J Heuristics (2017) 23:31–51
DOI 10.1007/s10732-017-9323-3

# Network design through forests with degree- and role-constrained minimum spanning trees

**Laura Anton-Sanchez**[1] · **Concha Bielza**[1] · **Pedro Larrañaga**[1]

Received: 18 November 2015 / Revised: 21 October 2016 / Accepted: 11 January 2017 /
Published online: 25 January 2017
© Springer Science+Business Media New York 2017

**Abstract** Finding the degree-constrained minimum spanning tree (DCMST) of a graph is a widely studied NP-hard problem. One of its most important applications is network design. Here we deal with a new variant of the DCMST problem, which consists of finding not only the degree- but also the role-constrained minimum spanning tree (DRCMST), i.e., we add constraints to restrict the role of the nodes in the tree to root, intermediate or leaf node. Furthermore, we do not limit the number of root nodes to one, thereby, generally, building a forest of DRCMSTs. The modeling of network design problems can benefit from the possibility of generating more than one tree and determining the role of the nodes in the network. We propose a novel permutation-based representation to encode these forests. In this new representation, one permutation simultaneously encodes all the trees to be built. We simulate a wide variety of DRCMST problem instances which we optimize using different evolutionary computation algorithms encoding individuals of the population using the proposed representation. To illustrate the applicability of our approach, we formulate the trans-European transport network as a DRCMST problem. In this network design, we simultaneously optimize nine transport corridors and show that it is straightforward using the proposed representation to add constraints depending on the specific characteristics of the network.

**Keywords** Degree- and role-constrained minimum spanning tree · Forest · Network design · Permutation problems · Evolutionary computation

✉ Laura Anton-Sanchez
l.anton-sanchez@upm.es

[1] Departamento de Inteligencia Artificial, Escuela Técnica Superior de Ingenieros Informáticos, Universidad Politécnica de Madrid, Madrid, Spain

## 1 Introduction

A spanning tree is a basic topological structure in network design problems such as transportation, telecommunications and distribution systems. Well-known-classical algorithms exist for building a minimum spanning tree (MST) (Kruskal 1956; Prim 1957). In practice a more realistic representation for network design is a degree-constrained minimum spanning tree (DCMST), i.e., a MST with constraints on the number of edges incident to each node. The DCMST problem can be applied in a transportation system, such as wires, pipes or canals, where the length of the connections of $m$ nodes should be minimum. The handling capacity of each node imposes a constraint on the number of edges that can be connected to that node. In communication networks, the degree constraint limits network vulnerability if a node fails. The DCMST problem could also be applied to the design of a computer network or a road network with a maximum number of roads at a crossing (Krishnamoorthy et al. 2001).

The DCMST problem is NP-hard [this can be shown by reduction of the Hamiltonian path problem, Garey and Johnson (1979)]. The problem of finding the DCMST of a graph, and particularly finding a good representation of the tree, has been widely studied in the literature. For example, Knowles et al. (2000) introduce the randomized primal method, a novel tree construction algorithm for stochastic iterative search techniques. This method builds low-cost degree-constrained trees. Krishnamoorthy et al. (2001) compare three heuristics (simulated annealing, a genetic algorithm and a method based on problem space search) and two exact algorithms (Lagrangian relaxation and branch-and-bound) for the DCMST problem. Further, they propose alternative tree representations to facilitate the genetic algorithm neighborhood searches. Raidl and Julstrom (2003) propose representing spanning trees for network design problems directly as sets of their edges. They demonstrate the usefulness of their encoding for the DCMST problem. Soak et al. (2004) develop another effective encoding method of a tree for use by black-box optimization methods.

The above representations are based on the construction of a single tree. Some more recent studies consider building a forest. This extension is not straightforward. In Delbem et al. (2004), the proposed forest representation, named node-depth encoding, is composed of the union of the encodings of all trees of the forest. The union is implemented using an array of pointers, where each pointer indicates a tree consisting of linear lists containing the tree nodes and their depths. The proposed approach is evaluated for the DCMST problem. Some years later, Delbem et al. (2012) propose a new data structure to generate and manipulate a set of spanning forests, called node-depth-degree representation. This structure improves the average running time of their previous node-depth encoding (the forest is again composed of the union of the trees). Also working with a group of trees, Czajko and Wojciechowski (2009) take a different approach. They formulate the hop- and degree-constrained minimum spanning forest problem with minimization of the number of trees (this problem is defined as part of an access network topology design).

Here we define another variation on the DCMST problem, which we call the degree- and role-constrained minimum spanning tree (DRCMST) problem. A DRCMST is a DCMST where we determine a priori the role of each node in the tree by choosing among the following: root node, intermediate node and leaf node. It may be useful

to constrain the role of the nodes in network design. In computer networking, for example, the service has to reach the leaf nodes, and these nodes are clearly different from the central processor (which has a fixed number of ports). In such a network the cost could be associated with distances between nodes or with the material costs needed to connect nodes. A DRCMST could also be useful in a business network, for example, for the design of the project staff structure, where we would differentiate between the project manager (root), middle managers (intermediate nodes) and the rest of the team who are not in charge of any other staff (leaf nodes). The cost of this problem might be associated with team member preferences for project managers.

The DRCMST problem is also NP-hard, because it contains the particular case where we determine one root node and one leaf node with the constraint that the degree of each node has to be less than or equal to two. This is equivalent to the shortest Hamiltonian path problem between two nodes. In addition, a forest rather than a single tree can be built, i.e., we do not limit the number of root nodes to one so we can solve more complex problems (e.g., we might design several computer networks and several business networks by simultaneously considering several central processors and several project managers in the above examples, respectively). In this paper, we introduce a new permutation-based representation for building forests of DRCMSTs. One permutation simultaneously encodes all the DRCMSTs in the forest. Due to problem complexity, we address a wide variety of DRCMST problem instances using evolutionary computation techniques. Individuals of the populations are encoded with the proposed representation.

The organization of this paper is as follows. Section 2 formally describes the DCMST and DRCMST problems. Section 3 introduces the proposed permutation-based representation to encode forests of DRCMSTs. This representation is used to approximate a variety of DRCMST instances using the evolutionary computation algorithms described in Sect. 4. Section 5 details the characteristics of the 30 simulated test problem instances used to compare the performance of the different evolutionary computation techniques for the problem of finding forests of DRCMSTs. Section 6 analyzes the results and compares the algorithms. Section 7 illustrates the construction of forests with DRCMSTs in a real-world application, specifically trans-European transport network design. Finally, some discussion and conclusions are provided in Sect. 8.

## 2 Problem definition

A DCMST is a minimum spanning tree where we assume that there is a degree constraint on each node such that, at node $v$, its degree value $\deg(v)$ (i.e., its number of incident edges) is at most a given value $d_v \in \mathbb{N}$. Formally, let $G = (V, E)$ be an undirected complete graph with a set of vertices (nodes) $V$ and a set of edges $E$. A spanning tree of $G$ is a subgraph $T = (V, E_T)$, $E_T \subset E$ that contains all vertices in $V$ which it connects with exactly $|V| - 1$ edges. Let $c_{uv} \geq 0$ be the cost of each edge $(u, v) \in E$, $u, v \in V$. The DCMST problem consists in finding a minimum spanning tree $T^* = (V, E_{T^*})$, $E_{T^*} \subset E$ such that

$$T^* = \underset{T}{\operatorname{argmin}} \sum_{(u,v)\in E_T} c_{uv},$$

subject to

$$\deg(v) \leq d_v \ \text{ for all } v \in V.$$

A DRCMST is a DCMST where the role of the nodes in the tree is determined a priori (by the user/expert). In a DRCMST problem, we define three subsets of nodes $R$, $I$ and $L$ for root nodes, intermediate nodes and leaf nodes, respectively, where each node $v \in V$ must belong to one and only one subset, $\{R, I, L\}$ is a partition of the set $V$ and $R \neq \emptyset$. Note that the following conditions must be met: $d_v \geq 1 \, \forall v \in R$, $d_v \geq 2 \, \forall v \in I$ and $d_v = 1 \, \forall v \in L$. If we choose only one root node ($|R| = 1$), we construct a single tree. With a higher number of roots, we build a forest of DRCMSTs.

Given an undirected complete graph $G = (V, E)$ defined as above, a forest of $G$ is a subgraph $F = (V, E_F)$, $E_F \subset E$ that contains all vertices in $V$ and consists of a spanning tree in each connected component of $F$. Given a definition of degree constraints and subsets $R$, $I$ and $L$ that satisfies the requirements specified in the previous paragraph, the DRCMST problem consists in finding a minimum forest $F^* = (V, E_{F^*})$, $E_{F^*} \subset E$ with $|R|$ connected components such that

$$F^* = \underset{F}{\operatorname{argmin}} \sum_{(u,v)\in E_F} c_{uv}, \tag{1}$$

subject to

$$\begin{aligned} &\deg(v) \leq d_v \ \text{ for all } v \in V \\ &\text{role}(v) = \textit{root} \ \text{ for all } v \in R \\ &\text{role}(v) = \textit{intermediate} \ \text{ for all } v \in I \\ &\text{role}(v) = \textit{leaf} \ \text{ for all } v \in L, \end{aligned}$$

where $\text{role}(v) \in \{root, intermediate, leaf\}$ gives the role of node $v$ in the forest.

**Proposition 1** *Given an undirected complete graph $G = (V, E)$, the subsets of nodes $R$, $I$ and $L$ such that $\{R, I, L\}$ is a partition of the set $V$, $R \neq \emptyset$, and a degree constraint for each $v \in V$ satisfying $d_v \geq 1 \, \forall v \in R$, $d_v \geq 2 \, \forall v \in I$ and $d_v = 1 \, \forall v \in L$, the DRCMST problem is feasible if and only if*

$$\sum_{v\in R} d_v + \sum_{v\in I} d_v - |I| \geq |I| + |L|. \tag{2}$$

*Proof* Note that the proof has a double implication. First, we assume that the problem is feasible, and we see that Inequality (2) holds.

$\Rightarrow$

Suppose the problem is feasible, i.e., the subgraph of $G$, $F = (V, E_F)$, consists of $|R|$ trees where the constraints of problem (1) are satisfied. Let us prove that Inequality

(2) holds when $d_v$ is replaced by $\deg(v)$ for each $v \in V$, and then it will also hold for $d_v$, $v \in V$, because $d_v \geq \deg(v)$, $\forall v \in V$. Then, we prove

$$\sum_{v \in R} \deg(v) + \sum_{v \in I} \deg(v) - |I| \geq |I| + |L| \Longleftrightarrow \sum_{v \in R} \deg(v) + \sum_{v \in I} \deg(v) \geq 2|I| + |L|$$

We add $\sum_{v \in L} \deg(v)$ to both sides of Inequality (2):

$$\sum_{v \in R} \deg(v) + \sum_{v \in I} \deg(v) + \sum_{v \in L} \deg(v) \geq 2|I| + |L| + \sum_{v \in L} \deg(v)$$

It is known that $\sum_{v \in V_G} \deg(v) = 2|E_G|$ holds for any graph $G_G = (V_G, E_G)$. Further, because $d_v = \deg(v) = 1 \, \forall v \in L$, $\sum_{v \in L} \deg(v) = |L|$ and we have

$$\sum_{v \in V} \deg(v) \geq 2|I| + |L| + |L| \Longleftrightarrow 2|E_F| \geq 2|I| + 2|L| \Longleftrightarrow |E_F| \geq |I| + |L|$$

Since $|E_T| = |V_T| - 1$ holds for every tree $T$ and our initial assumption was that the forest $F$ has $|R|$ trees $\Rightarrow |E_F| = |V| - |R|$. Then, we have that $|V| - |R| \geq |I| + |L|$ and this becomes an equality because $|V| = |R| + |I| + |L|$. Hence, because Inequality (2) is true for $\deg(v)$, $v \in V$, it is also true for $d_v$, $v \in V$, and we have proved the first part of the double implication.

Second, we assume that Inequality (2) holds and we prove that the problem is feasible.

$\Leftarrow$

If Inequality (2) is satisfied, we can build a forest satisfying the degree constraints with the following two steps. First, we incorporate into the forest a path starting at one of the roots and including all the intermediate nodes. Second, we include an edge linking each leaf to either a root node or an intermediate node. After the first step of the construction, the sum of the residual degrees is precisely $\sum_{v \in R} d_v + \sum_{v \in I} d_v - 2|I|$, which is greater or equal than $|L|$ because Inequality (2) holds, ensuring that the second step of the construction can be carried out. Hence, we have also proved this part of the implication. □

In other words, the DRCMST problem is feasible if and only if the maximum allowed number of "outputs" [left-hand side of (2)] is greater than or equal to the number of "inputs" [right-hand side of (2)]. See Fig. 1 for an infeasible example. Notice that we are working with undirected graphs so no input or output edges exist. By establishing root nodes, however, tree structure is implicitly directed since the roots (leaves) are considered to be the origins (ends) of a tree.

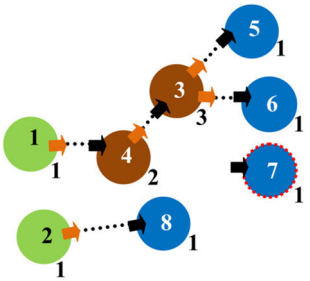



**Fig. 1** Example of an infeasible DRCMST instance. The maximum allowed degree $d_v$ is shown on the right of each node. Root nodes are shown in *green*, intermediate nodes in *brown* and leaf nodes in *blue*. Since $|R| = 2$, $|I| = 2$ and $|L| = 4$, we need six "inputs" (*black arrows*): two for intermediate nodes and four for leaf nodes. However, we only have five possible "outputs" (*orange arrows*): two from the root nodes and three from intermediate nodes. In this example, node number 7 cannot be connected to either of the two trees in this forest. This example does not satisfy Inequality (2): $2 + 5 - 2 \not\geq 2 + 4$ (Color figure online)

## 3 Problem representation

We set out to encode the DRCMST problem using a permutation representation. A permutation is understood as a vector $\boldsymbol{\sigma} = (\sigma_1, ..., \sigma_n)$ of the indices $1, ..., n$ such that $\sigma_k \neq \sigma_s$ for all $k \neq s$. We say that index $s$ is in position $k$ in $\boldsymbol{\sigma}$ when $\sigma_k = s$.

In a forest encoded by the proposed representation, all nodes have a degree $\deg(v)$ equal to their maximum allowed degree $d_v$. To enforce this constraint, we add a new type of nodes called dummy nodes, see Fig. 2. We add as many dummy nodes as are necessary to make $\deg(v) = d_v$, $\forall v \in V$. Let $D$ be the subset of dummy nodes. In a forest of DRCMSTs encoded by our representation, Inequality (2) becomes an equality:

$$\sum_{v \in R} d_v + \sum_{v \in I} d_v - |I| = |I| + |L| + |D|, \tag{3}$$

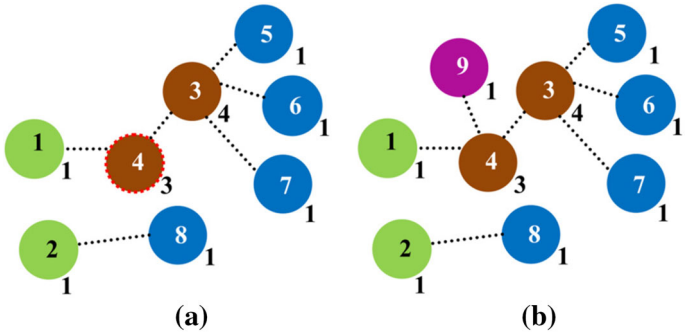



**Fig. 2** Example of a DRCMST forest with two trees. The maximum allowed degree $d_v$ is shown on the right of each node. In **a** the degree $\deg(v)$ of all nodes is equal to their maximum allowed degree $d_v$, except node number 4 where $\deg(4) = 2$ and $d_4 = 3$. To encode this forest with our permutation-based representation, we add one dummy node, node 9, connected to node 4. Forests **a**, **b** are equivalent because dummy nodes are added for representation purposes only and do not affect the calculation of tree costs (Color figure online)

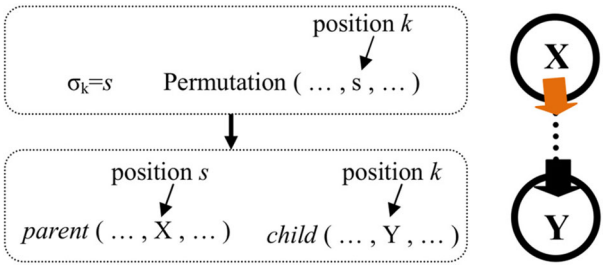



**Fig. 3** Decoding the proposed permutation-based representation. $\sigma_k = s$ represents that, in the forest, node $parent_s$ (node X) is the parent of node $child_k$ (node Y)

and hence the number of dummy nodes to be added is

$$|D| = \sum_{v \in R} d_v + \sum_{v \in I} d_v - 2|I| - |L|. \tag{4}$$

Dummy nodes are added for representation purposes only, and they are all leaf nodes so their degree is always equal to 1. The cost of every edge that reaches a dummy node is zero. Then, $m = |V| + |D| = |R| + |I| + |L| + |D|$ is the total number of nodes in the encoded forest.

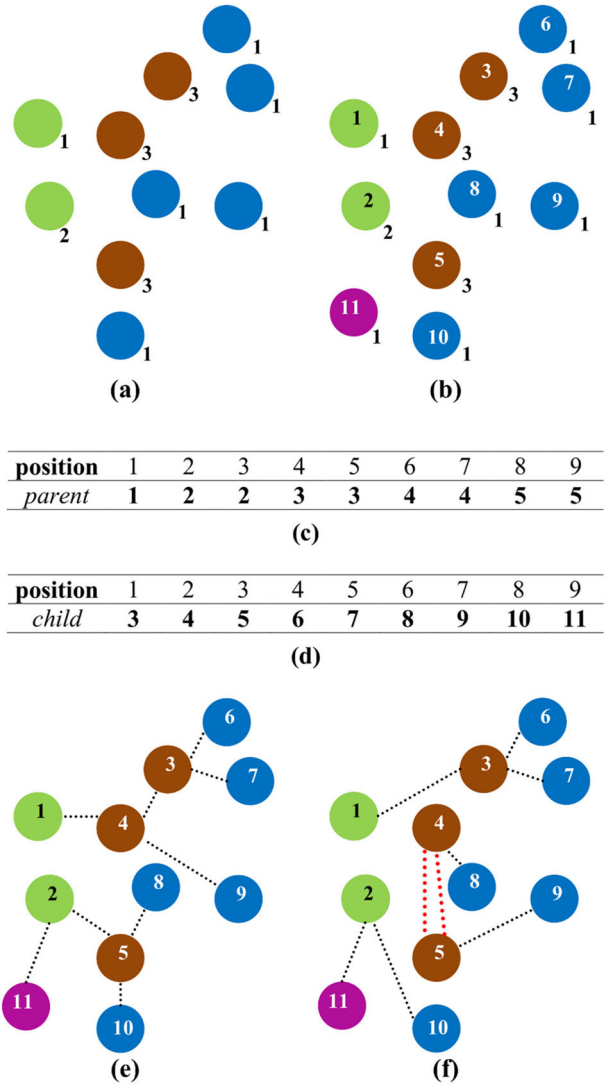
In our representation, each index of the permutation denotes a connection between two nodes, i.e., each index represents an edge in the forest. Since $|E_T| = |V_T| - 1$ holds for every tree $T$ and we encode $|R|$ trees in one permutation, the permutation length $n$ (total number of edges in the encoded forest) can be calculated as $n = m - |R|$.

The length of the permutation can also be obtained using Eq. (3):

$$n = \sum_{v \in R} d_v + \sum_{v \in I} d_v - |I| = |I| + |L| + |D|.$$

To find out which nodes are connected by the edges represented in each position of the permutation, we need two auxiliary arrays, *parent* and *child*, both of length $n$. These arrays remain unchanged for all permutations of the same problem. The *parent* auxiliary array represents the "outputs" of the edges in the forest. Since each root node has $d_v$ "outputs" and each intermediate node has $(d_v - 1)$ "outputs", each root node appears $d_v$ times for all $v \in R$ and each intermediate node appears $(d_v - 1)$ times for all $v \in I$ in the *parent* array. The *child* auxiliary array represents the "inputs" of the edges. All intermediate nodes, leaf nodes and dummy nodes have one "input", therefore the *child* array includes each node $v \in I \cup L \cup D$ once. With these arrays, our permutation is such that $\sigma_k = s$ represents that node $parent_s$ in the forest (note that we use the subscript to indicate the element in position $s$ of the auxiliary array) is the parent of node $child_k$, see Fig. 3. A simple version of this novel representation considering only binary trees was introduced in Anton-Sanchez et al. (2013).

To illustrate this representation, consider the example in Fig. 4. Figure 4a shows the $|V| = 10$ nodes of an example graph $G = (V, E)$. The maximum allowed degree $d_v$ is shown on the right-hand side of each node $v \in V$. Nodes selected as root nodes are




| position | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| *parent* | **1** | **2** | **2** | **3** | **3** | **4** | **4** | **5** | **5** |

**(c)**

| position | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| *child* | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** |

**(d)**

**Fig. 4** An example of permutation representation. **a** Nodes of an example graph. The maximum allowed degree $d_v$ is specified to the right of each node. The role of each node is indicated in different colors: root nodes in *green*, intermediate nodes in *brown* and leaf nodes in *blue*. **b** Numbered nodes. According to Eq. (4) ($|D| = 3 + 9 - 2 \cdot 3 - 5 = 1$), a dummy node is needed to solve the problem. This is added as node 11 in *pink*. **c**, **d** *parent* and *child* auxiliary arrays required to determine which forest each permutation represents. **e** Example of valid individual, permutation (6,1,2,4,5,8,7,9,3). **f** Example of invalid individual because it contains a cycle, permutation (1,8,6,4,5,7,9,2,3) (Color figure online)

shown in green, intermediate nodes in brown and leaf nodes are shown in blue, thus, $|R| = 2$, $|I| = 3$ and $|L| = 5$. This problem is feasible because it satisfies Inequality (2). We check whether it is necessary to add any dummy nodes to solve the problem using Eq. (4):

$$|D| = \sum_{v \in R} d_v + \sum_{v \in I} d_v - 2|I| - |L| = 3 + 9 - 2 \cdot 3 - 5 = 1,$$

i.e., we have to add one dummy node. Figure 4b shows the numbered nodes and the added dummy node (node number 11 in pink). Then, a forest in the example will have two trees ($|R| = 2$) with $m = |V| + |D| = 10 + 1 = 11$ nodes and it will be represented by permutations of length $n = m - |R| = 11 - 2 = 9$.

We build the *parent* and *child* auxiliary arrays both needed to encode the permutations. As indicated, we add each root node $d_v$ times ($v \in R$) and intermediate nodes ($d_v - 1$) times each ($v \in I$) to the *parent* array. A possible *parent* auxiliary array is shown in Fig. 4c. A possible *child* auxiliary array, including each intermediate, leaf and dummy node once, is shown in Fig. 4d. Note that *parent* and *child* auxiliary arrays must be established before starting to solve the problem because they determine which DRCMST problem solution each permutation represents. The order of the nodes in these arrays is in fact irrelevant.

Figure 4e represents the permutation (6,1,2,4,5,8,7,9,3) which would be a correct individual (forest) using the defined *parent* and *child* auxiliary arrays, i.e., $parent_6$ (node 4) is the parent of $child_1$ (node 3), $parent_1$ (node 1) is the parent of $child_2$ (node 4) and so on until the last position of the permutation, which indicates that $parent_3$ (node 2) is the parent of $child_9$ (node 11).

Our permutation-based representation implicitly ensures that all constraints of problem (1) are satisfied in an encoded forest. However, permutations encoding any cycle represent invalid forests. For example, Fig. 4f represents an invalid individual [permutation (1,**8**,**6**,4,5,7,9,2,3)] because it contains a cycle (in red) between nodes 4 and 5. The second position of the permutation indicates that $parent_8$ (node 5) is the parent of $child_2$ (node 4) and the next position indicates that $parent_6$ (node 4) is the parent of $child_3$ (node 5).

We can ensure that permutations corresponding to acyclic graphs are correct forests, i.e., they represent the required number of trees $|R| = m - n$ (it is trivial that if a graph $G = (V, E)$ has $m$ nodes, $n$ edges and no cycles, then it is a forest composed of $(m - n)$ trees).

Note that this representation yields several permutations encoding the same individual. For example, permutation (6,1,**3**,4,5,8,7,9,**2**) is the same individual as permutation (6,1,**2**,4,5,8,7,9,**3**) (Fig. 4e) because both $parent_3$ and $parent_2$ represent node number 2. We call these positions redundant positions, and we remove redundancy. To do this, we always choose the individual whose numbers of redundant positions are ordered from lowest to highest, i.e., (6,1,2,4,5,8,7,9,3) in the example.

Furthermore, note that cycles of length one, i.e., cycles that indicate that a node is its own parent, are very easy to detect with our representation. For example, as regards the problem illustrated in Fig. 4, we know that numbers 4 or 5 cannot occupy the first position of the permutation because this would indicate that $parent_4$ or $parent_5$, i.e., node 3, is the parent of $child_1$, also node 3. For longer cycles, we must traverse the permutation and build the trees that it encodes to identify any cycles. We read the permutation, in which each position indicates a connection between two nodes, sequentially. If the two nodes in the connection already belong to the same connected component, we will have detected a cycle. Otherwise, we join the nodes. To do this,

we use the weighted quick-union algorithm (Sedgewick and Wayne 2011). The worst-case order of growth of all operations of this algorithm is log$n$, where $n$ is the length of the permutation.

## 4 Problem solving approach

We used genetic algorithms (GAs) (Holland 1975) and estimation of distribution algorithms (EDAs) (Larrañaga and Lozano 2002) to solve and compare a variety of synthetic simulated DRCMST instances. GAs have been widely studied for solving permutation-based optimization problems (Larrañaga et al. 1999), and they are known to perform satisfactorily (Reeves 1995; Ruiz and Maroto 2005; Bielza et al. 2010). However, although several papers using probabilistic models on rankings with EDAs have recently been published (Ceberio et al. 2011, 2014, 2015; Aledo et al. 2013), EDAs have not been so extensively developed for permutation-based optimization problems (Ceberio et al. 2012). The probabilistic model learned in an EDA is expected to reflect the structure of the problem, and therefore this approach should provide an effective exploitation of promising solutions.

We opted for two GAs and two EDAs. Specifically, we used the generational genetic algorithm (gGA) (Cobb and Grefenstette 1993), the steady-state genetic algorithm (ssGA) (Syswerda 1991), the node histogram sampling algorithm (NHBSA) (Tsutsui 2006) and the Mallows kernel EDA (MKEDA) (Ceberio et al. 2015).

The gGA and the ssGA are two of the best-known families of GAs. Algorithm 1 shows the pseudo-code of a generic GA. Briefly, these algorithms evolve a population of individuals until a specified stop condition is met (line 4). The main steps are as follows: select the parents to be crossed (line 6), recombine them (line 7), and mutate the resulting children (line 8). Usually, the fittest individuals survive and the least fit individuals are discarded. In the particular case of the gGA, in each generation, two parents are selected from the whole population and recombined, generating two children that are then mutated. The resulting children are placed in an auxiliary population which will replace the current population when the auxiliary population is completely filled. In our case, the size of the auxiliary population is the same as the size of the current one. Note that this strategy could remove the best solution in the current population. By contrast, the ssGA is an elitist strategy because the best solution is always retained. In each generation of the ssGA, one of the resulting children is mutated and evaluated and then inserted back into the population if the new individual is better than the worst individual in the current population.

EDAs replace crossover and mutation operators with the estimation of probabilistic models of promising individuals and sampling from these models. Algorithm 2 shows the pseudo-code of a generic EDA. We start from an initial population of $M$ individuals (line 2). In the main loop, until the stop condition is met (line 4), a number $N$ ($N \leq M$) of individuals are selected (usually the individuals with best fitness) (line 6). Next, the probabilistic model of the selected individuals is estimated (line 7), and the new population of size $M$ is generated by sampling the learned model (line 8). With regard to the EDAs used in this paper, the NHBSA models frequencies at each absolute position in the permutation. This is closely related to performance in our problem

**Algorithm 1** Generic GA pseudo-code

```
1: t ⟵ 0
2: P(t) ⟵ GenerateInitialPopulation()
3: Evaluate(P(t))
4: while ! StopCondition() do
5:    t ⟵ t + 1
6:    P_p ⟵ SelectParents(P(t − 1))
7:    P_c ⟵ Crossover(P_p,CrossProb)
8:    Mutate(P_c, MutProb)
9:    Evaluate(P_c)
10:   P_t ⟵ BuildNextGeneration(P_c, P(t − 1))
11: end while
```

(since each number in the permutation denotes an edge selected to build the forest). The other EDA, the MKEDA, calculates as many Mallows models (distance-based exponential probability models over permutation spaces Mallows 1957) as individuals in the selected population of solutions. Particularly, we analyzed the MKEDA under the Cayley distance (Irurozki et al. 2014) because the algorithm performs better with this distance (Ceberio et al. 2015).

**Algorithm 2** Generic EDA pseudo-code

```
1: t ⟵ 0
2: P(t) ⟵ GenerateInitialPopulation(M)
3: Evaluate(P(t))
4: while ! StopCondition() do
5:    t ⟵ t + 1
6:    P_sel(t − 1) ⟵ Select(N, P(t − 1))
7:    Model(t) ⟵ EstimateModel(P_sel(t − 1))
8:    P(t) ⟵ Sample(M, Model(t))
9:    Evaluate(P(t))
10: end while
```

In order to compare the performance of these algorithms we used the jMetal framework (Durillo and Nebro 2011). jMetal stands for Metaheuristic Algorithms in Java, and it is an object-oriented Java-based framework for single and multi-objective optimization with a variety of metaheuristics techniques. jMetal already contained gGA and ssGA algorithms, and we plugged our NHBSA and MKEDA implementations into jMetal.

We made improvements to all the algorithms due to the specific characteristics of our representation. On the one hand, we ruled out the generation of individuals containing cycles of length one (which are easy to detect as described in Sect. 3) and, on the other hand, we removed the redundancy of our representation by selecting a representative individual from the redundant individuals as already explained. In order to detect cycles of length longer than one, it was necessary, as already mentioned, to traverse each permutation sequentially, building the trees that it encoded. If a cycle was identified, the individual was immediately ruled out.

For GAs, we used the default operators provided for permutations in jMetal: partially matched crossover (PMX) and swap mutation. PMX builds a child by choosing

a subsequence of one parent (permutation) using two random cut points, and it preserves the order and position of as many indices as possible from the other parent. Swap mutation selects two indices at random and swaps their positions. We set a crossover probability ($CrossProb$ in Algorithm 1) equal to 0.9 and a mutation probability ($MutProb$ in Algorithm 1) equal to $1/n$, where $n$ is the length of the permutation. For each problem, we established a population size equal to $10|V|$ for all GAs and EDAs. For each execution of each problem, the initial population was randomly generated including the improvements discussed above (length-one cycles and redundancy). We decided to stop any algorithm if there was no more than a 0.1% improvement of the best fitness over the last 500 generations.

We applied the non-parametric Friedman test to detect statistically significant differences considering the whole set of algorithms (Friedman 1937). The null hypothesis for the Friedman test states equality between all the algorithms. If the null hypothesis is rejected, a post-hoc test can be applied to find out which pairwise comparisons cause the differences. We opted for the Bergmann–Hommel procedure (Bergmann and Hommel 1988). Although computationally expensive, this is the best-performing procedure for comparing all the algorithms with one another (Derrac et al. 2011). We set a significance level of $\alpha = 0.05$. We used the implementation of the Friedman test for multiple comparison and the Bergmann–Hommel procedure provided in the MULTIPLETEST package available at the SCI2S public website.[1]

## 5 Test problem generation

We simulated five problem instances for each of the following sizes $|V|$= 20, 40, 60, 80, 100, 200 nodes. The number of roots and intermediate nodes were randomly generated, although some constraints were included. The number of problem root nodes was less than or equal to 20% of all problem nodes, i.e., $|R| \leq 0.2|V|$. The number of intermediate nodes was less than or equal to 75% of all problem nodes ($|I| \leq 0.75|V|$). The number of leaf nodes was derived as $|L| = |V| - |R| - |I|$. The maximum allowed degree of root and intermediate nodes (leaf nodes always have a degree equal to 1) was also simulated randomly for each node as follows. The maximum allowed degree $d_v$ was between 1 and 4 for root nodes and between 2 and 5 for intermediate nodes. We simulated the coordinates $(x, y, z)$ of each point between $x_{min} = y_{min} = z_{min} = 1$ and $x_{max} = y_{max} = z_{max} = 100$. Then, we computed the cost matrix with real Euclidean distances between pairs of points. A small fitness was preferred when we evaluated individuals (permutations) of our population.

Table 1 shows the characteristics of each of the simulated instances. For each instance, it lists the number of nodes of each role and the length of the permutation representing the forest. Note that the length of the permutation for each problem instance depends on the number of nodes of each role and their maximum allowed degree.

[1] http://sci2s.ugr.es/sicidm/.

**Table 1** Description of the simulated DRCMST instances. The table shows the number of root, intermediate, leaf and dummy nodes and the length of the permutations that encode the problem solutions

| Problem | $\|R\|$ Roots | $\|I\|$ Intermediate | $\|L\|$ Leaf | $\|D\|$ Dummy | $n$ Permutation length |
|---|---|---|---|---|---|
| Problem 1-20Nodes | 3 | 9 | 8 | 10 | 27 |
| Problem 2-20Nodes | 3 | 6 | 11 | 0 | 17 |
| Problem 3-20Nodes | 1 | 9 | 10 | 3 | 22 |
| Problem 4-20Nodes | 3 | 11 | 6 | 10 | 27 |
| Problem 5-20Nodes | 2 | 12 | 6 | 11 | 29 |
| Problem 1-40Nodes | 5 | 18 | 17 | 15 | 50 |
| Problem 2-40Nodes | 7 | 12 | 21 | 6 | 39 |
| Problem 3-40Nodes | 4 | 22 | 14 | 16 | 52 |
| Problem 4-40Nodes | 4 | 18 | 18 | 8 | 44 |
| Problem 5-40Nodes | 7 | 16 | 17 | 13 | 46 |
| Problem 1-60Nodes | 5 | 24 | 31 | 1 | 56 |
| Problem 2-60Nodes | 1 | 36 | 23 | 13 | 72 |
| Problem 3-60Nodes | 11 | 17 | 32 | 8 | 57 |
| Problem 4-60Nodes | 9 | 26 | 25 | 18 | 69 |
| Problem 5-60Nodes | 11 | 23 | 26 | 22 | 71 |
| Problem 1-80Nodes | 13 | 42 | 25 | 50 | 117 |
| Problem 2-80Nodes | 11 | 25 | 44 | 1 | 70 |
| Problem 3-80Nodes | 13 | 36 | 31 | 36 | 103 |
| Problem 4-80Nodes | 12 | 33 | 35 | 5 | 73 |
| Problem 5-80Nodes | 15 | 29 | 36 | 21 | 86 |
| Problem 1-100Nodes | 15 | 45 | 40 | 29 | 114 |
| Problem 2-100Nodes | 7 | 56 | 37 | 39 | 132 |
| Problem 3-100Nodes | 6 | 52 | 42 | 14 | 108 |
| Problem 4-100Nodes | 18 | 42 | 40 | 39 | 121 |
| Problem 5-100Nodes | 15 | 56 | 29 | 49 | 134 |
| Problem 1-200Nodes | 23 | 74 | 103 | 24 | 201 |
| Problem 2-200Nodes | 30 | 100 | 70 | 91 | 261 |
| Problem 3-200Nodes | 11 | 102 | 87 | 43 | 232 |
| Problem 4-200Nodes | 10 | 103 | 87 | 16 | 206 |
| Problem 5-200Nodes | 20 | 99 | 81 | 59 | 239 |

We obtained a wide variety of problem instances. The number of trees in the forest ranged from a single tree ($|R| = 1$, two times out of 30) to 30 trees (problem number 2 with 200 nodes). The number of intermediate nodes ranged from 28 to 60% of all network nodes. The number of leaf nodes ranged from 29 to 55% of $|V|$ depending on the problem. No dummy node had to be added in one of the problems (problem number 2 with 20 nodes), whereas problem number 1 with 80 nodes had 50 dummy nodes (62.5% of the problem size). On average, the permutation length of the problems was 15.6% greater than the number of nodes in the problem.

## 6 Results

We used the algorithms described in Sect. 4 to solve the 30 simulated DRCMST instances described in Sect. 5. Each problem was run 20 times with each algorithm (with a new randomly generated initial population in each run). All the results were obtained using the Magerit supercomputer. Magerit is offered by the high performance computing area at the Supercomputing and Visualization Center of Madrid (CeSViMa). Magerit is a general-purpose cluster with dual architecture (Intel and POWER). We used POWER7 nodes with 3.3 GHz (422.4 GFlops) 32 GB of RAM and 300 GB of local hard disk.

Figure 5a shows the best fitness values found by the NHBSA in the first 20 generations of a run for problem number 1 with 20 nodes. For this problem, we were interested in building a forest of three trees. Figure 5b–f shows the trees that represent the best individuals found in generations 1, 5, 10, 15 and 20. Again, the roots are represented in green, intermediate nodes are shown in brown and leaf nodes in blue. In each forest, the edges that differ from the best forest found by the algorithm are shown in red. We observe that the number of red edges gradually diminishes as the number of generations increases because the algorithm is approaching the best solution found. The forest output in generation 20 does not have any red edges because the algorithm did not improve after this generation, i.e., it provides the best fitness value found for this problem.

As described in Sect. 4, we applied the Friedman test to detect statistically significant differences among the algorithms (Friedman 1937). We applied the test two times: once on the mean best fitness found by the algorithms for the 30 problem instances and again on the mean execution time. The Friedman test rejected the null hypothesis of equality for both the fitness and execution time ($p$ value $\leq 10^{-11}$ in both cases). Once the null hypothesis of equality between all pairs of algorithms was rejected, we applied the Bergmann–Hommel procedure (Bergmann and Hommel 1988) to perform all the pairwise comparisons.

Figure 6 illustrates the results of both the Friedman test and the Bergmann–Hommel procedure. These diagrams were introduced in Demšar (2006) and neatly illustrate statistically significant differences between algorithms. The Friedman test ranks the algorithms such that the best-performing algorithm should have rank 1, the second best rank 2, etc. In the diagrams the lowest (best) ranks are to the right so the algorithms on the right-hand side can be viewed as better. Groups of algorithms that are not significantly different ($p$ value $>0.05$ in the Bergmann–Hommel procedure pairwise comparisons) are connected. Analyzing pairwise comparisons, the results showed that there were no significant differences in the best fitness for the NHBSA, the gGA and the ssGA (Fig. 6a). Looking at the execution times, however, we found significant differences between all the algorithms (Fig. 6b). Both EDAs had a longer execution time than GAs. The gGA and the ssGA had similar execution times but the hypothesis of equal mean times was rejected. We could, therefore, conclude that the ssGA was preferable because it had a better execution time.

We also wanted to compare the four heuristic algorithms with an exact method for further evaluation. The evaluation of all permutations of length $n$ for a DRCMST problem requires an execution time of order $n!$. This is unworkable even for small

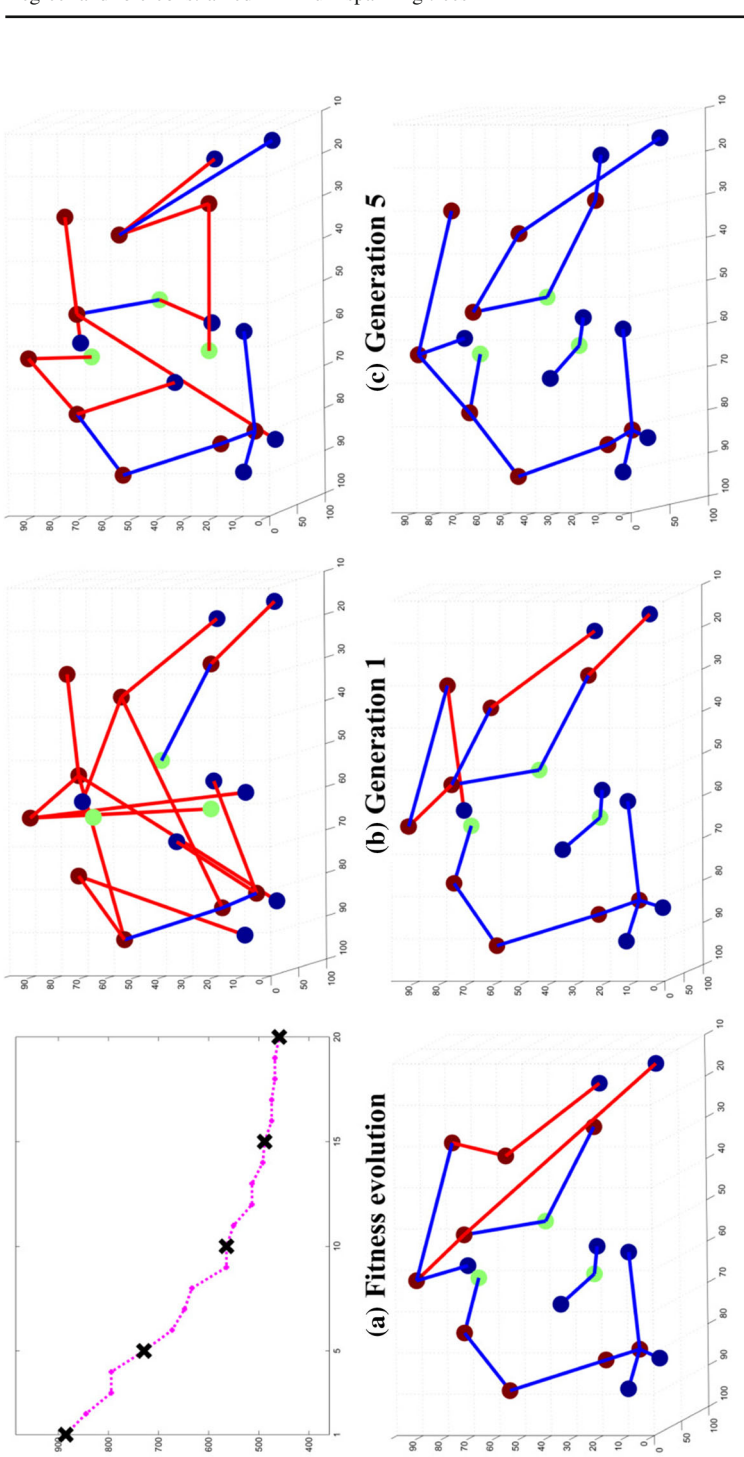



**Fig. 5** Evolution of the best fitness found in 20 generations by the NHBSA for problem number 1 with 20 nodes. **a** Fitness evolution over 20 generations [the crosses indicate the fitness of individuals shown in (**b**–**f**)]. **b**–**f** Forest encoded by the best solutions found in generations 1, 5, 10, 15 and 20. Root nodes are shown in *green*, intermediate nodes in *brown* and leaf nodes in *blue*. Edges that differ from the best forest found by the algorithm are shown in *red*. The algorithm did not improve after generation 20 (Color figure online)

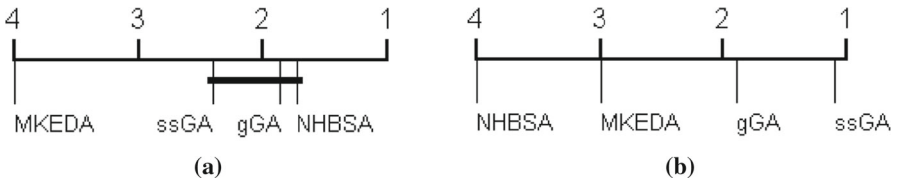



**Fig. 6** Comparison of the four algorithms using the Friedman test and the Bergmann–Hommel procedure. Groups of algorithms that are not significantly different ($p$ value $>0.05$) are connected. The lowest (best) ranks are to the right so the algorithms on the right-hand side can be viewed as better. **a** Fitness diagram. **b** Execution time diagram

values of $n$. Therefore, we implemented the following branch-and-bound method to solve small instances of DRCMST problems exactly.

We know that, in a DRCMST problem, each permutation position adds the connection cost between two points to the objective function. Moreover, we build the forest represented by a permutation, traversing the permutation and accumulating the cost of each position sequentially. Suppose that we want to solve exactly a DRCMST problem represented by permutations of length $n$. To do this, we start by generating and evaluating the permutations of length $n$ in lexicographical order. Let $x$ be the best solution found by our heuristic methods. If, in a specific permutation, we are at position $j$, such that the cost accumulated so far is greater than the $x$ cost, then we can rule out the following $(n - j)!$ permutations in lexicographical order, i.e., all permutations that have the same indices up to position $j$.

Although, using this branch-and-bound method, a lot of the permutations do not have to be evaluated, we were unable to solve exactly the instances of 20 nodes (Table 1), because it would have taken several months. Therefore, we simulated some smaller problem instances. Specifically, we simulated five problems with 10 nodes (with a permutation length equal to 10, 11 and 12) and five problems with 15 nodes (all with a permutation length equal to 13) as detailed in Sect. 5. We solved these 10 problem instances with the implemented branch-and-bound method, evaluating, on average, 23.63% of all permutations. The instances with 10 nodes were fairly easy to solve, whereas the execution time on a desktop computer for each instance with 15 nodes was of the order of several days.

We also solved these 10 problems 20 times using each of the four heuristic algorithms. This was done sequentially (10 problems $\times$ 20 times $\times$ 4 algorithms $=$ 800 runs) on the same computer as for the exact method. The execution time was just over two minutes. The gGA, ssGA and NHBSA found the global optimum in all cases, and the MKEDA found the global optimum for six out of the 10 problems (the solutions for the other four problems were 1.75% worse, on average, than the global optimum).

## 7 Trans-European transport network

We use the trans-European transport network in order to illustrate the applicability of forests with DRCMSTs in a real-world network design. This transport network,

available at the European Commission website,[2] is composed of nine corridors and comprises 138 cities. Its main purpose is to facilitate the transport of passengers and goods throughout the European Union providing for faster international long-distance travel. To illustrate the interest of our approach, we tried to design a similar network facilitating transport between the above European cities by formulating the network design as a DRCMST problem. As in the real network we built nine corridors and were interested in minimizing the total length of the transport network.

Not all the corridors of the trans-European transport network are trees in the sense that some contain cycles. To fit the design of a forest of DRCMSTs, we simplified the real network by deleting 18 connections between cities to remove cycles. Figure 7a shows the real transport corridors after removing these connections.

We used straight lines to represent connections between pairs of cities. In our simplified example, however, we considered the geodesic distance between two cities as their connection cost. The total geodesic length of the real corridors (with cycles removed) is 33,138 km (Fig. 7a).

There are 138 cities in the trans-European transport network. Given that some cities in the real network belong to several corridors, we replicated these cities as many times as the number of corridors they belong to. This strategy makes the network design more flexible. After replicating these cities, the result was a total of 204 nodes (cities) for our problem.

As regards node roles, we chose one city from each of the real corridors (located more or less in the middle of the corridors) to play the root role in order to build nine transport corridors. Specifically, the roots were Bilbao, Craiova, Frankurt, Hamburg, Katowice, London, Perpignan, Vienna and Warsaw in the examples shown in Fig. 7b–d. The cities where a corridor ended in the real network were considered leaf nodes, and the other cities were intermediate nodes.

Regarding the degree constraints of each city, we ran two different tests. First, we optimized the design of the nine corridors by defining the maximum allowed degree in each city as the real number of connections it had in the (simplified) real corridors. In this case, no dummy nodes had to be added and, since we built nine trees, the length of the permutations was $204 + 0 - 9 = 195$. We used the ssGA to solve the problems since this technique performed significantly better for DRCMST problems in Sect. 6. The best solution found for the real degree of each city is shown in Fig. 7b with a total length of 32,177 km. In this solution most of the European territory is covered by seven rather than nine transport corridors, since the orange and cyan corridors were composed of only two cities (Frankfurt and Mannheim) and three cities (Vienna, Brno and Bratislava), respectively.

Then we relaxed the degree constraints to make the network design more flexible. Specifically, we assigned a maximum number of connections equal to three to every intermediate node. In this case, we had to add 82 dummy nodes, and the permutation length needed to represent the nine corridors was $204 + 82 - 9 = 277$. Figure 7c shows one of the solutions using these degree constraints with a total length of 28,759 km. Figure 7c reveals that the corridors are divided into four groups: (1) yellow and

[2] http://ec.europa.eu/transport/themes/infrastructure/ten-t-guidelines/corridors/doc/ten-t-corridor-map-2013.

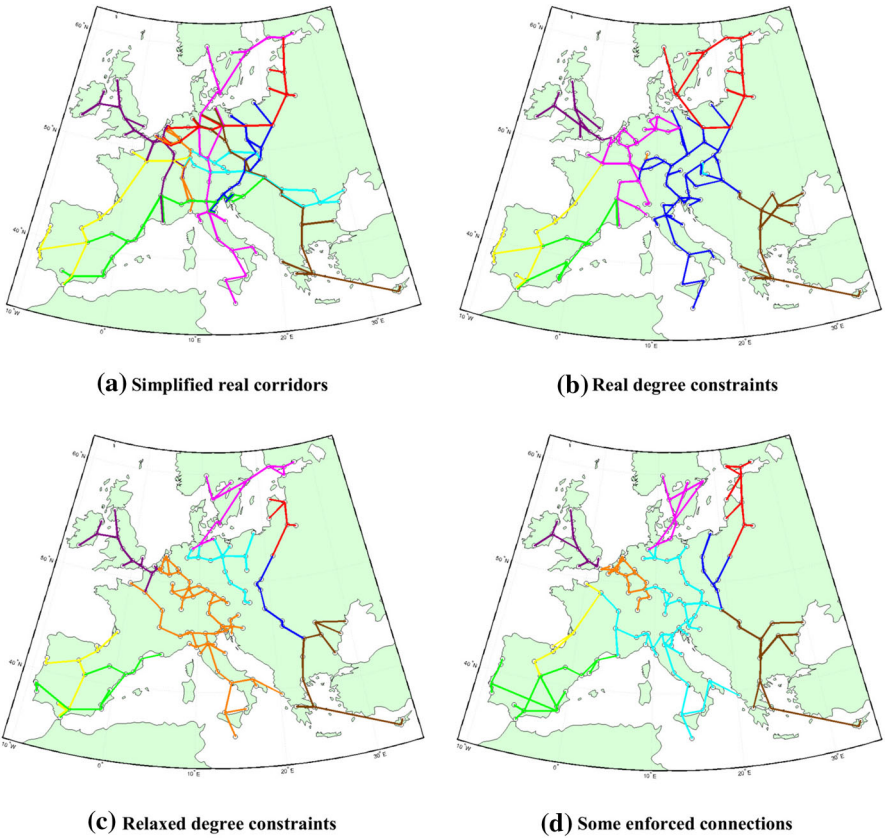



**Fig. 7** Application of forests with DRCSMTs to the nine trans-European transport network corridors. We consider the geodesic distance between two cities as their connection cost. **a** Simplified trans-European transport network where we removed 18 connections to cut out cycles in the real corridors. Total length = 33,138 km. **b**–**d** Solutions provided by the ssGA. **b** We chose one city from each corridor as the root node. In the example, the roots are Bilbao, Craiova, Frankfurt, Hamburg, Katowice, London, Perpignan, Vienna and Warsaw. The cities where a corridor ends in the real network are considered leaf nodes; the remaining nodes are intermediate nodes. We assigned a city a maximum degree equal to the number of connections it has in the simplified real corridors. Total length = 32,177 km. **c** Role constraints as in **b**. In this case, we relaxed degree constraints and allowed a maximum degree equal to 3 for all intermediate nodes. Total length = 28,759 km. **d** Degree and role constraints as in **c**. We established some compulsory connections between cities to enforce corridor interconnections, in particular, we enforced the following connections: Bordeaux–Paris, Marseille–Lyon, Vienna–Wels/Linz and Craiova–Timişoara. Total length = 31,166 km (Color figure online)

green corridors, (2) purple and orange corridors, (3) magenta and cyan corridors and, finally, (4) the remaining three corridors. The corridors in each group are connected with each other but not with other groups.

The ssGA looks for a forest, in our case with a minimum total length, where the degree and role constraints hold. We can easily add additional constraints, for example, require the solution to have specific connections between cities, using the proposed representation. Since each permutation position represents a connection between two nodes of the forest, if we want two nodes always to be connected, the permutation

should be required to have a specified number in a specified position. In an attempt to find a good solution with some interconnected corridors, we optimized the corridor design as in Fig. 7c but enforced the following four connections: Bordeaux–Paris, Marseille–Lyon, Vienna–Wels/Linz and Craiova–Timişoara. One of the solutions provided by the ssGA is shown in Fig. 7d. This solution had a total length of 31,166 km.

Besides adding constraints depending on the specific characteristics of the network design problem, the minimization criterion for building the trees could take into account more complex criteria, like the varying cost of building the connection between different cities. Furthermore, alternative distances more in line with reality than the geodesic distance could be used.

## 8 Conclusions

In this paper we have presented a novel permutation-based representation to solve a new variant of the DCMST problem, which we have called DRCMST problem. A DRCMST is a DCMST with supplementary constraints that determine the role of the nodes in the tree (root, intermediate or leaf nodes). Establishing the roles of the nodes may be useful in some problems such as network design. Most research about computing DCMST outputs a single tree. We increase the flexibility of the problem by not limiting the number of root nodes to one so, generally, we compute forests of DRCMSTs. We used metaheuristic techniques to approximate the problem solution because the DRCMST problem is NP-hard. Specifically, we opted to use two genetic algorithms (gGA and ssGA) and two estimation of distribution algorithms (NHBSA and MKEDA). Using the proposed representation, we solved a wide range of synthetic simulated DRCMST instances. The results showed that the NHBSA, the gGA and the ssGA found the best solutions, but the ssGA ran in significantly less time. Finally, we formulated the nine corridors of the trans-European transport network as a DRCMST problem and optimized it using the ssGA to illustrate the applicability and flexibility of our approach.

The main advantage of our permutation-based representation is that it can encode more than one tree simultaneously. Moreover, the degree constraint can be different for each node. Another strength is that it is simple to add constraints related to a specific problem. For example, if two nodes must (cannot) be connected in the problem statement, then a specific number will be enforced (forbidden) at a specific position of the permutation.

Probably the weakest point of the proposed representation is that it encodes invalid individuals (cycles). Cycles of length equal to one are easy to detect and thus avoid (this is the cause of the highest percentage of invalid individuals). However, the permutation must be decoded to detect the existence of cycles of length longer than one. We intend to work on improving cycle detection, which could speed up the algorithms. Furthermore, different permutations may encode the same forest. We remove this redundancy by selecting a representative individual within the set of redundant individuals.

Other aspects could be taken into account such as considering a more complete fitness evaluation function. For example, if the network is designed for signal transmission from server nodes to leaf nodes, then, distances from root nodes to leaf nodes

should be as short as possible, since distances are closely related to transmission time. In this case, besides minimizing the total cost (distance) of the resulting forest, it might also be beneficial to minimize the distances between roots and leaves. If there are several optimization criteria to be considered, we might also think about the convenience of optimizing either a single-objective problem (for example, weighting the different objectives) or moving towards a multi-objective problem. Another aspect to be considered is problem solving with an extremely large number of nodes. In this case, it might be handy to decompose the original problem into subproblems of smaller size and parallelize problem solving.

**Acknowledgements** This work has been partially supported by the Spanish Ministry of Economy and Competitiveness through the Cajal Blue Brain (C080020-09; the Spanish partner of the EPFL's Blue Brain initiative) and TIN2013-41592-P projects and by the Regional Government of Madrid through the S2013/ICE-2845-CASI-CAM-CM project. The authors thankfully acknowledge the computer resources, technical expertise and assistance provided by the Supercomputing and Visualization Center of Madrid (CeSViMa). L.A.-S. acknowledges support from the Spanish MINECO scholarship at the Residencia de Estudiantes.

## References

Aledo, J.A., Gámez, J.A., Molina, D.: Tackling the rank aggregation problem with evolutionary algorithms. Appl. Math. Comput. **222**, 632–644 (2013)

Anton-Sanchez, L., Bielza, C., Larrañaga, P.: Towards optimal neuronal wiring through estimation of distribution algorithms. In: Proceedings of the Fifteenth Annual Conference on Genetic and Evolutionary Computation, GECCO '13 Companion, pp. 1647–1650 (2013)

Bergmann, B., Hommel, G.: Improvements of general multiple test procedures for redundant systems of hypotheses. In: Multiple Hypotheses Testing, Medizinische Informatik und Statistik, vol. 70, pp. 100–115. Springer, Berlin (1988)

Bielza, C., Fernández del Pozo, J.A., Larrañaga, P., Bengoetxea, E.: Multidimensional statistical analysis of the parameterization of a genetic algorithm for the optimal ordering of tables. Expert Syst. Appl. **37**(1), 804–815 (2010)

Ceberio, J., Mendiburu, A., Lozano, J.A.: Introducing the Mallows model on estimation of distribution algorithms. In: Neural Information Processing. Lecture Notes in Computer Science, vol. 7063, pp. 461–470. Springer, Berlin (2011)

Ceberio, J., Irurozki, E., Mendiburu, A., Lozano, J.A.: A review on estimation of distribution algorithms in permutation-based combinatorial optimization problems. Prog. Artif. Intell. **1**(1), 103–117 (2012)

Ceberio, J., Irurozki, E., Mendiburu, A., Lozano, J.A.: A distance-based ranking model estimation of distribution algorithm for the flowshop scheduling problem. IEEE Trans. Evol. Comput. **18**(2), 286–300 (2014)

Ceberio, J., Mendiburu, A., Lozano, J.A.: Kernels of mallows models for solving permutation-based problems. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2015, Madrid, July 11–15, 2015, pp. 505–512 (2015)

Cobb, H., Grefenstette, J.: Genetic algorithms for tracking changing environments. In: Proceedings of the Fifth International Conference on Genetic Algorithms, Morgan Kaufmann, pp. 523–530 (1993)

Czajko, M.M., Wojciechowski, J.: Tree-based access network design under requirements for an aggregation network. Elektron. Konstr. Technol. Zastos. **4**, 23–27 (2009)

Delbem, A., de Carvalho, A., Policastro, C., Pinto, A., Honda, K., García, A.: Node-depth encoding for evolutionary algorithms applied to network design. In: Genetic and Evolutionary Computation. Lecture Notes in Computer Science, vol. 3102, pp. 678–687. Springer, Berlin (2004)

Delbem, A.C.B., de Lima, T.W., Telles, G.P.: Efficient forest data structure for evolutionary algorithms applied to network design. IEEE Trans. Evolut. Comput. **16**(6), 829–846 (2012)

Demšar, J.: Statistical comparisons of classifiers over multiple data sets. J. Mach. Learn. Res. **7**, 1–30 (2006)

Derrac, J., García, S., Molina, D., Herrera, F.: A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. Swarm Evolut. Comput. **1**(1), 3–18 (2011)

Durillo, J.J., Nebro, A.J.: jMetal: a java framework for multi-objective optimization. Adv. Eng. Softw. **42**(10), 760–771 (2011)

Friedman, M.: The use of ranks to avoid the assumption of normality implicit in the analysis of variance. J. Am. Stat. Assoc. **32**(200), 675–701 (1937)

Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman & Co, New York (1979)

Holland, J.H.: Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence. University of Michigan Press, Ann Arbor (1975)

Irurozki, E., Calvo, B., Lozano, J.A.: Sampling and Learning the Mallows and Generalized Mallows Models Under the Cayley Distance. University of the Basque Country, Spain (2014)

Knowles, J., Corne, D., Oates, M.: A new evolutionary approach to the degree constrained minimum spanning tree problem. IEEE Trans. Evol. Comput. **4**, 125–134 (2000)

Krishnamoorthy, M., Ernst, A., Sharaiha, Y.: Comparison of algorithms for the degree constrained minimum spanning tree. J. Heuristics **7**(6), 587–611 (2001)

Kruskal, J.B.: On the shortest spanning subtree of a graph and the traveling salesman problem. Proc. Am. Math. Soc. **7**(1), 48–50 (1956)

Larrañaga, P., Kuijpers, C.M.H., Murga, R.H., Inza, I., Dizdarevic, S.: Genetic algorithms for the travelling salesman problem: a review of representations and operators. Artif. Intell. Rev. **13**(2), 129–170 (1999)

Larrañaga, P., Lozano, J.A. (eds.): Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation. Kluwer, Boston (2002)

Mallows, C.L.: Non-null ranking models. Biometrika **44**(1–2), 114–130 (1957)

Prim, R.C.: Shortest connection networks and some generalizations. Bell Syst. Technol. J. **36**, 1389–1401 (1957)

Raidl, G., Julstrom, B.: Edge sets: an effective evolutionary coding of spanning trees. IEEE Trans. Evol. Comput. **7**(3), 225–239 (2003)

Reeves, C.R.: A genetic algorithm for flowshop sequencing. Comput. Oper. Res. **22**(1), 5–13 (1995)

Ruiz, R., Maroto, C.: A comprehensive review and evaluation of permutation flowshop heuristics. Eur. J. Oper. Res. **165**(2), 479–494 (2005)

Sedgewick, R., Wayne, K.: Algorithms, 4th edn. Addison-Wesley, Boston (2011)

Soak, S.M., Corne, D., Ahn, B.H.: A new encoding for the degree constrained minimum spanning tree problem. In: Knowledge-Based Intelligent Information and Engineering Systems. Lecture Notes in Computer Science, vol. 3213, pp. 952–958. Springer, Berlin (2004)

Syswerda, G.: A study of reproduction in generational and steady-state genetic algorithms. Found. Genet. Algorithms **1**, 94–101 (1991)

Tsutsui, S.: Node histogram vs. edge histogram: a comparison of probabilistic model-building genetic algorithms in permutation domains. In: IEEE Congress on Evolutionary Computation, 2006, pp. 1939–1946 (2006)