CARPATHIAN J. MATH. Online version at https://www.carpathian.cunbm.utcluj.ro
36 (2020), No. 3, 401 - 414 Print Edition: ISSN 1584 - 2851 Online Edition: ISSN 1843 - 4401

A novel genetic algorithm for solving the clustered shortest-path tree problem

OVIDIU COSMA, PETRICĂ C. POP and IOANA ZELINA

ABSTRACT. The clustered shortest-path tree problem is an extension of the classical single-source shortest-path problem, in which, given a graph with the set of nodes divided into a predefined, mutually exclusive and exhaustive set of clusters, we want to determine a shortest-path spanning tree from a given source to all the other nodes of the graph, with the property that each cluster should induce a connected subtree. The investigated problem proved to be NP-hard and therefore we proposed an efficient genetic algorithm in order to solve it. The preliminary computational results reported on a set of benchmark instances from the literature proved that our proposed solution approach yields high-quality solutions within reasonable running times.

1. INTRODUCTION

The clustered shortest-path tree problem (CluSPTP) generalizes the classical single-source shortest-path problem and looks for a spanning tree of a given graph with the property that all the sub-graphs induced by each of the clusters are connected and such that the total cost of the paths from a given source node to all the other nodes of the graph is minimized.

The current literature is rather scarce. The problem was introduced by D'Emidio et al. [4] justified by some practical applications in communication networks. The same authors, in an extended version of their paper [5], investigated the computational hardness and provided some approximation results for both cases of the problem: unweighted and weighted. Binh et al. [1] and Thanh et al. [17] presented two multifactorial evolutionary algorithms that use different ways to encode feasible solutions of the CluSPTP: one based on the Cayley code and the other one using an edge set representation. Thanh et al. [18] described a random heuristic search algorithm that combines a randomized greedy algorithm with a shortest path tree algorithm. Recently, Binh et al. [2] proposed a solution approach based on the reduction of the solution space of a genetic algorithm by decomposing the CluSPTP into two smaller sub-problems which are solved separately.

The clustered shortest-path tree problem belongs to the class of generalized combinatorial optimization problems. This category of problems naturally generalizes the classical combinatorial optimization problems, having the following primary features: the nodes of the underlying graph are partitioned into a certain number of clusters and, when considering the feasibility constraints of the initial problem, these are expressed in relation to the clusters rather than as individual nodes. A closely related problem to CluSPTP was introduced by Myung et al. [10] and was called the Generalized Minimum Spanning Tree Problem, whose objective is to find a minimum cost tree spanning a subset of nodes that includes exactly one node from each cluster. For more information regarding the generalized minimum spanning tree problem and its variants, we refer to Pop et al. [14, 16]. Some

Received: 27.03.2020. In revised form: 01.07.2020. Accepted: 07.07.2020
2010 *Mathematics Subject Classification.* 05C85, 68T20.
Key words and phrases. *single-source shortest-path problem, clustered shortest-path tree problem, genetic algorithms.*
Corresponding author: Petrică C. Pop; petrica.pop@cunbm.utcluj.ro

other generalized combinatorial optimization problems that have been investigated, are: the generalized traveling salesman problem and its variants [7, 13], the generalized vehicle routing problem and its variants [8, 11, 15], the selective graph coloring problem [3, 6], etc. For further reference on the class of generalized combinatorial optimization problems we refer to [12].

In this paper, we propose a novel genetic algorithm for solving the general CluSPTP and an exact algorithm that solves efficiently the euclidean instances defined on complete graphs. The proposed algorithms outperform the ones existing in literature in terms of speed and accuracy.

The present paper is organized as follows: the second section provides a formal definition of the clustered shortest-path tree problem, Section III presents the novel solution approach based on genetic algorithms. The next section (Section IV) provides a comparative analysis of the performance of our proposed genetic algorithm with the existing solution approaches from the literature, while in Section V some concluding results, as well as further research directions are presented.

2. Definition of the clustered shortest-path tree problem

We consider an undirected connected graph $G = (V, E)$ with the set of nodes $V = \{v_1, v_2, ..., v_n\}$ and the set of edges $E,\ E \subseteq \{\{v_i, v_j\} |\ v_i, v_j \in V,\ i \neq j \in \{1, 2, ..., n\}\}$.

The set of nodes V is partitioned into k mutually exclusive nonempty subsets denoted $C_1, ..., C_k$ and called *clusters*. The following conditions hold:

1. $V = C_1 \cup C_2 \cup ... \cup C_k$
2. $C_l \cap C_p = \emptyset$ for all $l, p \in \{1, ..., k\}$ and $l \neq p$.

The edges of the graph are classified into two categories: edges which connect vertices belonging to the same cluster, called intra-cluster edges and edges which connect vertices belonging to different clusters, called inter-cluster edges. In addition we define a cost function $c : E \to R_+$ which attaches to each edge $e \in E$ a positive cost c_e.

If S is a subset of nodes, $S \subseteq V$, then by $G[S]$ we will denote the subgraph induced by S. Given a spanning tree T of the graph G and two nodes $v_i, v_j \in V$, the length of the shortest path between v_i and v_j will be denoted by $d_T(v_i, v_j)$. Given a source node $s \in V$, we will denote by $\sum\limits_{v \in V} d_T(s, v)$ the total cost of the paths from the given source node s to all the other nodes of the graph.

The *clustered shortest-path tree problem* aims for finding a tree T with the following properties:

1. T spans all the nodes of the graph G;
2. The induced subgraphs $T[C_i]$, $\forall i \in \{1, ..., k\}$ are connected.

and such that the total cost of the paths from a given source node to all the other nodes of the graph is minimized, i.e.

$$\sum_{v \in V} d_T(s, v) \to \min . \tag{2.1}$$

In Figure 1 the CluSPTP defined on an undirected graph with 19 vertices divided into 6 clusters is illustrated. A feasible solution of the problem is presented in Figure 9.

Our genetic algorithm solves the general CluSPTP which is $\mathcal{NP}$*-hard*, but there exist four particular cases in which the problem can be solved differently, obtaining the exact solution efficiently, in polynomial time. We propose as well an exact algorithm for the particular case when the graph is euclidean and complete.

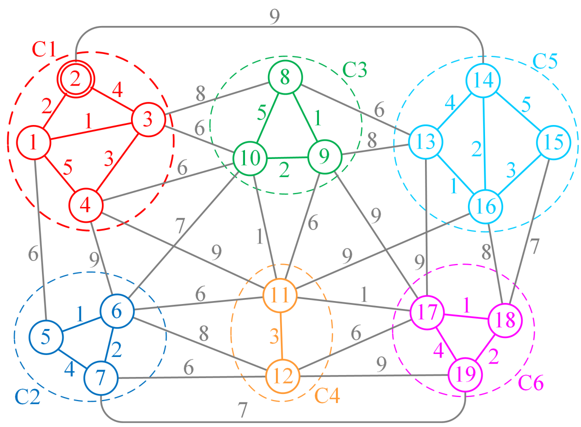

FIGURE 1. CluSPTP instance example

1. Considering $|V_i| = 1$ for all $i \in \{1, ..., k\}$, the CluSPTP is trivially reduced to the classical single-source shortest-path problem, which can be solved in polynomial time.
2. If $k = 1$, then the problem is the classical single-source shortest-path problem for the nodes belonging to the only existing cluster.
3. Considering that the number of clusters k is fixed then the CluSPTP can be solved in polynomial time (in the number of nodes n). In this case, a polynomial time procedure which solves the problem, based on dynamic programming, can be developed quite easily.
4. When the CluSPTP is defined on complete and euclidean graphs, the problem can be solved optimally in polynomial time. In this case, the shortest path between two nodes in the graph G is always the edge that connects them, so $d_G(v, u) = c_{v,u}$, where $c_{v,u}$ is the cost of the edge $\{v, u\} \in E$ (we consider $c_{v,v} = 0, v \in V$). The optimal solution for such a graph is a rooted tree that connects directly the root (source) node of the graph to the root node of each cluster in the graph, and all the nodes within each cluster are directly linked to the root node of the cluster, such that the total cost of the paths from a given source node to all the other nodes of the graph is minimized. The optimal solution can be obtained using a greedy algorithm to determine the source node for each cluster. If $s \in C_r$ is the root of the spanning tree and s_i is the root of C_i, $i \in \{1, 2, ..., k\} \setminus \{r\}$, the cost of reaching the nodes in C_i from s in the spanning tree is

$$cost_i = c_{s,s_i} \cdot |C_i| + \sum_{v \in C_i} c_{s_i,v}. \tag{2.2}$$

The optimal solution is obtained when all $cost_i$ are minimized

$$TC = \sum_{i=1}^{k} \min_{u \in C_i} \{ |C_i| \cdot c_{s,u} + \sum_{v \in C_i} c_{u,v} \}. \tag{2.3}$$

and can be efficiently found using a greedy algorithm as follows:
 a) $TC = cost_r$.

b) For each $i \in \{1, 2, ..., k\} \setminus \{r\}$
choose $s_i \in C_i$ that minimizes $cost_i$;
$TC = TC + cost_i$.

If the source node s is not given, only the root cluster C_r, we choose the minimum value of TC obtained for every node $v \in C_r$.

D'Emidio et al. [5] showed that in general the CluSPTP is $\mathcal{NP}$-*hard*, that is why in order to solve the investigated problem we propose an efficient genetic algorithm.

3. Description of the proposed genetic algorithm

In this section, we present our proposed genetic algorithm whose main feature is an innovative representation scheme that enables us to construct easily feasible CluSPTP solutions and to explore efficiently the solution space of the problem.

3.1. **The chromosome structure.** Let $G = (V, E)$ be the considered graph, as described in Section II. The genes of a chromosome contain a complete set of inter-cluster edges, one for each pair of clusters. Thus, for an instance with k clusters, the total number of genes that define a chromosome is $k \times (k-1)/2$. The gene corresponding to the pair of clusters C_x, C_y will be denoted g_{xy} for all $x > y$, $x, y \in \{1, .., k\}$. The gene g_{xy} corresponds to an edge between clusters C_x and C_y, if there is at least an edge $\{v_i, v_j\} \in E$, $v_i \in C_x$, $v_j \in C_y$, $x > y$, otherwise the gene g_{xy} is void.

The genes of a chromosome can be stored in a triangular array having the structure shown in Figure 2. The gene that connects the clusters C_x and C_y with $x > y$, is found on line x and column y in the genes array.

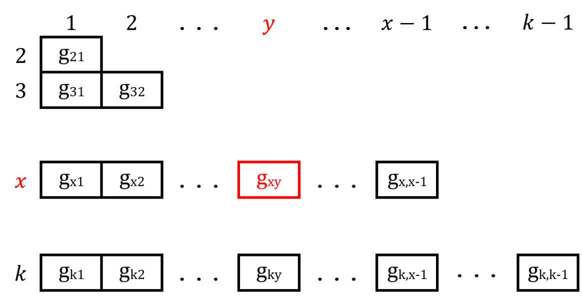

FIGURE 2. The structure of a chromosome gene array

In our GA, a chromosome A is defined as follows:

$$A = \{g_{xy} \ , \ x \in \{1, ..., k\} \ , \ y \in \{1, ..., k\} \ , \ x > y\}$$

Such a chromosome defines a subgraph $G_A = (V, A)$ of G, where the set of edges corresponds to the set of genes.

In Figure 3, we illustrate a chromosome gene array corresponding to the CluSPTP instance in Figure 1. The subgraph defined by the chromosome in Figure 3 is presented in Figure 4. This subgraph corresponds to a CluSTSP subproblem, in which the graph G is replaced by G_A.

x \ y	1	2	3	4	5
2	5, 1				
3	10, 3	10, 6			
4	11, 4	12, 6	11, 10		
5	14, 2	-	13, 9	16, 11	
6	-	19, 7	17, 9	17, 11	18, 15

FIGURE 3. A chromosome gene array for the instance in Figure 1

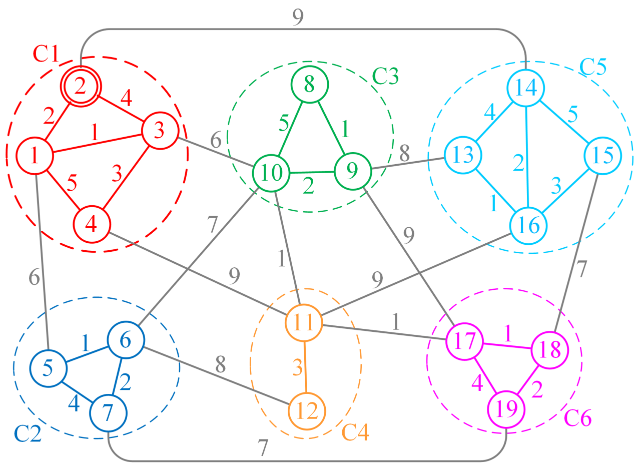

FIGURE 4. The subgraph defined by the chromosome illustrated in Figure 3

3.2. **Solving the CluSTSP subproblem.** We solve the CluSTSP subproblem defined by G_A, the subgraph built using chromosome A, using an efficient heuristic algorithm in five steps that speculates the fact that any two clusters of the instance are connected by at most an edge.

The first step of the algorithm is to build a skeleton S_A of the G_A subgraph in which every cluster is reduced to a single node. The skeleton of the subgraph in Figure 4 is shown in Figure 5. The source node of the skeleton is the node corresponding to the cluster that contains the source node of the instance. We will call this cluster the *source cluster*.

The second step is running the Shortest Path First (SPF) algorithm on the skeleton. The SPF algorithm produces a spanning tree that contains the optimal inter-cluster routes. The result of applying the SPF algorithm on the skeleton in Figure 5 is shown in Figure 6. This spanning tree is kept in a parent array that has k elements. The parent array for the tree in Figure 6 is shown in Figure 7.

In the third step, the source nodes for each of the clusters are determined, using the parent array $Parent$ of the skeleton tree and the genes array, as follows:

- The source node of the source cluster is the source node of the instance.
- For finding the source node of cluster C_b, its parent C_a, $a = Parent[b]$ is considered in the parent array of the skeleton tree. The source node of cluster C_b is the

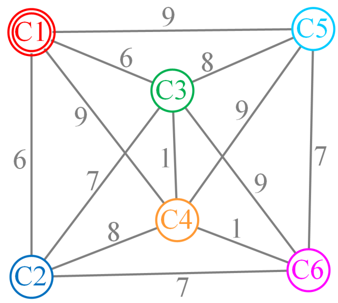

FIGURE 5. The skeleton of the subgraph in Figure 4

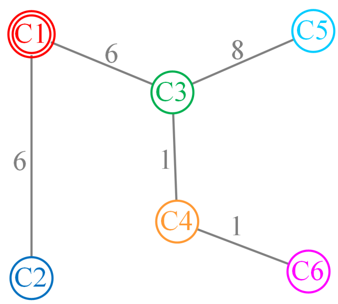

FIGURE 6. Spanning tree of the skeleton shown in Figure 5

1	2	3	4	5	6
-	1	1	3	3	4

FIGURE 7. The parent array for the tree in Figure 6

extremity in C_b of the edge represented by the gene g_{xy} with $x \neq y \in \{a, b\}$ in the chromosome gene array.

- If $b > a$ the gene is found on line b and column a of the chromosome gene array, otherwise the gene is found on line a and column b.

The result of this step is shown in Figure 8. All inter-cluster edges of the instance graph have been removed except those appearing in the skeleton tree in Figure 6. The source nodes in each cluster are represented with double line.

In the fourth step, the SPF algorithm is run for each cluster, thus the spanning trees inside the clusters are determined. A spanning tree for the entire instance is generated by connecting the cluster spanning trees with the edges of the skeleton tree. This instance spanning tree depends on the genes of the chromosome and satisfies the conditions of the CluSPTP.

For the instance in Figure 1, the spanning tree generated using the chromosome gene array from Figure 3 is shown in Figure 9.

The final step determines the total cost of the solution, TC, using the following relation on the instance spanning tree:

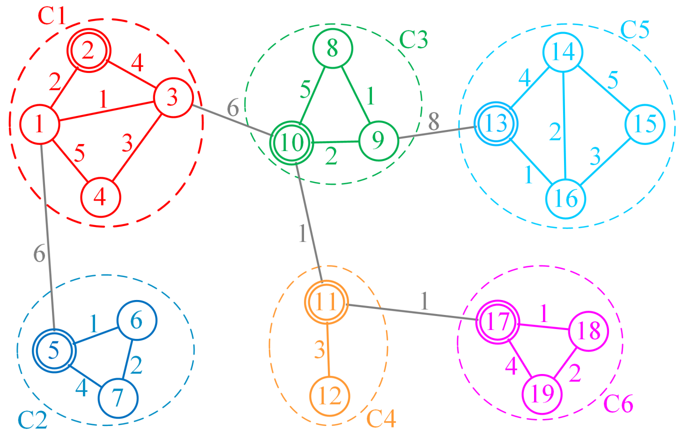

FIGURE 8. Inter-cluster tree of the subgraph in Figure 4

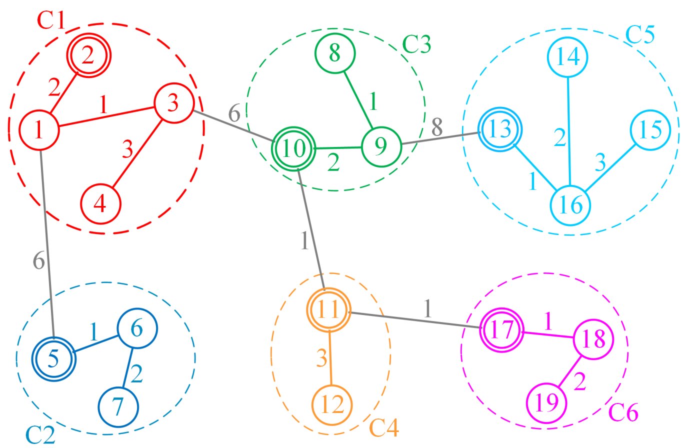

FIGURE 9. The spanning tree of the instance from Figure 1 generated using the chromosome from Figure 3

$$TC = \sum_{x=1}^{k} \left(\, |C_x| \cdot d_T(s, s_x) + cl_x \, \right), \tag{3.4}$$

where $|C_x|$ is the number of nodes in cluster C_x, s_x is the source node of cluster C_x and cl_x is the total cost of the routes inside cluster C_x. We will name cl_x the *total internal cost* of cluster C_x and we have that

$$cl_x = \sum_{v \in C_x} d_T(s_x, v) \tag{3.5}$$

For the instance illustrated in Figure 9, the values of the operands in the formula of the total cost are shown in Figure 10.

x	$\|C_x\|$	s_x	cl_x	$d_T(s, s_x)$
1	4	2	11	0
2	3	5	4	8
3	3	10	5	9
4	2	11	3	10
5	4	13	8	19
6	3	17	4	11

FIGURE 10. The costs of the solution presented in Figure 9

3.3. **Efficiency issues.** Since the optimization process may require the evaluation of a large number of chromosoms, the algorithm should avoid repeating the same operations. Thus, it is preferable to run the SPF algorithm within each cluster, for each possible source node, in the initialization phase of the algorithm, and to keep the results in a bi-dimensional array at cluster level. This operation performed for cluster 5 of the instance in Figure 1 is depicted in Figure 11, and the results are shown in Figure 12.1.

The costs of the routes from each node to the source node of the cluster they belong to, can be also evaluated only once, in the initialization phase, and kept in an array of costs at cluster level. The main diagonal of these arrays keeps the total internal costs of the clusters. The costs array for cluster 5 of the instance in Figure 1 is shown in Figure 12.2.

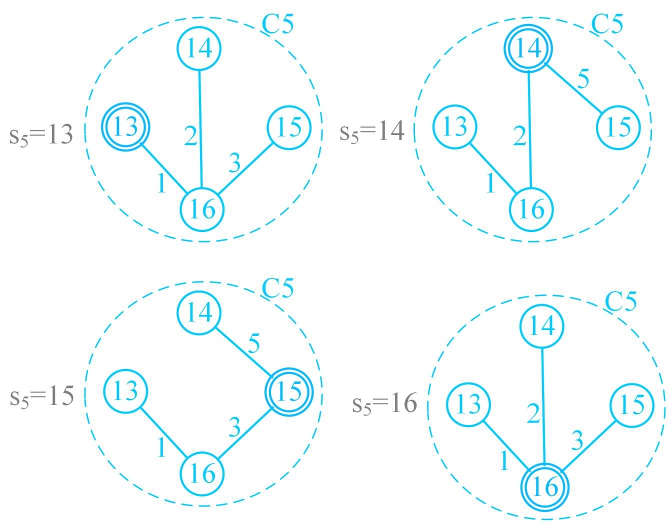

FIGURE 11. Spanning trees for cluster 5 of the instance from Figure 1

3.4. **Initial population.** The initial population is composed of random chromosomes. The genes array of these chromosomes are created element-by-element as follows: the element on line x and column y, $x > y$ is a randomly chosen edge from the instance, edge that connects a node in cluster C_x with a node in cluster C_y. If the instance does not contain such an edge, then this gene will be void. This generating mechanism has the advantage that it creates only valid chromosomes that can be used to create valid solutions of the CluSPTP.

C_5 nodes

s_5	13	14	15	16
13	-	16	16	13
14	16	-	14	14
15	16	15	-	15
16	16	16	16	-

1

C_5 nodes

s_5	13	14	15	16
13	8	3	4	1
14	3	10	5	2
15	4	5	12	3
16	1	2	3	6

2

FIGURE 12. Parent array and costs array associated to cluster 5 in the case of the instance presented in Figure 1

The fitness of each new created chromosome throughout the optimization process is evaluated by solving the CluSPTP subproblem defined by its genes.

We will denote by D the dimension of the current population. The number of chromosomes generated for the initial population is $3 \times D$. The initial population is processed by the selection mechanism, resulting the current population.

3.5. **Crossover.** The crossover mechanism selects from the current population two parents $p1$ and $p2$, which are used to create an offspring. The first parent is allways chosen randomly from the best 20% chromosomes in the current population, and the second parent is chosen randomly from the entire population. This is a combination between elitist and random selection strategies. The genes of the offspring are selected, according to the uniform crossover mechanism, either from $p1$ or from $p2$ with equal probabilitiers.

The number of crossover operations performed for completing a new generation of chromosomes is $3 \times D$. The new generation of chromosomes is processed by the selection mechanism, resulting a new current population.

3.6. **Selection.** The selection mechanism merges the newly created population with the current population, removes the duplicates, then sorts the resulting population by fitness value. Then the best D chromosomes are selected for the new current population. All the rest are discarded.

3.7. **Mutation.** We used uniform mutation. The mutation operator randomly selects one of the chromosome genes and replaces it with another edge that connects nodes from the same two clusters as the original gene. If the original gene is void or there is a single edge between the two clusters, then the mutation operator ends and the chromosome remains unchanged.

Typically, a mutation operator performs significant changes to the chromosome data, but is applied with a low probability. Our proposed mutation operator performs small changes to the chromosomes and there is a good probability that these changes do not affect in any way the built CluSPTP solutions. For this reason, we apply the mutation operator to each new chromosome created by the crossover mechanism. This way, the diversity of the generated chromosomes is improved.

3.8. **Genetic parameters.** The genetic parameters have an important impact on the performance of the GAs. That is why in our developed GA the values of the parameters have been chosen based on computational experiments and statistical analysis. The parameters have been chosen as follows: the dimension of the current population D ranges between 1500 and 5000, depending on instance dimensions, the initial population contains $3 \times D$ individuals, the algorithm is stopped when the best known solution is not improved over the last 30 generations of chromosomes, the number of crossover operations performed for completing each new generation of chromosomes is $3 \times D$ and the mutation probability is 0.5.

4. Computational results

This section contains the preliminary computational results achieved by our novel solution approach. In order to asses the performance of the proposed genetic algorithm, we tested our solution approach on two sets of instances: one that contains euclidean instances and the other one containing non-euclidean instances. We must point out that all the existing benchmark instances from the literature are euclidean and defined on complete graphs and therefore can be solved optimally by the greedy algorithm described in Chapter 2. For testing the performance of our proposed GA, we compared it to the existing state-of-the-art algorithm for solving the CluSPTP, the evolutionary algorithm developed by Binh et al. [2]. Our proposed algorithms: the exact polynomial time algorithm for euclidean and complete graphs described in Chapter 2 and the genetic algorithm were implemented in Java 8 and have been tested on a PC with Intel Core i5-4590 3.3GHz, 16GB RAM, Windows 10 Education 64 bit operating system. In our GA for each instance we carried out 10 independent trials.

4.1. **Computational results on euclidean instances.** In the case of euclidean instances defined on complete graphs, we tested the performance of our proposed GA on a set of 40 benchmark instances from the total set of 250 euclidean instances generated by Binh et al. [2]. In addition we delivered the optimal solutions obtained by the described exact algorithm. The previously mentioned instances are based on the MOM-lib provided by Mestria et al. [9] in the case of the Clustered Traveling Salesman Problem. The MOM-lib contains six kinds of instances which were obtained using different algorithms, see for more details [9] and classified into two groups according to the dimension: small instances and large instances. The instances used in our computational experiments belong to the Type 1 category of instances and have the following characteristics: the small euclidean instances contain between 51 and 105 nodes partitioned within a number of clusters ranging from 10 to 50 and the large instances contain between 262 and 1379 nodes partitioned within a number of clusters ranging from 10 to 100. The source node was selected randomly for each of the considered instances.

Tables 1 – 2 display the optimal solutions achieved by our exact algorithm, the results obtained by our GA for solving the considered instances of the CluSPTP and in addition the reported results by Binh et al. [2] for solving the problem with their evolutionary algorithm. The first two columns indicate the number of the instance and its name, the third and the fourth column show the cost of the optimal solutions achieved by our exact algorithm and the necessary computational times in seconds in order to achieve them, the next three columns contain the best and average solutions obtained by the evolutionary algorithm developed by Binh et al. [2] and the necessary average computational times reported in minutes in order to achieve the corresponding solutions and the last columns contain the best and average solutions obtained by our proposed GA and the necessary average computational times reported in minutes in order to achieve the corresponding solutions. The simbol "-" means that the corresponding results were not provided by Binh et al. [2]. The winning result among the one published in [2] and the proposed is marked with bold font.

Analyzing the computational results displayed in Table 1, one can notice that: the exact algorithm delivered the optimal solution in less than 1 millisecond; the evolutionary algorithm developed by Binh et al. [2] provided sub-optimal solutions within at most 0.08 minutes, but did not obtain in any of the instances the optimal solution and our proposed solution approach obtained the optimal solutions in 14 out of 20 small instances of Type

TABLE 1. Experimental results in the case of small euclidean instances of Type 1

Instance		Our Exact Algorithm		Evolutionary Algorithm [2]			Our Genetic Algorithm		
No.	Size	Optimal solution	Time (seconds)	Best solution	Average solution	Time (minutes)	Best solution	Average solution	Time (minutes)
1.	10berlin52	43724.0	< 0.001	43954.0	44237.6	0.02	**43724.0**	**43724.0**	0.02
2.	10eil51	1713.2	< 0.001	1741.5	1770.6	0.02	**1713.2**	**1713.2**	0.02
3.	10eil76	2203.2	< 0.001	2264.5	2315.6	0.02	**2203.2**	**2203.2**	0.02
4.	10kroB100	140522.2	< 0.001	143108.6	147539.7	0.02	**140522.2**	**140522.2**	0.02
5.	10rat99	7520.2	< 0.001	7697.8	7899.4	0.02	**7520.2**	**7520.2**	0.02
6.	10st70	3095.2	< 0.001	3098.7	3191.1	0.02	**3095.2**	**3095.2**	0.02
7.	15berlin52	26311.9	< 0.001	26463.1	26867.8	**0.03**	**26311.9**	**26311.9**	0.08
8.	15eil51	1306.4	< 0.001	1313.4	1336.5	0.03	**1306.4**	**1306.4**	0.03
9.	15eil76	2909.0	< 0.001	2955.3	3047.8	**0.03**	**2909.0**	**2909.0**	0.08
10.	15pr76	704600.5	< 0.001	714652.2	728128.0	**0.03**	**704600.5**	**704600.5**	0.07
11.	15st70	4120.0	< 0.001	4145.8	4230.1	**0.03**	**4120.0**	**4120.0**	0.06
12.	25eil101	4678.9	< 0.001	4826.6	4885.5	**0.03**	**4678.9**	**4687.3**	0.68
13.	25kroA100	147195.0	< 0.001	150157.7	153155.6	**0.03**	**147195.0**	**147195.0**	0.50
14.	25lin105	97944.7	< 0.001	98991.8	100615.8	**0.03**	**97957.0**	**97958.1**	0.54
15.	25rat99	6841.4	< 0.001	7056.0	7162.3	**0.03**	**6841.4**	**6841.4**	0.64
16.	50eil101	3825.2	< 0.001	3890.7	3919.7	**0.07**	**3834.8**	**3839.8**	6.62
17.	50kroA100	159647.2	< 0.001	160547.4	161889.6	**0.07**	**160479.2**	**160522.1**	4.89
18.	50kroB100	133104.5	< 0.001	134077.5	135332.2	**0.07**	**133104.5**	**133104.5**	1.57
19.	50lin105	145829.0	< 0.001	146367.1	147175.4	**0.07**	**146130.5**	**146284.2**	8.83
20.	50rat99	8007.4	< 0.001	8104.5	8132.4	**0.08**	**8007.4**	**8011.6**	7.85

1. Our novel GA provided the optimal solution in all the ten runs in 15 out of 20 instances. Our algorithm outperforms the evolutionary algorithm developed by Binh et al. [2] from the point of the quality of the achieved solutions: providing better solutions and smaller average percentage gaps in comparison to the evolutionary algorithm developed by Binh et al. [2] for each of the 20 instances. The computation times of our algorithm are similar to those of Binh et al. [2] in 7 out of 20 instances. Our algorithm needed longer computation times for the other instances, but that is explicable because our algorithm found better solutions for those instances and the algorithm proposed in [2] explores only a tiny subspace of the solutions space, in which the root node of each cluster is directly connected to the root nodes of all descendants, while our GA explores the entire solution space.

When taking a closer look at the computational results shown in Table 2, we can observe that: the exact algorithm delivered the optimal solution in less than 1 millisecond; the evolutionary algorithm developed by Binh et al. [2] has been able to solve only the first 14 instances providing sub-optimal solutions within at most 0.10 minutes, but did not obtain in none of the instances the optimal solution and our proposed solution approach obtained the optimal solutions in 10 out of 20 instances, in 5 of these instances being able to provide the optimal solution in all the ten runs. Our proposed GA outperforms the evolutionary algorithm developed by Binh et al. [2] w.r.t. the quality of the achieved solutions, providing better solutions and smaller average percentage gaps in comparison to the evolutionary algorithm developed by Binh et al. [2] for each instance. Concerning the running time, our algorithm needed longer computation time for solving the instances, but that is explicable because our algorithm found better solutions than those provided by Binh et al. [2] and explores the entire space of solutions.

4.2. **Computational results on non-euclidean instances.** The euclidean instances reported on Tables 1 – 2 were transformed into non-euclidean instances, as follows:

a) for each edge e of G
   if $c_e \neq 0$
      $r \leftarrow random\ value \in [-0.5 \cdot c_e,\ 0.5 \cdot c_e]$

TABLE 2. Experimental results in the case of large euclidean instances of Type 1

Instance		Our Exact Algorithm		Evolutionary Algorithm [2]			Our Genetic Algorithm		
No.	Size	Optimal solution	Time (seconds)	Best solution	Average solution	Time (minutes)	Best solution	Average solution	Time (minutes)
1.	10a280	27925.2	< 0.001	28690.9	29664.8	**0.02**	**27925.2**	**27925.2**	0.03
2.	10gil262	27637.4	< 0.001	29075.0	29568.4	**0.02**	**27637.4**	**27637.4**	0.08
3.	10lin318	809749.9	< 0.001	832299.5	841893.2	**0.02**	**809749.9**	**809749.9**	0.07
4.	10pcb442	741195.8	< 0.001	765561.0	796960.4	**0.02**	**741195.8**	**741195.8**	0.07
5.	10pr439	1904690.2	< 0.001	1971633.0	2022257.4	**0.02**	**1904690.2**	**1904690.2**	0.04
6.	25a280	29902.4	< 0.001	31481.2	32020.2	**0.03**	**29902.4**	**29909.7**	0.99
7.	25gil262	30325.6	< 0.001	31579.5	31949.7	**0.03**	**30325.6**	**30329.3**	1.46
8.	25lin318	584554.0	< 0.001	607029.0	617399.9	**0.03**	**584554.0**	**584590.4**	1.28
9.	25pcb442	740892.5	< 0.001	794217.4	805896.7	**0.03**	**740892.5**	**740910.2**	1.10
10.	25pr439	1511168.9	< 0.001	1585283.0	1612334.7	**0.03**	**1511168.9**	**1511275.5**	1.02
11.	50a280	36266.9	< 0.001	37458.4	37828.6	**0.10**	**36290.7**	**36322.6**	7.96
12.	50gil262	26523.2	< 0.001	27647.5	27836.2	**0.10**	**26524.4**	**26576.5**	5.68
13.	50lin318	688724.6	< 0.001	706854.9	713744.5	**0.10**	**688952.4**	**689357.2**	8.08
14.	50pcb442	910478.6	< 0.001	949830.8	954169.0	**0.10**	**911563.7**	**912364.6**	8.56
15.	50nrw1379	1831566.9	< 0.001	-	-	-	**1833381.3**	**1833926.0**	12.27
16.	50pcb1173	1108183.7	< 0.001	-	-	-	**1108602.6**	**1109313.6**	11.30
17.	50pr1002	5243008.6	< 0.001	-	-	-	**5245119.9**	**5246848.6**	6.97
18.	100pr1002	6213697.0	< 0.001	-	-	-	**6227722.6**	**6230444.8**	53.81
19.	100rat783	175893.9	< 0.001	-	-	-	**176430.7**	**176502.9**	51.47
20.	100vm1084	8504736.0	< 0.001	-	-	-	**8522466.4**	**8526639.9**	48.73

$c_e \leftarrow max\{\lfloor c_e + r \rfloor,\ 1\}$

b) for each cluster C_x

$n_x \leftarrow random\ integer \in [\ 1,\ |C_x| \cdot (|C_x| - 1)/2\ ]$

randomly choose n_x intra-cluster edges from C_x

for each chosen edge e

if $c_e \neq 0$

$r \leftarrow random\ value \in [0,\ 0.75 \cdot c_e]$

$c_e \leftarrow max\{\lfloor c_e - r \rfloor,\ 1\}$

In Tables 3 – 4 we report the solutions achieved by our GA for solving 40 non-euclidean instances of Type 1 of the CluSPTP. The first four columns indicate the number of the instance, its name and information about its dimension, the next two columns contain the best and average solutions obtained by our proposed GA, then we provide the percentage gap calculated as follows: $\%gap = 100 \times (Best\ sol. - Average\ sol.)/Best\ sol.$, where $Best\ sol.$ and $Average\ sol.$ are the costs of the best respectively the average solutions achieved by our algorithm in the ten runs of each instance, and the last column contains the necessary average computational times reported in minutes in order to achieve the corresponding solutions. The instances with no variation are marked with bold font.

Analyzing the results displayed in Tables 3 – 4, we can remark that:

1. In the case of the small instances of Type 1, our GA provided in 15 out of 20 instances the same best solutions in all the ten runs, and for the other instances the percentage gap is at most 0.21%. The necessary average computational time value reported in minutes in order to achieve the corresponding solutions is at most 1.19 minutes.
2. In the case of the large instances of Type 1, our GA achieved in 2 out of 20 instances the same best solutions in all the ten runs, and for the other instances the percentage gap is at most 2.83%. The necessary average computational time reported in minutes in order to achieve the corresponding solutions are bellow 4.83 minutes in 16 out of 20 instances and at most 42.54 minutes.
3. We can notice that the percentage gap is bellow 0.88% in 36 out of 40 instances and at most 2.83%, fact that proves the stability of our proposed solution approach.

TABLE 3. Experimental results in the case of small non-euclidean instances of Type 1

Instance				Our Genetic Algorithm			
No.	Name	No. nodes	No. clusters	Best solution	Average solution	gap %	Time (minutes)
1.	Nec-10berlin52	52	10	28027	28027.00	**0.00**	0.01
2.	Nec-10eil51	51	10	1009	1009.00	**0.00**	0.01
3.	Nec-10eil76	76	10	1258	1258.00	**0.00**	0.01
4.	Nec-10kroB100	100	10	76274	76274.00	**0.00**	0.01
5.	Nec-10rat99	99	10	4111	4111.00	**0.00**	0.01
6.	Nec-10st70	70	10	1628	1628.00	**0.00**	0.01
7.	Nec-15berlin52	52	15	16836	16836.00	**0.00**	0.02
8.	Nec-15eil51	51	15	867	867.00	**0.00**	0.01
9.	Nec-15eil76	76	15	1578	1578.00	**0.00**	0.02
10.	Nec-15pr76	76	15	404626	404626.00	**0.00**	0.03
11.	Nec-15st70	70	15	2204	2204.00	**0.00**	0.02
12.	Nec-25eil101	101	25	2514	2519.50	0.21	0.09
13.	Nec-25kroA100	100	25	86971	87050.00	0.09	0.07
14.	Nec-25lin105	105	25	56371	56393.50	0.04	0.10
15.	Nec-25rat99	99	25	3742	3742.00	**0.00**	0.07
16.	Nec-50eil101	101	50	2035	2036.00	0.04	1.19
17.	Nec-50kroA100	100	50	92131	92203.50	0.07	0.88
18.	Nec-50kroB100	100	50	78632	78632.00	**0.00**	0.60
19.	Nec-50lin105	105	50	80785	80785.00	**0.00**	0.81
20.	Nec-50rat99	99	50	4651	4651.00	**0.00**	0.66

TABLE 4. Experimental results in the case of large non-euclidean instances of Type 1

Instance				Our Genetic Algorithm			
No.	Name	No. nodes	No. clusters	Best solution	Average solution	gap %	Time (minutes)
1.	Nec-10a280	280	10	13659	13659.00	**0.00**	0.02
2.	Nec-10gil262	262	10	13804	13845.00	0.29	0.02
3.	Nec-10lin318	318	10	347715	349190.75	0.42	0.02
4.	Nec-10pcb442	442	10	289271	293341.50	1.40	0.03
5.	Nec-10pr439	439	10	821867	826615.25	0.57	0.03
6.	Nec-25a280	280	25	16115	16219.25	0.64	0.15
7.	Nec-25gil262	262	25	15394	15434.00	0.26	0.15
8.	Nec-25lin318	318	25	313801	314488.75	0.21	0.18
9.	Nec-25pcb442	442	25	368212	369320.75	0.30	0.23
10.	Nec-25pr439	439	25	724265	727710.50	0.47	0.19
11.	Nec-50a280	280	50	19630	19634.00	0.02	2.71
12.	Nec-50gil262	262	50	14234	14244.50	0.07	1.56
13.	Nec-50lin318	318	50	363560	363572.50	**0.00**	2.10
14.	Nec-50pcb442	442	50	492495	496873.25	0.88	2.81
15.	Nec-50nrw1379	1379	50	741498	762543.75	2.83	8.27
16.	Nec-50pcb1173	1173	50	462554	471152.00	1.85	4.83
17.	Nec-50pr1002	1002	50	2604500	2637842.00	1.28	4.43
18.	Nec-100pr1002	1002	100	3215011	3223283.75	0.25	35.62
19.	Nec-100rat783	783	100	89422	89543.50	0.13	30.83
20.	Nec-100vm1084	1084	100	4244669	4260804.00	0.38	42.54

5. CONCLUSIONS

In this paper, we described a novel genetic algorithm for solving the clustered shortest-path tree problem and an exact algorithm that solves the investigated problem in the case of euclidean instances defined on complete graphs. The proposed GA has two important characteristics: the employment of an efficient representation scheme which saves computer memory and allows us to explore as much of the solutions space as possible and the use of a selection strategy which is a combination between random selection and elitist selection.

We evaluated the performance of the proposed solution approach on two sets of benchmark instances: one set of 20 euclidean instances available in the literature and the other one containing 20 non-euclidean instances. In the case of the euclidean instances we provided the optimal solutions obtained by our developed exact algorithm and the results achieved by our GA and we compared it with the evolutionary algorithm proposed by Binh et al. [2] that constitutes the state-of-the-art for the CluSPTP with respect to solution quality and computation time. The computational results that we achieved prove the efficiency of our developed GA in yielding high quality solutions within reasonable running times, besides its superiority as compared to the evolutionary algorithm proposed by Binh et al. [2].

In future work, we plan to assess the generality and scalability of our developed solution approach by testing it on different types of instances.

REFERENCES

[1] Binh, H. T. T., Thanh, P. D., Trung, T.B. and Thao, L. P., *Effective multifactorial evolutionary algorithm for solving the cluster shortest path tree problem*, in Proc. of IEEE Congress on Evolutionary Computation (CEC), pp. 819–826, 2018

[2] Binh, H. T. T., Thanh, P. D. and 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** (2019), 12–25

[3] Demange, M., Monnot, J., Pop, P. C. and Ries, B., *On the complexity of the selective graph coloring problem in some special classes of graphs*, Theor. Comput. Sci., **540-541** (2014), 82–102

[4] D'Emidio, M., Forlizzi, L., Frigioni, D., Leucci, S. and Proietti, G., *On the clustered shortest-path tree problem*, in Proc. of Italian Conference on Theoretical Computer Science, pp. 263–268, 2016

[5] D'Emidio, M., Forlizzi, L., Frigioni, D., Leucci, S. and Proietti, G., *Hardness, approximability and fixed-parameter tractability of the clustered shortest-path tree problem*, J. Comb. Optim., **38** (2019), 165–184

[6] Fidanova, S. and Pop, P.C., *An improved hybrid ant-local search for the partition graph coloring problem*, J. Comput. Appl. Math., **293** (2016), 55–61

[7] Fischetti, M., Salazar-Gonzales, J. J. and Toth, P., *A Branch-and-Cut Algorithm for the Symmetric Generalized Traveling Salesman Problem*, Oper. Res., **45** (1997), No. 3, 378–394

[8] Ghiani, G. and Improta, G., *An efficient transformation of the generalized vehicle routing problem*, Eur. J. Oper. Res., **122** (2000), 11–17

[9] Thanh P. D., *CluSPT instances. Mendeley Data v3.* https://doi.org/10.17632/b4gcgybvt6.3 (2019)

[10] Myung, Y. S., Lee, C. H. and Tcha, D. W., *On the generalized minimum spanning tree problem*, Networks, **26** (1995), 231–241

[11] Pintea, C., Chira, C., Dumitrescu, D. and Pop, P. C., *Sensitive ants in solving the generalized vehicle routing problem*, Int. J. Comput. Commun., **6** (2011), No. 4, 734–741

[12] Pop, P. C., *Generalized Network Design Problems, Modelling and Optimization* De Gruyter, Germany, 2012

[13] Pop, P. C., Matei, O. and Sabo, C., *A hybrid diploid genetic based algorithm for solving the generalized traveling salesman problem*, in Proc. of HAIS 2017, Lect. Notes Comput. Sci., **10334** (2017), 149–160

[14] Pop, P. C., Matei, O., Sabo, C. and Petrovan, A., *A two-level solution approach for solving the generalized minimum spanning tree problem*, Eur. J. Oper. Res., **265** (2018), No. 2, 478–487

[15] Pop, P. C., Fuksz, L., Horvat Marc, A., and Sabo, C., *A novel two-level optimization approach for clustered vehicle routing problem*, Computers & Industrial Engineering, **115** (2018), 304–318

[16] Pop, P. C., *The generalized minimum spanning tree problem: an overview of formulations, solution procedures and latest advances*, Eur. J. Oper. Res., **283** (2020), No. 1, 1–15

[17] Thanh, P. D., Dung, D. A., Tien, T. N. and Binh, H. T. T., *An Effective Representation Scheme in Multifactorial Evolutionary Algorithm for Solving Cluster Shortest-Path Tree Problem*, in Proc. of IEEE Congress on Evolutionary Computation (CEC), art. No. 8477684, 2018

[18] Thanh, P. D., Binh, H. T. T., Dac, D. D., Binh Long, N. and Hai Phong, L. M., *A Heuristic Based on Randomized Greedy Algorithms for the Clustered Shortest-Path Tree Problem*, in Proc. of IEEE Congress on Evolutionary Computation (CEC), art. no. 8790070, pp. 2915–2922, 2019

DEPARTMENT OF MATHEMATICS AND COMPUTER SCIENCE
TECHNICAL UNIVERSITY OF CLUJ-NAPOCA
NORTH UNIVERSITY CENTRE AT BAIA MARE
VICTORIEI 76, RO-430122 BAIA MARE, ROMANIA
Email address: ovidiu.cosma@cunbm.utcluj.ro
Email address: petrica.pop@cunbm.utcluj.ro
Email address: ioana.zelina@cunbm.utcluj.ro