Computing and Informatics, Vol. 44, 2025, 1295–1335, doi: 10.31577/cai_2025_6_1295

AN IMPROVED GENETIC ALGORITHM FOR SOLVING THE CLUSTERED STEINER TREE PROBLEM

Tuan-Anh Do, Ha-Bang Ban*, Dang-Hai Pham, Le Minh Tu

School of Information and Communication Technology
Hanoi University of Science and Technology
Hanoi, Vietnam
e-mail: {AnhDT, BangBH, HaiPD}@soict.hust.edu.vn, minhtutx@gmail.com

Abstract. In a complex network comprising many devices, a set of nodes may be partitioned into multiple local clusters with distinct functions, properties, or communication protocols. Thus, there has been an increase in network design problems with additional constraints regarding the clustering of vertices, one of which is the Clustered Steiner Tree Problem – a variant of the Steiner Tree Problem. There have been a few studies working on this problem in the literature, but they either solve it only in the metric case or their exploration capability remains limited. Therefore, their results are not good in many cases. To overcome the drawbacks, we propose a Priority-Based Genetic Algorithm to solve the Clustered Steiner Tree Problem. The proposed algorithm maintains a balance between exploration and exploitation to prevent the search from getting stuck in local optima. Experiments and comparisons to existing works in non-metric and metric cases are carefully conducted to prove the remarkable performance of the proposed algorithm.

Keywords: Genetic algorithm, clustered Steiner Tree problem

1 INTRODUCTION

The Internet of Things (IoT) has led to a rapid increase in the number of interconnected devices with advanced features and capabilities, creating massive computer networks. To facilitate the transmission and management process, ensure the scalability of the network architecture, and protect privacy between different domains,

* Corresponding author

many problems related to partitioning nodes into clusters (or domains) have been proposed. This has led to the development of a class of clustered graph problems, which are important for theoretical and practical reasons, as they allow us to find computation paths between nodes within and between clusters. Some clustered graph problems have variants such as the Clustered Traveling Salesman Problem (CluTSP) [1], Minimum Routing Cost Clustered Tree Problem (CluMRCT) [2], Clustered Shortest-Path Tree Problem (CluSPT) [3].

By grouping devices with similar properties into one domain for better management and security, the Clustered Steiner Tree Problem (CluSteiner) was developed to meet the demands of modern network systems. Its output solution is a graph containing all vertices of different clusters, together with some independent nodes, with the lowest cost possible. Therefore, this problem has practical applications in multipoint networks with clusters of different functionalities and protocols. When costs, latency, power consumption, and bandwidth usage are minimized, networks will operate more efficiently. Throughout history, there have been many research related to this problem, such as designing customized wireless networks to minimize the number of intermediate vertices to reduce latency and collision [4], reducing energy consumption for transmitting signals [5], increasing data distribution speed and error handling in connecting data centers [6].

Belonging to the NP-hard class, these problems are usually approached by approximate methods such as Genetic Algorithm (GA) [7, 8] and Tabu search [9] for CluTSP problem, multifactorial evolutionary algorithms [10] and approximation algorithms [11] for the CluMRCT problem and the other related problems [12, 13]. These approaches divide the problem into two stages:

1. finding a subgraph for each cluster and
2. connecting the subgraphs into a full graph.

Following that trend, many studies on the CluSteiner problem also use two-level approaches to find approximate solutions. However, two-level approaches have some drawbacks. Some algorithms are not suitable for sparse or non-metric graphs. Greedy algorithms, which are used in some two-level approaches, may not explore all possible solutions and can get stuck in local optima. Additionally, some two-level algorithms can be computationally expensive.

In this paper, we propose a method using GA based on the evolutionary principle of nature that only the fittest individuals can survive. Compared with previous works, we use a priority-based encoding method, and a novel algorithm for finding the Steiner tree has been developed. Furthermore, we also conduct experiments on our approach and analyze the results to evaluate its efficiency. The main contributions are summarized as follows:

- Propose a representation method for the CluSteiner problem and a new scheme to find the Steiner tree according to the given order of vertices to improve the solution space exploration and yield more efficient results.

- Devise a prefiltering process for the graph to reduce the complexity of computing output solutions.
- Evaluate the performance of the proposed algorithm by conducting extensive experiments on various datasets and comparing the results to previous state-of-the-art methods.

The rest of the papers are organized as follows: Section 2 presents related works. Section 3 provides problem formulation, while Section 4 describes our proposed algorithm in detail. Section 5 contains experimental results and comparisons with other algorithms. Finally, Section 6 presents some final remarks and future aspects of our work.

2 RELATED WORKS

Steiner Tree Problem (STP), named after the mathematician Jakob Steiner, is a classical combinatorial problem. In [14], it was proved to be a problem belonging to the NP-Hard class and has been rigorously studied with various proposals. If the number of required vertices is equal to the number of vertices in the original graph, the problem converts to Minimum Spanning Tree (MST) problem. Therefore, proposals for the MST problem can produce a feasible solution for the CluSteiner problem. The authors in [15, 16] have utilized MST algorithms and prove that their approach is a 2-approximation algorithm, which means that the cost of its solution is at most twice the cost of the optimal solution in the case of metric graph. Two exact algorithms to solve the MST problem are Prim [17] and Kruskal algorithms [18]. In other studies, Wu and Lin proposed a two-level algorithm to solve the CluSteiner problem, Bilevel Minimum Spanning Tree (BMST), which finds the minimum spanning tree for each cluster and connects the sub-trees together using a p-approximation algorithm [15]. In the Clustered Selected-internal Steiner Tree Problem (CSISTP) [16], a variant of the CluSteiner problem with vertex constraints, Chen also used a similar two-level approach. The MST algorithm is suitable for finding sub-trees at the first level of the CluSteiner problem because the clusters of the input graph only contain destination vertices without intermediate vertices. However, this method can only be applied when the sub-graphs of G on each cluster are connected graphs. Additionally, the $(2+p)$ approximation ratio is only valid when the graph is metric.

Notably, in the research about multi-domain Steiner tree [19], Chen et al. proposed the Shortest-Path Heuristic (SPH), which initialized the tree with a random vertex in the required set of vertices, then it adds the nearest vertex one by one until all required vertices have been added. The algorithm was proved to have an upper bound of $2 \times (1 - \frac{1}{k})$ (k being the number of destination vertices) times the optimal solution. Additionally, the results can be improved by running the algorithm several times since each starting point gives a different solution. Consequently, under the context of non-metric graph, SPH is more efficient than MST algorithms.

In [19], the authors examined another clustering variant of the Steiner tree problem, with inputs being graphs with all vertices clustered. The problem is approached similarly, with both levels finding the Steiner tree using the SPH algorithm. This method can be applied to the CluSteiner problem by treating all unconnected intermediate vertices as the intermediate vertex set of the cluster when considering each cluster. At this point, the order of visiting the cluster significantly impacts the algorithm's results.

Based on this foundation, Anh et al. proposed a two-level SPH algorithm, Bilevel Shortest Path Heuristics (BSPH), and a genetic algorithm based on SPH, Shortest-Path Genetic Algorithm (SPGA) [20]. The BSPH algorithm randomly selects the order of finding sub-trees for each cluster, while the SPGA uses permutation encoding to represent the order. Experimental results show that SPGA and BSPH have overcome the limitations of BMST in non-metric space. However, both used the SPH algorithm to find the Steiner tree. Following only the shortest path renders SPH incapable of exploring all solution spaces. Additionally, optimizing each cluster separately can make it hard for the overall algorithm to find optimal results: the first visited cluster may use up all intermediate vertices, hence making subsequent clusters and inter-cluster connections less feasible, so the total weight of the tree is increased. Therefore, two-level algorithms similar to BMST or BSPH cannot improve the solution by using more efficient algorithms to find the Steiner tree, not to mention the possibility of longer computation time.

In this paper, we are dedicated to resolving the aforementioned disadvantages of existing algorithms by proposing a Priority-Based Search (PBS) algorithm to find the Steiner tree. To the best of our knowledge, PBS is historically an effective method for solving this type of problem. Therefore, it can have remarkable exploration capability in finding the clustered Steiner tree, hence maintaining the diversity of the proposed algorithm. As a result, the proposed algorithm obtains better solutions than others in the literature.

3 PROBLEM FORMULATION

The CluSteiner is a variant of the classical Steiner Tree Problem in graphs. In the CluSteiner, given an undirected weighted graph $G = (V, E, w)$ and a set of required vertices R, the objective is to find a minimum weighted acyclic connected subgraph, i.e., a tree, in G that spans all vertices in R. The non-required vertices (vertices belonging to $V \setminus R$) used as intermediate points in the tree are called Steiner vertices. In addition to the requirements of Steiner Tree Problem, the CluSteiner also provides a partition $R' = \{R_1, R_2, \ldots, R_k\}$ along with a clustering constraint such that a Steiner tree T must be a clustered tree for R'. A Steiner tree T is a clustered tree for R' if all the local trees are mutually disjoint, where a local tree of R_i in T is the minimal subtree of T spanning R_i.

A formal definition for CluSteiner is given in Table 1.

Input	– A weighted, complete graph $G = (V; E; w)$ – A set of required vertices $R \subset V$ – A partition $R' = \{R_1, R_2, \ldots, R_k\}$ of R; R_i is the i^{th} cluster
Output	A clustered Steiner tree $T = (V_T, E_T)$ for R'
Objective	Minimize $\Sigma_{e \in E_T} w(e)$
Constraints	– T is a Steiner tree: $R \subset V_T$ – A local tree $T_i = (V_{T_i}, E_{T_i})$ is a Steiner tree of cluster R_i: $R_i \subset V_{T_i}$ – All local trees are mutually exclusive: $\forall 1 \leq i < j \leq k$, $V_{T_i} \cap V_{T_j} = \emptyset$

Table 1. Clustered Steiner Tree problem definition

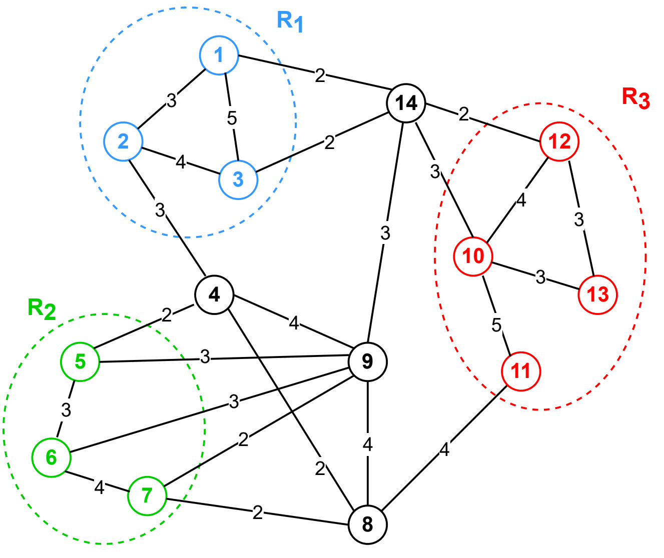

Figure 1. Graph G with a set of destination vertices partitioned into 3 clusters R_1, R_2, and R_3

An example of the input graph is illustrated in Figure 1, an invalid solution and a valid one are shown in Figures 2, 3, respectively. In Figure 2, both the minimal local trees of cluster R_1 and cluster R_2 must contain vertex 14, so they are not mutually exclusive; hence the invalidity. Meanwhile, in Figure 3, the graph can be divided into three separate local trees $\{1, 2, 3\}$, $\{5, 6, 7, 9\}$, and $\{10, 11, 12, 13\}$ spanning the three clusters R_1, R_2, and R_3, respectively.

4 PROPOSED ALGORITHM

Previous sections have introduced the Clustered Steiner Tree Problem. In this section, an approach based on a Genetic Algorithm to solve CluSteiner is described in detail.

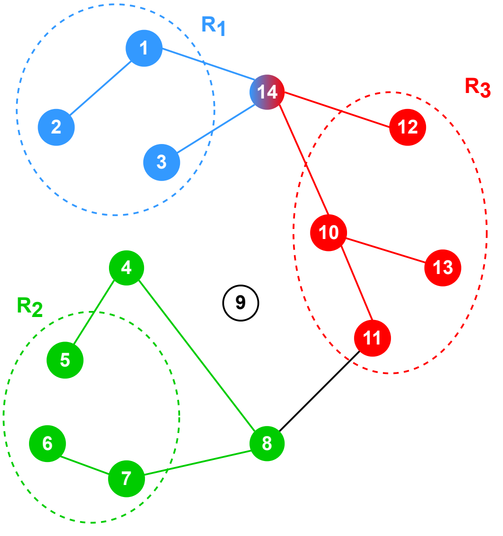

Figure 2. An example of invalid solution

4.1 Prefiltering

The process of transforming from the graph G to G' is carried out every time an individual is evaluated, thus avoiding repeating the same computational steps between

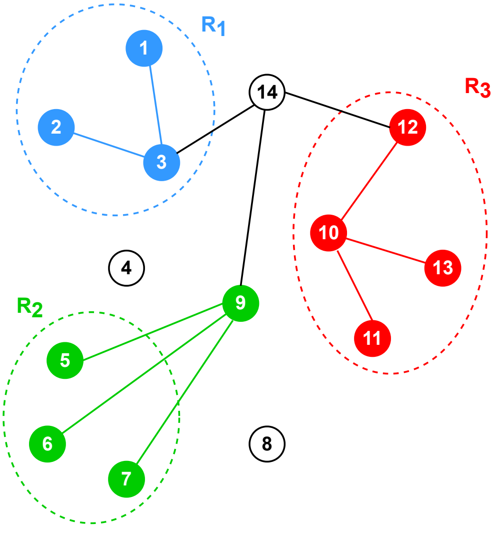

Figure 3. An example of valid solution

evaluations helps to reduce computation time significantly. This paper analyzes the calculations of the edges' weights in constructing G' and performs common operations in the prefiltering step. With the sub-tree T_i, the vertex set E_{T_i} is partitioned into two parts: the destination vertex set R_i and the Steiner vertex set S_i. Set S_i is determined by the results at the first level and differs between solutions, but R_i is determined by the input of the problem and remains unchanged. Therefore, the edge weights related to vertices in R_i are the same between solutions, and can be calculated beforehand to avoid repetition. Considering F and F' as the sets of intermediate vertices in G and G', respectively, we can examine the formula for determining the weights of edges in G' as follows:

- The weight of the edge between two intermediate vertices $u, v \in F'$ is equal to the corresponding weight between two vertices in G.

$$w'(u, v) = w(u, v).$$

- The weight of the edge between an intermediate vertex and the destination vertex: given the definition of the weight between vertex u and cluster R_i is $d(u, R_i) = \min_{v \in R_i} w(u, v)$, the formula $w'(u, t_i) = \min_{v \in V_{T_i}} w(u, v)$ is analyzed as follows:

$$\begin{aligned} w'(u, t_i) &= \min_{v \in V_{T_i}} w(u, v) \\ &= \min\left(\min_{v \in R_i} w(u, v), \min_{v \in S_i} w(u, v)\right) \\ &= \min\left(d(u, R_i), \min_{v \in S_i} w(u, v)\right). \end{aligned}$$

If the distance $d(u, R_i)$ is calculated beforehand, the computational complexity of calculating $w'(u, t_i)$ decreases by $O(|Ri|)$. In Figure 4, $w'(9, t_2)$ is computed only considering the weights to the cluster R_2 and vertices 4, 8 (set S_2) instead of considering all the edges from vertex 9 to 4, 5, 6, 7, 8.

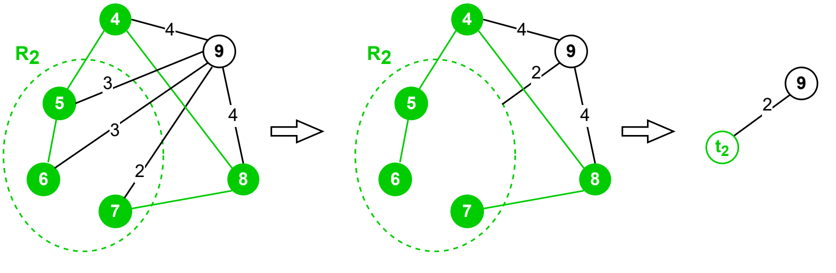

Figure 4. Weight of edge (9, t_2) with destination vertex t_2 simplified from the tree T_2

- Weight of the edge between two destination vertices: given the definition of weight of the edge between two clusters, $d(R_i, R_j) = \min_{\substack{u \in R_j \\ v \in R_j}} w(u, v)$, we have:

$$\begin{aligned}
w'(t_i, t_j) &= \min_{\substack{u \in V_{T_i} \\ v \in V_{T_j}}} w(u, v) \\
&= \min_{u \in V_{T_i}} \left(\min_{v \in R_j} w(u, v), \min_{v \in S_j} w(u, v) \right) \\
&= \min \left(\min_{\substack{u \in R_i \\ v \in R_j}} w(u, v), \min_{\substack{u \in R_i \\ v \in S_j}} w(u, v), \min_{\substack{u \in S_i \\ v \in R_j}} w(u, v), \min_{\substack{u \in S_i \\ v \in S_j}} w(u, v) \right) \\
&= \min \left(d(R_i, R_j), \min_{v \in S_j} d(v, R_i), \min_{u \in S_i} d(u, R_j), \min_{\substack{u \in S_i \\ v \in S_j}} w(u, v) \right).
\end{aligned}$$

Prefiltering step calculates the weights to destination vertices in advance, which helps reduce the computational complexity of the calculation $w'(t_i, t_j)$ from $O(|E_{T_i}| \times |E_{T_j}|)$ to $O(|S_i| \times |S_j|)$.

The process of constructing the graph G' for each evaluation has a computational complexity equal to the sum of the computational complexity of calculating the weights:

$$\begin{aligned}
O(compute(G')) &= O(|F'|^2) + O\left(|F'| \times \sum_{i=1}^{k} |S_i| \right) + O\left(\sum_{i=1}^{k-1} \sum_{j=i+1}^{k} |S_i||S_j| \right) \\
&< O\left(\left(|F' + \sum_{i=1}^{k} |S_i| \right)^2 \right) = O(|F|^2).
\end{aligned}$$

4.2 Priority-Based Search

The PBS algorithm is developed based on the SPH algorithm and applied to the Steiner tree problem. The tree T needs to cover a pre-defined set of vertices in a specific order. In each iteration, the algorithm traverses the vertices in the graph and updates the distances from their neighbors to the tree. When the target vertex with the shortest path is determined, PBS adds that path to the treetree T and starts a new iteration with the next target vertex. The modified PBS algorithm is shown in Algorithm 1. Some notations:

- Q: Set of vertices that have not been traversed yet, initially containing all vertices in V. After traversing a vertex, the algorithm determines the shortest distance from that vertex to tree T. When tree T adds a new vertex, the distances from the tree to other vertices may change. Therefore, the algorithm needs to re-traverse the recently added vertex and the vertices that are not part of tree T. Note that traversing a vertex does not necessarily mean it has been added to tree T.

- $d(v)$: Shortest distance from vertex v to tree T. Initially, $d(v) = \infty$ for all $v \in V$.
- $p(v)$: Predecessor vertex of v on the shortest path to tree T.

Algorithm 1 Implementation of PBS

Input: $G = (V, E, w)$, $R = r_1, r_2, \ldots, r_k \subset V$, the order $r_1, r_2, \ldots, r_k$
Output: A Steiner tree $T = (V_T, E_T) : R \subset V_T$

$V_T \leftarrow \emptyset$, $E_T \leftarrow \emptyset$
$Q \leftarrow V$
$\forall v \in V : d(v) \leftarrow \infty$
$V_T \leftarrow V_T \cup \{r_1\}$, $d(r_1) \leftarrow 0$
for $i \leftarrow 2$ to k **do**
  while $\min_{u \in Q}\ d(u) < d(r_i)$ **do**
    $u \leftarrow \arg\min_{u \in Q}\ d(u)$
    $Q \leftarrow Q \setminus \{u\}$
    for $\forall (u, v) \in E$ **do**
      if $d(u) + w(u, v) < d(v)$ **then**
        $d(v) \leftarrow d(u) + w(u, v)$
        $p(v) \leftarrow u$
      end if
    end for
  end while
  $u \leftarrow r_i$
  while $u \notin V_T$ **do**
    $V_T \leftarrow V_T \cup \{u\}$, $E_T \leftarrow E_T \cup \{(p(u), u)\}$
    $d(u) \leftarrow 0$, $Q \leftarrow Q \cup u$
    $u \leftarrow p(u)$
  end while
end for
return T

Both the SPH and PBS algorithms add paths from tree T to the destination vertices, but they differ in their strategies for selecting the vertices to be added. While SPH selects the nearest vertex, PBS chooses vertices in a predetermined order and considers it as part of the problem input. Therefore, though PBS is not as efficient as SPH in finding the Steiner tree, it explores the solution space better and ensures a better overall optimized result in the CluSteiner problem.

4.3 Priority-Based Genetic Algorithm

4.3.1 Population Initialization

The initial population has a significant impact on the results of algorithm, so it needs to be carefully selected based on the problem, representation and individual evaluation. The evaluation process of individual in Priority-Based Genetic Algorithm (PBGA) uses the PBS algorithm. Although the PBS algorithm has good exploration capabilities in the solution space, it is less efficient than greedy algorithms in constructing new solutions.

Therefore, the population is randomly initialized, which does not guarantee quality and requires the use of heuristic methods. The proposed algorithm uses BSPH to initialize the population. The BSPH algorithm can generate different solutions by randomly selecting a starting point and the order of searching for sub-trees. As a result, the initial population ensures both diversity and relatively good quality.

4.3.2 Chromosome Representation

The evaluation process is based on two-level approach, in which the order of clusters when searching for sub-trees at the first level must be determined. Additionally, the order of vertices needs to be determined in advance as it is part of the input for the PBS algorithm in finding Steiner trees (presented in Section 4.2). Therefore, the representation of an individual needs to express the following factors: the order of searching for sub-trees within clusters, the order of destination vertices within each cluster, and the order of searching for destination vertices (extracted from the sub-trees) at the second level.

In the PBGA algorithm, each individual is encoded using a real-valued sequence of length $|R|$ representing the priority of destination vertices and clusters. At the first level, the highest priority among the vertices in a cluster determines the order of searching for sub-trees. Within a cluster, the priority of vertices determines the traversal order of the PBS algorithm when searching for the sub-tree of that cluster. At the second level, the average priority of vertices within a cluster determines the corresponding order of destination vertices when searching for tree T'.

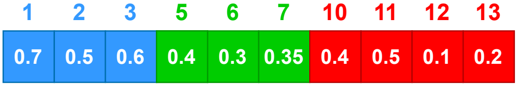

Figure 5. Real number sequence represents an individual

Figures 5, 6, 7, 8 illustrate an individual representation for the problem shown in Figure 1. The blue, red, and green colors of the cells in Figure 5 correspond to clusters R_1, R_2, and R_3, respectively. The numbers above the cells represent the vertices that the cells stand for. The order of clusters when searching for sub-trees is determined by the highest priority within each cluster. In Figure 6, the order is 0.7, 0.4, 0.5 corresponding to vertex 1 of cluster 1, vertex 5 of cluster 2, and vertex 11 of cluster 3. Therefore, the evaluation process sequentially searches for sub-trees T_1, T_3, and T_2. For sub-tree T_1, the PBS algorithm traverses vertices 1,3 and 2 in the priority shown in Figure 7. The order destination vertices in graph G' is determined by the average priority of the corresponding cluster. In Figure 8, the order is 0.6, 0.35 and 0.3. Thus, on G', the PBS algorithm searches for the tree T' in the order of vertices t_1, t_2, t_3.

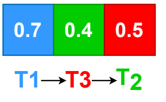

Figure 6. Order of clusters when finding sub-trees

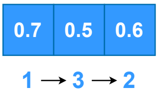

Figure 7. Order of vertices when finding sub-tree T_1

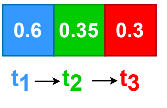

Figure 8. Order of clusters when connecting at the second level

4.3.3 Chromosome Evaluation

The evaluation process of an individual involves constructing a clustered Steiner tree and calculating its total weight. The solution search process consists of two levels, each level using the PBS algorithm to find a Steiner tree:

- First level: Using the PBS algorithm to find a sub-Steiner tree for each cluster in the order determined by cluster priority.
- Second level: Constructing graph G' from graph G and the sub-trees, then finding inter-cluster connections using PBS.

Let $\alpha(T)$ be the sum of all local trees' costs and $\beta(T)$ be the sum of all inter-cluster links. The fitness value of the chromosome, i.e., the cost of clustered Steiner tree T, is:

$$c(T) = \alpha(T) + \beta(T).$$

4.3.4 Find the Sub-Trees Within Clusters

The order of finding sub-trees is determined by the highest priority of vertices in each cluster. For cluster R_{p_i} considered at ith position, the process to find sub-tree $T_{p_i} = (V_{T_{p_i}}, E_{T_{p_i}})$, considering the graph $G_{p_i} = (V_{p_i}, E_{p_i})$ containing:

- The set of vertices $V_{p_i} = R_{p_i} \cup F_{p_i}$, with the intermediate vertex set F_{p_i} consisting of intermediate vertices of G that have not been used for any previous sub-tree. The set F_{p_i} is defined as:

$$F_{p_i} = F \setminus (S_{p_1} \cup S_{p_2} \cup \cdots \cup S_{p_{i-1}}),$$

  where $F = V \setminus R$ is the intermediate vertex set of G, and S_j is the set of Steiner vertices in T_j.

- The set of edges $E_{p_i} \subset E$ consists of edges with both endpoints in V_{p_i}:

$$E_{p_i} = \{(u, v) \in E \mid u, v \in V_{p_i}\}.$$

Using the input of the graph G_{p_i} and the traversal order of target vertices in R_{p_i}, the PBS algorithm finds the sub-tree T_{p_i} for the cluster p_i. For example, with the individual shown in Figure 7, the process of finding sub-trees is performed sequentially for clusters R_1, R_3, and R_2. Among them, the process of finding the sub-tree T_1 is illustrated in Figure 9.

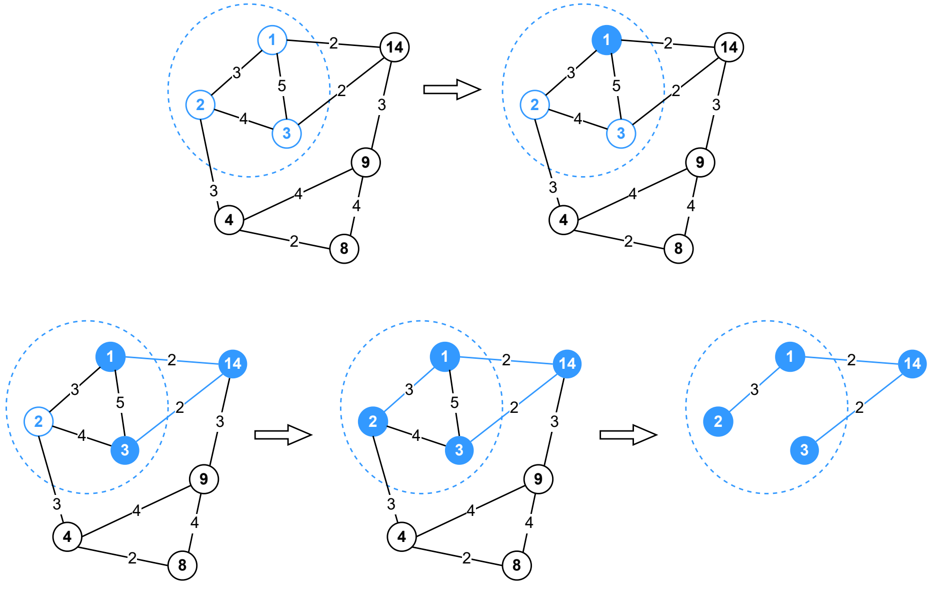

Figure 9. Finding the sub-tree of cluster 1 using PBS

Graph G1 consists of the set of target vertices R_1 and the intermediate vertex set F_1. Since cluster 1 is the first cluster to find a sub-tree, $F_1 = F$. According to the priority values of the vertices in R_1 in Figure 7, the traversal order of target vertices is 1, 3, and 2, with decreasing priorities of 0.7, 0.6, and 0.5, respectively. Tree T_1 is initialized with the starting vertex as 1. PBS adds vertex 3 to the tree with the shortest path being: 1, 14, 3. Finally, tree T_1 adds vertex 2 with the shortest path being the edge (1, 2).

Similarly, the sub-trees T_3 and T_2 found by the PBS algorithm are illustrated in Figures 10 and 11. Since tree T_1 has the Steiner vertex set $S_1 = \{14\}$, the graph G_3 does not contain vertex 14, and the set $F_3 = F \setminus S_1 = \{4, 8, 9\}$. Tree T_3 adds the target vertices in the order 11, 10, 13, and 12. Tree T_3 does not have any Steiner vertices, so the graph G_2 has the intermediate vertex set $F_2 = F_3 = \{4, 8, 9\}$. The PBS algorithm adds the target vertices to T_2 in the order 5, 7, and 6.

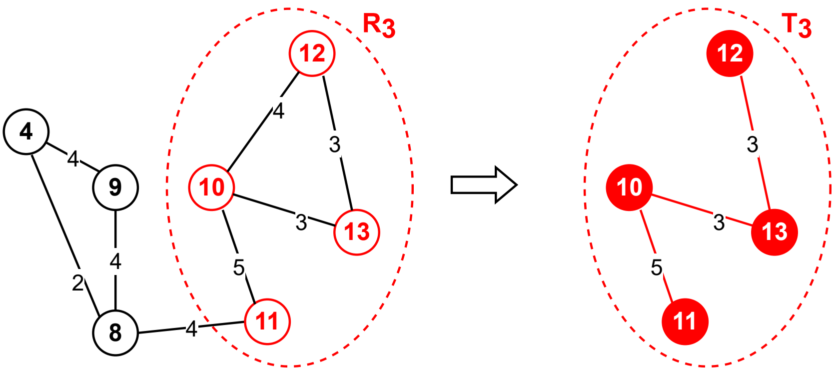

Figure 10. Sub-tree T_3

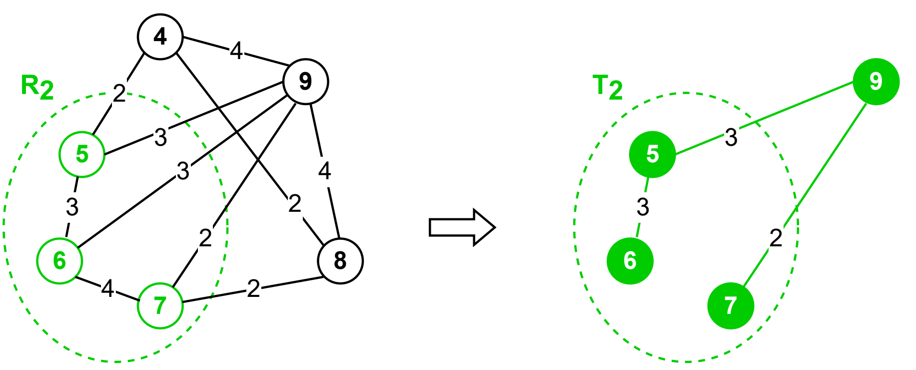

Figure 11. Sub-tree T_2

After finding the sub-trees, the total weight of the trees is calculated: $\alpha(T) = \sum_{i=1}^{k} c(T_i)$. In the example above, $\alpha(T) = c(T_1) + c(T_3) + c(T_2) = (3+2+2) + (5+3+3) + (3+2+3) = 26$.

4.3.5 Connecting the Sub-Trees of Each Cluster

After finding the sub-trees for each cluster $T_1, T_2, \ldots, T_k$, the graph G is reduced to $G' = (V', E', w')$, where the target vertex set is $R' = \{t_i \mid 1 \leq i \leq k\}$ corresponding to the sub-trees. The second level also utilizes the PBS algorithm with the vertex order determined based on the average priority of the clusters to find the tree T'. The edges of tree T' represent the connections between the sub-trees, and the cost of inter-cluster connections is denoted as $\beta(T) = c(T')$. For the sub-trees T_1, T_3, T_2 obtained in Section 4.3.4, the graph G' transformed from G is illustrated in Figures 12, 13, 14, 15, and 16. The PBS algorithm visits the target vertices in the order t_1, t_2, t_3 as determined in Figure 8.

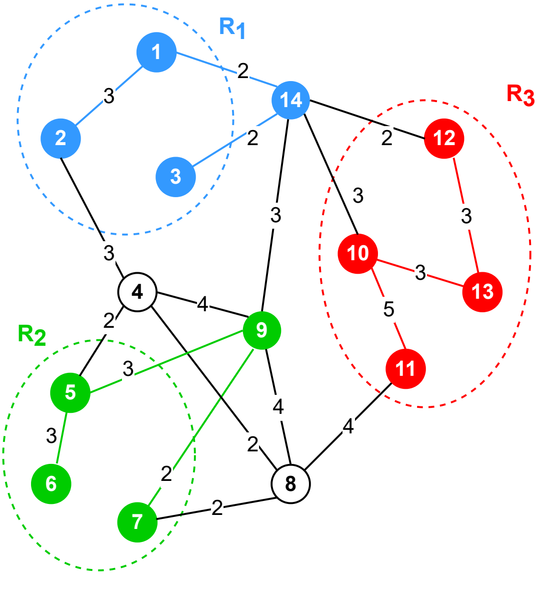

Figure 12. Graph G after finding sub-trees

The edges of the tree T' in Figure 15 are inter-cluster edges connecting the sub-trees, $\beta(T) = 3 + 2 = 5$. The tree T', along with the sub-trees T_1, T_3, T_2, forms the tree T, which is the complete solution to the problem. The total weight of the tree T is $c(T) = 26 + 5 = 31$.

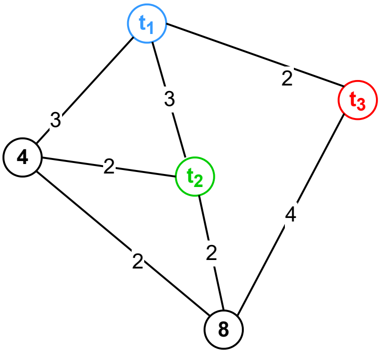

Figure 13. Graph G' reduced from G

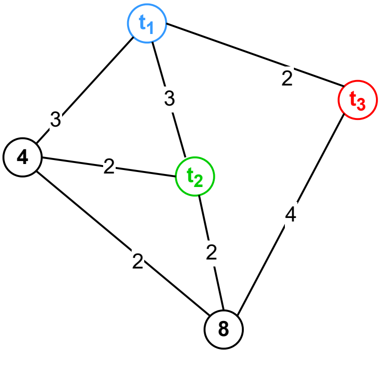

Figure 14. Level 2 input graph G'

4.4 Genetic Operator

4.4.1 Blended Crossover

In this study, the proposed algorithm utilizes the blend crossover method. Introduced in [21], Blend crossover (BLX) is commonly used with real-valued represen-

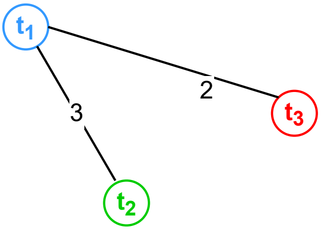

Figure 15. Inter-cluster tree T'

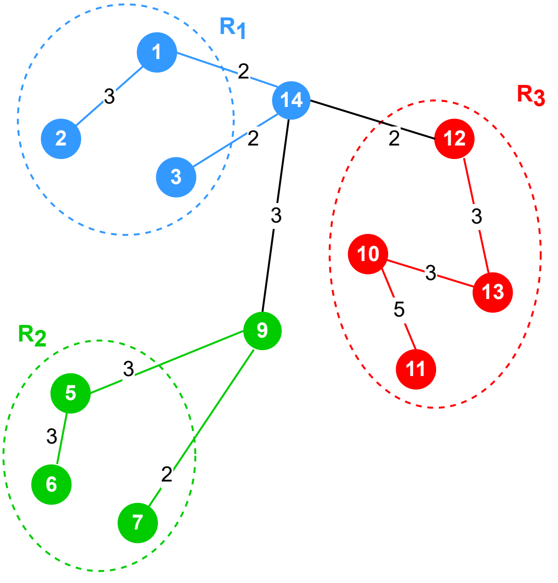

Figure 16. Cluster Steiner tree T

tations. For a pair of parent individuals $p_1, p_2 \in R^n$, the i^{th} element of the offspring individual c is randomly generated within the range $[P_{min} - I \times \alpha, P_{max} + I \times \alpha]$. Here, P_{min} is the minimum value between p_1^i and p_2^i, P_{max} is the maximum value between p_1^i and p_2^i, and I is calculated as $P_{max} - P_{min}$. The red area in Figure 17 illustrates the range of values for the offspring's element c^i generated from the parent individuals p_1 and p_2.

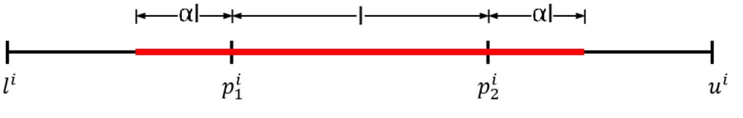

Figure 17. Range of values for the offspring's element generated from the parents p_1 and p_2 by BLX-α

In the case where the value range of the i^{th} element is $[l^i, u^i]$, BLX requires normalization of the value c^i that exceeds the value range: values greater than the range are reassigned to u^i, and values smaller than the range are reassigned to l^i.

4.4.2 Polynomial Mutation

The mutation operator used in the proposed algorithm is polynomial mutation. First introduced by Deb et al. [22, 23], polynomial mutation generates offspring c from parent $p \in R^n$ by:

- Generating a random value $u \sim U(0, 1)$.
- Using the polynomial distribution to generate parameters for the mutation process:
$$\begin{aligned} \delta_L &= (2u)^{\frac{1}{1+\eta_m}} - 1, && \text{for } u \leq 0.5, \\ \delta_U &= 1 - (2(1-u))^{\frac{1}{1+\eta_m}}, && \text{for } u > 0.5, \end{aligned} \tag{1}$$
where η_m is the mutation parameter.
- The value of element i of offspring c is calculated using the formula:
$$c^i = \begin{cases} p^i + \delta_L(p^i - l^i), & \text{for } u \leq 0.5, \\ p^i + \delta_U(u^i - p^i), & \text{for } u > 0.5, \end{cases} \tag{2}$$
where $[l^i, u^i]$ represents the value range of element i.

The values of offspring generated usually have slight differences from their parents.

4.4.3 Elitist Selection

Let P_i be the population of the current generation i, and C_i be the population consisting of offspring generated by crossover and mutation operators. Firstly, all the offspring individuals in C_i are evaluated. Then, the population of the next generation is selected as the top N fittest individuals among the combined population of P_i and C_i.

4.5 Time Complexity Analysis

The prefiltering step plays an important role in significantly reducing the computation time by precomputing the edge weights between the vertices of the graph G remaining constant in all solutions. These weights are related to the connections between destination vertices and Steiner vertices within each cluster, as well as between different clusters. By precomputing the weights before any evaluations are performed, the algorithm avoids recalculating them for each individual, thus reducing redundant computations. Specifically, the weight between an intermediate vertex u and a destination vertex in a cluster R_i, as well as the weights between vertices between clusters, are calculated and stored in advance. This step allows the algorithm to quickly reference the precomputed values during the evaluation process, improving efficiency. Without prefiltering step, recalculating these values can increase the time complexity. On the other hand, the PBS algorithm, used to find the Steiner tree for each cluster and the inter-cluster connections, also contributes to the overall time complexity. By processing vertices in a predefined order, PBS provides a more structured and efficient exploration of the solution space compared to other algorithms, which can get stuck in local optima. Effortless exploration improves the quality of solutions while maintaining a manageable computational cost.

The time complexity of PBGA is structured across three primary stages:

Graph Prefiltering: This stage calculates the minimum distances between free vertices and clusters, and between clusters. Its time complexity is $\mathcal{O}(|V|^2)$.

Population Initialization: The initial population of N individuals is generated using BSPH. The complexity for this step is $N \times \mathcal{O}(|R| \times |V| \times \log(|V|) + |R| \times |E|)$.

Offspring Reproduction: This iterative process occurs over GEN generations and includes crossover, mutation, cost evaluation, and selection.

- **Crossover:** Generating N new individuals has a time complexity of $\mathcal{O}(N \times |V|)$, as it operates on real-coded chromosomes of length $|V|$ for each individual.
- **Mutation:** For N individuals with a mutation rate p_m, the complexity is $\mathcal{O}(p_m \times N \times |V|)$, reflecting operations on chromosomes of length $|V|$.
- **Cost Evaluation (per individual):** This is a critical component, involving:
  - Building Steiner trees for each cluster using PBS: $\mathcal{O}(|R| \times |V| \times \log(|V|) + |R| \times |E|)$.
  - Constructing the intermediate graph G' from G: $\mathcal{O}(|V \setminus R|^2)$.
  - Finding inter-cluster connections on G' using PBS: $\mathcal{O}(|R| \times |V| \times \log(|V|) + |R| \times |E|)$.
  - The total complexity for evaluating one individual is approximately $\mathcal{O}(|R| \times |V|^2)$ (assuming dense graphs where $|E|$ is $\mathcal{O}(|V|^2)$), as the PBS and G' construction steps are dominant.
- **Selection:** The process of selecting N individuals for the next generation, typically has a complexity of $\mathcal{O}(N \log N)$.

In summary, the overall time complexity of PBGA is largely dominated by the iterative evaluation process. It can be expressed as: $\mathcal{O}(\text{PBGA}) = \mathcal{O}(|V|^2) + N \times \mathcal{O}(|R| \times |V| \times \log(|V|) + |R| \times |E|) + \text{GEN} \times (N \times \mathcal{O}(|V|) + p_m \times N \times \mathcal{O}(|V|) + N \times \mathcal{O}(|R| \times |V|^2) + \mathcal{O}(N \log N))$. The most prominent term contributing to the overall complexity is typically $\mathcal{O}(GEN \times N \times |R| \times |V|^2)$.

5 COMPUTATIONAL RESULTS

5.1 Problem Instances

The experiments use the only datasets published in [20]. There are seven datasets comprising 140 instances in total, categorized into two types regarding dimensionality: small instances, each of which has between 30 and 120 vertices, and large instances, each of which has over 260 vertices. Therefore, we have a variety of different classes of instances with varying sizes (different numbers of vertices, edges, and clusters). It allows us to evaluate the efficiency of the proposed algorithm in many scenarios. More details of the datasets are given in Table 2.

Type	**# NoIns**		**# Vertices**	**# Clusters**	**# ReqVertices**
Type_1_Small	27	Max	105	75	80
		Min	51	5	12
Type_5_Small	21	Max	120	10	28
		Min	30	5	8
Type_6_Small	37	Max	105	42	51
		Min	51	2	12
Type_1_Large	15	Max	242	50	110
		Min	262	10	49
Type_3_Large	10	Max	750	25	135
		Min	300	6	50
Type_5_Large	15	Max	500	25	96
		Min	300	5	49
Type_6_Large	15	Max	442	49	98
		Min	262	9	48

* NoIns: Number of instances, ReqVertices: Required Vertices

Table 2. Dataset information

5.2 Experiment Criteria

The following criteria assessed the quality of the algorithms:

AVG: The average objective function value over 30 runs.

BF: Best objective function value achieved over 30 runs

RPD: Relative Percentage Difference.

PI: Percentage of Improvement.

Let S^i_{ar} be the solution produced by algorithm a in r^{th} on instance i. Let B^i be the best solution among all algorithms, for instance, i. Then RPD value is calculated using the following equation. The smaller the RPD value, the better the quality of the solution found.

$$RPD^i_{ar} = \frac{S^i_{ar} - B^i}{B^i} \times 100\,\%.$$

The Improvement Percentage (PI) is used to signify the improvement of algorithm a over b. Let AVG_a^i and AVG_b^i be the average value over 30 runs on instance i of two algorithms a and b, respectively. Then the improvement of the algorithm over algorithm b is:

$$PI_{ab}^i = \frac{AVG_b^i - AVG_a^i}{AVG_b^i} \times 100\,\%.$$

5.3 Experimental Setting

Each algorithm is executed independently 30 times on each dataset. The computer configuration is Intel(R) Core(TM) i5-3470 CPU 3.2 GHz, 16 GB RAM. The algorithms are implemented in the Java language. The parameters of SPGA and PBGA are provided by Table 3.

Parameter	**Definition**	**Value**
POPSIZE	Number of individuals in the population	100
MAXGEN	Number of generations	500
MAXEVAL	Number of evaluations	50 000
p_m	Mutation rate	0.05
p_c	Crossover rate	0.9
n_m	Mutation parameter PM	15
α	BLX crossover parameter	0.3
MAPSIZE	Size of the SMS map	15

Table 3. Parameters of SPGA and PBGA

To ensure objectivity, the number of evaluations for the solutions of each algorithm must be the same. SPGA and PBGA have the same population size of 100 and the same number of generations of 500, which corresponds to 50 000 evaluations. The BSPH algorithm evaluates only one solution at a time, so it needs to be run 50 000 times (random starting vertex and subtree finding order) to find the best solution. In addition, these 50 000 runs share the preprocessed graph information in Section 4.1 to ensure fairness in terms of time with SPGA and PBGA.

For critical genetic operators, namely the crossover rate (p_c) and the mutation rate (p_m), a dedicated parameter tuning process was undertaken, as these parameters significantly influence the exploratory and exploitative capabilities of a genetic algorithm. The proposed PBGA was executed with various combinations of p_c and p_m values on 16 typical data instances, with each instance run 30 times to obtain the average of the best-found solutions. The objective of this tuning was to identify the parameter combination that consistently yielded the best average among the test runs, thereby optimizing the algorithm's performance. The detailed experimental results of this tuning process for p_c and p_m are presented in Table 4. The results in Table 4 (where a value of 5 indicates the most frequent occurrence of the best solution) indicate that p_c should be set to 0.9 and p_m to 0.05 for the main experiments.

This selection is consistent with general recommendations for mutation rates, which are typically small (e.g., 1–5 %) to avoid instability in the population [24].

Other parameters, including the polynomial mutation parameter (η_m), the blend crossover parameter (α), and the SMS map size (*MAPSIZE*), were set based on widely established practices in prior evolutionary computation research: $\eta_m = 15$ for Polynomial Mutation, a common operator for real-parameter representation, as introduced by Deb and Agrawal [22], $\alpha = 0.3$ for Blend Crossover (BLX), a frequently used crossover operator for real-parameter representation, introduced by Eshelman and Schaffer [21], and $MAPSIZE = 15$ for SMS, a selection strategy inspired by the Multi-dimensional Archive of Phenotypic Elites (MAP-Elites) algorithm. This size was chosen to ensure the maintenance of population diversity [25, 26, 27].

$P_m \backslash P_c$	0.80	0.85	0.90	0.95
0.05	1	2	5	1
0.10	3	1	2	1
0.15	2	3	1	3
0.20	2	2	1	1

Table 4. The number of best solution results on different pair parameters on some data sets

5.4 Experimental Scenarios

The performance of the proposed algorithm was compared with two other algorithms, BSPH and SPGA. There are two experiments as follows:

Experiment 1: Conduct non-parametric statistic tests to analyze the results of the algorithms.

Experiment 2: Compare results, run time, and convergence trend of difference algorithms.

5.5 Experimental Results

5.5.1 Non-Parametric Statistics to Compare the Results of the Proposed Algorithm and the Existing Algorithm

The detailed results obtained by all three algorithms are presented from Table 10 to Table 16. Statistical non-parametric tests are used to examine whether significant differences exist in the results of the BSPH, SPGA, and PBGA algorithms. The process of conducting non-parametric statistical analysis consists of two steps:

In the first step, the Friedman, Aligned Friedman, and Quade tests [28] are employed to evaluate the differences among the algorithm results.

In the second step, after confirming the differences among the algorithms, further post-hoc statistical tests are conducted to compare the best-performing algorithm (or the control algorithm) with the other two algorithms.

The results of the Friedman, Aligned Friedman, and Quade tests are presented in Table 5. The p-values for all tests are below the threshold of 0.05, indicating significant statistical differences in the obtained results among the algorithms.

Friedman Value 211.444		Aligned Friedman Value 90.666		Quade Value 221.894	
Value in X^2	p-value	Value in X^2	p-value	Value in F_F	p-value
5.991	$9.448 * 10^{-11}$	5.991	$8.546 * 10^{-11}$	3.029	$1.005 * 10^{-57}$

Table 5. The results of Friedman, Aligned Friedman and Quade statistical tests ($\alpha = 0.05$)

The average rankings of the algorithms are presented in Table 6. The results show that the proposed PBGA algorithm significantly outperforms BSPH and SPGA on all types of statistical tests.

Algorithm	Friedman	Friedman Aligned	Quade
BSPH	2.425	299.279	2.636
SPGA	2.004	202.967	1.977
PBGA	1.571	129.254	1.387

Table 6. The average rankings of the algorithms are computed using the Friedman, Aligned Friedman, and Quade statistical tests

In the second step, since PBGA has the lowest rank, it is chosen as the control algorithm for pairwise comparisons with the other two algorithms using the Holland and Holm statistical methods. The p-values in Table 7 are much smaller than the threshold $\alpha = 0.05$, indicating the superiority of PBGA over the other two algorithms.

i	Algorithm	$z = (R_0 - R_i)/SE$	p	Holm/Hochberg/ Hommel	Holland
2	BSPH	7.141	9.232E-13	0.025	0.025
1	SPGA	3.616	2.997E-4	0.05	0.05

Table 7. The z-values and p-values of the Friedman statistical test with the control algorithm PBGA

5.5.2 Detail of Comparison Among the Algorithms BSPH, SPGA and PBGA

This section provides a detailed comparison of the algorithms, including the results, running time, and convergence trend. The results from Table 10 to Table 16 are evaluated based on the criteria of average PI improvement on each dataset and the relative percentage difference (RPD).

	Dataset	PI Min	PI Average	PI Max	+/ ≈ /−
Small	Type_1_Small	−0.007416	0.362646	3.646409	6/19/2
	Type_5_Small	0.000000	0.202814	1.323529	5/16/0
	Type_6_Small	−0.040123	0.526952	10.554090	13/22/2
Large	Type_1_Large	−0.129874	2.208344	6.692581	14/0/1
	Type_3_Large	0.387083	3.330297	6.362738	10/0/0
	Type_5_Large	0.131776	2.391222	6.159098	15/0/0
	Type_6_Large	−0.159064	2.003998	4.675739	14/0/1

Table 8. PI (%) of PBGA compared with BSPH

	Dataset	PI Min	PI Average	PI Max	+/ ≈ /−
Small	Type_1_Small	−0.007416	0.362646	3.646409	6/19/2
	Type_5_Small	0.000000	0.202814	1.323529	5/16/0
	Type_6_Small	−0.040123	0.526952	10.554090	13/22/2
Large	Type_1_Large	−0.677928	0.370860	1.860520	9/0/6
	Type_3_Large	−0.167542	1.129364	2.608146	9/0/1
	Type_5_Large	−1.687057	0.529224	2.713064	12/0/3
	Type_6_Large	−0.639308	0.321106	1.886703	8/0/7

Table 9. PI (%) of PBGA compared with SPGA

The improvement in the average PI results of the PBGA algorithm compared to BSPH and SPGA is presented in Tables 8 and 9.

The BSPH and SPGA algorithms yield similar results on small datasets because they both use SPH to find Steiner trees, which does not fully explore the solution space. On the other hand, PBGA shows significant improvement, especially on the Type_6_Small dataset.

Although PBGA does not completely outperform the other algorithms on large datasets, it performs better on most samples. BSPH produces inferior results compared to PBGA on all samples, except for sample 50gil262 from the Type_1_Large dataset and 9a280-3x3 from the Type_6_Large dataset. While SPGA has better results on a few samples, PBGA overall proves to be more effective. PBGA demonstrates improvement compared to BSPH and SPGA, with an average PI improvement ranging from 0.20 to 3.33.

To evaluate the aspect of finding the best solutions, Figure 18 illustrates the distribution of RPD values for each algorithm.

PBGA achieves the lowest RPD values, around 2 %. Additionally, the consistency of the results across different runs and samples is evident from the short and closely located boxplots, indicating minimal variation.

To evaluate the running time, this study calculates the average running time in seconds of the algorithms on the samples of each dataset and illustrates it in Figures 19 and 20. On small datasets, PBGA has the lowest running time, but the difference is not significant. On large datasets, there is a noticeable difference

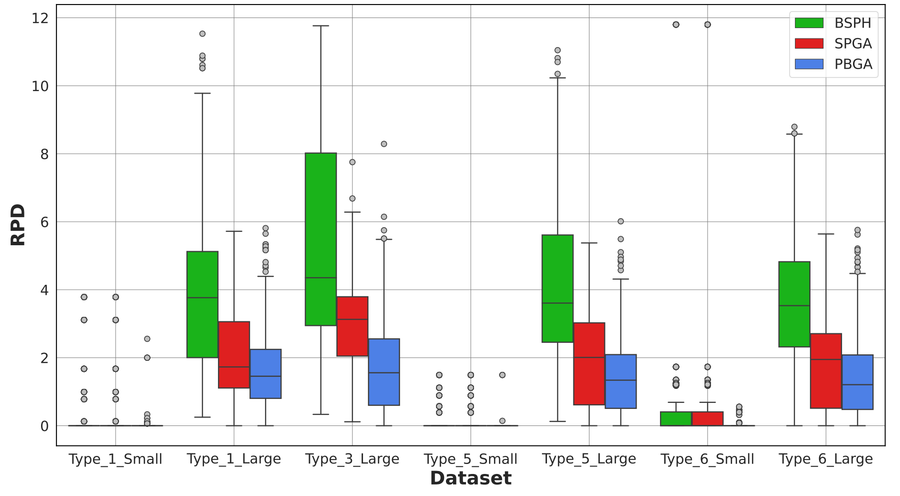

Figure 18. RPD values of PBGA compared to BSPH and SPGA

between the datasets. PBGA has the lowest running time on Type_1_Large and Type_6_Large but the highest on Type_3_Large and Type_5_Large.

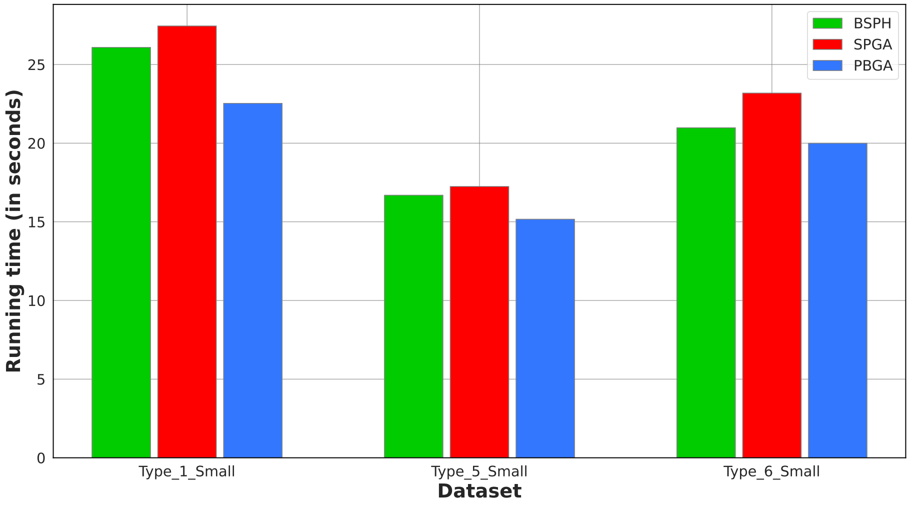

Figure 19. Average running time of the algorithms on small dataset BSPH, SPGA and PBGA

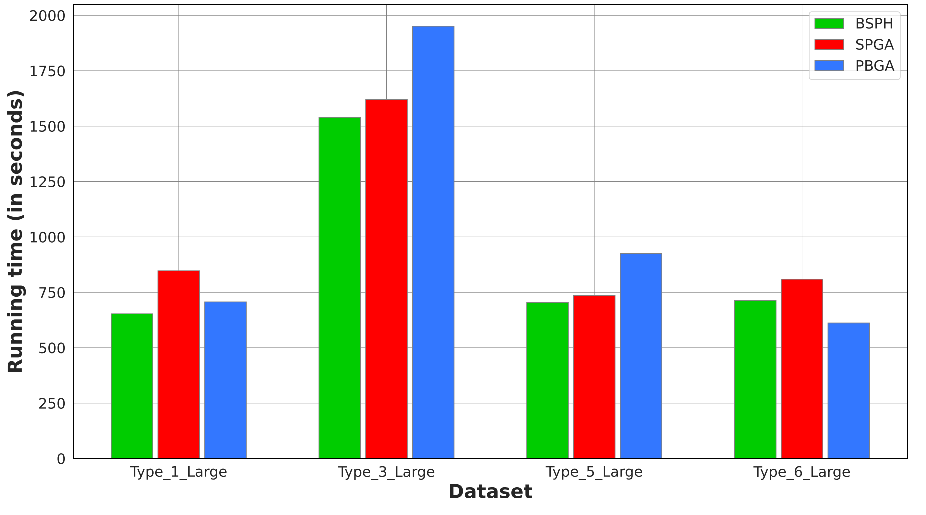

Figure 20. Average running time of the algorithms on large dataset BSPH, SPGA and PBGA

5.5.3 Convergence Trends and Time Running

As SPGA and PBGA tend to converge quickly within the first few generations on small datasets, this study focused on evaluating the convergence trends of these two algorithms on large datasets. The convergence trend of each dataset is assessed by calculating the average fitness value for each generation over 30 runs. The convergence trends of selected instances from different datasets are illustrated in Figures 21, 22, 23, and 24.

To analyze the convergence trend on the dataset, the convergence rate can be evaluated using the normalized average value of the dataset, based on the formula described in [29]. The process of calculating the normalized average value of algorithm A on dataset T is as follows:

- Compute the average solution value (based on 30 runs) for each generation on dataset i of T. The average solution value of algorithm A at generation t on dataset i is denoted as AVG^{i}_{At}.

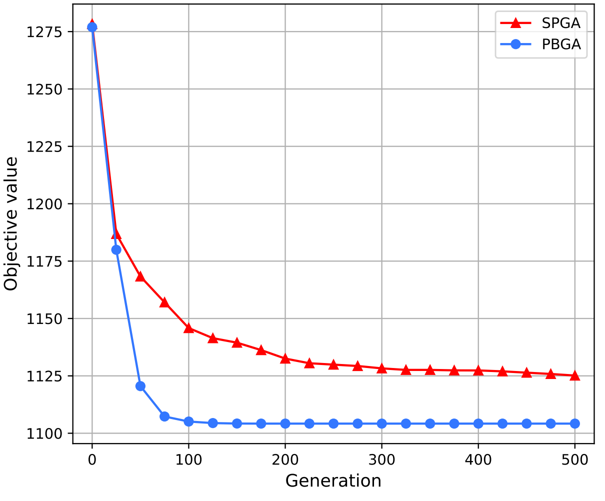

Figure 21. Convergence trends of SPGA and PBGA on Instance 50pr439 of Type_1_Large

- Calculate the normalized value for each dataset i. At generation t, the normalized value is computed using the following formula:

$$s^i_{At} = \frac{AVG^i_{At} - AVG^i_A}{AVG^i_{A0} - AVG^i_A}.$$

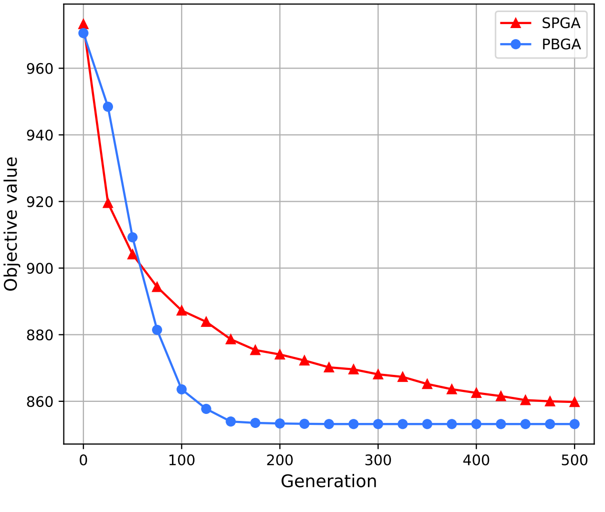

Figure 22. Convergence trends of SPGA and PBGA on Instance 25i750 of Type_3_Large

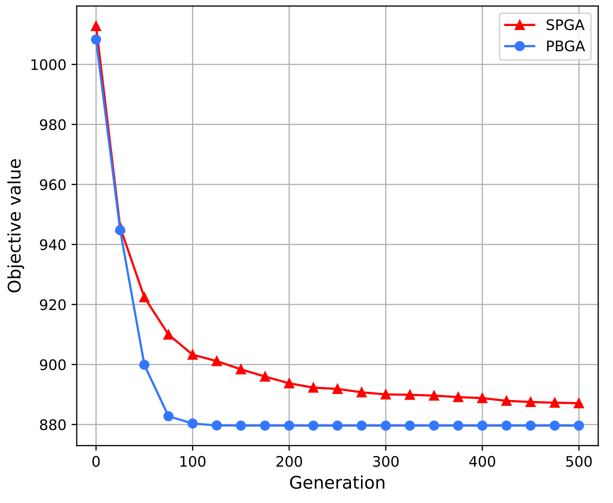

Figure 23. Convergence trends of SPGA and PBGA on Instance 25i500-308 of Type_5_Large

Here, AVG^i_{A0} represents the average solution value when initialized, and AVG^i_A represents the average solution value of algorithm A after 500 generations. In case there is no improvement compared to the initialization value ($AVG^i_{A0} = AVG^i_A$), we assume $s^i_{At} = 0$ for all t.

- Calculate the normalized value for the entire dataset T by taking the average of the normalized values across all dataset instances:

$$s^T_{At} = \frac{1}{|T|} \sum_{i \in T} s^i_{At}$$

the normalized value belongs to [0, 1].

Using this normalization approach, the convergence trend of SPGA and PBGA algorithms on several large datasets is illustrated in Figures 25, 26, 27, and 28.

The comparison between the two algorithms shows that PBGA tends to converge slower than SPGA in the first 50 generations, but it improves rapidly in the next 50 generations. Therefore, maintaining population diversity through tiered selection in PBGA helps ensure the exploration factor, surpassing SPGA in the 50–100 generation range. However, from generation 100 onwards, both algorithms gradually converged, and after generation 200, no significant improvement was observed.

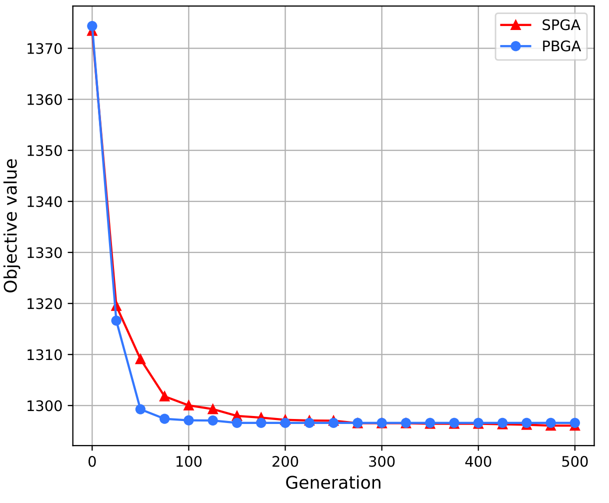

Figure 24. Convergence trends of SPGA and PBGA on Instance 49lin318-7x7 of Type_6_Large

5.6 Dicussions

Generally speaking, a good metaheuristic needs to balance exploration and exploitation capacity. The exploration characteristic means the search explores new promising solutions, while exploitation means the search effectively exploits the current solution space. Related to the problem, we found that only one metaheuristic SPGA was proposed to solve it in the literature, though it is an interesting problem.

The SPH method in SPGA is mainly based on a greedy approach to finding the Steiner tree. However, it is too greedy and does not support enough diversity to maintain the exploration capacity for SPGA. Therefore, SPGA can get trapped in the local optima. On the other hand, in finding the Steiner tree, the PBS in PBGA provides a promising list of candidate vertices to visit. In each step, we select a vertex from the list. Therefore, it brings randomness and greediness together. The size of the list controls the balance between greediness and randomness. This balance helps PBGA to maintain a diversity of the population. Additionally, the search is prevented from premature convergence in many cases. As a result, PBGA is better than SPGA in most cases in terms of solution quality and convergence trends.

6 CONCLUSION

This paper introduces a two-level Genetic Algorithm using Priority-Based Search to solve the CluSteiner problem with two contributions. First, we propose the PBS algorithm to find better Steiner trees, maintaining the exploration capacity. Second,

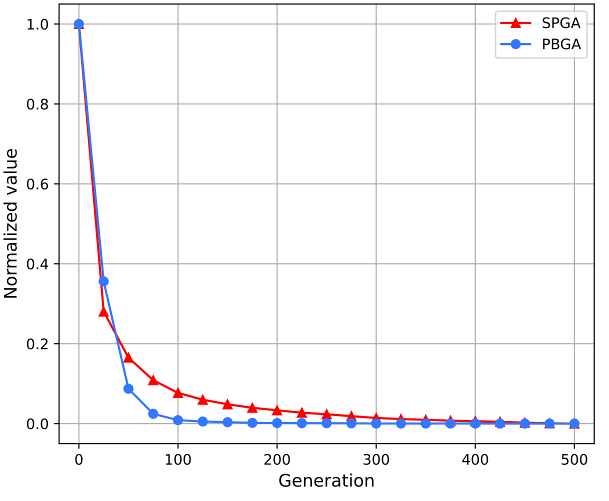

Figure 25. Convergence trends of SPGA and PBGA on Type_1_Large

we introduce the efficiency algorithm based on a genetic algorithm scheme with priority-based encoding. The algorithm has a good balance between exploration and exploitation. Its efficiency, in terms of solution quality, computational time, and convergence trends, was evaluated by extensive experiments. The results show

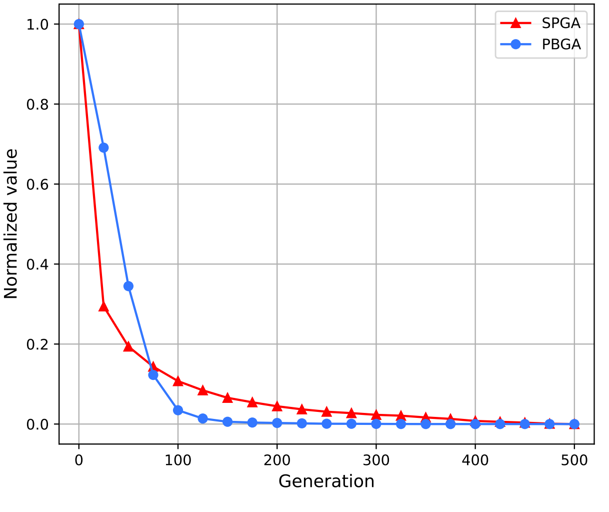

Figure 26. Convergence trends of SPGA and PBGA on Type_3_Large

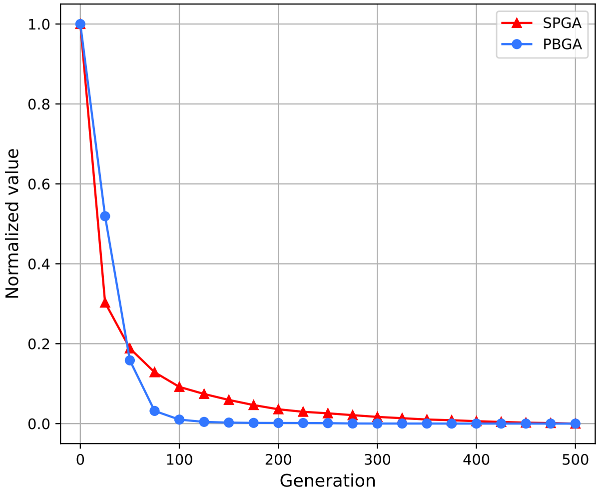

Figure 27. Convergence trends of SPGA and PBGA on Type_5_Large

that PBGA is superior to other existing algorithms for most cases. The new best solutions can be reached in many cases. However, the algorithm's time complexity may limit its scalability, especially for large, sparse, or non-metric graphs, and future work could focus on optimizing it through parallelization techniques. In addition,

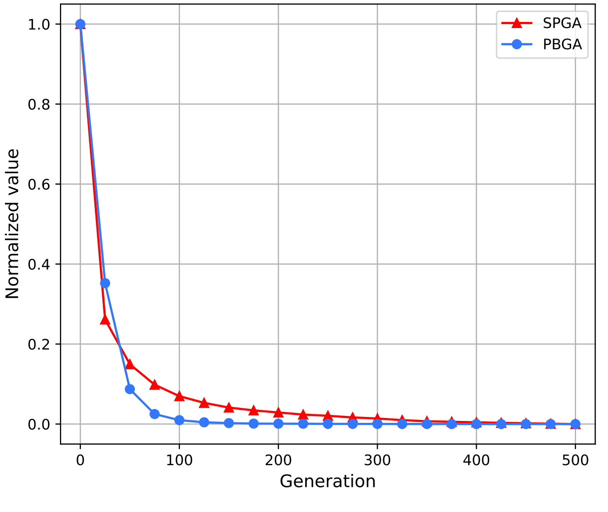

Figure 28. Convergence trends of SPGA and PBGA on Type_6_Large

the proposed algorithm can also be combined with other advanced optimization methods to improve the quality and efficiency of the solution, which could further enhance performance and reveal new areas for improvement.

7 DECLARATIONS

- Data and code are available upon request.
- Conflict of interest: The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.
- Authors' contributions:
  - Do Tuan Anh: Methodology, Algorithms, Coding, and Writing the manuscript.
  - Ha-Bang Ban: Methodology, Algorithms, Coding, Writing the manuscript.
  - Minh Tu Le: Methodology, Algorithms, Coding, Writing the manuscript.
  - Pham Dang Hai: Coding and writing the manuscript.

Acknowledgments

This research was funded by the Hanoi University of Science and Technology under project code T2024-PC-038.

REFERENCES

[1] Chisman, J. A.: The Clustered Traveling Salesman Problem. Computers & Operations Research, Vol. 2, 1975, No. 2, pp. 115–119, doi: 10.1016/0305-0548(75)90015-5.

[2] Lin, C. W.—Wu, B. Y.: On the Minimum Routing Cost Clustered Tree Problem. Journal of Combinatorial Optimization, Vol. 33, 2017, No. 3, pp. 1106–1121, doi: 10.1007/s10878-016-0026-8.

[3] D'Emidio, M.—Forlizzi, L.—Frigioni, D.—Leucci, S.—Proietti, G.: On the Clustered Shortest-Path Tree Problem. In: Bilò, V., Caruso, A. (Eds.): Proceedings of the 17th Italian Conference on Theoretical Computer Science (ICTCS 2016). CEUR Workshop Proceedings, Vol. 1720, 2016, pp. 263–268, https://ceur-ws.org/Vol-1720/short8.pdf.

[4] Seah, W. K. G.—Er, I. I.: Distributed Steiner-Like Multicast Path Setup for Mesh-Based Multicast Routing in Ad Hoc Networks. Vol. 2, 2006, pp. 192–197, doi: 10.1109/SUTC.2006.59.

[5] Li, D.—Jia, X.—Liu, H.: Energy Efficient Broadcast Routing in Static Ad Hoc Wireless Networks. IEEE Transactions on Mobile Computing, Vol. 3, 2004, No. 2, pp. 144–151, doi: 10.1109/TMC.2004.10.

Instance	BSPH				SPGA				PBGA			
	BF	Avg	Std	Time	BF	Avg	Std	Time	BF	Avg	Std	Time
5eil51	689.000	689.000	0.000	9.523	689.000	689.000	0.000	10.147	689.000	689.000	0.000	8.333
5berlin52	972.000	972.000	0.000	9.524	972.000	972.000	0.000	9.972	956.000	956.000	0.000	8.909
5st70	451.000	451.000	0.000	13.628	451.000	451.000	0.000	14.187	451.000	451.000	0.000	11.973
5pr76	630.000	630.000	0.000	18.791	630.000	630.000	0.000	19.474	630.000	630.000	0.000	16.039
5eil76	527.000	527.000	0.000	18.574	527.000	527.000	0.000	19.440	527.000	527.000	0.000	15.631
10eil51	887.000	887.000	0.000	9.551	887.000	887.000	0.000	10.957	887.000	887.000	0.000	8.916
10berlin52	786.000	786.000	0.000	9.571	786.000	786.000	0.000	10.558	786.000	786.000	0.000	8.777
10st70	899.000	899.000	0.000	17.447	899.000	899.000	0.000	17.938	899.000	899.067	0.359	14.659
10pr76	900.000	900.000	0.000	19.909	900.000	900.000	0.000	20.148	900.000	900.000	0.000	17.392
10eil76	905.000	905.000	0.000	22.545	905.000	905.000	0.000	24.287	872.000	872.000	0.000	19.620
10rat99	928.000	928.000	0.000	34.173	928.000	928.000	0.000	35.285	900.000	903.500	7.013	28.569
10kroB100	676.000	676.000	0.000	34.118	676.000	676.000	0.000	35.702	676.000	676.000	0.000	29.130
15eil51	886.000	886.000	0.000	9.154	886.000	886.000	0.000	9.477	886.000	886.000	0.000	8.315
15berlin52	1 120.000	1 120.000	0.000	9.071	1 120.000	1 120.000	0.000	9.853	1 109.000	1 109.000	0.000	8.761
15st70	1 071.000	1 071.000	0.000	19.199	1 071.000	1 071.000	0.000	20.396	1 071.000	1 071.000	0.000	16.545
15eil76	929.000	929.000	0.000	22.364	929.000	929.000	0.000	23.604	929.000	929.000	0.000	19.145
15pr76	961.000	961.000	0.000	22.119	961.000	961.000	0.000	22.989	961.000	961.000	0.000	19.148
25rat99	1 218.000	1 218.000	0.000	36.696	1 218.000	1 218.000	0.000	39.354	1 218.000	1 218.000	0.000	31.182
25kroA100	1 418.000	1 418.000	0.000	42.044	1 418.000	1 418.000	0.000	45.116	1 418.000	1 418.000	0.000	35.031
25eil101	1 161.000	1 161.000	0.000	40.740	1 161.000	1 161.000	0.000	43.643	1 152.000	1 152.000	0.000	35.590
25lin105	884.000	884.000	0.000	40.029	884.000	884.000	0.000	41.857	884.000	884.000	0.000	33.909
50rat99	1 313.000	1 313.000	0.000	38.384	1 313.000	1 313.000	0.000	39.082	1 313.000	1 313.000	0.000	34.239
50kroB100	1 383.000	1 383.000	0.000	37.232	1 383.000	1 383.000	0.000	38.999	1 383.000	1 383.000	0.000	32.344
50kroA100	1 347.000	1 347.000	0.000	38.107	1 347.000	1 347.000	0.000	40.633	1 347.000	1 347.033	0.180	32.455
50eil101	1 354.000	1 354.000	0.000	38.360	1 354.000	1 354.000	0.000	40.310	1 354.000	1 354.000	0.000	33.177
50lin105	1 556.000	1 556.000	0.000	44.693	1 556.000	1 556.000	0.000	47.203	1 554.000	1 554.267	0.512	39.274
75lin105	1 386.000	1 386.000	0.000	48.740	1 386.000	1 386.000	0.000	50.486	1 386.000	1 386.000	0.000	41.193

Table 10. Results of CluSteiner problem obtained by all algorithms on Type_1_Small

Instance	BSPH				SPGA				PBGA			
	BF	Avg	Std	Time	BF	Avg	Std	Time	BF	Avg	Std	Time
10gil262	764.000	766.167	1.293	226.052	760.000	763.533	1.821	220.910	753.000	755.933	3.224	270.766
10a280	740.000	750.500	3.170	266.693	733.000	737.800	3.807	251.925	725.000	734.233	4.310	332.574
10lin318	706.000	707.933	2.476	343.999	705.000	708.667	3.004	331.648	694.000	701.300	5.336	440.206
10pr439	602.000	609.033	3.610	670.125	600.000	603.967	2.689	801.906	584.000	595.800	5.224	923.730
10pcb442	638.000	651.100	4.784	725.247	629.000	637.967	4.317	872.675	621.000	634.033	5.660	1 070.496
25gil262	954.000	982.067	11.296	246.667	934.000	952.167	9.285	277.969	924.000	945.000	10.405	308.220
25a280	989.000	997.833	7.299	300.465	956.000	982.367	11.502	350.827	961.000	987.567	12.013	385.474
25lin318	932.000	944.533	5.402	389.120	921.000	930.500	4.113	454.159	922.000	932.533	3.801	525.866
25pr439	847.000	860.567	6.525	912.211	829.000	837.033	6.595	2 183.217	822.000	836.133	7.168	1 176.142
25pcb442	945.000	968.067	8.334	940.570	918.000	935.567	10.978	2 364.059	910.000	929.967	12.742	1 237.157
50gil262	1 198.000	1 206.300	6.283	273.347	1 195.000	1 199.733	4.589	307.166	1 198.000	1 207.867	6.883	325.088
50a280	1 164.000	1 176.300	4.706	306.116	1 156.000	1 161.100	4.150	344.403	1 156.000	1 162.633	4.207	355.754
50lin318	1 164.000	1 173.867	5.731	801.931	1 148.000	1 161.333	6.518	476.622	1 155.000	1 167.733	9.729	530.285
50pr439	1 157.000	1 183.400	13.819	2 302.168	1 100.000	1 125.133	12.865	1 093.487	1 084.000	1 104.200	13.154	1 325.603
50pcb442	1 211.000	1 225.833	6.039	1 089.128	1 172.000	1 189.967	8.292	2 375.286	1 179.000	1 191.567	5.506	1 391.649

Table 11. Results of CluSteiner problem obtained by all algorithms on Type_1_Large

Instance	**BSPH**				**SPGA**				**PBGA**			
	BF	Avg	Std	Time	BF	Avg	Std	Time	BF	Avg	Std	Time
6i300	602.000	602.800	0.872	230.866	602.000	602.000	0.000	241.642	600.000	600.467	0.884	314.177
6i350	664.000	664.167	0.522	341.509	664.000	664.267	0.998	373.064	645.000	652.633	3.582	493.518
6i400	571.000	574.967	4.393	474.806	571.000	580.233	3.630	498.655	561.000	565.100	3.187	685.710
6i450	538.000	538.200	0.600	561.583	538.000	539.133	1.893	580.407	521.000	531.367	4.771	885.480
6i500	530.000	533.133	1.454	786.500	531.000	533.833	0.582	862.624	512.000	520.367	5.498	1 150.292
20i550	872.000	885.033	6.199	1 771.105	834.000	855.233	8.682	1 757.557	832.000	848.367	9.207	2 124.418
20i600	898.000	914.200	8.252	2 026.978	863.000	875.400	6.591	2 233.854	862.000	876.867	8.172	2 712.777
20i650	811.000	824.067	5.859	2 312.930	762.000	782.200	8.953	2 706.054	748.000	774.900	12.924	2 964.807
20i700	833.000	846.167	7.267	2 814.103	784.000	804.733	8.266	3 068.840	781.000	801.800	12.316	3 534.884
25i750	898.000	911.033	6.221	4 078.662	841.000	859.800	10.750	3 880.965	837.000	853.067	11.219	4 644.745

Table 12. Results of CluSteiner problem obtained by all algorithms on Type_3_Large

Instance	**BSPH**				**SPGA**				**PBGA**			
	BF	Avg	Std	Time	BF	Avg	Std	Time	BF	Avg	Std	Time
5i30-17	765.000	765.000	0.000	2.911	765.000	765.000	0.000	3.061	765.000	765.000	0.000	2.936
5i45-18	469.000	469.000	0.000	6.352	469.000	469.000	0.000	6.753	469.000	469.000	0.000	5.685
5i60-21	785.000	785.000	0.000	12.361	785.000	785.000	0.000	13.157	785.000	785.000	0.000	11.644
5i65-21	547.000	547.000	0.000	12.533	547.000	547.000	0.000	13.368	547.000	547.000	0.000	11.351
5i70-21	618.000	618.000	0.000	18.195	618.000	618.000	0.000	18.965	618.000	618.000	0.000	14.992
5i75-22	577.000	577.000	0.000	18.285	577.000	577.000	0.000	19.256	577.000	577.000	0.000	16.711
5i90-33	944.000	944.000	0.000	31.419	944.000	944.000	0.000	31.732	944.000	944.000	0.000	26.969
5i120-46	612.000	612.000	0.000	34.303	612.000	612.000	0.000	36.125	603.000	603.900	2.700	45.014
7i30-17	905.000	905.000	0.000	3.348	905.000	905.000	0.000	3.539	895.000	895.000	0.000	3.292
7i45-18	747.000	747.000	0.000	6.543	747.000	747.000	0.000	6.725	747.000	747.000	0.000	6.196
7i60-21	612.000	612.000	0.000	12.527	612.000	612.000	0.000	12.756	612.000	612.000	0.000	10.973
7i65-21	989.000	989.000	0.000	15.132	989.000	989.000	0.000	16.227	989.000	989.000	0.000	13.227
7i70-21	705.000	705.000	0.000	16.037	705.000	705.000	0.000	16.219	701.000	701.033	0.180	13.901
10i30-17	567.000	567.000	0.000	3.156	567.000	567.000	0.000	3.506	562.000	562.000	0.000	3.466
10i45-18	449.000	449.000	0.000	5.662	449.000	449.000	0.000	6.067	449.000	449.000	0.000	5.716
10i60-21	777.000	777.000	0.000	13.012	777.000	777.000	0.000	13.394	774.000	774.000	0.000	11.158
10i65-21	750.000	750.000	0.000	14.096	750.000	750.000	0.000	14.662	750.000	750.000	0.000	12.492
10i70-21	800.000	800.000	0.000	16.119	800.000	800.000	0.000	17.304	800.000	800.000	0.000	14.324
10i75-22	778.000	778.000	0.000	20.942	778.000	778.000	0.000	21.924	778.000	778.000	0.000	17.562
10i90-33	831.000	831.000	0.000	29.770	831.000	831.000	0.000	31.200	831.000	831.000	0.000	25.142
10i120-46	844.000	844.000	0.000	57.763	844.000	844.000	0.000	56.198	844.000	844.000	0.000	45.666

Table 13. Results of CluSteiner problem obtained by all algorithms on Type_5_Small

Instance	BSPH				SPGA				PBGA			
	BF	Avg	Std	Time	BF	Avg	Std	Time	BF	Avg	Std	Time
5i300-108	556.000	556.500	0.719	204.639	556.000	556.333	1.795	197.400	554.000	555.767	2.667	301.670
5i400-205	572.000	572.300	0.586	445.520	572.000	572.600	0.757	439.493	554.000	562.100	4.693	638.446
5i500-304	456.000	458.467	1.310	623.949	456.000	457.100	1.325	719.214	448.000	451.800	2.857	1 002.451
10i300-109	644.000	646.167	1.098	265.457	643.000	644.367	0.547	261.202	643.000	643.567	1.309	380.307
10i400-206	641.000	646.967	2.575	527.709	639.000	643.567	2.616	501.080	630.000	639.433	4.709	786.949
10i500-305	673.000	686.200	4.445	1 013.052	673.000	678.200	2.868	961.736	651.000	659.800	5.540	1 433.476
15i300-110	794.000	814.900	7.213	339.357	794.000	802.467	8.196	363.863	793.000	802.833	7.118	431.191
15i400-207	883.000	898.367	6.036	671.879	869.000	882.167	4.852	761.643	860.000	873.700	5.962	988.147
15i500-306	754.000	762.967	4.270	1 118.036	739.000	747.967	5.010	1 212.155	725.000	742.533	8.582	1 517.571
20i300-111	989.000	1 000.733	6.976	392.597	979.000	984.533	4.129	410.548	966.000	979.500	5.117	474.609
20i400-208	975.000	991.333	7.162	851.535	940.000	953.733	11.281	901.060	949.000	957.533	5.560	1 047.011
20i500-307	789.000	810.400	7.468	1 286.546	765.000	776.500	8.433	1 226.162	769.000	789.600	9.660	1 634.027
25i300-112	994.000	1 003.067	4.396	404.833	981.000	984.567	3.084	441.376	979.000	983.867	3.896	494.045
25i400-209	998.000	1 024.867	9.113	884.646	968.000	987.567	9.514	888.959	970.000	983.767	9.397	971.879
25i500-308	917.000	937.367	10.071	1 533.057	870.000	887.100	9.137	1 758.053	860.000	879.633	9.290	1 784.166

Table 14. Results of CluSteiner problem obtained by all algorithms on Type_5_Large

Instance	BSPH				SPGA				PBGA			
	BF	Avg	Std	Time	BF	Avg	Std	Time	BF	Avg	Std	Time
2lin105-2x1	439.000	439.000	0.000	26.954	439.000	439.000	0.000	29.501	436.000	436.000	0.000	24.133
4eil51-2x2	750.000	750.000	0.000	7.749	750.000	750.000	0.000	8.657	740.000	740.000	0.000	6.644
4berlin52-2x2	646.000	646.000	0.000	8.767	646.000	646.000	0.000	9.616	638.000	638.000	0.000	7.569
4pr76-2x2	491.000	491.000	0.000	16.098	491.000	491.000	0.000	17.256	491.000	491.000	0.000	12.786
4eil76-2x2	407.000	407.000	0.000	19.663	407.000	407.000	0.000	20.870	407.000	407.000	0.000	15.051
6berlin52-2x3	696.000	696.000	0.000	9.757	696.000	696.000	0.000	10.549	696.000	696.000	0.000	8.686
6st70-2x3	746.000	746.000	0.000	18.106	746.000	746.000	0.000	19.629	746.000	746.000	0.000	15.213
6pr76-2x3	758.000	758.000	0.000	21.593	758.000	758.000	0.000	24.766	678.000	678.000	0.000	17.989
8berlin52-2x4	823.000	823.000	0.000	9.694	823.000	823.000	0.000	11.019	809.000	809.000	0.000	9.014
9eil51-3x3	863.000	863.000	0.000	9.583	863.000	863.000	0.000	10.671	863.000	863.000	0.000	8.893
9st70-3x3	845.000	845.000	0.000	17.869	845.000	845.000	0.000	18.900	845.000	845.000	0.000	15.774
9pr76-3x3	900.000	900.000	0.000	19.725	900.000	900.000	0.000	21.849	900.000	900.000	0.000	17.612
9eil76-3x3	857.000	857.000	0.000	20.996	857.000	857.000	0.000	23.650	847.000	847.000	0.000	19.549
9eil101-3x3	891.000	891.000	0.000	37.839	891.000	891.000	0.000	29.242	890.000	890.000	0.000	31.937
10berlin52-2x5	972.000	972.000	0.000	10.126	972.000	972.000	0.000	11.679	971.000	971.000	0.000	9.601
12eil51-3x4	742.000	742.000	0.000	8.468	742.000	742.000	0.000	9.381	742.000	742.000	0.000	8.448
12st70-3x4	835.000	835.000	0.000	17.556	835.000	835.000	0.000	19.131	825.000	825.000	0.000	15.834
12eil76-3x4	740.000	740.000	0.000	18.637	740.000	740.000	0.000	20.807	740.000	740.000	0.000	17.904
12pr76-3x4	614.000	614.000	0.000	18.950	614.000	614.000	0.000	20.746	614.000	614.000	0.000	17.162
15pr76-3x5	695.000	695.000	0.000	19.358	695.000	695.000	0.000	21.545	695.000	695.000	0.000	18.655
16eil51-4x4	555.000	555.000	0.000	9.083	555.000	555.000	0.000	10.100	555.000	555.000	0.000	9.224
16st70-4x4	909.000	909.000	0.000	17.007	909.000	909.000	0.000	18.855	909.000	909.000	0.000	16.336
16eil76-4x4	771.000	771.000	0.000	19.601	771.000	771.000	0.000	22.516	771.000	771.000	0.000	18.649
16lin105-4x4	1 093.000	1 093.000	0.000	40.789	1 093.000	1 093.000	0.000	46.250	1 091.000	1 091.000	0.000	39.024
18pr76-3x6	1 161.000	1 161.000	0.000	21.194	1 161.000	1 161.000	0.000	23.451	1 161.000	1 161.000	0.000	20.586
20eil51-4x5	875.000	875.000	0.000	9.859	875.000	875.000	0.000	10.763	875.000	875.000	0.000	10.054
20st70-4x5	1 325.000	1 325.000	0.000	18.776	1 325.000	1 325.000	0.000	20.045	1 325.000	1 325.000	0.000	17.722
20eil76-4x5	965.000	965.000	0.000	19.866	965.000	965.000	0.000	22.626	959.000	959.000	0.000	20.356
25eil51-5x5	988.000	988.000	0.000	10.741	988.000	988.000	0.000	11.683	984.000	984.000	0.000	11.566
25eil76-5x5	1 089.000	1 089.000	0.000	20.756	1 089.000	1 089.000	0.000	22.033	1 089.000	1 089.000	0.000	20.548
25rat99-5x5	961.000	961.000	0.000	34.351	961.000	961.000	0.000	38.416	961.000	961.000	0.000	33.137
25eil101-5x5	1 320.000	1 320.000	0.000	41.285	1 320.000	1 320.000	0.000	45.870	1 320.000	1 320.167	0.373	39.315
28kroA100-4x7	1 035.000	1 035.000	0.000	36.373	1 035.000	1 035.000	0.000	39.901	1 035.000	1 035.000	0.000	33.606
30kroB100-5x6	1 080.000	1 080.000	0.000	35.467	1 080.000	1 080.000	0.000	39.222	1 079.000	1 080.433	1.647	37.173
35kroB100-5x5	1 179.000	1 179.000	0.000	29.370	1 179.000	1 179.000	0.000	43.485	1 179.000	1 179.000	0.000	36.382
36eil101-6x6	1 063.000	1 063.000	0.000	38.014	1 063.000	1 063.000	0.000	42.763	1 063.000	1 063.000	0.000	38.060
42rat99-6x7	1 264.000	1 264.000	0.000	36.137	1 264.000	1 264.000	0.000	40.279	1 257.000	1 260.833	2.282	39.521

Table 15. Results of CluSteiner problem obtained by all algorithms on Type_6_Small

Instance	BSPH				SPGA				PBGA			
	BF	Avg	Std	Time	BF	Avg	Std	Time	BF	Avg	Std	Time
9gil262-3x3	704.000	705.867	0.499	181.501	704.000	704.933	0.998	213.229	690.000	691.633	1.888	230.943
9a280-3x3	709.000	712.500	2.419	279.955	709.000	709.100	0.539	637.465	708.000	713.633	3.430	308.500
9lin318-3x3	742.000	757.067	5.709	315.022	741.000	750.467	5.252	360.749	729.000	751.500	10.161	369.540
9pr439-3x3	548.000	549.667	1.535	612.453	548.000	548.267	1.123	724.352	547.000	548.433	1.407	705.782
9pcb442-3x3	575.000	580.433	2.276	689.842	573.000	574.867	1.284	827.364	564.000	573.433	3.757	803.849
18pr439-3x6	692.000	705.633	4.593	1 528.109	682.000	687.200	3.070	1 389.903	670.000	685.433	6.893	751.443
20pr439-4x5	674.000	681.867	3.956	1 585.656	668.000	670.033	1.906	674.259	652.000	660.200	4.942	784.013
25gil262-5x5	925.000	935.433	4.544	259.140	906.000	913.767	7.986	250.576	906.000	915.367	5.023	258.988
25a280-5x5	986.000	998.600	6.092	318.408	973.000	990.933	5.452	647.201	960.000	978.133	12.290	335.640
25lin318-5x5	998.000	1009.233	5.207	426.355	977.000	985.900	8.158	418.245	972.000	981.533	6.951	479.685
25pcb442-5x5	894.000	912.067	7.398	2 264.957	856.000	877.300	10.264	2 305.040	851.000	874.800	11.912	1 334.744
36pcb442-6x6	1 075.000	1 107.133	11.829	1 123.131	1 038.000	1 065.100	11.250	2 482.785	1035.000	1055.367	9.499	1 541.803
42a280-6x7	1 106.000	1 126.900	9.278	308.694	1 080.000	1 091.267	8.668	337.389	1 078.000	1 098.200	13.265	372.472
49gil262-7x7	1 159.000	1 167.533	5.696	368.672	1 159.000	1 161.500	4.272	397.933	1 159.000	1 165.200	8.064	338.239
49lin318-7x7	1 304.000	1 317.000	6.377	423.700	1 284.000	1 296.067	5.994	474.788	1 284.000	1 296.600	4.930	558.123

Table 16. Results of CluSteiner problem obtained by all algorithms on Type_6_Large

[6] Cao, J.—Guo, C.—Lu, G.—Xiong, Y.—Zheng, Y.—Zhang, Y.—Zhu, Y.—Chen, C.—Tian, Y.: Datacast: A Scalable and Efficient Reliable Group Data Delivery Service for Data Centers. IEEE Journal on Selected Areas in Communications, Vol. 31, 2013, No. 12, pp. 2632–2645, doi: 10.1109/JSAC.2013.131205.

[7] Ding, C.—Cheng, Y.—He, M.: Two-Level Genetic Algorithm for Clustered Traveling Salesman Problem with Application in Large-Scale TSPs. Tsinghua Science & Technology, Vol. 12, 2007, No. 4, pp. 459–465, doi: 10.1016/S1007-0214(07)70068-8.

[8] Potvin, J. Y.—Guertin, F.: The Clustered Traveling Salesman Problem: A Genetic Approach. In: Osman, I. H., Kelly, J. P. (Eds.): Meta-Heuristics. Springer, Boston, MA, 1996, pp. 619–631, doi: 10.1007/978-1-4613-1361-8_37.

[9] Laporte, G.—Potvin, J. Y.—Quilleret, F.: A Tabu Search Heuristic Using Genetic Diversification for the Clustered Traveling Salesman Problem. Journal of Heuristics, Vol. 2, 1997, No. 3, pp. 187–200, doi: 10.1007/BF00127356.

[10] Thang, T. B.—Long, N. B.—Hoang, N. V.—Binh, H. T. T.: Adaptive Knowledge Transfer in Multifactorial Evolutionary Algorithm for the Clustered Minimum Routing Cost Problem. Applied Soft Computing, Vol. 105, 2021, Art. No. 107253, doi: 10.1016/j.asoc.2021.107253.

[11] Lin, C. W.—Wu, B. Y.: A 2-Approximation Algorithm for the Clustered Minimum Routing Cost Tree Problem. In: Chu, W. C. C., Chao, H. C., Yang, S. J. H. (Eds.): Intelligent Systems and Applications (ICS 2014). IOS Press, Frontiers in Artificial Intelligence and Applications, Vol. 274, 2015, pp. 3–10, doi: 10.3233/978-1-61499-484-8-3.

[12] Ban, H. B.: The Hybridization of ACO + GA and RVNS Algorithm for Solving the Time-Dependent Traveling Salesman Problem. Evolutionary Intelligence, Vol. 15, 2022, No. 1, pp. 309–328, doi: 10.1007/s12065-020-00510-9.

[13] Ban, H. B.: Multifactorial Evolutionary Algorithm for Simultaneous Solution of TSP and TRP. Computing and Informatics, Vol. 40, 2021, No. 6, pp. 1370–1397, doi: 10.31577/cai_2021_6_1370.

[14] Garey, M. R.—Graham, R. L.—Johnson, D. S.: The Complexity of Computing Steiner Minimal Trees. SIAM Journal on Applied Mathematics, Vol. 32, 1977, No. 4, pp. 835–859, doi: 10.1137/0132072.

[15] Wu, B. Y.—Lin, C. W.: On the Clustered Steiner Tree Problem. Journal of Combinatorial Optimization, Vol. 30, 2015, No. 2, pp. 370–386.

[16] Chen, Y. H.: The Clustered Selected-Internal Steiner Tree Problem. International Journal of Foundations of Computer Science, Vol. 33, 2022, No. 1, pp. 55–66, doi: 10.1142/S0129054121500362.

[17] Prim, R. C.: Shortest Connection Networks and Some Generalizations. The Bell System Technical Journal, Vol. 36, 1957, No. 6, pp. 1389–1401, doi: 10.1002/j.1538-7305.1957.tb01515.x.

[18] Kruskal, J. B.: On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem. Proceedings of the American Mathematical Society, Vol. 7, 1956, No. 1, pp. 48–50, doi: 10.1090/S0002-9939-1956-0078686-7.

[19] Chen, L.—Abdellatif, S.—Gayraud, T.—Berthou, P.: A Steiner Tree Based Approach for the Efficient Support of Multipoint Communications in a Multi-Domain

Context. 2017 IEEE Symposium on Computers and Communications (ISCC), 2017, pp. 316–321, doi: 10.1109/ISCC.2017.8024549.

[20] Do, T. A.—Ban, H. B.—Huynh, T. T. B.—Le, M. T.—Nguyen, B. L.: Genetic Algorithm Based Approach to Solve the Clustered Steiner Tree Problem. Evolutionary Intelligence, Vol. 17, 2024, No. 3, pp. 1547–1566, doi: 10.1007/s12065-023-00848-w.

[21] Eshelman, L. J.—Schaffer, J. D.: Real-Coded Genetic Algorithms and Interval-Schemata. Vol. 2, 1993, pp. 187–202, doi: 10.1016/B978-0-08-094832-4.50018-0.

[22] Deb, K.—Agrawal, S.: A Niched-Penalty Approach for Constraint Handling in Genetic Algorithms. Artificial Neural Nets and Genetic Algorithms, Springer, Vienna, 1999, pp. 235–243, doi: 10.1007/978-3-7091-6384-9_40.

[23] Deb, K.—Deb, D.: Analyzing Mutation Schemes for Real-Parameter Genetic Algorithms. International Journal of Artificial Intelligence and Soft Computing, Vol. 4, 2014, No. 1, pp. 1–28, doi: 10.1504/IJAISC.2014.059280.

[24] Yu, X.—Gen, M.: Introduction to Evolutionary Algorithms. Springer, 2010, doi: 10.1007/978-1-84996-129-5.

[25] Mouret, J. B.—Clune, J.: Illuminating Search Spaces by Mapping Elites. 2015, doi: 10.48550/arXiv.1504.04909.

[26] Tarapore, D.—Clune, J.—Cully, A.—Mouret, J. B.: How Do Different Encodings Influence the Performance of the MAP-Elites Algorithm? Proceedings of the Genetic and Evolutionary Computation Conference 2016 (GECCO '16), 2016, pp. 173–180, doi: 10.1145/2908812.2908875.

[27] Nikfarjam, A.—Neumann, A.—Neumann, F.: On the Use of Quality Diversity Algorithms for the Travelling Thief Problem. 2022, pp. 260–268, doi: 10.1145/3512290.3528752.

[28] Derrac, J.—García, S.—Molina, D.—Herrera, F.: A Practical Tutorial on the Use of Nonparametric Statistical Tests as a Methodology for Comparing Evolutionary and Swarm Intelligence Algorithms. Swarm and Evolutionary Computation, Vol. 1, 2011, No. 1, pp. 3–18.

[29] Gupta, A.—Ong, Y. S.—Feng, L.: Multifactorial Evolution: Toward Evolutionary Multitasking. IEEE Transactions on Evolutionary Computation, Vol. 20, 2016, No. 3, pp. 343–357, doi: 10.1109/TEVC.2015.2458037.

Tuan-Anh Do received his M.Sc. degree in 2008 and his Ph.D. degree in 2025 from the Hanoi University of Technology (now the Hanoi University of Science and Technology—HUST), Hanoi, Vietnam. He is currently a Lecturer with the Faculty of Computer Science, School of Information and Communication Technology, HUST. His research interests include algorithm analysis, combinatorial optimization, software testing and debugging, and artificial intelligence. He also has extensive professional experience in the technology industry, having worked in the business sector for several years.

Ha-Bang Ban received his Ph.D. degree in computer science from the Hanoi University of Science and Technology (HUST), Hanoi, Vietnam, in 2015. He is currently Associate Professor with the School of Information and Communication Technology (SoICT), HUST. His research interests include algorithms, graph theory, optimization, and logistics. He has authored and coauthored numerous articles published in top-tier international journals and conferences.

Dang-Hai Pham received his engineering diploma in information technology from the Hanoi University of Science and Technology (HUST), Hanoi, Vietnam, in 1995, and his Ph.D. degree in computer science from the École Pratique des Hautes Études (EPHE), Paris, France, in 2011. He is currently a Senior Lecturer with the School of Information and Communication Technology (SoICT), HUST. His current research interests include algorithms, parallel and distributed simulation, multiagent-based simulation, and high-performance computing.

Le Minh Tu received his M.Sc. degree in data science from the Hanoi University of Science and Technology (HUST), Hanoi, Vietnam, in 2024. He is currently an AI Engineer. His research interests include artificial intelligence, deep learning, and speech processing. He has several years of professional experience in the technology industry, particularly in business-oriented environments.