Memetic Computing (2020) 12:185–217
https://doi.org/10.1007/s12293-020-00309-2

REGULAR RESEARCH PAPER

Multifactorial evolutionary algorithm for solving clustered tree problems: competition among Cayley codes

Case studies on the clustered shortest-path tree problem and the minimum inter-cluster routing cost clustered tree problem

Thanh Pham Dinh[1] · Binh Huynh Thi Thanh[2] · Trung Tran Ba[2] · Long Nguyen Binh[2]

Received: 12 July 2019 / Accepted: 24 July 2020 / Published online: 5 August 2020

Abstract
The Multifactorial Evolutionary Algorithm (MFEA) has emerged as an effective variant of the evolutionary algorithm. MFEA has been successfully applied to deal with various problems with many different types of solution encodings. Although clustered tree problems play an important role in real life, there haven't been much research on exploiting the strengths of MFEA to solve these problems. One of the challenges in applying the MFEA is to build specific evolutionary operators of the MFEA algorithm. To exploit the advantages of the Cayley Codes in improving the MFEA's performance, this paper introduces MFEA with representation scheme based on the Cayley Code to deal with the clustered tree problems. The new evolutionary operators in MFEA have two different levels. The purpose of the first level is to construct a spanning tree which connects to a vertex in each cluster, while the objective of the second one is to determine the spanning tree for each cluster. We focus on evaluating the efficiency of the new MFEA algorithm on known Cayley Codes when solving clustered tree problems. In the aspect of the execution time and the quality of the solutions found, each encoding type of the Cayley Codes is analyzed when performed on both single-task and multi-task to find the solutions of one or two different clustered tree problems respectively. In addition, we also evaluate the effect of those encodings on the convergence speed of the algorithms. Experimental results show the level of effectiveness for each encoding type and prove that the Dandelion Code outperforms the remaining encoding mechanisms when solving clustered tree problems.

Keywords Multifactorial evolutionary algorithm · Clustered tree problems · Evolutionary algorithms · Cayley code · Prüfer code · Dandelion code · Blob code

✉ Binh Huynh Thi Thanh
binhht@soict.hust.edu.vn

Thanh Pham Dinh
thanhpd05@gmail.com

Trung Tran Ba
batrung97@gmail.com

Long Nguyen Binh
binhlongmm99@gmail.com

1 Faculty of Natural Science and Technology, Taybac University, Son La, Vietnam

2 School of Information and Communication Technology, Hanoi University of Science and Technology, Hanoi, Vietnam

1 Introduction

Along with the rapid development of technology, the demand for connecting devices through the network is increasing. Therefore, optimization of the network design to save the building cost, but still to allow the devices to communicate with each other quickly and simultaneously has attracted a lot of interests from the research community. The common problem in fiber-optic network design is to ensure communication, the speed of sending, receiving information between devices and to optimize the cost of network construction.

All of the above problems can be modeled as an optimization problem on a fully connected graph. In particular, each device is considered as a vertex of the graph, each pair of devices is assigned with a weight representing the cost of the connection between them. To ensure that there is always a connection between the two devices, designing the network as a spanning tree of the graph is a suitable solution. When the number of devices is large, many research work have the ideas of aggregating similar devices into clusters, which will be used for designing a bigger network system. In reality, numerous types of clustered network systems are designed with different goals in term of optimizing communication costs. Those network systems are then modeled as clustered spanning trees to ensure inter-device connectivity and inter-cluster communication connectivity. In some recent studies on problems with clustered tree structures, D'Emidio et al. [8] defined the Clustered Shortest-Path Tree Problem (CluSPT), and Chen-Wan Lin et al. [14] introduced the Minimum Inter-cluster Routing Cost Clustered Tree Problem (InterCluMRCT). The solution to each of the 2 problems mentioned above is a spanning tree of a graph which satisfies a constrain that the sub-graph in each cluster belonging to that tree is also a spanning tree. Nevertheless, the goals of two problems are different. While the goal of the CluSPT is to find a spanning tree of a graph which minimizes the total length of all paths from a given source vertex to other vertices, the InterCluMRCT tries to minimize the total communication cost among vertices of different clusters. In this study, we aim to design an algorithm to solve simultaneously multiple network problems that can be transformed into the clustered tree problems with different objective functions. The effectiveness of the algorithm is then experimented on two examples of clustered tree problems, which are the CluSPT and the InterCluMRCT. Based on the similarities of the solution structures and constraints, we try to build a mechanism to simultaneously solve those 2 above problems specifically, with the purpose of solving various different types of clustered network design problems in general.

Recently, many studies have proposed multitasking algorithms to find solutions to multiple different combinatorial optimization problems at the same time. In particular, Multifactorial Evolutionary Algorithm (MFEA) [2] is emerging as one of the most effective variants of the Evolutionary Algorithm (EA). MFEA which has been proposed by A. Gupta et al. [11] can solve multiple independent optimization problems simultaneously using a single population of solutions in the unified search space. Since a part of the genetic material of individuals in unified search space is shared among different problems, there is always a process of exchanging information among problems inside MFEA through changing the shared genetic material. Based on these advantages, this research introduces the MFEA algorithm to simultaneously solve 2 problems which are the CluSPT problems and the InterCluMRCT problem. We propose a method of combining those two problems with different dimensions into the same search space. The main reason for doing that is due to the similar properties of the solution structures for those problems, so exchanging the components of the two solutions will assist in searching to find a better solution in the process of exploring the solution space.

There are several ways for representing solution as spanning tree such as edge-sets [20], characteristic vectors [9], etc. in which Cayley Code [19,24] has been investigated for a long time and attracted a lot of attention from the research community because of its three main advantages. Cayley Code represents each tree with n labeled vertices as a string of $n-2$ labels, such that each tree corresponds to a unique string. Firstly, Cayley Code can encode a solution into spanning tree easier than other methods. Instead of representing chromosome as a list of edges like edge-set method [20], chromosome can be encoded as a series of numbers. Secondly, it takes full advantage of existing evolutionary operators such as one-point crossover, two-point crossover, swap mutation, etc. Lastly, there are various effective algorithms for solving a lot of problems based on the Cayley Code [19,24].

With its outstanding advantages, Cayley Code is chosen to be the solution representation for the solution to the 2 above-mentioned problems in our research. We introduce a method to represent a solution of clustered tree problems as a series of integers based on Cayley code. A spanning tree corresponding to each cluster is encoded into a segment of Cayley string which is arranged next to each other to form a solution for clustered tree problem. A valid solution can be constructed by decoding all Cayley strings from encoded solution and combining them altogether. As a result, some operations on the solution will be less complex since performing the operations on a string of numbers is simpler than that of a graph. This is especially useful when applying for EA due to the fact that in each generation such evolutionary operations like crossover operator, mutation operator,... constantly happened, so the simplification of those operations reduces the computational time for the entire algorithm. This paper also investigates the effectiveness of different types of encodings in the Cayley Code to find the most appropriate representations for solving clustered problems.

Consequently, we propose MFEA to solve clustered tree problems based on the Cayley Code (called CC-MFEA). The major contributions of this work are as following:

1. Propose an effective encoding mechanism for representing many solutions of clustered tree problems into unified representation in Unified Search Space (USS) in order to apply different types of Cayley Codes: Prüfer Code, Dandelion Code, and Blob Code for encoding trees.
2. Propose an effective decoding method to obtain the solution corresponding to each specific task problem.
3. Propose evolutionary operations for CC-MFEA.
4. Analyze the effectiveness of the proposed algorithm in some different clustered tree problems.

5. Analyze the effectiveness of the Cayley Code for solving the clustered tree problems.

The rest of this paper is organized as follows. Section 2 introduces related works and Sect. 3 provides foundation knowledge of the Cayley Code. Section 4 presents the notations and definitions used for formulating problem. The proposed CC-MFEA for the clustered tree problems is elaborated in Sect. 5. Section 6 explains the setup of our experiments and reports the computed results. The paper concludes in Sect. 7 with discussions on the future extension of this research.

2 Related works

2.1 Multitasking in evolutionary algorithms

Evolutionary Algorithm (EA) [1] is one of the most well-known algorithms for finding global optimization and is a part of evolutionary computation. The mechanisms of EA are inspired by the biological evolution. Being able to solve complex problems, EA has been successfully applied in many fields such as engineering, art, economics, marketing, genetics, operations research, robotics and social sciences. Despite its various strengths, EA has shown several weaknesses, one of them is the ability to do multiple tasks at a time. Therefore, EA is not compatible with some computational model such as cloud computing. To overcome this restriction, Abhishek Gupta et al [11] proposed a new paradigm of EA, called Multifactorial Evolutionary Algorithm. This new model can perform many tasks at a time on a sole search space. R. Chandra et al [5] applied the MFEA to train neural networks. In this model, the required tasks are neural network topologies with different number of hidden neurons. The authors use the encoding and decoding rules in [11] for solution representation whose dimensionality is that of one single task (in this case, the number of hidden neurons). For evaluation of the proposed algorithm, MFEA is compared with evolutionary single-task learning (ESTL) by using the n-bit Parity Problem. The results showed that MFEA is better than ESTL on all of the cases.

In Da et al. [6], the authors used MFEA to solve a single-objective problem by transforming it into a multi-objective one. With this method, MFEA is expected to overcome the local optima and exploit better areas of the search space. To demonstrate the competence of proposed MFEA, the authors compared it with the classic single-objective evolutionary algorithm (SOEA) and the standard multi-objective evolutionary algorithm (MOEA). The symmetric Traveling Salesman Problem is selected as test problem. Order crossover, random swap mutation and 2-opt local search are used. The results show that by overcoming the local optima, the proposed algorithm can get the optimal solutions on 4 out of the 5 runs, while SOEA gets trapped at a local optimum. The proposed algorithm performs at least as well as SOEA in 20 cases and better in 11 cases. In comparison with MOEA, the proposed algorithm performs at least as well on 18 out of the 20 instances and better in 11 cases. Concerning convergence trends, the convergent speeds of the proposed algorithm and SOEA are relatively fast, and equal to each other in the first 25 generations, but in the rest generations, the proposed algorithm continues to successfully explore latent regions of the search space, while SOEA gets stuck.

A. Gupta et al [10] used the characteristics of the Evolutionary multitasking in the bi-level optimization. The authors assume that lower level optimization tasks are corresponding to the neighboring upper level population members. Therefore, the individuals in that upper level population are clustered into groups. For comparing the proposed algorithm with the basic Nested Bi-Level Evolutionary Algorithm (N-BLEA), ten-dimension variants of SMD benchmarks are selected. The representation is random-key encoding scheme. The selected crossover and mutation operator are SBX operator and polynomial mutation respectively. The experimental results show that N-BLEA is worse than the proposed algorithm with respect to the number of function evaluations of the lower level. The authors also select the composites manufacturing problem for evaluating the proposed algorithm. The results show that the convergence trend of the proposed algorithm is considerably hastened compared with that of N-BLEA.

In Gupta et al. [12], A.Gupta et al further extended the concept of multitasking into a Multi-Objective MultiFactorial Optimization (MO-MFO) paradigm, in which each constitutive task is a Multi-Objective Optimization (MOO) problem that contributes a distinct factor influencing the evolutionary search process. One of the main characteristics that makes evolutionary multitasking appealing for multi-objective problems is the implicit parallelism provided by a unified population which may allow for simultaneous convergence toward the Pareto front. Accordingly, the authors proposed the Multi-Objective Multi-Factorial Evolutionary Algorithm (MO-MFEA), and compared the performance of the proposed algorithm with the classical NSGA-II which deals with a single task at a time. To demonstrate the improvement in the convergence trends of the proposed algorithm, experiments on several synthetic benchmark functions was conducted with identical solution encoding, genetic operators, and parameter settings of the proposed MO-MFEA and the NSGA-II counterpart. The results showed that while solving a problem instance separately by NSGA-II, the IGD progressed slowly and had the tendency of getting trapped in a local Pareto front. In contract, the implicit genetic transfer provided a strong motivation to the evolutionary search process of multiple task instances being integrated, which explains the accelerated convergence characteristics achieved by the MO-MFEA.

Y.S Ong et al [17] pointed out from a practical point of view that the potential applications of effective multitask optimization are truly remarkable since by taking advantage of the underlying commonalities between different optimization tasks, multitaking provides the scope for improved performance in real-world problem solving. To prove this claim, the authors compared the performance of the proposed multitasking engine with the traditional methodology on 3 insightful computational studies of the continuous benchmark functions, combinatorial optimization problems, as well as for a more realistic optimization domain involving the path-planning of multiple unmanned aerial vehicles (multi-UAVs). In the first study, three continuous benchmark functions, namely Sphere function, Ackley function, and Rastrigin function, were combined pairwise to create the multitaking instances. Then, in order to highlight the performance of multitasking, the proposed MFEA shall be compared to an elitist single-objective evolutionary algorithm (SOEA) with similar algorithmic specifications and parameter settings. Experimental results showed that the improvement in convergence characteristics achieved during multitasking is substantial in comparison to the SOEA. As a result of intersecting global optimums between different functions which leads to predominantly positive transfer of knowledge, the MFEA can exploit the evolving population of both tasks simultaneously, efficiently overcome obstacles and converge faster, while traditional SOEA causes the population to often stuck at a local optimum. Interestingly, prior familiarity about the relationship between tasks is not a prerequisite for successful evolutionary multitasking, as the proposed MFEA can exploit latent complementarities autonomously even between apparently disparate optimization tasks, thereby successfully accelerating convergence trends. Such claim was successfully demonstrated in the second study about combinatorial optimization problems, when the authors combined a QAP(Sko100a) and a JSP(la27) to be solved together. While the single-tasking method is found to get trapped in local optimum regions of the search space, the diversified search facilitated by multitasking substantially improved performance characteristics due to constant transfer of genetic materials. Finally, by tackling 2 different multi-UAV path-planning missions at once (a pair of UAVs flying through two narrow openings in a barrier, and four UAVs flying around a centrally located geo-fence of circular planform without colliding into one another), the authors had been able to improved the obtained results which showcase the utility of multitasking in realistic domains.

2.2 Cayley code: a representation scheme for spanning tree in genetic algorithms

There are a lot of research works about representation for Trees, one of the earliest is Cayley Code when Palmer et al. [18] introduced Prüfer code. In recent years, many research works demonstrate the effect of Cayley code on Genetic Algorithm (GA), such as Julstrom et al. (Julstrom [13]) deployed the Blob Code as an individual representation in GA. The results showed that the Blob Code has high locality, and could outperform the Prüfer Code in a simple test problem. In 2005, Julstrom el al. (Julstrom [13]) showed that the Blob Code was competitive with edge-sets representation on the minimum routing cost spanning tree problem. Thompson E. B. [24] indicated that the Dandelion Code has even higher locality than the Blob Code, and offers superior performance in a GA. Futher details of the Cayley Code will be presented in Sect. 3.

2.3 Clustered tree optimization problems and several solving approaches

The network optimization problems have many applications in real life [8,14] and many of those problems have structure being clustered [8]. For that reason, a lot of researchers have shown their keen interests in such problems. One of the most interesting research in clustered problems is Clustered Steiner Tree Problem (CluSteiner). Bang Ye Wu et al. [26] considered a version of the Steiner Minimum Tree whose vertices are allocated into clusters. Firstly, the authors show that the Steiner ratios of problems can get maximum at 4 and minimum at 3. The authors also propose an approximation algorithm for solving CluSteiner. In the new algorithm, the Hamiltonian paths are added to all local tree of the clusters, then the edges belonging to the inter-cluster topology are removed. The authors demonstrate that new algorithm can be approximate in $(2+\rho)$ for CluSteiner in $O(nlog(n)+f(n))$ time, when local topologies are given, the problem CluSteiner can be $(1+\rho)$-approximated in $O(nlog(n)+f(n))$ time. Chen-Wan Lin and Bang Ye Wu [14] studied another variant of clustered tree, Minimum Routing Cost Clustered Tree Problem (CluMRCT). The new constraint in CluMRCT problem is that sub-graph in each cluster is a spanning tree. The authors proved that the CluMRCT having more than 2 clusters is NP-Hard and proposed an approximation algorithm for solving the CluMRCT problem. The new approximation algorithm finds the solutions of CluMRCT problem by creating a two-level star-like graph and based on the R-Star graph. The authors demonstrate that the routing cost of R-Star graph is less than two times the cost of the optimal solution of CluMRCT problem.

In Myung et al. [16], the authors studied a generalized version of the Minimum Spanning Tree Problem. The vertices were divided into distinct groups. The goal was to find a minimum-cost tree which contains only one vertex from each group. After proving that Generalized Minimum Spanning Tree Problem (GMSTP) was NP-hard, the authors proposed two mathematical programming formulations and compared them in terms of linear programming relaxations when applied to GMSTP. The test problems were partitioned into two sets: big and small set. The small one were selected from generalized traveling salesman instances whose number of vertices ranged from 0 to 100. The larger one were randomly created through an algorithm. The experimental results showed that the gap between the upper and the lower bounds grew when the number of groups and vertices in each group increased. The authors [8] have proven that the CluSPT is in the NP-hard class and provided an approximation algorithm for CluSPT problem (denoted as ALL). ALL performs well when the number of vertices in all clusters is small. The approximation approaches including EAs are used to solve the CluSPT when the problem has large dimensionality or at least one cluster has a great number of vertices. Recently, new variants of EA, MFEA are applied to solve the CluSPT and improve the resulting solutions.

In Binh et al. [3], the authors proposed MFEA (hereinafter E-MFEA)with new genetic operator algorithm for solving the CluSPT. The major ideas of the novel genetic operators is that first constructing spanning tree for smallest sub-graph then the spanning tree for larger sub-graph are construed based on the spanning tree for smaller sub-graph. This paper also proposed the new individual encoding in unified search space. The number of clusters of individual is equal to maximum number of clusters of all tasks and the number of vertices of cluster i is maximum number of vertex of cluster i of all tasks. The experimental results point out the effect of novel MFEA. However the novel MFEA has disadvantage, for instance, to construct spanning tree for the sub-graph the edges set of the sub-graph must contain edges set of smaller sub-graph; the individual encoding can make no good solutions for each task in the initial population.

In Thanh et al. [23], the authors take the advantage of Cayley code to encode the solution of CluSPT and proposed genetic operators based on Cayley code. The ideas of genetic operators are taken ideas of genetic operator for binary and permutation representation. In this paper, effective MFEA (hereinafter C-MFEA) for CluSPT is proposed. The decoding method processes segments of genes by removing the genes which have invalid values. To present the spanning tree for a graph by Cayley code, the vertex of the graph must has larger than 2 so novel MFEA can only apply to solve CluSPT instances in which the vertex of its clusters must has larger than 2. Another disadvantage of Cayley code is only applying to complete graph therefore novel MFEA is suitable for complete graph.

The authors [4] proposed a new algorithm based on the EA and Dijkstra Algorithm. The proposed algorithm decomposes the CluSPT problem into two sub-problems, the solution of the first sub-problem is found by an EA while that of the second sub-problem is found by Dijkstra Algorithm. The goal of the first sub-problem is to determine a spanning tree which connects among the clusters, while that of the second sub-problem is to determine the best spanning tree for each cluster. In some real-life applications, the communication cost between devices of different clusters may need to be considered more than between those of the same cluster. In Wu and Lin [25], Chen-Wan Lin et al investigated the problem of finding a clustered spanning tree with minimum inter-cluster cost (InterCluMRCT) and denoted the variant of InterCluMRCT with the number of clusters is k as k-InterCluMRCT. After proving that the InterCluMRCT problem with 2 clusters is solvable in polynomial time, the authors have shown that k-InterCluMRCT is *NP-hard* for any fixed $k > 2$ and presented a 2-approximation algorithm for 3-InterCluMRCT.

Although there have been many studies on the use of Cayley encoding mechanisms, evaluating the effectiveness of the MFEA algorithm when applying the Cayley Code to solve the clustered tree problems is still a novel topic and attracts a lot of attention in the scientific community. As a result, this paper focuses on analyzing the efficiency of the MFEA algorithm when using the Cayley Code to solve the clustered tree problems.

3 Preliminaries of the Cayley code

Cayley Code is one of the most classical representations for spanning trees. It offers a fast and simple coding of spanning trees whose decoding algorithm is originated from a constructive proof of the Cayley's formula. This formula identifies the number of possible spanning trees in a complete graph on n nodes is n^{n-2}. From this formula, the Cayley Code has been proposed as a tree coding scheme that represents each tree on n labelled vertices (in which the underlying graph to form the tree is a complete graph) as a string of $(n-2)$ vertex labels. The Cayley Code indicates a bijective mapping between a tree and a corresponding string which ensures that each tree corresponds to a unique string, and each string corresponds to a unique tree.

In the following subsections, this paper introduces 3 typical coding types in the Cayley Code families and demonstrates their corresponding encoding and decoding mechanisms.

3.1 Prüfer code

Prüfer Code is one of the earliest example of the Cayley Code as it was firstly used in 1918 when Prufer described a mapping between a spanning tree with n nodes and a string of $(n-2)$ node labels. Since then, this code had become well-known and been widely considered as a convenient way for representing trees, especially in the application of the genetic algorithms.

Given a tree T on n nodes. The algorithm to convert a tree into a Prüfer string is described as follows:

1. Let i be the leaf with the smallest label (node of degree 1) in T and j be the neighbour node of i. Set the label of j to be the rightmost digit in the Prüfer string.
2. Remove node i and the edge (i, j) from T.
3. Repeat steps (1) and (2) until there are only 2 remaining nodes in T. The final Prüfer string is read from left to right.

It is also possible to convert from a Prüfer string to a unique tree via the following algorithm. Let $a_1 a_2 ... a_{n-2}$ be a Prüfer string. In the corresponding spanning tree, each node's degree is one more than the number of times the node's label appears. The edges of that tree is identified as follows:

1. Scan the Prüfer number to identify each node's degree. Initialize a variable t to 1.
2. Find a node i of degree 1 with the smallest label. (i, a_t) is a spanning tree edge.
3. Decrease the degree of node i and a_t by 1; increase t by 1.
4. Repeat steps (2) and (3) until all nodes have degree 0, except for two nodes with degree 1 which create the spanning tree's last edge.

Several researchers have pointed out the advantages of the Prüfer Code such as: quick and simple decoding, easy to generate a random Prüfer string and supportive in the application of conventional evolutionary operators like k-point crossover and position-by-position mutation. However, Prüfer Code actually does not support effective evolutionary search since this coding mechanism lacks the essential properties of locality *(small changes in genotypes should result in small changes in the phenotypes)* and heritability *(solutions generated by evolutionary operators should combine characteristics of their parents)*.

3.2 Blob code

In 2001, another type of Cayley Code called Blob Code was identified as a GA representation for spanning trees. The Blob mapping appears to be adapted well for evolutionary search when a GA using the Blob Code tends to considerably outperform one using the Prüfer code for a simple test problem. This can be explained by the higher locality and heritability under positional crossover and mutation that the Blob Code possesses.

In this research, we reuse the encoding and decoding algorithms for Blob Code mentioned in Julstrom [13]. These two algorithms which has the time complexity of $O(n)$, can run much faster compared to the traditional ones for the Blob Code, which may require quadratic time in the worst case. For the notation convenience, we define vertex 1 as the root vertex; the vertex i is a descendant of vertex j if there is a directed path from i to j, and successor of vertex i ($succ(i)$) is the unique neighbour of i on the path from i to 1 on the tree. Two algorithms below help to convert a tree T into a corresponding Blob string $a_2 a_3 \dots a_{n-1}$ and vice versa.

Encoding algorithm for the Blob Code:

1. Define $succ(i)$ for each vertex $i \in [2, n]$ and add the directed edge $(i \rightarrow succ(i))$ into a temporary tree T'.
2. Colour each vertex $v \in [1, n]$ black if none of the descendants of v in the directed tree T' has a label exceeding v; and colour white for the remaining vertices.
3. Label the black vertices as $x_1 < x_2 < \dots < x_t$, where $t \in [2, n]$ is the total number of black vertices. Noting that $x_1 = 1$ and $x_t = n$ since vertices 1 and n must be black.
4. To construct the Blob string, set $a_i = succ(i)$ for every white vertex $i \in [2, n-1]$, and set $a_{x_i} = succ(x_{i+1})$ for each $i \in [2, t-1]$.

Decoding algorithm for the Blob Code:

1. Define a digraph G with the vertex set as $[1, n]$ and the directed edge set as $(i \rightarrow a_i) : i \in [2, n-1]$.
2. Colour each vertex $v \in [1, n]$ black if none of the descendants of v in the directed tree Gs has a label exceeding v; and colour white for the remaining vertices.
3. Label the black vertices as $x_1 < x_2 < \dots < x_t$, where $t \in [2, n]$ is the total number of black vertices. Noting that $x_1 = 1$ and $x_t = n$ since vertices 1 and n must be black.
4. To construct the tree T, create the edge (i, a_i) for each white vertex $i \in [2, n-1]$, and create the edge $(x_i, a_{x_{i-1}})$ for each $i \in [3, t]$ with the final edge is $(x_2, 1)$.

However, EAs using the representation of Blob Code still have not performed as well as EAs using other representations (like edge-sets, etc) on most of the problems which have the search spaces of spanning trees.

3.3 Dandelion code

Recently, Dandelion Code has been proposed to be a strong alternative as it offers efficient performance for representing tree in a GA. Researchers show that this coding scheme exhibits even higher locality than the Blob Code and could potentially outperform other current Cayley code representations.

Given a Dandelion string $a_2a_3...a_{n-1}$, the algorithm to decode the Dandelion Code into a tree T works as follows:

1. Define a function $f : [2, n-1] \rightarrow [1, n]$ such that $f(i) = a_i$ for each $i \in [2, n-1]$
2. Determine the cycles associated with the function f so that in each cycle, the largest element is rightmost. Arrange the cycles from left to right in the decreasing order of the largest element of each cycle.
3. Create a list L containing all the elements of every cycle with their corresponding orders after arranging in step 2.
4. To construct the tree T from a set of n isolated vertices, build a path from 1 to n such that the intermediate nodes on this path are the nodes in the list L in left-to-right order. Then, create the edge (i, a_i) for every $i \in [2, n-1]$ not occurring in the list L.

To encode a tree T into its corresponding Dandelion string, the encoding algorithm works simply by reversing the above procedure:

1. Find a unique path from 1 to n in T, and let L be the ordered list of intermediate nodes
2. Split this list L into cycles in which each cycle ends when an element that is larger than all elements to its left still can be found.
3. The Dandelion Code corresponding to the tree T is the unique string $a_2a_3...a_{n-1}$ which satisfies: a) for those nodes $i \in [2, n-1]$ appear in the list L, insert the cycle information of each node i with respect to function $f(i) = a_i$; b) for each $i \in [2, n-1]$ that does not occur in the list L, the unique neighbour of i on the path from vertex i to vertex n in the tree T is a_i.

An example of applying 3 above-mentioned types of Cayley Code for representing a spanning tree is demonstrated in Fig. 1. On the left of this figure is a spanning tree on 8 nodes with its corresponding Prüfer string, Blob string and Dandelion string shown respectively on the right.

4 Problem formulation

In this paper, a graph is a simple, connected and undirected graph. For a graph $G = (V, E, w)$, V and E are the vertex

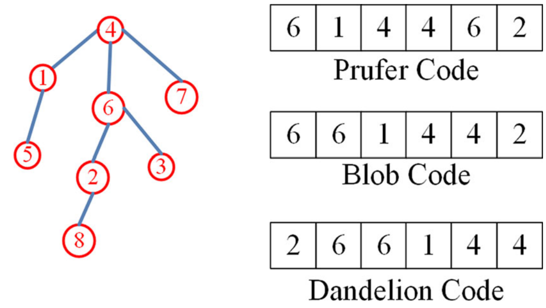


Fig. 1 An example of applying 3 types of Cayley Code for representing a spanning tree

and the edge sets, respectively, and w is the non-negative edge length function. An edge between 2 vertices u and v is denoted by (u, v), and its weight is denoted by $w(u, v)$.

For a vertex subset U, the sub-graph of G induced by U is denoted by $G[U]$. For a vertex set V, a collection $C = \{C_i | 1 \leq i \leq h\}$ of subsets of V is a partition of V if the subsets are mutually disjoint and their union is exactly V. A path of G is simple if no vertex appears more than once on the path. This paper only considers simple paths.

For a given spanning tree T of $G = (V, E, w)$ and $u, v \in V$, let $d_T(u, v)$ denote length of the shortest path between u and v on T.

Local root of a cluster is a vertex which connects to a vertex in each other cluster and denoted as L_r.

The CluSPT [8] is defined as following

Input: - A weighted undirected graph $G = (V, E, w)$.
- Vertex set V is partitioned into h clusters $C_1, C_2, \ldots, C_h$.
- A source vertex $s \in V$.

Output: - A spanning tree T of G.
- Sub-graph $T[C_i] (i = 1, \ldots, h)$ is a connected graph.

Objective: $\sum_{v \in V} d_T(s, v) \rightarrow \min$

The InterCluMRCT [14] is defined as following

Input: - A weighted undirected graph $G = (V, E, w)$.
- Vertex set V is partitioned into h clusters $C_1, C_2, \ldots, C_h$.

Output: - A spanning tree T of G.
- Sub-graph $T[C_i] (i = 1, \ldots, h)$ is a connected graph.

Objective: $\sum_{i=1}^{k} \sum_{j \neq i} d_T(C_i, C_j) \rightarrow \min$
where $d_T(C_i, C_j) = \sum_{u \in C_i} \sum_{v \in C_j} d_T(u, v)$.

Figure 2 illustrates the cases of valid and invalid solutions of the CluSPT and the InterCluMRCT. Figure 2(a) shows the input graph G with 6 clusters, 18 vertices and vertex 1 as source vertex. Figure 2(b) presents a valid solution of

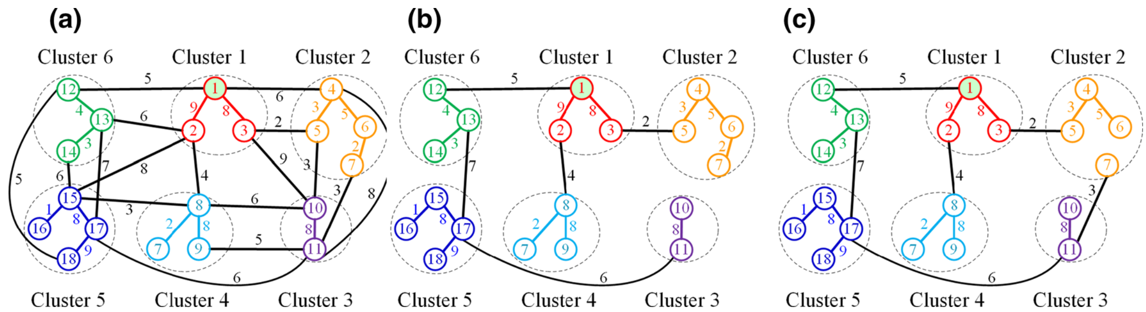


Fig. 2 An example of valid and invalid solutions for the CluSPT

the CluSPT and InterCluMRCT. In Fig. 2(c), the vertex 6 and vertex 7 in cluster 2 are not connected, so this solution violates the second condition of the output of the CluSPT and the InterCluMRCT problems.

5 Proposed algorithm

This section presents a novel approach by using the MFEA to deal with the clustered tree problems. The algorithm includes: a new individual encoding, and new evolutionary operators, i.e. new population initialization, new crossover operator and new mutation operator.

5.1 MFEA scheme to solve clustered tree problems

Multifactorial Evolutionary Algorithm is used to find the solutions of k clustered tree problems $G_i = (V_i, E_i, w_i, C_i)$, $i = 1, .., k$ where V_i, E_i, w_i and C_i are set of vertices, set of edges, weight matrix and set of clusters, respectively.

V_i is partitioned into m_i $(i = 1, \ldots, k)$ clusters $C_i = \{C_i^1, C_i^2, \ldots, C_i^{m_i}\}$.

For each clustered tree problem G_i, we decompose the problem into two subproblems: The first subproblem aims to determine the spanning tree for each cluster (also called local tree), while the second one intents to construct a spanning tree connecting all of the clusters (also called global tree). Global Graph is denoted by a graph $G'_i = (V'_i, E'_i)$ in which a cluster C_i is represented as a vertex in V'_i. An edge connects two vertices of G'_i if there exists at least one edge connecting the corresponding clusters in G_i.

To use CC-MFEA for solving many clustered tree problems simultaneously, a new scheme of individuals encoding based on Cayley code and a two-level approach are proposed. Cayley codes are used for representation of local trees and the global tree in which the position of the Cayley codes for local tree is placed before that of the global tree in an individual. We provide a mechanism for decoding the individual into the solution for each clustered tree problem as well as efficient evolutionary operators.

The pseudocode of our approach is presented in Algorithm 1. It should be noted that to compute the fitness of an individual ind_j for a given task, both local trees and global tree must be constructed for the corresponding solution of the clustered tree problem s_j (lines 5 and 27 in the Algorithm 1). Then, the algorithm computes the fitness of solution of the clustered tree problem s_j. Finally, the fitness of individual ind_j is computed through the cost of the clustered tree problem s_j (lines 6 and 28 in the Algorithm 1).

5.2 Individual encoding in unified search space

Individuals in the unified search space are encoded through two stages: the first stage will rearrange clusters of input graphs to reduce resource consumption, while the second stage encodes the solutions of all tasks into a unified representation.

5.2.1 Encoding solution of input graphs

In order to combine many clustered tree problems into unified search space, we define an individual IND which is a graph $G = (V, E, C)$ with properties as follows:

- The number of clusters of IND is equal to the maximum number of clusters in all graph G_i, denoted by N.
- The number of vertices in tth cluster of an IND is equal to the maximum number of vertices in tth cluster of all tasks, denoted by n_t.

Encoding procedures transform an individual into the string of genes by using Cayley encoding algorithms for representing all of the local trees and the global tree. Moreover, local trees and global tree are combined via the set of vertices which is the root of the local tree. The components of an individual are arranged as follows:

- Spanning tree for tth cluster is presented as tth segment, denoted by s_t (if a task constitutes of the number of clusters which is smaller than t, that task is ignored). Values

Algorithm 1: New approach to solve the clustered tree problem

begin
/* An individual as an array of vertices and clusters */
$P_0 \leftarrow$ Randomly generate N individuals ▷ Refer to Algorithm 2;
foreach *individual* $ind_j \in P_0$ **do**
Randomly assign skill factor τ_j;
Construct local and global trees of solution of the clustered tree problem s_j corresponding individual ind_j for task τ_j only;
Evaluate the fitness of ind_j based on the cost of solution s_j;
$t \leftarrow 0$;
while *stopping conditions are not satisfied* **do**
$O_t \leftarrow \emptyset$;
while *number of generated offspring* $< N$ **do**
Randomly select two individuals p_a and p_b from P_t;
if $\tau_a = \tau_b$ **then**
Perform crossover on Parents p_a and p_b to give two offspring individuals o_a and o_b;
Assign skill factor τ_a to offspring o_a and o_b ;
else
$rand \leftarrow$ Generate a random number between 0 and 1;
if $rand < rmp$ **then**
Perform crossover on Parents p_a and p_b to give two offspring individuals o_a and o_b;
Each offspring is randomly assigned skill factor τ_a or τ_b;
else
Parent p_a is mutated to give an offspring o_a ▷ Refer to Algorithm 3;
Assign skill factor τ_a to offspring o_a;
Parent p_b is mutated to give an offspring o_b ▷ Refer to Algorithm 3;
Assign skill factor τ_b to offspring o_b;
$O_t \leftarrow O_t \cup \{o_a, o_b\}$;
/* Evaluate offspring individuals for their assigned skill factors only */
foreach *individual* $o_j \in O_t$ **do**
Construct local and global trees of solution of the clustered tree problem s'_j corresponding individual o_j for task τ_j only ▷ Refer to Algorithm 5;
Evaluate the fitness of o_j based on the cost of solution s'_j;
$R_t \leftarrow O_t \cup P_t$;
$P_{t+1} \leftarrow$ Select N fittest members from R_t;
$t \leftarrow t + 1$;

of genes in each segment are in the comprehensive range of 1 to n_t.
– Spanning tree for Global Graph is represented as a segment, called G-Seg.
– Genes in the last segment (called M-Seg) is used for representing the set of local roots. The ith gene of this segment represents the vertex in the ith cluster.
– Spanning tree are represented by Cayley Code [19].

A chromosome is divided into $N + 2$ segments. For the convenience of notation, the $N+2$ segments will be indexed from 1 to $N+2$. Each of the first N segments is a Cayley code corresponding to a unique local tree. The segment $N+1$ is the Cayley code corresponding to the global tree. The segment $N + 2$ is the M-seg contains all of the root vertices of the local trees.

An example of individual encoding in new MFEA is depicted in Fig. 3. Assume that the vertices of the tasks are partitioned into clusters as Fig. 3a. An individual in the USS is illustrated in Fig. 3b. The representation of individual via Prüfer Code is depicted in Fig. 3c. This chromosome has 6 segments in which the numbers of vertices in cluster 1, 2 3 and 4 are 4, 5, 5 and 4 respectively; the G-Seg has 2 genes; M-Seg has 4 genes which are 1, 1, 3 and 2.

To find labels of vertices in all clusters for a task, the proposed CC-MFEA also builds a mapping function (called M-function) which maps vertices in every cluster of an individual in unified search space to vertices in relative cluster in that task. Take Fig. 3a, b as an example, the M-function maps vertices in cluster 2 of the individual in USS to vertices in cluster 2 of task 1 is 1 matches with 5; 2 matches with 6; 3 matches 7; 4 matches 8. Vertex 5 in cluster 2 in Fig. 3b is not mapped to task 2 because the number of vertices in cluster 2 in task 2 is 4 whereas cluster 2 in Fig. 3b has 5 vertices. The mapping functions are used in proposed decoding method.

5.2.2 Rearrange clusters on input graphs

When solutions of all tasks are encoded as an individual in USS, if the order of clusters of input graphs are kept then it may become resource consuming. For example, in the case shown in Fig. 4, Fig. 4a show as vertices in clusters of two input graphs. If the order of clusters on input graphs are kept, with the solution in USS, the number of vertices in cluster 1 is 6 while the number of vertices needed to encode cluster 2 is 5 (detail in Fig. 4b). However, if clusters of input graphs are sorted in the order of increasing of the number of vertices, then the solution in USS only needs 3 vertices for encoding cluster 1 and 6 vertices for encoding cluster 2 (detail in Fig. 4c). Clearly, this procedure reduces resources for encoding solutions.

Therefore, the clusters in the input graph are sorted in ascending (descending) order of the number of vertices in a cluster, then applied the encoding procedure in Sect. 5.2.1. For the sake of simplicity, this article assumes that the number of vertices in clusters of an input graph is sorted in increasing order.

5.3 Individual initialization method

To generate valid individuals for a problem, different representations require different methods correspondingly. We

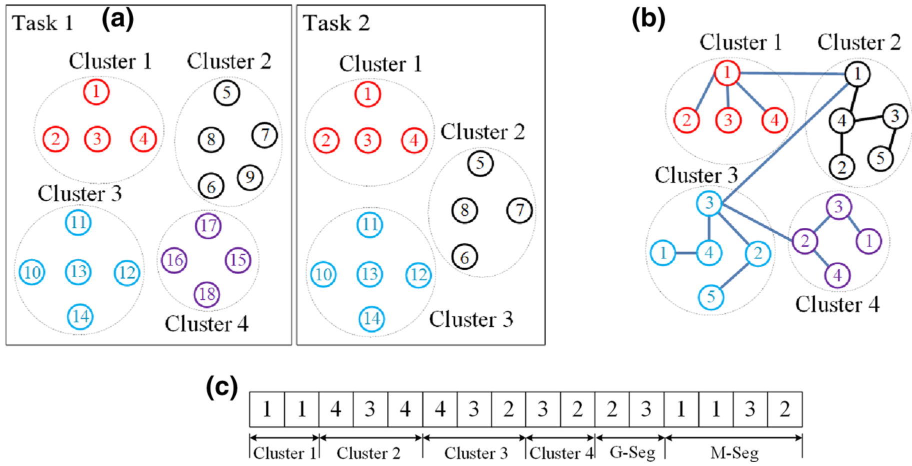


Fig. 3 The representation of individual in unified search space for MFEA with two tasks

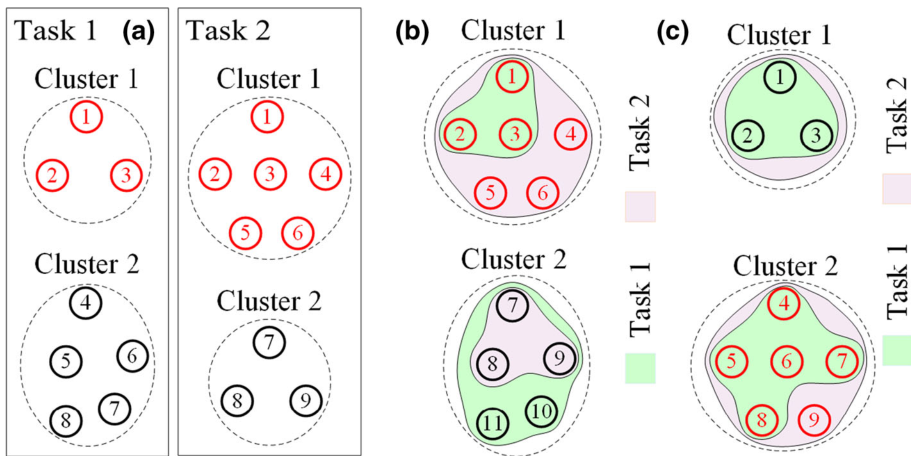


Fig. 4 Illustration of a disadvantage of the encoding solution

introduce a method to generate a valid individual, in which each segment corresponding to a Cayley string is generated randomly. In order to generate the valid solution for clustered tree problem, spanning tree for each cluster and G-Graph are constructed based on random Cayley Code. The detail of individual initializing method (IIM) is presented in Algorithm 2.

Algorithm 2: Individual Initialization

Input: $G_i = (V_i, E_i, C_i), i = 1, \ldots, k$
Output: An individual Ind

begin
- $N \leftarrow$ maximum number of cluster of all tasks;
- **for** $i \leftarrow 1$ **to** N **do**
  - $n_i \leftarrow$ maximum number of vertex in ith cluster of all tasks ;
  - $S_i \leftarrow$ Generate_Random_Cayley_Code(n_i);
  - Insert S_i to the end of Ind;
- $S_{N+1} \leftarrow$ Generate_Random_Cayley_Code(N);
- Insert S_{N+1} to the end of Ind;
- $S_{N+2} \leftarrow$ Select_Random_Vertices_In_Clusters();
- Insert S_{n+2} to the end of Ind;
- **return** Ind;

In Algorithm 2, Generate_Random_Cayley_Code(l) method generates randomly a Cayley code whose length is equal to $l-2$. Select_Random_Vertices_In_Clusters() method selects a random vertex in each cluster for creating edges which connect among clusters.

5.4 Crossover operator

An advantage of Cayley Code is that it can use many existing genetic operators such as one-point crossover, PMX crossover [1], etc. In the proposed algorithm, the One-point crossover operator [1] is used. Observe that, one point in chromosome is selected then 2 gene segments after that point are swapped for generating two offspring.

Figure 5 illustrates the crossover operator in CC-MFEA in which red vertical line presents crossover point. In our encoding method, the Cayley strings of each local tree and the global one are placed adjacent to each other, and the last segment stores the information of the vertices that will be used to construct the complete solution from the local and global tree. The one-point crossover can keep characteristics of the pair of parents, and the affectivity of that crossover

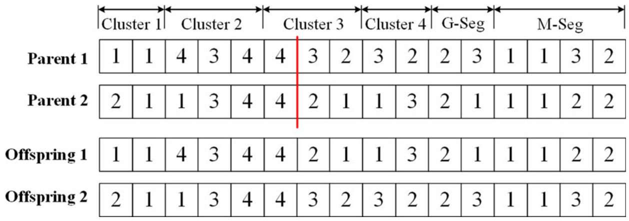


Fig. 5 An example for the crossover operator in CC-MFEA

is different depending on the position of the selected point. If the crossover point is in the local tree segment, that segment will generate a different local tree in comparison to the pair of parents. If the point crosses in the G-seg segment then the global tree is constructed differently from the pair of parents. Finally, if that point lies in the last segment, it will produce a new set of the root of the local tree. Exchanging two good partial genes of parents through that crossover point more often could result in generating high quality solution. In the crossover operation, the probability of a crossover point is located in a cluster/G-seg/M-seg is actually proportional to the corresponding length of gene. Search space of each segment is corresponding to its length. Segment with larger length has a larger search space, thus requires more generation iterations to obtain optimal structures. In contrast, the segment with shorter length requires less generation iterations to explore good structures. By making the probability of a crossover point located in a segment proportional to its corresponding length, the larger segment can have greater opportunities to exchange good partial genes in an effort to acquire more optimal structures. As a result, crossover operation reduces the number of generation loops needed in order to obtain high quality solutions.

Because of the properties of the encoding method, the One-point crossover always produces valid offsprings, and increases the diversity of individual.

5.5 Mutation operator

The Swap-Change Mutation operator (SCM) performs two types of variation: The first type of variation generates new Cayley string by swapping two genes in the selected segment, and the effect of the change to the spanning tree depends on the level of locality of which type of Cayley code has been used. If high locality code is used for decoding Cayley codes into the tree, the swapping of two genes in that Cayley string generates the new offspring which is less different from the parent. Otherwise, the low locality decoding methods change the spanning tree significantly different from parents. The second type of variation is executed on M-Segment to change the root of the local tree with the aim of increasing the diversity of individual. The SCM is described in details in Algorithm 3.

Algorithm 3: Proposed Mutation Operator

```
Input: An individual I
Output: A new solution I'
begin
    N ← Number of segments;
    /* The first type of variation                */
    Select a random segement S_r, r ∈ [1, N − 2];
    Swap two random genes in segment S_r;
    /* The second type of variation               */
    v ← A random vertex in segment S_r;
    Replace gene rth in M-Segment by v;
    return;
```

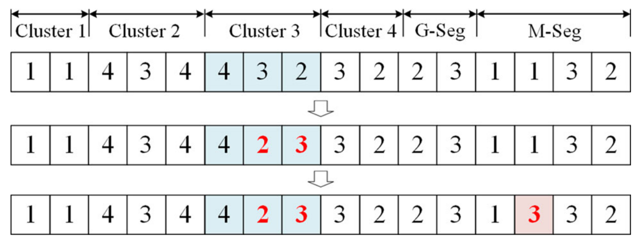


Fig. 6 An example for the new mutation operator in MFEA for solving the CluSPT problem

Figure 6 provides an example of the SCM. In this example, the 3rd cluster is randomly selected. Two genes in third segment which have the values of 2 and 3 respectively (the values are in red and bold), are swapped. The value on 2nd gene in M-Seg, which is equal to 1, is replaced by the value of 3 (the values are in red and bold).

5.6 Decoding method

In this subsection, we present association between an individual in USS and an individual in task T_j. The main reason for selecting Cayley Code to represent a chromosome is that we can use simple methods for decoding between tasks.

5.6.1 Decoding method for spanning tree problem

Let D_j be the dimensionality of jth task.

- Step 1: Remove genes whose values are greater than D_j and store them in the corresponding order into a queue Q.
- Step 2: If the number of remaining genes R is greater than $(D_j - 2)$, the solution will select the first $D_j - 2$ genes from the remaining. Otherwise, while the number of remaining genes R is less than $(D_j - 2)$, repeat following procedure: Let a be the first element in Q; b is the result of a *modulo* D_j; Append b to R; Delete a from front of Q.

Algorithm 4: S-decoding method

Input: A Cayley code segment M_i of an individual in unified search space I
Output: A Cayley code segment M'_i of an individual in jth task

begin
  $n'_i \leftarrow$ Number of vertices in ith cluster on jth task;
  $R \leftarrow$ genes in M_i whose values are less or equal than n'_i;
  Queue $Q \leftarrow$ genes in M_i whose values are greater than n'_i;
  if *length of* $R > n'_i - 2$ **then**
    $M'_i \leftarrow$ the first $n'_i - 2$ from R ;
  else
    $M'_i \leftarrow R$;
    while *Length of* $M'_i \leq (n'_i - 2)$ **do**
      $a \leftarrow$ first element in Q;
      $b \leftarrow a\ modula\ n'_i$;
      Append b to M'_i;
      Delete a from front of Q;
  return M'_i

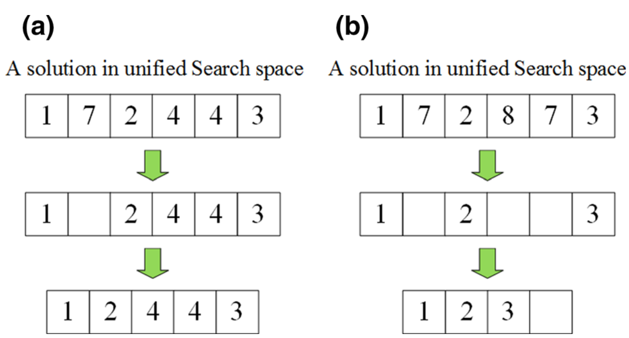


Fig. 7 The decoding method for representing spanning tree via Cayley code

Figure 7 presents the use of S-Decoding method for decoding a representing spanning tree into Cayley code of two tasks. The dimensionality of these two tasks are 6 and 8 respectively, and an individual in the unified search space is transformed to a task whose dimensionality is 6.

Figure 7a depicts a case in which after step 1, the number of remaining genes is greater than the dimensionality of the associated task. In step 1, as the value of second gene is 7 which is greater than 6, this gene is removed then stored in $Q = \{7\}$. In step 2, the number of the remaining genes is 5 which is greater than $6 - 2 = 4$, so first 4 genes are chosen to create an individual for the associated task.

Figure 7b illustrates another case of a decoding method in which the number of remaining genes is less than the dimensionality of the associated task. In step 1, genes 7, 8, 7 are removed from a segment then stored to a queue $Q = \{7, 8, 7\}$. In step 2, the number of the remaining genes is 3 which is less than $6 - 2 = 4$ is stored in $R = \{1, 2, 3\}$, one missing gene are created by getting the remainder of the first element the queue Q is 7 divided by the dimensionality of particular task is 6, and the result of 7 $modulo$ 6 is 1, then append 1 to the end of R.

5.6.2 Decoding method for Clustered Tree Problems

New decoding method for the Clustered Tree Problems (NDM) has two stages. The first stage finds spanning tree for clusters and G-Graph by using S-Decoding method. The second stage performs on last segment for finding edges which connect among clusters.

Detail of new decoding method is described in Algorithm 5.

Algorithm 5: Proposed Decoding Method

Input: An individual in unified search space I
Output: A solution T(V, E) in jth task

1 **begin**
2   An individual of jth task is denoted I';
3   $nc' \leftarrow$ Number of cluster on jth task;
4   **for** $i \leftarrow 1$ **to** nc' **do**
5     $s_i \leftarrow$ Cayley code segment ith of I;
6     $S'_i \leftarrow$ S-Decoding(s_i)   ▷ Refer to Algorithm 4;
7   Construct a tree T_i rooted at ith gene on M-Seg from S'_i ;
8     $T_i \leftarrow$ M-functions(T_i);
9   $G - seg \leftarrow$ Cayley code segment of global tree ;
10   $S'_m \leftarrow$ S-Decoding($G - seg$)   ▷ Refer to Algorithm 4;
11   Set of local roots Lr of $I' \leftarrow nc'$ first genes in M-Seg on I;
12   Construct a tree T_m from S'_m;
13   $T_m \leftarrow$ M-function(T_m);
14   V $\leftarrow \cup_{i=1}^{nc'} V(T_i)$;
15   E $\leftarrow \cup_{i=1}^{nc'} E(T_i) \cup E(T_m)$;
16   **return** T;

In Algorithm 5, the S-Decoding method is a proposed method in Sect. 5.6.1; Line 8 and 13 use the mapping functions to relabel for vertices in clusters.

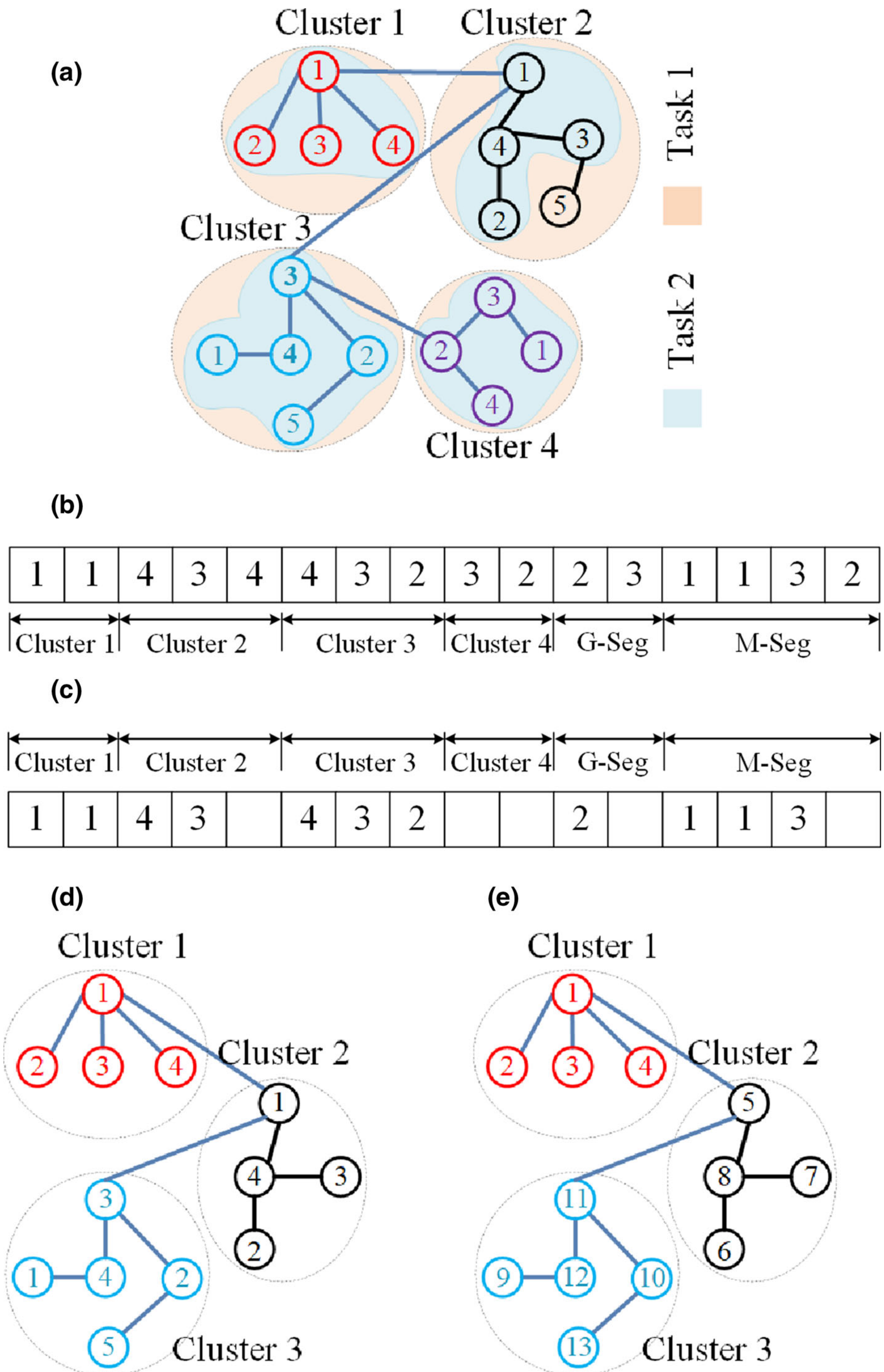
Figure 8 illustrates the proposed decoding method. Figure 8a presents an individual in USS. Representation of individual via Prüfer code is depicted in Fig. 8b. Figure 8c illustrates chromosome via Prüfer code after performing from line 1 to line 12 of Algorithm 5. Figure 8d represents a solution which is constructed from chromosome in Fig. 8c. Figure 8e illustrates the complete solution which receives from Fig. 8d after performing the M-functions in Algorithm 5.

6 Computational results

6.1 Problem instances

To assess the performance of the proposed algorithm, CluSPT instances are selected from Binh et al. [4] and InterCluMRCT instances are selected from Clustered Traveling Salesman Problem (CluTSP) instances [15]. The main reason for selecting those instances is that the instances had been generated

Fig. 8 An illustrative example of propose decoding method for CluSPT



with various algorithms to be suitable for clustered problems [15].

The experiments is performed on all small instances of Type 1, Type 5 and Type 6. The instances are selected randomly for implementing CC-MFEA. All tested instances are available via Thanh [22].

The Prüfer Code [18], Blob Code (Julstrom [13]) and Dandelion Code [19] are the well-known example of Cayley codes, so those Codes are chosen to implement experiments.

6.2 Experimental setup

We focus on the following criteria to assess the quality of the output of the algorithms.

Criteria	
Average (Avg)	Average function value over all runs
Best-found (BF)	Best function value achieved over all runs

Each problem instance was evaluated 30 times on Intel Core i7 - 4790, 16GB RAM computer using windows 8 64-bit. The source codes were written in the Java language.

The simulation parameters include population size = 100, number of evaluations = 50000, probability of random mating = 0.5, mutation rate = 0.05 and number of tasks = 2.

6.3 Experimental results

In this section, we analyze the results obtained by the proposed algorithm using different types of Cayley codes. The performance of each Cayley Code is evaluated in both single-task and multi-task environment. In addition, the effectiveness of combining 2 types of variation in the proposed SCM is also examined. All of the obtained results are then carefully investigated under some statistical analysis methods such as Wilcoxon Signed-rank Test to show the effectiveness of the proposed algorithm.

In tables of results in this section, the italic, red cells on the Prüfer Code column mean that on those instances, CC-MFEA using Prüfer Code outperforms using Dandelion Code. Similar for other encoding techniques, cells in italic and red color on the Dandenion Code and Blob Code column mean that CC-MFEA using Dandelion Code exceeds the CC-MFEA using the Blob Code and CC-MFEA using Blob Code surpasses CC-MFEA using Prüfer Code, respectively. The values in bold on encoding techniques column means that CC-MFEA using the encoding techniques outperforms using two other encoding techniques.

6.3.1 The efficacy of the proposed variation of mutation operator

To evaluate the effectiveness of each variation to the proposed mutation, Dandelion Code is chosen to conduct the following experiments: In the first experiment, we perform only the first type of variation which is swapping two genes in the selected segments, while the second experiment only uses the second type of variation that changes an element in the M-seg segment. The obtained results from the two experiments above are then compared to those of the proposed SCM which is a combination of both variation types.

Table 1 presents the results obtained by performing the swap variation, change variation and the proposed SCM. The experimental results show that the proposed SCM is better than swap variation and change variation by 8.7% and 8.6% in average, respectively.

To demonstrate the effectiveness of the proposed SCM operator, a Wilcoxon signed-rank test is conducted to check whether there was an improvement in the performance of the CC-MFEA between applying the swap variation or change variation separately and combining them intro the proposed SCM. In particular, 2 pairs of samples SCM-Change and SCM-Swap are examined to discover if these changes of mutation operator led overall to a statistically significant difference in the obtained results. Detailed of the statistic test is shown in Table 2.

As can be seen from Table 2, for the CluSPT problem, the SCM mutation produces better results than using either change or swap mutation separately on 10 out of 11 test instances. A Wilcoxon signed-rank test showed that the combination of 2 variations of mutation operator inside the SCM provided a statistically significant improvement in the quality of obtained solutions with the Asymp. Sig. (2-tailed) p of 0.026 and 0.021 for the SCM-Change and the SCM-Swap, respectively. Similarly for the InterCluMRCT problem, a considerable advance in performance of the proposed CC-MFEA can be witnessed when applying the SCM operator instead of either change or swap with $p = 0.028$ and $p = 0.005$ respectively. In conclusion, the proposed SCM mutation operator does statistically improve the overall performance of the algorithm.

Explanation: The first type of variation makes changes to the segments which are used for representing local spanning trees. Altering any two genes in those segments helps to modify the local spanning trees. In addition, changing local-root vertices is also an important factor since they contribute a major role in making connections between clusters, from which a global tree will be constructed. Therefore, the combination of these two types of variation in the process of creating new individuals helps to increase the diversity of the population, thereby helping the proposed algorithm to explore better solutions.

6.3.2 The effect of encoding techniques in single task

In this section, the performance of each Cayley Code is evaluated in either CluSPT or InterCluMRCT problem.

A. Overall results of singletasking

Table 3 summarizes the comparison results among 3 encoding types when performing in single task. The "Num.Ins" column denotes the total number of instances, while the "Dandelion", "Blob" and "Prüfer" columns represent the number of instances where the encoding type has better results than the other two.

A notable point in the results shown on those tables is that the performance of Dandelion Code is by far the best. In particular, the Dandelion Code outperforms the remaining codes on 2 out of 3 types of instances (Type 1 and Type 6). The result obtained by using the Prüfer Code seems to be the

Table 1 Results obtained by variations of the mutation operator using Dandelion code on instances in Type 1

	Instances	Change variation		Swap variation		SCM	
		BF	Avg	BF	Avg	BF	Avg
CluSPT	10berlin52	54234.3	61719.5	52614.2	61693.7	49068.3	*58599.4*
	10eil76	3341.5	4046	3189.8	4120.9	2762.6	*3447.1*
	10pr76	734240.7	893274.5	737821.8	996312.6	644302.7	*828170.9*
	15eil51	2328.2	2903.7	2504.2	3144	2125.2	*2758.5*
	15pr76	992773.1	1305765.5	1049556.1	1331127.9	952262.1	*1214188.8*
	25eil101	6844.1	9383.9	8052.7	10318.7	6708.3	*9294*
	25lin105	202693.9	298671.7	209999.5	315898.4	155233.0	*281917*
	50eil101	8275.1	12774.7	7446.2	14153.6	9788.0	*12654*
	50kroB100	373556.8	*553207.3*	432215.9	596317.9	491024.2	606069.8
	5eil51	2247.5	2616	1940.1	2350.7	1955.0	*2198.5*
	5pr76	894432.2	1029626.4	739250.3	907498.9	667837.8	*822934.7*
InterCluMRCTSPT	10eil51	60507.4	71196.5	52260.9	70634.7	50494.5	*62486.8*
	10kroB100	12524450.7	14206655.2	10315981.7	12897154.7	9405236.7	*12039989.7*
	10rat99	798031.8	890791.1	683229.7	797539.2	589667.7	*749772.7*
	15eil76	163299.6	198846.7	164718.9	201447.4	146785.6	*181834.5*
	15st70	168131.6	238685.5	172627.3	229877.8	149009.4	*203731.9*
	25kroA100	14332444.4	19635063.4	14818748.8	20262451.9	12872635.2	*18974511.3*
	25rat99	770558.3	998865.2	725557.7	1089673.5	638380.1	*968377.2*
	50kroA100	24515550.2	30445137.8	24369101.3	30863904.2	22757582.7	*30014740.7*
	50lin105	14569661.1	*22189079.5*	14751843.8	23326248.7	14536077.1	23186711.8
	50rat99	1236037.0	1497889.5	1119216.1	1547733.4	1040326.9	*1404530*
	5eil76	169865.6	196664.3	115177.8	*142784*	129445.0	143326.5
	5st70	218895.4	244249.2	154823.3	*183927.2*	168499.6	187038.6

Table 2 Wilcoxon signed-ranks test statistics between SCM-Swap and SCM-change to show the efficiency of proposed mutation operator

		SCM-Change			SCM-Swap		
		N	Mean rank	Sum of ranks	N	Mean rank	Sum of ranks
CluSPT	Negative Ranks	10	5.8	58	10	5.9	59
	Positive Ranks	1	8	8	1	7	7
	Z	−2.223			−2.312		
	Asymp. Sig. (2-tailed)	**0.026**			**0.021**		
InterCluMRCT	Negative Ranks	11	6.09	67	10	7.5	75
	Positive Ranks	1	11	11	2	1.5	3
	Z	−2.197			−2.824		
	Asymp. Sig. (2-tailed)	**0.028**			**0.005**		

N: number of instances
Negative Rank: SCM's Avg $<$ Swap/Change's Avg
Positive Rank: SCM's Avg $>$ Swap/Change's Avg

Table 3 Comparison among 3 Cayley Codes to summarize the number of instances in which one encoding type outperforms the other two when performing in single task

Types	CluSPT			InterCluMRCT			Num.Ins
	Dandelion	Blob	Prüfer	Dandelion	Blob	Prüfer	
Type 1	15	3	6	12	4	8	24
Type 5	6	7	3	4	3	9	16
Type 6	18	10	6	17	8	9	34

worst when its performance is poorer than the Blob and the Dandelion Code in all 3 types.

B. Detailed results of singletasking

For more details of the CluSPT, in Type 1, the Dandelion Code outperforms both Prüfer Code and Blob Code in 15 out of 24 test cases. In Type 6, the Dandelion Code exceeds Prüfer Code and Blob Code on 21 out of 34 and 20 out of 34 test instances respectively.

Unlike Type 1 and Type 6, in Type 5, the Blob Code is slightly superior to 2 remaining encoding techniques, i.e. the Blob Code exceeds the Prüfer Code and Dandelion Code in 9 out of 16 and a half of test cases respectively.

Tables 4, 5 and 6 show more details of the the results obtained by the proposed algorithm on single-task when solving the CluSPT on instances of Type 1, Type 5 and Type 6.

In the InterCluMRCT, the Dandelion Code receives the best performance when that encoding mechanism exceeds two remaining encoding techniques on two-thirds of the total number of instances. In detail, Dandelion Code outperforms Blob Code on 14 out of 24 instances of Type 1 and on 22 out of 34 instances of Type 5. In comparison with Prüfer Code, Dandelion Code surpasses 15 out of 24 instances of Type 1 and on 22 out of 34 instances of Type 6.

A remarkable point occurs on the results in Type 5 where Prüfer Code exceeds both Dandelion and Blob Code on 10 out of 16 instances. However, in general, the performance of Prüfer Code is not as well as either Dandelion Code or Blob Code when its results are worse than those of Dandelion Code and Blob Code in two out of three instance types.

The details of the InterCluMRCT's results received from single-tasking of the proposed algorithm with different Cayley codes are presented in Tables 7, 8 and 9.

C. Statistical analysis of 3 Cayley Codes in singletasking

In order to examine the effect of different Cayley Codes on the obtained results in singletasking environment, a Friedman test is firstly carried out to see if there were differences in perceiving better performance of the algorithm based on 3 encoding types.

For the CluSPT problem, detailed of the statistic test is given in Table 10. The Friedman Test compares the *Mean* ranks between the related groups and provides the test statistic χ^2 value *Chi-square*, degrees of freedom *df* and the significance level *Asymp. Sig.* to indicate how the groups differed. Results of the Friedman Test show that there was a statistically significant difference in the CluSPT's results depending on which type of the encoding mechanism was used, with $\chi^2(2) = 10.0811,\ p = 0.0065$.

However, at this stage, we only know that there are differences somewhere between the related groups, but the Friedman Test does not pinpoint which groups in particular differ from each other. As a result, another post hoc test needs to be run. In this case, separate Wilcoxon signed-rank tests on the different combinations of related groups (Dandelion–Prufer, Blob–Dandelion and Prufer–Blob) are chosen to be conducted. In addition, a Bonferroni adjustment on the Wilcoxon test's results must be applied since we are making multiple comparisons. The initial significance level (0.05) is simply divided by the number of running tests, which results in a new significance level of 0.05/3 = 0.017 in our case.

Table 10 also shows the output of the Wilcoxon signed-rank test on each of our combinations. We can see that at the $p < 0.017$ significance level, a statistically significantly improvement can be obtained by applying the Dandelion Code instead of the other two for the CluSPT problem in singletasking, with Dandelion–Prufer and Blob–Dandelion of $p = 0.0003$ and $p = 0.0099$ respectively. Therefore, Dandelion Code is considered as the best among 3 encoding types for evolutionary algorithm to solve the CluSPT problem.

Similarly, Table 11 shows the results of statistical analysis among 3 Cayley Codes for the InterCluMRCT problem. There was a statistically significant difference in perceived effort depending on which type of encoding used for this problem with $\chi^2(2) = 3.4324,\ p = 0.0179$. Post hoc analysis with Wilcoxon signed-rank tests was conducted with a Bonferroni correction applied, resulting in a significance level set at $\text{p} < 0.017$. Median (IQR) achieved for Prufer, Dandelion and Blob Code were 1112182.75, 1111663.90 and 1130830.75, respectively. There were no significant differences between Blob to Dandelion Code ($Z = -0.606,\ p = 0.5445$) or between Prufer to Blob Code ($Z = -0.773,\ p = 0.4395$) despite an overall improvement in the Blob vs Prufer Code. However, there was a statistically significant reduction in performance in the Prufer Code compare to Dandelion Code ($Z = -1.872,\ p = 0.0161$). As a consequence, it can be concluded that Dandelion Code outperforms the remaining two encoding mechanisms when solving clustered tree problems.

D. Explanation of the obtained results

In type 1 and type 6, all of the instances are modified from the TSPLIB [21]. In details, type 1 uses a k-means algorithm to create clusters, then a source vertex is randomly selected to create instances for CluSPT; and type 6 uses grid method clustering: all vertices are divided into $m * n$ rectangles with the same area. The properties of these instances in type 1 and type 6 are: clusters are separated clearly and less messed up, the vertices in a cluster are close to each other, and the distance between two vertices belonging to two different clusters is greater than the distance between two vertices belonging to the same one. Using the code with high locality (a small change in genotype causes a small change in phenotype) like Blob code and Dandelion code is effective for instances with the above properties. In our encoding method, each cluster is encoded as a segment with different types of Cayley codes, and a small change in these segments will have different

Table 4 CluSPT'S results obtained by single task on instances in Type 1

Instances	Prüfer Code			Dandelion Code			Blob Code		
	BF	Avg	Time	Dande.BF	Avg	Time	BF	Avg	Time
10berlin52	48231.6	***56930.5***	0.02	49389.1	*57340.9*	0.03	50538.3	57971	0.02
10eil51	2259.3	2748.9	0.02	2143.2	***2652.3***	0.03	2292.3	*2693.2*	0.02
10eil76	3054.4	4037.4	0.03	3035.7	***3851.7***	0.05	3147.4	*3960*	0.03
10kroB100	194594.2	279317.4	0.05	213508.1	***272930.7***	0.08	219140.4	*277438*	0.03
10pr76	739479.0	919138.6	0.03	745736.3	***906817.1***	0.07	812444.9	970703.8	0.03
10rat99	11284.8	14780.4	0.05	11689.4	14520.2	0.08	10826.1	***14297.2***	0.03
15eil51	1977.7	2655.6	0.02	1923.4	***2619.9***	0.03	2217.5	2658.4	0.02
15eil76	3898.9	***4899.1***	0.03	3910.9	5008.1	0.07	3864.9	4935.5	0.03
15pr76	931016.2	1153653.4	0.03	916803.3	1142816.9	0.07	948248.6	***1132786.***2	0.03
15st70	5628.6	***6702.7***	0.03	5275.8	6897.8	0.05	5556.4	6802.4	0.03
25eil101	7117.2	***8445.7***	0.05	6854.5	8777.3	0.08	7013.1	8638.6	0.05
25kroA100	239682.2	322796.9	0.05	210245.2	***315251.9***	0.08	246531.4	332016.3	0.05
25lin105	187275.2	242850	0.05	190351.0	239917.7	0.08	183321.3	***235841.7***	0.07
25rat99	13956.4	17371.4	0.05	13529.3	***16323.5***	0.07	13383.1	*17054.8*	0.07
50eil101	8269.5	10885.3	0.05	7961.6	***9965.7***	0.07	8719.7	*10644.5*	0.10
50kroA100	372389.2	511675.4	0.05	363383.9	***460977.6***	0.07	399591.5	*498709.7*	0.10
50kroB100	366297.9	478398.2	0.05	329645.2	***445994.3***	0.07	368230.9	*473867.2*	0.10
50lin105	294923.8	385231.3	0.05	275393.3	***349959.6***	0.07	274885.4	*374014.8*	0.10
50rat99	16793.8	26184.1	0.05	16679.2	***22536***	0.07	17506.0	*24348*	0.08
5berlin52	31166.2	38102.9	0.02	28238.3	***36913.1***	0.02	31346.0	38123	0.03
5eil51	2275.6	***2717.1***	0.02	2328.4	*2746*	0.02	2293.7	2753	0.03
5eil76	3855.3	4852.4	0.03	4069.9	***4849.2***	0.03	3930.3	4938.1	0.05
5pr76	879111.5	1071822.5	0.03	821911.6	***1046564.7***	0.03	859905.0	*1050015.2*	0.05
5st70	5808.4	***6830***	0.03	5541.7	*6930.5*	0.03	6022.6	7071	0.05

Table 5 CluSPT'S RESULTS OBTAINED BY SINGLE TASK ON INSTANCES IN TYPE 5

Instances	Prüfer Code			Dandelion Code			Blob Code		
	BF	Avg	Time	BF	Avg	Time	BF	Avg	Time
10i120-46	127591.3	147596.8	0.07	123892.0	***144535.6***	0.10	129441.2	147613.6	0.07
10i45-18	31533.1	36981.6	0.02	29406.9	35997.6	0.03	29941.3	***35617.5***	0.02
10i60-21	46134.5	55344.3	0.02	48239.6	***55184.1***	0.05	46324.9	57107.7	0.03
10i65-21	48825.4	61886.7	0.03	52769.9	61366.4	0.05	50870.7	***61348.4***	0.03
10i70-21	52408.0	***62816.2***	0.03	54582.1	64193.1	0.05	50099.1	63887.9	0.05
10i90-33	72058.8	86808.5	0.03	73900.3	84450.3	0.08	74143.4	***83540.8***	0.07
5i120-46	88215.5	***106548.5***	0.07	89735.6	*110200.1*	0.12	90910.7	110628.1	0.12
5i30-17	15019.7	*18559.9*	0.02	15123.9	19059.1	0.02	15707.2	***18460***	0.02
5i45-18	18732.5	*25755.5*	0.02	21365.2	25788.2	0.03	19776.6	***25369***	0.03
5i60-21	42073.7	50615.6	0.02	39619.2	50585.7	0.05	39399.4	***48939.4***	0.03
5i65-21	44363.7	52971.5	0.03	40048.8	***50534.3***	0.05	43271.2	53701.7	0.05
5i70-21	46501.6	***57623.9***	0.03	47731.6	58653.2	0.05	45167.9	57788.5	0.05
5i75-22	46924.0	59739.7	0.03	48160.9	***58589.9***	0.05	47359.3	*58892.7*	0.05
5i90-33	70015.4	80680.3	0.03	68261.6	***80092***	0.05	69887.0	81086.3	0.07
7i60-21	48741.4	58940.3	0.02	48699.2	***57659.1***	0.03	49495.7	59543.4	0.03
7i65-21	49922.9	62428.3	0.03	48497.8	60235.8	0.03	48083.2	***59650.1***	0.05

Table 6 CluSPT'S results obtained by single task on instances in Type 6

Instances	Prüfer Code			Dandelion Code			Blob Code		
	BF	Avg	Time	BF	Avg	Time	BF	Avg	Time
10berlin52-2x5	34600.0	***45911.4***	0.02	34757.6	46754.6	0.03	34651.4	46368.7	0.02
12eil51-3x4	2273.3	2650.5	0.02	2126.1	***2588.3***	0.03	2149.6	*2631.1*	0.02
12eil76-3x4	3729.8	*4536.6*	0.03	3476.3	4544.3	0.07	3734.8	***4513.9***	0.05
12pr76-3x4	872118.5	1061867.5	0.03	873703.6	1053138.3	0.05	829010.5	***1045190.8***	0.05
12st70-3x4	5215.2	6506.6	0.03	5178.9	***6317***	0.05	5164.2	*6460.9*	0.05
15pr76-3x5	881691.8	1059162.6	0.03	811904.5	***989953.8***	0.07	834225.6	*1009128.6*	0.05
16eil51-4x4	2042.5	2591.7	0.02	2095.8	***2510.7***	0.03	2053.0	2596.7	0.03
16eil76-4x4	3641.9	*4524.7*	0.03	3737.5	4730	0.07	3249.8	***4518.6***	0.05
16lin105-4x4	187298.4	241744.6	0.05	193278.6	***236213.9***	0.10	189307.9	243894.9	0.08
16st70-4x4	4732.5	6089.1	0.03	4748.5	5994.3	0.05	4769.8	***5965.2***	0.05
18pr76-3x6	910086.9	1134622.3	0.03	932884.2	***1112377.7***	0.07	1022655.1	1169389	0.05
20eil51-4x5	2935.7	***3545.7***	0.02	2986.3	3644.7	0.03	2997.9	3630.6	0.03
20eil76-4x5	3691.7	***4909.4***	0.03	3987.1	*5117*	0.05	4254.4	5421	0.05
20st70-4x5	4770.3	6539	0.03	5020.9	***6349.6***	0.03	5180.3	*6481.7*	0.05
25eil101-5x5	6067.5	8315.9	0.05	6571.4	***8200.6***	0.07	6719.3	*8304*	0.08
25eil51-5x5	2607.5	3309.9	0.02	2547.3	***3270.9***	0.02	2748.1	3416.9	0.03
25eil76-5x5	3500.3	5346.9	0.03	4340.3	***5287.4***	0.03	4061.7	5516.6	0.05
25rat99-5x5	16537.0	***19547.5***	0.05	16220.1	*20357.3*	0.05	16370.5	20868.9	0.07
28kroA100-4x7	278119.8	356025.1	0.05	225349.3	***332745.4***	0.05	283240.3	359197.4	0.05
30kroB100-5x6	318830.6	402822.5	0.05	308115.6	***381350.7***	0.05	309228.4	*390918.3*	0.05
35kroB100-5x5	278483.0	334016.4	0.05	239402.3	***313986.9***	0.05	243024.2	*328924.3*	0.05
36eil101-6x6	6908.9	8924.4	0.05	6409.0	***8870.3***	0.07	7503.1	9460.9	0.05
42rat99-6x7	16454.9	22549.1	0.05	15773.8	***20780***	0.07	16262.3	*22342.9*	0.05
4berlin52-2x2	31117.4	*41498.2*	0.02	31507.1	43049.8	0.02	31747.9	***40229.4***	0.02
4eil51-2x2	2530.9	3068.7	0.02	2567.9	3028.9	0.02	2545.1	***3009.3***	0.02
4eil76-2x2	3989.6	*5327.5*	0.03	4204.5	5353.7	0.03	4269.2	***5251***	0.03
4pr76-2x2	841497.0	*1063294.6*	0.03	813612.6	1098560.2	0.03	820843.4	***1036132***	0.03
6berlin52-2x3	41748.2	*49495.8*	0.02	40940.2	50510.3	0.02	41837.6	***49244.1***	0.02
6pr76-2x3	880775.9	1104705.5	0.03	856589.4	***1068609.2***	0.03	879161.2	*1082865.7*	0.03
6st70-2x3	4499.2	6268.2	0.02	5044.0	***6174.5***	0.03	5130.1	6308.3	0.03
8berlin52-2x4	34594.1	***44883.6***	0.02	37810.1	47899.4	0.02	35684.8	45298.4	0.02
9eil101-3x3	4973.3	6427.5	0.03	4866.7	***6112***	0.05	4839.3	*6282.6*	0.05
9eil51-3x3	2437.7	***2841.3***	0.02	2276.9	2922.7	0.02	2412.1	2874.1	0.02
9eil76-3x3	3876.1	*4887.3*	0.03	4072.3	4913.8	0.03	4054.3	***4856.7***	0.03

affects on the solution depending on the types of encoding has been used. For instance, in CluSPT, the Dandelion code is the highest locality among Cayley codes, so a small change in a single cluster segment is similar to performing a local search to obtain optimal structure for a cluster. As a result, the Dandelion code is the most effective encoding for the CluSPT in type 1 and type 6.

6.3.3 The effect of encoding techniques when CC-MFEA solves two different problems

To evaluate the effectiveness of each Cayley codes' encoding technique for the CC-MFEA algorithm when solving clustered problems, the CluSPT and the InterCluMRCT problems with different input graphs are implemented.

A. Overall results of multitasking

Table 12 presents a summary of comparison results among 3 encoding types when performing in multi-tasks. Similarly, the "Num.Ins" column indicates the number of instances,

Table 7 InterCluMRCT'S results obtained by single task on instances in Type 1

Instances	Prüfer Code			Dandelion Code			Blob Code		
	BF	Avg	Time	BF	Avg	Time	BF	Avg	Time
10berlin52	746471.5	1023526.6	0.02	792361.5	***1010288.6***	0.03	783512.1	*1021982.4*	0.03
10eil51	54112.3	***71066***	0.02	63468.3	*71817.7*	0.03	61643.5	72022.6	0.03
10eil76	151067.5	178340.7	0.03	141267.2	***172323.6***	0.05	142672.4	*174096.1*	0.05
10kroB100	13796654.8	***16553385***	0.05	14268396.3	17162543.8	0.08	14088040.1	16702067	0.08
10pr76	33652106.5	39763551.6	0.05	33210205.5	39015916.6	0.05	31661498.3	***38824375.1***	0.05
10rat99	663406.3	***790745.9***	0.07	707663.1	814165.9	0.08	668456.1	800100.2	0.07
15eil51	67491.4	***78011.2***	0.03	65482.8	79942.7	0.03	63989.5	78261.9	0.03
15eil76	159242.1	***191385.1***	0.05	175205.2	199709.5	0.07	161537.2	194755.4	0.07
15pr76	33722952.2	***40019937.8***	0.05	33588177.6	42043159.5	0.07	33387637.8	41600496.1	0.07
15st70	190660.0	*237189.8*	0.05	190905.2	241981.3	0.05	184618.6	***235648.7***	0.05
25eil101	327629.2	408284.1	0.08	306631.3	***381127.1***	0.10	308306.7	411172.7	0.10
25kroA100	13818342.2	***16681116.5***	0.08	13938211.0	17397605.3	0.10	13767588.8	16922818.9	0.10
25lin105	10132848.0	12438176	0.08	9737057.8	***12165628.2***	0.08	9968455.1	*12298921.1*	0.10
25rat99	755224.0	941903.2	0.07	723944.8	925427.4	0.07	727798.7	***916487.1***	0.07
50eil101	425487.2	528523.3	0.13	393815.4	***491051.1***	0.15	419381.6	*516375.6*	0.13
50kroA100	20264432.1	25425696.2	0.15	19336846.6	***23015435.9***	0.13	18039553.6	*25149684.4*	0.13
50kroB100	17103857.1	23695144.6	0.13	15593970.2	***21992772.3***	0.15	16782080.2	*22632886.3*	0.13
50lin105	15073577.9	18619720.9	0.15	12435688.7	***17134173.1***	0.15	13574535.9	*18447964.4*	0.13
50rat99	893463.0	1299325	0.15	861555.2	***1169691.2***	0.15	1011872.1	*1266109.5*	0.15
5berlin52	872174.2	1099731.5	0.02	916097.0	***1089938***	0.02	910127.2	1138311.5	0.02
5eil51	54876.1	69552.6	0.02	51640.7	***66775.5***	0.02	54916.6	*68553.1*	0.02
5eil76	159213.2	***186874.1***	0.03	163540.7	*189590.1*	0.03	158062.0	191326.2	0.03
5pr76	31210870.0	41050974.8	0.03	31235143.3	40849463	0.03	30754662.4	***40766023***	0.03
5st70	191689.3	228669.2	0.03	178043.5	***223776.9***	0.03	188860.3	229533	0.03

Table 8 InterCluMRCT'S results obtained by single task on instances in Type 5

Instances	Prüfer Code			Dandelion Code			Blob Code		
	BF	Avg	Time	BF	Avg	Time	BF	Avg	Time
10i120-46	6824402.6	***8467621.1***	0.07	7034856.6	*8617081.5*	0.10	6907481.4	8836473.3	0.10
10i45-18	670232.5	***791809***	0.02	661873.4	822888.2	0.03	699077.3	805769.4	0.03
10i60-21	1595726.5	1769811.2	0.03	1514518.4	1753339.3	0.03	1502691.2	***1725612.3***	0.03
10i65-21	1747048.7	*2125538.4*	0.03	1712375.6	2159277.8	0.05	1753710.3	***2024495.4***	0.05
10i70-21	1889135.5	***2344561.7***	0.03	1922651.3	*2344844.3*	0.05	2053948.1	2385415.6	0.05
10i90-33	3274350.0	3979138.6	0.05	3362592.3	3907614.6	0.07	3335125.1	***3907300.4***	0.07
5i120-46	5094230.5	***6245030.4***	0.08	4993145.2	6268512.6	0.10	5169899.8	6251659.6	0.10
5i30-17	211380.1	***260666.6***	0.02	211397.1	*266981.9*	0.02	223833.1	268543.2	0.02
5i45-18	472667.1	***660714.7***	0.02	534147.9	669612.7	0.03	497659.8	*642935*	0.03
5i60-21	1192221.8	1474762.6	0.03	1182136.1	***1448177.2***	0.03	1248651.0	1499027.6	0.03
5i65-21	1357589.0	***1670700.8***	0.03	1254074.3	1698251.5	0.05	1404842.2	1685034.5	0.03
5i70-21	1664789.5	***2082563.3***	0.03	1805550.2	2126418.5	0.05	1797140.8	2119328	0.05
5i75-22	1851383.3	2500164.3	0.03	2109987.4	***2490614.9***	0.05	2116155.1	2501844.7	0.05
5i90-33	2770257.3	3258250.6	0.05	2506432.2	***3231446.7***	0.07	2591577.0	*3239316*	0.07
7i60-21	1348066.8	***1574816.5***	0.03	1285355.4	*1602551.3*	0.03	1253996.4	1617098.8	0.03
7i65-21	1597584.5	1907126.4	0.03	1542184.8	***1885584.3***	0.03	1466751.4	*1898387.3*	0.03

Table 9 InterCluMRCT'S results obtained by single task on instances in Type 6

Instances	Prüfer Code			Dandelion Code			Blob Code		
	BF	Avg	Time	BF	Avg	Time	BF	Avg	Time
10berlin52-2x5	1140052.7	*1361030.6*	0.02	1104343.8	1365297.6	0.03	1127904.6	***1359052.8***	0.03
12eil51-3x4	66649.8	***76266.4***	0.02	66696.7	*76374*	0.03	68473.6	77301.6	0.03
12eil76-3x4	172400.9	201436.8	0.05	168784.8	***198603.7***	0.05	167095.7	*200373.3*	0.05
12pr76-3x4	36169117.0	44045982.4	0.05	37317537.7	43716173.7	0.05	37724322.6	***43453583.2***	0.05
12st70-3x4	211691.9	236117.5	0.03	200523.6	***232949.1***	0.05	208799.4	236681.7	0.05
15pr76-3x5	38339513.5	***44573369.7***	0.05	37648842.0	*45577478.1*	0.05	39989085.6	46864019	0.05
16eil51-4x4	65847.9	82175.1	0.03	67430.3	***80433.7***	0.03	70019.3	*81640.9*	0.03
16eil76-4x4	182349.8	222359.5	0.05	191031.3	***218976.4***	0.05	175988.5	222384.3	0.07
16lin105-4x4	11289300.3	13843221	0.07	11935794.2	13676727.1	0.08	11860484.3	***13503620.2***	0.08
16st70-4x4	212835.0	***239932.2***	0.05	205291.7	246809.1	0.05	207937.4	244510	0.05
18pr76-3x6	39845642.1	***47281209.2***	0.05	38661052.9	48371882.1	0.05	38469508.1	47289023.3	0.07
20eil51-4x5	72259.1	***84701.2***	0.03	72763.9	*85757.1*	0.03	72913.1	87367	0.05
20eil76-4x5	185956.9	***230027.2***	0.05	192716.0	231254.6	0.05	198690.1	230352.6	0.07
20st70-4x5	228864.6	278513.4	0.05	227233.3	***259492.4***	0.05	226995.0	*264383.6*	0.05
25eil101-5x5	398649.6	*485888.5*	0.07	442646.3	485930.1	0.07	403997.5	***481192.1***	0.08
25eil51-5x5	72992.5	94947.6	0.03	74214.8	***89979.5***	0.03	77283.9	*93512.4*	0.03
25eil76-5x5	191113.1	***241099.7***	0.05	210432.0	*248977.5*	0.05	205127.4	249123.2	0.05
25rat99-5x5	871893.4	1008712.9	0.07	880323.2	***988345.9***	0.07	885567.7	1031054.7	0.07
28kroA100-4x7	16340000.7	20243861.6	0.07	16036172.5	20158800.9	0.08	15839245.3	***19923884.2***	0.07
30kroB100-5x6	16860172.3	20439054.5	0.08	15635937.5	***19359205.2***	0.08	15935680.9	*19905827.5*	0.08
35kroB100-5x5	15960189.5	19265632.3	0.07	15988573.7	***19053165***	0.07	14955246.4	19389989	0.07
36eil101-6x6	443323.4	524565.8	0.10	457212.6	***522975.8***	0.10	401033.0	*523380.3*	0.10
42rat99-6x7	1024158.8	1242803.5	0.12	1024908.1	***1184611.1***	0.12	970807.6	*1208429.7*	0.10
4berlin52-2x2	907425.9	*1124634*	0.02	905153.4	1133389.8	0.02	889462.2	***1123350***	0.02
4eil51-2x2	56459.2	74854.5	0.02	58505.9	***72206.9***	0.02	52288.9	*74762*	0.02
4eil76-2x2	160715.6	195154.7	0.03	147420.9	***187954.3***	0.03	156467.5	*193645.5*	0.03
4pr76-2x2	32689617.2	43385238.1	0.03	32119944.7	42506559.6	0.03	31044018.0	***41670527.5***	0.03
6berlin52-2x3	972924.2	***1185002.8***	0.02	896361.7	*1206810.3*	0.02	1037577.5	1213282.5	0.02
6pr76-2x3	33564662.7	43174503.3	0.03	32209495.7	42994204.3	0.03	32625734.1	***42284141.5***	0.03
6st70-2x3	211351.4	244688.3	0.03	208281.3	***239580.6***	0.03	208027.5	247581.4	0.03
8berlin52-2x4	1224228.2	1383237.7	0.02	1086319.5	***1358871.1***	0.02	1115977.8	*1366045.4*	0.02
9eil101-3x3	327227.3	398819.2	0.05	317411.2	***392483.3***	0.05	345169.4	406165.2	0.05
9eil51-3x3	63735.2	76139.3	0.02	65287.6	***75633.9***	0.02	69260.5	76391.5	0.02
9eil76-3x3	170199.5	***187158.2***	0.03	171578.0	196198.3	0.03	169731.9	192252.2	0.03

while the "Dandelion", "Blob" and "Prüfer" columns show the number of instances where the encoding type outperforms the two remaining codes.

Tables 13, 14 and 15 show more detail of the results obtained by 3 different types of Cayley codes on instances of Type 1, Type 5 and Type 6. In those tables, the selection of pairs of instances to apply CC-MFEA is chosen randomly. For the convenience of tracking, pairs of instances of the CluSPT problem and the InterCluMRCT problem appear in the corresponding order of occurrence of instances of each problem. For example, Instance 10berlin51 in the first row in the result table of the CluSPT is matched with Instance 10Eil51 in the first row in the result table of the InterCluMRCT problem. Similarly, Instance 10Eil76 is paired with instance 10KroA100, etc.

Results on those tables point out that on the CluSPT problem, the Blob Code achieves the best results, while the performance of Prüfer Code is the worst. CC-MFEA using Blob Code exceeds both the Dandelion Code and Prüfer Code on 7 out of 12 instances of Type 1, 5 out of 8 instances of Type 5 and 11 out of 18 instances of Type 6.

Table 10 Statistical analysis on the effect of 3 Cayley Codes for CluSPT in singletasking

		Prufer Code	Dandelion Code	Blob Code
Friedman Test	Mean	184825.4378	179898.7216	183283.1108
	Median Percentiles	37542.25	36455.35	36870.25
	Mean Rank	2.2297	1.7162	2.054
	Number of instances	74		
	Chi-square	10.0811		
	df	2		
	Asymp. Sig.	0.0065		

		Dandelion–Prufer		Blob–Dandelion		Prufer–Blob	
		Negative	Positive	Negative	Positive	Negative	Positive
Wilcoxon Signed Ranks Test	Number of instances	50	24	29	45	33	41
	Mean Ranks	41.24	29.71	31.34	41.47	33.85	40.44
	Sum of Ranks	2062	713	909	1866	1117	1658
	Z	−3.634		-2.578		-1.457	
	(based on)	(positve rank)		(negative rank)		(negative rank)	
	Asymp. Sig. (2-tailed)	0.0003		0.0099		0.145	

Negative Rank: former encoding's Avg < latter encoding's Avg
Positive Rank: former encoding's Avg > latter encoding's Avg

Table 11 Statistical analysis on the effect of 3 Cayley Codes for InterCluMRCT in singletasking

		Prufer Code	Dandelion Code	Blob Code
Friedman Test	Mean	7978735.7649	7918805.3203	7943743.8473
	Median Percentiles	1112182.75	1111663.90	1130830.75
	Mean Rank	2.0811	1.8243	2.0946
	Number of instances	74		
	Chi-Square	3.4324		
	df	2		
	Asymp. Sig.	0.0179		

		Dandelion–Prufer		Blob–Dandelion		Prufer–Blob	
		Negative	Positive	Negative	Positive	Negative	Positive
Wilcoxon Signed Ranks Test	Number of instances	43	31	30	44	37	37
	Mean Ranks	40.35	33.55	42.59	34.09	41.38	33.62
	Sum of Ranks	735	1040	1275	1500	1531	1244
	Z	−1.872		−0.606		−0.773	
	(based on)	(positve rank)		(negative rank)		(positive rank)	
	Asymp. Sig. (2-tailed)	0.0161		0.5445		0.4395	

Negative Rank: former encoding's Avg < latter encoding's Avg
Positive Rank: former encoding's Avg > latter encoding's Avg

Table 12 Comparison among 3 Cayley Codes to summarize the number of instances in which one encoding type outperforms the other two when performing in multi-task

Types	CluSPT			InterCluMRCT			Num.Ins
	Dandelion	Blob	Prüfer	Dandelion	Blob	Prüfer	
Type 1	3	5	4	5	3	4	12
Type 5	2	4	2	2	5	1	8
Type 6	5	8	4	4	5	8	17

Table 13 Results of CluSPT and InterCluMRCT obtained by multitasking in Type 1

	Instances	Prüfer Code			Dandelion Code			Blob Code		
		BF	Avg	Time	BF	Avg	Time	BF	Avg	Time
CluSPT	10berlin52	49171.8	***57723***	0.02	49068.3	*58599.4*	0.03	51017.3	59520.4	0.02
	10eil76	2821.9	3562.1	0.03	2762.6	***3447.1***	0.10	2970.1	3576.1	0.05
	10pr76	615090.9	891572.8	0.05	644302.7	***828170.9***	0.10	686257.4	*876266.3*	0.07
	15eil51	1963.1	***2571.3***	0.03	2125.2	2758.5	0.07	1915.2	2621.8	0.03
	15pr76	924602.5	1243668.1	0.03	952262.1	1214188.8	0.08	892070.6	***1153573.3***	0.05
	25eil101	7361.5	9524	0.08	6708.3	9294	0.15	6366.4	***9073.5***	0.10
	25lin105	200038.7	283849.1	0.08	155233.0	281917	0.15	222637.2	***273875.1***	0.10
	50eil101	10303.0	13901.6	0.10	9788.0	***12654***	0.20	10448.3	*13736.1*	0.13
	50kroB100	466620.5	***597271.3***	0.10	491024.2	*606069.8*	0.20	471673.5	609307.2	0.17
	5berlin52	26908.4	32936.3	0.07	27351.5	32335.8	0.15	28618.6	***32145.8***	0.12
	5eil51	1956.3	***2134.6***	0.02	1955.0	2198.5	0.07	1928.1	2159.6	0.05
	5pr76	696302.8	833947.7	0.03	667837.8	822934.7	0.08	700079.7	***817502.7***	0.05
CluMRCT	10eil51	50068.5	***61907.6***	0.02	50494.5	62486.8	0.03	52288.7	62395.7	0.02
	10kroB100	10681358.7	*12034128.9*	0.03	9405236.7	12039989.7	0.10	9655529.9	***11725746.2***	0.05
	10rat99	648755.1	754502.2	0.05	589667.7	749772.7	0.10	656638.0	***748928.5***	0.07
	15eil76	151472.3	*180803.1*	0.03	146785.6	181834.5	0.07	151820.9	***179928***	0.03
	15st70	154134.6	***200761.1***	0.03	149009.4	203731.9	0.08	153113.6	202267.5	0.05
	25kroA100	12731472.8	***18556355.8***	0.08	12872635.2	*18974511.3*	0.15	14045547.0	19684920.5	0.10
	25rat99	640757.7	***947489.7***	0.08	638380.1	968377.2	0.15	767393.0	967642.1	0.10
	50kroA100	23367861.1	32942804.3	0.10	22757582.7	***30014740.7***	0.20	22739300.1	*31405049.2*	0.13
	50lin105	16160280.4	24914576.8	0.10	14536077.1	***23186711.8***	0.20	17307668.0	*23414936.6*	0.17
	50rat99	1139849.3	1617337.4	0.07	1040326.9	***1404530***	0.15	1150837.1	*1462733.9*	0.12
	5eil76	128634.6	151316.2	0.02	129445.0	***143326.5***	0.07	122439.5	*146601.8*	0.05
	5st70	170213.3	192282.7	0.03	168499.6	***187038.6***	0.08	166349.8	*189877.4*	0.05

Table 14 Results of CluSPT and InterCluMRCT obtained by multitasking in Type 5

	Instances	Prüfer Code			Dandelion Code			Blob Code		
		BF	Avg	Time	BF	Avg	Time	BF	Avg	Time
CluSPT	10i120-46	119274.5	***139729.8***	0.05	118007.6	*140695.5*	0.08	115367.0	142599	0.07
	10i60-21	39534.4	47114.1	0.03	37250.3	***44523.4***	0.05	39775.1	*45504.2*	0.03
	10i70-21	48511.5	***59162.3***	0.05	50452.1	59659.1	0.07	49256.1	59607.2	0.05
	5i120-46	80064.0	89246.7	0.05	79175.5	87539.1	0.08	78417.2	***86796.3***	0.07
	5i45-18	16027.7	19156.8	0.02	16808.7	***19140.4***	0.03	17164.5	19213.8	0.03
	5i65-21	35443.2	*39918*	0.03	34721.1	40453.4	0.05	35242.6	***39240.6***	0.05
	5i75-22	41200.5	*49636*	0.05	42183.6	50163.9	0.07	41935.5	***47678.7***	0.05
	7i60-21	41187.9	48769	0.03	41137.8	48030.8	0.03	40475.1	***47654.2***	0.03
CluMRCT	10i45-18	567251.9	***679895.5***	0.05	544324.8	*684692.6*	0.08	569770.7	707540	0.07
	10i65-21	1818548.3	2100265.2	0.03	1814976.1	***2025563.9***	0.05	1759075.8	*2069747.6*	0.03
	10i90-33	2820389.7	3205787.6	0.05	2792129.1	***3169196.4***	0.07	2660942.6	3234809.2	0.05
	5i30-17	210324.6	243252.3	0.05	218646.8	238543.8	0.08	204029.1	***236567.3***	0.07
	5i60-21	1102170.0	1228521.7	0.02	1053277.3	1192616.8	0.03	987225.0	***1170107.7***	0.03
	5i70-21	1670976.7	1954677.7	0.03	1611844.7	1916022.8	0.05	1644466.2	***1892638.6***	0.05
	5i90-33	2322865.9	*2686761.9*	0.05	2382107.8	2728685.9	0.07	2362352.9	***2676810***	0.05
	7i65-21	1496971.5	1726105.2	0.03	1399396.7	1685914.9	0.03	1487345.1	***1677823.4***	0.03

Table 15 Results of CluSPT AND InterCluMRCT obtained by multitasking in Type 6

	Instances	Prüfer Code			Dandelion Code			Blob Code		
		BF	Avg	Time	BF	Avg	Time	BF	Avg	Time
CluSPT	10berlin52-2x5	35834.9	46931.4	0.03	34863.0	***46741.1***	0.03	33905.9	54973.2	0.03
	12eil76-3x4	3257.3	***4056.1***	0.03	3387.9	4251.7	0.07	3418.4	4173.2	0.05
	12st70-3x4	5068.7	5933.6	0.03	5063.0	5919.9	0.07	4777.0	***5752.8***	0.05
	16eil51-4x4	2136.3	*2793.1*	0.03	2209.2	2952.3	0.05	2183.9	***2766.5***	0.05
	16lin105-4x4	198717.1	248286.2	0.07	197083.0	243695.8	0.08	181840.8	***237764.6***	0.08
	18pr76-3x6	992677.4	***1252081.2***	0.03	956350.5	1256947	0.07	1013355.5	1255231	0.05
	20eil76-4x5	3394.7	*5388.2*	0.05	3742.1	5609.2	0.07	4013.9	***5232.6***	0.05
	25eil101-5x5	6008.2	*8799.4*	0.05	6884.9	8915.7	0.07	6397.2	***8546.7***	0.07
	25eil76-5x5	4295.7	5726.3	0.05	4306.2	***5684.3***	0.07	4655.6	5961.3	0.07
	28kroA100-4x7	318142.2	402637.9	0.07	264152.5	***375928.5***	0.07	304285.8	*398713.1*	0.12
	35kroB100-5x5	350446.9	414469.1	0.07	285071.3	***364120.4***	0.08	276242.1	*376515*	0.13
	4berlin52-2x2	29131.2	*35850.2*	0.07	30759.0	36405.7	0.07	26919.7	***34162.2***	0.10
	4eil51-2x2	2204.9	2444.7	0.02	2193.6	2418.4	0.03	2242.3	***2403.1***	0.05
	4pr76-2x2	613529.3	719710.4	0.02	626890.3	718192	0.03	610440.0	***700149.8***	0.05
	6pr76-2x3	774395.7	891498.1	0.03	756620.4	***857282.9***	0.03	777367.1	*864282.5*	0.05
	8berlin52-2x4	32415.0	***46142.7***	0.03	36346.7	*46563.1*	0.03	33562.9	47652	0.05
	9eil51-3x3	2232.0	***2664.8***	0.02	2219.6	*2698.3*	0.03	2260.7	2718.1	0.03

Table 15 continued

	Instances	Prüfer Code			Dandelion Code			Blob Code		
		BF	Avg	Time	BF	Avg	Time	BF	Avg	Time
CluMRCT	12eil51-3x4	49484.7	***62777.3***	0.03	54579.6	*64792.1*	0.03	52817.0	67518.3	0.03
	12pr76-3x4	26326404.7	***31141001.1***	0.03	28919786.3	33431600.9	0.07	24628212.0	31514378	0.05
	15pr76-3x5	33420379.8	42116143.5	0.03	34817555.1	***41845723.8***	0.07	34809392.6	43097842	0.05
	16eil76-4x4	157654.8	*196131.1*	0.03	156072.8	198750.1	0.05	150217.8	***194484.1***	0.05
	16st70-4x4	167567.1	***232538***	0.07	174256.0	*235853.1*	0.08	205818.5	239904.2	0.08
	20eil51-4x5	59591.2	*89018.8*	0.03	71202.0	93139.4	0.07	66925.9	***85378***	0.05
	20st70-4x5	183633.7	***260185***	0.05	203220.6	275138	0.07	217563.2	274436.4	0.05
	25eil51-5x5	83194.3	107479.5	0.05	72078.3	104534.7	0.07	75270.7	***100790.1***	0.07
	25rat99-5x5	683775.3	***877228.6***	0.05	681391.6	*904444.6*	0.07	634651.2	932973.7	0.07
	30kroB100-5x6	15722376.8	22045120.6	0.07	15506618.6	***20385294.4***	0.07	15601002.1	*20400329.2*	0.12
	36eil101-6x6	413991.1	590240.3	0.07	462753.3	***576625.8***	0.08	435198.2	611359.3	0.13
	42rat99-6x7	978151.6	1381000.5	0.07	943034.6	***1272388.1***	0.07	1011099.0	*1310773.6*	0.10
	4eil76-2x2	132184.5	*151324.7*	0.02	130475.7	152563.8	0.03	127217.3	***148938.6***	0.05
	6berlin52-2x3	1031308.7	*1128051.5*	0.02	1018408.3	1154612.3	0.03	1020152.7	***1115266.8***	0.05
	6st70-2x3	181125.4	***202159.3***	0.03	176641.0	*202886.1*	0.03	177450.3	204943.5	0.05
	9eil101-3x3	262459.3	***319620.2***	0.03	293009.2	337025.8	0.03	273240.1	328933.3	0.05
	9eil76-3x3	142909.2	***163513.1***	0.02	144334.4	163773.9	0.03	141693.7	*161441.4*	0.03

Unlike the CluSPT, with the InterCluMRCT problem, the Dandelion Code achieves worst performance while the results obtained by Prüfer Code is the best on instances of Type 6. However, the difference between results obtained by Prüfer Code and Blob Code are not large since Prüfer Code outperforms Blob Code on 9 out of 17 instances. On instances of Type 1 and Type 5, the Blob Code exceeds two remaining ones.

Comparisons among three different encoding techniques on instances of Type 1, Type 5 and Type 6 are shown in Fig. 9. In this figure (also similar mean for other figures in this subsection), the symbol “>>” denotes “outperform”, i.e. “Prüfer Code >> Dandelion Code” means that results obtained by CC-MFEA using Prüfer Code outperform those using Dandelion Code”.

B. The influence of the number of clusters

To analyze the relationship between the number of clusters and the results obtained by different types of Cayley codes on both CluSPT and InterCluMRCT problems, we created scattered plots of results obtained by those types according to the number of clusters and problems instances as shown in Figs. 10, 11 and 12.

Figure 10 illustrates the comparison between the results obtained by Prüfer Code and Dandelion Code. The triangular symbols indicate those instances where Dandelion Code outperformed Prüfer Code. As can be seen from Fig. 10b, since all of the symbols on the right of the vertical red line are triangle, in the InterCluMRCT, Dandelion Code produces better results than Prüfer Code on all instances with the number of clusters above 25. In Fig. 10a, most of the symbols on the right of the vertical red line are triangular symbols. Therefore, it can be inferred that in the CluSPT, Dandelion Code also outperforms Prüfer Code on most instances with over 20 clusters.

Figure 11 shows the comparison between Dandelion and Blob Code. In the CluSPT (Fig. 11a), Blob Code exceeds Dandelion Code on most instances with the number of clusters under 7 (the left of vertical red line) or in a range from 12 to 25 (between the vertical green and blue line). In the InterCluMRCT (Fig. 11b), Blob Code performs not as well as Dandelion Code on all instances with the number of clusters above 25.

The comparison between Blob and Prüfer Codes is depicted in Fig. 12. In the CluSPT (Fig. 12a), Blob Code outperforms Prüfer Code on most instances with over 12 clusters (on the right of the vertical red line). In the InterCluMRCT (Fig. 12b), the results obtained by Blob Code are not as good as Prüfer Code on most instances with the number of clusters between 9 and 25 (between two vertical lines in the figure).

C. The influence of the number of vertices

Figures 13, 14 and 15 depict scattered plots of results obtained by those types according to the average number of vertices and problems instances.

A notable point in Fig. 15b is that in the InterCluMRCT, Blob Code outperforms Prüfer Code on all instances with the average number of vertices under 2.5 or over 12.

In comparison to Dandelion Code, Blob Code exceeds it on all instances with the average number of vertices above 12.8 in the CluSPT (see Fig. 14a) while in the InterCluMRCT, Blob Code outperforms Dandelion Code on most instances with the average number of vertices above 4.7 (see Fig. 14b).

In comparison between Blob Code and Prüfer Code, in the CluSPT, Blob Code exceeds Prüfer Code on all instances with the average number of vertices above 12.7 (see Fig. 15a). A remarkable point in the InterCluMRCT is that performance of Blob Code is better than that of Prüfer Code on instances which have the smallest or the largest average number of vertices, i.e. those instances with the average of under 2.5 or above 12 vertices (see Fig. 15b).

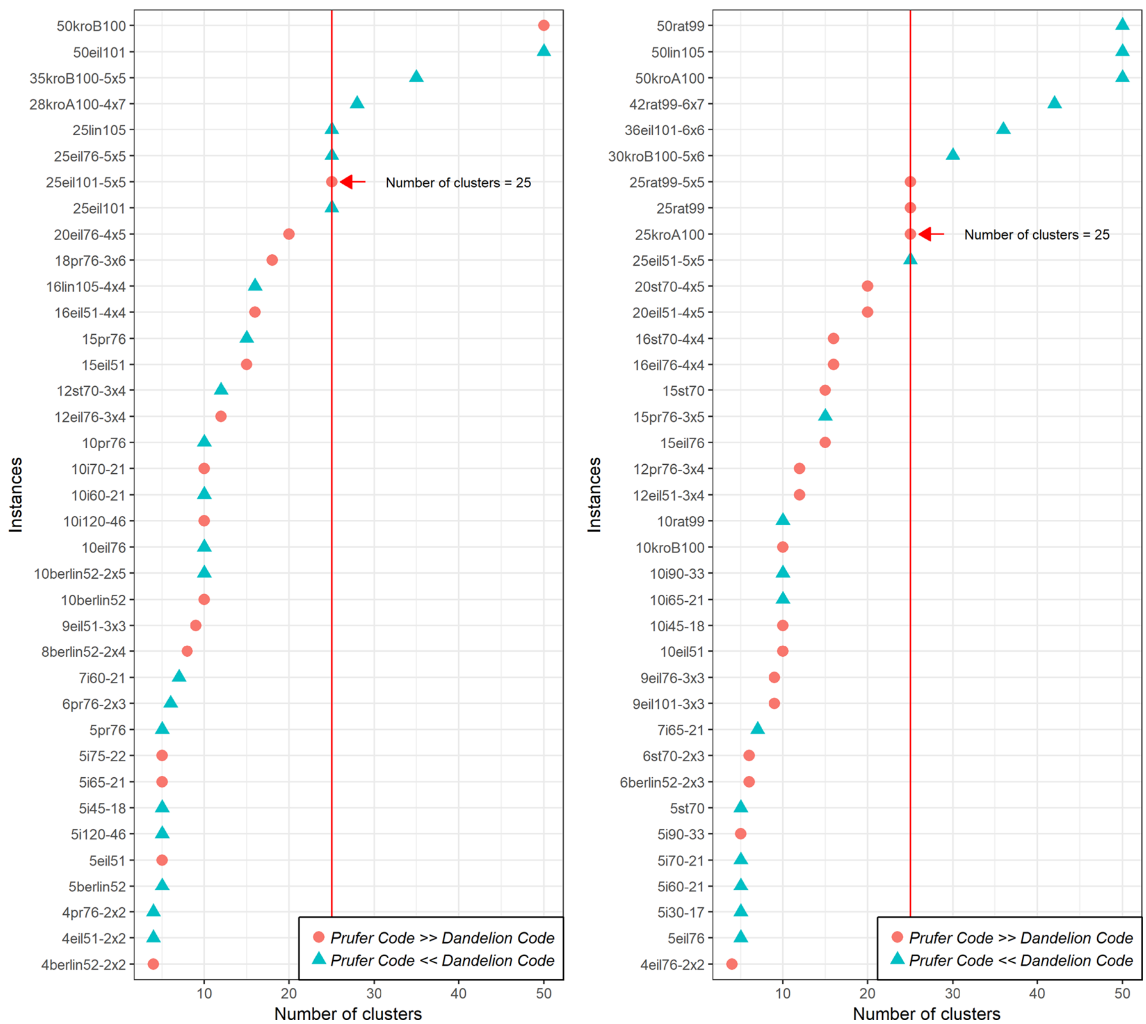
6.3.4 Comparison of the performances of Single-tasking and Multi-tasking

A. Results obtained by Single-tasking and Multi-tasking

The summary of comparison between results obtained by single-task and multi-task is presented in Table 16 in which the number of instances that multi-task outperforms single-task is presented. The results show that multi-task is better than single-task for most of the test cases on both the CluSPT and the InterCluMRCT problems. The reason behind this is that, due to the similarity of the solution structures of the CluSPT problem and the InterCluMRCT problem, when encoded in the USS, these problems support each other in the process of finding the optimal solutions. In particular, with the solution representation mechanism proposed in the paper, larger tasks contain part of the solution of smaller tasks. Therefore, genes having satisfied values in the number of dimensions of the problems construct part of the shared gene structure. These shared genes are reused among tasks leading to better interaction among them. More specifically, if the shared gene structure contains many gene segments that have good characteristics for the problems then the process of exchanging genetic material is the effectiveness when properties of the input data of two tasks are similar.

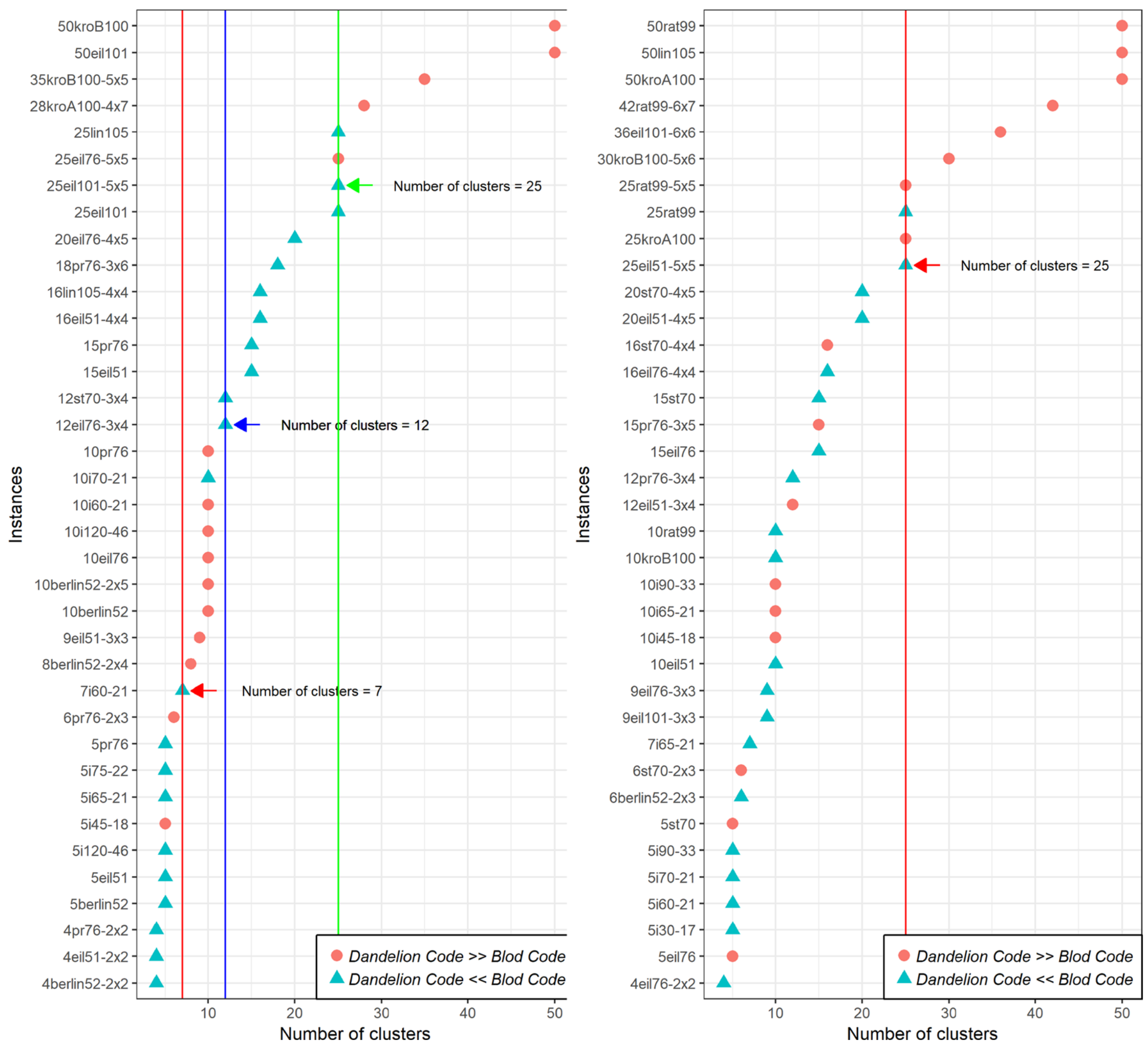
A notable point in the results shown on the Table 16 is that in Type 5, single-task exceeds multi-task on only one instance in Blob code. For instances of Type 6, the results are slightly similar to Type 5 when multi-task outperforms single-task on most of the test cases. However, the gap between results obtained by those algorithms is not as large as in Type 5, i.e. In the InterCluMRCT, multi-task surpasses single-task on 11 out of 17 instances (for the Dandelion code), on 12 out of 17 instances (for the Blob code and the Prüfer Code). In the CluSPT, multi-task also exceeds single-task on two encoding techniques which are Dandelion code and Blob code on 9 out of 17 instances but multi-task performs not as

Fig. 9 Comparisons among three different encoding techniques on instances of each Type



(a) CluSPT problem **(b)** InterCluMRCT problem

Fig. 10 Comparison between Prüfer and Dandelion Code on the scatter of Instances and Number of clusters



(a) CluSPT problem **(b)** InterCluMRCT problem

Fig. 11 Comparison between Dandelion and Blob Code on the scatter of Instances and Number of clusters

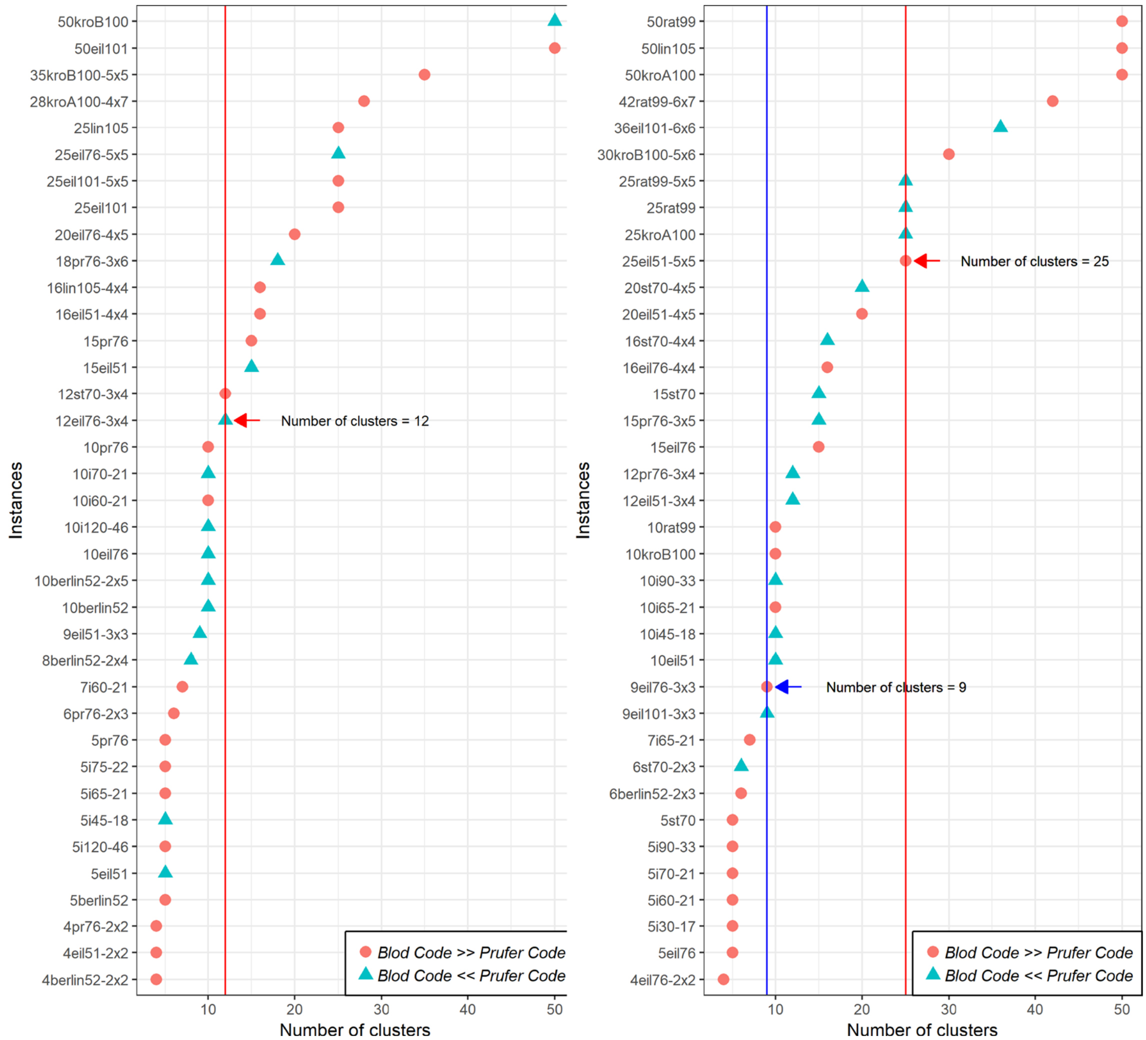
Table 16 Summary of the number of instances that multitasking outperforms singletasking on each Type

	CluSPT			InterCluMRCT			Num.Ins
	Dandelion Code	Blob Code	Prüfer Code	Dandelion Code	Blob Code	Prüfer Code	
Type 1	5	6	6	7	7	7	12
Type 5	8	8	8	8	7	8	8
Type 6	9	9	7	11	12	12	17

Num.Ins: Number of instances in a Type

well as single-task on the Prüfer Code when it is only better on 7 out of 17 instances. Last but not least, for instances of type 1, although multi-task is still superior to single-task, the difference is not as significant as in the cases of Type 5 and Type 6. For the InterCluMRCT problem, multi-task achieves better results than single-task in 7 out of 12 instances for all 3 encodings. For the CluSPT problem, with the total number of 12 instances, each of single-task and multi-tasks has 6 better cases with the encoding types of Blob code and Prüfer code. With the Dandelion code, single-task outperforms multi-task in 7 out of 12 instances.

In the InterCluMRCT, the results in all 3 Types are slightly similar where multi-task exceeds single-task in all test cases. More specifically, in three encoding techniques, there are 7 out of 12 instances on which multi-task outperforms single-task. However, in the CluSPT, unlike Type 5 and Type 6, in



(a) CluSPT problem **(b)** InterCluMRCT problem

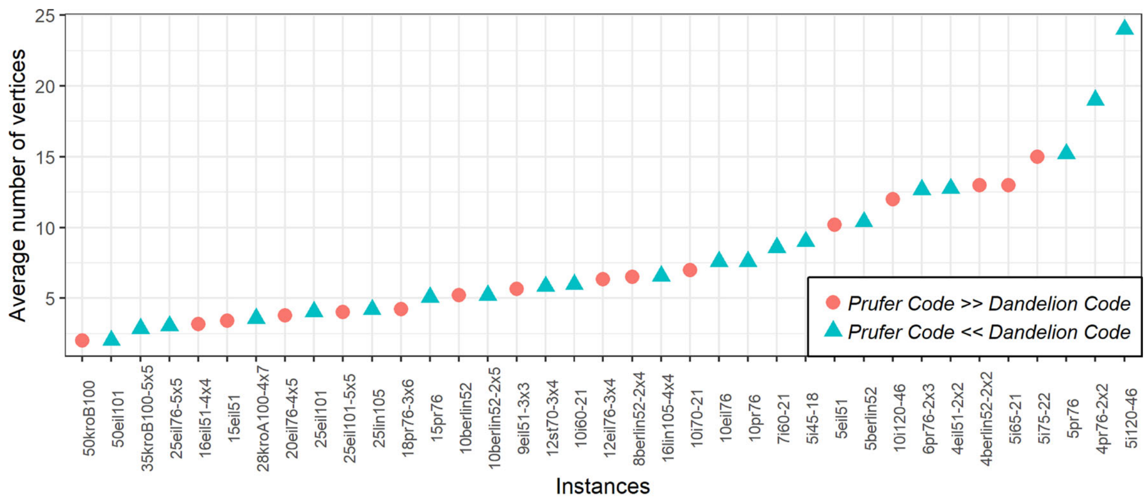
Fig. 12 Comparison between Blob and Prüfer Code on the scatter of Instances and Number of clusters

Type 1, single-task using Prüfer Code is slightly better than the multi-task on 5 out of 12 instances. From the comparison between single task and CC-MFEA, we find that CC-MFEA may not always be beneficial due to the possibility of negative knowledge transfer. Negative knowledge transfer occurs when the genotype of two optimal solutions of CluSPT and InterCluMRCT are too different from each other. The solutions of CluSPT and InterCluMRCT are encoded into an individual in the shared representation. For each generation, each task adjusts and exchanges its valuable genes in shared representation to converge toward its optimal solutions. In particular case of the optimal solutions corresponding to tasks converged, if genotypes are too different from other tasks, incompatible partial solutions transferred among tasks could lead to bad results. As a result, the combination of unrelated problems in CC-MFEA only expands search space without obtaining better solutions. Thus, single task can obtain better solutions than multitasking in those cases. In this experiment, those CluSPT and InterCluMRCT instances with close number of clusters are selected in the same type of instances to solve simultaneously. However, inferring the underlying similarity between tasks is extremely challenging in the case of combinatorial optimization like CluSPT and InterCluMRCT.

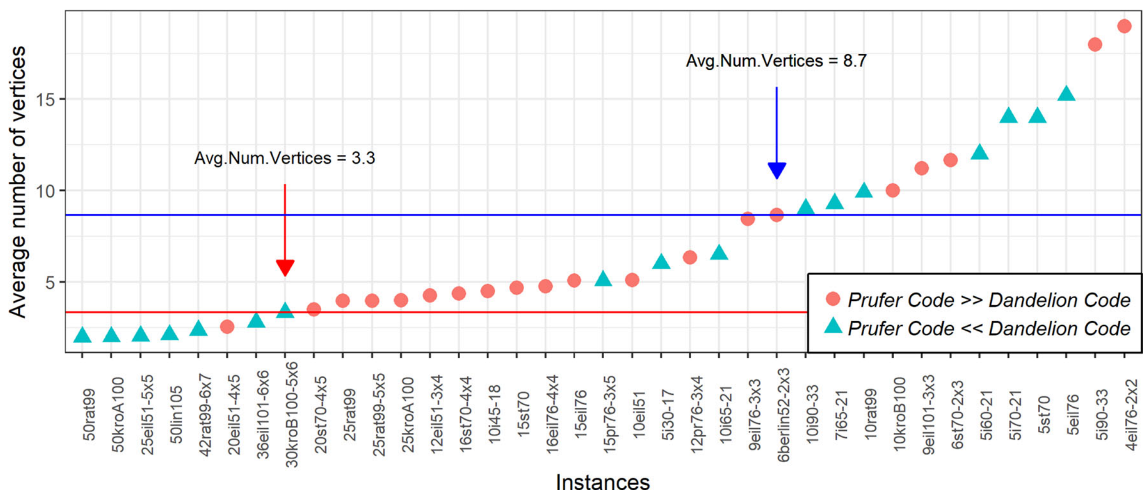
B. Convergence trends

To analyze convergence performance of single task and multi-task, a methodology based the use of Page's trend test [7] is selected. The results obtained by proposed algorithms on Dandelion code of all instances in Type 1, Type 5 and Type 6 are selected for analyzing the convergence trend. The number of cut-points has been fixed at 10, one after each 10% of fitness function evaluations.

Table 17 presents the sum of ranks at cut-points while Table 18 shows the comparison of the convergence of the multi-task (MT) and single task (ST) on Types 1, 5 and 6.



(a) CluSPT problem

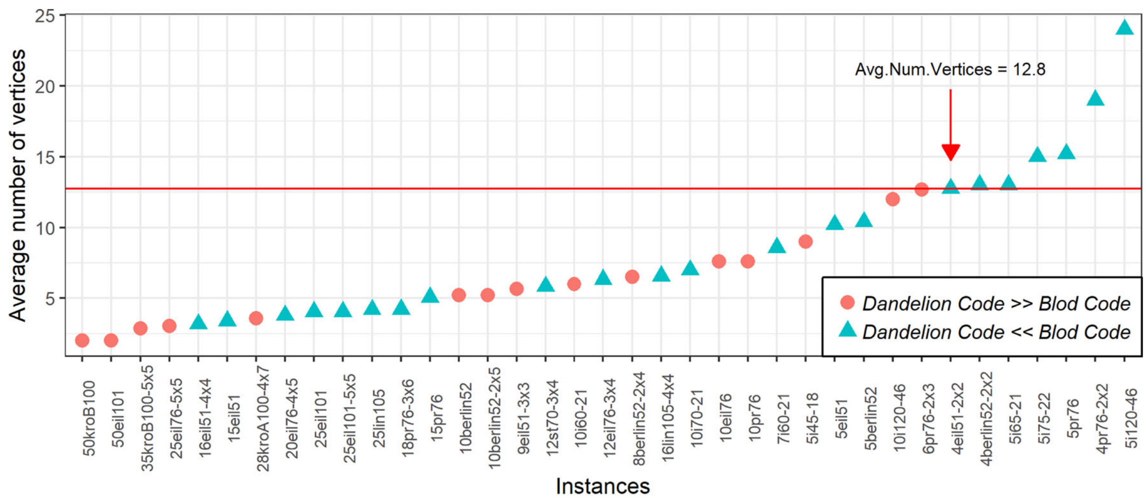


(b) InterCluMRCT problem

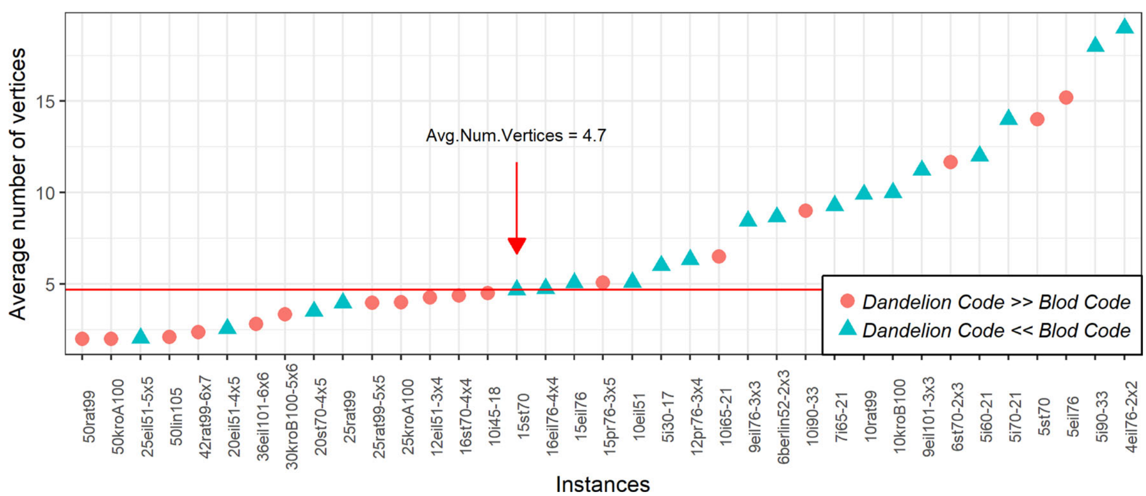
Fig. 13 Comparison between Prüfer and Dandelion Code on the scatter of Instances and Number of vertices

Table 17 Sum of ranks for the experiments on Dandelion code

Algorithms	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9	C_{10}
Type 1										
MT–ST	100	49	27	21	28	36	54	69	78	88
ST–MT	10	61	83	89	82	74	56	41	32	22
Type 5										
MT–ST	142	83	47	43	54	72	89	107	116	127
ST–MT	34	93	129	133	122	104	87	69	60	49
Type 6										
MT–ST	120	48	27	29	46	54	66	80	92	98
ST–MT	12	84	105	103	86	78	66	52	40	34



(a) CluSPT problem



(b) InterCluMRCT problem

Fig. 14 Comparison between Dandelion and Blob Code on the scatter of Instances and Number of vertices

Table 18 Convergence results (p values) for the experiments on Dandelion code

	Type 1		Type 5		Type 6	
	MT–EA	ST–MT	MT–ST	ST–MT	MT–ST	EA–MT
L Statistic	3231	2819	5116	4564	3877	3383
p value	0.009	0.991	0.006	0.994	0.0048	0.995

Table 18 points out that the comparison MT–ST shows an increasing trend in the ranks which is confirmed by a very low p value. Moreover, the opposite comparison ST–MT, shows clearly that the ranks are not increasing, which is rejected by a p value near to 1.0. These results show that the algorithm MT is converging faster than algorithm ST.

We use the functions in Gupta et al. [11] for computing the normalized objectives and averaged normalized objectives and analyzing the convergence trends of the proposed algorithms.

Figure 16 depicts the convergence trends during the initial stages of the multi-task for instances 5eil51 (the input of the CluSPT) and 5eil76 (as the input of the InterCluMRCT) in Type 1; instances 5i45-18 (as the input of the CluSPT) and 5i60-21 (as the input of the InterCluMRCT) in Type 5; instances 4berlin52-2x2 (as the input of the CluSPT) and 4eil51-2x2 (as the input of the InterCluMRCT) in Type 6. This figure illustrates the main convergence trends when comparing single task and multi-task.

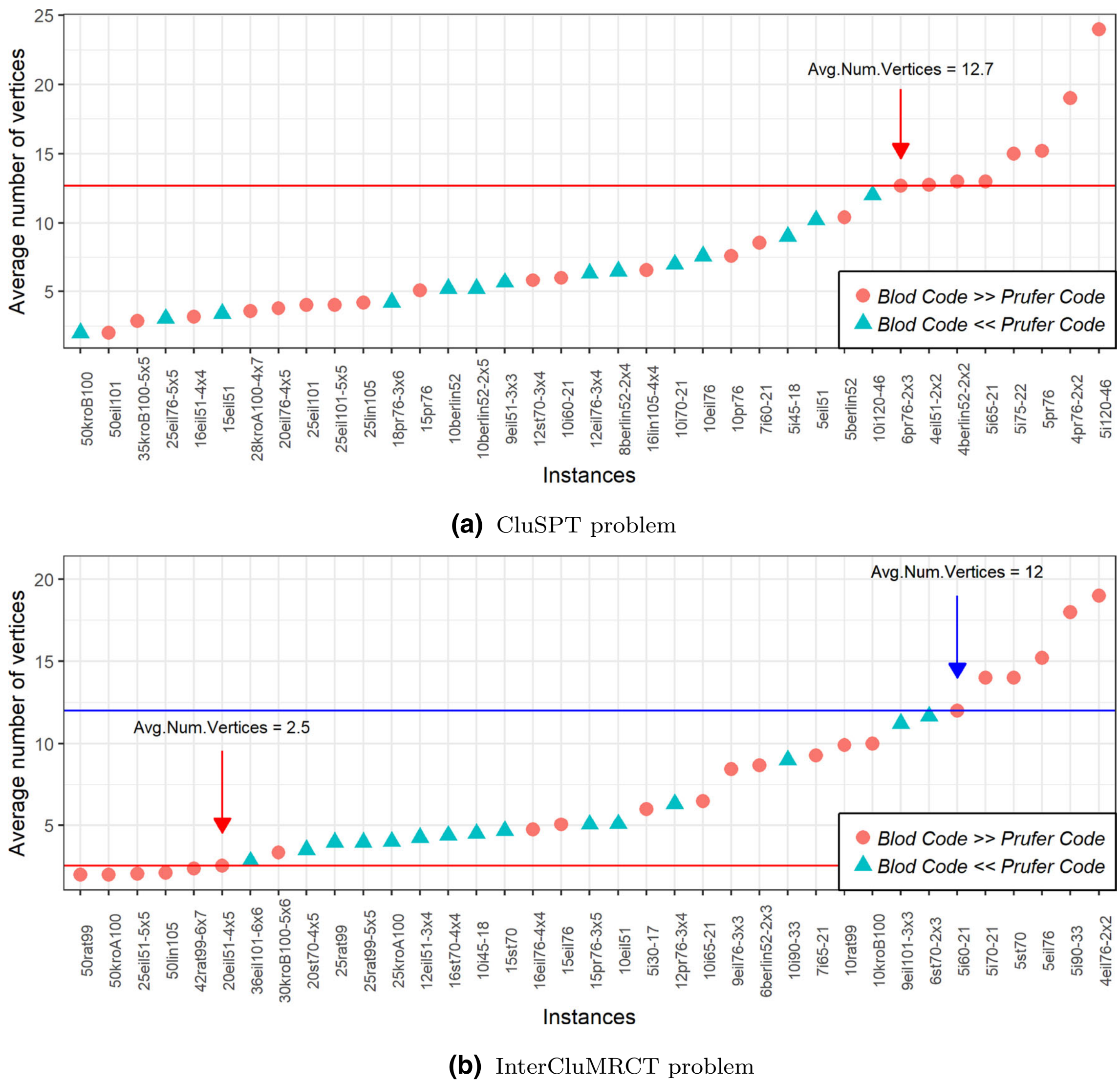


Fig. 15 Comparison between Blob and Prüfer Code on the scatter of Instances and Number of clusters

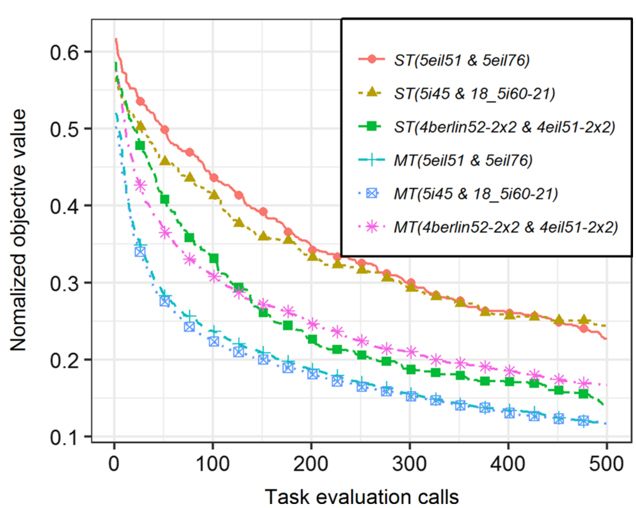


Fig. 16 Convergence trends of $\tilde{f}$ in multi-tasks and serial single task for instances 5eil5 and 5eil76 in Type 1; instances 5i45-18 and 5i60-21 in Type 5; instances 4berlin52-2x2 and 4eil51-2x2 in Type 6

– The first trend of convergence: multi-task converges faster than single task. This case is shown in Fig. 16 which corresponds to the convergence curves of Type 1 and Type 5 instances.
– The second trend of convergence: single task has a faster convergence speed than multi-task. This case is illustrated by the convergence curves of instances of Type 6. Note that, not all of cases when single task converges faster than multi-task are similar to the case of convergence curves of Type 6 instances in Fig. 16. Since initially, the convergence rate of single task is slower in early generations; however, in later generations, the convergence curve of single task converges faster than multi-task.

Refer to Fig. 17 for a better understanding of the improved performance as a consequence of multi-task. The figure

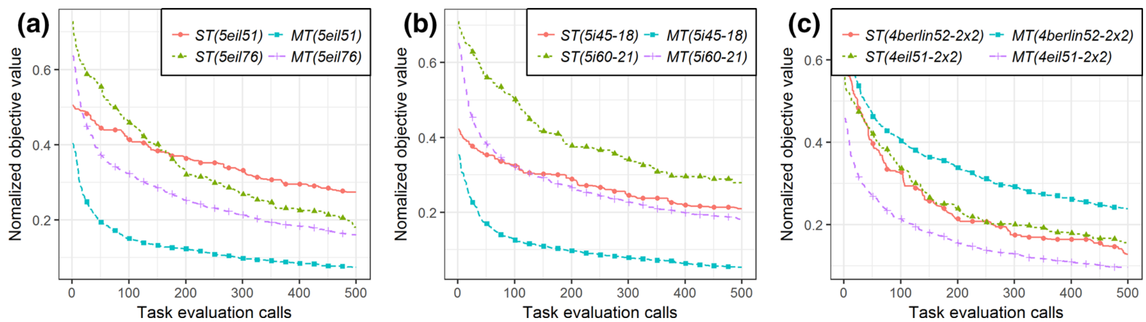


Fig. 17 Comparing convergence trends of $\tilde{f}_1$ and $\tilde{f}_2$ in multi-tasks and serial single task for instances 5eil5 and 5eil76 in Type 1; instances 5i45-18 and 5i60-21 in Type 5; instances 4berlin52-2x2 and 4eil51-2x2 in Type 6

depicts the convergence trends corresponding to each individual task.

One remark in Fig. 17 is the normalized objective values difference between single-task and multi-task at start time. The cause of this difference is that the initial individual of a task obtained by decoding from unified search space may differ from initial solution in single-task.

According to the Fig. 17c, multi-task is compared with single-task, the convergence speed on instance 4Berlin54-2x2 is slightly slower. The explanation for this phenomenon is that the evolutionary multitasking does not necessarily guarantee the improvement of performance for every task in MFEA [2,27]; some tasks are positively impacted, while other tasks may be negatively impacted by the implicit genetic transfer available during multitasking. In contrast, as shown in Fig. 17a, b, the convergence rate of each task in CC-MFEA is better than the corresponding task in single-task. Because the multitasking and the single-task use the same evolutionary operators, the improvement can be attributed entirely to the exploitation of multiple function landscapes via implicit genetic transfer, as it is afforded by the evolutionary multitasking paradigm.

7 Conclusion

This paper introduces an algorithm based on MFEA with the Cayley Code encoding mechanism to solve clustered tree problems. Evolutionary operators are proposed to exploit the advantages of the Cayley code. Those operators are applied to find solutions to the clustered problems in two nested levels: the first level constructs the tree which spans all clusters, while the second level builds the spanning tree in each cluster. The performance of the proposed algorithm is analyzed on variant types of problem instances.

This paper focuses on analyzing the effectiveness of different encoding types in the Cayley Codes (including Dandelion Code, Prüfer Code and Blob Code) when applying the CC-MFEA algorithm to solve 2 problems: the CluSPT and the InterCluMRCT problem. Experimental results show that the Dandelion Code has superior performance over the remaining encoding types, while the Prüfer Code has the lowest efficiency. The dependence of the encoding performance on some factors such as: the number of clusters, the average number of vertices in a cluster is analyzed. The obtained results also indicate that the selection of problems with similar properties of the data and solution structure help them to find better solutions.

In the near future, we are going to continue researching on MFEA in order to resolve the clustered tree problems with a variety of different representation methods, especially the representation of an arbitrary graph.

Acknowledgements This research was sponsored by the U.S. Army Combat Capabilities Development Command (CCDC) Pacific and CCDC Army Research Laboratory (ARL) under Contract Number W90GQZ-93290007. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the CCDC Pacific and CCDC ARL and the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

Ethical approval This article does not contain any studies with human participants or animals performed by any of the authors.

Informed consent Informed consent was obtained from all individual participants included in the study.

References

1. Back T (1996) Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms. Oxford University Press, Oxford

2. Bali KK, Ong YS, Gupta A, Tan PS (2019) Multifactorial evolutionary algorithm with online transfer parameter estimation: Mfea-II. IEEE Trans Evol Comput 24:69–83
3. Binh HTT, Thanh PD, Trung TB, Thao LP (2018) Effective multifactorial evolutionary algorithm for solving the cluster shortest path tree problem. In: 2018 IEEE congress on evolutionary computation (CEC). IEEE, pp 819–826
4. Binh HTT, Thanh PD, Thang TB (2019) New approach to solving the clustered shortest-path tree problem based on reducing the search space of evolutionary algorithm. Knowl Based Syst 180:12–25
5. Chandra R, Gupta A, Ong YS, Goh CK (2016) Evolutionary multitask learning for modular training of feedforward neural networks. In: International conference on neural information processing, Springer, pp 37–46
6. Da B, Gupta A, Ong YS, Feng L (2016) Evolutionary multitasking across single and multi-objective formulations for improved problem solving. In: 2016 IEEE congress on evolutionary computation (CEC), IEEE, pp 1695–1701
7. Derrac J, García S, Hui S, Suganthan PN, Herrera F (2014) Analyzing convergence performance of evolutionary algorithms: a statistical approach. Inf Sci 289:41–58
8. D'Emidio M, Forlizzi L, Frigioni D, Leucci S, Proietti G (2019) Hardness, approximability, and fixed-parameter tractability of the clustered shortest-path tree problem. J Comb Optim 38:165–184
9. Franz R (2006) Representations for genetic and evolutionary algorithms. Springer, Berlin
10. Gupta A, Mańdziuk J, Ong YS (2015) Evolutionary multitasking in bi-level optimization. Complex Intell Syst 1(1–4):83–95
11. Gupta A, Ong YS, Feng L (2016a) Multifactorial evolution: toward evolutionary multitasking. IEEE Trans Evol Comput 20(3):343–357
12. Gupta A, Ong YS, Feng L, Tan KC (2016b) Multiobjective multifactorial optimization in evolutionary multitasking. IEEE Trans Cybern 47:1652–1665
13. Julstrom BA (2005) The blob code is competitive with edge-sets in genetic algorithms for the minimum routing cost spanning tree problem. In: Proceedings of the 7th annual conference on Genetic and evolutionary computation, ACM, pp 585–590
14. Lin CW, Wu BY (2017) On the minimum routing cost clustered tree problem. J Comb Optim 33(3):1106–1121
15. Mestria M, Ochi LS, de Lima Martins S (2013) GRASP with path relinking for the symmetric euclidean clustered traveling salesman problem. Comput Oper Res 40(12):3218–3229
16. Myung YS, Lee CH, Tcha DW (1995) On the generalized minimum spanning tree problem. Networks 26(4):231–241
17. Ong YS, Gupta A (2016) Evolutionary multitasking: a computer science view of cognitive multitasking. Cogn Comput 8(2):125–142
18. Palmer C, Kershenbaum A (1994) Representing trees in genetic algorithms. IEEE, Orlando, FL, USA, pp 379–384. https://doi.org/10.1109/ICEC.1994.349921, http://ieeexplore.ieee.org/document/349921/
19. Perfecto C, Bilbao MN, Del Ser J, Ferro A, Salcedo-Sanz S (2016) Dandelion-encoded harmony search heuristics for opportunistic traffic offloading in synthetically modeled mobile networks. In: Harmony search algorithm, Springer, pp 133–145
20. Raidl GR, Julstrom BA (2003) Edge sets: an effective evolutionary coding of spanning trees. IEEE Trans Evol Comput 7(3):225–239
21. Reinelt G (1991) TSPLIB—A traveling salesman problem library. ORSA J Comput 3(4):376–384
22. Thanh PD (2019) CluSPT instances. Mendeley Data v3. https://doi.org/10.17632/b4gcgybvt6.3
23. Thanh PD, Dung DA, Tien TN, Binh HTT (2018) An effective representation scheme in multifactorial evolutionary algorithm for solving cluster shortest-path tree problem. In: 2018 IEEE congress on evolutionary computation (CEC), IEEE, pp 811–818
24. Thompson E, Paulden T, Smith DK (2007) The dandelion code: a new coding of spanning trees for genetic algorithms. IEEE Trans Evol Comput 11(1):91–100
25. Wu BY, Lin CW (2014) Clustered trees with minimum inter-cluster distance. In: 2014 IEEE 17th International conference on computational science and engineering (CSE), IEEE, pp 1138–1141
26. Wu BY, Lin CW (2015) On the clustered Steiner tree problem. J Comb Optim 30(2):370–386
27. Yuan Y, Ong YS, Gupta A, Tan PS, Xu H (2016) (2016) Evolutionary multitasking in permutation-based combinatorial optimization problems: Realization with tsp, qap, lop, and jsp. In: Region 10 conference (TENCON). IEEE, IEEE, pp 3157–3164