuristic crossover are adopted during the experiment of our proposed technique. The proposed algorithm is tested on 47 well-known TSP instances and its performance is compared with other clustering-based algorithms and several TSP solving state-of-the-art algorithms.

This paper is organized as follows: the mathematical description of the TSP is provided in Section 2. Prior studies related to this work are presented in Section 3, while the detailed proposed method for solving the TSP is explained in Section 4. Section 5 covers the simulation environment configuration, simulation results, and analysis from multiple perspectives, including comparisons with other algorithms. The convergence curves and statistical analyses are also presented in this section. Finally, the conclusion of this paper is offered in Section 6.

2. Mathematical Model of TSP

The Travelling Salesman Problem (TSP) is a classical optimization problem in both combinatorial mathematics and computer science. It involves finding the shortest possible route for a salesman to visit a set of cities, ensuring each city is visited exactly once, with a return to the starting point. The TSP is widely studied due to its simple formulation and its complex nature as an NP-hard problem. A common approach to representing the TSP is through Graph Theory. In this framework, the TSP can be described by a weighted graph $G=(V,E)$, where $V=\{v_1, v_2, \cdots, v_n\}$ represents the set of vertices (or nodes), and $E=\left\{e_{ij}=(v_i,v_j): v_i,v_j \in V; i \neq j\right\}$ denotes the set of weighted edges. Each edge e_{ij} is assigned a distance function $d(i,j)$, which represents the weight of the edge in graph G. In the symmetric TSP, $d(i,j)=d(j,i)$, while in the asymmetric TSP, at least one edge has $d(i,j)\neq d(j,i)$. In both cases, $d(i,i)=0$. The

mathematical model of the TSP problem can be expressed as follows [24]:

The objective function (Z) of the TSP:

$$Z = \min \sum_{i=1}^{n} \sum_{j=1, j \neq i}^{n} d(i, j) x_{ij} \tag{1}$$

The decision variable x_{ij}:

$$x_{ij} = \begin{cases} 1, & \text{if travels city from } i^{th} \text{ to } j^{th} \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

with the following constraints:

$$\sum_{i=1, j \neq i}^{n} x_{ij} = 1, \quad j \in V \tag{3}$$

$$\sum_{j=1, i \neq j}^{n} x_{ij} = 1, \quad i \in V \tag{4}$$

$$\sum_{i, j \in S, i \neq j}^{n} x_{ij} \leq |S| - 1, \qquad \forall S \subset V \tag{5}$$

The variable x_{ij} is a binary decision variable; if the salesman travels from the i^{th} city to j^{th}, the decision is made based on Eq. (2). The main objective is to find the shortest path to all cities, which is mathematically formulated in Eq. (1). Constrains Eq. (3) and Eq. (4) confirm that the traveler must visit each city exactly once. The possibility of any sub-tour is eliminated by Eq. (5).

3. Related Works

The Travelling Salesman Problem (TSP) has been extensively studied, and numerous approaches have been proposed to solve it, ranging from exact algorithms to heuristic and metaheuristic techniques. This section focuses on two-phase approaches, which employ different strategies, including clustering, to tackle the TSP problem. Paquete and Stutzle proposed a two-phase local search method for solving the TSP [27]. In the first phase, they generated initial solutions applying a 3-exchange neighborhood and a double bridge move. In the second phase, they improved these initial solutions through a local search algorithm. Their work highlighted the critical role of generating quality initial solutions in solving the TSP effectively. Another two-stage approach, proposed by Rahman and Parvez [28], employed multiple nearest neighbor (NN) and dual nearest neighbor (DNN) methods in the first stage, followed by the application of a simulated annealing (SA) algorithm in the second stage.

Emek et al. proposed a two-phase approach that uses fuzzy c-means clustering to generate initial populations or feasible routes [29]. These initial populations are then utilized in an evolutionary algorithm on each cluster to build an ideal sub-tour. Actually, the sub-tours are subsequently combined side by side to produce optimal routes, providing a solution to the TSP problem. Yoon and Cho suggested a method for generating initial populations after determining the optimum route for TSP, involving three phases [30]. First, data are grouped based on membership values and similarities using fuzzy c-means clustering. Next, these clusters are placed side by side to form the entire TSP route. Finally, a typical genetic algorithm is applied on the generated initial populations to solve the TSP problem. On the other hand, Fuentes et al. clustered the TSP dataset and applied a meta-heuristic local search mechanism to each cluster [31]. These clusters were then combined to form a complete TSP tour. Jaradat et al. initialized the populations with k-means clustering and then implemented the firefly algorithm to solve TSP [32]. In the work of Deng et al., k-means clustering was used for initial population generation in combination with a genetic algorithm to solve the TSP [33]. El-Samak and Ashour proposed affinity propagation clustering on the dataset, followed by the application of a genetic algorithm to solve the TSP [34]. Martikainen and Ovaska evaluated fuzzy programming as an efficient divide-and-conquer method to solve the TSP [35].

Numerous studies have used genetic algorithms to solve the TSP Problem. The articles by Potvin [36] and Larranga et al. [37] provide overviews of genetic algorithm approaches for solving TSP. These papers discuss various selection methods, the elitism approach, and a range of crossover operators, including one-point crossover, partially mapped crossover, cycle crossover, order crossover, order-based crossover, position-based crossover, alternate edges

crossover, edge recombination crossover, heuristic crossover, shortest match crossover, maximal preservative crossover, voting recombination crossover, and alternating position crossover. For mutation, several types of mutation operators are explored, such as swap, local hill-climbing, scramble, displacement mutation, exchange mutation, insertion mutation, and simple inversion mutation.

Chudasama et al. compared three different selection mechanisms of genetic algorithm [38]. Roulette wheel selection is based on the fitness value, while tournament selection involves a random choice from a set of parent candidates. In these two selection methods, the entire generation is replaced by a new generation, which may not yield optimal results. An alternative approach, called elitism, replaces only a portion of the population with new individuals, preserving the best parents for the next generation. Razali and Geraghty described two types of Roulette wheel selection: Proportional Roulette Wheel selection and Rank-based Roulette Wheel selection [39]. On the other hand, the family father approach to the selection method was introduced in the work of Chen et al. [40]. In terms of hybridizing the genetic algorithm, Wang improved the genetic algorithm with two local optimization strategies [41]. He applied the second local optimization in the mutation phase, where he used reversion, and the first local optimization was applied after the mutation phase of the genetic algorithm. Geng et al. proposed a hybrid simulated annealing algorithm in which the simulated annealing algorithm was combined with a greedy search algorithm to solve the TSP [20].

4. Proposed Algorithm

The proposed approach seeks the optimal route for the Traveling Salesman Problem (TSP) through a two-stage process. In the first stage, initial populations are generated using clustering techniques, including k-means, fuzzy c-means, and k-medoids. By clustering, the TSP data is divided into multiple groups, which are then positioned sequentially to form a valid or feasible route for the TSP. This process continues until the required number of unique populations is created, with any duplicate routes removed. In the second stage, an enhanced genetic algorithm refines the populations from the first stage. These initial populations serve as the starting generation for the improved genetic algorithm. Selection is performed using a roulette wheel method with elitism, also known as family-father selection. During the crossover phase, three methods—ordered crossover, heuristic crossover, and position-based heuristic crossover—are tested. A key enhancement of this genetic algorithm is in the mutation phase, where a first-improvement-based (FIB) local search optimizer refines individuals, generating improved solutions from previous ones. Algorithm 1 presents the proposed approach for solving the TSP problem. As shown in the algorithm, DistanceMatrix(d) creates the distance matrix for the data d, and InitializeParameters() sets the initial parameters. The cluster-based initial population, which represents the first stage of the proposed method, is denoted as CPOP in Step 4. Steps 6 to 9 describe the modified genetic algorithm, where the First Improvement-Based Local Searching Optimizer is applied during the mutation phase. The UpdateEntirePop and UpdateElitismPop are two separate operators used to update the population. All of these operators are discussed in the following subsections.

Algorithm 1: Pseudocode of the proposed approach for solving the Traveling Salesman Problems (TSP)

Input: Data and the parameter values
d: data, nump: number of populations, nc: number of clusters, gen: Generation, mp: mutation probability
Output: The best solution and its fitness value
R_{best}: Best Route, F_{best}: Fitness value of the best route
1 **begin**
// Create distance matrix DM
2 DM = DistanceMatrix(d)
// initialization of parameters nump, nc, gen, mp
3 InitializeParameters ()
//Generate initial populations using Clustering Mechanisms: k-medoids, k-means, and fuzzy c-means
4 //BRoutes: Population
BRoutes = CPOP (d, nc, nump)
5 // Update the entire population by its fitness value
BRoutes=UpdateEntirePop(BRoutes)
6 // The start of generation count
for i = 1 **to** gen
// First improvement based Genetic Algorithm
// NRoutes: new generation
7 NRoutes=GAFIB (DM, BRoutes, mp) obtained from Algorithm 5
// update the population by taking the best route
// as a first route of next generation (elitism)
8 BRoutes=UpdateElitismPop (NRoutes)

```
9           end for
10          R_best = BRoutes(1, :)
11          F_best = Routelength(R_best)
12      end
```

4.1. Initial Population Generation

Clustering is a widely used machine-learning technique that groups similar items based on their characteristics. In this work, we employ three clustering methods- K-medoids, K-means, and fuzzy c-means- to generate initial populations or valid TSP routes. In each case, the population creation follows the same process: first, the data is divided into clusters, and then these clusters are arranged side by side to form an individual population. To generate the required number of populations, one of these three methods is applied multiple times. The clustering methods tested are briefly described in the following subsections:

A. Soft Clustering: Fuzzy C-Means

The fuzzy c-means (FCM) algorithm is a widely used clustering method for grouping data sets. In this soft clustering approach, each data point is assigned to multiple clusters with a probability score indicating its likelihood of belonging to each cluster. The FCM clustering generally outperforms k-means clustering in cases where data sets overlap. For TSP data, clustering organizes cities into clusters based on proximity [42]. In our case, the FCM method is used to generate initial populations. First, TSP data are clustered using FCM. Clusters are then formed by selecting the cities with the highest likelihood of belonging to each cluster. These clusters are arranged sequentially to create a complete TSP path. This process is repeated until the desired number of populations is generated. The pseudocode for generating initial populations using the FCM function is provided in Algorithm 2, while the pseudocode for the FCM function itself is shown in Algorithm 3.

In Algorithm 2, step 4 calls the fuzzy C-means clustering function, referred to as "FuzzyC," which returns a fuzzy matrix (from Algorithm 3) that indicates the probability of each data point's association with a given cluster. The maximum likelihood for each data point is determined in step 6 of Algorithm 2. Steps 4 through 8 outline the formation of a single population, while steps 3 through 9 detail the creation of the full set of populations. Step 10 removes duplicate routes. The operators "find," "max," "cell2mat," and "unique" are built-in functions in MATLAB.

Algorithm 2: Pseudocode for generating initial population through fuzzy c-means function (FuzzyC)

```
Input: Data and Parameters
       d: Data, nump: number of populations, nc: number of clusters, m: Fuzzy Parameter, r: number of iterations,
       ep: stop parameter
Output: Initial Population
       BRoutes: Initial population
1      begin
2              InitializeParameters ()
               //Iteration of Population
3              for j= 1 to nump
                  //U: Fuzzy matrix
4                 U=FuzzyC(d,nc,m,r,ep) obtained from Algorithm 3
5                 for i=1 to nc
                        //put every cluster side by side in a cell array
6                       route{j,i} = find(U(i,:) == max(U))
7                 end for
                  // conversion of cell array to matrix array
8                 Routes (j, :)=cell2mat(route)
9              end for
               // keep unique routes only
10             BRoutes=unique (Routes, 'rows')

11     end
```

B. Hard Clustering: K-Means and K-Medoids

In the hard clustering techniques, the generation of initial populations differs from the fuzzy C-means clustering approach described in the previous section. However, the procedure for constructing the initial population is the same for both the k-means and k-medoids clustering strategies. Actually, MATLAB provides two built-in clustering operators for these algorithms: idx=kmeans (data, nc) for k-means clustering and idx=kmedoids (data, NC) for k-medoids clustering. The kmeans operator is used for k-means clustering, while kmedoids is used for the k-medoids approach. These operators produce a column matrix containing the cluster indices for each data point. A matrix with dimensions

data_$n \times nc$ is then created from the result of the chosen operator. If a data point belongs to a specific cluster, the corresponding entry in that row will be 1; otherwise, it will be 0. Once the clustered matrix is generated, the remaining steps for creating the initial population follow steps 6 to 11 in Algorithm 4. Algorithm 4 presents the pseudocode for generating the initial population using the hard clustering approaches, k-means and k-medoids.

Algorithm 3: Pseudocode of fuzzy c-means function (FuzzyC) for the execution of Algorithm 2

```
Input: Data and parameter values: d: Data, nc: Number of clusters, m: Fuzzy Parameter, r: number of iterations, ep: stop parameter
Output: obj_fcn: Objective function, U: Fuzzy Matrix, center: Center
1     begin
2         InitializeParameters ()
          // Initialization Fuzzy matrix
          //data_n: Size of Data
3         U=rand(nc, data_n)
4         col_sum=sum(u)
5         U=U/col_sum(ones(nc,1),:)
          //initial objective function
6         obj_fcn=zeros(r,1)
          //fuzzy iteration begins
7         for i= 1 to r
8             mf=U^m
9             center=(mf*data)./(sum(mf,2)*ones(1,size(data,2)));
10            out= zeros(size(center, 1), size(data, 1))
              //iteration for objective function
11            for k= 1 to size (center, 1)
12                out(k, :)=sqrt(sum(((data-ones(size(data, 1), 1) *center(k, :)).^2), 2))
13            end for
14            dist=out
15            obj_fcn(l) = sum(sum((dist.^2).*mf));
16            tmp = dist.^(-2/(m-1));
17            U_new = tmp./(ones(nc,1)*sum(tmp));
18            if i>1
19                if abs(obj_fcn(i) - obj_fcn(i-1)) < ep
20                    break
21                end if
22            end if
23        U=U_new
24        end for
25    end
```

4.2. Traditional Genetic Algorithm

The Genetic algorithm (GA) is a common approach for solving the Traveling Salesman Problem (TSP). Based on Charles Darwin's theory of natural evolution, this algorithm simulates processes similar to natural selection. Each individual, or "child," inherits a chromosome from its "parents," created through crossover and mutation. Although initially not designed for combinatorial optimization problems, the GA was developed by John Holland and his students at the University of Michigan during the 1960s and 1970s. The GA belongs to a class of optimization techniques inspired by natural selection and genetic principles. This algorithm operates on a population of potential solutions, treating each as an individual within that population. Over successive generations, these individuals undergo selection, crossover, and mutation to evolve toward improved solutions, making the GA particularly well-suited to complex optimization problems like the TSP problem. The traditional GA includes three main stages: selection, crossover, and mutation. The initial population is generated randomly. Then, in the selection phase, individuals are chosen as parents. Crossover operations between selected parents produce offspring, which then undergo mutation to create the final "child" solutions. Repeating this process across the population produces a new generation of solutions. This cycle continues until a predetermined maximum generation number is reached. The GA can sometimes converge to a local optimum instead of the global optimum. This occurs when the algorithm finds a suboptimal solution better than its neighbors but not the best possible solution overall. Actually, the GA may not always achieve high accuracy, and the quality of its solutions depends on several parameters, such as the choice of crossover and mutation operators, population size, and convergence criteria.

Algorithm 4: Pseudocode for generating initial populations through k-means and k-medoids Clustering approaches

```
Input: Data and Parameters
       d: Data, nump: number of populations, nc: number of clusters
Output: Initial Population
       BRoutes: Initial population
1      begin
2              InitializeParameters ()
               //Iteration of Population
3              for j= 1 to nump
4                 idx=kmeans (data,nc) or idx=kmedoids (data,nc)
                  //U: Clustered-matrix
5                 U= Create a clustered matrix from idx
6                 for i=1 to nc
                        //put every cluster side by side in a cell array
7                       route{j,i} = find(U(i,:) == max(U))
8                 end for
                  // conversion of cell array to matrix array
9                 Routes (j, :)=cell2mat(route)
10             end for
               // keep unique routes only
11             BRoutes=unique (Routes, 'rows')
12     end
```

4.3. Enhanced Genetic Algorithm

In the second stage, an Improved Genetic Algorithm (IGA) is used to refine and develop the initial valid TSP routes obtained from the first stage. The traditional genetic algorithm has been improved in several key areas. For the selection phase, a customized family-father selection technique, more particularly, an elitism-based roulette wheel selection technique, is implemented. The crossover phase incorporates order crossover, heuristic crossover, and a newly developed position-based heuristic crossover. Additionally, a first improvement-based (FIB) local search optimizer is used in the mutation phase. These mechanisms, along with their corresponding pseudocode, are discussed in the following subsections. Moreover, the architecture of the traditional Genetic Algorithm and proposed clustering-based Improved Genetic Algorithm is displayed in Fig. 3.

A. Selection Process

The selection mechanism used in this work is a customized version of the family father selection described by Chen et al. [40]. Specifically, it is an elitism-based roulette wheel selection technique. In this mechanism, the best route among the population is designated as the father, while all remaining routes serve as the mothers. The father's chromosome undergoes crossover with each mother's chromosome, producing a group of offspring. Due to elitism, only the father survives to the next generation, ensuring that the best solution is preserved. The main advantage of elitism is that it guarantees the survival of the best chromosome, preventing it from being lost in subsequent generations. The population is updated by placing the best route of the new generation as the leading route. If any offspring exhibits better fitness than the father, the best offspring replaces the father in the next generation. This process repeats for each new generation, with the best route consistently becoming the father for the next round of reproduction, continuing until the maximum generation limit is reached. Essentially, the route with the lowest cost is selected as the father for the next generation. This study claims that the whole population will be replaced by the child population save the best one of the previous generations, and that one can be taken over if there are any better children generated by the following generation. The selection strategy aligns well with the principle of "survival of the fittest." The implementation of this selection mechanism is represented in Algorithm 1 under the function UpdateElitismPop.

B. Crossover Mechanism

Crossover is a unique feature of Genetic Algorithms (GA) that distinguishes them from other algorithms. In crossover, two selected parent solutions are recombined to produce a new child solution. The main types of crossovers are path representation crossover and adjacency representation crossover. Within path representation, there are two key subtypes: absolute position preservation and relative order preservation. Examples of absolute position preservation include partially mapped crossover and cycle crossover, while examples of relative order preservation include modified crossover, ordered crossover, order-based crossover, and position-based crossover. For adjacency representation, common types include alternate edge crossover, edge recombination crossover, and heuristic crossover [36]. In this paper, we employ three types of crossover mechanisms: Ordered Crossover, Heuristic Crossover, and Position-based Heuristic Crossover. A brief description of each mechanism is provided below:

Ordered Crossover: Order Crossover involves randomly selecting two cut points in the first parent and inserting this segment into the second parent. For Order Crossover to be effective, a crossover rate between 60% and 80% is recommended. Lower crossover rates risk losing valuable genes, while overly high rates may not significantly impact the next generation. Thus, the suggested rate is optimal for producing favorable results [36]. In this study, we begin by selecting two points on the father chromosome and removing the genes between these points from the mother chromosome. Then, we insert the segment from the father chromosome into the same positions on the mother chromosome. Fig.1(a) illustrates this process. For example, genes '2 10 3' are cut from Parent1, removed from Parent2, and then placed in the same positions in Parent2 as they were in Parent1. The remaining positions in Parent2 are then filled sequentially with the corresponding gene sequence.

Heuristic Crossover: The heuristic crossover, introduced by Grefenstette et al. [43], involves the following steps [36]:

Step 1: Start with a random city from either Parent1 or Parent2.

Step 2: Compare the edges leaving the current city in both parents and select the shorter edge.

Step 3: If the shorter edge creates a cycle in the partial tour, extend the tour with a random edge that does not create a cycle.

Step 4: Repeat Steps 2 and 3 until the tour includes all cities.

An alternate approach to Step 3 is

Step 3': If the shorter parental edge forms a cycle, attempt to use the opposite parental edge. If this also forms a cycle, then extend the tour with random edges that do not create a cycle.

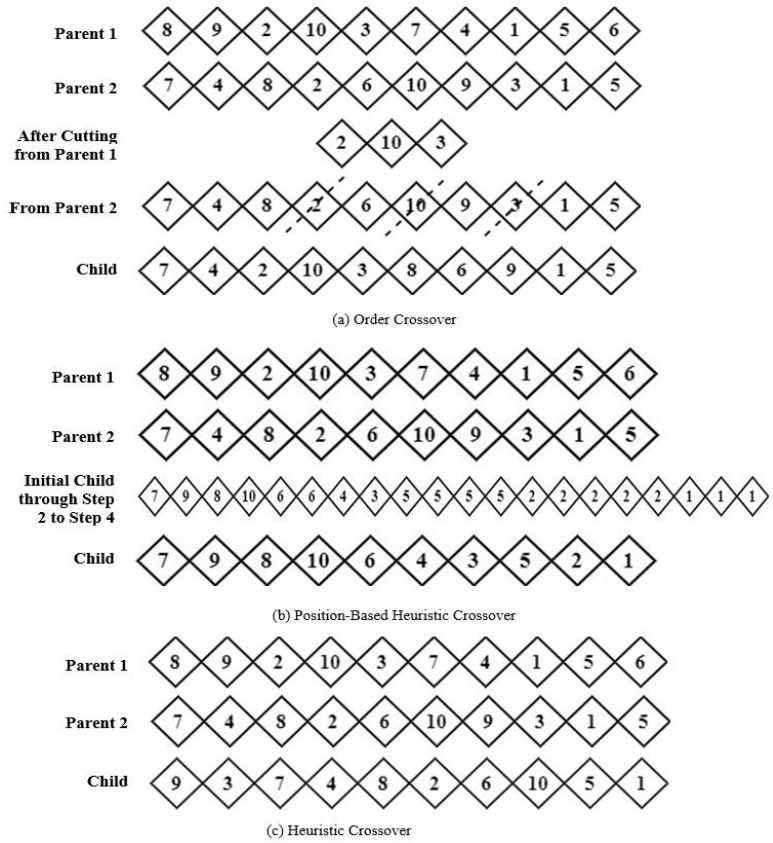


Fig.1. Illustration of different crossover mechanisms (a) Order crossover;(b) Position-based heuristic crossover; (c) Heuristic crossover

In this study, we explore an alternative to Step 3, referred to as Step 3'. An example of the heuristic crossover process is shown in Fig.1(c). Suppose Parent1 is represented as 8-9-2-10-3-7-4-1-5-6, and Parent2 as 7-4-8-2-6-10-9-3-1-5, as depicted in Fig.1(c). Following Step 1, we select a random starting city, which is 9. From city 9, we compare the edges leading to cities 2 and 3. Let city 3 be closer than city 2, so it is chosen as the next city. Proceeding to Step 2, we consider the next available cities: 7, 4, and 8. For city 8, the paths lead to city 9 in Parent1 and city 2 in Parent2. Since city 9 is already included in the chromosome, we select city 2 as the next city, following Step 3'. Continuing this process, we arrive at the next cities, 6 and 10. Since city 10 has the required edge but already exists in the chromosome, we select city 5. By iterating through these steps, we include all the cities in the child chromosome, resulting in the sequence 9-3-7-4-8-2-6-10-5-1.

Position-Based Heuristic Crossover: This crossover method uses city positions to guide selection. The proposed heuristic crossover can be described step by step as follows:

Step 1: Put the entire Parent1 chromosome side by side; for this, the initial size of the Parent1 will be double. Do the same for the Parent2 chromosome. The starting node will be the node that is situated in the first position from the Parent 2 chromosome.

Step 2: Compare the next positions in both parents (excluding the current node's position), and choose the closer node based on position.

Step 3: If the shorter parental positional node starts a cycle, try the opposite parental node. If it also introduces a cycle, then skip that round and then the searching should be extended using step 2 to find nodes that do not introduce a cycle. Because of skipping round the current node appears more than once and step 1 helps to extend searching.

Step 4: Repeat Steps 2 to 4 until all nodes are included in the sequence.

Step 5: Finally, remove any duplicate nodes from the tour.

An example of this crossover mechanism is shown in Fig. 1(b). In this example, Parent1 is 8-9-2-10-3-7-4-1-5-6, and Parent2 is 7-4-8-2-6-10-9-3-1-5. Following Step 1, we start with the first node from Parent2, which is 7. In Step 2, we select the closest city between the second positions of both parents. Here, the options are 9 from Parent1 and 4 from Parent2, considering 9 is closer to 7 than 4, we choose 9 as the next city. Repeating Step 2, we continue with the next closest cities, adding 8, 10, and 6. At this point, selecting the next city would create a cycle, as both 7 from Parent1 and 10 from Parent2 are already in the chromosome. To avoid this, we revisit city 6 and proceed to the next level. This leads to cities 4, 3, and 5 being added in sequence. After repeating Step 2 through Step 4, an initial child sequence of 20 nodes is created, even though the parents each have only 10 nodes. In this sequence, cities 6, 5, 2, and 1 appear multiple times—city 6 appears twice, city 5 appears four times, city 2 five times, and city 1 three times. After removing these duplicates, the final child sequence is 7-9-8-10-6-4-3-5-2-1.

C. Mutation Mechanism

A mutation is a small, random adjustment applied in a Genetic Algorithm (GA) to introduce diversity into the population of solutions, helping the algorithm explore the solution space more effectively. In classical GAs, mutation typically involves an unsupervised, random change in the chromosome, applied at a low probability compared to selection and crossover. If the mutation probability is too high, the GA may struggle to converge, while a very low probability may cause the GA to get stuck near-optimal solutions. The probability of mutation determines the proportion of individuals in the population that will undergo mutation [41]. In this method, the mutation probability is set at 0.3, and a supervised local search approach is introduced to refine the mutation process. This approach uses two well-known local search operators—INVERSION and SWAP—combined in a supervised way, named the First Improvement-Based Local Searching Optimizer (FIB).

The INVERSION operator reverses the segment of the tour between positions *i* and *j*. The SWAP operator, on the other hand, simply interchanges two positions. The city at the i^{th} position is moved to the j^{th} position, and the city at the j^{th} position is moved to the i^{th} position. An example of the INVERSION and SWAP operators is shown in Fig. 2(a). The initial path is 7-9-8-10-6-4-3-5-2-1. Suppose the cities 8 and 5 are selected from this path. After applying the INVERSION operator, the path changes to 7-9-5-3-4-6-10-8-2-1. If the SWAP operator is applied instead, the path becomes 7-9-5-10-6-4-3-8-2-1. In general, these two operators are applied randomly to the tour, producing different results each time.

In the proposed method, there is no random search or random swapping of cities. After each inversion, the tour is evaluated: if a better solution is found, the tour is updated; otherwise, it remains unchanged. The SWAP operation begins only after the entire INVERSION process is completed, following the same evaluation criteria. Specifically, for the inversion process, the first two cities are inverted, and then the fitness of the population is checked. If this change improves the solution, the population is updated; if not, no update is made. Next, the inversion is applied from the first city to the third city, and the fitness is evaluated again to decide whether to update. This process continues sequentially from the first city to the $(n-1)^{th}$ city. For the second city, the search starts from the second city to the third, then the second to the fourth, and so on, until it reaches the $(n-1)^{th}$ city. This approach is repeated similarly for the third city, fourth city, and so on, inverting one city at a time until the entire path from the starting city to the n^{th} city has been explored. A similar procedure is applied for the SWAP operator. First, the 1st and 2nd cities swap positions, and the fitness of the tour is checked for improvement. Then, the 1st city is swapped with the 3rd city, the 1st with the 4th, and so on. This process repeats until all possible city pairs have been evaluated. This approach yields consistent results on the same route each time.

This method is called "supervised" because, after each iteration, the specific change in position and its impact on fitness can be evaluated. Here, both operators (INVERSION and SWAP) are applied in a structured, supervised way, referred to as the First Improvement-Based (FIB) local search. In Fig. 2(b), the initial route is 7-9-8-10-6-4-3-5-2-1. After applying the supervised INVERSION, the route changes to 1-2-6-10-8-3-9-7-4-5. Next, the supervised SWAP is applied to the output of the supervised INVERSION, resulting in the final route: 1-2-6-10-8-7-9-3-4-5. Algorithm 6 presents the implementation of the supervised mutation operations in the Genetic algorithm.

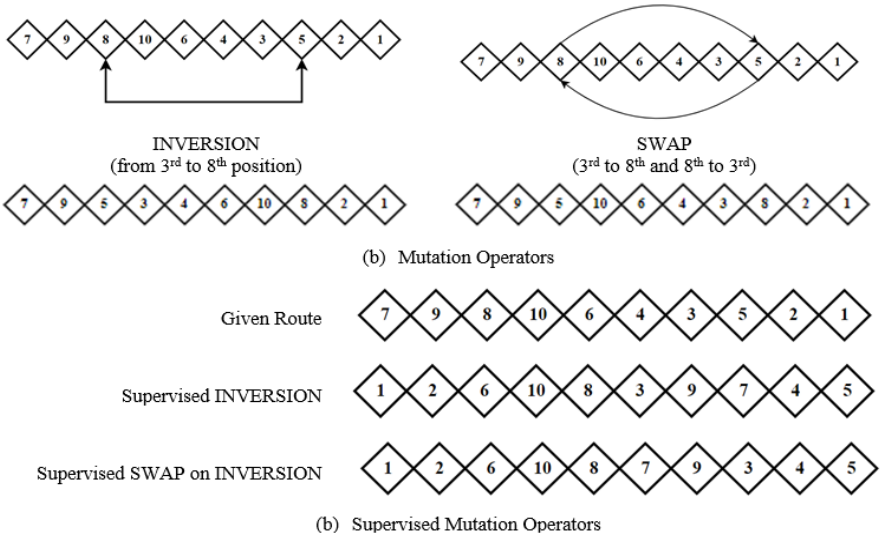


Fig.2. Visual presentation of the mechanism of (a) Mutation operators; and (b) Supervised mutation operators

Algorithm 5: Pseudocode for generating a new generation of Genetic Algorithm Through Supervised Mutation Operators

```
Input: Distance Matrix, Population, Mutation Probability
        DM: distance matrix, BRoutes: population, mp: mutation probability
Output: New Generation
        BRoutes: new generation
1       begins
2               InitializeParameter ()
3               n=size (BRoutes)
                //elitism
4               BRoutes (1,:)=BRoutes (1,:)
                // Parents, P1, P2
5               P1= BRoutes(1,:)
                //generating new population
6               for i= 2 to n
7                       P2= BRoutes(i,:)
8                       child=ApplyCrossoverOperator(P1, P2)
9                       if rand<mp
10                              child=FIB(DM, child) obtained from Algorithm 6
11                      end if
12                      BRoutes (i, :)=child
13              end for
14      End
```

Algorithm 6: Pseudocode for the implementation of the supervised mutation operations

```
Input: Distance matrix, Route
        DM: Distance Matrix, Route: Route
Output: New best route after applying supervised mutation operations
1       begin
            // INVERSION
            // n is the size of Route, fit: initial route fitness
2           fit=Fitness (Route)
3           for i = 1 to n
4               for j = 2 to n
5               Route1=Route
6                   if i < j
7                       Route1=inversion (Route1, i, j)
8                       fit1=Fitness (Route1)
9                       if fit1 < fit       // check update
```

```
10                          Route=Route1;
11                          fit=fit1;
12                      end if
13                  end if
14              end for
15          end for
            // SWAP
16          for i = 1 to n
17              for j = 2 to n
18              Route1=Route
19                  if i < j
20                      Route1=swap (Route1, i, j)
21                      fit1=Fitness (Route1)
22                      if fit1 < fit          // check update
23                          Route=Route1;
24                          fit=fit1;
25                      end if
26                  end if
27              end for
28          end for
29      end
```

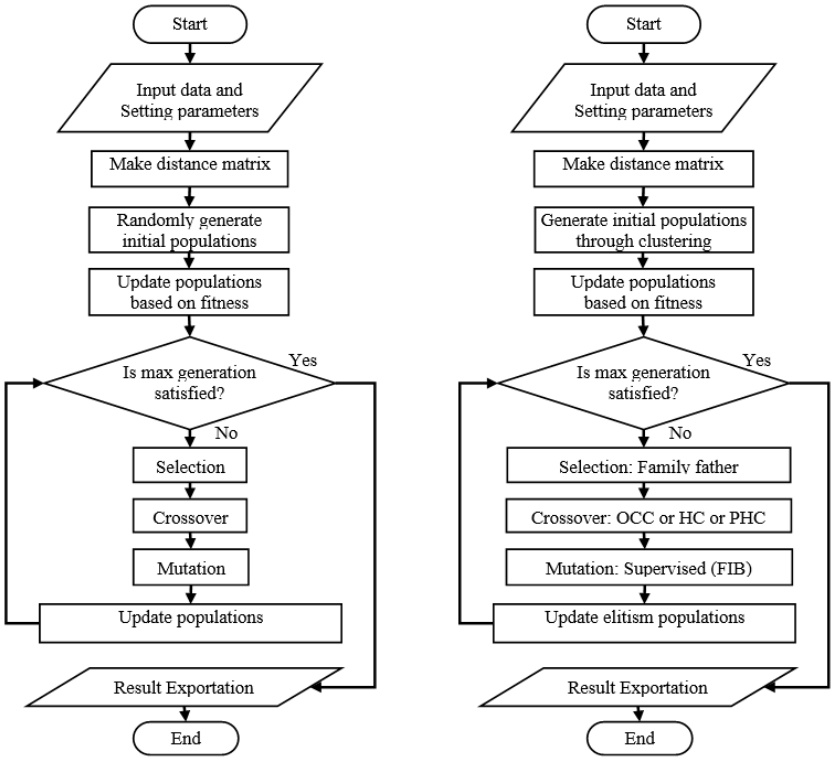


Fig.3. Flowchart of the traditional genetic algorithm (left) and proposed clustering-based improved genetic algorithm (right)

5. Simulation Results and Discussion

In this section, the proposed algorithm is evaluated through simulations on benchmark TSP datasets. Specifically, we used 47 benchmark TSP datasets sourced from publicly available repositories [44]. The datasets vary in size, ranging from small-scale (29 cities) to large-scale (1400 cities) instances, allowing for a comprehensive assessment across different levels of complexity. The numerical value in each dataset's name indicates the number of cities involved. Each approach—fuzzy c-means with Improved Genetic Algorithm (FCM+IGA), k-means with Improved Genetic Algorithm (KM+IGA), and k-medoids with Improved Genetic Algorithm (KMD+IGA)—was tested independently on all 47 benchmark TSP datasets. Simulation results, discussion, and comparisons are presented in the following subsections.

5.1. Simulation Environment

The simulations were conducted on MATLAB 2018a environment. All simulations were run on a system equipped with an AMD Ryzen 5 5600G (3.90 GHz) processor, 16 GB of RAM, and a standard desktop PC with Windows 11 Pro operating system. This configuration ensures consistency in computational performance and comparability across different runs.

5.2. Evaluation Metrics

The algorithm's performance is evaluated based on solution quality (total path distance) and convergence rate. Solution quality reflects the optimality of the path found, while convergence rate indicates how quickly the algorithm approaches the best solution. Solution quality is measured by the deviation of the obtained solution from BKS, denoted as Ebest and defined in Eq. (6), and by the success rate (SR), as defined in Eq. (7). Actually, BKS represents the best-known solution of each dataset reported by the data library [44].

$$\text{Ebest}(\%) = \frac{\text{Obtained Solution - BKS}}{\text{BKS}} \times 100\% \tag{6}$$

$$\text{SR}(\%) = \frac{\text{Favorable Cases}}{\text{Total Cases}} \times 100\% \tag{7}$$

5.3. Parameter Settings

This method uses several parameters: the number of generations, the crossover rate (applicable only for order crossover), and the mutation probability. The main variations in this algorithm are the population size and the number of generations. Table 1 presents the parameters and their respective values.

Table 1. Required parameters and their respective values

Parameters		**Values**
Name	**Symbol**	
Number of Populations	(nump)	100, 500
Number of Clusters	(nc)	10
Number of Generations	(gen)	50, 100, 200, 300, 500
Order Crossover Rate	(CR)	.7
Mutation Probability	(mp)	.3

5.4. Performance Analysis

In this section, we examine the performance of the proposed solution pathways: fuzzy c-means with Improved Genetic Algorithm (FCM+IGA), k-means with Improved Genetic Algorithm (KM+IGA), and k-medoids with Improved Genetic Algorithm (KMD+IGA), to identify the best approach among them. Table 2 displays the simulation results from the proposed two-stage solution approach, which utilizes a clustering-based enhanced genetic algorithm. The proposed two-stage algorithm is tested on 47 benchmark TSP datasets. In the first stage, a set of clustering algorithms is applied to generate feasible routes, using either hard clustering methods, such as k-medoids and k-means, or a soft clustering method, like fuzzy c-means. The second stage incorporates an enhanced genetic algorithm with a family-father selection procedure. During crossover, three different operators are tested, and a supervised local search optimizer is used in the mutation phase. Table 2 provides the best results from all experiments with varied parameter values and crossovers of the different clustering methods.

In Table 2, "BKS" represents the best-known solution for each dataset, as reported by the data library [44]. "KMD+IGA" shows the results when using the k-medoids clustering method in the first step of the enhanced genetic algorithm, while "KM+IGA" and "FCM+IGA" show results from using k-means and fuzzy c-means clustering algorithms, respectively. The table results are obtained by running the experiment 10 times independently. "Best" refers to the best result among the 10 runs, "Avg" is the average result, and "Std" is the standard deviation across all runs. "Ebest" is defined in Eq. (7). The "Average" row at the bottom of the table summarizes the averages of the best, average, standard deviation, and Ebest values across all 47 datasets. Bold text indicates the best performer among the algorithms.

From Table 2, it is evident that each approach—KMD+IGA, KM+IGA, and FCM+IGA—achieves solutions that are equal to or close to the best-known solution (BKS), with small standard deviations and acceptable error rates. Across the 47 datasets, FCM+IGA demonstrates superior performance, producing an Average of Best of 34,448.62, an Average of Avg of 34,648.07, an Average of Std of 52.84, and an Average of Ebest (%) of 0.43. These results outperform KMD+IGA, which achieves 34,475.31, 34,674.82, 158.42, and 0.52, respectively, as well as KM+IGA, which achieves 34,452.73, 34,696.52, 184.32, and 0.45. While each algorithm excels in certain cases, they also fall short or tie in others when compared against one another. As such, it is challenging to identify a definitive best-

performing approach among KMD+IGA, KM+IGA, and FCM+IGA. To address this, we evaluate additional metrics—such as the number of distinct outputs and success rates—to further differentiate their performance.

Table 2. Performance comparison among fuzzy c-means with improved genetic algorithm (FCM+IGA), k-means with improved genetic algorithm (KM+IGA), and k-medoids with improved genetic algorithm (KMD+IGA)

S/N	Dataset	BKS	KMD+IGA				KM+IGA				FCM+IGA			
			Best	Avg.	Std.	Ebest (%)	Best	Avg.	Std.	Ebest (%)	Best	Avg.	Std.	Ebest (%)
1	wi29	27603	**27601.17**	**27601.17**	**0**	-0.01	**27601.17**	**27601.17**	**0**	-0.01	**27601.17**	**27601.17**	**0**	-0.01
2	eil51	426	**428.87**	431.13	2.15	0.67	**428.87**	431.33	2.06	0.67	**428.87**	429.79	1.75	0.67
3	berlin52	7542	**7544.37**	**7544.37**	**0**	0.03	**7544.37**	**7544.37**	**0**	0.03	**7544.36**	**7544.36**	**0**	0.03
4	st70	675	**677.1**	678.94	2.56	0.31	**677.1**	680.53	3.71	0.31	**677.1**	680.92	4.08	0.31
5	eil76	538	**544.36**	547.26	2.03	1.18	**544.36**	550	5.79	1.18	**544.36**	548.87	3.82	1.18
6	pr76	108159	**108159.43**	108607.74	280.62	0	**108159.43**	108565.92	286.84	0	**108159.43**	108513.1	365.15	0
7	rat99	1211	**1219.93**	1236.32	13.37	0.74	**1219.24**	1238.9	11.91	0.68	**1219.24**	1224.96	5.47	0.68
8	kroA100	21282	21294.44	21451.56	131.39	0.06	**21285.44**	21351.1	62.29	0.02	**21285.44**	21312.27	34.96	0.02
9	kroB100	22141	22154.15	22405.66	236.43	0.06	**22139.07**	22320.9	102.49	-0.01	**22139.07**	22244.64	253.98	-0.01
10	kroC100	20749	**20750.76**	20877.52	118.69	0.01	**20750.76**	20896.37	102.05	0.01	**20750.76**	20858.2	133.59	0.01
11	kroD100	21294	**21294.29**	21540.01	247.37	0	**21294.29**	21600.17	304.58	0	**21294.29**	21374.13	75.48	0
12	kroE100	22068	**22068.75**	22354.68	188.26	0	**22068.75**	22390.69	187.85	0	**22068.75**	22118.6	66.94	0
13	rd100	7910	**7910.4**	7984.21	53.79	0	**7910.4**	7950.74	55.22	0	**7910.4**	7965.67	47.74	0
14	eil101	629	642.24	648.21	4.45	2.1	**640.21**	652.26	8.46	1.78	**640.21**	647.52	4.82	1.78
15	lin105	14379	**14382.99**	14421.44	42.62	0.03	**14382.99**	14436.57	44.47	0.03	**14382.99**	14408.32	31.27	0.03
16	pr107	44303	**44301.68**	44420.96	109.85	-0.01	**44301.68**	44420.51	109.92	-0.01	**44301.68**	44414.96	116.06	-0.01
17	pr124	59030	**59030.74**	59160.99	192.39	0	**59030.74**	59390.29	268.09	0	**59030.74**	59159.84	131.4	0
18	bier127	118282	**118293.52**	118531.61	231.74	0.01	**118293.52**	118502.64	211.06	0.01	**118293.52**	118544	218.86	0.01
19	ch130	6110	**6110.72**	6165.89	29.79	0	6110.84	6198.85	60.53	0	**6110.72**	6234.97	87.5	0
20	xqf131	564	**566.42**	568.56	3.22	0.43	**566.42**	573.99	4.53	0.43	**566.42**	569.49	2.71	0.43
21	pr136	96772	97090	98194.98	802.02	0.33	**96922.41**	97937.53	611.79	0.16	**96922.41**	98673.61	1185.81	0.16
22	pr144	58537	**58535.22**	58614.09	83.07	-0.01	**58535.22**	58589.3	32.49	-0.01	**58535.22**	58541.93	14.15	-0.01
23	kroA150	26524	26528.56	26835.45	189.02	0.02	**26524.86**	26695.61	98.65	0	**26524.86**	26635.87	83.38	0
24	kroB150	26130	26140.3	26300.4	108.31	0.04	26140.3	26232	82.72	0.04	**26127.71**	26275.24	179.21	-0.01
25	pr152	73682	**73683.64**	73811.2	134.93	0	**73683.64**	73982.33	313.33	0	**73683.64**	73793.76	182.42	0
26	u159	42080	**42075.67**	42152.94	135.72	-0.01	**42075.67**	42109.57	81.36	-0.01	**42075.67**	42185.47	177.23	-0.01
27	rat195	2323	2353.75	2394.2	30.84	1.32	2345.14	2367.38	12.78	0.95	**2342.74**	2360.27	13.84	0.85
28	d198	15780	15829.91	15875.26	51.32	0.32	**15817.74**	15851.37	28.48	0.24	15830.51	15883.42	44.78	0.32
29	kroA200	29368	29438.96	29713.96	187.03	0.24	**29369.41**	29610.54	173.61	0	**29369.41**	29716.55	216.78	0
30	kroB200	29437	29490.66	29865.71	228.84	0.18	**29440.97**	29951.95	325.59	0.01	29476.17	29818.63	269.26	0.13
31	ts225	126643	126811.96	127098.27	202.6	0.13	**126645.93**	127712.85	843.64	0	**126645.93**	127676.9	487.5	0
32	tsp225	3916	3894.11	3968.68	51.3	-0.56	3883.35	3959.17	39.15	-0.83	**3875.05**	3928.25	42.23	-1.05
33	pr226	80369	**80370.26**	80491.25	266.8	0	**80370.26**	80406.21	113.69	0	**80370.26**	80436.46	152.56	0
34	gil262	2378	2398.72	2420.98	17.96	0.87	2394.17	2415.18	17.2	0.68	**2393.89**	2431.81	23.56	0.67
35	pr264	49135	**49135**	49264.71	197.55	0	**49135**	49181.85	98.94	0	**49135**	49200.15	69.18	0
36	a280	2579	**2586.77**	2616.65	21.81	0.3	**2586.77**	2601.12	14.23	0.3	2587.8	2608.8	22.56	0.34
37	pr299	48191	**48303.21**	48730.78	284.92	0.23	48469.97	48895.76	312.69	0.58	48350.4	48815.47	263.37	0.33
38	lin318	42029	**42307.58**	42859.88	318.31	0.66	42400.29	42738.81	199.6	0.88	42471.04	42777.84	202.22	1.05
39	linhp318	41345	42451.41	42814.53	249.81	2.68	**42200.18**	43117.19	498.26	2.07	42344.19	42887.35	303.74	2.42
40	rd400	15281	15536.97	15760.19	117.72	1.68	15481.44	15612.91	98.92	1.31	**15441.91**	15682.86	131.33	1.05
41	fl417	11861	11974.41	12010.45	20.63	0.96	**11941.62**	12023.16	62.85	0.68	11959.58	12040.4	59.63	0.83
42	pr439	107217	107845.28	108553.88	532.49	0.59	107631.99	109886.54	1974.41	0.39	**107533.2**	108120.6	659.3	0.29
43	pcb442	50778	51678.83	52264.85	509.04	1.77	51684.98	51941.83	184.02	1.79	**51653.46**	52143.39	268.27	1.72
44	d493	35002	35681.07	35917.34	147.72	1.94	**35566.54**	35904.98	236.46	1.61	35632.7	35908.95	151.97	1.8
45	u574	36905	37917.79	38411.67	323.47	2.74	37709.56	38045.79	175.9	2.18	**37540.37**	37920.43	167.8	1.72
46	p654	34643	34736.47	34914.65	222.47	0.27	**34732.67**	34788.41	47.29	0.26	34736.02	34793.71	64.47	0.27
47	fl1400	20127	**20566.83**	20681.15	119.13	2.19	20638.49	20879.73	153.19	2.54	20576.54	20805.32	161.6	2.23
Average			34475.31	34674.82	158.416	0.52	34452.73	34696.52	184.32	0.45	**34448.62**	**34648.07**	**152.84**	**0.43**

Table 3. Comparison of performances among FCM+IGA, KM+IGA, and KMD+IGA in terms of the number of datasets based on Table 2

	KMD+IGA	KM+IGA	FCM+IGA
KMD+IGA	3	1	1
KM+IGA	1	6	8
FCM+IGA	1	8	8
All Ties		20	
Total	47		

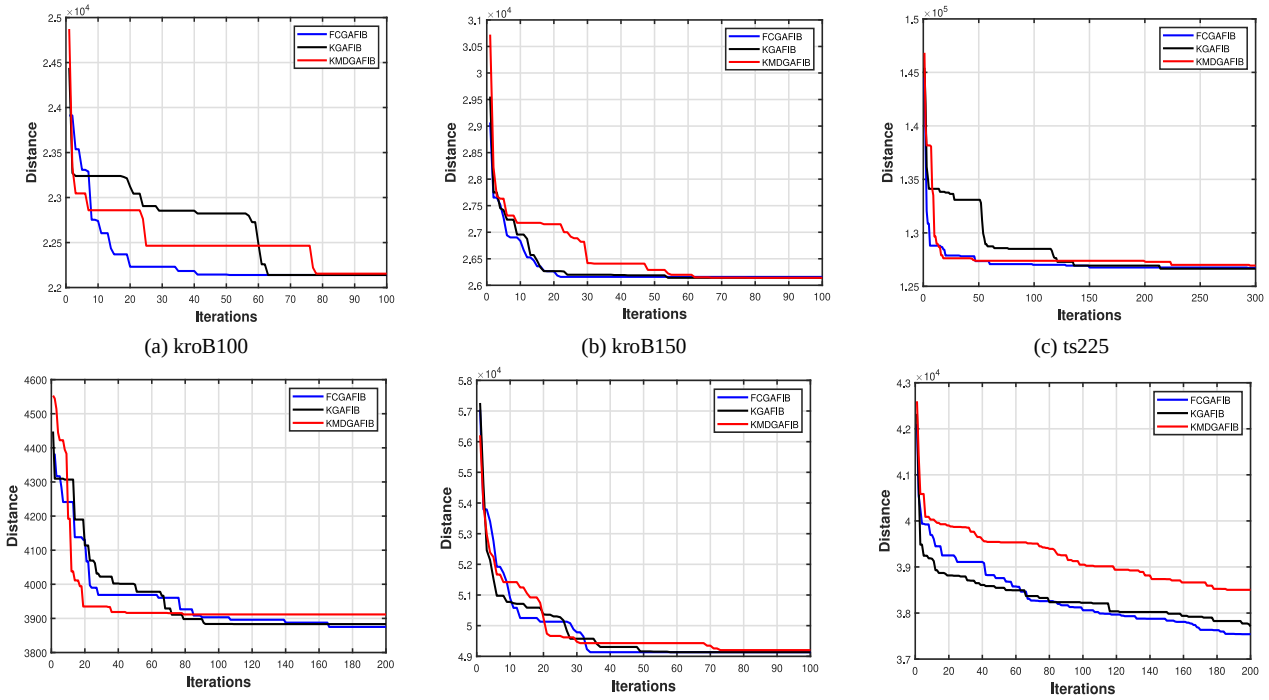
Table 3 illustrates the overall number of distinct outputs produced by three algorithms in terms of the number of datasets. It is important to note that the best outcomes shown are not always the optimal solutions or equivalent to the best-known solutions (BKS). The table exhibits differences in performance among the algorithms. In the table, KMD+IGA and KMD+IGA have three, which suggests that KMD+IGA has the exclusive best outcomes in three datasets compared to others. KMD+IGA and KM+IGA, as well as FCM+IGA and KMD+IGA, share one unique common: the best solution among the performance comparisons. KM+IGA shows exclusive best outcomes in six cases and tying with FCM+IGA in eight instances. On the other hand, FCM+IGA uncommonly wins in 8 datasets compared to the performance of the other two algorithms. Across all datasets, the three algorithms achieve the same best solution in 20 cases.

Table 4. Success rate comparison among KMD+IGA, KM+IGA, and FCM+IGA

Ebest in range	KMD+IGA		KM+IGA		FCM+IGA	
	N	SR (%)	N	SR (%)	N	SR (%)
≤ 0.5	32	68.09	32	68.09	33	70.21
(0.5, 1]	6	12.76	7	14.89	5	10.63
> 1	9	19.14	8	17.02	9	19.14

The success rates shown in Table 4 are based on the Ebest values presented in Table 2. These success rates are categorized into three classes: error rates (Ebest) less than or equal to 0.5%, greater than 0.5% but less than 1%, and greater than 1%. The formula for calculating the success rate (SR) is provided in Eq. (7). In the table, *N* represents the total number of datasets. The table highlights that FCM+IGA achieves a higher success rate compared to the other algorithms when the error rate is less than or equal to 0.5%. Specifically, KMD+IGA and KM+IGA achieve 32 cases with an Ebest rate below 0.5%, resulting in a success rate of 68.09%. In contrast, FCM+IGA has 33 instances where the error rate is less than 0.5%, yielding a success rate of 70.21% under the same conditions.

By comparing the Average of Best, Average of Avg., Average of Std., and Average of Ebest across all datasets in Table 2, the highest number of unique solutions in Table 2, and the success rate in Table 4, it can be concluded that the performance of FCM+IGA is superior to the other two approaches. Therefore, we compare the results obtained by FCM+IGA with other clustering-based algorithms and state-of-the-art algorithms in the next subsection. For visualization purposes, Fig. 4 illustrates the comparisons among the convergence curves of KMD+IGA, KM+IGA, and FCM+IGA for some datasets. As shown in Fig. 4, FCM+IGA demonstrates early convergence with the best solution compared to the other two algorithms.



(a) kroB100 (b) kroB150 (c) ts225

(d) tsp225 (e) pr264 (f) u574

Fig.4. Comparisons among the convergence curves of KMD+IGA, KM+IGA, and FCM+IGA for some datasets (a) kroB100; (b) kroB150; (c) ts225; (d) tsp225; (e) pr264; (f) u574

5.5. State-of-the-art Comparison

In this section, we compare the performance of our proposed best-performing approach, FCM+IGA, with other state-of-the-art algorithms for solving the TSP. First, we evaluate its performance against clustering-based state-of-the-art algorithms, followed by a comparison with other state-of-the-art methods. Specifically, we compare our simulation results with those reported in the reference articles. Table 5 presents a performance comparison of FCM+IGA with other clustering-based methods, while Table 6 shows a comparison between FCM+IGA and other two-phase or hybrid state-of-the-art approaches for solving the TSP. In the tables, the symbol "-" indicates that the reference article did not provide results for the corresponding datasets, while bold values highlight the best-performing algorithms.

Table 5 presents a comparison of the results of FCM+IGA with other clustering-based methods. Specifically, the clustering-based firefly algorithm proposed by Jaradat et al. [32], the affinity programming-based clustering with genetic algorithm (APGA) developed by El-Samak et al. [34], and the clustering-based meta-heuristic local search method (CTSPMRSILS) proposed by Fuentes et al. [31] were considered for comparison with our proposed FCM+IGA. As shown in Table 5, FCM+IGA consistently outperforms all the compared algorithms across all datasets. Compared to the clustering-based firefly algorithm in four datasets, FCM+IGA demonstrates superior performance in every instance. Similarly, when compared to APGA, FCM+IGA performs better in all three scenarios. Additionally, FCM+IGA emerges as the best performer in all eight instances when compared with CTSPMRSILS.

Table 5. Performance comparison of FCM+IGA with other clustering-based algorithms

S/N	Datasets	BKS	Clustering Firefly [32]	APGA [34]	CTSPMRSILS [31]	FCM+IGA (Proposed)
1	eil51	426	485.31	437	442.78	**428.87**
2	berlin52	7542	-	7974	-	**7544.36**
3	st70	675	-	702	-	**677.1**
4	eil76	538	613.02	-	568.94	**544.36**
5	pr76	108159	112386.74	-	111083.63	**108159.43**
6	kroB100	22141	-	-	33811.15	**22139.07**
7	kroE100	22068	-	-	23247.92	**22068.75**
8	gil262	2378	-	-	2785.6	**2393.89**
9	lin318	42029	45226	-	49407.42	**42471.04**
10	pcb442	34643	-	-	61453.07	**34736.02**

Table 6. Performance comparison of FCM+IGA with other state-of-the-art algorithms

S/N	Datasets	BKS	ASA-GS [20]	HGA [41]	RNNSA [28]	FCM+IGA (proposed)
1	eil51	426	428.872	428.87	428.87	428.87
2	berlin52	7542	7544.37	7544.37	7544.37	**7544.36**
3	st70	675	677.11	677.11	677.11	**677.1**
4	eil76	538	544.369	544.37	544.37	**544.36**
5	pr76	108159	**108159**	108159.42	-	108159.43
6	rat99	1211	1219.24	1219.24	1219.24	1219.24
7	kroA100	21282	21285.4	21285.44	21285.44	21285.44
8	kroB100	22141	22139.1	-	-	**22139.07**
9	kroC100	20749	20750.8	20750.76	-	20750.76
10	kroD100	21294	21294.3	**21294.29**	-	**21294.29**
11	kroE100	22068	22106.3	-	22119.92	**22068.75**
12	rd100	7910	7910.4	**7910.39**	-	**7910.39**
13	eil101	629	640.21	640.21	644.92	640.21
14	lin105	14379	14383	**14382.99**	-	**14382.99**
15	pr107	44303	44301.7	**44301.68**	-	**44301.68**
16	pr124	59030	**59030.7**	59030.73	-	59030.74
17	bier127	118282	118294	-	119331.15	**118293.52**
18	ch130	6110	6110.72	6110.72	6171.87	6110.72
19	pr136	96772	96966.3	**96785.852**	96922.41	96922.4
20	pr144	58537	58535.2	58535.22	-	58535.22
21	kroA150	26524	26524.9	**26524.86**	26821.83	**26524.86**
22	kroB150	26130	26140.7	**26127.35**	26327.09	26127.71
23	pr152	73682	**73683.6**	73683.63	73873.09	73683.64
24	u159	42080	42392.9	-	42162.75	**42075.67**
25	rat195	2323	2345.22	2347.24	2348.7	**2342.09**
26	d198	15780	15830.6	15896.95	15852.33	**15830.51**
27	kroA200	29368	29411.5	**29369.4**	29541.83	**29369.4**
28	kroB200	29437	29504.2	**29450.5**	29825.16	29476.17
29	ts225	126643	126646	128141.92	-	**126645.93**
30	tsp225	3916	-	3878.66	-	**3875.05**
31	pr226	80369	80542.1	80436.04	-	**80370.26**
32	pr264	49135	**49135**	49151.22	49197.32	**43135**
33	lin318	42029	**42306.7**	42624.34	42832.5	42471.04
34	linhp318	41345	-	43050.69	-	**42344.19**
35	pr439	107217	110020	110171.34	-	**107533.2**
36	fl1400	20127	20647.4	-	-	**20576.54**

The performance of our proposed FCM+IGA is also compared with ASA-GS [20], HGA [41], and RNNSA [28], as shown in Table 6. ASA-GS and HGA are hybrid strategies for solving the TSP, while RNNSA is a two-phase method. According to Table 6, FCM+IGA outperforms ASA-GS in 25 instances, matches its performance in 3 instances, and is outperformed by ASA-GS in 6 instances. When compared to HGA, FCM+IGA demonstrates superior performance in 12 instances, ties in 13 instances, and is outperformed by HGA in 6 instances. Finally, in the comparison with RNNSA, FCM+IGA achieves better results in 18 datasets, ties in 3, and is never outperformed by RNNSA. To further evaluate the statistical differences in performance between our proposed best-performing approach, FCM+IGA, with other state-of-the-art methods, a rigorous statistical analysis is provided in the next subsection.

5.6. Statistical Analysis

Statistical analyses, including normality tests and non-parametric tests, are conducted to evaluate the effectiveness of our proposed FCM+IGA. These tests are based on the results presented in Table 2 and Table 6. Table 5 is excluded from the statistical analysis due to insufficient data from other compared algorithms to perform these tests. To assess the normality of the results, the Shapiro-Wilk test and the Anderson-Darling test are employed. For the non-parametric analysis, the Wilcoxon signed-rank test is used to determine statistically significant differences between our proposed FCM+IGA and the other approaches. The significance level, α, is set at 0.05, meaning that if the p-value is less than or equal to this threshold, the null hypothesis will be rejected.

A. Normality Test

As mentioned earlier, the Shapiro-Wilk test and the Anderson-Darling test are used to assess the normality of the resulting data. For these tests, the null hypothesis *(H_0)* assumes that the outcomes are normally distributed, while the alternative hypothesis *(H_a)* assumes that the outcomes are not normally distributed. The results of the normality test based on Table 2 are presented in Table 7. The tests are performed on all three approaches across 47 instances. In every case, the p-value is found to be less than the significance level, indicating that the outcomes of each algorithm are not normally distributed. Similarly, the normality test results based on Table 6 are summarized in Table 8. As shown in Table 8, the p-values for ASAGS [20], HGA [4], RNNSA [28], and our FCM+IGA are all below the significance level. This confirms that the data are not normally distributed, highlighting a significant difference in performance between our FCM+IGA and the compared algorithms.

Table 7. Normality test based on the Table 2

Algorithm	Shapiro-Wilk test			Anderson-Darling test		
	N	**Statistic**	**p-value**	***N***	**Statistic**	**p-value**
KMD+IGA	47	0.859	< 0.0001	47	2.106	< 0.0001
KM+IGA	47	0.859	< 0.0001	47	2.105	< 0.0001
FCM+IGA	47	0.859	< 0.0001	47	2.104	< 0.0001

Table 8. Normality test based on the Table 6

Algorithm	Shapiro-Wilk test			Anderson-Darling test		
	N	**Statistic**	**p-value**	***N***	**Statistic**	**p-value**
ASA-GS	34	0.854	< 0.0001	34	1.758	< 0.0001
HGA	31	0.871	0.002	31	1.425	< 0.0001
RNNSA	21	0.821	0.002	21	1.254	0.002
FCM+IGA	36	0.860	< 0.0001	36	1.855	< 0.0001

B. Non-parametric Tests

The Wilcoxon signed-rank test, a widely used non-parametric test, is conducted to determine whether there is a significant difference between the performance of our proposed FCM+IGA and the compared algorithms. In this test, the null hypothesis *(H_0)* states that there are no significant differences between the performances, while the alternative hypothesis *(H_a)* states that there are significant differences. The significance level, α , is set at 0.05. If the p-value from the test is less than or equal to this threshold, the null hypothesis *(H_0)* will be rejected, indicating significant differences between the performances of the compared algorithms. Conversely, if the p-value is greater than the significance level, the null hypothesis will not be rejected, meaning there are no significant differences between the performances, and the alternative hypothesis *(H_a)* will be rejected.

Table 9. Wilcoxon signed-rank test of FCM+IGA with KMD+IGA and KM+IGA

Comparison		***N***	**Statistic**	**p-value**
FCM+IGA vs KMD+IGA	FCM+IGA wins	21	-3.08	0.002
	KMD+IGA wins	5		
	Ties	21		
FCM+IGA vs KM+IGA	FCM+IGA wins	11	-0.4829	0.629
	KM+IGA wins	8		
	Ties	28		

Table 10. Wilcoxon signed-rank test of FCM+IGA with ASA-GS, HGA, and RNNSA

Comparison		***N***	**Statistic**	**p-value**
FCM+IGA vs ASA-GS	FCM+IGA wins	25	-3.05	0.002
	ASA-GS wins	6		
	Ties	3		
FCM+IGA vs HGA	FCM+IGA wins	12	-1.97	0.049
	HGA wins	6		
	Ties	13		
FCM+IGA vs RNNSA	FCM+IGA wins	18	-3.72	< 0.0014
	RNNSA wins	0		
	Ties	3		

The test results based on Table 2 are presented in Table 9. According to Table 9, when comparing FCM+IGA with KMD+IGA, the p-value indicates a significant difference between their performances. In contrast, the comparison of FCM+IGA with KM+IGA yields a p-value of 0.629, suggesting no statistically significant difference in their performances. However, FCM+IGA outperforms KM+IGA on other metrics, as discussed earlier. The results of the Wilcoxon signed-rank test based on Table 6 are summarized in Table 10. As shown in Table 10, the p-values for the comparisons with ASA-GS, HGA, and RNNSA are 0.002, 0.049, and 0.0014, respectively—all of which are less than the significance level of 0.05. This indicates that the null hypothesis is rejected, confirming significant differences in performance between FCM+IGA and these algorithms.

6. Conclusions

We have established a two-stage algorithm for solving the Traveling Salesman Problem (TSP) through a Genetic Algorithm (GA) combined with the clustering concept. In the first stage, the initial population for the GA is generated using clustering algorithms, specifically fuzzy c-means, k-means, and k-medoids. In the second stage, these initial solutions are iteratively improved through an enhanced GA. Actually, the traditional GA is improved by introducing new mechanisms in the crossover and mutation phases. For the crossover phase, we incorporate order crossover, heuristic crossover, and a new position-based heuristic crossover. In the mutation phase, we apply supervised mutation operators based on a first-improvement strategy. This solution framework results in three distinct pathways: fuzzy c-means with Improved Genetic Algorithm (FCM+IGA), k-means with Improved Genetic Algorithm (KM+IGA), and k-medoids with Improved Genetic Algorithm (KMD+IGA). Simulation results and discussions demonstrate that FCM+IGA performs better than KM+IGA and KMD+IGA. Furthermore, our best-performing approach FCM+IGA outperforms other clustering-based methods and state-of-the-art algorithms in terms of solution quality. While the basic GA is too slow to achieve optimal results, and other metaheuristic algorithms like Simulated Annealing and Ant Colony Optimization require extensive parameter fine-tuning, our FCM+IGA approach stands out. It offers better solutions with fewer parameters, achieves results within a reasonable time frame, and requires significantly fewer generations compared to traditional GAs. In the future, this framework could be extended to address asymmetric TSP, Multiple-TSP, Capacitated Vehicle Routing Problems (CVRP), and other combinatorial optimization problems.

Conflict of Interest

The authors confirm that there is no conflict of interest with the publication of this article.

Acknowledgment

We thank the anonymous reviewers for their valuable comments and suggestions, which helped improve the quality of this paper.

References

[1] N. Deo, “Paths and circuits,” *in Graph Theory with Applications to Engineering and Computer Science*. Prentice-Hall, nc, 1974, pp. 34–35.

[2] Q. T. Luu, “Traveling salesman problem: Exact solutions vs. heuristic vs. approximation algorithms,” *Baeldung on Computer Science*, 2023. [Online]. Available: https:/ /www.baeldung.com /cs/tsp- exact-solutions-vs-heuristic-vs-approximation-algorithms.

[3] J. Clausen, “Branch and bound algorithms-principles and examples,” *Department of Computer Science, University of Copenhagen*, pp. 1–30, 1999.

[4] M. S. Levine, “Finding the right cutting planes for the tsp,” *Journal of Experimental Algorithmics (JEA)*, vol. 5, 6–es, 2000. https://doi.org/10.1145/351827.384248.

[5] D. Karapetyan and G. Gutin, “Lin–kernighan heuristic adaptations for the generalized traveling salesman problem,” *European Journal of Operational Research*, vol. 208, pp. 221–232, 2011. https://doi.org/10.1016/j.ejor.2010.08.011.

[6] A. B. Kahng and S. Reda, “Match twice and stitch: A new tsp tour construction heuristic,” *Operations Research Letters*, vol. 32, pp. 499–509, 2004. https://doi.org/10.1016/j.orl.2004.04.001.

[7] H. Mo and L. Xu, “Biogeography migration algorithm for traveling salesman problem,” *International Journal of Intelligent Computing and Cybernetics*, vol. 4, pp. 311–330, 2011. DOI10.1108/17563781111160002.

[8] X. S. Yang, *Nature-Inspired Metaheuristic Algorithms*. Luniver Press, 2010.

[9] H. F. Amaral, S. Urrutia, and L. M. Hvattum, “Delayed improvement local search,” *Journal of Heuristics*, vol. 27, pp. 923–950, 2021. https://doi.org/10.1007/s10732-021-09479-9.

[10] D. Gupta, “Solving tsp using various meta-heuristic algorithms,” *International Journal of Recent Contributions from Engineering, Science IT (iJES)*, vol. 1, pp. 22–26, 2013. DOI: http://dx.doi.org/10.3991/ijes.v1i2.3233.

[11] V. J. Sudhakar and V. N. Kumar, “A new approach to solve the classical symmetric traveling salesman problem by zero suffix method,” *Int. J. Contemp. Math. Sciences*, vol. 6(23), pp. 1111–1120, 2011.

[12] A. Philip, A. A. Taofiki, and O. Kehinde, "A genetic algorithm for solving travelling salesman problem," *International Journal of Advanced Computer Science and Applications (IJACSA)*, vol. 2(1), pp. 26–29, 2011. [Online]. Available: https://ir.unilag.edu.ng/handle/123456789/5461.

[13] Y. Wang and Z. Han, "Ant colony optimization for traveling salesman problem based on parameters optimization," *Applied Soft Computing*, vol. 107, p. 107 439, 2021. https://doi.org/10.1016/j.asoc.2021.107439.

[14] I. Khan and M. K. Maiti, "A swap sequence based artificial bee colony algorithm for traveling salesman problem," *Swarm and Evolutionary Computation*, vol. 44, pp. 428–438, 2019. https://doi.org/10.1016/j.swevo.2018.05.006.

[15] E. F. Goldbarg, M. C. Goldbarg, and G. Souza, *Particle Swarm Optimization Algorithm for the Traveling Salesman Problem*. IntechOpen, 2008, pp. 75–96.

[16] H. A. Abdulkarim and I. F. Alshammari, "Comparison of algorithms for solving traveling salesman problem," *International Journal of Engineering and Advanced Technology (IJEAT)*, vol. 4(6), pp. 76–79, 2015.

[17] L. Wang, R. Cai, M. Lin, and Y. Zhong, "Enhanced list-based simulated annealing algorithm for large-scale traveling salesman problem," *IEEE Access*, vol. 7, pp. 144 366–144 380, 2019. https://doi.org/10.1109/ACCESS.2019.2945570.

[18] B. Zarei and M. R. Metbodi, "A hybrid method for solving traveling salesman problem," *6th IEEE/ACIS International Conference on Computer and Information Science (ICIS)*, pp. 394–399, 2007. https://doi.org/10.1109/ICIS.2007.24.

[19] M. Rahbari and A. Jahed, "A hybrid simulated annealing algorithm for traveling salesman problem with three neighbor generation structures," *10th International Conference of Iranian Operations Research Society (ICIORS)*, 2017. https://hal.science/hal-01962049v1.

[20] X. Geng, Z. Chen, W. Yang, D. Shi, and K. Zhao, "Solving the traveling salesman problem based on an adaptive simulated annealing algorithm with greedy search," *Applied Soft Computing*, vol. 11(4), pp. 3680–3689, 2011. https://doi.org/10.1016/j.asoc.2011.01.039.

[21] L. Xiong and S. Li, "Solving tsp based on the improved simulated annealing algorithm with sequential access restrictions," *Proceedings of the 2016 6th International Conference on Mechatronics, Computer and Education Informationization (MCEI 2016)*, vol. 130, pp. 610–616, 2016. https://doi.org/10.2991/mcei-16.2016.127.

[22] M. A. Rahman, A. Islam, and L. E. Ali, "Intelligent route construction algorithm for solving traveling salesman problem," *International Journal of Computer Science and Network Security*, vol. 21, pp. 33–40, 2021. https://doi.org/10.22937/IJCSNS.2021.21.4.5.

[23] A. H. Zhou, L. P. Zhu, B. Hu, et al., "Traveling salesman problem algorithm based on simulated annealing and gene-expression programming," *Information*, vol. 10(1), pp. 1–15, 2019. https://doi.org/10.3390/info10010007.

[24] I. Ilhan and G. Gokmen, "A list-based simulated annealing algorithm with crossover operator for the traveling salesman problem," *Nural Computing and Applications*, vol. 34, pp. 7627–7652, 2022. DOI: https://doi.org/10.1007/s00521-021-06883-x.

[25] S. M. Chen and C. Y. Chien, "Solving the traveling salesman problem based on the genetic simulated annealing ant colony system with particle swarm optimization techniques," *Expert System with Applications*, vol. 38(12), pp. 14 439–14 450, 2011. https://doi.org/10.1016/j.eswa.2011.04.163.

[26] J. Y. Potvin, "State-of-the-art survey—the traveling salesman problem: A neural network perspective," *ORSA Journal on Computing*, vol. 5(4), pp. 328–348, 1993. https://doi.org/10.1287/ijoc.5.4.328.

[27] L. Paquete and T. Stutzle, "A two-phase local search for the biobjective traveling salesman problem," *Evolutionary Multi-Criterion Optimization*, vol. 2632, pp. 479–493, 2003. https://doi.org/10.1007/3-540-36970-8_34.

[28] M. A. Rahman and H. Parvez, "Repetative nearest neighbor based simulated annealing search optimization algorithm for traveling salesman problem," *Open Access Library Journal*, vol. 8(6), pp. 1–17, 2021. https://doi.org/10.4236/oalib.1107520.

[29] S. Emek, S. Bora, and A. Ugur, "Optimization of traveling salesman problem using genetic algorithm and fuzzy c-means clustering," *Global Journal on Technology*, vol. 8, pp. 143–150, 2015. [Online]. Available: http://awer-center.org/gjt/.

[30] J. W. Yoon and S. B. Cho, "An efficient genetic algorithm with fuzzy c-means clustering for traveling salesman problem," *2011 IEEE Congress of Evolutionary Computation (CEC)*, pp. 1452–1456, 2011. https://doi.org/10.1109/CEC.2011.5949786.

[31] G. E. A. Fuentes, E. S. H. Gress, J. C. S. T. Mora, and J. M. Marin, "Solution to travelling salesman problem by clusters and a modified multi-restart iterated local search metaheuristic," *PLoS ONE*, vol. 13(8), e0201868, 2018. https://doi.org/10.1371/journal.pone.0201868.

[32] A. Jaradat, B. Matalkeh, and W. Diabat, "Solving traveling salesman problem using firefly algorithm and k-means clustering," *2019 IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT)*, pp. 586–589, 2019. https://doi.org/10.1109/JEEIT.2019.8717463.

[33] Y. Deng, Y. Lui, and D. Zhou, "An improved genetic algorithm with initial population strategy for symmetric tsp," *Mathematical Problems in Engineering*, vol. 2015, p. 12 794, 2015. https://doi.org/10.1155/2015/212794.

[34] A. Elsamak andW. Ashour, "Optimization of traveling salesman problem using affinity propagation clustering and genetic algorithm," *Journal of Artificial Intelligence and Soft Computing Research*, vol. 5(4), pp. 239–245, 2015. https://doi.org/10.1515/jaiscr-2015-0032.

[35] J. Martikainen and S. J. Ovaska, "Using fuzzy evolutionary programming to solve travelling salesman problem," *9th IASTED International Conference on Artificial Intelligence and Soft Computing*, pp. 49–54, 2005.

[36] J. Y. Potvin, "Genetic algorithms for the traveling salesman problem," *Annals of Operations Research*, vol. 63, pp. 339–370, 1996. https://doi.org/10.1007/BF02125403.

[37] P. Larra˜naga, C. M. H. Kuijpers, R. Murga, I. Inza, and S. Dizdarevic, "Genetic algorithms for the travelling salesman problem: A review of representations and operators," *Artificial Intelligence Review*, vol. 13, pp. 129–170, 1999. https://doi.org/10.1023/A:1006529012972.

[38] C. Chudasama, S. M. Shah, and M. Panchal, "Comparison of parents selection methods of genetic algorithm for tsp," *International Conference on Computer Communication and Networks CSI-COMNET*, pp. 85–87, 2011.

[39] N. M. Razali and J. Geraghty, "Genetic algorithm performance with different selection strategies in solving tsp," *Proceedings of the World Congress on Engineering*, vol. II, 2011.

[40] P. Y. Chen, C. H. Chen, and H. Wang, "A neural network: Family competition genetic algorithm and its applications in electromagnetic optimization," *Applied Computational Intelligence and Soft Computing*, vol. 2009(1), pp. 474 125, 2009. https://doi.org/10.1155/2009/474125.
[41] Y. Wang, "The hybrid genetic algorithm with two local optimization strategies for traveling salesman problem," *Computers Industrial Engineering*, vol. 70, pp. 124–133, 2014. https://doi.org/10.1016/j.cie.2014.01.015.
[42] L. X. Wang, "The fuzzy c-means algorithm," *in A Course in Fuzzy System and Control*. Prentice-Hall International, Inc, 1997, pp. 342–353.
[43] J. J. Grefenstette, R. Gopal, B. J. Rosmaita, and D. V. Gucht, "Genetic algorithms for the traveling salesman problem," *Proceedings of the 1st International Conference on Genetic Algorithms, ICGA'85*, pp. 160–168, 1985.
[44] *Tsplib*, Accessed on 2024-11-17. [Online]. Available: http://comopt.ifi.uni- heidelberg.de/software/TSPLIB95.

Authors' Profiles

Dr. Md. Azizur Rahman is currently an Associate Professor at Mathematics Discipline, Khulna University, Khulna, Bangladesh. He earned his Ph.D. degree in Applied Mathematics from Peking University, Beijing, China, and holds a Master's and a Bachelor's degree from Khulna University, Khulna, Bangladesh. His research interest includes Combinatorial Optimization, Transportation Engineering, Evolutionary Algorithms, Machine Learning, Differential Equations, Integral Equations, and Finite Element Analysis.

Kazi Mohammad Nazib is a researcher in applied mathematics, with a strong interest in algorithm development, optimization, and machine learning. He earned his M.Sc. in Applied Mathematics and B.Sc. in Mathematics from Khulna University. Nazib has contributed to several publications in diverse fields, including combinatorics, numerical methods, and sustainable energy modeling. His work reflects a strong commitment to addressing real-world problems through innovative mathematical solutions, reflects his passion for applying theoretical concepts to practical applications.

Md. Rafsan Islam is currently working as a Lecturer in Mathematics at Hamdard University Bangladesh, Gazaria, Munshiganj, Bangladesh. He received his B.Sc. degree in Mathematics and M.Sc. degree in Applied Mathematics from Khulna University, Bangladesh. His research interests are centered on Applied Mathematics, especially on combinatorial optimization and mathematical modeling. He wishes to contribute in this field as well as for humankind.

Dr. Lasker Ershad Ali received the Doctor of Natural Science degree from Peking University in 2018. After working as, Lecturer, Assistant Professor, and Associate Professor, he has been a Professor of Mathematics at Khulna University since 2019. Dr. Lasker Ershad Ali has professional competencies, especially in the area of teaching and research. His research interests include Biometric, Image Processing, Pattern Recognition, Machine Learning as well as Deep Learning, Computer Vision, Statistical Learning, Information Intelligence and Applied Mathematics.

How to cite this paper: Md. Azizur Rahman, Kazi Mohammad Nazib, Md. Rafsan Islam, Lasker Ershad Ali, "Solving Traveling Salesman Problem Through Genetic Algorithm with Clustering", International Journal of Intelligent Systems and Applications(IJISA), Vol.17, No.3, pp.15-33, 2025. DOI:10.5815/ijisa.2025.03.02