JST: Smart Systems and Devices
Volume 35, Issue 1, January, 2025, 020–029

Application of Genetic Algorithms to Solve the Clustered Generalized Traveling Salesman Problem

Ta Anh Son*, Nguyen Thi Hien, Hoang Manh Hieu
Hanoi University of Science and Technology, Ha Noi, Vietnam
**Corresponding author email: son.taanh@hust.edu.vn*

Abstract

This paper explores the application of genetic algorithms (GAs) to solve the Traveling Salesman Problem (TSP) and its variants, specifically the Clustered Generalized Traveling Salesman Problem (CGTSP). In CGTSP, nodes (or cities) are grouped into clusters, and the primary objective is to determine the most efficient route that includes precisely one node from each cluster while minimizing the overall travel distance This particular variation plays a crucial role in practical scenarios, such as logistics, vehicle routing, and network design. In these applications, destinations are naturally grouped, and visiting one representative from each group is often essential or mandatory. CGTSP effectively tackles the complexity and real-world constraints more adeptly than the traditional TSP by integrating the element of clustering. This feature makes it a versatile and applicable model for a wide range of industrial and scientific problems. The study aims to assess the effectiveness of Genetic Algorithms (GAs) in solving these complex optimization problems. The paper provides an overview of the TSP and CGTSP, shows how to rewrite a CGTSP in the form of a TSP and vice versa, discusses the fundamentals of GAs, and presents the application of GAs to the problem variants. In addition, we investigate a new method to generate the initial population in the genetic algorithm and evaluate the proposed algorithm's performance through experimental results. The findings highlight the potential of GAs as powerful tools for solving challenging optimization problems.

Keywords: Genetic Algorithms, Traveling Salesman Problem, Clustered Generalized Traveling Salesman Problem (CGTSP), Combinatorial Optimization.

1. Introduction

Nature has and always continues to spark the fire of boundless imagination for countless generations of humanity, delivering new unique, diverse, and captivating phenomena, and fostering creative and bold ideas. By observing the innate efficiency of organisms in nature, we are inspired to investigate their possible applications, especially in the field of optimization.

Traveling Salesman Problem (TSP) is a classic combinatorial optimization problem that has been extensively studied in the field of mathematics and computer science [1]. The problem involves finding the shortest possible route that a salesman can take to visit a set of cities and return to the starting point. TSP has practical applications in various domains, including transportation planning, logistics, and network routing [2].

The TSP is an NP-hard problem. Due to its NP-hardness, finding an optimal solution for TSP with large instances is computationally challenging. Therefore, researchers have developed approximation algorithms, heuristics, and metaheuristic methods to efficiently solve TSP and search for near-optimal solutions [1]. Many algorithms are extensively utilized to solve the problem: dynamic programming, branch and bound, linear programming, and cutting plane methods..., while these algorithms possess different complexities and can solve a wide range of TSP problems, they are not suitable for large-scale scenarios [3].

Evolutionary Algorithms (EAs) are natural-inspired algorithms, that utilize the concept of survival of the fittest to tackle optimization problems. EAs have garnered popularity in the fields of benchmarking and learning problems, and are now widely applied in various real-world situations. They have also been extended to solve learning problems. EAs evolve a set of solutions using selection, crossover, and mutation operators. Their versatility makes them valuable in various domains. Ongoing research aims to improve EAs and apply them to new problem areas and implement the algorithm in novel domains. The Genetic Algorithms (GAs), a subset of EAs, are widely used for solving complex optimization problems that involve single or multiple objective functions [4]. GAs are inspired by the process of natural selection and evolution. They mimic the principles of genetics, to generate and improve candidate solutions iteratively. GAs are commonly used to generate high-quality

ISSN 2734-9373
https://doi.org/10.51316/jst.179.ssad.2025.35.1.3
Received: Jun 11, 2024, revised: Oct 22, 2024;
accepted: Oct 31, 2024.

solutions to optimization and search problems by relying on biologically inspired operators such as mutation, crossover, and selection [5]. Some examples of GAs applications include optimizing decision trees for better performance, solving sudoku puzzles [6], hyperparameter optimization, causal inference [7], etc.

The Generalized Traveling Salesman Problem (GTSP) extends the TSP by introducing the clusters, therefore making the salesman to undertake a single visit to each cluster and find a minimum length trip that includes precisely one node from each cluster, whereas the Clustered Traveling Salesman Problem (CTSP) adds further constraints by forcing the salesman to visit every node in each cluster. Clustered Generalized Traveling Salesman Problem (CGTSP) is a two-layer expansion of traditional TSP in which the external layer of the problem is CTSP and the internal is GTSP [8]. The existing literature concerning the CGTSP is scarce: the problem was introduced by Sepehri, Motlagh, and Ignatius (in [9]) motivated by a practical application for optimal routing of dangerous substances in warehousing operations. The same authors proposed a mixed integer programming formulation of the problem containing another interesting application of the CGTSP in optimizing the robotic automated storage and retrieval system and presented a solution approach based on Cross-Entropy method. CGTSP is a generalized version of TSP where cities are divided into smaller clusters, and these clusters form larger groups [10]. This problem arises in various real-world scenarios, such as vehicle routing in logistics and network optimization in telecommunications. The paper [11], presents a novel solution approach for CGTSP by integrating a genetic algorithm with Dijkstra's shortest path algorithm and a TSP solver. This hybrid method is designed to address the specific constraints and requirements of the CGTSP, which generalizes the well-known TSP by incorporating clusters and sub-clusters of vertices. A new set of instances for the CGTSP derived from the GTSPLIB is introduced in this paper. Nevertheless, the paper does not provide a comprehensive comparison with other existing methods for solving CGTSP. While it mentions that instances used by other researchers could not be obtained, a more detailed comparative analysis would strengthen the validation of their approach. In [12], the transformation method for solving CGTSP by converting it into TSP instances and leveraging powerful TSP solvers like Concorde and Lin-Kernighan-Helsgaun (LKH) significantly outperforms existing methods in terms of solution quality and scalability. It has been applied to large-scale problems and practical logistics scenarios. Heuristic solutions using LKH (see [10]) on transformed TSP instances and exact solutions using Concorde on transformed TSP instances were compared with Integer Programming (IP) formulation solutions provided by CPLEX. However, the results of the experiments heavily rely on certain assumptions, which may not be universally applicable. This means that the efficiency and effectiveness of the CGTSP model as demonstrated in the paper may not be generalizable to all warehouse logistics scenarios.

This paper aims to explore the application of GAs in solving TSP and its variants. Specifically, we will investigate the transformation of CGTSP into TSP and apply GAs to solve TSP instances. We will also present experimental results and evaluate the performance of the proposed approach.

The remainder of the paper is organized as follows: Section 1 provides an overview of TSP and CGTSP. Section 2 introduces GAs and their relevance in optimization. Section 3 presents the conversion of the CGTSP into TSP and outlines the application of GAs to solve TSP. Section 4 describes the experimental setup and evaluates the results. Finally, Section 5 concludes the paper and discusses future research directions.

By exploring the potential of GAs in solving TSP and its variants, this research contributes to advancements in optimization techniques and offers insights into the application of evolutionary algorithms in real-world problem-solving.

2. Transformation from Clustered generalized traveling salesman problem to Traveling Salesman Problem

2.1. Notation

Below are some conventions used throughout this section.

First, the CGTSP problem is represented as a graph. Consider a directed or undirected graph $\Gamma = (V, \mathcal{E}, w)$, where V is the set of nodes, $\mathcal{E}$ is the set of edges connecting pairs of nodes, and $w\colon \mathcal{E} \rightarrow R$ is a mapping function that assigns the corresponding distances between the nodes in $\mathcal{E}$. The set of nodes V is divided into non-intersecting clusters, $V = \bigcup_{c=0}^{N_C-1} C_c$. Each cluster C_c is further divided into non-intersecting $NS(c)$ small clusters

$$C_c = \bigcup_{s=1}^{NS(c)} S_{c,s} \tag{1}$$

where, $S_{c,s}$ represents a subcluster within the cluster C_c, ensuring that each node in V belongs to exactly one subcluster $S_{c,s}$ within its respective cluster C_c. We define $NV(c,s) := |S_{c,s}|$, so the number of nodes in the graph is defined as

$$n = |V| = \Sigma_{c=0}^{N_C-1} \Sigma_{s=1}^{NS(c)} NV(c,s) \tag{2}$$

Similarly, N_S is the total number of subclusters.

$$N_S = \Sigma_{c=0}^{N_C-1} NS(c) \tag{3}$$

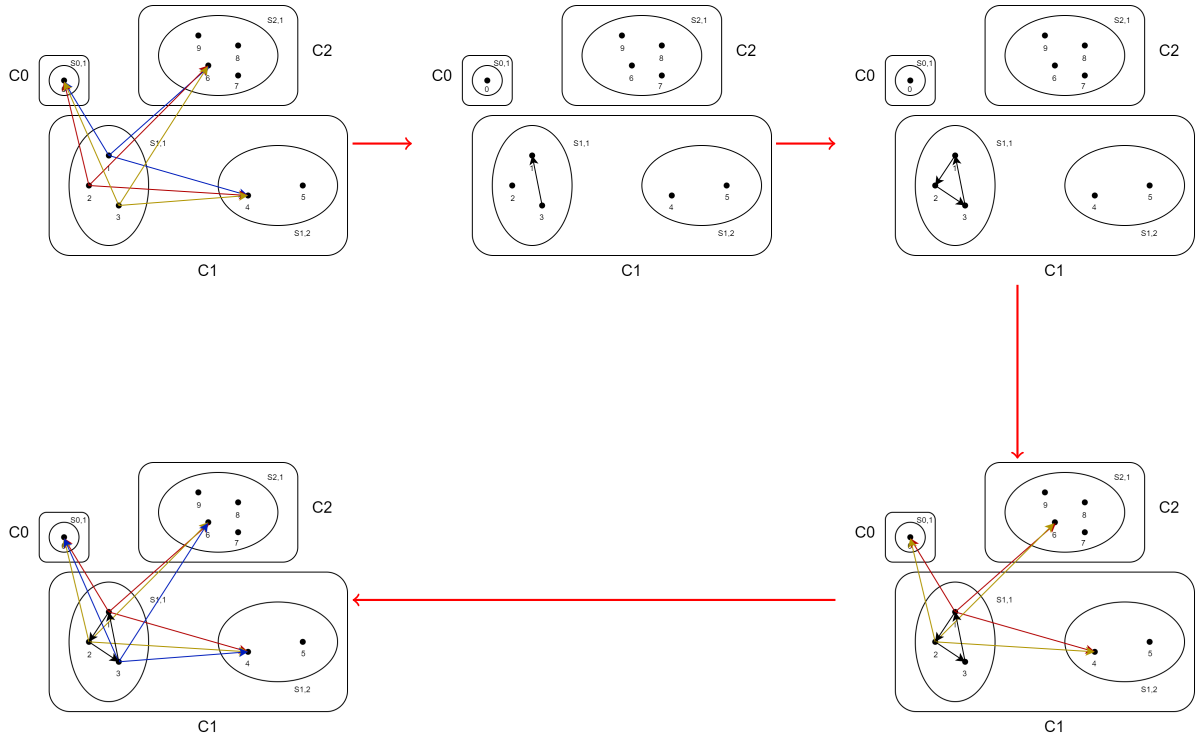


Fig. 1. An example illustrating the steps for converting CGTSP to TSP with constraints on a subcluster of a graph consisting of 10 nodes, 3 clusters, and 4 subclusters.

A node $v = [c, s, i]$ is defined as the i–th node in the subcluster $S_{c,s}$ of cluster C_c. We assume that v_0 is the starting node of the CGTSP problem, and it belongs to the subcluster $S_{0,1}$ of cluster C_0. Each edge $(u, v) \in \mathcal{E}$ represents a path from u to v, and the length of the edge is calculated by $w(u, v)$.

In contrast to TSP, where a tour is a permutation of all nodes, a CGTSP tour contains exactly one node for each subcluster. Specifically, a tour consisting of consecutive nodes in the sequence $\gamma = (v_0, ..., v_0)$ is considered a valid CGTSP tour if it satisfies the following conditions:

- The nodes are visited consecutively in a sequence, starting from v_0 and ending at v_0;
- Exactly one node is selected from each subcluster;
- Suppose v is the ith element in the sequence γ and v belongs to the cluster C_c. Then either the $(i + 1)$-th element also belongs to C_c, or v is the last node in that is visited in the sequence γ.

A CGTSP tour γ is called valid if any consecutive pair of nodes in γ has a valid edge in $\mathcal{E}$.

2.2. Transformation from Clustered Generalized Traveling Salesman Problem to Traveling Salesman Problem with Constraints

Consider $G = (V, E, w_G)$ as a weighted directed graph, where V is the set of vertices corresponding to the vertex set in graph Γ. The set of edges E is constructed as follows, for every small subcluster s of cluster C_c, the following edges are added to the set :

- Edge $([c, s, NV(c, s)], [c, s, 1])$ with weight 0;
- Edge $([c, s, i], [c, s, i + 1])$ for i from 1 to $NV(c, s) - 1$ with weight 0;
- Edge $(u, v), u = [c, s, i]$ and v, where:
  - If $i \neq NV(c, s)$, the weight of edge (u, v) is $w([c, s, i + 1], v)$ in graph Γ;
  - If $i = NV(c, s)$, the weight of edge (u, v) is $w([c, s, 1], v)$ in graph Γ.

The conversion steps are illustrated in the following example in Fig. 1.

Definition 2.2.1 (TSP tour)**.** A TSP tour in G, denoted as $X_T = [x_{u,v}]$, is called a valid TSP tour if it satisfies the following conditions:

$$\sum_{u \in S_{c,s}} \sum_{v \notin S_{c,s}} x_{u,v} = 1, \forall S_{c,s}, \tag{4}$$

$$\sum_{u \in C_c} \sum_{v \notin C_c} x_{u,v} = 1, \forall C_c. \tag{5}$$

Definition 2.2.2. The concepts of incoming node and outgoing node in a tour are defined as following:

- The incoming node of a cluster is a node within the cluster that appears immediately after a node located outside the cluster in the tour T *i.e.*, it is the node that follows a node outside the cluster.
- The outgoing node of a cluster is a node within the cluster that appears immediately before a node located outside the cluster in the tour T *i.e.*, it is the node that precedes a node outside the cluster.

Conditions in Definition 2.2.1 are equivalent to the requirement that a valid route must have exactly one outgoing node for each subcluster and one outgoing node for each cluster. Since T is a route, there is exactly one corresponding incoming node for each outgoing node in T. Therefore, each subcluster and each cluster in the route satisfying the conditions have exactly one incoming node.

Lemma 2.2.1 (Lemma A.1 from [12])**.** *Consider the function $f(T) = T'$, which maps a valid tour T to T'. It is constructed as follows:*

- *Start with the initial tour's starting node v_0.*
- *Iterate through each node in T, excluding v_0, and perform the following steps for each element:*
  - *If the current element is an incoming node, add it to the new sequence.*
  - *Otherwise, skip it and move on to the next element.*
- *Add v_0 to the end of the new sequence.*
- *Then, T' is a valid tour for the CGTSP problem.*

Lemma 2.2.2 (Lemma A.3 from [12])**.** *The total length of a tour γ in CGTSP problem Γ is equal to the total length of a tour T in TSP problem G, and vice versa.*

Theorem 2.2.1 (Theorem 3.2 from [12])**.** *The optimal solution of the CGTSP problem defined on Γ is equal to the optimal solution of the TSP problem with constraints defined on G.*

2.3. Transformation from Traveling Salesman Problem (TSP) with Constraints to Traveling Salesman Problem (TSP) Classic

The TSP is a widely recognized issue that has been thoroughly researched, thus producing beneficial real-world outcomes. However, solving the TSP with constraints still faces limitations, such as the lack of readily available support in some TSP-solving libraries.

Therefore, in this section, we will present the approach of transforming the constrained TSP into a classical TSP problem for the convenience of solving the CGTSP.

First, we partition the edge set of graph G into three disjoint sets: E_1, E_2, E_3:

- E_1 contains edges with nodes from the same subcluster;
- E_2 contains edges with nodes from the same subcluster but not in E_1;
- E_3 includes the remaining edges where the nodes belong to different subclusters.

Let $w(u, v)$ denote the length of the edge (u, v) in graph G, and P_1, P_2 be two specified constants. We construct the graph $G' = (V, E, w'_G)$ according to the following rules:

- The set of vertices V in G' is equivalent to the set of vertices V in G;
- The set of edges E in G' is constructed as following:
  - If $(u, v) \in E_1$ then $w'_G(u, v) = w_G(u, v)$.
  - If $(u, v) \in E_2$ then $w'_G(u, v) = w_G(u, v) + P_1$.
  - If $(u, v) \in E_1$ then $w'_G(u, v) = w_G(u, v) + P_1 + P_2$.

The complete algorithm is described in Algorithm 1.

Algorithm 1: Transformation from CGTSP to TSP

Input CGTSP graph $\Gamma = (V, E, \omega)$ with edge lengths $\omega(u, v)$ for all pairs $(u, v) \in E$, $P1$ and $P2$, NC, NV, $NS(c)$;
Output TSP graph $G' = (V, E, \omega'_G)$;
for $c = 0$ *to* $NC - 1$ **do**
  for $s = 1$ *to* $NS(c)$ **do**
    foreach $u = [c, s, i] \in Sc, s$ **do**
      if $i = NV(c, s)$ **then** $k = 1$
      else
        $k = i + 1$
      end
    end
    $\omega'_G(u, [c, s, k]) = 0$;
    foreach $v \notin Sc, s$ **do**
      if $c \in C_c$ **then** $\omega'_G(u, v) = P1 + \omega_G(k, v)$;
      else
        $\omega'_G(u, v) = P2 + P1 + \omega_G(k, v)$;
      end
    end
  end
end
return *TSP graph* $G' = (V, E, \omega'_G)$

3. Methodology

The genetic algorithm is arguably one of the most well-known and prevalent meta-heuristic optimization algorithms. The novelties in this algorithm created a tectonic shift in how these algorithms were generally perceived, to the point that most of these ideas were implemented in shaping the next generation of meta-heuristic algorithms [13]. It is widely utilized to solve a wide range of optimization problems.

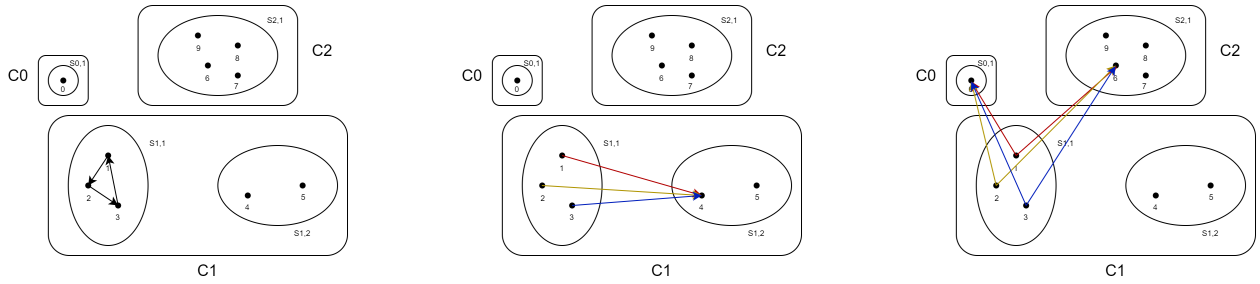


Fig. 2. An example illustrating the classification of edges in TSP. From left to right: E_1, E_2, E_3.

The GAs operates by encoding potential solutions into chromosomes, which are then evaluated for their fitness using a fitness function. Through iterative processes of crossover, mutation, and selection, the algorithm aims to evolve the population towards finding the optimal solution. The key objective is to strike a balance between exploration (to discover new and potentially better solutions) and exploitation (to refine and optimize existing solutions). The detailed algorithmic steps can be found in Algorithm 2.

Algorithm 2: GAs

Input TSP G, T is the number of generations, $r \in [0, 1]$ is the mutation rate
Output A solution to the problem;
Population initialization Population evaluation;
for $k = 0$ *to* T **do**
  Crossover;
  Mutation with rate r;
  Select individual for the next generation;
end
return *Best individual in the last generation.*

The population, an important attribute of the genetic algorithm, plays a crucial role in determining the speed and convergence of the algorithm. For the CGTSP problem, with its specific characteristics of clusters and subclusters, we can leverage these features to initialize the initial population and improve the performance of the algorithm. In the following sections, we will propose three algorithms for population initialization.

3.1. Randomly Constructed Population

Randomly Constructed Population (Algorithm 3) is a simple initialization method for generating a population of tours for TSP. It creates a population of P tours by randomly shuffling the city indices (excluding the starting point) and appending the starting point to each tour. This initialization process guarantees the diversity in the initial population for subsequent genetic operations in solving the TSP.

Algorithm 3: Randomly constructed population

Input number of city C, number of population P;
Output Population;
start_point := 0 ;
for $i = 1, ..., Psize$ **do**
  tour_tmp $\leftarrow [j$ for $j = 1, \ldots, C]$;
  shuffle(tour_tmp);
  tour := start_point + tour_tmp ;
  append(tour) to P;
end
return P

3.2. Population Constructed Based on Clusters

Population Constructed Based on Clusters (Algorithm 4) sets up the population of P tours by randomly shuffling the city indices within each cluster and shuffling the tours from different clusters. The starting point is added to each tour to create a complete tour. This process ensures the diversity and coverage of different clusters in the initial population for subsequent operations in solving the CGTSP.

Algorithm 4: Population constructed based on clusters

Input number of city C, number of population P, number of cluster NC;
Output Population;
start_point := 0;
for $i = 1, ..., Psize$ **do**
  tour_c = [] ;
  for $c = 1, ..., NC$ **do**
    C_c := All nodes in cluster c ;
    tour_tmp := $[j$ for $j = 1, ..., C_c]$;
    shuffle (tour_tmp) ;
    append tour_tmp to tour_c ;
  end
  shuffle (tour_c) ;
  tour := start_point + tour_c ;
  append tour to P ;
end
return P

3.3. Population Constructed Based on Subclusters

Population Constructed Based on Subclusters (Algorithm 5) initializes the population of P tours by randomly selecting nodes within each subcluster,

performing circular shifts on the node sets, and shuffling the tours within each subcluster and cluster. The starting point is added to each tour to create a complete tour. This initialization process maintains the diversity and coverage of different subclusters and clusters in the initial population for subsequent operations in solving the CGTSP.

The "circular_shift" function performs a circular shift operation on a given tour. This operation involves shifting a portion of the tour to the beginning while maintaining the order of the remaining elements.

Algorithm 5: Population constructed based on subclusters

Input number of city C, number of population P, number of cluster NC;
Output Population;
start_point := 0 ;
for $i = 1, ..., Psize$ **do**
  tour_c = [];
  for $c = 1, ..., NC$ **do**
    tour_s = [] ;
    C_c := All nodes in cluster c ;
    for $s = 1, ..., NS(c)$ **do**
      $E_{c,s}$:= Node set in subcluster s of cluster;
      Random choice x in $E_{c,s}$;
      tour_s_tmp := circular_shift($x, E_{c,s}$) ;
      append tour_s_tmp to tour_s ;
    end
    shuffle(tour_s);
    append tour_s to tour_c;
  end
  shuffle tour_c;
  tour := start_point + tour_C ;
  append tour to P;
end
return P

For example, let's say we have a tour represented by the list $(1, 2, 3, 4, 5)$ and we want to perform a circular shift with a random number of 3. The function would check if the random number (3) is present in the tour. If it is, the shift index is determined as the index of 3 in the tour, which is 2.

Then, the function creates a shifted tour by slicing the original tour from the shift index (2) to the end and appending the elements from the beginning of the tour up to the shift index. In this case, the shifted tour would be $(3, 4, 5, 1, 2)$, where the elements after the shift index $(3, 4, 5)$ have been moved to the beginning of the tour while maintaining the order of the remaining elements $(1, 2)$.

In the experimental section, we combine the Nearest Neighbor (NN) algorithm with the proposed genetic algorithm to perform comparisons. We use the NN algorithm to generate a set of tours starting from different cities, apply circular shifts to ensure that the tours start and end at v_0, and then, depending on the population size parameter, either select the tours obtained from the NN algorithm or continue using Algorithm 3, Algorithm 4, or Algorithm 5 to form the initial population.

After obtaining the solution to TSP, we can use the following conversion Algorithm to convert the TSP solution to the original CGTSP. The detailed algorithmic steps can be found in Algorithm 6.

Algorithm 6: Convert result from TSP to CGTSP

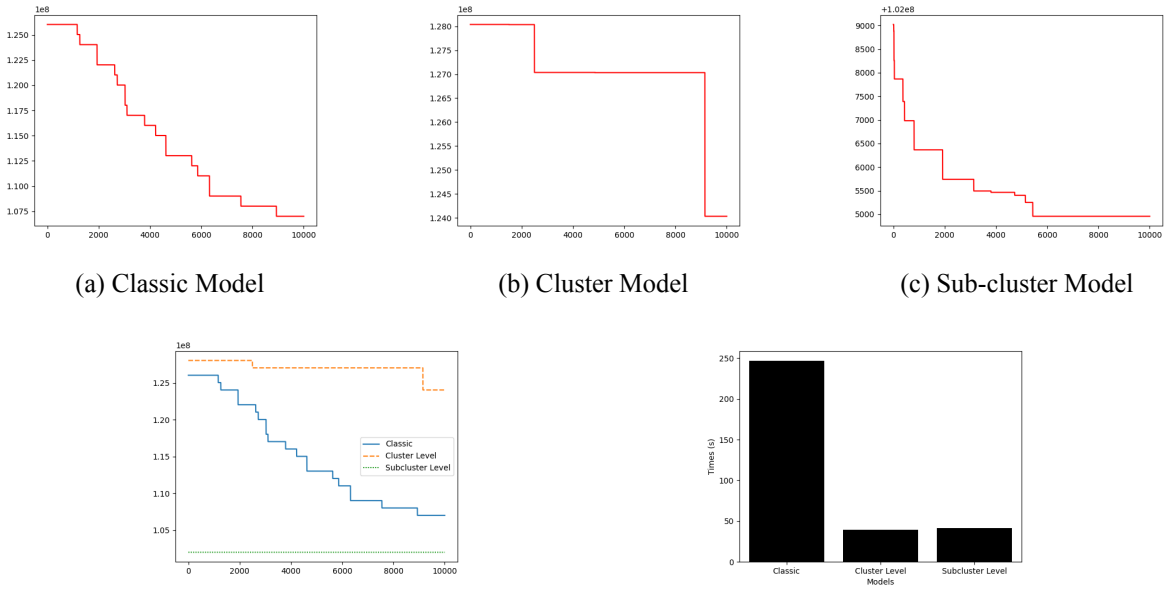
Input TSP tour T;
Output CGTSP tour Υ;
append $T'[0]$ to Υ;
last_tag := cluster c + subcluster s of $T'[0]$;
for $i = 1, ...,$ len(T') **do**
  current_tag := cluster c + subcluster s of T'[index];
  if $current\_tag \neq last\_tag$ **then**
    append T'[index] to Υ;
    last_tag := current_tag;
  end
end
return Υ

4. Experiments

In this section, a dataset will be used that is randomly generated from vertices in a 2D Euclidean space with dimensions of 1000×1000, based on the paper [12]. These datasets are named using a common format rnd<NC>-<NS>-<NV>, where NC is the number of clusters, NS is the number of subclusters within each cluster, and NV is the number of vertices in each subcluster (except the cluster containing the starting city). For each size, problems are generated with high and low values of NC, NS, and NV to distinguish the differences in various cases. By varying parameters such as the number of clusters, the number of subclusters, and the number of vertices in each cluster, we can explore and understand the differences between different combinations of NC, NS, and NV. This helps us gain insights into the factors that influence the performance and results of the algorithm in different scenarios. In summary, by generating problems with different combinations of the number of clusters, the number of subclusters, and the number of vertices (see example in Table1), we can analyze and compare the effectiveness of the algorithm in different situations and draw conclusions regarding the impact of these factors on the algorithm's results. Algorithms are implemented in Python (Python 3.11.9). Experimental results were run on Macbook Air M1, 8GB RAM and macOS Monterey 12.7.5 operating system.

4.1. Evaluating the Results across Different Population Initialization Levels

During the experimentation process, we ran the algorithm on the same dataset, rnd3_7_3n.cgtsp, with the same number of generations to compare the results.



(a) Classic Model (b) Cluster Model (c) Sub-cluster Model

(d) Comparing convergence speed (e) Comparing the runtime for 10000 iterations

Fig. 3. Comparing the results of the three population initialization levels based on the $rnd3\_7\_3n.cgtsp$ dataset.

The results described in Fig. 3. From Fig. 3a to Fig. 3d, the vertical axis represents the distance traveled, and the horizontal axis represents the number of loops. These results indicate that the subcluster-level initialization method performed better than the other two.

With the subcluster-level initialization approach, the population is initially divided into subclusters based on important characteristics and similarities. This helps create an initial diversity in the population and enhances the exploration of the search space. Additionally, the convergence speed of the subcluster-level initialization method is also evaluated to be faster than the other initialization methods. This means that the algorithm has the ability to evolve and find better solutions in a shorter period of time. Over subsequent generations, the population is effectively optimized, leading to improved results.

Table 1. Parameter table

Parameter	**Value**
Number of cities	63
Number of clusters	5
Number of subclusters	17
Number of iterations	10000
Population size	1000

4.2. Evaluating the Algorithm

We compare our genetic algorithm with improved subcluster-level initialization to Google's OR-tools, CPLEX from IBM, and the Nearest Neighbor algorithm. OR-tools is a powerful tool developed by Google to support solving optimization problems, version 9.9, updated in March 2024, used in our calculations. CPLEX is developed by IBM, which is one of the best solvers for solving nonconvex optimization problems. In this paper, CPLEX is utilized through the CPLEX Python API of IBM ILOG CPLEX Optimization Studio version 22.1.1. It was configured with a maximum runtime of 3600 seconds. Remarkable results were obtained from different datasets, leading to improvements in our genetic algorithm.

The Table 2 present the algorithm's performance using the Subcluster Model for population construction with a maximum of 30,000 iterations, and a population of 100 individuals. The algorithm stops when the population does not improve after 1,000 iterations.

The comparison results show that our model, which uses the Subcluster Model to build the population, outperforms both OR-Tools and CPLEX on most datasets. This is proof that our algorithm with the enhanced subcluster-level initialization method implemented is more effective than OR-Tools. When compared with the Nearest Neighbor algorithm, our algorithm gives better or equivalent results when the number of cities is small. However, when the number of cities is large, the Nearest Neighbor algorithm performs much better than our algorithm. The algorithm depends heavily on random factors, making it challenging to find good results when the number of cities is large. For better results, we can add the journeys found by the Nearest Neighbor algorithm to the initial population. In this way, the genetic algorithm will ensure better or at least equivalent results compared to the Nearest Neighbor algorithm. Indeed, when adding tours generated using the NN algorithm, GAs-NN results outperform NN in most datasets.

Table 2. Dataset of CGTSP problems and result

Dataset	N_C	N_S	$NV(c,s)$	n	Model	No. of cities	Tour length	Time (s)	No. of iter.
rnd2_2_2n	2	2	2	8	OR-tools	4	3,215.71	0.04	-
					CPLEX	4	**2,017.50**	0.80	-
					GAs	4	**2,017.50**	3.12	1,189
					NN	4	2,146.01	0.00	-
					GAs-NN	4	**2,017.50**	2.04	1,236
rnd3_3_3n	3	3	3	27	OR-tools	9	5,379.91	0.02	-
					CPLEX	9	4,093.12	3,600	-
					GAs	9	3,114.23	14.00	2,976
					NN	9	2,757.58	0.00	-
					GAs-NN	9	**2,755.12**	7.82	2,552
rnd3_3_7n	3	3	7	63	OR-tools	9	6,023.22	0.02	-
					CPLEX	9	4,569.22	3,600	-
					GAs	9	2,054.94	87.02	10,044
					NN	9	1,945.08	0.04	-
					GAs-NN	9	**1,899.40**	38.98	7,155
rnd3_7_3n	3	7	3	63	OR-tools	21	11,816.69	0.04	-
					CPLEX	21	7,297.64	3,600	-
					GAs	21	6,994.72	41.84	4,949
					NN	21	5,819.11	0.00	-
					GAs-NN	21	**5,033.11**	6.94	1,234
rnd7_3_3n	7	3	3	63	OR-tools	21	14,129.20	0.04	-
					CPLEX	21	9,301.79	3,600	-
					GAs	21	6,193.63	86.24	8,904
					NN	21	5,279.85	0.07	-
					GAs-NN	21	**5,123.21**	43.28	7,072
rnd4_4_4n	4	4	4	64	OR-tools	16	9,171.72	0.04	-
					CPLEX	16	5,914.08	3,600	-
					GAs	16	4,453.49	111.74	12,435
					NN	16	3,118.49	0.05	-
					GAs-NN	16	**3,030.12**	100.07	17,436
rnd5_5_5n	5	5	5	125	OR-tools	25	12,320.31	0.05	-
					CPLEX	25	12,201.83	3,600	-
					GAs	25	7,519.89	190.01	11,799
					NN	25	**4,111.29**	0.26	-
					GAs-NN	25	**4,111.29**	11.81	1,141
rnd14_3_3n	14	3	3	126	OR-tools	42	19,268.74	0.04	-
					CPLEX	42	14,079.06	3,600	-
					GAs	42	9,604.16	217.59	11,990
					NN	42	10,052.22	0.00	-
					GAs-NN	42	**8,403.07**	32.75	2,735
rnd3_14_3n	3	14	3	126	OR-tools	42	21,498.46	0.03	-
					CPLEX	42	15,733.79	3,600	-
					GAs	42	14,662.30	139.72	9,055
					NN	42	7,135.61	0.00	-
					GAs-NN	42	**6,923.76**	10.83	1,089
rnd3_3_14n	3	3	14	126	OR-tools	9	4,195.29	0.02	-
					CPLEX	9	4,158.06	3,600	-
					GAs	9	2,454.93	132.46	8,842
					NN	9	1,784.47	0.13	-
					GAs-NN	9	**1,769.72**	79.30	8,129
rnd24_3_3n	24	3	3	216	OR-tools	72	38,511.95	0.05	-
					CPLEX	75	28,380.62	3,600	-
					GAs	72	21,402.36	939.30	30,000
					NN	72	15,295.22	0.01	-
					GAs-NN	72	**14,275.94**	22.02	1,021

Dataset	N_C	N_S	$NV(c,s)$	n	Model	No. of cities	Tour length	Time (s)	No. of iter.
rnd3_24_3n.	3	24	3	216	OR-tools	72	37,659.39	0.04	-
					CPLEX	72	23,760.38	3,600	-
					GAs	72	29,076.82	362.14	14,205
					NN	72	bf 9,310.53	0.01	-
					GAs-NN	72	bf 9,310.53	20.73	1,257
rnd3_3_24n.	3	3	24	216	OR-tools	9	6,332.79	0.06	-
					CPLEX	10	5,415.27	3,600	-
					GAs	9	2,257.53	472.51	18,951
					NN	9	bf 1,163.83	0.00	-
					GAs-NN	9	bf 1,163.83	272.79	17,191
rnd6_6_6n.	6	6	6	216	OR-tools	36	20,347.10	0.06	-
					CPLEX	41	19,098.85	3,600	-
					GAs	36	10,448.24	345.83	13,334
					NN	36	5,772.67	0.01	-
					GAs-NN	36	bf 5,084.61	18.80	1,100
rnd38_3_3n.	38	3	3	342	OR-tools	114	58,618.70	0.10	-
					CPLEX	149	72,770.72	3,600	-
					GAs	114	39,335.42	599.47	14,190
					NN	114	24,615.13	0.03	-
					GAs-NN	114	bf 23,972.38	36.26	1,056
rnd3_38_3n.	3	38	3	342	OR-tools	114	57,728.84	0.10	-
					CPLEX	127	48,217.65	3,600	-
					GAs	114	48,412.03	61.90	2,640
					NN	114	11,873.87	10.26	-
					GAs-NN	114	bf 11,491.81	45.31	1,745
rnd3_3_38n.	3	3	38	342	OR-tools	9	6,174.50	0.10	-
					CPLEX	11	6,991.74	3,600	-
					GAs	9	2,163.70	149.83	6,469
					NN	9	1,188.36	0.00	-
					GAs-NN	9	bf 970.54	456.04	18,312
rnd7_7_7n.	7	7	7	343	OR-tools	49	27,340.87	0.10	-
					CPLEX	67	36,467.62	3,600	-
					GAs	49	16,207.55	556.48	22,332
					NN	49	7,220.46	0.01	-
					GAs-NN	49	bf 6,381.95	28.98	1,112
rnd8_8_8n.	8	8	8	512	OR-tools	64	31,373.66	0.28	-
					CPLEX	93	44,437.42	3,600	-
					GAs	64	22,110.48	753.95	17,890
					NN	64	7,580.50	0.01	-
					GAs-NN	64	bf 7,440.87	42.94	1,094
rnd3_3_57n.	3	3	57	513	OR-tools	9	5,612.53	0.26	-
					CPLEX	12	5,518.44	3,600	-
					GAs	9	1,424.49	518.10	12,396
					NN	9	870.16	0.00	-
					GAs-NN	9	bf 714.59	501.73	13,187
rnd3_57_3n.	3	57	3	513	OR-tools	171	91,959.73	0.28	-
					CPLEX	513	274,056.25	3,600	-
					GAs	171	78,479.76	115.98	3,144
					NN	171	15,421.55	0.07	-
					GAs-NN	171	bf 14,798.62	62.89	1,546
rnd57_3_3n.	57	3	3	513	OR-tools	171	89,717.16	0.04	-
					CPLEX	230	111,029.69	3,600	-
					GAs	171	56,135.50	981.67	21,056
					NN	171	35,130.73	0.06	-
					GAs-NN	171	bf 32,684.63	53.56	1,011

5. Conclusion

This paper investigates the application of genetic algorithms to combinatorial optimization problems that have various practical applications. Specifically:

- Represent the Traveling Salesman Problem and the Clustered Traveling Salesman Problem problems.
- Presentation of the results regarding the application of the genetic algorithm to the traveling salesman problem and the clustered traveling salesman problem.

The experimental results demonstrate the great potential of the proposed genetic algorithm. It can also be a good solution approach for similar problems. Future directions of research could explore the integration of genetic algorithms with other methods and algorithms, such as machine learning or parameter tuning optimization, to enhance performance and the ability to solve complex problems.

6. References

[1] D. L. Applegate *et al.*, The Traveling Salesman Problem: A Computational Study, Princeton University Press, 2006, pp. 606.

[2] G. Prokudin *et al.*, Traveling Salesman Problem in the function of freight transport optimization, Journal of Sustainable Development of Transport and Logistics, vol. 3, pp. 29–36, Apr. 2018. https://doi.org/10.14254/jsdtl.2018.3-1.3

[3] G. Laporte, The Traveling Salesman Problem: An overview of exact and approximate algorithms, European Journal of Operational Research, vol. 59, no. 2, pp. 231–247, 1992.

[4] Agash Uthayasuriyan *et al.*, A comparative study on Genetic Algorithm and Reinforcement Learning to solve the Traveling Salesman Problem, Research Reports on Computer Science, vol. 3, pp. 1–12, 2023. https://doi.org/10.37256/rrcs.2320232642

[5] M. Mitchell, An Introduction to Genetic Algorithms, The MIT Press, 1998.

[6] F. Gerges, G. Zouein, and D. Azar, Genetic algorithms with local optima handling to solve sudoku puzzles, Proceedings of the 2018 International Conference on Computing and Artificial Intelligence, pp. 19–22, 2018. https://doi.org/10.1145/3194452.3194463

[7] M. C. Burkhart and G. Ruiz, Neuroevolutionary representations for learning heterogeneous treatment effects, Journal of Computational Science, vol. 71, pp. 102054, 2023.

[8] Y. Du *et al.*, Simultaneous pickup and delivery Traveling Salesman Problem considering the express lockers using attention route planning network, Computational Intelligence and Neuroscience, vol. 2021, no. 1, pp. 5590758, 2021.

[9] M. M. Sepehri, S. M. H. Motlagh, and J. Ignatius, A model for optimal routing of dangerous substances in warehousing operations via k-nested GTSP, International Journal of Nonlinear Sciences and Numerical Simulation, vol. 11, no. 9, pp. 701–704, 2010.

[10] K. Helsgaun, Solving the Clustered Traveling Salesman Problem using the Lin-Kernighan-Helsgaun Algorithm, Roskilde University, May 2014, pp. 1–13.

[11] O. Cosma, P. Pop, and L. Cosma, A hybrid based genetic algorithm for solving the Clustered Generalized Traveling Salesman Problem, Hybrid Artificial Intelligent Systems, LNCS, pp. 352–362, Aug. 2023. https://doi.org/10.1007/978-3-031-40725-3_30

[12] P. Baniasadi *et al.*, A transformation technique for the Clustered Generalized Traveling Salesman Problem with applications to logistics, European Journal of Operational Research, vol. 285, pp. 444–457, 2020.

[13] B. Zolghadr-Asli, Computational Intelligence-based Optimization Algorithms: From Theory to Practice, 1st ed., CRC Press, 2024, pp. 356.