Genetic Algorithm based Approach to Solve the Clustered Steiner Tree Problem

Tuan Anh Do[1], Ha-Bang Ban[1*], Thi Thanh Binh Huynh[1], Minh Tu Le[1] and Binh Long Nguyen[1]

[1]School of Information and Communication Technology, Hanoi University of Science and Technology, Dai Co Viet, Ha Noi, Vietnam.

*Corresponding author(s). E-mail(s): bangbh@soict.hust.edu.vn; Contributing authors: anhdt@soict.hust.edu.vn; binhht@soict.hust.edu.vn; minhtutx@gmail.com; binhlongmm99@gmail.com;

Abstract

In a complex network application, a set of nodes might be partitioned into multiple local clusters with different functions, properties, or communication protocols, and the communication is restricted between nodes of the same cluster to maximize efficiency and other security concern. Thus, there has been a rise in network design problems with additional constraints regarding the clustering of vertices, one of them being Clustered Steiner Tree Problem - a variant of the Steiner Tree Problem. Recently, a heuristic-based algorithm was proposed to solve the problem. However, its obtained result is limited in terms of solution quality when applied to the problem in the Euclidean case. This paper proposes a Genetic Algorithm called Shortest-Path Genetic Algorithm for solving the CluSteiner. In the proposal, a permutation-based individual representation that reduces the dimensionality of chromosomes to the number of clusters is devised. The proposed algorithm can solve the problem in both Euclidean and non-Euclidean cases. Experiment results compared to existing works in the literature are analyzed in detail to prove the effectiveness of the proposed algorithm.

Keywords: Genetic Algorithm, Clustered Steiner Tree Problem

1 Introduction

Steiner Tree Problem (STP) [1] is a well-known classical NP-Hard problem in optimization and graph theory. The problem revolves around finding a minimal cost tree T - a connected acyclic subgraph - of a graph $G = (V, E, w)$ that spans all the vertices in a certain set of destinations $R \subset V$. This tree represents multipoint communications in the network, which are useful for many applications since it reduces the overhead of maintaining multiple one-on-one connections. For their theoretical significance and real-world applications, many of the Steiner Tree problem variants have been developed and extensively studied, one of which is Clustered Steiner Tree Problem (CluSteiner) [2].

The CluSteiner belongs to the class of clustered graph problems, alongside others such as Clustered Traveling Salesman Problem (CluTSP) [3], Clustered Shortest-Path Tree Problem (CluSPT) [4], Minimum Routing Cost Clustered Tree Problem (CluMRCT) [5], etc. These problems are distinguished in terms of their input graphs and objective functions. Table 1 compares the differences among these three types of clustered problems. In a complex network application, a set of nodes might be partitioned into multiple local clusters with different functions, properties, or communication protocols. The communication is restricted between nodes of the same cluster to maximize efficiency and other security concerns. Thus, there has been a rise in network design problems with an additional constraint on the clustering of vertices.

This problem was introduced by Bang et al. and was shown to be an NP-hard problem even when all the local topologies and the inter-cluster topology are given [2]. Given an undirected graph with non-negative edge weights and a subset of vertices, usually referred to as required vertices, the classical Steiner Tree Problem in graphs requires a tree of minimum weight that contains all required vertices (but may include additional vertices). The CluSteiner problem further employs an addition constraint used in many other clustered tree problems, such that the nodes composing each cluster must also be a connected sub-tree in the resulting tree T. More specifically, each destination node belongs to one and only one cluster, and all the local trees are mutually disjoint, where a local tree is a minimal subtree in T that spans all nodes in a cluster.

Despite the practicality and theoretical significance of CluSteiner, the research and studies of the problem are currently few and far between. Motivated by the practicality of CluSteiner, we propose a first approximate method to tackle the CluSteiner problem, more specifically, using Genetic Algorithm (GA). GA is a search heuristic inspired by Darwin's theory of natural evolution. This algorithm reflects the process of natural selection and "survival of the fittest". It has been utilized to solve various combinatorial optimization problems and has demonstrated effectiveness on some clustered graph problems [6, 7].

Consequently, this paper proposes a novel Genetic Algorithm for the CluSteiner problem called Shortest-Path Genetic Algorithm (SPGA). The main contributions of this paper are summarized as follows:

Table 1: Problem Comparison

Problem	Input	Objective
CluTSP	Graph consists of vertices and edges in which the vertex set is partitioned into a certain number of clusters.	Finding the shortest Hamilton cycle consecutively visits every vertex in each cluster.
CluSPT	Graph consists of vertices and edges in which the vertex set is partitioned into a certain number of clusters.	Finding a shortest-path spanning tree from a given source to all the other nodes while each cluster must induce a connected subtree.
CluSteiner	Graph consists of free vertices, required vertices, edges, in which the required vertex set is partitioned into a certain number of clusters.	Finding a minimum-cost Steiner tree inducing mutually disjoint minimal spanning trees in the clusters of a required vertex set.

- Propose a GA combined with Shortest-Path Heuristic (SPH) to solve the CluSteiner.
- Devise a permutation-based encoding that reduces the dimensionality of chromosomes to the number of clusters and a corresponding decoding mechanism.
- Analyse the computational complexity and experiment results on various test instances to portray the efficiency of the proposal.

This paper is organized as follows. Section 2 examines the literature related to CluSteiner and GA. Section 3 provides the problem formulation, while some preliminaries used in our proposal are stated in Section 4. Section 5 describes our proposed algorithm in detail. An empirical study to analyze the algorithm's effectiveness is provided in section 6. Section 7 provides some discussions. Lastly, section 8 concludes some insights into the proposal and presents some future directions.

2 Related Works

With many applications, especially in the field of multicast and multipoint communication, Steiner Tree Problem is a class of combinatorial optimization problems that has been rigorously studied by many researchers. Numerous studies have been carried out to address multiple variants of this problem; for example, the Euclidean Steiner Tree Problem is a popular NP-hard problem where the Steiner points lie on a Euclidean plane or the Rectilinear Steiner tree problem where the rectilinear distance is used instead of Euclidean distance. Due to NP-hard, unless P = NP, a polynomial time approximation scheme is unlikely to exist. Therefore, it is not easy to solve well by using simple graph algorithms.

In [8], L. Chen et al. proposed a heuristic algorithm to solve the Steiner Tree Problem in a multi-domain context. The algorithm starts with a random node from the set of terminal vertices R and chooses it as the Steiner tree T. At each iteration, the solution is grown by adding the shortest path from T to the next element of R. When all elements of R are reached, the algorithm ends. The authors proved that the total length of the resulting tree is no greater than $2 \times (1 - \frac{1}{q})$ times the optimal tree, where q is the number of terminal nodes.

Since most versions of the Steiner Tree problems are NP-hard, popular approaches have been developing approximation algorithms, heuristics, or metaheuristics. Among those approaches, GA - a metaheuristic that belongs to the larger class of Evolutionary Algorithm (EA) - has proved to perform well for this problem type [9–13]. GA have been commonly used to generate high-quality solutions to optimization and search problems by relying on biologically inspired operators such as mutation, crossover, and selection [14].

In recent years, there has been a growing interest in the class of minimum cost clustering graph problems such as the CluTSP, CluSPT, CluMRCT, and many others. For instance, the NP-hard CluSPT has recently been widely investigated in its theoretical aspects [15], not to mention many approximation algorithms have been developed to tackle this problem [16–20]. Another example is the CluMRCT, which is an NP-hard problem with numerous practical applications, especially in network design, computational biology, and transportation [21–23].

The CluSteiner is no exception to this trend. This problem was introduced in [2], where Bang and Chen showed that the Steiner ratio for CluSteiner is lower and upper bounded by three and four, respectively. For CluSteiner, the Steiner ratio is the largest possible ratio of the minimal cost without using any Steiner vertex to the optimal cost. The authors investigated the Clustered Steiner tree problem on metric graphs, which are (non-negative) weighted graphs that satisfy the triangle inequality, i.e., $w(a, b) + w(b, c) > w(a, c)$ and proved that this problem is NP-hard. They proposed a $(2 + p)$-approximation algorithm, where p is the approximation ratio for the Steiner tree problem. However, we identify two drawbacks to their algorithm. Initially, their algorithm is only applied to metric problems. Secondly, the ratio of the approximation approach is not good in the general case. Therefore, solution quality is impossible to meet practical requirements. To fulfill the omission, we propose a metaheuristic including two subproblems. The first one is the minimum inter-cluster Steiner tree problem (InterCluster), in which we want to find a cluster Steiner tree with minimum inter-cluster cost and ignore the local cost. The second is the minimum local-cost Steiner tree problem (LocalCluster) which asks for the minimum local cost among all clustered Steiner trees with minimum intercluster cost.

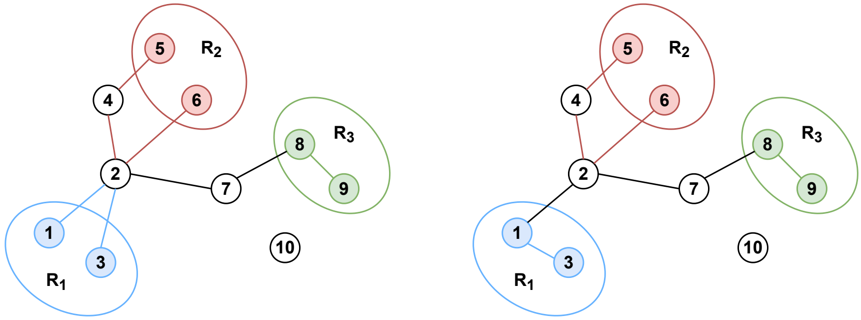
3 Problem Formulation

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 belong 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 $R' = \{R_1, R_2, ..., R_k\}$ 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 2.

Table 2: Clustered Steiner Tree problem definition

Input	- A weighted, complete graph $G = (V;\ E;\ w)$ - A set of required vertices $R \subset V$ - A partition $R' = \{R_1, R_2, ..., R_k\}$ 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: $\forall 1 \leq i < j \leq k, V_{T_i} \cap V_{T_j} = \emptyset$



(a) Invalid

(b) Valid

Fig. 1: An example of invalid and valid solution

An example of invalid and valid solution is illustrated in figure 1. In figure 1a, both the minimal local trees of cluster R_1 and cluster R_2 must contain vertex 2 so they are not mutually exclusive, hence the invalidity. Meanwhile, in figure 1b, the graph can be divided into three separate local trees {1, 3}, {2, 4, 5, 6}, and {8, 9} spanning the three clusters R_1, R_2, and R_3 respectively.

4 Preliminaries

4.1 CluMST algorithm

The CluSteiner problem was first introduced by Bang and Chen in [2]. After proving that this problem is NP-hard, they proposed a $(2 + p)$-approximation algorithm, where p is the approximation ratio for the Steiner Tree Problem. For each $1 \leq i \leq k$, the algorithm determines the local tree T_i of R_i as the Minimum Spanning Tree (MST) of $G_{R_i} = (R_i, E_{R_i})$, which is a subgraph of G containing only the vertices in R_i. Then, the graph G is transformed by contracting each T_i into a new vertex; the edges that are previously connected to any vertex in T_i are now connected to the contracted vertex instead. Formally, for a graph $G = (V, E, w)$:

- Contracting an edge (u, v) means replacing u, v with a new vertex s, removing every other edge (u, t) or (v, t), adding edge (s, t) with $w(s, t) = min(w(u, t), w(v, t))$. s is a contracted vertex.
- Contracting a tree T_i means contracting all edges in T_i, in any order.
- Contracting the graph G means contracting all local trees T_i, in any order.

Consequently, the remaining task is to solve the original Steiner Tree Problem on the newly contracted graph with the contracted vertices as required vertices. Any p-approximation algorithm of the Steiner Tree Problem can now be plugged in to solve on the reduced graph.

This algorithm will hereby be called CluMST algorithm. Its pseudocode is shown in algorithm 1, where SMT(G', S) is a p-approximation algorithm that can solve the Steiner Tree Problem.

Algorithm 1 CluMST algorithm

Input: $G = (V, E, w)$, $R \subset V$, partition $R' = \{R_1, R_2, ..., R_k\}$ of R
Output: A clustered Steiner tree $T = (V_T, E_T)$ for R'

1: $\forall 1 \leq i \leq k$, local tree $T_i \leftarrow$ MST of $G_{R_i} = (R_i, E_{R_i}, w)$
2: $G' \leftarrow$ the contracted graph of G
3: $S \leftarrow$ the set of contracted vertices of G'
4: $T_0 \leftarrow$ SMT(G', S)
5: $T \leftarrow$ combination of $T_i, 0 \leq i \leq k$
6: **return** T

The $(2 + p)$-approximation of this algorithm is based on the fact that, in a metric graph, the total weight of the minimum spanning tree for a set of vertices, MST(G,R), is no more than twice the total weight of the Steiner minimum tree, SMT(G,R).

$$MST(G, R) \leq 2 \times SMT(G, R)$$

This inequality is already verified in the paper [2]. First, we obtain an Eulerian tour by doubling the edges of the Steiner minimum tree. Then, because the graph is metric, by repeatedly taking a shortcut in the tour, we obtain a Hamiltonian path of G_R with less total weight than the Eulerian tour. Moreover, the Hamiltonian path of a graph always weighs more than its minimum spanning tree (removing an edge on the path makes a spanning tree), hence the inequality. However, this paper will not focus on it because the inequality does not hold on the non-metric graph.

4.2 The Shortest-Path Heuristic

In [8], L. Chen et al. consider a multi-domain network composed of multiple network domains interconnected from their border nodes with inter-domain links. This problem is similar to the CluSteiner in that both are variants of the Steiner Tree Problem with a clustering constraint. However, instead of only R being clustered, the problem input provides the clustering information of the whole vertex set V of G.

Table 3: Steiner Tree problem in multi-domain context

Input	- A weighted undirected graph $G = (V;\ E;\ w)$ - A set of required vertices $R \subset V$ - A partition $V' = \{V_1, V_2, ..., V_k\}$ of V. $R_i = R \cap V_i$
Output	A Steiner tree $T = (V_T, E_T)$
Objective	Minimize $\Sigma_{e \in E_T} w(e)$
Constraints	- T is a Steiner tree: $R \subset V_T$ - A subtree $T_i = (V_{T_i}, E_{T_i})$ is a Steiner tree of cluster V_i. $R_i \subset V_{T_i}$, $V_{T_i} \subset V_i$

In this paper, the authors proposed to use an algorithm called SPH to solve the Steiner Tree Problem. The first step of SPH, which uses the Floyd–Warshall algorithm on the subgraph of every domain, constructs a domain abstraction. Then, they use SPH to build the Steiner tree that spans nodes belonging to different domains. This paper will only focus on the algorithm SPH itself.

The algorithm starts with a random node from the set of required vertices R and chooses it as the Steiner tree T. At each iteration, T grows by adding the shortest path from T to the nearest vertex of R that is not in T. This nearest vertex can be found using Dijkstra's algorithm with multiple starting points. When all elements of R are reached, the algorithm ends.

Algorithm 2 SPH

Input: $G = (V, E, w)$, $R \subset V$
Output: A Steiner tree $T = (V_T, E_T)$
T $\leftarrow$ random $r \in R$
while $\exists r \in R, r \notin V_T$ **do**
  $r \leftarrow$ the nearest vertex in R but not in T
  $T = T +$ the shortest path to r
end while
return T

The authors proved that the total length of the resulting tree is no greater than $2 \times (1 - \frac{1}{k})$ times the optimal tree, where k is the number of required vertices. The complexity of the whole algorithm is $O(knlog_2n + km)$, where $n = |V|$ and $m = |E|$.

Interestingly, when the graph is metric, the SPH algorithm is equivalent to the Prim's minimum spanning tree algorithm. This is because, in metric graphs, the shortest path from a vertex u to another vertex v is the edge (u, v). Therefore, in metric graphs, a Steiner tree generated by SPH algorithm is the minimum spanning tree of $G_R = (R, E_R, w)$.

5 Proposed Algorithm

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

5.1 Algorithm framework

The main idea of this approach is to divide the CluSteiner into two subproblems:

- Finding the local trees of k clusters.
- Finding edges connecting all k clusters into one tree.

For the first subproblem, the SPH algorithm can be used on every subgraph $G_i = (R_i \cup (V \setminus R), E_{G_i}, w)$ to find a local tree corresponding to the i^{th} cluster. Let the free vertex set be the set of non-required vertices (belong to $V \setminus R$) that have not been added to the solution component. Any free vertex can only be in a maximum of one local tree. Therefore, when constructing a local tree, any Steiner vertex that has already been included in previously built local tree is excluded from the free vertex set. However, this means a change in the order of clusters to construct a local tree would heavily affect the final result. Since the number of choices is equivalent to the number of cluster permutations ($k!$), an exhaustive search solution would be computationally infeasible. Hence, the author hereby proposes using GA for finding the best permutation of clusters. The algorithm is called SPGA, and its general framework is shown in Algorithm 3.

Algorithm 3 SPGA

Input: $G = (V, E, w)$, $R \subset V$, $R = \{R_1, R_2, ...R_k\}$
Output: A clustered Steiner tree $T = (V_T, E_T)$

```
P_0 ← Initialize N individuals
Evaluate(P_0)                                   ▷ refer to 5.3
T ← Best solution in P_0
i ← 0
while (Stopping conditions not satisfied) do
    C_i ← Crossover(P_i) + Mutation(P_i)        ▷ refer to 5.4.1, 5.4.2
    Evaluate(C_i)                               ▷ refer to 5.3
    P_{i+1} ← Selection(P_i + C_i)              ▷ refer to 5.4.3
    T ← Best solution in P_{i+1}
    i ← i + 1
end while
return T
```

5.2 Permutation Representation

In SPGA, an individual is encoded as a permutation of k, an array of distinct integers representing the priority of clusters. A cluster with higher priority will be visited and have its local tree constructed before another cluster with lower priority.

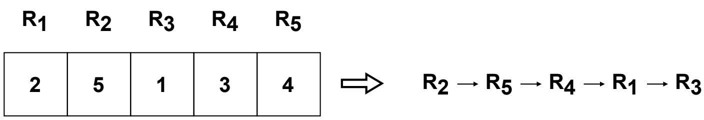


Fig. 2: An example chromosome representation

Figure 2 illustrates an example of permutation representation. In the example, cluster R_2 has a higher priority (priority 5) than any other cluster so it would be visited first. Conversely, cluster R_3 having the lowest priority (priority 1) will be visited last.

The first population P_0 is initialized by generating N random permutations, representing N individuals.

5.3 Chromosome Evaluation

To evaluate a chromosome represented by a permutation, we need to build a clustered Steiner tree and calculate its cost. The evaluation procedure of SPGA consists of two main steps corresponding to the two above-mentioned subproblems:

- Use SPH algorithm to construct the local tree of each cluster following the order determined by the priority permutation. The cost of each local tree T_i is the sum of all its edges' weight: $c(T_i) = \Sigma_{e \in E_{T_i}} w(e)$.
- Contract the graph by shrinking each cluster into a vertex, then use SPH algorithm to construct the inter-cluster links.

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:

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

5.3.1 Construct local trees

Following the order determined by the chromosome, for each cluster R_i, reduce the initial graph G to construct subgraph $G_i = (V_i, E_i)$:

- $V_i = R_i \cup F_i$ where F_i is the set of free vertices that have not yet been included in any local tree. Let $F = V \setminus R$ be the set of all free vertices, then F_i is determined by:

$$F_i = F \setminus (F \cap (V_{T_1} \cup V_{T_2} \cup ... \cup V_{T_{i-1}}))$$

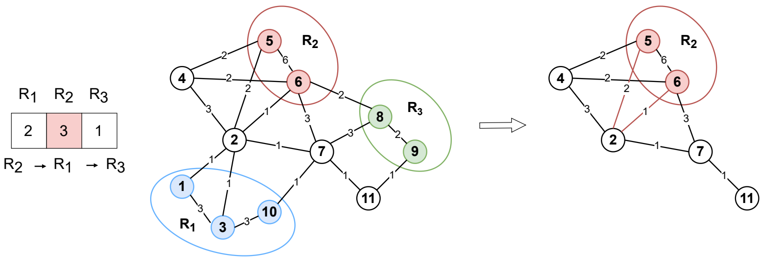
- $E_i \subset E$ is the set of edges that have both endpoints are vertices in V_i :

$$E_i = \{(u, v) \in E | u \in V_i, v \in V_i\}$$

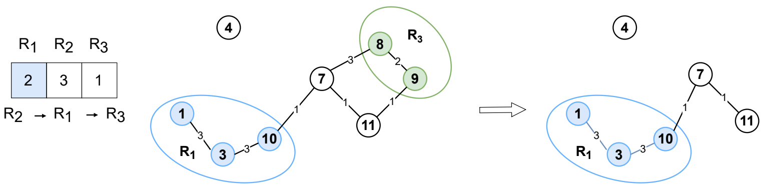
On each subgraph G_i, the SPH algorithm (Algorithm 2) is utilized to construct the Steiner local tree $T_i = (V_{T_i}, E_{T_i})$ of cluster i.

Figure 3 demonstrates an example of a graph G with 3 clusters and the chromosome indicating the order of clusters: $R_2 \rightarrow R_1 \rightarrow R_3$. In figure 3a, the local tree T_2 of cluster 2 found by SPH contains three vertices (2, 5, 6) with the cost: $c(T_2) = w(2, 5) + w(2, 6) = 2 + 1 = 3$. Similarly, the costs of the other two local trees are $c(T_1) = w(1, 3) + w(3, 10) = 3 + 3 = 6$ and $c(T_3) = w(8, 9) = 2$. So the total cost of all local trees is $\alpha(T) = 3 + 6 + 2 = 11$.

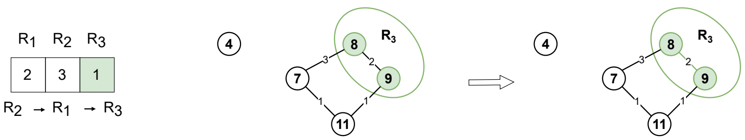
Notice that in another chromosome where the priority of cluster 1 is higher than that of cluster 2, the local tree T_1 for cluster 1 would be constructed first, and it would consist of vertices (1, 2, 3, 7, 10) with cost $c(T_1) = w(1, 2) + w(2, 3) + w(2, 7) + w(7, 10) = 1 + 1 + 1 + 1 = 4$. Consequently, the cost of T_2, which now consists of vertices (4, 5, 6), would be $c(T_2) = w(4, 5) + w(4, 6) = 2 + 2 = 4$, making the total cost of all local trees be $\alpha(T) = 4 + 4 + 2 = 10$.



(a) Constructing G_1 of cluster 2



(b) Constructing G_2 of cluster 1



(c) Constructing G_3 of cluster 3

Fig. 3: Construction of local trees example

5.3.2 Construct inter-cluster connections

For a graph $G = (V, E, w)$, *contraction* of an edge $(u, v) \in E$ means replacing vertices u, v with a new vertex s, which s is considered a contracted vertex. For any other vertex t that has a connection to u or v, the edge is replaced with a new edge (s, t), and the weight of (s, t) is set to $w(s, t) = min(w(t, u), w(t, v))$. Furthermore, let the graph $G' = (V', E')$ be a subgraph of G; contraction of graph G' means contracting all of the edges in E' in an arbitrary order in a new graph denoted as G/G'.

A contracted graph G_0 of G is a graph produced by contracting all its local trees $T_1, T_2, ..., T_k$ constructed in the previous step. In other words, $G_0 = G/T_1/T_2/.../T_k$. Figure 4 demonstrates how graph G in the previous example is contracted.

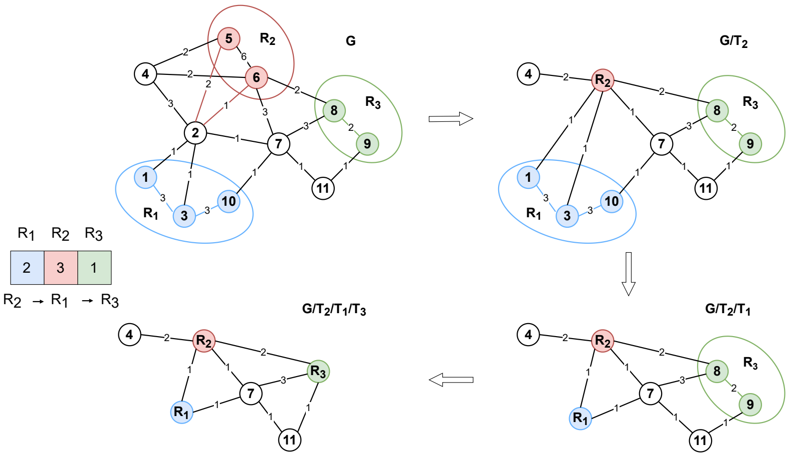


Fig. 4: Contraction of graph G

Let S be the set of contracted vertex on G_0. The problem of finding inter-cluster edges on G now becomes finding the minimum Steiner tree on the contracted graph G_0, and the set of required vertices is S. And we can once again utilize the SPH algorithm. The cost of the inter-cluster links in G is equivalent to the cost of the Steiner tree yielded by the SPH algorithm on (G_0, S).

$$\beta(T) = c(SPH(G_0, S))$$

5.3.3 SPH Implementation

The SPH algorithm was already shown in the previous section (Algorithm 2). However, the complexity of SPH is quite large $O(knlog_2n + km)$, where $k = |R|, n = |V|, m = |E|$. This is mainly due to the repetitive use of Dijkstra's algorithm to find the closest required vertex.

This section introduces a different SPH implementation that only applies Dijkstra's algorithm once but yields the same result. Instead of keep resetting Dijkstra's algorithm whenever we find a new closest required vertex, we add every vertex on the path to the closest required vertex back to the unvisited set with a tentative distance of 0.

The modified SPH algorithm is shown in Algorithm 4. Some notations:

- S: Unvisited vertex set, initially contains all vertices.
- S_R: Unvisited required vertex set, initially contains all required vertices.
- $d(v)$: Tentative minimum distance to v from among vertices currently in T. Initially, d(v) is set to infinity for all v.
- $p(v)$: Predecessor of v in the tentative shortest path from T.

Algorithm 4 Modified SPH

Input: $G = (V, E, w)$, $R \subset V$
Output: A Steiner tree $T = (V_T, E_T)$ spanning R
1: $V_T \leftarrow \emptyset, E_T \leftarrow \emptyset$
2: $S \leftarrow V$
3: $S_R \leftarrow R$
4: $\forall v \in V : d(v) \leftarrow \infty$
5: $r \leftarrow$ random in R
6: $d(r) \leftarrow 0$
7: $S \leftarrow S \setminus \{r\}$
8: **while** S_R is not empty **do**
9:     $u \leftarrow argmin_{u \in S}\ d(u)$
10:     $S \leftarrow S \setminus \{u\}$
11:     **if** $u \in S_R$ **then**
12:         $S_R \leftarrow S_R \setminus u$
13:         **while** $u \notin V_T$ **do**
14:             $V_T \leftarrow V_T \cup u, \quad E_T \leftarrow E_T \cup (p(u), u)$
15:             $d(u) \leftarrow 0, \quad S \leftarrow S \cup u$
16:             $u \leftarrow p(u)$
17:         **end while**
18:     **end if**
19:     **for** $\forall (u, v) \in E$ **do**
20:         **if** $d(u) + w(u, v) < d(v)$ **then**
21:             $d(v) \leftarrow d(u) + w(u, v)$
22:             $p(v) \leftarrow u$
23:         **end if**
24:     **end for**
25: **end while**
26: **return** T

In the modified SPH algorithm, the while loop in line 13 backtracks from the current required vertex u to its predecessor in tree T. Each iteration adds vertex u and its corresponding edge into the tree (line 14) and adds vertex u back to the unvisited set with distance 0 (line 15). Since each vertex can only be added to the tree once, the total number of iterations here is $n = |V|$. Furthermore, because each vertex can only be added back to the unvisited set at most once (when it is added to tree T), the number of iterations of the main while loops (line 8) is at most $2n = 2|V|$. Hence, each $edge(u, v)$ in E can only be referred to at most twice (line 19) for a total of $2m = 2|E|$ times. Overall, if we use binary data structure like set (binary search tree) or heap (priority queue) to store and retrieve minimum unvisited vertex (line 9), the complexity of the modified SPH algorithm is $O(n + 2nlog_2n + 2m) = O(nlog_2n + m)$.

5.4 Genetic Operator

5.4.1 Order Crossover

Order Crossover (OX1) is the crossover operator used in SPGA to create child individuals from two-parent individuals.

- Create two random crossover points.
- Copy the segment between the crossover points from the first parent to the child.
- Starting from after the second crossover point in the second parent, copy the remaining unused number to the child, wrapping around the list.

By swapping the role of the two parents, each crossover operation can create two offspring individuals.

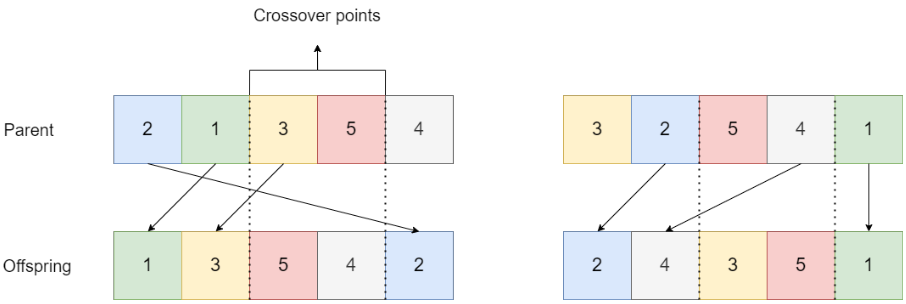


Fig. 5: An example Order Crossover (OX1)

In each generation, crossover operator is executed $N/2$ times on $N/2$ random pairs of parent individuals, where N is the population size. In total, the size of the offspring population produced by the crossover operator is N.

5.4.2 Swapping Mutation

The mutation operator of SPGA is Swapping Mutation. This operator simply swaps the position of two random genes in the chromosome.

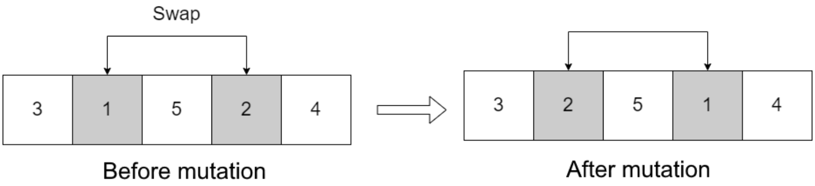


Fig. 6: Swap Mutation

The mutation operator is executed $\lceil p_m \times N \rceil$ times in each generation on random individuals, where p_m is the mutation percentage.

5.4.3 Elitish 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.

5.5 Time complexity analysis

The time complexity of SPGA is estimated by the following formula:

$$O(SPGA) = O(crossover) + O(mutation) + O(selection) + O(evaluation)$$

Let k be the number of clusters and also the length of a chromosome. Each crossover or mutation operator has linear complexity with respect to k. Each generation generates a number of child individuals linear to POP_SIZE. So:

$$O(crossover) = O(mutation) = O(MAX\_GENERATIONS \times POP\_SIZE \times k)$$

The selection operator needs to sort the population in order to select the fittest individuals. The sort function has linearithmic time complexity.

$$O(selection) = MAX\_GENERATIONS \times POP\_SIZE \times log_2 POP\_SIZE$$

The time complexity of SPH is $O(nlog_2 n + m)$ (subsection 5.3.3). Each evaluation finds the local Steiner tree of k clusters and the inter-cluster Steiner tree by using SPH. So the time complexity of all evaluations is:

$$O(evaluation) = MAX\_EVALUATIONS \times k \times (nlog_2 n + m)$$

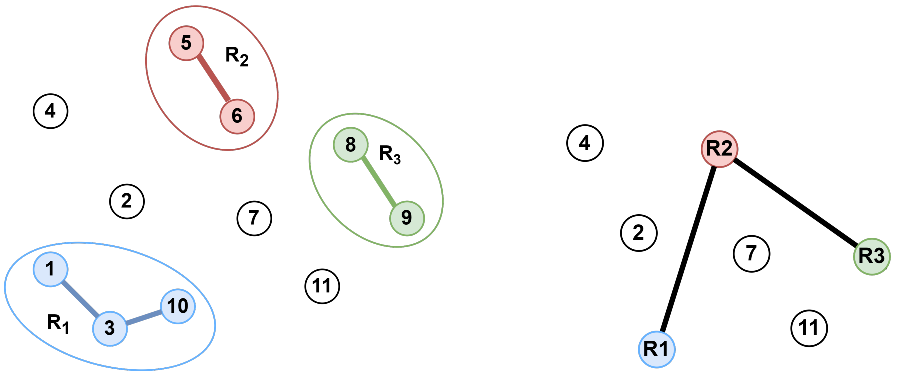
It is clear that the main bottleneck of SPGA is the time complexity of the evaluation procedure. The time complexity of SPGA can be considered equivalent to the time complexity of evaluation:

$$O(SPGA) = O(evaluation) = MAX\_EVALUATIONS \times k \times (nlog_2 n + m)$$

5.6 SPGA in metric case

In metric graphs, SPH is equivalent to the Prim's algorithm (subsection 4.2), so the local tree of cluster R_i is the minimum spanning tree of $G_{R_i} = (R_i, E_{R_i}, w)$. Therefore, each local tree constructed by SPH spans all required vertices of the corresponding cluster and does not contain any Steiner vertex as in figure 7a.

As a result, the order of clusters has no influence on the solution construction for each cluster's local tree. After each cluster is contracted to a vertex in the graph G', an inter-cluster connection is constructed. This stage in SPGA uses SPH to determine the shortest distance between 2 clusters R_i and R_j in G' by finding the shortest path between a vertex in V_i and a vertex in V_j. In other words, the inter-cluster tree constructed using SPH also does not contain any Steiner node, as in figure 7b. It leads to a consequence when CluMST uses SPH as a Steiner Tree Problem solver. It gives the same results as SPGA on the metric graphs. The problem then becomes trivial in the case of metric graphs; therefore, we are interested in the non-metric graph case in this paper.



(a) Building local trees

(b) Connecting local trees on G'

Fig. 7: An example of SPGA on metric graph

6 Computational Results

6.1 Problem instances

A set of instances in our experiments are chosen from Non-Euclidean CluSPT ([24]) in which each instance is an input graph. We take the first vertex in each cluster and randomly select some vertices in the remaining ones. The non-selected vertices become the non-required vertices. In this way, the required vertices in each instance are different, and each cluster has at least one vertex. There are seven datasets comprising 140 instances in total, categorized into two kinds 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 4.

Table 4: Dataset information

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: Instances, ReqVertices: Required Vertices

6.2 Experiment criteria

The quality of the algorithms was assessed by the following criteria:

- **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 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^i_a and AVG^i_b be the average value over 30 runs on instance i of two algorithms a and b, respectively. Then the improvement of the algorithm a over algorithm b is:

$$PI^i_{ab} = \frac{AVG^i_b - AVG^i_a}{AVG^i_b} \times 100\%$$

6.3 Experimental Setting

To evaluate the efficiency of the proposed metaheuristic, we implement it on a benchmark dataset and compare it with the other algorithms. However, there is only an approximation algorithm for the CluSteiner in the metric case [4]. Besides this algorithm, we do not find any effort on metaheuristics to solve

Table 5: A list of algorithms in this paper

Algorithm	Type	metric	non-metric
CluMST [2]	Approximation ratio	x	
SPMST	Heuristic	x	x
BSPH	Heuristic	x	x
SPGA	Metaheuristic	x	x

this problem. Therefore, we adjust the approximation ratio algorithm with the Steiner Tree Solver being SPH to solve CluSteiner in both metric and non-metric cases. Therefore, these algorithms can be compared with the same instances. Table 5 demonstrates all algorithms used in the experiments.

- **SPMST** is the algorithm 1, which was published in [2] with the Steiner Tree Solver being SPH.
- **BSPH** is a variation of SPGA. It is similar to the evaluation function of SPGA, but instead of using GA to optimize the permutation of clusters, we generate 50000 ($MAX\_EVALUATION$ of SPGA) random permutations and evaluate them to find the permutation that gives the best solutions. The comparison with this algorithm will show how the evolution mechanism of GA improves solutions compared to just randomly exploring the search space.

Each algorithm was simulated 30 times on each instance. The hardware environment is Intel(R) Core(TM) i5-3470 CPU 3.2GHz, 16GB RAM. The source codes were implemented in Java language.

The parameters used for SPGA are provided in Table 6. In the pilot study, these parameters are chosen so that the proposed algorithm takes the best solutions. This parameter setting has thus been used in the following experiments.

Table 6: Parameters for SPGA

Parameter	**Definition**	**Value**
POP_SIZE	Number of individuals in population	100
$MAX\_GENERATIONS$	Maximum number of GA generations	500
$MAX\_EVALUATIONS$	Maximum number of evaluations	50000
P_c	Crossover probability	0.90
P_m	Mutation probability	0.05

6.4 Experimental Scenario

To evaluate the performance in detail of the proposed algorithms, we perform two experiments as follows:

Experiment 1: Analysis of the obtained results by the Non-parametric statistic.

Table 7: Results of the Friedman, Iman-Davenport, Aligned Friedman and Quade tests (α=0.05)

Friedman Value	Value in X^2	p-value	Iman-Davenport Value	Value in F_F	p-value
204.471	5.991	$1.18 * 10^{-10}$	**376.301**	3.028	$7.98 * 10^{-80}$
Aligned Friedman Value	Value in X^2	p-value	Quade Value	Value in F_F	p-value
93.350	5.991	$8.41 * 10^{-11}$	**216.297**	3.028	$2.22 * 10^{-57}$

Experiment 2: Analyze the quality of the proposed algorithm in comparison with other algorithms.

6.5 Experimental results

6.5.1 Non-parametric statistics to compare the results of the proposed algorithm and the existing algorithm

Non-parametric statistics are utilized to compare the efficacy of the algorithms BSPH, SPMST and the proposed algorithm SPGA. There are two significant steps in the comparing method:

- First, we employ statistical methods such as Friedman, Aligned Friedman, and Quade [25, 26] to compare the results obtained by the aforementioned algorithms.
- After rejecting the hypothesis of equivalence of means of results obtained by algorithms in the first step, post-hoc statistical procedures [25, 26] are utilized to compute the concrete differences among algorithms and to compare a control algorithm with the remaining algorithms.

The results obtained by algorithms on types of instances are pointed out in Table 1, Table 4, and Table 6. In these tables, the bold, black cells in a column of an algorithm denote instances where this algorithm outperforms the other three.

Table 7 illustrates the result of applying Friedman's, Iman-Davenport's, Aligned Friedman's, and Quade's tests. The results in this table show that all of Friedman, Iman-Davenport, Aligned Friedman, and Quade values are greater than their associated critical values. In addition, all p-values are less than 0.05, so all the null hypotheses, i.e., the equivalence of the medians of the results of the different benchmarks are rejected. In other words, there are significant differences among the observed results with a probability error of $p \leq 0.05$.

The ranking obtained by the Friedman, Friedman Aligned, and Quade tests is presented in Table 8. Results in this table strongly suggest the existence of significant differences among the algorithms considered. The results in Table 8 also point out that the SPGA algorithm has the smallest ranking. Thus it is selected as the control algorithm. After that, we compare the control algorithm

Table 8: Average rankings achieved by the Friedman, Friedman Aligned, and Quade tests

Algorithms	Friedman	Friedman Aligned	Quade
SPMST	2.9714	350.400	2.999
BSPH	1.6642	140.842	1.745
SPGA	1.3642	140.257	1.255

(SPGA) with two other algorithms (BSPH, and SPMST) by using more powerful statistical methods, i.e., Holland, Holm, etc. Table 9 shows all the possible hypotheses of comparison between the control algorithm and other algorithms, ordered by their p-value and associated with their level of significance.

Table 9: The z-values and p-values of the Friedman, Quade procedures (SPGA is the control algorithm)

		Friedman			**Quade**		
i	Algorithms	z	p	Holm	z	p	Holm
2	SPMST	13.44	$3.23 * 10^{-41}$	0.025	12.65	$1.01 * 10^{-36}$	0.025
1	BSPH	2.50	0.012	0.050	3.55	$3.74 * 10^{-4}$	0.050

The adjusted values p of the Friedman and Quade produced for comparisons between SPMST and BSPH algorithms with the control algorithm are presented in Table 10 and Table 11. The two algorithms, SPMST and BSPH, are worse than the control algorithm considering a level of significance $\alpha = 0.05$.

Table 10: Adjusted p-values for the Friedman test (SPGA is the control method)

i	Algorithms	Unadjusted p	p_{Bonf}	p_{Holm}
1	SPMST	$3.23 * 10^{-41}$	$6.47 * 10^{-41}$	$6.47 * 10^{-41}$
2	BSPH	0.012	0.024	0.012

6.5.2 Detail of comparison among the algorithms BSPH and SPGA

Tables 12 and 13 shows the PI values of SPGA over the two algorithms SPMST and BSPH respectively. It can be apparent that the results of SPGA dominate those of SPMST in all instances. With the largest improvement of 94.8% for instance 25pcb442-5x5 of Type 6 Large.

In a non-metric graph, the triangle inequality no longer holds. The weight of a direct edge between two vertices A and B might be significantly larger than a path through an intermediate vertex C. Hence, the cost could be geometrically high when building local trees with only direct edges among required vertices, like in SPMST. As for BSPH, its performance compared to SPGA is almost

Table 11: Adjusted p-values for the QUADE test (SPGA is the control method)

i	Algorithms	Unadjusted p	p_{Bonf}	p_{Holm}	p_{Finn}	p_{Pli}
1	SPMST	$1.01 * 10^{-36}$	$2.03 * 10^{-36}$	$2.03 * 10^{-36}$	0.0	$1.01 * 10^{-36}$
2	BSPH	$3.74 * 10^{-4}$	$7.49 * 10^{-4}$	$3.74 * 10^{-4}$	$3.74 * 10^{-4}$	$3.74 * 10^{-4}$

Table 12: PI (%) of SPGA over SPMST

	Type	**Minimum PI**	**Average PI**	**Maximum PI**	**Better**
Small	Type_1_Small	23.024055	60.643470	82.580147	27/27
	Type_5_Small	-34.834835	52.641868	79.989995	19/21
	Type_6_Small	8.395062	60.086472	81.314408	37/37
Large	Type_1_Large	73.470002	85.595204	92.720882	15/15
	Type_3_Large	73.087508	82.277263	90.520467	10/10
	Type_5_Large	63.635640	81.227604	90.823673	15/15
	Type_6_Large	71.970007	83.051579	89.714611	15/15

Table 13: PI (%) of SPGA over BSPH

	Type	**Minimum PI**	**Average PI**	**Maximum PI**	**Better**
Small	Type_1_Small	0.000000	0.000000	0.000000	0/27
	Type_5_Small	0.000000	0.000000	0.000000	0/21
	Type_6_Small	0.000000	0.000000	0.000000	0/37
Large	Type_1_Large	-0.103588	1.847157	4.923666	14/15
	Type_3_Large	-0.915995	2.210894	5.623651	6/10
	Type_5_Large	-0.052420	1.865452	5.362540	14/15
	Type_6_Large	0.132225	1.687693	3.811856	15/15

exactly equal for small instances. On the other hand, the results for large instances are largely in favor of SPGA, with its superiority in almost all large instances. In fact, there are only three instances where SPGA produces results with slightly worse AVG values (instance 25eil101 of Type 1 Small, instance 10i120-46 of Type 5 Small, instance 5i400-205 of Type 5 Large). The differences are negligible, and it is only for AVG value, both algorithms find the same BF value over 30 runs.

Although one could argue that the differences are small, only around 5.6% improvement at max, it can be seen more clearly in figure 8, where the RPD values of the algorithms are illustrated. It shows that SPGA produced much better results in terms of stability between runs, and the majority of results of SPGA are closer to 0. This has proven that GA is effective in improving the result over generations, which produces better overall results compared to just randomly exploring the search space like in BSPH.

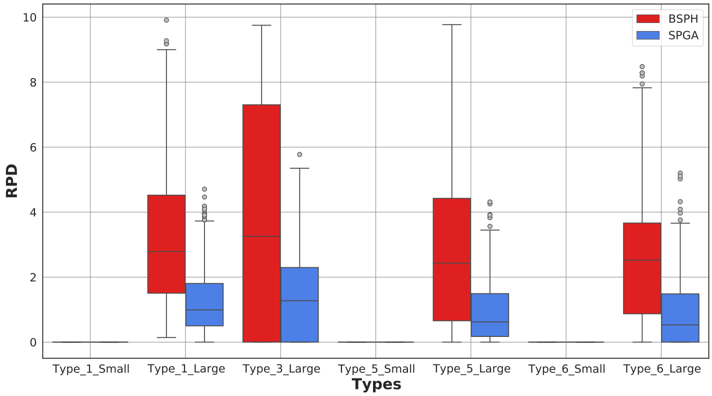


Fig. 8: RPD values of SPGA compared to BSPH

The detailed results of all instances are shown in Appendix.

6.5.3 Convergence trends and time running

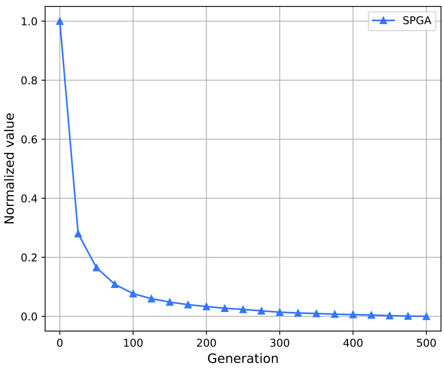
Figure 8 shows the convergence trends of SPGA in selected type. In Figure 8, the vertical axis is the normalized value while the horizontal axis is the number of generations. The normalized objective function is calculated as follows:

$$\overline{f} = \frac{(f - f^{min})}{(f^{max} - f^{min})},$$

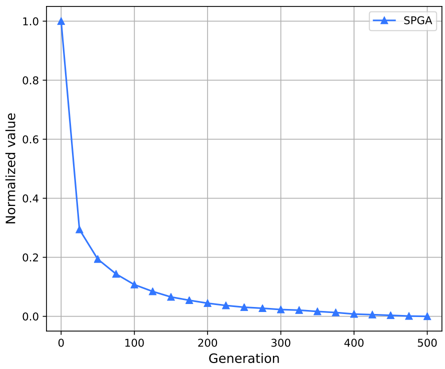
where f^{min}, f^{max} are the minimum and maximum function cost values across all test runs.
In Figure 8, the rate of improvement of solution quality increases very quickly after the first few generations and gradually slows down in later generations. The converge curve is smooth and spreads relatively evenly across generations. It demonstrates that the convergence rate is acceptable, and the algorithm SPGA preserves the population diversity during the evolution.

Figure 9 shows the running time of three algorithms. To compare their time complexity, three main perspectives can be considered. The same evaluation method can be found in some other works [27–29].

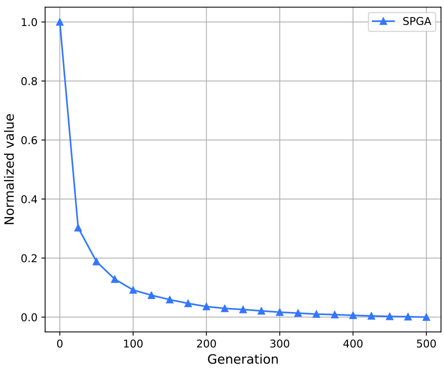
- The theoretical complexity: The time complexity of the SPMST, BSPH, and SPGA is $O((n+m) \times log(n)), O(log(n)+m)$, and $MAX\_EVALUATIONS \times k \times (n \times log(n)+m)$, respectively. Thus, the three algorithms are equivalent in terms of theoretical time complexity.
- The time complexity by CPU times: All algorithms are run on the same computer languages, platforms, and compilers. It is convenient to compare their running time by CPU times. To compare the running time of the algorithms, we visualize the average time on many types in figure 10. The running time of SPGA grows moderately compared to BSPH and SPMST.



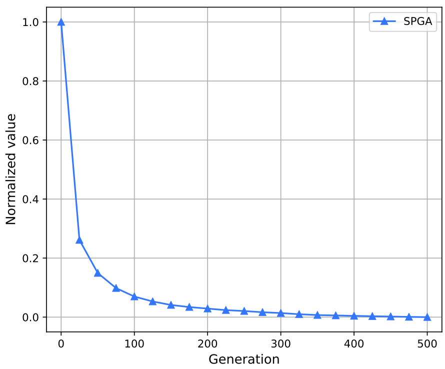
(a) Type 1



(b) Type 3



(c) Type 5

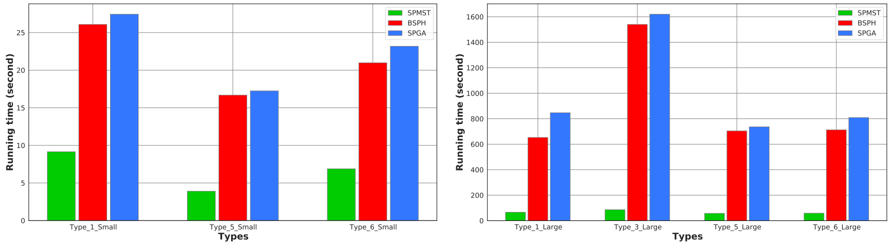


(d) Type 6

Fig. 9: The convergence trends of SPGA

The result is understandable because a metaheuristic-based algorithm often consumes more time than a heuristic-based one.

- The time complexity by function evaluations: All algorithms run with the same number of fitness evaluations. As we know, the results of BSPH and SPMST still remain unchanged. That means BSPH and SPMST fail to find any better solutions with the additional number of fitness evaluations. They might exploit well, but they do not have enough diversification to bring the search to unexplored search regions. That is the reason why they stuck to local optima. Increasing the additional number of fitness runs cannot improve the solution quality. On the other hand, the balance between diversification and intensification helps SPGA to explore extensive solution space. It increases the chance of finding good solutions. Moreover, the additional number of fitness does not make SPGA consume too much time because the complexity of fitness evaluation is only $O(n \times log(n))$.



(a) Small types

(b) Large types

Fig. 10: Average running time of SPMST, BSPH and SPGA by seconds

7 Discussion

The class of clustered problems on graphs includes some variants: Clustered Traveling Salesman Problem (CluTSP) [3], Clustered Shortest Path Tree Problem (CluSPT) [18], and Clustered Steiner Tree Problem (CluSteiner) [2]. These problems differ in input graph and objective function. To the best of our knowledge, good algorithms to solve this problem might not work for others. Therefore, we cannot directly compare our algorithm with the ones for the other problems. Developing an efficient algorithm for CluSteiner is necessary.

- CluSteiner has many real-world applications. In network design, the aim is to connect a set of cell towers using the minimum number of transmission towers while the distances between towers are satisfied. Another application is for VLSI design, in which the aim is to connect a set of pins on a chip such that the number of wires is minimum while the maximum lengths of the wires are held. CluSteiner belongs to the class of NP-hard [2]. For NP-hard, three common approaches are usually used to solve the problem such as 1) exact algorithm, 2) approximation algorithm, and 3) heuristic or metaheuristic algorithm. First, the exact algorithm finds the optimal solution, and its time complexity is exponential time in the worst case. Therefore, it only solves the problem with small sizes. Secondly, an α-approximation algorithm produces a solution within some factor of α of the optimal solution. The work [2] is of this type. Nevertheless, its best ratio is far from the optimal solution. Thirdly, heuristic algorithms perform well in reality and evaluate their performance on benchmark datasets. The proposed metaheuristic algorithm falls into the last approach. A good metaheuristic must balance intensification (exploitation) and diversification (exploration), in which diversification tends to explore promising solution spaces, while intensification focuses on the search for explored solution spaces. This paper proposes a metaheuristic in which GA finds the best permutation of clusters (exploration) while SPH finds the minimum local trees (exploitation). Therefore, the proposed algorithm ensures a good balance between intensification and diversification.

Additionally, the algorithm in [2] only solves the problem in the metric cases where all edge costs are nonnegative and satisfy the triangle inequality. The proposed metaheuristic, on the other hand, is applied well to both metric and non-metric graphs. Moreover, its theoretical complexity remains unchanged though graph input is either metric or non-metric.

-To evaluate the efficiency of the proposed metaheuristic, we implement it on a benchmark dataset and compare it with the other heuristic and metaheuristic algorithms in the literature. However, there is only a (2+p)-approximation algorithm, where p is the best-known approximation ratio for the Minimum Steiner Tree Problem. In addition, their algorithm only applies to the problem in the metric case. Besides the algorithm, we do not find any effort on metaheuristics in the literature to directly solve CluSteiner to compare with our metaheuristic. To fulfill the omission, we adjust the approximation ratio algorithm [2] with the Steiner Tree Solver being SPH to solve CluSteiner in metric and non-metric cases. Therefore, the algorithms can be compared with the same instances. The results show that the SPGA is able to find better solutions than the existing algorithms, BSPH and SPMST, even with the same number of fitness evaluations. Our algorithm performs better than the others because of two reasons as followings:

- Heuristics (BSPH and SPMST) are often too greedy. Therefore, it can get stuck into a local optimum. SPGA, otherwise, is not greedy and thoroughly explores a more solution space. Therefore, the chance of obtaining better solutions is higher.
- SPGA balances between diversification and intensification, helping to explore a wider solution space and increasing the chance of finding good solutions.

8 Conclusion

This paper proposed a first metaheuristic combining GA with SPH to tackle the CluSteiner. The proposed algorithm introduces a new encoding method that decreases the chromosome dimensionality to the number of clusters and corresponding decoding. The proposed algorithm is applied to Euclidean and non-Euclidean cases well. Experiments and comparisons with several algorithms on numerous data sets were conducted to evaluate the proposal's efficiency. In most instances, the SPGA is better than other algorithms. The improvement is large and significant. However, the running time needs to be improved. That is our aim for future research.

Acknowledgments. This work is funded by the Ministry of Education and Training of Vietnam for Do Tuan Anh under project code B2023-BKA-05. In addition, Nguyen Binh Long is funded by Vingroup JSC and supported by the Master, Ph.D. Scholarship Programme of Vingroup Innovation Foundation (VINIF), Institute of Big Data, code VINIF.2021.ThS.BK.02.

Declarations

- 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, Writing the manuscript.
  Ha-Bang Ban: Methodology, Algorithms, Coding, Writing the manuscript.
  Huynh Thi Thanh Binh: Methodology, Algorithms, Writing the manuscript.
  Minh Tu Le: Algorithms, Coding, Writing the manuscript.
  Binh Long Nguyen: Methodology, Coding, Writing the manuscript.

Table 1: Results obtained by SPMST, BSPH and SPGA on instances in Type_1_Small.

	SPMST				**BSPH**				**SPGA**			
Instance	BF	Avg	Std	Time	BF	Avg	Std	Time	BF	Avg	Std	Time
5eil51	1618.000	1618.000	0.000	3.042	689.000	**689.000**	0.000	9.523	689.000	**689.000**	0.000	10.147
5berlin52	2994.000	2994.000	0.000	1.977	972.000	**972.000**	0.000	9.524	972.000	**972.000**	0.000	9.972
5st70	2589.000	2589.000	0.000	2.778	451.000	**451.000**	0.000	13.628	451.000	**451.000**	0.000	14.187
5pr76	1697.000	1697.000	0.000	4.934	630.000	**630.000**	0.000	18.791	630.000	**630.000**	0.000	19.474
5eil76	1225.000	1225.000	0.000	3.728	527.000	**527.000**	0.000	18.574	527.000	**527.000**	0.000	19.440
10eil51	2262.000	2262.000	0.000	2.946	887.000	**887.000**	0.000	9.551	887.000	**887.000**	0.000	10.957
10berlin52	2083.000	2083.000	0.000	4.173	786.000	**786.000**	0.000	9.571	786.000	**786.000**	0.000	10.558
10st70	2497.000	2497.000	0.000	5.870	899.000	**899.000**	0.000	17.447	899.000	**899.000**	0.000	17.938
10eil76	1832.000	1832.000	0.000	5.170	905.000	**905.000**	0.000	22.545	905.000	**905.000**	0.000	24.287
10pr76	1202.000	1202.000	0.000	7.221	900.000	**900.000**	0.000	19.909	900.000	**900.000**	0.000	20.148
10rat99	2859.000	2859.000	0.000	7.733	928.000	**928.000**	0.000	34.173	928.000	**928.000**	0.000	35.285
10kroB100	1373.000	1373.000	0.000	8.140	676.000	**676.000**	0.000	34.118	676.000	**676.000**	0.000	35.702
15eil51	2104.000	2104.000	0.000	5.282	886.000	**886.000**	0.000	9.154	886.000	**886.000**	0.000	9.477
15berlin52	1455.000	1455.000	0.000	3.601	1120.000	**1120.000**	0.000	9.071	1120.000	**1120.000**	0.000	9.853
15st70	3507.000	3507.000	0.000	4.997	1071.000	**1071.000**	0.000	19.199	1071.000	**1071.000**	0.000	20.396
15eil76	3552.000	3552.000	0.000	5.753	929.000	**929.000**	0.000	22.364	929.000	**929.000**	0.000	23.604
15pr76	1724.000	1724.000	0.000	5.883	961.000	**961.000**	0.000	22.119	961.000	**961.000**	0.000	22.989
25rat99	4655.000	4655.000	0.000	14.544	1218.000	**1218.000**	0.000	36.696	1218.000	**1218.000**	0.000	39.354
25kroA100	2790.000	2790.000	0.000	14.741	1418.000	**1418.000**	0.000	42.044	1418.000	**1418.000**	0.000	45.116
25eil101	4254.000	4254.000	0.000	10.275	1161.000	**1161.000**	0.000	40.740	1161.000	**1161.000**	0.000	43.643
25lin105	4025.000	4025.000	0.000	10.615	884.000	**884.000**	0.000	40.029	884.000	**884.000**	0.000	41.857
50rat99	4321.000	4321.000	0.000	14.942	1313.000	**1313.000**	0.000	38.384	1313.000	**1313.000**	0.000	39.082
50kroB100	3896.000	3896.000	0.000	22.115	1383.000	**1383.000**	0.000	37.232	1383.000	**1383.000**	0.000	38.999
50kroA100	2954.000	2954.000	0.000	15.102	1347.000	**1347.000**	0.000	38.107	1347.000	**1347.000**	0.000	40.633
50eil101	3895.000	3895.000	0.000	15.260	1354.000	**1354.000**	0.000	38.360	1354.000	**1354.000**	0.000	40.310
50lin105	4716.000	4716.000	0.000	15.788	1556.000	**1556.000**	0.000	44.693	1556.000	**1556.000**	0.000	47.203
75lin105	4066.000	4066.000	0.000	30.637	1386.000	**1386.000**	0.000	48.740	1386.000	**1386.000**	0.000	50.486

Table 2: Results obtained by SPMST, BSPH and SPGA on instances in Type_1_Large.

	SPMST				BSPH				SPGA			
Instance	BF	Avg	Std	Time	BF	Avg	Std	Time	BF	Avg	Std	Time
10gil262	2878.000	2878.000	0.000	17.083	764.000	766.167	1.293	226.052	760.000	**763.533**	1.821	220.910
10a280	3139.000	3139.000	0.000	33.274	740.000	750.500	3.170	266.693	733.000	**737.800**	3.807	251.925
10lin318	3325.000	3325.000	0.000	28.906	706.000	**707.933**	2.476	343.999	705.000	708.667	3.004	331.648
10pr439	2723.000	2723.000	0.000	71.424	602.000	609.033	3.610	670.125	600.000	**603.967**	2.689	801.906
10pcb442	3972.000	3972.000	0.000	37.699	638.000	651.100	4.784	725.247	629.000	**637.967**	4.317	872.675
25gil262	6932.000	6932.000	0.000	36.265	954.000	982.067	11.296	246.667	934.000	**952.167**	9.285	277.969
25a280	9238.000	9238.000	0.000	46.097	989.000	997.833	7.299	300.465	956.000	**982.367**	11.502	350.827
25lin318	9094.000	9094.000	0.000	42.642	932.000	944.533	5.402	389.120	921.000	**930.500**	4.113	454.159
25pr439	6634.000	6634.033	0.180	89.562	847.000	860.567	6.525	912.211	829.000	**837.033**	6.595	2183.217
25pcb442	7935.000	7935.000	0.000	84.111	945.000	968.067	8.334	940.570	918.000	**935.567**	10.978	2364.059
50gil262	10490.000	10490.000	0.000	86.306	1198.000	1206.300	6.283	273.347	1195.000	**1199.733**	4.589	307.166
50a280	14543.000	14543.000	0.000	68.512	1164.000	1176.300	4.706	306.116	1156.000	**1161.100**	4.150	344.403
50lin318	9494.000	9494.000	0.000	81.971	1164.000	1173.867	5.731	801.931	1148.000	**1161.333**	6.518	476.622
50pr439	15457.000	15457.000	0.000	140.362	1157.000	1183.400	13.819	2302.168	1100.000	**1125.133**	12.865	1093.487
50pcb442	13932.000	13932.600	0.490	130.904	1211.000	1225.833	6.039	1089.128	1172.000	**1189.967**	8.292	2375.286

Table 3: Results obtained by SPMST, BSPH and SPGA on instances in Type_3_Large.

	SPMST				BSPH				SPGA			
Instance	BF	Avg	Std	Time	BF	Avg	Std	Time	BF	Avg	Std	Time
6i300	2631.000	2631.000	0.000	23.298	602.000	602.800	0.872	230.866	602.000	**602.000**	0.000	241.642
6i350	2521.000	2521.000	0.000	20.319	664.000	**664.167**	0.522	341.509	664.000	664.267	0.998	373.064
6i400	2156.000	2156.000	0.000	30.159	571.000	**574.967**	4.393	474.806	571.000	580.233	3.630	498.655
6i450	2405.000	2405.000	0.000	38.329	538.000	**538.200**	0.600	561.583	538.000	539.133	1.893	580.407
6i500	2511.000	2511.000	0.000	49.881	530.000	**533.133**	1.454	786.500	531.000	533.833	0.582	862.624
20i550	6147.000	6177.800	35.097	97.236	872.000	885.033	6.199	1771.105	834.000	**855.233**	8.682	1757.557
20i600	8070.000	8088.667	17.568	120.698	898.000	914.200	8.252	2026.978	863.000	**875.400**	6.591	2233.854
20i650	8110.000	8115.833	13.069	131.477	811.000	824.067	5.859	2312.930	762.000	**782.200**	8.953	2706.054

20i700	5897.000	5906.800	7.176	151.224	833.000	846.167	7.267	2814.103	784.000	**804.733**	8.266	3068.840
25i750	9011.000	9070.067	43.567	201.035	898.000	911.033	6.221	4078.662	841.000	**859.800**	10.750	3880.965

Table 4: Results obtained by SPMST, BSPH and SPGA on instances in Type_5_Small.

	SPMST				**BSPH**				**SPGA**			
Instance	BF	Avg	Std	Time	BF	Avg	Std	Time	BF	Avg	Std	Time
5i30-17	2023.000	2023.000	0.000	1.283	765.000	**765.000**	0.000	2.911	765.000	**765.000**	0.000	3.061
5i45-18	1643.000	1643.000	0.000	1.624	469.000	**469.000**	0.000	6.352	469.000	**469.000**	0.000	6.753
5i60-21	1020.000	1020.000	0.000	2.209	785.000	**785.000**	0.000	12.361	785.000	**785.000**	0.000	13.157
5i65-21	1670.000	1670.000	0.000	2.472	547.000	**547.000**	0.000	12.533	547.000	**547.000**	0.000	13.368
5i70-21	1694.000	1694.000	0.000	3.358	618.000	**618.000**	0.000	18.195	618.000	**618.000**	0.000	18.965
5i75-22	2518.000	2518.000	0.000	2.678	577.000	**577.000**	0.000	18.285	577.000	**577.000**	0.000	19.256
5i90-33	2179.000	2179.000	0.000	3.387	944.000	**944.000**	0.000	31.419	944.000	**944.000**	0.000	31.732
5i120-46	2834.000	2834.000	0.000	7.517	612.000	**612.000**	0.000	34.303	612.000	**612.000**	0.000	36.125
7i30-17	1480.000	1480.000	0.000	1.466	905.000	**905.000**	0.000	3.348	905.000	**905.000**	0.000	3.539
7i45-18	1070.000	1070.000	0.000	2.220	747.000	**747.000**	0.000	6.543	747.000	**747.000**	0.000	6.725
7i60-21	1370.000	1370.000	0.000	3.072	612.000	**612.000**	0.000	12.527	612.000	**612.000**	0.000	12.756
7i65-21	1922.000	1922.000	0.000	3.384	989.000	**989.000**	0.000	15.132	989.000	**989.000**	0.000	16.227
7i70-21	1556.000	1556.000	0.000	3.489	705.000	**705.000**	0.000	16.037	705.000	**705.000**	0.000	16.219
10i30-17	529.000	**529.000**	0.000	1.684	567.000	567.000	0.000	3.156	567.000	567.000	0.000	3.506
10i45-18	333.000	**333.000**	0.000	2.254	449.000	449.000	0.000	5.662	449.000	449.000	0.000	6.067
10i60-21	2821.000	2821.000	0.000	5.018	777.000	**777.000**	0.000	13.012	777.000	**777.000**	0.000	13.394
10i65-21	2147.000	2147.000	0.000	4.242	750.000	**750.000**	0.000	14.096	750.000	**750.000**	0.000	14.662
10i70-21	3998.000	3998.000	0.000	3.990	800.000	**800.000**	0.000	16.119	800.000	**800.000**	0.000	17.304
10i75-22	1489.000	1489.000	0.000	6.641	778.000	**778.000**	0.000	20.942	778.000	**778.000**	0.000	21.924
10i90-33	3584.000	3584.000	0.000	7.710	831.000	**831.000**	0.000	29.770	831.000	**831.000**	0.000	31.200
10i120-46	3873.000	3873.000	0.000	12.370	844.000	**844.000**	0.000	57.763	844.000	**844.000**	0.000	56.198

Table 5: Results obtained by SPMST, BSPH and SPGA on instances in Type_5_Large.

SPMST	**BSPH**	**SPGA**

Instance	BF	Avg	Std	Time	BF	Avg	Std	Time	BF	Avg	Std	Time
5i300-108	1891.000	1891.000	0.000	27.101	556.000	556.500	0.719	204.639	556.000	**556.333**	1.795	197.400
5i400-205	1764.000	1764.000	0.000	33.511	572.000	**572.300**	0.586	445.520	572.000	572.600	0.757	439.493
5i500-304	1257.000	1257.000	0.000	36.323	456.000	458.467	1.310	623.949	456.000	**457.100**	1.325	719.214
10i300-109	4632.000	4632.000	0.000	26.956	644.000	646.167	1.098	265.457	643.000	**644.367**	0.547	261.202
10i400-206	3791.000	3791.000	0.000	34.938	641.000	646.967	2.575	527.709	639.000	**643.567**	2.616	501.080
10i500-305	2730.000	2730.000	0.000	58.134	673.000	686.200	4.445	1013.052	673.000	**678.200**	2.868	961.736
15i300-110	5832.000	5832.000	0.000	26.472	794.000	814.900	7.213	339.357	794.000	**802.467**	8.196	363.863
15i400-207	6217.000	6217.000	0.000	52.162	883.000	898.367	6.036	671.879	869.000	**882.167**	4.852	761.643
15i500-306	4635.000	4638.000	3.768	88.949	754.000	762.967	4.270	1118.036	739.000	**747.967**	5.010	1212.155
20i300-111	5274.000	5274.000	0.000	40.747	989.000	1000.733	6.976	392.597	979.000	**984.533**	4.129	410.548
20i400-208	6044.000	6044.000	0.000	70.603	975.000	991.333	7.162	851.535	940.000	**953.733**	11.281	901.060
20i500-307	4941.000	4941.000	0.000	98.740	789.000	810.400	7.468	1286.546	765.000	**776.500**	8.433	1226.162
25i300-112	7834.000	7834.000	0.000	81.616	994.000	1003.067	4.396	404.833	981.000	**984.567**	3.084	441.376
25i400-209	8493.000	8493.033	0.180	67.729	998.000	1024.867	9.113	884.646	968.000	**987.567**	9.514	888.959
25i500-308	9667.000	9667.267	1.263	109.768	917.000	937.367	10.071	1533.057	870.000	**887.100**	9.137	1758.053

Table 6: Results obtained by SPMST, BSPH and SPGA on instances in Type_6_Small.

	SPMST				**BSPH**				**SPGA**			
Instance	BF	Avg	Std	Time	BF	Avg	Std	Time	BF	Avg	Std	Time
2lin105-2x1	592.000	592.000	0.000	2.563	439.000	**439.000**	0.000	26.954	439.000	**439.000**	0.000	29.501
4eil51-2x2	1505.000	1505.000	0.000	1.556	750.000	**750.000**	0.000	7.749	750.000	**750.000**	0.000	8.657
4berlin52-2x2	1726.000	1726.000	0.000	1.841	646.000	**646.000**	0.000	8.767	646.000	**646.000**	0.000	9.616
4pr76-2x2	1055.000	1055.000	0.000	2.397	491.000	**491.000**	0.000	16.098	491.000	**491.000**	0.000	17.256
4eil76-2x2	1583.000	1583.000	0.000	3.073	407.000	**407.000**	0.000	19.663	407.000	**407.000**	0.000	20.870
6berlin52-2x3	946.000	946.000	0.000	3.538	696.000	**696.000**	0.000	9.757	696.000	**696.000**	0.000	10.549
6st70-2x3	1595.000	1595.000	0.000	5.275	746.000	**746.000**	0.000	18.106	746.000	**746.000**	0.000	19.629
6pr76-2x3	1539.000	1539.000	0.000	5.860	758.000	**758.000**	0.000	21.593	758.000	**758.000**	0.000	24.766
8berlin52-2x4	1441.000	1441.000	0.000	3.576	823.000	**823.000**	0.000	9.694	823.000	**823.000**	0.000	11.019
9eil51-3x3	2878.000	2878.000	0.000	3.953	863.000	**863.000**	0.000	9.583	863.000	**863.000**	0.000	10.671
9st70-3x3	2287.000	2287.000	0.000	4.462	845.000	**845.000**	0.000	17.869	845.000	**845.000**	0.000	18.900
9pr76-3x3	2869.000	2869.000	0.000	4.512	900.000	**900.000**	0.000	19.725	900.000	**900.000**	0.000	21.849

9eil76-3x3	2273.000	2273.000	0.000	4.664	857.000	**857.000**	0.000	20.996	857.000	**857.000**	0.000	23.650
9eil101-3x3	1583.000	1583.000	0.000	5.384	891.000	**891.000**	0.000	37.839	891.000	**891.000**	0.000	29.242
10berlin52-2x5	2979.000	2979.000	0.000	2.935	972.000	**972.000**	0.000	10.126	972.000	**972.000**	0.000	11.679
12eil51-3x4	810.000	810.000	0.000	3.446	742.000	**742.000**	0.000	8.468	742.000	**742.000**	0.000	9.381
12st70-3x4	2621.000	2621.000	0.000	4.638	835.000	**835.000**	0.000	17.556	835.000	**835.000**	0.000	19.131
12eil76-3x4	1619.000	1619.000	0.000	5.048	740.000	**740.000**	0.000	18.637	740.000	**740.000**	0.000	20.807
12pr76-3x4	2092.000	2092.000	0.000	5.262	614.000	**614.000**	0.000	18.950	614.000	**614.000**	0.000	20.746
15pr76-3x5	2037.000	2037.000	0.000	6.422	695.000	**695.000**	0.000	19.358	695.000	**695.000**	0.000	21.545
16eil51-4x4	1964.000	1964.000	0.000	4.176	555.000	**555.000**	0.000	9.083	555.000	**555.000**	0.000	10.100
16st70-4x4	3044.000	3044.000	0.000	5.501	909.000	**909.000**	0.000	17.007	909.000	**909.000**	0.000	18.855
16eil76-4x4	1415.000	1415.000	0.000	5.975	771.000	**771.000**	0.000	19.601	771.000	**771.000**	0.000	22.516
16lin105-4x4	3280.000	3280.000	0.000	8.737	1093.000	**1093.000**	0.000	40.789	1093.000	**1093.000**	0.000	46.250
18pr76-3x6	2767.000	2767.000	0.000	5.993	1161.000	**1161.000**	0.000	21.194	1161.000	**1161.000**	0.000	23.451
20eil51-4x5	3731.000	3731.000	0.000	6.193	875.000	**875.000**	0.000	9.859	875.000	**875.000**	0.000	10.763
20st70-4x5	4322.000	4322.000	0.000	6.141	1325.000	**1325.000**	0.000	18.776	1325.000	**1325.000**	0.000	20.045
20eil76-4x5	2398.000	2398.000	0.000	6.687	965.000	**965.000**	0.000	19.866	965.000	**965.000**	0.000	22.626
25eil51-5x5	1798.000	1798.000	0.000	7.451	988.000	**988.000**	0.000	10.741	988.000	**988.000**	0.000	11.683
25eil76-5x5	4425.000	4425.000	0.000	10.576	1089.000	**1089.000**	0.000	20.756	1089.000	**1089.000**	0.000	22.033
25rat99-5x5	5143.000	5143.000	0.000	14.388	961.000	**961.000**	0.000	34.351	961.000	**961.000**	0.000	38.416
25eil101-5x5	3972.000	3972.000	0.000	15.667	1320.000	**1320.000**	0.000	41.285	1320.000	**1320.000**	0.000	45.870
28kroA100-4x7	4659.000	4659.000	0.000	10.074	1035.000	**1035.000**	0.000	36.373	1035.000	**1035.000**	0.000	39.901
30kroB100-5x6	2709.000	2709.000	0.000	18.179	1080.000	**1080.000**	0.000	35.467	1080.000	**1080.000**	0.000	39.222
35kroB100-5x5	3552.000	3552.000	0.000	15.163	1179.000	**1179.000**	0.000	29.370	1179.000	**1179.000**	0.000	43.485
36eil101-6x6	4094.000	4094.000	0.000	12.512	1063.000	**1063.000**	0.000	38.014	1063.000	**1063.000**	0.000	42.763
42rat99-6x7	5288.000	5288.000	0.000	20.753	1264.000	**1264.000**	0.000	36.137	1264.000	**1264.000**	0.000	40.279

Table 7: Results obtained by SPMST, BSPH and SPGA on instances in Type_6_Large.

	SPMST				**BSPH**				**SPGA**			
Instance	BF	Avg	Std	Time	BF	Avg	Std	Time	BF	Avg	Std	Time
9gil262-3x3	3026.000	3026.000	0.000	16.265	704.000	705.867	0.499	181.501	704.000	**704.933**	0.998	213.229
9a280-3x3	3419.000	3419.000	0.000	18.486	709.000	712.500	2.419	279.955	709.000	**709.100**	0.539	637.465
9lin318-3x3	2716.000	2716.000	0.000	19.905	742.000	757.067	5.709	315.022	741.000	**750.467**	5.252	360.749
9pr439-3x3	1956.000	1956.000	0.000	77.835	548.000	549.667	1.535	612.453	548.000	**548.267**	1.123	724.352
9pcb442-3x3	2475.000	2475.000	0.000	53.734	575.000	580.433	2.276	689.842	573.000	**574.867**	1.284	827.364

18pr439-3x6	6017.000	6017.000	0.000	77.563	692.000	705.633	4.593	1528.109	682.000	**687.200**	3.070	1389.903
20pr439-4x5	5315.000	5315.000	0.000	96.279	674.000	681.867	3.956	1585.656	668.000	**670.033**	1.906	674.259
25gil262-5x5	5760.000	5760.000	0.000	36.457	925.000	935.433	4.544	259.140	906.000	**913.767**	7.986	250.576
25a280-5x5	6148.000	6148.000	0.000	43.589	986.000	998.600	6.092	318.408	973.000	**990.933**	5.452	647.201
25lin318-5x5	7952.000	7952.000	0.000	54.356	998.000	1009.233	5.207	426.355	977.000	**985.900**	8.158	418.245
25pcb442-5x5	7203.000	7203.000	0.000	74.548	894.000	912.067	7.398	2264.957	856.000	**877.300**	10.264	2305.040
36pcb442-6x6	10355.000	10355.467	0.499	113.129	1075.000	1107.133	11.829	1123.131	1038.000	**1065.100**	11.250	2482.785
42a280-6x7	6459.000	6459.000	0.000	62.341	1106.000	1126.900	9.278	308.694	1080.000	**1091.267**	8.668	337.389
49gil262-7x7	10857.000	10857.000	0.000	61.193	1159.000	1167.533	5.696	368.672	1159.000	**1161.500**	4.272	397.933
49lin318-7x7	10097.000	10097.000	0.000	80.091	1304.000	1317.000	6.377	423.700	1284.000	**1296.067**	5.994	474.788

References

[1] Prömel, H.J., Steger, A.: The steiner tree problem a tour through graphs, algorithms, and complexity

[2] Wu, B.Y., Lin, C.W.: On the clustered steiner tree problem. Journal of Combinatorial Optimization **30**, 370–386 (2015). https://doi.org/10.1007/s10878-014-9772-7

[3] Chisman, J.A.: The clustered traveling salesman problem. Computers & Operations Research **2**(2), 115–119 (1975)

[4] D'Emidio, M., Forlizzi, L., Frigioni, D., Leucci, S., Proietti, G.: On the clustered shortest-path tree problem. In: ICTCS, pp. 263–268 (2016)

[5] Lin, C.-W., Wu, B.Y.: On the minimum routing cost clustered tree problem. Journal of Combinatorial Optimization **33**(3), 1106–1121 (2017)

[6] Menéndez, H.D., Barrero, D.F., Camacho, D.: A genetic graph-based approach for partitional clustering. International journal of neural systems **24**(03), 1430008 (2014)

[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 **12**(4), 459–465 (2007)

[8] 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. In: 2017 IEEE Symposium on Computers and Communications (ISCC), pp. 316–321 (2017). https://doi.org/10.1109/ISCC.2017.8024549

[9] Haghighat, A.T., Faez, K., Dehghan, M., Mowlaei, A., Ghahremani, Y.: A genetic algorithm for steiner tree optimization with multiple constraints using prüfer number. In: EurAsia-ICT 2002: Information and Communication Technology, pp. 272–280 (2002). https://doi.org/10.1007/3-540-36087-5_32

[10] Hesser, J., Männer, R., Stucky, O.: Optimization of steiner trees using genetic algorithms. In: Third International Conference on Genetic Algorithms, pp. 231–236 (1989)

[11] Julstrom, B.: A genetic algorithm for the rectilinear steiner problem. In: 5th International Conference on Genetic Algorithms, pp. 474–480 (1993)

[12] Kapsalis, A., Rayward-Smith, V., Smith, G.: Solving the graphical steiner tree problem using genetic algorithms. Journal of the Operational

Research Society **44** (1993). https://doi.org/10.1038/sj/jors/0440408

[13] Esbensen, H.: Computing near-optimal solutions to the steiner problem in a graph using a genetic algorithm. Networks **26** (1995). https://doi.org/10.1002/net.3230260403

[14] Mitchell, M.: An Introduction to Genetic Algorithms. MIT Press, Cambridge, (1998). https://doi.org/10.7551/mitpress/3927.001.0001

[15] D'Emidio, M., Forlizzi, L., Frigioni, D., Leucci, S., Proietti, G.: Hardness, approximability, and fixed-parameter tractability of the clustered shortest-path tree problem. Journal of Combinatorial Optimization **38**, 165–184 (2018). https://doi.org/10.1007/s10878-018-00374-x

[16] Thanh, P.D., Binh, H.T.T., Trung, T.B.: An efficient strategy for using multifactorial optimization to solve the clustered shortest path tree problem. Applied Intelligence **50**(4), 1233–1258 (2020)

[17] Thanh, P.D., Binh, H.T.T., Long, N.B., *et al.*: A heuristic based on randomized greedy algorithms for the clustered shortest-path tree problem. In: 2019 IEEE Congress on Evolutionary Computation (CEC), pp. 2915–2922 (2019). IEEE

[18] Binh, H.T.T., Thanh, P.D., Trung, T.B., *et al.*: Effective multifactorial evolutionary algorithm for solving the cluster shortest path tree problem. In: 2018 IEEE Congress on Evolutionary Computation (CEC), pp. 1–8 (2018). IEEE

[19] Thanh, P.D., Dung, D.A., Tien, T.N., Binh, H.T.T.: An effective representation scheme in multifactorial evolutionary algorithm for solving cluster shortest-path tree problem. In: 2018 IEEE Congress on Evolutionary Computation (CEC), pp. 1–8 (2018). IEEE

[20] Binh, H.T.T., Thanh, P.D., Thang, T.B.: New approach to solving the clustered shortest-path tree problem based on reducing the search space of evolutionary algorithm. Knowledge-Based Systems **180**, 12–25 (2019). https://doi.org/10.1016/j.knosys.2019.05.015

[21] Lin, C.W., Wu, B.Y.: On the minimum routing cost clustered tree problem. Journal of Combinatorial Optimization **33**, 1106–1121 (2017). https://doi.org/10.1007/s10878-016-0026-8

[22] Trung, T., Thanh, L., Hieu, L., Pham Dinh, T., Binh, H.: Multifactorial evolutionary algorithm for clustered minimum routing cost problem, pp. 170–177 (2019). https://doi.org/10.1145/3368926.3369712

[23] 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 **105**, 107253 (2021)

[24] Pham Dinh, T.: Cluspt instances,mendeley data, v3 (2019). https://doi.org/10.17632/b4gcgybvt6.3

[25] Carrasco, J., García, S., Rueda, M.M., Das, S., Herrera, F.: Recent trends in the use of statistical tests for comparing swarm and evolutionary computing algorithms: Practical guidelines and a critical review. Swarm and Evolutionary Computation, 100665 (2020)

[26] 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 **1**(1), 3–18 (2011)

[27] D. Stakic, M.Z., Anokic, A.: A reduced variable neighborhood search approach to the heterogeneous vector bin packing problem. ICT **50**(4), 808–826 (2021)

[28] BAN, H.-B.: Applying metaheuristic for time-dependent traveling salesman problem in postdisaster. International Journal of Computational Intelligence Systems **14**(1), 1087–1107 (2021)

[29] BAN, H.-B.: A metaheuristic for the delivery man problem with time windows. Journal of Combinatorial Optimization **41**(4), 794–816 (2021)