Received December 17, 2020, accepted January 10, 2021, date of publication January 21, 2021, date of current version January 27, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3053295

An Effective Genetic Algorithm for Solving the Clustered Shortest-Path Tree Problem

OVIDIU COSMA, PETRICĂ C. POP, AND IOANA ZELINA
Department of Mathematics and Computer Science, North University Center of Baia Mare, Technical University of Cluj-Napoca, 430122 Baia Mare, Romania
Corresponding author: Petrică C. Pop (petrica.pop@cunbm.utcluj.ro)

ABSTRACT The clustered shortest-path tree problem (CluSPTP) is an extension of the classical single-source shortest-path problem, in which, given a graph with the set of nodes partitioned into a predefined, mutually exclusive and exhaustive set of clusters, we are looking for 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. CluSPTP belongs to the class of generalized combinatorial optimization problems, and, in general, is proved to be a non-deterministic polynomial time hard (NP-hard) problem. In this paper, we propose a novel genetic algorithm (GA), which is designed to fit the challenges of the investigated problem. The main features of our GA are: the use of an innovative representation scheme that allows us to define meaningful genetic operators and the use of a hybrid initial population. Extensive computational results are reported and discussed for two sets of instances: euclidean and non-euclidean. The performance of the proposed algorithm was evaluated on six types of benchmark euclidean instances available in the literature and on six types of non-euclidean instances obtained from the corresponding euclidean ones. The obtained results show an improvement with respect to existing methods from the literature, both in terms of the quality of the achieved solutions and the computation times necessary to obtain them. They demonstrate that our genetic algorithm outperforms all the existing methods from the literature, providing for all the existing benchmark instances the optimal solutions in all 30 independent trials.

INDEX TERMS Single-source shortest-path problem, clustered shortest-path tree problem, genetic algorithms.

I. INTRODUCTION

In this paper, we consider the clustered shortest-path tree problem, which generalizes the classical single-source shortest-path problem, and looks for a spanning tree of a given graph with the property that each sub-graph induced by a cluster is connected, and the total cost of the paths from a given source node to all the other nodes of the graph is minimized.

Why is it important to investigate the CluSPTP? CluSPTP is a variant of the classical shortest path problem (SPP), and unlike the problem that it generalizes, it is a complex combinatorial optimization problem, and it belongs to the class of NP-hard problems. CluSPTP is worth to be studied due to its theoretical properties and many interesting and important applications, especially in communication networks, agriculture irrigation, distribution problems, etc. We can observe that SPP is a special case of CluSPTP in the case when all the clusters are singletons.

The current literature is rather scarce. The problem was introduced by D'Emidio *et al.* [6] justified by some practical applications in communication networks. The same authors, in an extended version of their paper [7], 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.* [22] 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.* [23] 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, Cosma *et al.* [3] presented four

The associate editor coordinating the review of this manuscript and approving it for publication was Hisao Ishibuchi.

particular cases in which CluSPTP is solvable in polynomial time, proposed a genetic algorithm for solving the general case, and reported as well some preliminary computational results, and Hahn *et al.* [12] described an evolutionary algorithm and a multifactorial evolutionary algorithm for solving the CluSPTP. We should point out that all the proposed solution approaches, except the GA proposed by Cosma *et al.* [3], eventhough they present different strategies to explore and exploit the solution space of the CluSPTP, they are only tested on Euclidean instances, which can be solved optimally by the exact algorithm described by Cosma *et al.* [3].

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 problem and aims to model some aggregation phenomena occurring between similar entities. It has 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. The generalized combinatorial optimization problems are more difficult compared to corresponding problems in non-clustered settings, and have been intensively studied in the last years due to their theoretical properties and practical applications. For further reference on this class of problems we refer to [8], [16]. A closely related problem to CluSPTP was introduced by Myung *et al.* [15], 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 recent advances and more information concerning the generalized minimum spanning tree problem and its variants, we refer to Pop *et al.* [18], [20]. Some other generalized combinatorial optimization problems that have been investigated, are: the generalized traveling salesman problem and its variants [10], [17], the generalized vehicle routing problem and its variants [11], [19], the selective graph coloring problem [4], [9], the selective vehicle routing problem [21], other related problems [25], [26] etc.

The purpose of this paper is to propose a novel solution approach that fits the challenges of CluSPTP. Our developed genetic algorithm has certain features, that differentiate it from the other existing methods from the literature:

- the use of a compact representation scheme, that concentrates the essential solution information and enables the efficient exploration of the entire solutions space, with large populations of chromosomes.
- the use of efficient mutation and crossover operators that do not generate invalid offspring that would require subsequent adjustments.
- the use of a hybrid initial population that contains both random and constructed chromosomes. For boosting the quality of the solutions, the constructed initial population is merged with the current population at the right stage of evolution.

As will be shown in the computational experiments section, our proposed solution approach provides the optimal solutions within a very short computational time for all the existing benchmark instances from the literature, outperforming the best developed algorithms for solving the CluSPTP in terms of both solution quality and CPU time required.

In addition to the existing benchmark instances which all are euclidean and defined on complete graphs, we have described a collection of 248 non-euclidean instances divided into six classes, and we have reported the achieved results using our novel solution approach.

The present paper is organized as follows: Section II provides a formal definition of the clustered shortest-path tree problem and information about its complexity and particular cases when the problem is solvable in polynomial time. In Section III, we describe the genetic algorithm that has some novel features, and it exploits the structure and properties of the investigated problem. The next section, Section IV contains the extensive computational results achieved for two sets of instances: euclidean and non-euclidean, and provides a comparative analysis of the performance of our proposed genetic algorithm against the best existing solution approaches from the literature, while in Section V, we present some concluding results, as well as further research directions.

II. DEFINITION OF THE CLUSTERED SHORTEST-PATH TREE PROBLEM

We consider $G = (V, E, c)$ an undirected, connected, and weighted graph characterized by the set of nodes $V = \{v_1, v_2, \dots, v_n\}$, the set of edges $E = \{e_1, \dots, e_m\}$, where

$$E \subseteq \{(v_i, v_j) | \; v_i, v_j \in V, \; i < j, \; i, j \in \{1, 2, \dots, n\}\}, \quad (1)$$

and the cost function $c : V \to R_+$, which assigns to every edge $e = (u, v) \in E$ of the graph, a positive number $c(e) = c_e = c_{(u,v)} \in R_+$, called the cost of the edge e.

The shortest path problem (SPP) in non-clustered settings was intensively investigated, and it is defined as the problem of finding a rooted spanning tree such that the total cost of the paths from the root to all other nodes in the graph is minimum. SPP can be solved optimally in $O(m+n \log n)$ using the Dijkstra's algorithm [5]. Dijkstra's original algorithm found the shortest path between two given nodes, but a more common variant fixes a single node as the *source node* and finds the shortest paths from the source to all other nodes in the graph, producing a *shortest-path tree*. The cost of the shortest path between v_i and v_j in a spanning tree T is denoted by $d_T(v_i, v_j)$, and the total cost of the paths from a given source node s to all the other nodes of the graph is calculated as the sum of the shortest paths, $\sum_{v \in V} d_T(s, v)$.

In order to define the clustered shortest-path tree problem, we consider a partition of the entire set of nodes V, which means that the set V is divided into k subsets, $C_1, \dots, C_k$ for which:

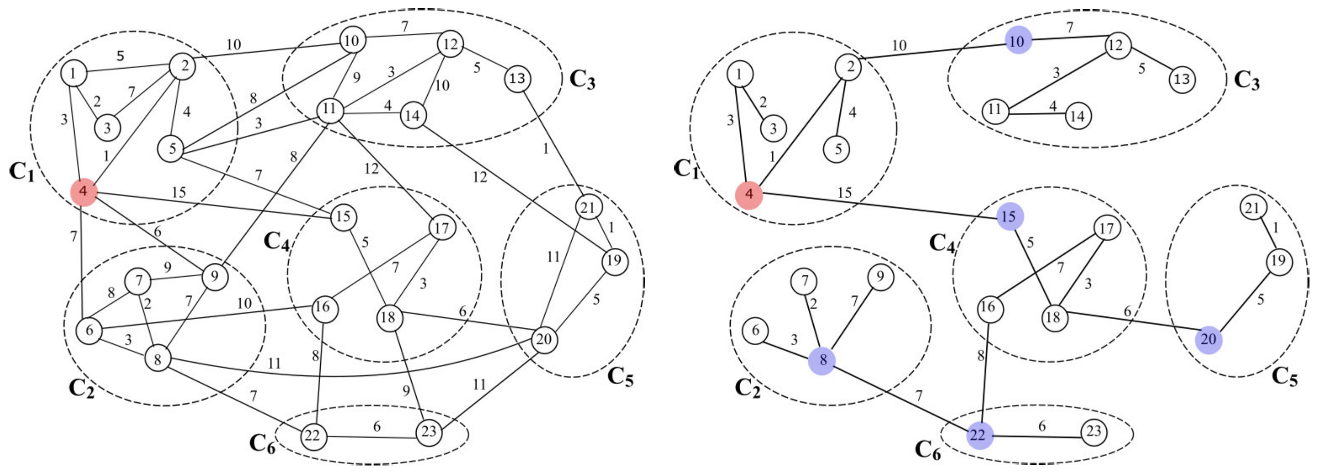


FIGURE 1. An example of CluSPTP and a feasible solution of the problem.

1. $V = C_1 \cup C_2 \cup \ldots \cup C_k$;
2. $C_i \cap C_j = \emptyset$ for all $i \neq j \in \{1, \ldots, k\}$.

The subsets of nodes $C_i,\ i \in \{1, \ldots, k\}$ are called *clusters*. The number of nodes in each cluster C_i is denoted by n_i, $n_i = |C_i|,\ i \in \{1, 2, \ldots, k\}$, and we have $n_1 + \ldots + n_k = n$.

There are two categories of edges in the set E of graph G: edges connecting nodes from the same cluster, $e = (u, v) \in E,\ u, v \in C_i,\ i \in \{1, \ldots, k\}$, called intra-cluster edges, and edges connecting nodes belonging to different clusters, $e = (u, v) \in E,\ u \in C_i,\ v \in C_j$ with $i \neq j$ and $i, j \in \{1, \ldots, k\}$, called inter-cluster edges. We denote by E_1 the set of intra-cluster edges and by E_2 the set of inter-cluster edges, obviously we have that $E_1 \cup E_2 = E$ and $E_1 \cap E_2 = \emptyset$.

If S is a subset of nodes, $S \subseteq V$, then by $G[S]$ we will denote the subgraph induced by S. As in the case of the SPP in non-clustered settings, given a source node $s \in V$, we will use the same notation $\sum_{v \in V} d_T(s, v)$ for the total cost of the paths from the given source node s to all the other nodes of the graph.

Then the *clustered shortest-path tree problem* is the problem of finding a minimum cost spanning tree T for the graph G partitioned into clusters, with the following properties:

1. T spans all the nodes of the graph G;
2. For each cluster C_i, $i \in \{1, \ldots, k\}$, the induced subgraph $T[C_i]$ is connected;

such that the total cost of the paths from a given source node $s \in V$ to all the other nodes of the graph is minimized, i.e.

$$\sum_{v \in V} d_T(s, v) \rightarrow \min\,. \tag{2}$$

In Figure 1, we illustrated an example of the CluSPTP defined on an undirected, connected, weighted graph with $n = 23$ nodes partitioned in $k = 6$ clusters, with the source node $s = 4 \in C_1$ (marked with red color) and a feasible solution of the problem.

We observe that the feasible solution is a tree spanning all the nodes of the graph with the property that the induced subgraph $T[C_i]$ is connected for each cluster. In addition, in the figure illustrating the feasible solution of the problem we highlighted the source node of each cluster by blue color.

D'Emidio *et al.* [7] showed that in general the CluSPTP is $\mathcal{NP}$*-hard* and in addition provided the following approximability results:

1. CluSPTP is hard to approximate within a factor of $n^{1-\epsilon}$ for any constant $\epsilon \in (0, 1]$, unless $\mathcal{P} = \mathcal{NP}$;
2. There exists a polynomial-time n-approximation algorithm for CluSPTP;
3. CluSPTP is fixed-parameter tractable.

Cosma *et al.* [3] presented four special cases of the CluSPTP which are solvable in polynomial time. An important case that was used to test all the developed solution approaches is the situation when the CluSPTP is defined on complete and euclidean graphs. We briefly describe an algorithm that solves optimally the CluSPTP in polynomial time in this case.

Because the graph is euclidean, the triangle inequality holds, and the shortest path between two nodes in the graph G is always the edge that connects them, therefore $d_G(v, u) = c_{(v,u)}$, where $c_{(v,u)}$ is the cost of the edge $e = (v, u) \in E$, 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 $C_i,\ i \in \{1, 2, \ldots, k\} \setminus \{r\}$, the cost of reaching the nodes in C_i from s in the spanning tree is:

$$CT_i = \sum_{v \in C_i} d_T(s, v) = c_{(s,s_i)} \cdot n_i + \sum_{v \in C_i} c_{(s_i,v)}. \tag{3}$$

The optimal solution $OptC$ is obtained by minimizing the total cost $\sum_{i=1}^{k} CT_i$,

$$OptC = \sum_{i=1}^{k} \min_{u \in C_i} \left\{ n_i \cdot c_{(s,u)} + \sum_{v \in C_i} c_{(u,v)} \right\} \qquad (4)$$

and can be efficiently found using a greedy algorithm. If the source node $s \in C_r$ is given, the algorithm can be described as follows:

a) $OptC = CT_r$;

b) For each $i \in \{1, 2, \ldots, k\} \setminus \{r\}$

  b_1) choose $s_i \in C_i$ that minimizes CT_i;

  b_2) calculate $OptC = OptC + CT_i$.

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

III. DESCRIPTION OF THE PROPOSED GENETIC ALGORITHM

In this section, we describe our novel genetic algorithm (GA). Genetic algorithms were first introduced by Holland [13] and are search heuristic methods inspired from the theory of natural evolution developed by Charles Darwin based on natural genetics and natural selection. GAs have the ability to deliver a ''good-enough'' solution ''fast-enough'', making them very attractive in solving optimization problems.

The proposed optimization algorithm has the specific components of a genetic algorithm, with the following elements of originality, which have proved effective in the case of the investigated problem. The initial population consists of two parts: a constructed part and a random part. The constructed chromosomes are calculated based on the greedy algorithm presented in Section II. The constructed chromosomes have the advantage of very good fitness, while the random ones have the advantage of diversity. The selection operator chooses the best chromosomes that will form the current population. This operator will process only once each of the two components of the original population. The random part of the initial population will be processed at the initialization of the algorithm, but the constructed part will only be processed when the offspring have become good enough so that their fitness will approach that of the constructed chromosomes. If the constructed population is processed too soon by the selection operator, the constructed chromosomes become dominant, the diversity of random chromosomes is lost, the algorithm converges too quickly, and there is a good chance of missing the optimal solution. The crossover operator selects two parents from the current population and uses their properties to create an offspring. The parent selection mechanism is a combination between elitist and random selection strategies. A uniform crossover strategy is used for creating the offspring genes. The mutation operator applies weak mutations with 100% probability.

The structure of the proposed genetic algorithm is given in Figure 2, and its description is provided next.

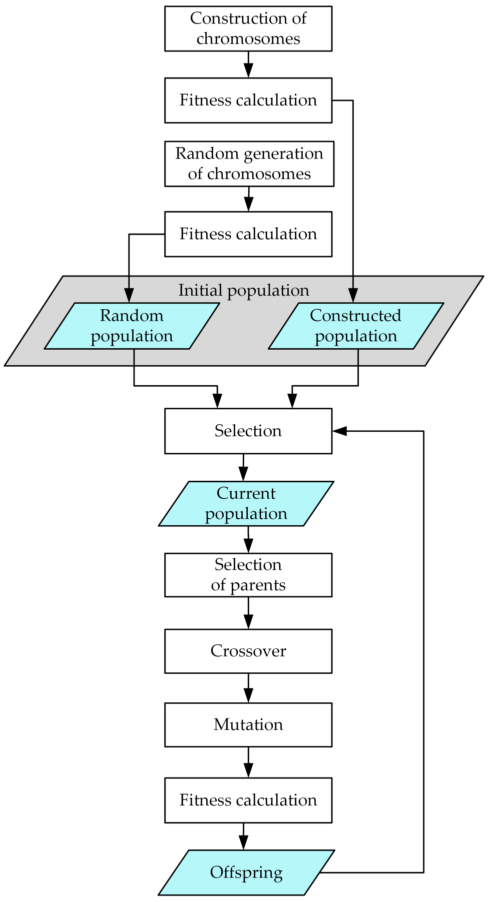


FIGURE 2. The flowchart of the proposed genetic algorithm for solving the CluSPTP.

A. THE CHROMOSOME STRUCTURE

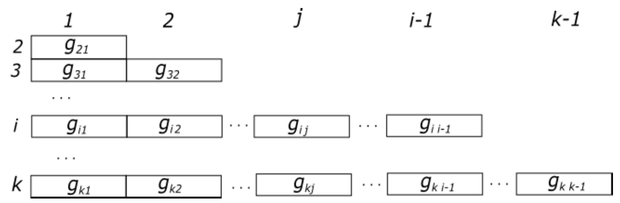
It is well-known that a good representation scheme has an important effect on the performance of the GA, and it should define meaningful genetic operators in order to minimize the computational effort within these procedures.

In order to meet this requirement, we use an efficient representation in which the genes of a chromosome contain a complete set of inter-cluster edges, one for each pair of clusters. Therefore, 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 a given pair of clusters C_i, C_j will be denoted by g_{ij} for all $i > j$ and $i, j \in \{1, .., k\}$. The gene g_{ij} corresponds to an edge between clusters C_i and C_j, if there is at least an edge $(u, v) \in E$, $u \in C_i$ $v \in C_j$ with $i > j$, otherwise the gene g_{ij} is void.

The genes of a chromosome will be stored in a triangular array with $k-1$ lines, in which the element g_{ij} belonging to the line i and column j in the array is the gene that connects the clusters C_i and C_j where $i \in \{2, \ldots, k\}$ and $j \in \{1, \ldots, i-1\}$.

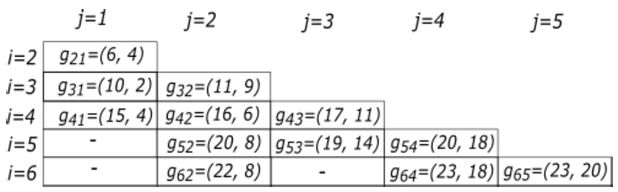
In the proposed genetic algorithm, we define a chromosome A as a set of $\frac{k(k-1)}{2}$ genes corresponding to a set of inter-cluster edges as follows:

$$A = \{g_{ij} \mid i \in \{2, \ldots, k\},\ j \in \{1, \ldots, i-1\}\}. \qquad (5)$$



	1	2		j		i-1		k-1
2	g_{21}							
3	g_{31}	g_{32}						
. . .								
i	g_{i1}	g_{i2}	. . .	g_{ij}	. . .	$g_{i\ i-1}$		
. . .								
k	g_{k1}	g_{k2}	. . .	g_{kj}	. . .	$g_{k\ i-1}$	. . .	$g_{k\ k-1}$

FIGURE 3. The structure of a chromosome gene array.



	j=1	j=2	j=3	j=4	j=5
i=2	g_{21}=(6, 4)				
i=3	g_{31}=(10, 2)	g_{32}=(11, 9)			
i=4	g_{41}=(15, 4)	g_{42}=(16, 6)	g_{43}=(17, 11)		
i=5	-	g_{52}=(20, 8)	g_{53}=(19, 14)	g_{54}=(20, 18)	
i=6	-	g_{62}=(22, 8)	-	g_{64}=(23, 18)	g_{65}=(23, 20)

FIGURE 4. A chromosome gene array for the instance presented in Figure 1.

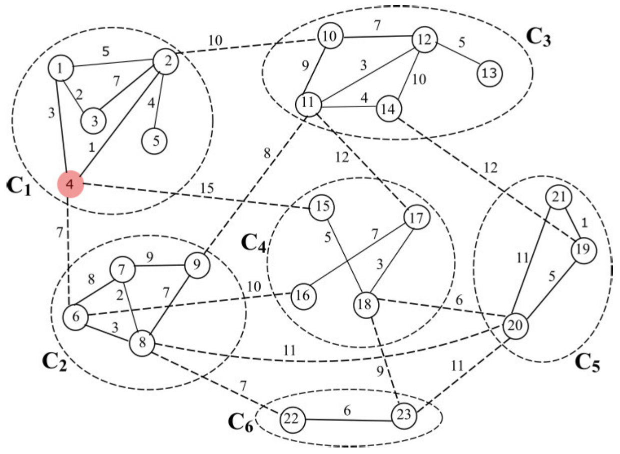


FIGURE 5. The subgraph defined by the chromosome A illustrated in Figure 4.

A chromosome $A \subseteq E_2$ defines a subgraph $G_A = (V, A \cup E_1)$ of G with the inter-cluster edges corresponding to the set of genes in A. This subgraph corresponds to a CluSTSP subproblem, in which the graph G is replaced by its subgraph G_A.

For the example presented in Figure 1, a chromosome gene array A is represented in Figure 4, and the corresponding subgraph defined by the chromosome is illustrated in Figure 5.

There exist several feasible solutions of the CluSTSP associated to a given chromosome in the corresponding subgraph G_A. Next we will describe an efficient heuristic algorithm that determines a feasible solution of CluSTSP problem.

B. DETERMINING A FEASIBLE SOLUTION OF THE CluSTSP CORRESPONDING TO A GIVEN CHROMOSOME

We describe an efficient heuristic algorithm that determines a feasible solution of the CluSTSP associated to the subgraph G_A corresponding to the chromosome A. The algorithm consists of five steps and uses the fact that any two clusters are connected by at most an edge based on the representation of the chromosomes in our GA.

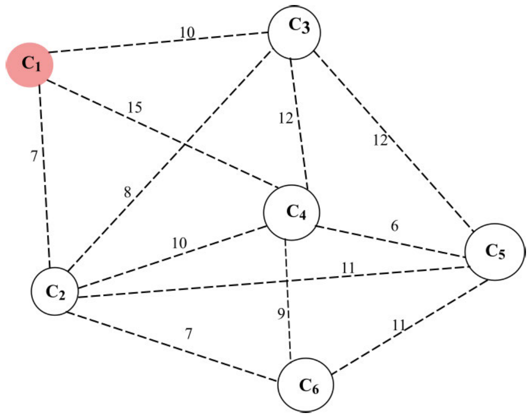


FIGURE 6. The macro-level layout S_A of the subgraph defined by the chromosome A presented in Figure 4.

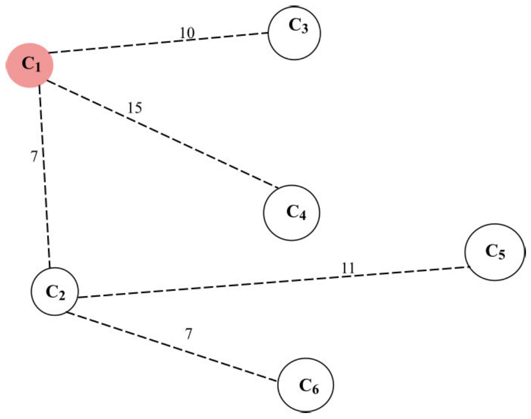
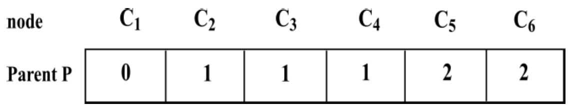


FIGURE 7. The spanning tree T_{S_A} of the macro-level layout S_A illustrated in Figure 6.

STEP 1: A macro-level layout S_A induced by the subgraph G_A is constructed. The macro-level layout is a graph with k nodes, $V_{macro} = \{C_1, \ldots, C_k\}$, each node obtained after replacing all the vertices of a cluster C_i with a supernode representing it, and the set of edges which contains $k(k-1)/2$ that constitute the chromosome A, $S_A = (V_{macro}, A)$. The source node of the macro-level layout is the node corresponding to the source cluster C_r that contains the source node s of the instance. We will call this cluster the *source cluster*. The macro-level layout of the subgraph defined by the chromosome A presented in Figure 4 is shown in Figure 6.

STEP 2: We apply the Shortest Path First (SPF) algorithm on the macro-level graph S_A. The SPF produces a spanning tree T_{S_A} that contains the optimal inter-cluster routes. The spanning tree T in the case of the macro-level layout S_A shown in Figure 6 is presented in Figure 7.

The spanning tree associated to the macro-level graph S_A is stored in a parent array P with k elements. The parent of the source cluster C_r is $P[r] = 0$ and for every other cluster C_i, $i \in \{1, 2, \ldots, k\} \setminus \{r\}$ the parent is $P[i] = j$, where C_j is the parent of C_i in the spanning tree of the macro-level graph. In the case of the spanning tree illustrated in Figure 7 the parent array is shown in Figure 8.



node	C_1	C_2	C_3	C_4	C_5	C_6
Parent P	0	1	1	1	2	2

FIGURE 8. The parent array for the tree illustrated in in Figure 7.

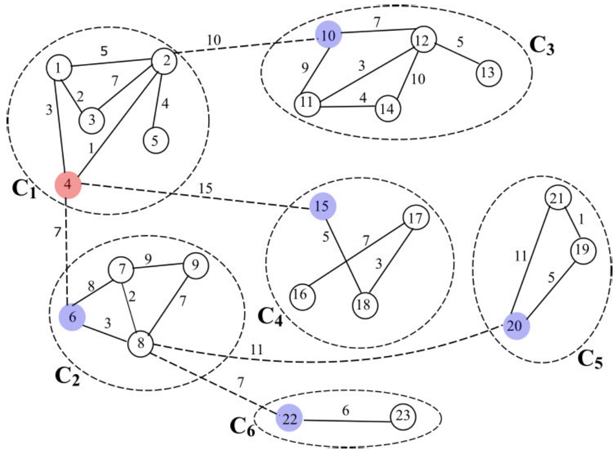


FIGURE 9. Inter-cluster tree of the subgraph illustrated in Figure 4.

STEP 3: The source nodes for each of the k clusters are determined, using the parent array P of the macro-level tree and the genes array, as follows:

- The source node s of the source cluster C_r is the source node of the instance.
- The source node of cluster C_y is determined considering its parent C_x, $x = P[y]$ in the parent array of the macro-level tree. The source node of cluster C_y is the extremity in C_y of the edge represented by the gene g_{ab} in the chromosome gene array, where $a = max\{x, y\}$, $b = min\{x, y\}$.

The resulting subgraph from step 3 in our algorithm in the case of the example presented in Figure 4, is shown in Figure 9. All inter-cluster edges of the instance graph have been removed except those in the macro-level tree T_{S_A} in Figure 7. The source node in each cluster is highlighted.

STEP 4: The spanning tree T_i inside each cluster of the graph C_i, $i \in \{1, 2, \ldots, n\}$, is determined. The spanning tree T_i is obtained by running the SPF algorithm within the cluster C_i. By connecting the cluster spanning trees with the edges of the skeleton tree, a spanning tree for the entire instance is generated. Considering different genes in the chromosome, we obtain different instance spanning trees, but each instance spanning tree satisfies the feasibility conditions of the CluSPTP.

For the instance presented in Figure 1, the feasible solution of the CluSPTP generated using the chromosome gene array from Figure 4 is shown in Figure 10.

STEP 5: the total cost of the solution, $TotC$, is determined, using the following relation on the instance spanning tree:

$$TotC = \sum_{i=1}^{k} (n_i \cdot d_T(s, s_i) + cl_i) \tag{6}$$

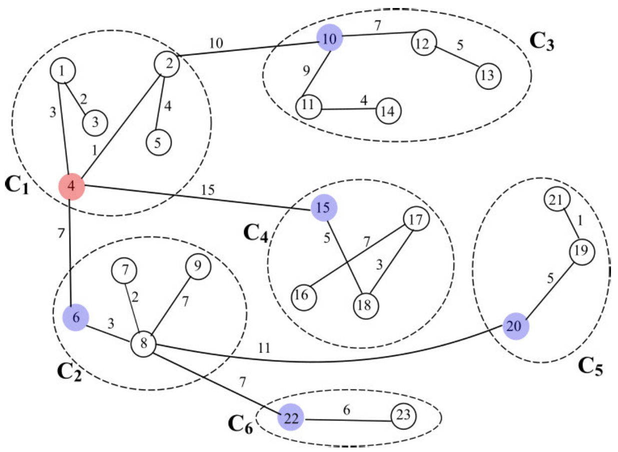
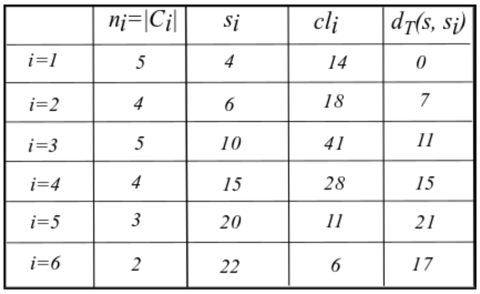


FIGURE 10. The feasible solution of the CluSPTP corresponding to the instance from Figure 1 generated using the chromosome illustrated in Figure 4.



	$n_i=\|C_i\|$	s_i	cl_i	$d_T(s, s_i)$
i=1	5	4	14	0
i=2	4	6	18	7
i=3	5	10	41	11
i=4	4	15	28	15
i=5	3	20	11	21
i=6	2	22	6	17

FIGURE 11. The values of the operands from the total cost formula of the solution presented in Figure 10.

where $n_i = |C_i|$ is the number of nodes in cluster C_i, s_i is the source node in cluster C_i and cl_i is the cost of all the routes from source node s_i inside cluster C_i, named the *total internal cost* of cluster C_i. We have that

$$cl_i = \sum_{v \in C_i} d_T(s_i, v) \tag{7}$$

For the instance illustrated in Figure 10, the values of the operands in the formula of the total cost are shown in Figure 11.

Using the formula for computing the total cost, we obtain that the cost of the feasible solution of the CluSPTP illustrated in Figure 10 is $TotC = 358$.

C. EFFICIENCY ISSUES

Since the optimization process may require the evaluation of a large number of chromosomes, the algorithm should avoid repeating the same operations. The proposed solution is to run the SPF algorithm within each cluster C_i, for each possible source node $s_i \in C_i$, in the initialization phase of the algorithm, and to keep the results in a bi-dimensional array at cluster level. This operation performed in the case of cluster C_3 of the instance illustrated in Figure 1 with the different source nodes highlighted, is shown in Figure 12.

The costs of the routes from each node $u \in C_i$ to the source node $s_i \in C_i$ of the cluster they belong to, can be

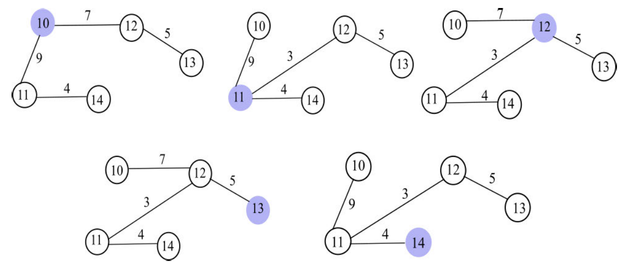
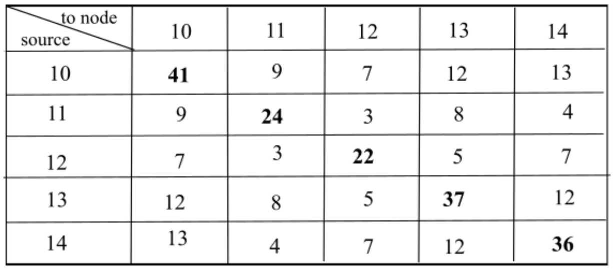


FIGURE 12. Spanning trees in the case of cluster C_3 of the instance illustrated Figure 1.



to node / source	10	11	12	13	14
10	**41**	9	7	12	13
11	9	**24**	3	8	4
12	7	3	**22**	5	7
13	12	8	5	**37**	12
14	13	4	7	12	**36**

FIGURE 13. Cost array associated to the cluster C_3 in the case of the instance presented in Figure 1.

evaluated only once, in the initialization phase, and stored in a bi-dimensional array of costs at cluster level. For the cost from the source node s_i to itself, we store the total internal costs of the cluster C_i, determined for the case when the source node is s_i. The costs array in the case of cluster C_3 of the instance illustrated in Figure 1 is shown in Figure 13.

D. FITNESS FUNCTION

The fitness of each new created chromosome throughout the optimization process is evaluated by determining the cost of the CluSPTP solution corresponding to the chromosome, as shown in Section III.B. The cost of the CluSPTP solution that gives the fitness of a chromosome is given by relation (6). For example, the feasible solution of the CluSPTP corresponding to the instance from Figure 1 generated using the chromosome illustrated in Figure 4, given by relation (6) is: $TotC = 5 \times 0 + 14 + 4 \times 7 + 18 + 5 \times 11 + 41 + 4 \times 15 + 28 + 3 \times 21 + 11 + 2 \times 17 + 6 = 358$, the values of n_i, $d_T(s, s_i)$ and cl_i are given in Figure 11.

E. INITIAL POPULATION

Choosing the initial population is a very important step of the GA because it directly affects the quality of the results. In the literature there are described two procedures for generating the initial population, both having their own advantages and disadvantages. The first procedure is random generation and the second is based on heuristics. Random generation has the advantage of better covering the solutions space, but the convergence of the genetic algorithm is slower. The second procedure has the advantage of faster convergence, but it does not cover the entire solutions space. The initial population in our GA is composed of two parts: a constructed part and a random part.

The first chromosome in the constructed part of the initial population is created using the greedy algorithm described in Section II. This algorithm determines the source node s_i for each cluster C_i, $i \in \{1, 2, \ldots, k\}$. The genes array of the first constructed chromosome are initialized based this information, as follows: the g_{ij} element on line i and column j, $i > j$ corresponding to the pair of clusters C_i, C_j is the edge connecting the nodes s_i and s_j. If there is no such edge in G, then g_{ij} is void. If the graph represented by the constructed chromosome contains isolated clusters, then the chromosome is rejected and no other constructed chromosomes are generated.

Otherwise the constructed part of the initial population is completed with a set of modified variants of the first constructed chromosome, built as follows:

a) Consider the root nodes s_i, $i \in \{1, 2, \ldots, k\}$ already determined when building the first constructed chromosome.
b) Randomly choose n_c clusters, different from the source cluster, where n_c is a random integer, $n_c \in [1, k-1]$;
c) Change the root node or each of the chosen clusters with another node selected randomly from the same cluster;
d) Build the genes of the chromosome as described above.

The genes arrays of the chromosomes in the random part of the initial population are created element-by-element as follows: the element on line i and column j, $i > j$ is a randomly chosen edge from the instance, edge that connects a node in cluster C_i with a node in cluster C_j. 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.

The initial population is processed by the selection mechanism, resulting the current population. Because the constructed chromosomes are much better than the random ones, in order to avoid the premature convergence of the algorithm, the selection operator ignores them until one of the following conditions is met:

- The best offspring created by the crossover operator has better fitness than the best constructed chromosome.
- Chromosome evolution stagnated during the last generations.

F. 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 other chromosomes are discarded.

G. CROSSOVER

The crossover mechanism selects from the current population two parents P_1 and P_2, which are used to create an offspring. The first parent is always chosen randomly from the best 20% chromosomes in the current population, and the second parent

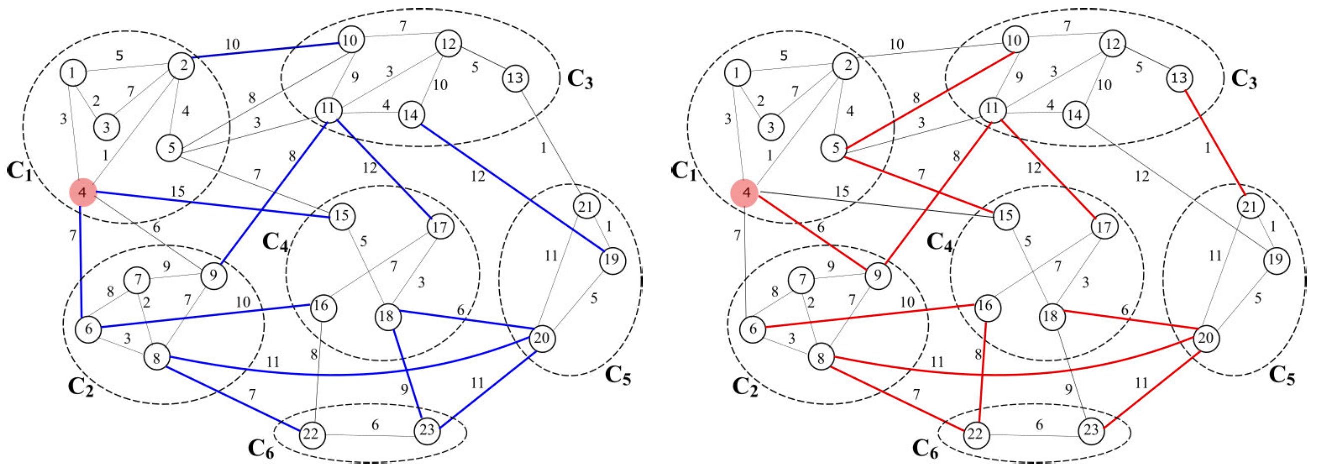


FIGURE 14. Example of two parrent chromosomes P_1 and P_2.

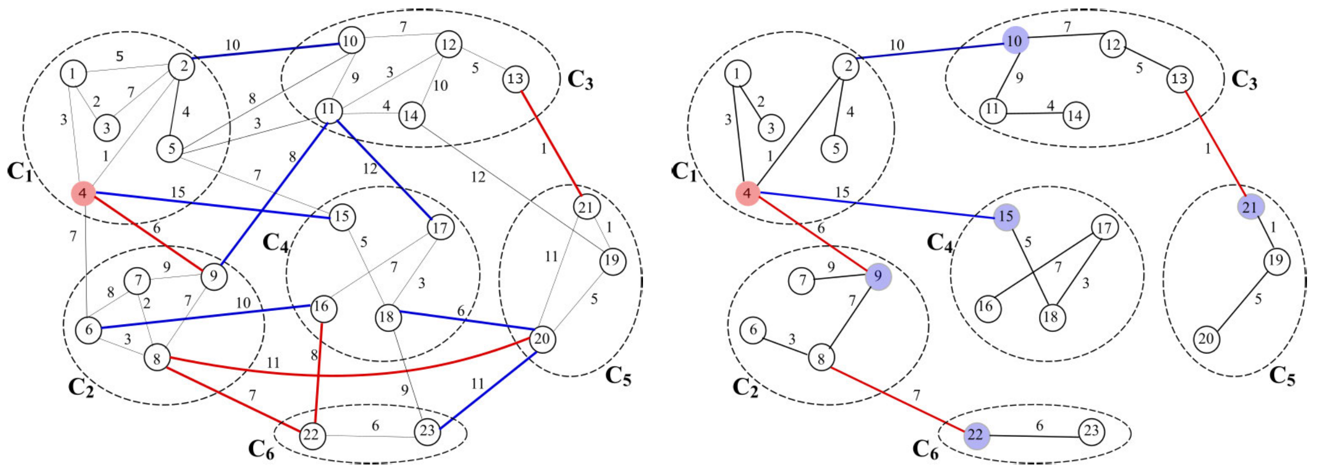


FIGURE 15. The resulting offspring after applying the crossover operator and its corresponding feasible solution of the CluSPTP.

is chosen randomly from the entire population. The genes of the offspring are selected either from P_1 or from P_2 with equal probabilities.

The operation of the crossover operator is illustrated in Figures 14-15. Figure 14 shows two parent chromosomes, and in Figure 15 is illustrated the resulting offspring after applying the crossover genetic operator and its corresponding feasible solution of the CluSPTP. The cost of the feasible solution of the CluSPTP corresponding to the offspring resulted after applying the crossover operator, presented in Figure 15, is $TotC = 372$.

The new generation of chromosomes is processed by the selection mechanism, resulting a new current population.

H. 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.

The operation of the mutation operator is illustrated in Figure 16. In the left part is shown the resulting offspring after applying the mutation operator on the offspring illustrated in Figure 15, and in the right part is shown its corresponding feasible solution of the CluSPTP.

The cost of the feasible solution of the CluSPTP corresponding to the offspring resulted after applying the mutation operator, presented in Figure 16, is $TotC = 349$.

Typically, in genetic algorithms, the mutation operator performs significant changes to the chromosome data, but it is applied with a low probability. We propose a different strategy in which the 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.

I. GENETIC PARAMETERS

The genetic parameters have an important impact on the performance of the GAs. That is why in our developed GA

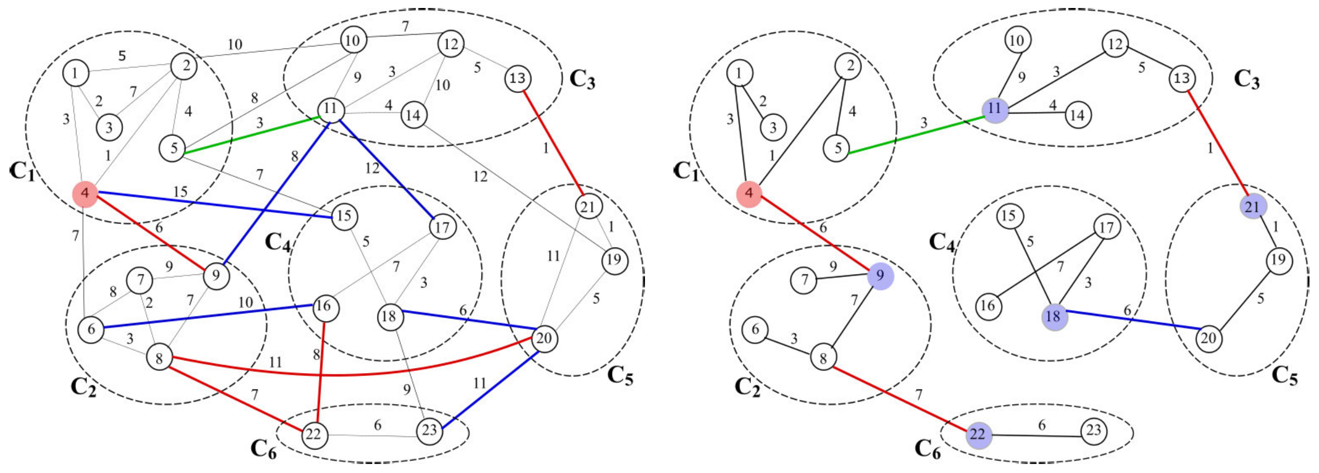


FIGURE 16. The resulting offspring after applying the mutation operator and its corresponding feasible solution of the CluSPTP.

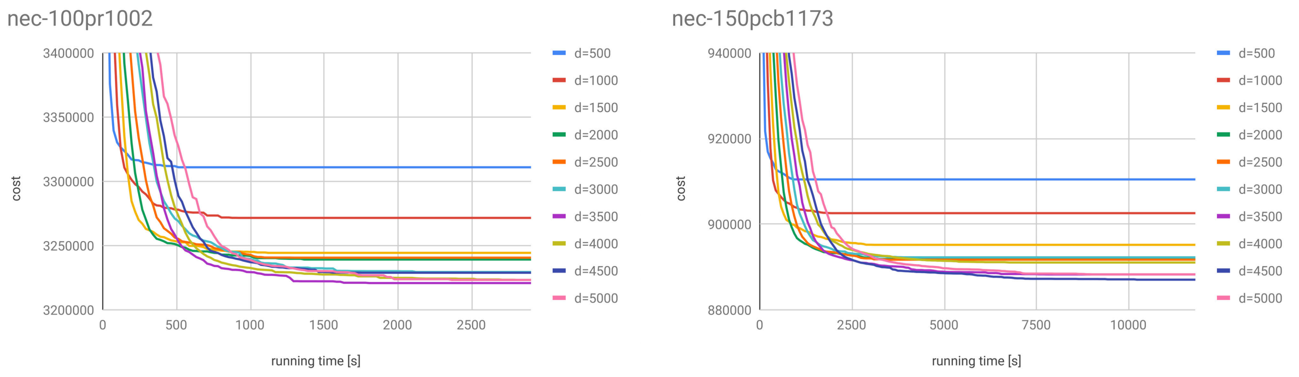


FIGURE 17. Convergence study for two instance with 100 and 150 clusters.

the values of the parameters have been chosen based on preliminary computational experiments and statistical analysis. The parameters have been chosen as follows:

The dimension of the current population D was selected to be 3500 in order to provide sufficient diversity and to allow the exploration of the entire solutions space. The size of the current population affects both the convergence of the algorithm and the quality of the solutions. Decreasing the size of the current population sppeds up convergence, but the quality of the solutions worsense, because the search space is explored less thoroughly. Increasing the size of the current population has oposite effects. In Figure 17, we show some partial results of the convergence study that we performed in order to choose the dimension of the current population. The plots show the evolution of the best solution cost in time for different population dimensions.

The other parameters of the genetic algorithm were chosen based on experiments, as follows: The number of constructed chromosomes in the initial population is at most D / 100 and the number of random chromosomes is $3 \times D$. Chromosome evolution is considered to be stagnant when the best solution was not improved over the last 15 generations of chromosomes. The algorithm is stopped when the best known solution was not improved over the last 30 generations of chromosomes. Because we apply weak mutations, the mutation probability is 1. The number of crossover operations performed for completing each new generation of chromosomes is $3 \times D$.

IV. COMPUTATIONAL EXPERIMENTS

This section contains the extensive 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 point out that all the existing benchmark instances from the literature are euclidean and defined on complete graphs and therefore are solved optimally by the first constructed chromosome created according to the proposed algorithm.

For testing the performance of our proposed GA, we compared it to the existing state-of-the-art algorithms for solving the CluSPTP:

- the evolutionary algorithm developed by Binh *et al.* [2] and denoted *NEA*;
- the multifactorial evolutionary algorithm proposed by Thanh *et al.* [24] and denoted *N-MFEA*.

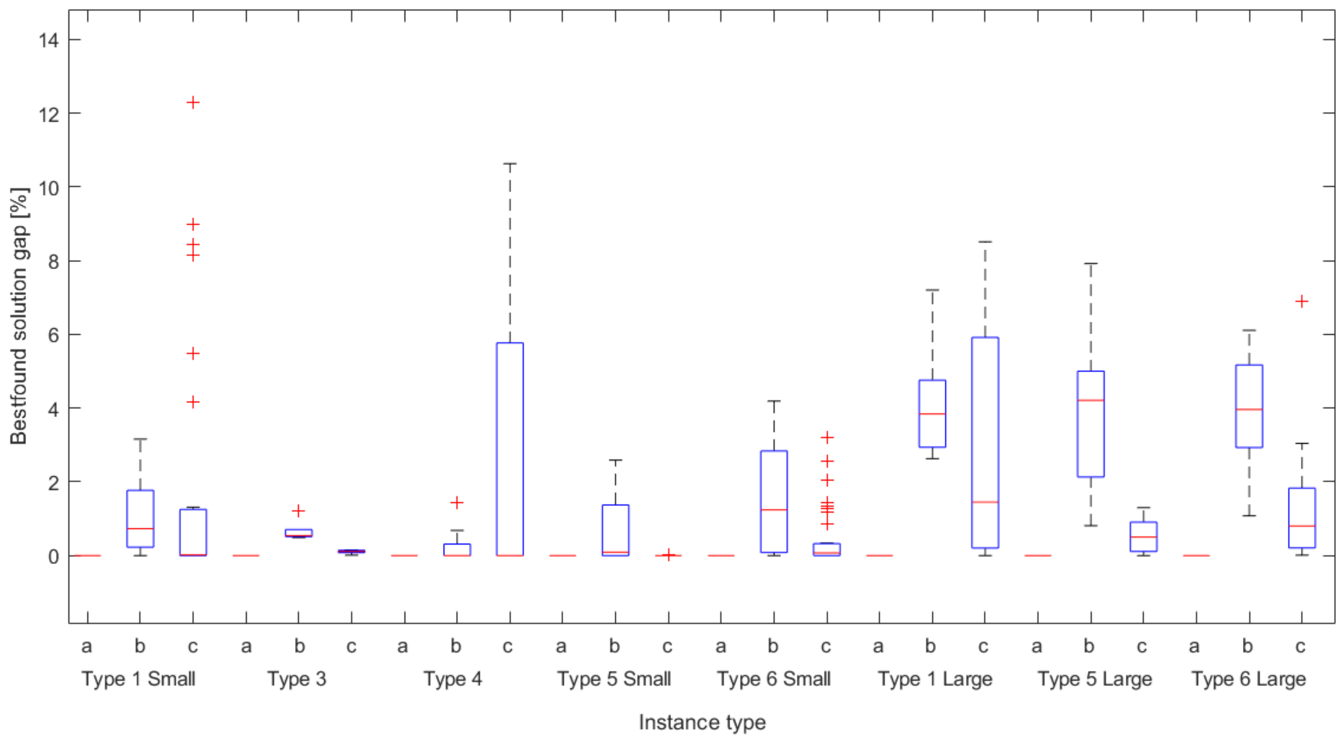


FIGURE 18. **Bestfound solutions accuracy: a) Our proposed algorithm, b) NEA, c) N-MFEA.**

Our proposed genetic algorithm was implemented in Java and has been tested on a PC with Intel Core i3-8100 @ 3.6GHz, 8GB RAM, Windows 10 Education 64 bit operating system. In our GA, for each instance, we carried out the same number of experiments Binh *et al.* [2] and Thanh *et al.* [24] did, namely 30 independent trials.

A. 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 248 benchmark instances contained in the MOM-lib provided by Mestria *et al.* [14] in the case of the Clustered Traveling Salesman Problem and used by Binh *et al.* [2] and Thanh *et al.* [24] in their computational experiments. The MOM-lib contains six kinds of instances which were obtained using different algorithms, see for more details [14], and classified into two groups according to the dimension: small instances containing nodes ranging beetwen 30 and 120 vertices grouped within a number of clusters ranging from 2 to 42, and large instances containing nodes ranging beween 108 and 3000 divided into a number of clusters ranging from 4 to 200. The source node was selected randomly for each of the considered instances.

Our proposed genetic algorithm delivered the optimal solutions in less than 1 millisecond for all the 248 benchmark euclidean instances. Details and comparison with existing solutions from lierature are presented in tabels 10-18 from Appendix A.

In Figures 18 and 19, we present a statistical analysis of our proposed genetic algorithm results in comparison to the state-of-the-art existing solution approaches from the literature: the evolutionary algorithm developed by Binh *et al.* [2] and the multifactorial evolutionary algorithm proposed by Thanh *et al.* [24]. The best found solution gaps are presented in Figure 18, and the average solution gaps are presented in Figure 19. A separate box and whisker plot was represented for each instance type and for each of the three algorithms: our proposed genetic algorithm represented by *a*, NEA algorithm [2] represented by *b* and N-MFEA algorithm [24] represented by *c*. It can be easily observed that our algorithm finds each time the optimal solution, that is why, the box and whisker plots are reduced to single lines in both representations. In the case of the NEA and M-FEA algorithms, the accuracy of the solutions is far from constant. The results seem to be better in the case of small instances, but there are exceptions even for those types. The gaps are greater in the case of the large instance types. The results for large instances of Type 5 look better then the rest, but this probably is explained by the fact that only the first smaller instances of this type were tested.

Taking into account the results displayed in Appendix A, Tables 10-18 and the presented statistical analysis, we can conclude that our novel genetic algorithm outperforms the state-of-the-art existing solution approaches from the literature: the evolutionary algorithm developed by Binh *et al.* [2] and the multifactorial evolutionary algorithm proposed by Thanh *et al.* [24], both in terms of the quality of the

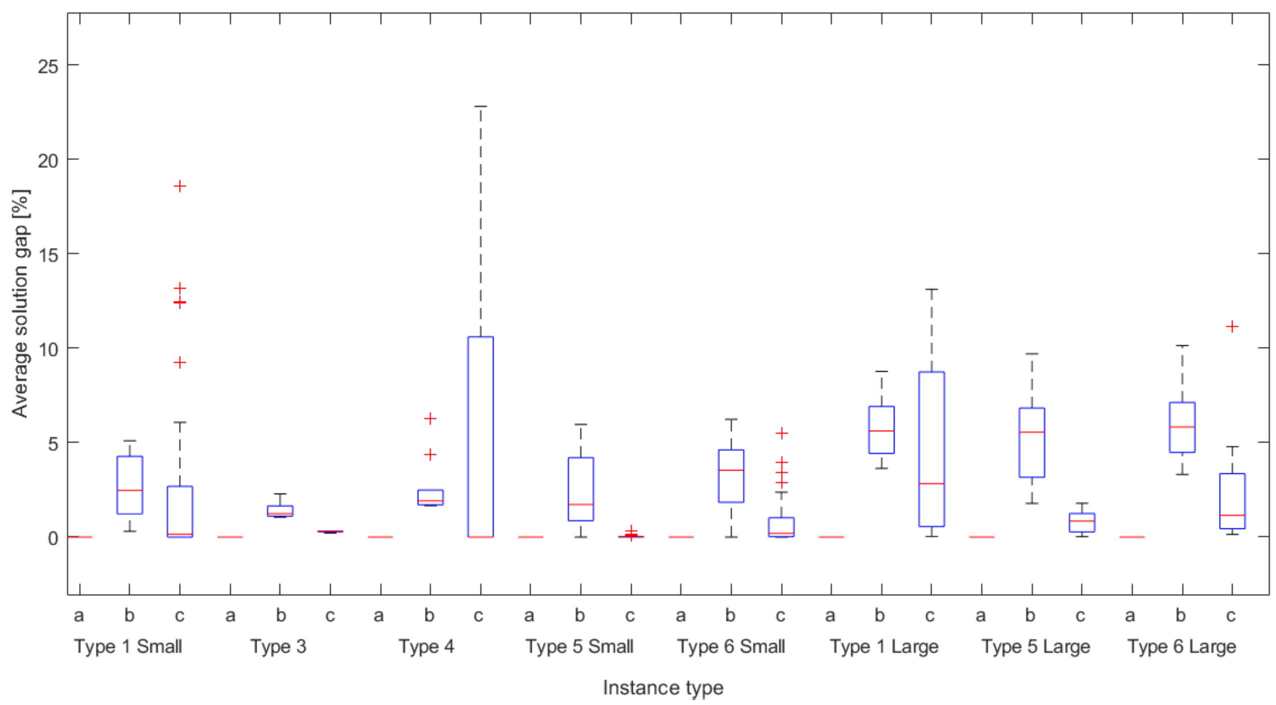


FIGURE 19. Average solutions accuracy: a) Our proposed algorithm, b) NEA, c) N-MFEA.

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

Instance				GA-CSPTP				Our GA			
No.	Name	k	n	BF	Avg.	ART [s]	gap [%]	BF	Avg.	ART [s]	gap [%]
1.	nec-5eil51	5	51	-	-	-	-	986	=	0.23	0
2.	nec-5berlin52	5	52	-	-	-	-	13456	=	0.23	0
3.	nec-5st70	5	70	-	-	-	-	2398	=	0.27	0
4.	nec-5eil76	5	76	-	-	-	-	1491	=	0.25	0
5.	nec-5pr76	5	76	-	-	-	-	345489	=	0.28	0
6.	nec-10eil51	10	51	1009	=	0.6	0	1009	=	0.78	0
7.	nec-10berlin52	10	52	28027	=	0.6	0	28027	=	1.27	0
8.	nec-10st70	10	70	1628	=	0.6	0	1628	=	0.97	0
9.	nec-10eil76	10	76	1258	=	0.6	0	1258	=	1.25	0
10.	nec-10pr76	10	76	-	-	-	-	297535	=	1.03	0
11.	nec-10rat99	10	99	4111	=	0.6	0	4111	=	1.15	0
12.	nec-10kroB100	10	100	76274	=	0.6	0	76274	=	1.40	0
13.	nec-15eil51	15	51	867	=	0.6	0	867	=	1.88	0
14.	nec-15berlin52	15	52	16836	=	1.2	0	16836	=	2.33	0
15.	nec-15st70	15	70	2204	=	1.2	0	2204	=	2.29	0
16.	nec-15eil76	15	76	1578	=	1.2	0	1578	=	2.54	0
17.	nec-15pr76	15	76	404626	=	1.8	0	404626	=	3.98	0
18.	nec-25rat99	25	99	3742	=	4.2	0	3742	=	8.77	0
19.	nec-25kroA100	25	100	86971	87050.00	4.2	0.09	86971	=	9.78	0
20.	nec-25eil101	25	101	2514	2519.50	5.4	0.21	2514	2516.40	12.26	0.10
21.	nec-25lin105	25	105	56371	56393.50	6.0	0.04	56371	=	13.27	0
22.	nec-50rat99	50	99	4651	=	39.6	0	4651	=	37.32	0
23.	nec-50kroA100	50	100	92131	92203.50	52.8	0.07	92131	92161.40	78.54	0.03
24.	nec-50kroB100	50	100	78632	=	36.0	0	78632	=	34.13	0
25.	nec-50eil101	50	101	2035	2036.00	71.4	0.04	2035	=	70.94	0
26.	nec-50lin105	50	105	80785	=	48.6	0	80785	=	46.98	0
27.	nec-75lin105	75	105	-	-	-	-	87793	=	61.24	0

achieved solutions and the corresponding computational times, providing in all 248 euclidean instances the optimal solutions in all 30 runs.

B. COMPUTATIONAL RESULTS ON NON-EUCLIDEAN INSTANCES

In general the CluSPTP is $\mathcal{NP}$-*hard*, but as we have already seen in the particular case when the underlying graph is complete and euclidean, the problem is solvable in polynomial time, that is why we transformed the 248 euclidean instances contained in the MOM-lib into non-euclidean instances. The transformation is done as follows:

```
a) for each edge $e$ of $G$
      if $c_e \neq 0$
         $r \leftarrow random\ value \in [-0.5 \cdot c_e,\ 0.5 \cdot c_e]$
         $c_e \leftarrow max\{\lfloor c_e + r \rfloor,\ 1\}$
```

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

Instance				GA-CSPTP				Our GA			
No.	Name	k	n	BF	Avg.	ART [s]	gap [%]	BF	Avg.	ART [s]	gap [%]
1.	nec-10gil262	10	262	13804	13845.00	1.2	0.29	13804	=	2.57	0
2.	nec-10a280	10	280	13659	=	1.2	0	13659	=	2.12	0
3.	nec-10lin318	10	318	347715	349190.75	1.2	0.42	347715	=	2.56	0
4.	nec-10pr439	10	439	821867	826615.25	1.8	0.57	821867	=	2.75	0
5.	nec-10pcb442	10	442	289271	293341.50	1.8	1.40	289271	=	3.98	0
6.	nec-25gil262	25	262	15394	15434.00	9.0	0.26	15394	=	19.67	0
7.	nec-25a280	25	280	16115	16219.25	9.0	0.64	16116	=	23.52	0
8.	nec-25lin318	25	318	313801	314488.75	10.8	0.21	313801	=	21.14	0
9.	nec-25pr439	25	439	724265	727710.50	11.4	0.47	724265	724295.17	24.32	0.004
10.	nec-25pcb442	25	442	368212	369320.75	13.8	0.30	368116	368295.33	26.58	0.05
11.	nec-50gil262	50	262	14234	14244.50	93.6	0.07	14234	14236.73	131.15	0.02
12.	nec-50a280	50	280	19630	19634.00	162.6	0.02	19626	19630.43	171.02	0.02
13.	nec-50lin318	50	318	363560	363572.50	126.0	0.00	363560	=	108.75	0
14.	nec-50pr439	50	439	-	-	-	-	1082808	1082968.07	192.39	0.01
15.	nec-50pcb442	50	442	492495	496873.25	168.6	0.88	492698	494273.67	205.67	0.32
16.	nec-50rat783	50	783	-	-	-	-	69235	69582.37	243.40	0.50
17.	nec-50pr1002	50	1002	2604500	2637842.00	265.8	1.28	2613692	2627770.20	294.48	0.54
18.	nec-50vm1084	50	1084	-	-	-	-	4200486	4245408.47	440.09	1.07
19.	nec-50pcb1173	50	1173	462554	471152.00	289.8	1.85	463339	467000.67	248.85	0.79
20.	nec-50nrw1379	50	1379	741498	762543.75	496.2	2.83	743859	752114.87	473.32	1.11
21.	nec-100rat783	100	783	89422	89543.50	1849.8	0.13	89438	89506.03	1506.37	0.08
22.	nec-100pr1002	100	1002	3215011	3223283.75	2137.2	0.25	3211197	3220590.70	1916.91	0.29
23.	nec-100vm1084	100	1084	4244669	4260804.00	2442.4	0.38	4241792	4250280.80	2299.97	0.20
24.	nec-150rat783	150	783	-	-	-	-	106000	106218.20	5290.57	0.21
25.	nec-150pr1002	150	1002	-	-	-	-	3136121	3139666.37	4411.16	0.11
26.	nec-150vm1084	150	1084	-	-	-	-	3993644	3998988.90	5283.85	0.13
27.	nec-150pcb1173	150	1173	-	-	-	-	886036	887874.20	6370.97	0.21
28.	nec-150nrw1379	150	1379	-	-	-	-	656521	658287.07	5734.60	0.27

TABLE 3. Experimental results in the case of large non-euclidean instances of Type 2.

Instance				Our GA			
No.	Name	k	n	BF	Avg.	ART [s]	gap [%]
1.	nec-10C1k0	10	1000	263891794	265502558.80	5.24	0.61
2.	nec-10C1k1	10	1000	237069395	237845511.20	4.94	0.33
3.	nec-10C1k2	10	1000	333388260	335088466.73	5.52	0.51
4.	nec-10C1k3	10	1000	260223276	261216537.03	5.58	0.38
5.	nec-10C1k4	10	1000	241189191	241477693.80	5.00	0.12
6.	nec-10C1k5	10	1000	255156519	256874983.37	5.32	0.67
7.	nec-10C1k6	10	1000	217947962	221726158.33	5.82	1.73
8.	nec-10C1k7	10	1000	144634553	145960903.43	6.10	0.92
9.	nec-10C1k8	10	1000	259894281	260710060.47	4.36	0.31
10.	nec-10C1k9	10	1000	173576174	176729302.83	5.65	1.82

TABLE 4. Experimental results in the case of large non-euclidean instances of Type 3.

Instance				Our GA			
No.	Name	k	n	BF	Avg.	ART [s]	gap [%]
1.	nec-6i300	6	300	9053	=	0.83	0
2.	nec-6i350	6	350	10001	=	0.77	0
3.	nec-6i400	6	400	12537	=	0.95	0
4.	nec-6i450	6	450	14887	=	0.97	0
5.	nec-6i500	6	500	15394	=	1.08	0
6.	nec-20i550	20	550	16980	16991.20	12.74	0.07
7.	nec-20i600	20	600	18186	18202.57	13.83	0.09
8.	nec-20i650	20	650	21055	21081.07	13.89	0.12
9.	nec-20i700	20	700	24000	24066.13	14.09	0.28
10.	nec-25i750	25	750	25730	25787.17	30.81	0.22
11.	nec-25i850	25	850	39104	39259.10	31.68	0.40
12.	nec-25i900	25	900	38424	38562.30	34.32	0.36
13.	nec-30i950	30	950	26643	26760.43	45.75	0.44
14.	nec-30i1000	30	1000	31934	32029.90	50.86	0.30

```
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 $c_e \neq 0$
         $r \leftarrow random\ value \in [0,\ 0.75 \cdot c_e]$
         $c_e \leftarrow max\{\lfloor c_e - r \rfloor,\ 1\}$
```

All the non-euclidean instances used in the experiments are available at https://sites.google.com/view/tstp-instances.

In Tables 1 - 9, we report the solutions achieved by our GA for solving 248 non-euclidean instances of different types of the CluSPTP. Tables 3–9 have the following structure: the first four columns indicate the number of the instance, its name and information about its size, the next

TABLE 5. Experimental results in the case of non-euclidean instances of Type 4.

Instance				Our GA			
No.	Name	k	n	BF	Avg.	ART [s]	gap [%]
1.	nec-4i200a	4	200	45512	=	0.28	0
2.	nec-4i200h	4	200	36324	=	0.57	0
3.	nec-4i200x1	4	200	42180	=	0.29	0
4.	nec-4i200x2	4	200	49046	=	0.27	0
5.	nec-4i200z	4	200	49186	=	0.28	0
6.	nec-4i400a	4	400	79201	=	0.43	0
7.	nec-4i400h	4	400	81811	=	0.41	0
8.	nec-4i400x1	4	400	71695	=	0.39	0
9.	nec-4i400x2	4	400	55160	=	0.39	0
10.	nec-4i400z	4	400	64991	=	0.85	0
11.	nec-8i600a	8	600	141135	=	1.65	0
12.	nec-8i600h	8	600	127066	127073.07	1.99	0.01
13.	nec-8i600x1	8	600	97891	97922.97	2.82	0.03
14.	nec-8i600x2	8	600	116896	116910.30	2.16	0.01
15.	nec-8i600z	8	600	139438	139645.20	2.27	0.15
16.	nec-8i1000a	8	1000	192290	193042.53	3.63	0.39
17.	nec-8i1000h	8	1000	158706	158989.93	2.74	0.18
18.	nec-8i1000x1	8	1000	185199	186679.77	2.95	0.80
19.	nec-8i1000x2	8	1000	177655	178502.43	2.80	0.48
20.	nec-8i1000z	8	1000	181106	182332.87	4.34	0.68
21.	nec-10i1400	10	1400	253993	256398.30	6.16	0.95
22.	nec-10i1400a	10	1400	231915	234646.07	4.99	1.18
23.	nec-10i1400h	10	1400	184048	185591.03	5.02	0.84
24.	nec-10i1400x1	10	1400	200679	203111.50	6.31	1.21
25.	nec-10i1400x2	10	1400	241402	248176.50	5.80	2.81
26.	nec-10i1400z	10	1400	265138	270706.77	5.88	2.10
27.	nec-10i2000a	10	2000	360508	368620.63	7.03	2.25
28.	nec-10i2000h	10	2000	303988	308539.37	6.75	1.50
29.	nec-10i2000x1	10	2000	283301	286770.67	6.83	1.22
30.	nec-10i2000x2	10	2000	258427	263172.47	5.37	1.84
31.	nec-10i2000z	10	2000	252782	255917.27	6.21	1.24
32.	nec-20i2400a	20	2400	503378	509198.63	39.30	1.16
33.	nec-20i2400h	20	2400	438410	445948.00	41.04	1.72
34.	nec-20i2400x1	20	2400	433154	456269.23	54.76	5.34
35.	nec-20i2400x2	20	2400	611291	627790.80	50.79	2.70
36.	nec-20i2400z	20	2400	429869	437003.43	45.70	1.66
37.	nec-20i3000a	20	3000	479567	485993.10	32.96	1.34
38.	nec-20i3000h	20	3000	723205	745085.50	58.97	3.03
39.	nec-20i3000x1	20	3000	743400	775319.87	62.25	4.29
40.	nec-20i3000x2	20	3000	572503	591354.90	61.94	3.29
41.	nec-20i3000z	20	3000	618564	651109.83	56.74	5.26

TABLE 6. Experimental results in the case of small non-euclidean instances of Type 5.

Instance				Our GA			
No.	Name	k	n	BF	Avg.	ART [s]	gap [%]
1.	nec-5i30-17	5	30	7352	=	0.11	0
2.	nec-5i45-18	5	45	9336	=	0.22	0
3.	nec-5i60-21	5	60	14897	=	0.23	0
4.	nec-5i65-21	5	65	14493	=	0.26	0
5.	nec-5i70-21	5	70	18948	=	0.28	0
6.	nec-5i75-22	5	75	16200	=	0.28	0
7.	nec-5i90-33	5	90	25298	=	0.34	0
8.	nec-5i120-46	5	120	31947	=	0.48	0
9.	nec-7i30-17	7	30	11813	=	0.30	0
10.	nec-7i45-18	7	45	13374	=	0.36	0
11.	nec-7i60-21	7	60	21286	=	0.46	0
12.	nec-7i65-21	7	65	18889	=	0.63	0
13.	nec-7i70-21	7	70	24250	=	0.66	0
14.	nec-10i30-17	10	30	8713	=	0.78	0
15.	nec-10i45-18	10	45	15617	=	0.85	0
16.	nec-10i60-21	10	60	18397	=	1.08	0
17.	nec-10i65-21	10	65	24026	=	1.01	0
18.	nec-10i70-21	10	70	22866	=	1.35	0
19.	nec-10i75-22	10	75	36104	=	1.35	0
20.	nec-10i90-33	10	90	30948	=	2.16	0
21.	nec-10i120-46	10	120	48787	48818.53	1.85	0.06

TABLE 7. Experimental results in the case of large non-euclidean instances of Type 5.

Instance				Our GA			
No.	Name	k	n	BF	Avg.	ART [s]	gap [%]
1.	nec-5i300-108	5	300	73259	=	0.64	0
2.	nec-5i400-205	5	400	87279	=	0.63	0
3.	nec-5i500-304	5	500	78939	78949.73	1.57	0.01
4.	nec-10i300-109	10	300	45845	45884.00	3.34	0.09
5.	nec-10i400-206	10	400	94072	94072.73	2.51	0.001
6.	nec-10i500-305	10	500	161115	=	3.31	0
7.	nec-10i1000-407	10	1000	182251	183421.60	4.20	0.64
8.	nec-10i1500-503	10	1500	244955	247318.53	6.58	0.96
9.	nec-15i300-110	15	300	53120	=	6.12	0
10.	nec-15i400-207	15	400	79048	79091.30	9.16	0.05
11.	nec-15i500-306	15	500	145695	145708.67	7.16	0.01
12.	nec-20i300-111	20	300	80719	=	14.23	0
13.	nec-20i400-208	20	400	98971	98983.23	12.99	0.01
14.	nec-20i500-307	20	500	89619	89717.73	18.48	0.11
15.	nec-20i1000-408	20	1000	191464	192714.60	23.08	0.65
16.	nec-20i1500-504	20	1500	415449	417944.00	35.44	0.60
17.	nec-20i2000-602	20	2000	450664	454292.77	30.46	0.81
18.	nec-20i2500-706	20	2500	613847	622509.13	38.59	1.41
19.	nec-20i3000-801	20	3000	572242	578538.27	44.26	1.10
20.	nec-25i300-112	25	300	55598	55695.70	16.63	0.18
21.	nec-25i400-209	25	400	118592	=	21.56	0
22.	nec-25i500-308	25	500	138945	139508.10	35.80	0.41
23.	nec-50i1000-409	50	1000	238982	241053.00	297.77	0.87
24.	nec-50i1500-505	50	1500	472142	477473.50	430.99	1.13
25.	nec-50i2000-603	50	2000	434170	436681.93	386.23	0.58
26.	nec-50i2500-707	50	2500	462621	465181.67	452.64	0.55
27.	nec-50i3000-802	50	3000	747661	754642.60	517.24	0.93
28.	nec-100i1000-410	100	1000	279285	279888.93	1772.40	0.22
29.	nec-100i1500-506	100	1500	469067	474098.83	3097.30	1.07
30.	nec-100i2000-604	100	2000	472158	477880.60	3135.08	1.21
31.	nec-100i2500-708	100	2500	639662	645606.33	3397.53	0.93
32.	nec-100i3000-803	100	3000	834892	839982.50	3621.25	0.61
33.	nec-150i1000-411	150	1000	254575	255970.47	5462.28	0.55
34.	nec-150i1500-507	150	1500	381196	384321.87	7541.91	0.82
35.	nec-150i2000-605	150	2000	421839	425045.67	7873.26	0.76
36.	nec-150i2500-709	150	2500	536367	541061.23	8749.95	0.88
37.	nec-150i3000-804	150	3000	904898	909794.90	11116.84	0.54
38.	nec-200i2000-606	200	2000	729974	731800.40	19397.86	0.25
39.	nec-200i2500-710	200	2500	625816	632107.93	21267.49	1.01
40.	nec-200i3000-805	200	3000	793248	795579.83	22732.82	0.29

two columns contain the best found (BF) and average (Avg) solutions obtained by our proposed GA, then we provide the average running times (ART) necessary in order to achieve the corresponding solutions, reported in seconds, and the last column contains the percentage gap calculated as follows: $gap = 100 \times (Avg - BF)/BF$. Avg is the average of the solutions calculated in the 30 runs of each instance. The symbol "=" means that the algorithm found the same solution in each of the 30 runs, i.e. $BF = Avg$ and $gap = 0$.

Tables 1 and 2 have in the middle four additional columns, containing the computational results of the Genetic Algorithm for solving the CluSPTP (GA-CSPTP) proposed by Cosma *et al.* [3]. It can be easily observed that our algorithm outperforms the GA-CSPTP algorithm, in terms of solution qualities and gaps. For some instances (especialyy the smaller ones), Cosma *et al.* reported smaller computational times. This is explicable by the fact that they used smaller populations of chromosomes in their experiments, and the GA-CSPTP algorithm does not have a hybrid initial population.

Analyzing the computational results achieved by our genetic algorithm and reported in Tables 1-9 in the case of the 248 non-euclidean instances of different sizes and types we can observe that: in 119 out of 248 instances we obtained the same solutions in all 30 runs and, when the algorithm does not provide the same solutions in all the runs, the percentage gap is at most 1% for 98 instances, and for the remaining 31 instances the percentage gap ranges between 1.01% and 5.34%, facts that confirm the accuracy and robustness of the proposed solution approach. The necessary average computational times reported in seconds in order to achieve the corresponding solutions are bellow 10 seconds in 140 out of 248 instances, between 10 and 60 seconds for 51 instances, and for the other instances it is at most 22800 seconds.

Overall, the comparison between the proposed solution approach and the best existing algorithms for solving the CluSPTP can be summarized as follows:

1. In the case of euclidean instances our novel genetic algorithm outperforms the best existing solution approaches from the literature: the evolutionary algorithm

TABLE 8. Experimental results in the case of small non-euclidean instances of Type 6.

Instance				Our GA			
No.	Name	k	n	BF	Avg.	ART [s]	gap [%]
1.	nec-2lin105-2x1	2	105	42303	=	0.00	0
2.	nec-4eil51-2x2	4	51	951	=	0.13	0
3.	nec-4berlin52-2x2	4	52	12324	=	0.14	0
4.	nec-4eil76-2x2	4	76	1315	=	0.16	0
5.	nec-4pr76-2x2	4	76	225436	=	0.13	0
6.	nec-6berlin52-2x3	6	52	17574	=	0.34	0
7.	nec-6st70-2x3	6	70	1768	=	0.40	0
8.	nec-6pr76-2x3	6	76	348878	=	0.41	0
9.	nec-8berlin52-2x4	8	52	16250	=	0.69	0
10.	nec-9eil51-3x3	9	51	930	=	0.74	0
11.	nec-9st70-3x3	9	70	1664	=	1.01	0
12.	nec-9eil76-3x3	9	76	1663	=	0.92	0
13.	nec-9pr76-3x3	9	76	303398	=	0.88	0
14.	nec-9eil101-3x3	9	101	1705	=	1.22	0
15.	nec-10berlin52-2x5	10	52	16698	=	1.01	0
16.	nec-12eil51-3x4	12	51	1055	=	1.37	0
17.	nec-12st70-3x4	12	70	2237	=	1.71	0
18.	nec-12eil76-3x4	12	76	1515	=	1.89	0
19.	nec-12pr76-3x4	12	76	341766	=	1.72	0
20.	nec-15pr76-3x5	15	76	338293	=	2.88	0
21.	nec-16eil51-4x4	16	51	867	=	2.17	0
22.	nec-16st70-4x4	16	70	1839	=	2.73	0
23.	nec-16eil76-4x4	16	76	1172	=	3.27	0
24.	nec-16lin105-4x4	16	105	73340	=	3.57	0
25.	nec-18pr76-3x6	18	76	376628	=	4.60	0
26.	nec-20eil51-4x5	20	51	1314	=	3.78	0
27.	nec-20st70-4x5	20	70	1854	=	4.10	0
28.	nec-20eil76-4x5	20	76	1420	=	5.22	0
29.	nec-25eil51-5x5	25	51	857	=	5.55	0
30.	nec-25eil76-5x5	25	76	1271	=	8.16	0
31.	nec-25rat99-5x5	25	99	6159	=	18.05	0
32.	nec-35kroB100-5x5	25	100	74344	=	12.87	0
33.	nec-25eil101-5x5	25	101	2051	=	12.99	0
34.	nec-28kroA100-4x7	28	100	87762	87781.67	16.65	0.02
35.	nec-30kroB100-5x6	30	100	115511	=	29.38	0
36.	nec-36eil101-6x6	36	101	1887	=	30.49	0
37.	nec-42rat99-6x7	42	99	5216	=	43.22	0

developed by Binh *et al.* [2] and the multifactorial evolutionary algorithm proposed by Thanh *et al.* [24], in terms of the quality of the achieved solutions and the corresponding computational times, providing in all the euclidean instances the optimal solutions in all 30 runs.

2. In the case of of non-euclidean instances our GA achieved in 119 out 248 instances the same best solution in all the 30 runs, in 98 out of 248 the percentage is at most 1%, and for the other instances the percentage gap is at most 5.34%, which confirms the robustness of our proposed solution approach. The necessary average computational times reported in seconds in order to achieve the corresponding solutions are bellow 60 seconds in 191 out of 248 instances, and for the other instances it is at most 22800 seconds.

V. CONCLUSION

This paper investigates an extended variant of the classical single-source shortest-path problem, called the clustered shortest-path tree problem (CluSPTP), motivated by some important applications in communication networks, agriculture irrigation, and distribution problems.

We have developed a novel genetic algorithm for solving the CluSPTP. Our proposed solution approach fits the challenges of the investigated problem and it has certain important characteristics: the use of an innovative representation scheme that enables us to construct easily feasible solutions of the CluSPTP and to explore efficiently the entire solution space of the problem, and the use of a seeded initial population that, in addition to the randomly selected individuals, contains feasible solutions generated by means of a heuristic algorithm.

An extensive computational experience on a set of 248 benchmark euclidean instances existing in the literature shows that our genetic algorithm obtained the optimal solutions in all 30 runs within 1 millisecond for all the instances, outperforming the best developed algorithms for solving CluSPTP in terms of both solution quality and length of computing-time required. Moreover, we provided a set of 248 non-euclidean instances and the reported achieved results confirm the accuracy and robustness of our proposed solution approach. Therefore, our proposed genetic algorithm may be considered as a new state-of-the-art heuristic.

Future work involves developing local search procedures that can help the algorithm to achieve better solutions in lower computing time in the case of non-euclidean instances.

TABLE 9. Experimental results in the case of large non-euclidean instances of Type 6.

Instance				Our GA			
No.	Name	k	n	BF	Avg.	ART [s]	gap [%]
1.	nec-9gil262-3x3	9	262	9516	=	1.73	0
2.	nec-9a280-3x3	9	280	12172	=	2.83	0
3.	nec-9lin318-3x3	9	318	263420	=	1.93	0
4.	nec-9pr439-3x3	9	439	767640	767647.53	2.24	0.001
5.	nec-9pcb442-3x3	9	442	321506	=	2.91	0
6.	nec-10nrw1379-2x5	10	1379	393137	398254.73	5.63	1.30
7.	nec-12nrw1379-2x6	12	1379	343396	348369.10	8.30	1.45
8.	nec-12nrw1379-3x4	12	1379	466256	469129.37	7.43	0.62
9.	nec-18pr439-3x6	18	439	610033	610799.20	10.26	0.13
10.	nec-20pr439-4x5	20	439	737111	737681.23	16.22	0.08
11.	nec-25gil262-5x5	25	262	14331	=	20.28	0
12.	nec-25a280-5x5	25	280	20345	20371.20	27.83	0.13
13.	nec-25lin318-5x5	25	318	325611	=	17.29	0
14.	nec-25pcb442-5x5	25	442	375947	376014.03	24.78	0.02
15.	nec-36pcb442-6x6	36	442	438012	438726.90	77.37	0.16
16.	nec-36pr1002-6x6	36	1002	2156193	2169679.00	114.76	0.63
17.	nec-42a280-6x7	42	280	22115	22192.93	133.60	0.35
18.	nec-42pr1002-6x7	42	1002	2792567	2805259.40	168.68	0.45
19.	nec-49gil262-7x7	49	262	16712	16741.80	142.64	0.18
20.	nec-49lin318-7x7	49	318	303362	=	106.00	0
21.	nec-49rat783-7x7	49	783	101269	101808.43	296.08	0.53
22.	nec-49pr1002-7x7	49	1002	3123335	3142783.37	328.97	0.62
23.	nec-49vm1084-7x7	49	1084	2984316	2998239.70	272.72	0.47
24.	nec-49pcb1173-7x7	49	1173	541704	545357.23	237.42	0.67
25.	nec-72vm1084-8x9	72	1084	3401796	3409284.70	619.69	0.22
26.	nec-81vm1084-9x9	81	1084	3568194	3585186.33	1301.81	0.48
27.	nec-100rat783-10x10	100	783	67279	67381.37	1265.36	0.15
28.	nec-100prb1173-10x10	100	1173	806417	808563.00	2095.92	0.27
29.	nec-144rat783-12x12	144	783	112081	112278.97	4038.80	0.18
30.	nec-144pcb1173-12x12	144	1173	604632	606658.97	4834.09	0.34

TABLE 10. Experimental results in the case of small euclidean instances of Type 1.

Instance				NEA [2]			N-MFEA [24]		Our GA	NEA gap		N-MFEA gap	
No	Name	k	n	BF	Avg	Time	BF	Avg	OPT	BFG	AvgG	BFG	AvgG
1.	5eil51	5	51	**1769.4**	1775.3	0.00	**1769.4**	**1769.4**	**1769.367**	0.00	0.34	0.00	0.00
2.	5berlin52	5	52	**22746.4**	22938.2	0.00	**22746.4**	**22746.4**	**22746.414**	0.00	0.84	0.00	0.00
3.	5eil76	5	76	**2630.8**	2693.1	0.00	**2630.8**	**2630.8**	**2630.839**	0.00	2.37	0.00	0.00
4.	5pr76	5	76	**585008.0**	591547	0.00	**585008.0**	**585008.0**	**585008.030**	0.00	1.12	0.00	0.00
5.	5st70	5	70	**4520.1**	4544.9	0.00	**4520.1**	**4520.1**	**4520.068**	0.00	0.55	0.00	0.00
6.	10eil51	10	51	1741.5	1770.6	0.02	**1713.2**	**1713.2**	**1713.246**	1.65	3.35	0.00	0.00
7.	10berlin52	10	52	43954.0	44237.6	0.02	**43724.1**	**43724.1**	**43724.060**	0.53	1.17	0.00	0.00
8.	10st70	10	70	3098.7	3191.1	0.02	**3095.2**	3095.7	**3095.238**	0.11	3.10	0.00	0.01
9.	10eil76	10	76	2264.5	2315.6	0.02	**2203.3**	**2203.3**	**2203.259**	2.78	5.10	0.00	0.00
10.	10pr76	10	76	531536.7	544954.5	0.02	**522213.8**	522340.4	**522213.794**	1.79	4.35	0.00	0.02
11.	10rat99	10	99	7697.8	7899.4	0.02	**7520.2**	7524.0	**7520.245**	2.36	5.04	0.00	0.05
12.	10kroB100	10	100	143108.6	147539.7	0.02	140551.2	140579.9	**140522.244**	1.84	4.99	0.02	0.04
13.	15eil51	15	51	1313.4	1336.5	0.03	1306.8	1309.1	**1306.421**	0.53	2.30	0.03	0.21
14.	15berlin52	15	52	26463.1	26867.8	0.03	26315.5	26351.7	**26311.973**	0.57	2.11	0.01	0.15
15.	15st70	15	70	4145.8	4230.1	0.03	4126.7	4135.5	**4120.066**	0.62	2.67	0.16	0.37
16.	15eil76	15	76	2955.3	3047.8	0.03	**2909.1**	2913.1	**2909.076**	1.59	4.77	0.00	0.14
17.	15pr76	15	76	714652.2	728128.0	0.03	705226.1	706505.5	**704600.556**	1.43	3.34	0.09	0.27
18.	25rat99	25	99	7056.0	7162.3	0.03	6930.9	7022.3	**6841.467**	3.14	4.69	1.31	2.64
19.	25kroA100	25	100	150157.7	153155.6	0.03	148767.9	149708.1	**147195.025**	2.01	4.05	1.07	1.71
20.	25eil101	25	101	4826.6	4885.5	0.03	4700.4	4727.9	**4678.965**	3.16	4.41	0.46	1.05
21.	25lin105	25	105	98991.8	100615.8	0.03	98941.4	100585.3	**97944.742**	1.07	2.73	1.02	2.70
22.	50rat99	50	99	8104.5	8132.4	0.08	8728.0	9002.1	**8007.438**	1.21	1.56	9.00	12.42
23.	50kroA100	50	100	160547.4	161889.6	0.07	173113.3	179506.1	**159647.239**	0.56	1.40	8.43	12.44
24.	50kroB100	50	100	134077.5	135332.2	0.07	149465.6	157831.1	**133104.503**	0.73	1.67	12.29	18.58
25.	50eil101	50	101	3890.7	3919.7	0.07	4034.7	4178.1	**3825.292**	1.71	2.47	5.47	9.22
26.	50lin105	50	105	146367.1	147175.4	0.07	151901.5	154680.7	**145829.067**	0.37	0.92	4.16	6.07
27.	75lin105	75	105	157234.0	157411.7	-	169739.8	177610.8	**156944.684**	0.18	0.30	8.15	13.17

APPENDIX

DETAILED RESULTS FOR EUCLIDEAN INSTANCES

Tables 10 – 18 display the optimal solutions achieved by our GA for solving the considered euclidean instances of the CluSPTP and, in addition, the results reported by Binh *et al.* [2] and Thanh *et al.* [24] for solving the problem with their evolutionary algorithm, respectively the multifactorial evolutionary algorithm. The Tables 10, 11, and 13 – 18 have the following structure: the first two columns indicate the number of the instance and its size, the third and fourth columns show the number of clusters (k) and the number of nodes (n). The next three columns contain

TABLE 11. Experimental results in the case of large euclidean instances of Type 1.

Instance				NEA [2]			N-MFEA [24]		Our GA	NEA gap		N-MFEA gap	
No	Name	k	n	BF	Avg	Time	BF	Avg	OPT	BFG	AvgG	BFG	AvgG
1.	10gil262	10	262	29075.0	29568.4	0.02	**27637.5**	27645.7	**27637.472**	5.20	6.99	0.00	0.03
2.	10a280	10	280	28690.9	29664.8	0.02	27936.1	28079.4	**27925.202**	2.74	6.23	0.04	0.55
3.	10lin318	10	318	832299.5	841893.2	0.02	812744.1	814264.5	**809749.985**	2.78	3.97	0.37	0.56
4.	10pr439	10	439	1971633.0	2022257.4	0.02	1907568.9	1911815.3	**1904690.238**	3.51	6.17	0.15	0.37
5.	10pcb442	10	442	765561.0	796960.4	0.02	742112.4	742678.9	**741195.810**	3.29	7.52	0.12	0.20
6.	25gil262	25	262	31579.5	31949.7	0.03	30695.8	30953.2	**30325.698**	4.13	5.36	1.22	2.07
7.	25a280	25	280	31481.2	32020.2	0.03	30373.2	30654.1	**29902.430**	5.28	7.08	1.57	2.51
8.	25lin318	25	318	607029.0	617399.9	0.03	593017.4	601118.8	**584554.046**	3.84	5.62	1.45	2.83
9.	25pr439	25	439	1585283.0	1612334.7	0.03	1531948.9	1556475.7	**1511168.935**	4.90	6.69	1.38	3.00
10.	25pcb442	25	442	794217.4	805896.7	0.03	757524.1	762707.5	**740892.565**	7.20	8.77	2.24	2.94
11.	50gil262	50	262	27647.5	27836.2	0.10	28780.1	30004.0	**26523.294**	4.24	4.95	8.51	13.12
12.	50a280	50	280	37458.4	37828.6	0.10	38596.8	39872.5	**36266.906**	3.29	4.31	6.42	9.94
13.	50lin318	50	318	706854.9	713744.5	0.10	730023.7	748203.2	**688724.632**	2.63	3.63	6.00	8.64
14.	50pr439	50	439	2213598.5	2232892.5	-	2302262.5	2375177.6	**2152986.562**	2.82	3.71	6.93	10.32
15.	50pcb442	50	442	949830.8	954169.0	0.10	962101.7	990436.0	**910478.667**	4.32	4.80	5.67	8.78
16.	50rat783	50	783	-	-	-	-	-	**141871.934**	-	-	-	-
17.	50pr1002	50	1002	-	-	-	-	-	**5243008.673**	-	-	-	-
18.	50vm1084	50	1084	-	-	-	-	-	**10144571.318**	-	-	-	-
19.	50pcb1173	50	1173	-	-	-	-	-	**1108183.787**	-	-	-	-
20.	50nrw1379	50	1379	-	-	-	-	-	**1831566.938**	-	-	-	-
21.	100rat783	100	783	-	-	-	-	-	**175893.992**	-	-	-	-
22.	100pr1002	100	1002	-	-	-	-	-	**6213697.088**	-	-	-	-
23.	100vm1084	100	1084	-	-	-	-	-	**8504736.073**	-	-	-	-
24.	150rat783	150	783	-	-	-	-	-	**204410.379**	-	-	-	-
25.	150pr1002	150	1002	-	-	-	-	-	**6258631.932**	-	-	-	-
26.	150vm1084	150	1084	-	-	-	-	-	**7741281.083**	-	-	-	-
27.	150pcb1173	150	1173	-	-	-	-	-	**1728190.771**	-	-	-	-
28.	150nrw1379	150	1379	-	-	-	-	-	**1261803.882**	-	-	-	-

the best found (*BF*) and average (*Avg*) 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. The next two columns contain the best found (*BF*) and average (*Avg*) solutions obtained by the algorithm developed by Thanh *et al.* [24]. Thanh *et al.* [24] did not provide any information regarding the computational times. The next column contains the optimal solutions (*OPT*) obtained by our proposed GA, that finds each of those solutions in less than 1 millisecond running time. The running time represents the time interval measured from the start of the algorithm, until the first apparition of the best solution. The next two columns contain the percentage gaps between the best found solutions (*BFG*) respectively the average solutions (*AvgG*) found by the NEA algorithm and the optimal solutions found by our proposed algorithm. The gaps were calculated as follows: $BFG = 100 \times (BF - OPT)/OPT$, $AvgG = 100 \times (Avg - OPT)/OPT$. The last two columns contain the gaps between the best found solutions (*BFG*) respectively the average solutions (*AvgG*) found by the N-MFEA algorithm and our achieved solutions. The "−" symbol means that the corresponding results were not provided by Binh *et al.* [2] or Thanh *et al.* [24]. Table 12 contains the results corresponding to the instances of Type 2, that have not been tested in the computational experiments of Binh *et al.* [2] and Thanh *et al.* [24]. The optimal solutions are marked with bold font.

Analyzing the computational results displayed in Table 10, one can notice that our proposed genetic algorithm delivered

TABLE 12. Experimental results in the case of euclidean instances of Type 2.

Instance				Our GA
No.	Name	k	n	OPT.
1.	10C1k0	10	1000	**603896769.135**
2.	10C1k1	10	1000	**555519829.513**
3.	10C1k2	10	1000	**740549053.578**
4.	10C1k3	10	1000	**594412757.232**
5.	10C1k4	10	1000	**533153746.936**
6.	10C1k5	10	1000	**582557709.033**
7.	10C1k6	10	1000	**580986510.077**
8.	10C1k7	10	1000	**343312412.637**
9.	10C1k8	10	1000	**564496866.867**
10.	10C1k9	10	1000	**423562402.544**

the optimal solutions in less than 1 millisecond for all the considered small euclidean instances of Type 1. The evolutionary algorithm developed by Binh *et al.* [2] provided the optimal solutions in 5 out of 27 instances within at most 0.08 minutes and the multifactorial evolutionary algorithm proposed by Thanh *et al.* [24] provided the optimal solutions in 12 out of 27 instances. The gaps between the best provided solution and the optimal solution ranges between 0 and 3.16% in the case of the NEA algorithm, and between 0 and 12.29% in the case of the N-MFEA algorithm. Our novel GA provided the optimal solution in all the 30 runs, while in the case of the evolutionary algorithm developed by Binh *et al.* [2] for all the instances the average solutions are different from the best solutions provided. In the case the multifactorial evolutionary algorithm proposed by Thanh *et al.* [24], in 8 out of 27 instances, the average solutions are equal to the

TABLE 13. Experimental results in the case of euclidean instances of Type 3.

Instance				NEA [2]			N-MFEA [24]		Our GA	NEA gap		N-MFEA gap	
No	Name	k	n	BF	Avg	Time	BF	Avg	OPT	BFG	AvgG	BFG	AvgG
1.	6i300	6	300	19358.8	19467.0	0.02	19286.3	19320.3	**19264.453**	0.49	1.05	0.11	0.29
2.	6i350	6	350	21472.8	21702.2	0.02	21218.5	21261.3	**21217.195**	1.20	2.29	0.01	0.21
3.	6i400	6	400	29506.9	29677.7	0.02	29389.5	29437.5	**29348.223**	0.54	1.12	0.14	0.30
4.	6i450	6	450	35866.3	36124.5	0.02	35715.7	35795.9	**35681.526**	0.52	1.24	0.10	0.32
5.	6i500	6	500	37711.6	38045.9	0.02	37567.6	37631.4	**37510.090**	0.54	1.43	0.15	0.32
6.	20i550	20	550	-	-	-	-	-	**35329.520**	-	-	-	-
7.	20i600	20	600	-	-	-	-	-	**38468.964**	-	-	-	-
8.	20i650	20	650	-	-	-	-	-	**45835.331**	-	-	-	-
9.	20i700	20	700	-	-	-	-	-	**51246.060**	-	-	-	-
10.	25i750	25	750	-	-	-	-	-	**55962.621**	-	-	-	-
11.	25i850	25	850	-	-	-	-	-	**87318.634**	-	-	-	-
12.	25i900	25	900	-	-	-	-	-	**88019.992**	-	-	-	-
13.	30i950	30	950	-	-	-	-	-	**59033.615**	-	-	-	-
14.	30i1000	30	1000	-	-	-	-	-	**72819.214**	-	-	-	-

TABLE 14. Experimental results in the case of euclidean instances of Type 4.

Instance				NEA [2]			N-MFEA [24]		Our GA	NEA gap		N-MFEA gap	
No	Name	k	n	BF	Avg	Time	BF	Avg	OPT	BFG	AvgG	BFG	AvgG
1.	4i200a	4	200	**97959.6**	102256.3	0.00	**97959.6**	**97959.6**	**97959.598**	0.00	4.39	0.00	0.00
2.	4i200h	4	200	**87675.3**	89628.9	0.00	**87675.3**	**87675.3**	**87675.308**	0.00	2.23	0.00	0.00
3.	4i200x1	4	200	**123669.7**	125782.7	0.00	**123669.7**	**123669.7**	**123669.702**	0.00	1.71	0.00	0.00
4.	4i200x2	4	200	**114012.3**	116256.5	0.00	**114012.3**	**114012.3**	**114012.325**	0.00	1.97	0.00	0.00
5.	4i200z	4	200	**131683.5**	133873.8	0.00	**131683.5**	**131683.5**	**131683.504**	0.00	1.66	0.00	0.00
6.	4i400a	4	400	217171.4	227530.6	0.02	**214115.3**	**214115.3**	**214115.333**	1.43	6.27	0.00	0.00
7.	4i400h	4	400	257954.4	260916.0	0.02	**256200.5**	256291.2	**256200.506**	0.68	1.84	0.00	0.04
8.	4i400x1	4	400	188786.2	191694.7	0.02	199389.3	222805.4	**188196.748**	0.31	1.86	5.95	18.39
9.	4i400x2	4	400	**159254.8**	163222.8	0.02	176188.2	195580.8	**159254.766**	0.00	2.49	10.63	22.81
10.	4i400z	4	400	221460.6	225096.3	0.02	234203.3	244894.6	**221423.874**	0.02	1.66	5.77	10.60
11.	8i600a	8	600	-	-	-	-	-	**375788.585**	-	-	-	-
12.	8i600h	8	600	-	-	-	-	-	**329585.905**	-	-	-	-
13.	8i600x1	8	600	-	-	-	-	-	**281682.096**	-	-	-	-
14.	8i600x2	8	600	-	-	-	-	-	**340116.559**	-	-	-	-
15.	8i600z	8	600	-	-	-	-	-	**419289.486**	-	-	-	-
16.	8i1000a	8	1000	-	-	-	-	-	**641409.736**	-	-	-	-
17.	8i1000h	8	1000	-	-	-	-	-	**438922.663**	-	-	-	-
18.	8i1000x1	8	1000	-	-	-	-	-	**630218.781**	-	-	-	-
19.	8i1000x2	8	1000	-	-	-	-	-	**576409.017**	-	-	-	-
20.	8i1000z	8	1000	-	-	-	-	-	**541788.654**	-	-	-	-
21.	10i1400	10	1400	-	-	-	-	-	**707420.264**	-	-	-	-
22.	10i1400a	10	1400	-	-	-	-	-	**691859.946**	-	-	-	-
23.	10i1400h	10	1400	-	-	-	-	-	**640241.458**	-	-	-	-
24.	10i1400x1	10	1400	-	-	-	-	-	**642774.652**	-	-	-	-
25.	10i1400x2	10	1400	-	-	-	-	-	**730597.004**	-	-	-	-
26.	10i1400z	10	1400	-	-	-	-	-	**832303.272**	-	-	-	-
27.	10i2000a	10	2000	-	-	-	-	-	**1284288.959**	-	-	-	-
28.	10i2000h	10	2000	-	-	-	-	-	**965681.410**	-	-	-	-
29.	10i2000x1	10	2000	-	-	-	-	-	**963188.941**	-	-	-	-
30.	10i2000x2	10	2000	-	-	-	-	-	**830906.177**	-	-	-	-
31.	10i2000z	10	2000	-	-	-	-	-	**858499.696**	-	-	-	-
32.	20i2400a	20	2400	-	-	-	-	-	**1360314.412**	-	-	-	-
33.	20i2400h	20	2400	-	-	-	-	-	**1342643.227**	-	-	-	-
34.	20i2400x1	20	2400	-	-	-	-	-	**1282515.614**	-	-	-	-
35.	20i2400x2	20	2400	-	-	-	-	-	**1768004.362**	-	-	-	-
36.	20i2400z	20	2400	-	-	-	-	-	**1367752.559**	-	-	-	-
37.	20i3000a	20	3000	-	-	-	-	-	**1430689.340**	-	-	-	-
38.	20i3000h	20	3000	-	-	-	-	-	**2236103.835**	-	-	-	-
39.	20i3000x1	20	3000	-	-	-	-	-	**2372172.744**	-	-	-	-
40.	20i3000x2	20	3000	-	-	-	-	-	**1568342.024**	-	-	-	-
41.	20i3000z	20	3000	-	-	-	-	-	**1735491.931**	-	-	-	-

corresponding best solutions, while in the remaining ones they are different.

When taking a closer look at the computational results shown in Table 11, we can observe that our proposed genetic algorithm delivered the optimal solutions in less than 1 millisecond for all the 28 large euclidean instances of Type 1. Binh *et al.* [2] provided the solutions only for the first 15 instances, and all the best solutions are different from the

TABLE 15. Experimental results in the case of small euclidean instances of Type 5.

Instance				NEA [2]			N-MFEA [24]		Our GA	NEA gap		N-MFEA gap	
No	Name	k	n	BF	Avg	Time	BF	Avg	OPT	BFG	AvgG	BFG	AvgG
1.	5i30-17	5	30	**14399.9**	**14399.9**	0.00	**14399.9**	**14399.9**	**14399.941**	0.00	0.00	0.00	0.00
2.	5i45-18	5	45	**14884.3**	14925.6	0.00	**14884.3**	**14884.3**	**14884.310**	0.00	0.28	0.00	0.00
3.	5i60-21	5	60	**28422.7**	28769.6	0.00	**28422.7**	**28422.7**	**28422.695**	0.00	1.22	0.00	0.00
4.	5i65-21	5	65	**30907.8**	31254.4	0.00	**30907.8**	30911.7	**30907.844**	0.00	1.12	0.00	0.01
5.	5i70-21	5	70	**35052.8**	35298.8	0.00	**35052.8**	**35052.8**	**35052.803**	0.00	0.70	0.00	0.00
6.	5i75-22	5	75	**34692.5**	35098.7	0.00	**34692.5**	**34692.5**	**34692.471**	0.00	1.17	0.00	0.00
7.	5i90-33	5	90	**51977.0**	52533.8	0.00	**51977.0**	51977.3	**51976.960**	0.00	1.07	0.00	0.00
8.	5i120-46	5	120	61695.7	62620.1	0.02	**61451.5**	61495.3	**61451.510**	0.40	1.90	0.00	0.07
9.	7i30-17	7	30	**20438.9**	20454.2	0.02	**20438.9**	**20438.9**	**20438.892**	0.00	0.07	0.00	0.00
10.	7i45-18	7	45	**20512.0**	20700.8	0.02	**20512.0**	**20512.0**	**20512.042**	0.00	0.92	0.00	0.00
11.	7i60-21	7	60	36295.4	37780.7	0.02	**36263.9**	**36263.9**	**36263.946**	0.09	4.18	0.00	0.00
12.	7i65-21	7	65	35201.2	36136.4	0.02	**34847.6**	**34847.6**	**34847.631**	1.01	3.70	0.00	0.00
13.	7i70-21	7	70	39613.4	40819.5	0.02	**39487.6**	39491.1	**39487.634**	0.32	3.37	0.00	0.01
14.	10i30-17	10	30	**13276.6**	13290.0	0.02	**13276.6**	**13276.6**	**13276.620**	0.00	0.10	0.00	0.00
15.	10i45-18	10	45	23227.3	23985.5	0.02	**22890.4**	22892.2	**22890.420**	1.47	4.78	0.00	0.01
16.	10i60-21	10	60	34147.0	35233.3	0.02	**33694.8**	33702.8	**33694.816**	1.34	4.57	0.00	0.02
17.	10i65-21	10	65	38318.8	39578.8	0.02	**37353.1**	37353.6	**37353.094**	2.59	5.96	0.00	0.00
18.	10i70-21	10	70	38816.6	39687.3	0.02	38066.7	38187.3	**38059.509**	1.99	4.28	0.02	0.34
19.	10i75-22	10	75	65923.2	66485.1	0.02	65362.0	65397.3	**65361.907**	0.86	1.72	0.00	0.05
20.	10i90-33	10	90	53076.0	54636.2	0.02	51943.2	51975.6	**51931.228**	2.20	5.21	0.02	0.09
21.	10i120-46	10	120	96168.2	97752.1	0.02	93956.9	94034.3	**93925.044**	2.39	4.07	0.03	0.12

TABLE 16. Experimental results in the case of large euclidean instances of Type 5.

Instance				NEA [2]			N-MFEA [24]		Our GA	NEA gap		N-MFEA gap	
No	Name	k	n	BF	Avg	Time	BF	Avg	OPT	BFG	AvgG	BFG	AvgG
1.	5i300-108	5	300	178628.1	180397.2	0.02	**177185.9**	177220.7	**177185.925**	0.81	1.81	0.00	0.02
2.	5i400-205	5	400	211603.0	213125.6	0.02	209488.0	209970.9	**209389.820**	1.06	1.78	0.05	0.28
3.	5i500-304	5	500	183656.4	185924.4	0.02	182206.2	182416.4	**182024.032**	0.90	2.14	0.10	0.22
4.	10i300-109	10	300	117421.2	119952.9	0.02	112876.2	113017.1	**112681.011**	4.21	6.45	0.17	0.30
5.	10i400-206	10	400	214604.4	217399.0	0.02	207778.5	208087.4	**207521.674**	3.41	4.76	0.12	0.27
6.	10i500-305	10	500	355952.4	359614.5	0.02	350897.4	351929.6	**349675.225**	1.80	2.84	0.35	0.64
7.	10i1000-407	10	1000	-	-	-	-	-	**443584.790**	-	-	-	-
8.	10i1500-503	10	1500	-	-	-	-	-	**598942.732**	-	-	-	-
9.	15i300-110	15	300	119922.2	122146.2	0.02	112935.9	113358.9	**112096.660**	6.98	8.97	0.75	1.13
10.	15i400-207	15	400	171349.6	175081.7	0.02	165328.2	165854.9	**164117.828**	4.41	6.68	0.74	1.06
11.	15i500-306	15	500	310122.7	313184.9	0.02	304128.7	304949.4	**300734.099**	3.12	4.14	1.13	1.40
12.	20i300-111	20	300	163927.8	167104.4	0.03	157371.4	157990.3	**156347.689**	4.85	6.88	0.65	1.05
13.	20i400-208	20	400	231753.3	236373.9	0.03	226383.9	227069.5	**224012.479**	3.46	5.52	1.06	1.36
14.	20i500-307	20	500	212306.3	215644.7	0.03	202938.5	203910.7	**200328.742**	5.98	7.65	1.30	1.79
15.	20i1000-408	20	1000	-	-	-	-	-	**467562.362**	-	-	-	-
16.	20i1500-504	20	1500	-	-	-	-	-	**917765.230**	-	-	-	-
17.	20i2000-602	20	2000	-	-	-	-	-	**1005626.841**	-	-	-	-
18.	20i2500-706	20	2500	-	-	-	-	-	**1373157.854**	-	-	-	-
19.	20i3000-801	20	3000	-	-	-	-	-	**1269081.272**	-	-	-	-
20.	25i300-112	25	300	125392.7	127466.5	0.05	-	-	**116193.577**	7.92	9.70	-	-
21.	25i400-209	25	400	241529.0	243994.1	0.05	-	-	**229913.566**	5.05	6.12	-	-
22.	25i500-308	25	500	312805.6	316116.8	0.05	-	-	**299498.167**	4.44	5.55	-	-
23.	50i1000-409	50	1000	-	-	-	-	-	**524030.785**	-	-	-	-
24.	50i1500-505	50	1500	-	-	-	-	-	**1013700.525**	-	-	-	-
25.	50i2000-603	50	2000	-	-	-	-	-	**929872.368**	-	-	-	-
26.	50i2500-707	50	2500	-	-	-	-	-	**1067627.037**	-	-	-	-
27.	50i3000-802	50	3000	-	-	-	-	-	**1643325.261**	-	-	-	-
28.	100i1000-410	100	1000	-	-	-	-	-	**553975.609**	-	-	-	-
29.	100i1500-506	100	1500	-	-	-	-	-	**971706.810**	-	-	-	-
30.	100i2000-604	100	2000	-	-	-	-	-	**1041983.845**	-	-	-	-
31.	100i2500-708	100	2500	-	-	-	-	-	**1403934.538**	-	-	-	-
32.	100i3000-803	100	3000	-	-	-	-	-	**1821678.317**	-	-	-	-
33.	150i1000-411	150	1000	-	-	-	-	-	**497818.548**	-	-	-	-
34.	150i1500-507	150	1500	-	-	-	-	-	**782859.997**	-	-	-	-
35.	150i2000-605	150	2000	-	-	-	-	-	**882270.309**	-	-	-	-
36.	150i2500-709	150	2500	-	-	-	-	-	**1124133.647**	-	-	-	-
37.	150i3000-804	150	3000	-	-	-	-	-	**1928332.948**	-	-	-	-
38.	200i2000-606	200	2000	-	-	-	-	-	**1475366.866**	-	-	-	-
39.	200i2500-710	200	2500	-	-	-	-	-	**1355011.159**	-	-	-	-
40.	200i3000-805	200	3000	-	-	-	-	-	**1601021.239**	-	-	-	-

optimal ones. The gap between the best provided solution and the optimal solution ranges between 2.63% and 7.20%. The computational times range between 0.02 minutes and 0.10 minutes. Thanh *et al.* [24] also provided solutions only for the first 15 instances, and 14 of the best solutions are different from the optimal ones. The gap between the best provided solution and the optimal solution ranges between 0 and 8.51%.

TABLE 17. Experimental results in the case of small euclidean instances of Type 6.

Instance				NEA [2]			N-MFEA [24]		Our GA	NEA gap		N-MFEA gap	
No	Name	k	n	BF	Avg	Time	BF	Avg	OPT	BFG	AvgG	BFG	AvgG
1.	2lin105-2x1	2	105	**152729.7**	**152729.7**	0.00	-	-	**152729.676**	-	-	-	-
2.	4eil51-2x2	4	51	**1898.5**	1901.3	0.00	**1898.5**	**1898.5**	**1898.544**	0.00	0.15	0.00	0.00
3.	4berlin52-2x2	4	52	**23287.9**	**23287.9**	0.00	**23287.9**	**23287.9**	**23287.916**	0.00	0.00	0.00	0.00
4.	4eil76-2x2	4	76	**2948.7**	2955.8	0.00	**2948.7**	**2948.7**	**2948.744**	0.00	0.24	0.00	0.00
5.	4pr76-2x2	4	76	**442693.0**	446682.1	0.00	**442693.0**	**442693.0**	**442692.994**	0.00	0.90	0.00	0.00
6.	6berlin52-2x3	6	52	**32128.6**	32354.7	0.00	**32128.6**	**32128.6**	**32128.574**	0.00	0.70	0.00	0.00
7.	6st70-2x3	6	70	3478.2	3525.3	0.00	**3476.7**	**3476.7**	**3476.726**	0.04	1.40	0.00	0.00
8.	6pr76-2x3	6	76	648713.1	659658.8	0.00	**648275.7**	648507.9	**648275.700**	0.07	1.76	0.00	0.04
9.	8berlin52-2x4	8	52	**26783.2**	27060.4	0.02	**26783.2**	26795.4	**26783.226**	0.00	1.03	0.00	0.05
10.	9eil51-3x3	9	51	1916.0	1963.8	0.02	1908.0	1909.9	**1907.745**	0.43	2.94	0.01	0.11
11.	9st70-3x3	9	70	-	-	-	-	-	**3028.765**	-	-	-	-
12.	9eil76-3x3	9	76	2999.4	3057.2	0.02	**2937.4**	2938.6	**2937.384**	2.11	4.08	0.00	0.04
13.	9pr76-3x3	9	76	558349.3	567987.4	0.02	553685.6	553849.7	**553400.634**	0.89	2.64	0.05	0.08
14.	9eil101-3x3	9	101	3184.4	3274.5	0.02	**3117.6**	3120.2	**3117.562**	2.14	5.03	0.00	0.08
15.	10berlin52-2x5	10	52	**27471.4**	27805.3	0.02	**27471.4**	27473.0	**27471.379**	0.00	1.22	0.00	0.01
16.	12eil51-3x4	12	51	1720.1	1762.7	0.02	**1699.0**	1699.1	**1698.960**	1.24	3.75	0.00	0.01
17.	12st70-3x4	12	70	4148.4	4219.2	0.02	**4106.5**	4110.1	**4106.487**	1.02	2.74	0.00	0.09
18.	12eil76-3x4	12	76	2738.6	2802.0	0.02	**2650.8**	**2650.8**	**2650.777**	3.31	5.70	0.00	0.00
19.	12pr76-3x4	12	76	604837.0	621228.6	0.02	600430.9	600818.7	**600008.613**	0.80	3.54	0.07	0.14
20.	15pr76-3x5	15	76	534613.0	544174.0	0.03	525170.3	526166.2	**524335.181**	1.96	3.78	0.16	0.35
21.	16eil51-4x4	16	51	1323.8	1351.1	0.03	1302.7	1305.6	**1301.448**	1.72	3.82	0.10	0.32
22.	16st70-4x4	16	70	2963.8	3050.4	0.03	2935.4	2949.2	**2932.644**	1.06	4.02	0.09	0.56
23.	16eil76-4x4	16	76	2088.1	2163.0	0.03	2040.0	2052.2	**2035.952**	2.56	6.24	0.20	0.80
24.	16lin105-4x4	16	105	128713.1	130815.3	0.03	**125052.2**	125289.8	**125052.236**	2.93	4.61	0.00	0.19
25.	18pr76-3x6	18	76	641209.6	657524.3	0.03	639723.3	641700.1	**638164.479**	0.48	3.03	0.24	0.55
26.	20eil51-4x5	20	51	2286.4	2331.3	0.03	2288.7	2295.2	**2283.749**	0.12	2.08	0.22	0.50
27.	20st70-4x5	20	70	2976.9	3032.7	0.03	2942.8	2967.0	**2934.795**	1.43	3.34	0.27	1.10
28.	20eil76-4x5	20	76	2478.2	2520.3	0.02	2390.5	2402.2	**2385.863**	3.87	5.63	0.19	0.68
29.	25eil51-5x5	25	51	1483.7	1512.0	0.03	1487.4	1507.3	**1474.610**	0.62	2.54	0.87	2.22
30.	25eil76-5x5	25	76	2264.3	2312.1	0.03	2219.1	2245.2	**2193.083**	3.25	5.43	1.19	2.38
31.	25rat99-5x5	25	99	11754.1	11869.1	0.03	11434.9	11485.9	**11395.808**	3.14	4.15	0.34	0.79
32.	35kroB100-5x5	25	100	133662.2	137003.4	0.03	130935.1	132840.4	**129078.740**	3.55	6.14	1.44	2.91
33.	25eil101-5x5	25	101	3711.7	3780.9	0.03	3649.2	3670.5	**3603.528**	3.00	4.92	1.27	1.86
34.	28kroA100-4x7	28	100	138682.6	141334.2	0.03	136501.1	138342.8	**133101.625**	4.19	6.19	2.55	3.94
35.	30kroB100-5x6	30	100	201813.7	204967.3	0.03	200596.8	202209.8	**197934.573**	1.96	3.55	1.35	2.16
36.	36eil101-6x6	36	101	3977.6	4028.6	0.05	3929.2	3981.6	**3850.716**	3.30	4.62	2.04	3.40
37.	42rat99-6x7	42	99	9093.5	9182.7	0.07	9187.0	9393.5	**8902.148**	2.15	3.15	3.20	5.52

We can observe that in the case of all the euclidean instances of Type 2, our genetic algorithm delivered in all 30 runs the optimal solution in less than 1 millisecond.

Analyzing the results displayed in Table 13, we can observe that our proposed genetic algorithm delivered the optimal solutions in less than 1 millisecond for all the 14 euclidean instances of Type 3. Binh *et al.* [2] provided the solutions only for the first 5 instances and all the best solutions are different from the optimal ones. The gap between the best provided solution. and the optimal solution ranges between 0.49% and 1.2%. The computational time for all the instances is 0.02 minutes. Thanh *et al.* [24] provided as well the solutions only for the first 5 instances, and all the best solutions are different from the optimal ones. The gap between the best provided solution and the optimal solution ranges between 0.01% and 0.15%.

Analyzing the results displayed in Table 14, we can observe that our proposed genetic algorithm delivered the optimal solutions in less than 1 millisecond for all the 41 euclidean instances of Type 4. Binh *et al.* [2] provided the solutions only for the first 10 instances, and for 6 of these instances they achieved optimal solutions. The computational times are shorter than 0.02 minutes. The gap between the best provided solutions and the optimal ones ranges between 0 and 1.43%. Thanh *et al.* [24] provided as well the solutions only for the first 10 instances, and for 7 of these instances they achieved optimal solutions. The gap between the best provided solutions and the optimal ones ranges between 0 and 10.63%, which is huge for the relatively small instances they used in their experiments. Our genetic algorithm delivered in all 30 runs the optimal solutions.

When taking a closer look at the computational results shown in Table 15, we can observe that our proposed genetic algorithm delivered the optimal solutions in less than 1 millisecond for all the 21 small euclidean instances of Type 5. Binh *et al.* [2] achieved the optimal solution for 10 out of the 21 instances. The gap between the best provided solution and the optimal solution ranges between 0 and 2.59%. The computational times are shorter than 0.02 minutes. Thanh *et al.* [24] obtained the optimal solutions for 17 out of the 21 instances. The gap between the best provided solution and the optimal solution ranges between 0 and 0.03%. Our novel genetic algorithm provided the optimal solutions in all 30 runs, while in the case of the evolutionary algorithm developed by Binh *et al.* [2] only for the first instance the average solution is equal to the best solution in rest they are different, and in the case the multifactorial evolutionary algorithm proposed by Thanh *et al.* [24] in 10 out of 21 instances the average solutions are equal to the corresponding best solutions and in rest they are different.

TABLE 18. **Experimental results in the case of large euclidean instances of Type 6.**

Instance				NEA [2]			N-MFEA [24]		Our GA	NEA gap		N-MFEA gap	
No	Name	k	n	BF	Avg	Time	BF	Avg	OPT	BFG	AvgG	BFG	AvgG
1.	9gil262-3x3	9	262	22158.9	23059.2	0.02	20937.8	20993.1	**20935.932**	5.84	10.14	0.01	0.27
2.	9a280-3x3	9	280	30443.2	31011.6	0.02	29045.1	29105.0	**28947.465**	5.17	7.13	0.34	0.54
3.	9lin318-3x3	9	318	730038.2	740588.6	0.02	718479.4	719450.3	**716850.156**	1.84	3.31	0.23	0.36
4.	9pr439-3x3	9	439	1820176.1	1881943.0	0.03	1803288.2	1809146.2	**1800753.860**	1.08	4.51	0.14	0.47
5.	9pcb442-3x3	9	442	803179.2	821884.9	0.02	760484.4	761269.1	**760238.263**	5.65	8.11	0.03	0.14
6.	10nrw1379-2x5	10	1379	-	-	-	-	-	**1317301.426**	-	-	-	-
7.	12nrw1379-2x6	12	1379	-	-	-	-	-	**1239386.024**	-	-	-	-
8.	12nrw1379-3x4	12	1379	-	-	-	-	-	**1486648.636**	-	-	-	-
9.	18pr439-3x6	18	439	1525370.2	1553940.8	0.03	1483618.3	1488675.8	**1471788.741**	3.64	5.58	0.80	1.15
10.	20pr439-4x5	20	439	2035939.4	2075349.7	0.05	1993350.8	2000099.7	**1978001.296**	2.93	4.92	0.78	1.12
11.	25gil262-5x5	25	262	32172.6	32674.3	0.05	31116.2	31372.9	**30649.534**	4.97	6.61	1.52	2.36
12.	25a280-5x5	25	280	43408.0	44268.6	0.05	42123.1	42388.8	**41690.327**	4.12	6.18	1.04	1.68
13.	25lin318-5x5	25	318	-	-	-	-	-	**715009.585**	-	-	-	-
14.	25pcb442-5x5	25	442	786167.2	802854.1	0.05	752573.9	762391.2	**740883.313**	6.11	8.36	1.58	2.90
15.	36pcb442-6x6	36	442	899354.2	913260.7	0.08	887178.2	901697.8	**860978.199**	4.46	6.07	3.04	4.73
16.	36pr1002-6x6	36	1002	-	-	-	-	-	**5800806.387**	-	-	-	-
17.	42a280-6x7	42	280	45163.4	45660.2	0.12	45030.0	46000.9	**43896.784**	2.89	4.02	2.58	4.79
18.	42pr1002-6x7	42	1002	-	-	-	-	-	**6974749.689**	-	-	-	-
19.	49gil262-7x7	49	262	33206.0	33514.1	0.15	-	-	**32130.496**	3.35	4.31	-	-
20.	49lin318-7x7	49	318	591374.9	595249.9	0.15	609127.8	633218.4	**569746.275**	3.80	4.48	6.91	11.14
21.	49rat783-7x7	49	783	-	-	-	-	-	**231140.818**	-	-	-	-
22.	49pr1002-7x7	49	1002	-	-	-	-	-	**6894864.478**	-	-	-	-
23.	49vm1084-7x7	49	1084	-	-	-	-	-	**7466097.603**	-	-	-	-
24.	49pcb1173-7x7	49	1173	-	-	-	-	-	**1379851.273**	-	-	-	-
25.	72vm1084-8x9	72	1084	-	-	-	-	-	**7357094.536**	-	-	-	-
26.	81vm1084-9x9	81	1084	-	-	-	-	-	**8286234.525**	-	-	-	-
27.	100rat783-10x10	100	783	-	-	-	-	-	**135833.427**	-	-	-	-
28.	100prb1173-10x10	100	1173	-	-	-	-	-	**1665836.844**	-	-	-	-
29.	144rat783-12x12	144	783	-	-	-	-	-	**225297.666**	-	-	-	-
30.	144pcb1173-12x12	144	1173	-	-	-	-	-	**1185361.424**	-	-	-	-

Analyzing the computational results displayed in Table 16, we can observe that our proposed genetic algorithm delivered the optimal solutions in less than 1 millisecond for all the 40 large euclidean instances of Type 5. Binh *et al.* [2] provided the solutions only for 15 instances, and all the best solutions are different from the optimal ones. The gap between the best provided solution and the optimal solution ranges between 0.81% and 7.92%. The computational times range between 0.02 minutes and 0.05 minutes. Thanh *et al.* [24] provided solutions only for 12 instances, and only for the first instance the achieved best solution is qual to the optimal solution. The gap between the best provided solution and the optimal solution ranges between 0 and 1.3%. Our novel genetic algorithm provided the optimal solutions in all 30 runs, while in the case of the evolutionary algorithm developed by Binh *et al.* [2] and the multifactorial evolutionary algorithm proposed by Thanh *et al.* [24] for all the considered instances the average solutions are different from the corresponding best solutions.

When taking a closer look at the computational results shown in Table 17, we can observe that our proposed genetic algorithm delivered the optimal solutions in less than 1 millisecond for all the 37 small euclidean instances of Type 6. Binh *et al.* [2] provided solutions for 36 out of 37 instances and achieved the optimal solution for 7 of them. The gap between the best provided solution and the optimal solution ranges between 0 and 4.19%. The computational times are shorter than 0.07 minutes. Thanh *et al.* [24] provided solutions for 35 out of 37 instances and obtained the optimal solution for 15 of them. The gap between the best provided solution and the optimal solution ranges between 0 and 3.20%. Our novel genetic algorithm provided the optimal solutions in all 30 runs, while in the case of the evolutionary algorithm developed by Binh *et al.* [2] only for two instances the average solutions are equal with the best solutions. In the case the multifactorial evolutionary algorithm proposed by Thanh *et al.* [24] in 7 out of 37 instances the average solutions are equal to the corresponding best solutions.

Analyzing the computational results displayed in Table 18, we can observe that our proposed genetic algorithm delivered the optimal solutions in less than 1 millisecond for all the 30 large euclidean instances of Type 6. Binh *et al.* [2] provided the solutions only for 14 instances out of 30, and all the best solutions are different from the optimal ones. The gap between the best provided solution and the optimal solution ranges between 1.08% and 6.11%. The computational times range between 0.02 minutes and 0.15 minutes. Thanh *et al.* [24] provided solutions only for 13 instances out of 30, and all the best solutions are different from the optimal ones. The gap between the best provided solution and the optimal solution ranges between 0.01% and 6.91%. Our novel genetic algorithm provided the optimal solutions in all 30 runs, while in the case of the evolutionary algorithm developed by Binh *et al.* [2] and the multifactorial evolutionary algorithm proposed by Thanh *et al.* [24] in all the considered instances the average solutions are different from the corresponding best solutions.

Regarding the efficiency of the proposed algorithm, it is rather difficult to make a fair comparison with the competing algorithms, as the experiments were carried out on different computers and the algorithms were implemented in different languages. Thanh *et al.* [24] reported computational results only for 141 out of 248 euclidean benchmark instances of small and medium size and did not provide any information about computational times. Binh *et al.* [2] used in the experiments a computer with Intel Core i7 - 4790 - 3.60 GHz, 16 GB RAM, which is slightly better than the one we used, and reported computational results only for 141 out of 248 euclidean benchmark instances of small and medium size. In the case of the considered euclidean instances, the execution times reported by Binh *et al.* [2] are under 0.15 minutes. For these instances our algorithm finds the optimal solution each time in less than 1 millisecond. That is 9000 times faster.

REFERENCES

[1] H. T. T. Binh, P. D. Thanh, T. Ba Trung, and L. P. Thao, "Effective multifactorial evolutionary algorithm for solving the cluster shortest path tree problem," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Jul. 2018, pp. 819–826.

[2] H. T. T. Binh, P. D. Thanh, and T. B. Thang, "New approach to solving the clustered shortest-path tree problem based on reducing the search space of evolutionary algorithm," *Knowl.-Based Syst.*, vol. 180, pp. 12–25, Sep. 2019.

[3] O. Cosma, P. C. Pop, and I. Zelina, "A novel genetic algorithm for solving the clustered, shortest-path tree problem," *Carpathian J. Math.*, vol. 36, no. 3, pp. 403–416, 2020.

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

[5] E. W. Djikstra, "A note on two problems in connection with graphs," *Numer. Math.*, vol. 1, pp. 269–271, Dec. 1959.

[6] M. D'Emidio, L. Forlizzi, D. Frigioni, S. Leucci, and G. Proietti, "On the clustered shortest-path tree problem," in *Proc. Italian Conf. Theor. Comput. Sci.*, 2016, pp. 263–268.

[7] M. D'Emidio, L. Forlizzi, D. Frigioni, S. Leucci, and G. Proietti, "Hardness, approximability, and fixed-parameter tractability of the clustered shortest-path tree problem," *J. Combinat. Optim.*, vol. 38, no. 1, pp. 165–184, Jul. 2019.

[8] C. Feremans, M. Labbé, and G. Laporte, "Generalized network design problems," *Eur. J. Oper. Res.*, vol. 148, no. 1, pp. 1–13, Jul. 2003.

[9] S. Fidanova and P. Pop, "An improved hybrid ant-local search algorithm for the partition graph coloring problem," *J. Comput. Appl. Math.*, vol. 293, pp. 55–61, Feb. 2016.

[10] M. Fischetti, J. J. Salazar González, and P. Toth, "A branch-and-cut algorithm for the symmetric generalized traveling salesman problem," *Oper. Res.*, vol. 45, no. 3, pp. 378–394, Jun. 1997.

[11] G. Ghiani and G. Improta, "An efficient transformation of the generalized vehicle routing problem," *Eur. J. Oper. Res.*, vol. 122, no. 1, pp. 11–17, Apr. 2000.

[12] P. T. H. Hanh, P. D. Thanh, and H. T. T. Binh, "Evolutionary algorithm and multifactorial evolutionary algorithm on clustered shortest-path tree problem," *Inf. Sci.*, vol. 553, pp. 280–304, Apr. 2021.

[13] J. H. Holland, "Adaptation in natural and artificial systems: An introductory analysis with applications to biology," in *Control and Artificial Intelligence*. Cambridge, MA, USA: MIT Press, 1992.

[14] M. Mestria, L. Satoru Ochi, and S. de Lima Martins, "GRASP with path relinking for the symmetric Euclidean clustered traveling salesman problem," *Comput. Oper. Res.*, vol. 40, no. 12, pp. 3218–3229, Dec. 2013.

[15] Y.-S. Myung, C.-H. Lee, and D.-W. Tcha, "On the generalized minimum spanning tree problem," *Networks*, vol. 26, no. 4, pp. 231–241, Dec. 1995.

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

[17] P. C. Pop, O. Matei, and C. Sabo, "A hybrid diploid genetic based algorithm for solving the generalized traveling salesman problem," in *Hybrid Artificial Intelligent Systems* (Lecture Notes in Computer Science), vol. 10334. Cham, Switzerland: Springer, 2017, pp. 149–160.

[18] P. C. Pop, O. Matei, C. Sabo, and A. Petrovan, "A two-level solution approach for solving the generalized minimum spanning tree problem," *Eur. J. Oper. Res.*, vol. 265, no. 2, pp. 478–487, Mar. 2018.

[19] P. C. Pop, L. Fuksz, A. H. Marc, and C. Sabo, "A novel two-level optimization approach for clustered vehicle routing problem," *Comput. Ind. Eng.*, vol. 115, pp. 304–318, Jan. 2018.

[20] P. C. Pop, "The generalized minimum spanning tree problem: An overview of formulations, solution procedures and latest advances," *Eur. J. Oper. Res.*, vol. 283, no. 1, pp. 1–15, May 2020.

[21] C. Sabo, P. C. Pop, and A. Horvat-Marc, "On the selective vehicle routing problem," *Mathematics*, vol. 8, no. 5, p. 771, May 2020.

[22] P. D. Thanh, D. A. Dung, T. N. Tien, and H. T. T. Binh, "An effective representation scheme in multifactorial evolutionary algorithm for solving cluster shortest-path tree problem," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Jul. 2018, pp. 1–8.

[23] P. D. Thanh, H. T. T. Binh, D. D. Dac, N. B. Long, and L. M. H. Phong, "A heuristic based on randomized greedy algorithms for the clustered shortest-path tree problem," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Jun. 2019, pp. 2915–2922.

[24] P. D. Thanh, H. T. T. Binh, and T. B. Trung, "An efficient strategy for using multifactorial optimization to solve the clustered shortest path tree problem," *Int. J. Speech Technol.*, vol. 50, no. 4, pp. 1233–1258, Apr. 2020.

[25] H. Yang, Y. Deng, and R. Mayne, "A bio-inspired network design method for intelligent transportation," *Int. J. Unconventional Comput.*, vol. 14, nos. 3–4, pp. 199–215, 2019.

[26] H. Yang and Y. Deng, "A bio-inspired optimal network division method," *Phys. A, Stat. Mech. Appl.*, vol. 527, pp. 210–219, Aug. 2019.

OVIDIU COSMA received the B.S. degree in automatic control and computer science and the Ph.D. degree in automatic control from Politehnica University, Bucharest, Romania, in 1986 and 2004, respectively. From 1986 to 1992, he was an Analyst Programmer with the Electronic Computing Center, Baia Mare. In 1992, he began his activity as an Assistant Professor at North University, Baia Mare. He is currently an Associate Professor with Technical University, Cluj Napoca, and North University Center, Baia Mare. His research interests include combinatorial optimization, image processing, computer networks, and artificial intelligence. He is the author of six books, more than 50 research articles, and two inventions.

PETRICĂ C. POP received the B.S. degree in mathematics from the Babes-Bolyai University of Cluj-Napoca, Romania, the M.S. and Ph.D. degrees in operations research from the University of Twente, The Netherlands, and the Habilitation degree in informatics from the Babes-Bolyai University of Cluj-Napoca. He currently serves as a Professor with the Department of Mathematics and Computer Science, Technical University of Cluj-Napoca, and North University Center of Baia Mare, Romania. He wrote more than 140 papers from which more than 100 appeared in ISI journals, ISI proceedings, and international journals. His research interests include combinatorial optimization, mathematical modeling, artificial intelligence, and operations research. Several research stays have taken him to Italy, U.K., Japan, France, Austria, Greece, The Netherlands, Portugal, Spain, Norway, and Canada.

IOANA ZELINA received the B.S. degree in informatics and the Ph.D. degree in informatics from the Babes-Bolyai University of Cluj-Napoca, Romania, in 1992 and 2008, respectively. In 1992, she became an Assistant Professor with North University, Baia Mare, and she is currently an Associate Professor with the Technical University, Cluj-Napoca, and North University Center, Baia Mare. Her research interests include combinatorial optimization, graph theory, and distributed systems. She wrote four books and more than 30 research articles published in international journals.