Trees and Forests

Andréa Cynthia Santos, Christophe Duhamel, and Rafael Andrade

Contents

A.C. Santos (✉)
ICD-LOSI, UMR CNRS 6281, Université de Technologie de Troyes, Troyes Cedex, France
e-mail: andrea.duhamel@utt.fr

C. Duhamel
LIMOS-UBP, UMR CNRS 6158, Université Blaise Pascal, Aubière Cedex, France
e-mail: christophe.duhamel@isima.fr

R. Andrade
Departamento de Estatística e Matemática Aplicada, Centro de Ciências, Universidade Federal do Ceará, Fortaleza, Brasil
e-mail: rca@lia.ufc.br

© Springer International Publishing AG 2016

R. Martí et al. (eds.), *Handbook of Heuristics*,
DOI 10.1007/978-3-319-07153-4_49-1

Abstract

Trees and forests have been a fascinating research topic in Operations Research (OR)/Management Science (MS) throughout the years because they are involved in numerous difficult problems, have interesting theoretical properties, and cover a large number of practical applications. A tree is a finite undirected connected simple graph with no cycles, while a set of independent trees is called a forest. A spanning tree is a tree covering all nodes of a graph. In this chapter, key components for solving difficult tree and forest problems, as well as insights to develop efficient heuristics relying on such structures, are surveyed. They are usually combined to obtain very efficient metaheuristics, hybrid methods, and matheuristics. Some emerging topics and trends in trees and forests are pointed out. This is followed by two case studies: a Lagrangian-based heuristic for the minimum degree-constrained spanning tree problem and an evolutionary algorithm for a generalization of the bounded-diameter minimum spanning tree problem. Both problems find applications in network design, telecommunication, and transportation fields, among others.

Keywords

bi-objective heuristic • DBMST • DCMST • forest • heuristics • Lagrangian heuristic • tree

Introduction

Trees and forests have been a fascinating topic in Operations Research (OR)/Management Science (MS) throughout the years because they are the core of difficult problems, have interesting theoretical properties, and cover a large number of practical applications. Their interest remains intact due to the technical and scientific challenges and the rich diversity of applications using such structures. For instance, several applications have been recently addressed: people tracking is modeled as a minimum cost arborescence problem by [32]; cluster-based topologies with connecting requirements are defined as a minimum spanning tree (MST) to improve wireless sensor network lifetime in [51]; peer-to-peer distributed interactions are studied by [39] as a spanning tree, where end-to-end delays are minimized; reliable telecommunication networks have been investigated by [54], in which redundancy is added to spanning trees by introducing k-cliques ($k \geq 2$); and difficult MST problems have been investigated including uncertain data [35], new variants for optical multicast network design [55], or even multiobjective MST versions [52,56].

Let $G = (V, E)$ be a finite, undirected, connected, and simple graph with a set V of vertices and a set E of edges, where $n = |V|$ and $m = |E|$. A tree $\mathcal{T} = (V', E')$ is a connected subgraph of G with no cycles, such that $V' \subseteq V$ and $E' \subset E$. Whenever $V' = V$, the corresponding tree defines a spanning tree of G. An arborescence is a special directed tree with a root node r. Moreover, a forest is a set of disjoint trees $\mathcal{F} = \{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_l\}$. An example of graph G' is given

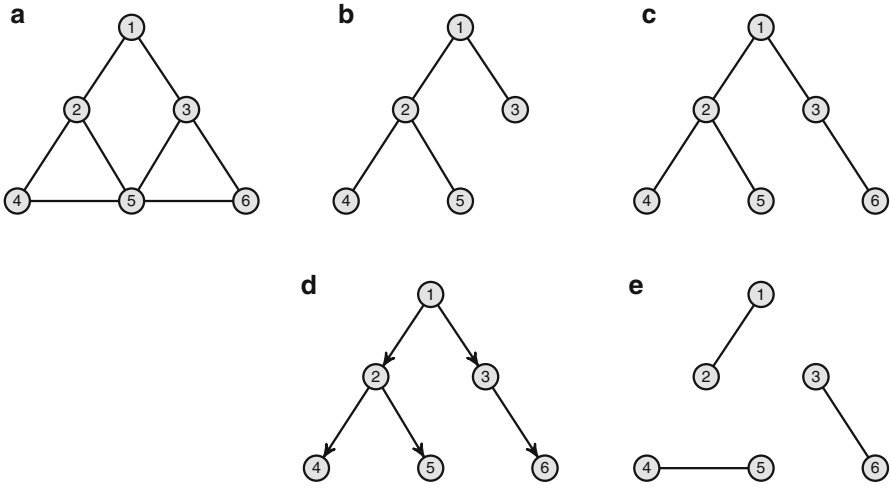

Fig. 1 Examples of tree, spanning tree, arborescence and forest. (**a**) Graph G'. (**b**) Tree. (**c**) Spanning tree. (**d**) Arborescence. (**e**) Forest

in Fig. 1a, followed by examples of a tree, a spanning tree, an arborescence, and a forest of G', respectively, in Fig. 1b–d where $r = 1$, and (e). In this chapter, it is assumed familiarity with basic graph definitions such as connected graphs, connected components, subgraphs, paths, cycles, edges incident to nodes, node degree, etc.

The basic minimum spanning tree (MST) problem is defined in a weighted graph G, where a cost $c_{ij} \geq 0$ is associated with every edge $[i, j] \in E$. It aims at finding a spanning tree $\mathcal{T}$ of G such that its total cost is minimized. The number of possible MSTs for a graph is very high with up to n^{n-2} for complete graphs [12]. However, polynomial-time algorithms such as Prim's, Kruskal's, Boruvka's, and cycle elimination algorithm [14] are available to compute an MST. The general idea of Prim's algorithm is to extend an MST from an initial node $i \in V$. At each step it includes the cheapest-weight edge $[i, j] \in E$ such that one of its extremities belongs to the tree, while the other one does not. Thus, the connected component is extended until all nodes $i \in V$ belong to the solution. Kruskal's algorithm performs forest merging as follows. Initially, each vertex is a tree. Then, at each iteration, the cheapest edge connecting two forests is selected, and the two forests are merged. Boruvka's algorithm is quite similar to Kruskal's algorithm, except that, at each iteration, the cheapest edge connecting each forest to another one is selected. Then the merges are done accordingly. Both Kruskal's and Boruvka's algorithms can be developed with asymptotic complexity of $O(m \log n)$, as well as Prim's algorithm using binary heaps. Moreover, Prim's algorithm complexity can be improved to $O(m + n \log n)$ by using Fibonacci heaps; see [14] for details. The cycle elimination algorithm scans the graph using a depth-first search (DFS) strategy. The edge $[i, j]$ is added to the tree whenever a vertex $j \in V$ not yet visited is found from the current

node i of the DFS. However, if j has already been visited, it means a cycle has been found. In this case, the highest-weight edge from the found cycle is removed from the incumbent partial solution. DFS complexity is $O(m + n)$. Thus, considering a complete graph, the cycle elimination algorithm runs in $O(mn)$ since there are $O(m-n-1)$ possible back edges and a cycle scan is done in $O(n)$. Several NP-hard problems rely on an MST with additional constraints on nodes, on edges, or even on the MST general structure. Thus, algorithms that compute an MST are usually adapted for such NP-hard extensions in order to define basic heuristics and obtain initial feasible solutions.

A large number of difficult problems rely on trees and forests. Studies [10, 22, 33, 41] provide a collection of such problems, along with their complexity analysis. An example where constraints on the tree structure make the problem difficult is the Steiner Tree problem [3, 24, 34, 38]. Given a subset $N \subset V$ of Steiner nodes (also referred as terminal nodes), the Steiner Tree problem consists in determining a particular minimum tree covering N. Also, setting constraints on the nodes or on the edges (paths) of an MST can transform the problem into an NP-hard problem. For instance, the degree-constrained minimum spanning tree (DCMST) problem [4, 13, 15] consists in finding an MST such that each vertex has a degree not larger than a maximum given value $k \in \mathbb{N}^*$. The bounded-diameter minimum spanning tree (BDMST) problem [26,40,52] is an example of a difficult problem where constraints are imposed on the MST diameter, i.e., the diameter of a tree is the number of edges in the longest path among any pair of vertices. Thus, the BDMST aims at finding an MST where the unique path between any pair of nodes does not exceed a given number of edges. Basic problems can also become difficult due to the nature of the data. For instance, the Minimum Arborescence problem [20] is defined in a digraph with real cost values associated with each arc. The objective is to find a minimum cost arborescence. Thus, for problems involving trees and forests, adding constraints or even changing few parameters or data can turn a P problem into an NP-complete problem.

Key components for solving difficult tree and forest problems, as well as insights to develop efficient heuristics relying on such structures, are surveyed here. They are usually combined to obtain very efficient metaheuristics, hybrid methods, and matheuristics. Section "Basic Features for Trees and Forests" is dedicated to a number of basic heuristics, local searches, etc. Then, two applications are described as examples, respectively, in sections "A Lagrangian Heuristic for DCMST and Evolutionary Heuristic for a Generalization of BDMST". Two different strategies to handle the difficult constraints are presented in these sections. The strategy used for DCMST consists of removing the difficult constraints and adding them into the objective function, following a Lagrangian relaxation. Another strategy is considered for the BDMST, where the difficult constraints are addressed as a new criterion and a bi-objective genetic algorithm is used.

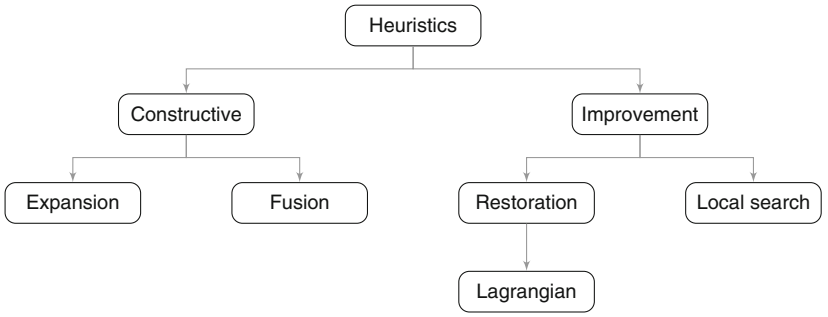

Fig. 2 A classification of heuristics for trees

Basic Features for Trees and Forests

Several heuristics, local searches, operators, different encodings, and perturbations are available in the literature to build trees and forests. They are very often combined to produce sophisticated methods. The most common heuristic structures are classified as shown in Fig. 2. The idea of this classification is not to cover all available heuristics for trees and forests, but to introduce the most used strategies having a strong potential to derive other heuristics for various difficult problems. There are two major classes of heuristics: constructive and improvement heuristics. The former includes edges or nodes at each iteration to obtain a feasible solution. The latter starts with an initial solution (e.g., a spanning tree), not necessarily feasible, and try to improve it in order to obtain better feasible solutions. Constructive and improvement heuristics are presented below, in such a way they can be adapted and applied to different difficult problems on trees and forests, where nodes, edges, or structural additional constraints are considered.

Constructive heuristics generate a feasible solution iteratively, using mainly two strategies: tree expansion and tree fusion. In the tree expansion strategy, one node is added at a time by means of an edge $e = [i, j] \in E$, where one extremity of e belongs to the solution under construction and the other extremity does not. Thus the partial incumbent solution is always a connected component. Tree expansion heuristics are usually implemented using Prim's algorithm. Examples of tree expansion heuristics are the One Time Tree (OTT) [1] and randomized greedy heuristic (RGH) [47] developed for the BDMST. Fusion heuristics start from initial disconnected forests and try to connect them by adding an edge $e = [i, j] \in E$ at a time. Thus a partial solution contains several connected components and Kruskal's algorithm can be adapted and applied [37], see section "A Greedy Heuristic for DCST". The way new edges/vertices are selected for inclusion in partial solutions (e.g., constructive heuristics) or for removal from a solution (e.g., improvement heuristics), after evaluating the objective function, has a strong impact on the final solution quality. The most common ways to select edges/vertices are greedy, semi-greedy, and random. They can also be handled considering a single criterion or

multi-criteria. Without loss of generality and considering a minimization function, a greedy criterion selects the edge/vertex insertion that leads to the smallest increase in the objective function. One way to handle a semi-greedy criterion is to build a list of good candidates to enter the solution, not necessarily the best ones, and make a random choice from this list. Another idea frequently found in the literature is to consider a normalized ratio between the cost and the impact on difficult constraints, e.g., a ratio between the edge cost and the additional increase in the diameter for the BDMST. Various criteria can also be addressed using a priority order instead of a ratio. For instance, one can use a second optimization criterion to select one edge, e.g., the impact on difficult constraints, whenever several edges with the smallest costs occur.

Improvement heuristics start from a solution, not necessarily feasible, and try to obtain a feasible solution by exchanging edges in the tree (resp. in a forest) with edges outside the solution. They can be roughly classified as restoration and local search heuristics. The restoration heuristics try to transform solutions in feasible ones by minimizing violations. A classical example of restoration heuristics is Lagrangian heuristics which have appeared with the pioneering works of [29, 30]. They have been broadly applied to a number of difficult problems relying on trees and forests [5, 16, 42]. The general idea is to relax difficult constraints and transfer them to the objective function, keeping a simple problem that is refined until a good solution to the overall problem is obtained. Consider a generic NP-hard optimization problem given by $Z = \{\min cx : Ax \geq b, Bx \geq d, x \in \{0,1\}^t\}$, where A and B are matrix with rational coefficients and with dimensions $r \times t$ and $s \times t$, respectively; b, c, and d are rational vectors, and $x \in \{0,1\}^t$ are integer variables. Let $\{\min cx : Bx \geq d, x \in \{0,1\}^t\}$ be an easy problem that can be solved in polynomial time, and $Ax \geq b$ be the difficult constraints. Thus, Lagrangian multipliers $\lambda \in \mathbb{R}^r_+$ are associated with the difficult constraints and added into the objective function. The Lagrangian problem (LP) is given by $LP = \{\min cx + \lambda(b - Ax) : Bx \geq d, x \in \{0,1\}^t\}$ which is a lower bound on Z. The best lower bound is given by $LP(\lambda^*) = \max\{LP(\lambda), \lambda \geq 0\}$, called dual Lagrangian problem. The dual Lagrangian problem can be solved by using, for example, the subgradient methods of [21] or the volume algorithm of [7] or even by computing the Lagrange multipliers approximately using the multiplier adjustment method [9]. Theoretical results to determine the conditions stating when a Lagrangian relaxation can be better than a linear relaxation are provided by [28]. Obviously, the mathematical formulation and the decomposition will strongly impact the quality of solutions produced using Lagrangian relaxation. This kind of restoration heuristic has an interesting property: at each iteration dual and primal solutions can be built.

Local searches are very sophisticated improvement heuristics which contribute to producing very good local optima (eventually global optima). The most common local search moves, broadly applied to trees, are the *edge drop* move and the *edge insertion* move [18, 40]. Given an initial feasible solution for a difficult problem relying on a tree, the *edge drop* move consists of removing an edge from the tree. This produces two distinct connected components which are reconnected using an

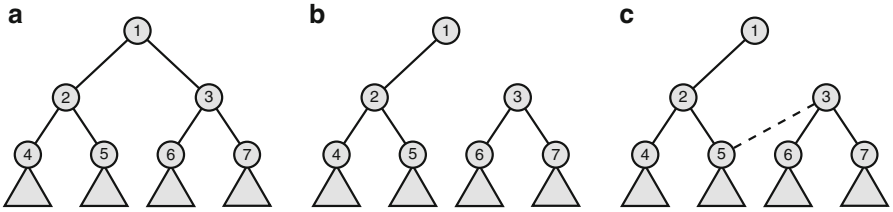

Fig. 3 Example of an *edge drop* move. (**a**) Tree. (**b**) Disconnecting 1 and 3. (**c**) Reconnecting 3 and 5

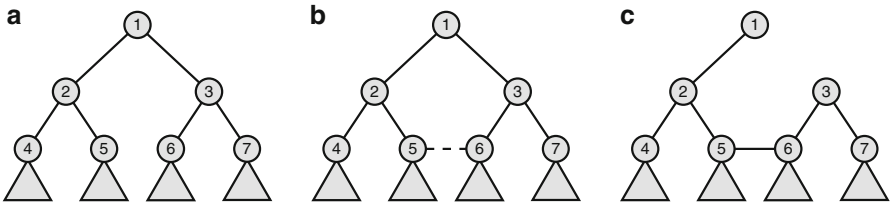

Fig. 4 Example of an *edge insertion* move. (**a**) Tree. (**b**) Connecting 5 and 6. (**c**) Disconnecting 1 and 3

edge not in the tree. The greedy and semi-greedy criteria and multi-criteria can be used to chose the edge to enter the tree. Figure 3 illustrates an example, where Fig. 3a is a feasible initial solution. Then edge $[1, 3]$ is removed as shown in Fig. 3b, generating two separate connected components. Consider $[3, 5]$ as the edge which produces the best improvement possible. Thus it enters the solution following a greedy strategy as shown in Fig. 3c.

On the other hand, in the *edge insertion* move, an edge is first inserted in the tree. By definition of a tree, it necessarily results in a cycle. Then, an edge may be removed from the cycle. Figure 4 depicts an example for this move. A feasible solution is presented in Fig. 4a. In the sequel, edge $[5, 6]$ is introduced and the cycle $\{5, 6, 3, 1, 2, 5\}$ is formed as shown in Fig. 4b. Suppose that removing edge $[1, 3]$ will produce the highest gain in the objective function. Thus this edge is dropped following a greedy strategy as depicted in Fig. 4c.

A number of variations on the *edge drop* and *edge insertion* moves are found in the literature. These moves are also referred as 1-opt since one edge in the incumbent solution is replaced by another one that does not belong to the solution. When applied to trees, two strategies are available to perform such a 1-opt move: either first dropping an edge or first inserting an edge. This may lead to different final solutions. The k-opt generalizes the 1-opt move by replacing k edges from an incumbent solution. This move is formalized in section "A Lagrangian Heuristic for DCMST."

Data Structures for Trees

Choosing the right data structure for representing and manipulating a solution is a critical task since it directly impacts the complexity of basic operations. They include both building and modifying a solution. Two mapping functions are associated with the data structures: how to store the information from a given tree (encoding) and how to obtain a tree out of the information (decoding). In this section, a basic direct representation is presented as well as several data to handle additional constraints on the trees. In section "Encodings for Trees," other representations are presented, along with the way to obtain the associated tree. Several insights are given regarding their use in evolutionary algorithms.

Let $G = (V, E)$ and $s \in V$ be, respectively, a graph defined as before and an arbitrary root vertex [14]. The following data structure can be used for directed and undirected trees, depending on the root node which can be artificial or not. The predecessor j of a vertex i in a tree is the first vertex in the path from i to s. By definition of a tree, j is unique and only depends on the choice of s. A successor j of a vertex i is a vertex whose predecessor is i. One of the simplest ways to store the information of a tree in a fixed-size data structure is to use the predecessors. This direct representation keeps the predecessors in a vertex-indexed vector $pred$. Thus $pred(i)$ is the predecessor of vertex i or, alternatively, the edge between i and $pred(i)$. The root has no predecessors; then $pred(s) = \emptyset$. Such a structure allows an $O(1)$ access to each predecessor and to each edge connecting the subtree rooted at i with the rest of the tree. It also allows a scan of all edges of a tree in $O(n)$.

Additional redundant data may be useful to perform tree manipulation and to allow efficient checking operations. One such data is the list of all the direct successors of a vertex. It can be explicitly defined as a list of vertices for each vertex, as well as be implicitly stored by adding two attributes to each vertex i: its first successor $succ(i)$ and $next(i)$, the next successor for $pred(i)$. For instance, if the direct successors of vertex a are $\{b, c, d\}$, the first successor of a is $b = succ(a)$. The successors of a are as follows: $c = next(b)$, $d = next(c)$ and $next(d) = \emptyset$. This encoding shares similarities with the forward star structure presented in [2] for graphs, and the implicit list can be made bidirectional by adding an attribute $prev(i)$ to each vertex which stores the previous successor in the list of $pred(i)$. This way, $O(1)$ operations such as successor insertion or deletion are ensured. The number of direct successors $nb\_succ(i)$ can be kept as well.

Successor and predecessor structures allow managing several operations such as finding a cycle for *edge insertion* moves, finding the cheapest-weight costs for *edge drop* moves, and, with a few additional information, addressing difficult constraints like degree and diameter constraints. For instance, the $depth(i)$ of a vertex i is equal to the number of edges in the unique path from a given root vertex s to i. It can be used for checking hop and diameter constraints.

Scanning a cycle created by adding an edge $[i, j] \in E$ into a spanning tree can be done efficiently using the predecessor structure. First, each node in the path $i \rightarrow s$ is set as visited. The same is done in the path $j \rightarrow s$, stopping as soon as a given vertex

k has already been visited. Thus k belongs to both paths $i \rightarrow s$ and $j \rightarrow s$. Getting the edge with the highest cost can be done by keeping the edge with the highest cost found when scanning the path $i \rightarrow s$ (resp. $j \rightarrow s$) at each visited node. A modified DFS or breadth-first search (BFS) starting at i and stopping as soon as j is found could also be used. However, this would require more sophisticated tree data structures to obtain $O(n)$ complexity as for the predecessor vector. This way, the *edge insertion* move can be performed in $O(n)$.

Another useful data can be defined to quickly check if a vertex belongs to a subtree. Let $first(i)$ and $last(i)$ be two indexes associated with each vertex i. A DFS is applied on the tree, starting from s. Then $first(i)$ stores the DFS vertex counter the first time i is visited and $last(i)$ gets the DFS vertex counter when the DFS leaves the subtree of i. Thus, vertex j belongs to the subtree of vertex i if and only if $first(j) \in [first(i), last(i)]$. Besides, the number of vertices in the subtree of i, i included, is $size(i) = last(i) - first(i) + 1$. This information can be used to check for special cases in *edge insertion* moves, if one extremity belongs to the subtree of the other one. The same idea can be used to check if a tree is feasible in the capacitated MST [18, 22]. A demand D_i is associated with each vertex i and the sum of the demands in any subtree must not exceed a capacity Q. A demand counter is used instead of a vertex counter in the DFS.

The *edge drop* move requires reconnecting two trees disconnected after an edge $[i, j]$ has been removed. Without loss of generality, suppose i is the predecessor of j, i.e., $i = pred(j)$. First, all the vertices are set as unvisited. Then, all vertices in the subtree of j (included) are set as visited by performing a DFS or a BFS, using the list of successors presented before. Getting the lowest cost edge reconnecting the two trees can be done in $O(m)$ by finding the smallest edge in E with one extremity visited and the other one unvisited. This last operation is the most expensive and the *edge drop* move is in $O(m)$.

As mentioned before, the depth of a vertex from the root in a tree can be used to check hop and diameter constraints. For the root vertex, $depth(s) = 0$. A simple DFS or BFS starting from s is sufficient to compute the depth of each vertex in $O(n)$. They may require an artificial vertex 0 playing the role of center, $depth(0) = 0$. The artificial vertex connects the central vertex whenever D is even and one of the extremities for a central edge if D is odd. Let $L = D/2$ resp. $L = (D-1)/2$ be the maximum depth allowed for a node, respectively, when D is even resp. odd. Thus, whenever an artificial vertex is used, a diameter limit D on the tree corresponds to a depth limit $L + 1$ for each vertex from vertex 0. Checking the degree is simpler; one may just add the number of successors and predecessors for each vertex.

Encodings for Trees

The tree representation given in the section "Data Structures for Trees" can be used for most constructive heuristics and for moves in local searches and metaheuristics. However, evolutionary algorithms (EAs) usually require additional properties on the representation, and several other tree encodings have been proposed. In EAs,

operations are mostly done on encodings (the chromosomes) rather than on the solutions. These usually consist of crossovers and mutations in order to make the set of encodings (the population) evolve. Selection is done to obtain the best elements. It requires the evaluation of each element through the construction of the associated solution, named decoding.

Several properties on the encoding/decoding, i.e., the mapping between encodings and solutions, are needed to fully benefit from the design of EAs: (i) the time and space complexities for those operations should be as low as possible, (ii) any encoding should correspond to a feasible solution, (iii) the decoding should be unbiased, (iv) there should be at least one encoding leading to each optimal solution, (v) any offspring obtained from crossover and mutation should be valid, and (vi) the offsprings should share as much similarities as possible with their parents.

According to those criteria, the predecessor structure presented before offers a $O(n)$ complexity for encoding/decoding. However, a random vector of predecessors is very unlikely to be feasible. Classical crossovers and mutations also generate infeasible offsprings and a repair operator is required. On the other hand, there always exists one encoding for each optimal solution and the feasible offsprings inherit a lot of similarities from their parents.

Another classical representation is the Boolean edge-indexed vector. The status of each edge (used/unused) is stored in the solution. Thus it follows the binary encoding paradigm suggested in the early versions of genetic algorithms (GAs). However, a random vector is highly unlikely to be feasible and crossovers and mutations seldom produce feasible offsprings. Thus the repair operator is mandatory, unless defining dedicated crossovers and mutations.

Prüfer sequences have also been used to encode trees. They were introduced in [45] to prove the Cayley's theorem [12] and there is a bijection between the set of spanning trees and the set of Prüfer sequences. They are basically repetition vectors of size $n-2$ in which each vertex i appears $degree(i)-1$ times. Encoding and decoding can be done in $O(n \log n)$. There is no need for repair operator since every sequence corresponds to a feasible spanning tree. However, while appealing, this representation suffers from several drawbacks [23], especially with respect to property (vi) since a small change in the sequence might lead to a complete different solution.

Random keys [8] have also been used and are quite popular. A random key is a weight in $[0, 1]$. For trees, one random key is associated with each edge, leading to a real-valued vector of size m. The decoding first consists of sorting the edges according to their random key and then applying a Kruskal-based constructive heuristic. Thus, it is done in $O(m \log m)$. It always leads to a feasible spanning tree and does not require a repair operator.

All the encodings suffer from shortcomings with respect to the properties (i)–(vi) mentioned before [46]. Moreover, they may not be able to handle sparse graph or additional constraints on the tree structure. Aside from random keys, a good alternative seems to be edge-set encoding [46]: the edges of the tree are directly stored in a variable-length vector. Thus, this is an explicit variable-size encoding whose encoding/decoding can be done in $O(n)$. It is less biased toward special

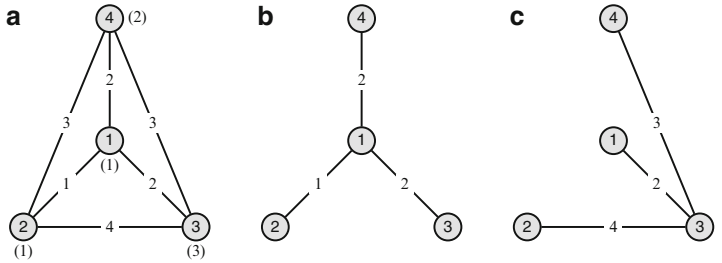

Fig. 5 Example of a tree with degree constraints. (**a**) A graph G. (**b**) MST of G. (**c**) A DCST of G

trees than random keys. It has been shown to be quite effective for solving NP-hard extensions of MST, provided dedicated crossovers and mutations are used.

A Lagrangian Heuristic for DCMST

The use of Lagrangian relaxation in heuristic procedures has been proved to be quite a powerful tool for solving certain problems, especially DCMST. A Lagrangian heuristic (LH) involves a small number of ingredients that can be implemented efficiently: (i) determining Lagrange multipliers and evaluating Lagrange subproblems, (ii) obtaining feasible solutions from the Lagrange multipliers, and (iii) improving feasible solutions by a local search. The main advantages of an LH are that it is simple to implement, solutions of good quality can be obtained in a reasonable computational time, and lower bounds (LB) and upper bounds (UB) are available at each iteration. In addition, an LH can be used to obtain starting incumbent solutions in more complex exact algorithms.

Let $G = (V, E)$ be a graph defined as before. Costs $c_e \in \mathbb{R}^+$ are associated with each edge $e \in E$, and a maximum degree $d_v \in \mathbb{N}^*$ is associated with each node $v \in V$. The DCMST problem consists in finding a minimal cost spanning tree $\mathcal{T}$ of G such that the degree constraints are ensured for each node.

Figure 5a illustrates an example of a graph G, in which edge costs are reported in the middle of each edge and the maximum node degree constraints for spanning trees of G are given in brackets near each node. An MST of G, with cost equal to 5, is given in Fig. 5b. One may note that this MST violates the degree constraint in node 1. A feasible solution for the degree-constrained spanning tree (DCST) for G is provided in Fig. 5c, with cost equal to 9.

Problem Formulation

In order to present a mathematical model for DCMST problem, let $E(S) \subseteq E$ be the set of edges with both extremities in $S \subseteq V$. Let $\delta(v) \subseteq E$ be the set of edges adjacent to $v \in V$. Binary variables x_e, for all $e \in E$, represent the characteristic vector for a spanning tree of G, where $x_e = 1$ if an edge e belongs to the solution, and $x_e = 0$ otherwise. The mathematical formulation is given from (1) to (4).

$$(P) \quad \min_{x \in \{0,1\}^m} \sum_{e \in E} c_e x_e \tag{1}$$

$$s.t. \sum_{e \in E} x_e = n - 1, \tag{2}$$

$$\sum_{e \in E(S)} x_e \leq |S| - 1, \quad S \subset V, \tag{3}$$

$$\sum_{e \in \delta(v)} x_e \leq d_v, \quad \forall\, v \in V. \tag{4}$$

An MST is defined from (1) to (3) and constraints (4) control the degree of each vertex. These last constraints are the difficult ones. Thus they are relaxed in the LH presented next. Note that when $d_v = 2$, for all $v \in V$, the problem is reduced to finding a Hamiltonian path of minimum cost in G. Thus the DCMST is NP-hard [22].

The role of Lagrange penalties or multipliers is to take into account violations on some relaxed constraints. Given a tree $\mathcal{T}$ and for every node $v \in V$, the amount of node violation is the difference $\sum_{e \in \delta(v)} x_e - d_v$ between its maximum degree d_v and its degree in $\mathcal{T}$. In a DCST only nonpositive node violations are allowed. Positive node violation is penalized by associating a nonnegative penalty λ_i with each violated node i. In this case, every edge incident to i has its cost increased by λ_i. When both end nodes i and j of an edge $[i, j] \in E$ have positive penalties λ_i and λ_j, respectively, the resulting Lagrange cost becomes $c_{ij} + \lambda_i + \lambda_j$.

Consider the MST in Fig. 5b. It violates the degree constraint of node 1 (three edges are adjacent to this node and the maximum allowed is one). Figure 6 illustrates an example of MSTs computed with edge costs modified by Lagrange multipliers, where spanning trees of minimum cost are defined by solid lines in each graph. If a penalty $\lambda_1 = 10$ is set and $\lambda_i = 0$ for $i \in V \setminus \{1\}$ and an MST is computed for the graph with modified edge costs, the spanning tree in Fig. 6a is obtained. The degree constraint of node 2 is violated. Alternatively, if $\lambda_1 = 10$, $\lambda_2 = 2$, and $\lambda_i = 0$ for $i \in V \setminus \{1, 2\}$, an MST on the modified graph is in Fig. 6b. It is also feasible for the original graph G in Fig. 5a.

Thus, the idea is to determine the "best" set of Lagrange multipliers leading to an MST on the graph with modified edge costs whose Lagrangian solution value is equal to the value of an optimal DCMST. To do so, the Lagrangian problem (P_λ) is defined by associating Lagrange multipliers $\lambda \in \mathbb{R}^n_+$ with constraints (4) and bringing them to the objective function (1).

$$(P_\lambda) \quad \min_{x \in \{0,1\}^m} \sum_{[i,j] \in E} (c_{ij} + \lambda_i + \lambda_j) x_{ij} - \sum_{i \in V} \lambda_i d_i \tag{5}$$

$$s.t. \quad (2)\text{–}(3).$$

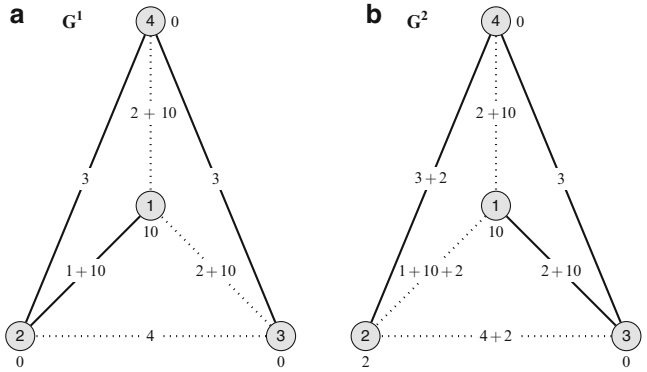

Fig. 6 Example of graphs with edge cost modified by Lagrange multipliers. (**a**) MST with λ_1=10. (**b**) MST with $\lambda_1 = 10, \lambda_2 = 2$

For any $\lambda \geq 0$, the solution value of (P_λ) is a lower bound on the optimal solution of (P). Consequently, the solution of the problem (D) is the best lower bound one can get for the solution of (P). Problem (D) is the Lagrangian dual problem of (P).

$$(D) \quad \max_{\lambda \geq 0} \min_{x \in \{0,1\}^m} \sum_{[i,j] \in E} (c_{ij} + \lambda_i + \lambda_j) x_{ij} - \sum_{i \in V} \lambda_i d_i \tag{6}$$

$$s.t. \quad (2)\text{–}(3).$$

The MST on the graph G^2 in Fig. 6b has a cost $(3+5+12)-(10\times1+2\times1) = 8$. This is a lower bound on the optimal DCMST value of the graph in Fig. 5a. It is optimal since that solution is feasible for G and its original cost in G is 8.

A Subgradient Procedure for DCMST Problem

The classical subgradient (SG) method of Held et al. [31] is used to compute Lagrange multipliers λ iteratively for problem (P_λ). At each iteration k of the method, the idea is to find a direction s^k and a step size t_k to move from λ^k to a new set of multipliers λ^{k+1}:

$$\lambda^{k+1} = \max\{0, \lambda^k + t_k s^k\} \tag{7}$$

s^k is the subgradient of (P_{λ^k}) with respect to the Lagrangian solution x^k. Its coordinates are given by

$$s_i^k = \sum_{e \in \delta(i)} x_e^k - d_i, \quad \forall\, i \in V \tag{8}$$

Determining the step size t_k requires an incumbent UB on the optimal solution value of (P), the Lagrangian value LB^k referred to x^k, and the norm of the direction s^k. It is defined as

$$t_k = \frac{\alpha(UB - LB^k)}{||s^k||^2} \tag{9}$$

where α is a scaling factor. Initially $\alpha = 2$ and it is reduced (usually $\alpha \leftarrow \alpha/2$) after some iterations with no improvement on the best LB. Thus, convergence of the SG algorithm is ensured. The number of iterations can be limited to a maximum value determined according to the characteristic of the problem instances being solved. If the Lagrangian solution is feasible for (P) and its Lagrangian value is equal to UB, it is optimal and the algorithm stops. Further details on the SG algorithm for DCMST problem can be found in [5].

As an example, the SG algorithm is applied to the graph G in Fig. 5a. Initially $\lambda^0 = 0$ and the resulting graph $G(\lambda^0)$ has the same edge costs as in G. The Lagrangian solution $\mathcal{T}^0$ for $G(\lambda^0)$ is the MST of G (Fig. 5b), and its Lagrangian cost is $\text{LB}^0 = 5$. The degree of nodes 1, 2, 3, and 4 in $\mathcal{T}^0$ are 3, 1, 1, and 1, respectively. The initial subgradient direction is $(s^0)^t = (2, 0, -2, -1)$. Figure 8 presents graphs $G(\lambda^k)$ of each SG iteration k. The Lagrange edge costs are displayed near each edge and Lagrange multipliers near each node. Moreover, Lagrangian solutions $\mathcal{T}^k$ of $G(\lambda^k)$ are defined by solid lines. Lagrange multipliers are computed according to Fig. 5 as follows: suppose the incumbent $\text{UB} = 9$ is given by the DCST in Fig. 5c, as computed by heuristic in section "A Greedy Heuristic for DCST". The new set of Lagrange multipliers is $(\lambda^1)^t = (16/9, 0, 0, 0)$. The corresponding graph $G(\lambda^1)$ leads to the Lagrangian solution $\mathcal{T}^1$ in Fig. 7, in solid lines. The new direction s^1 is carried forward in Fig. 8, column $\mathcal{T}^1$. The SG algorithm iterates until iteration 4, where the Lagrangian solution $\mathcal{T}^4$ is a feasible DCST for G. The original cost of $\mathcal{T}^4$ in G is 8 and its Lagrangian cost is $\text{LB}^4 = 596/75$. As all edges of G have integer costs, the optimal solution value must be integer. Therefore, the optimal solution value UB^* is such that $\lceil 596/75 \rceil \leq \text{UB}^* \leq 8$. Thus $\mathcal{T}^4$ is optimal and the algorithm stops.

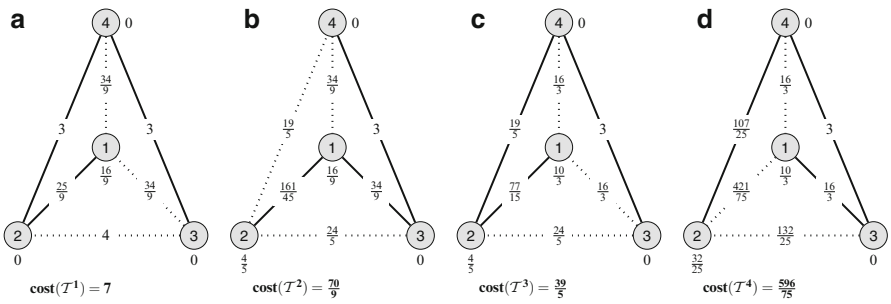

Fig. 7 Example of iterations for the SG method. (**a**) $\mathbf{G(\lambda^1)}$. (**b**) $\mathbf{G(\lambda^2)}$. (**c**) $\mathbf{G(\lambda^3)}$. (**d**) $\mathbf{G(\lambda^4)}$

$$\mathcal{T}^1$$

$$\lambda^1 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} + \frac{2(9-5)}{9} \begin{pmatrix} 2 \\ 0 \\ -2 \\ -1 \end{pmatrix} \overset{\lambda \geqq 0}{=} \begin{pmatrix} \frac{16}{9} \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

$$s^1 = \begin{pmatrix} 0 \\ 1 \\ -2 \\ 0 \end{pmatrix}$$

$$\mathcal{T}^2$$

$$\lambda^2 = \begin{pmatrix} \frac{16}{9} \\ 0 \\ 0 \\ 0 \end{pmatrix} + \frac{2(9-7)}{5} \begin{pmatrix} 0 \\ 1 \\ -2 \\ 0 \end{pmatrix} \overset{\lambda \geqq 0}{=} \begin{pmatrix} \frac{16}{9} \\ \frac{4}{5} \\ 0 \\ 0 \end{pmatrix}$$

$$s^2 = \begin{pmatrix} 1 \\ 0 \\ -1 \\ -1 \end{pmatrix}$$

$$\mathcal{T}^3$$

$$\lambda^3 = \begin{pmatrix} \frac{16}{9} \\ \frac{4}{5} \\ 0 \\ 0 \end{pmatrix} + \frac{2(9-\frac{70}{9})}{3} \begin{pmatrix} 1 \\ 0 \\ -1 \\ -1 \end{pmatrix} \overset{\lambda \geqq 0}{=} \begin{pmatrix} \frac{10}{3} \\ \frac{4}{5} \\ 0 \\ 0 \end{pmatrix}$$

$$s^3 = \begin{pmatrix} 0 \\ 1 \\ -2 \\ 0 \end{pmatrix}$$

$$\mathcal{T}^4$$

$$\lambda^4 = \begin{pmatrix} \frac{10}{3} \\ \frac{4}{5} \\ 0 \\ 0 \end{pmatrix} + \frac{2(9-\frac{39}{5})}{5} \begin{pmatrix} 0 \\ 1 \\ -2 \\ 0 \end{pmatrix} \overset{\lambda \geqq 0}{=} \begin{pmatrix} \frac{10}{3} \\ \frac{32}{25} \\ 0 \\ 0 \end{pmatrix}$$

$$s^4 = \begin{pmatrix} 0 \\ 0 \\ -1 \\ 0 \end{pmatrix}$$

Fig. 8 Updating Lagrange multipliers during the SG iterations shown in Fig. 7

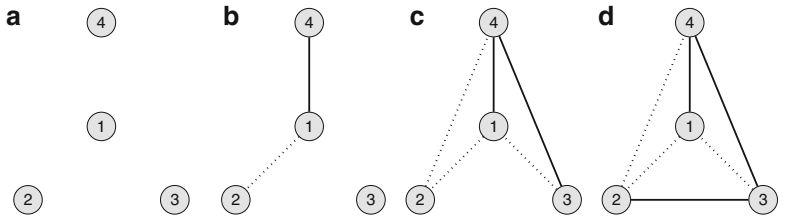

Fig. 9 Example of Kruskal-based heuristic for DCST. (**a**) Initial. (**b**) First edge. (**c**) Second edge. (**d**) Third edge

A Greedy Heuristic for DCST

Kruskal's algorithm can be adapted [5] to compute DCSTs on a graph $G = (V, E)$. It consists of (i) ordering the edges in E by nondecreasing edge costs; (ii) creating a forest with $|V|$ trivial trees $C_i = \{i\}$, for all $i \in V$; and (iii) connecting all disjoint trees (forests) by including one edge at a time, using the ordered list until a spanning tree is obtained. A vertex is saturated if its degree matches the maximum degree allowed. A tree is saturated if all its vertices are saturated. Then, an edge $[u, v] \in E$ is inserted in the solution if both u and v are not saturated and if they do not belong to the same tree. Moreover, merging the two trees using $[u, v]$ must not lead to a saturated tree, except if the resulting tree spans V.

Consider the graph G in Fig. 5a and assume that the ordered list of edges is $\{[1,2],[1,4],[1,3],[2,4],[3,4],[2,3]\}$. Initially, each vertex is a trivial tree. The first edge to be considered is $[1,2]$. It is discarded as its inclusion would result in a saturated tree. In the sequel, edge $[1,4]$ can be added since it does not saturate vertices 1 and 4 nor create a cycle. Node 1 is saturated; thus, the third edge $[1,3]$ is rejected. The fourth edge $[2,4]$ is also not accepted because it would result in a saturated tree. The fifth edge $[3,4]$ is included, *idem* for the sixth edge $[2,3]$. Then the algorithm stops with a feasible DCST. Note that this heuristic stops with a feasible solution for complete graphs only. Figure 9 presents this example with the ordered list of edges defined above. Solid lines correspond to edges included, while dotted lines correspond to discarded edges.

This heuristic can also be applied to graphs with modified Lagrange edge costs during the SG and compute its cost on the graph G. Nevertheless, doing this every SG iteration is time consuming. Finally, this heuristic can be used only when a Lagrangian solution improves the best incumbent LB on the optimal solution of (P).

Improving DCSTs with a Local Search

Given a graph $G = (V, E)$ and a spanning tree $\mathcal{T}$ of G, let $E(\mathcal{T})$ be the set of edges in $\mathcal{T}$. Moreover, consider a given subset $\{e_1, e_2, \cdots, e_k\} \subset E(\mathcal{T})$ of the edges belonging to $\mathcal{T}$. If there is a subset of edges $\{\hat{e}_1, \hat{e}_2, \cdots, \hat{e}_k\} \subset E \setminus E(\mathcal{T})$ such that $E(\mathcal{T}) \setminus \{e_1, e_2, \cdots, e_k\} \cup \{\hat{e}_1, \hat{e}_2, \cdots, \hat{e}_k\}$ induces a spanning tree $\hat{\mathcal{T}}$ of G, then $\hat{\mathcal{T}}$ is said to belong to a k-neighborhood of $\mathcal{T}$, denoted by $\hat{\mathcal{T}} \in N_k(\mathcal{T})$, and that $\mathcal{T} \in N_k(\hat{\mathcal{T}})$.

Consider $k+1$ connected components $C_1, C_2, \cdots, C_{k+1}$ obtained after removing k edges $\{e_1, e_2, \cdots, e_k\} \subset E(\mathcal{T})$ from a given spanning tree $\mathcal{T}$. If $\{\bar{e}_1, \bar{e}_2, \cdots, \bar{e}_k\} \subset E \setminus E(\mathcal{T})$ is a minimum cost subset of k edges to reconnect the $k+1$ components $C_1, C_2, \cdots, C_{k+1}$ forming a new spanning tree, then $\mathcal{T} \cup \{\bar{e}_1, \bar{e}_2, \cdots, \bar{e}_k\} \setminus \{e_1, e_2, \cdots, e_k\}$ is called a *k-opt edge exchange*. Such a move corresponds to applying k times an *edge drop* or an *edge insertion* move, as seen in section "Basic Features for Trees and Forests".

Figure 10 illustrates an example of a 2-*opt edge exchange*. Tree $\mathcal{T}^1$ in Fig. 10b is obtained from $\mathcal{T}$ in Fig. 10a by removing the edges $[1,4],[2,3]$ and adding the edges $[1,3],[2,4]$ while keeping the feasibility of the resulting tree.

Determining the best *k-opt edge exchange* move among all $\binom{|E(\mathcal{T})|}{k}$ combinations of k edges of $E(\mathcal{T})$ is very time consuming. A reasonable trade-off between solution quality and processing time is to limit k to 3. A greedy local search then iteratively performs *k-opt edge exchange* moves, until reaching a local optimum.

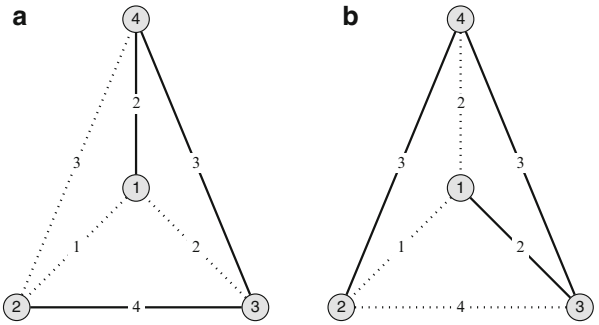

Fig. 10 Example of a 2-*opt edge exchange* move. (**a**) Tree $\mathcal{T}$. (**b**) Tree $\mathcal{T}^1$

A Lagrangian Heuristic for DCMST

Using the components presented before, Algorithm 1 is a Lagrangian heuristic for DCMST problem. Given a graph $G = (V, E)$, a feasible starting DCST $\mathcal{T}^*$ is computed by the heuristic in section "A Greedy Heuristic for DCST". A limit $MaxIter$ is set to the number of SG iterations. Basically, the subgradient method (section "A Subgradient Procedure for DCMST Problem") is used to update Lagrange multipliers until the stopping criterion (number of iterations or optimal solution) is met. At each iteration k, the classical Kruskal's algorithm [37] computes a Lagrangian solution $\mathcal{T}^k$ on the graph $G(\lambda^k)$: $\mathcal{T}^k \backslash MST(G(\lambda^k))$. The Lagrangian solution cost of $\mathcal{T}^k$ ($LagrCost(\mathcal{T}^k)$) gives a lower bound LB^k on the optimal solution value. If it improves LB, the new value LB^k is updated. Eventually, $\mathcal{T}^k$ is a DCST. In this case, a local search procedure (section "Improving DCSTs with a Local Search") is called to improve $\mathcal{T}^k$ if the cost $Cost(\mathcal{T}^k)$ of $\mathcal{T}^k$ in G improves UB: $\mathcal{T}^* \backslash LocalSearch(\mathcal{T}^k)$.

As mentioned before, halving α after some SG iterations without any improvement on LB guarantees the convergence of the subgradient procedure. Also, one can work with a reduced instance $G' = (V, E')$ of the problem, by considering only a subset $E' \subset E$ of ordered edges required to obtain a feasible DCST for the problem, by using the modified Kruskal's algorithm from section "A Greedy Heuristic for DCST." Moreover, more sophisticated local searches and even metaheuristics can be used instead of the local search presented in section "Improving DCSTs with a Local Search". For instance, variable neighborhood search (VNS), dynamic variable neighborhood descent (VND), and absorption function [57] have been used in [4]. As the k-opt local search procedure is based on a greedy criterion, the Lagrangian heuristic may not escape local optima. To overcome this issue, the authors in [4] suggest to integrate a VNS and a dynamic VND, as well as absorption functions of [57].

The LH presented here has been applied to a number of benchmark instances from the literature: Euclidean And-inst, Hamiltonian Ham-inst, Shrd [36], M-graph and R-graph instances, and the new Dr-inst and De-inst instances [15]. In addition,

Algorithm 1: Lagrangian heuristic [5]

Data: a graph $G = (V, E)$, a DCST $\mathcal{T}^*$, a maximum number of iterations $MaxIter$;
Result: a DCST of G;
initialization: $k \leftarrow 0$; $\lambda^0 \leftarrow \mathbf{0}$; $T^0 \leftarrow MST(G(\lambda^0))$; LB$\leftarrow LagrCost(\mathcal{T}^0)$; UB$\leftarrow Cost(\mathcal{T}^*)$; **while** *(LB<UB* **and** $k < MaxIter$*)* **do**
  compute λ^{k+1} by using Equation (7);
  set $\mathcal{T}^k \leftarrow MST(G(\lambda^k))$ and LB$^k \leftarrow LagrCost(\mathcal{T}^k)$;
  if *(LB*$^k >$ *LB)* **then**
    LB$\leftarrow$LB^k;
  end
  if *(*$\mathcal{T}^k$ *is a DCST* **and** $Cost(\mathcal{T}^k) <$*UB)* **then**
    $\mathcal{T}^* \leftarrow LocalSearch(\mathcal{T}^k)$;
    UB$\leftarrow Cost(\mathcal{T}^*)$;
  end
  $k \leftarrow k + 1$;
end
return $\mathcal{T}^*$;

the solutions obtained by LH have been used as cutoff to speed up exact methods [4, 15] for DCMST problems.

Results obtained by LH have provided new average gaps of up to 0.188 % and 6.230 % for And-inst and Ham-inst instances, respectively. Moreover, it has solved to optimality the Shrd, M-graph, and R-graph sets. After introducing the VNS, optimality was reached for 23 out of 25 And-inst instances, 4 out of 15 Ham-inst instances, and 8 out of 18 De-inst instances. Furthermore, optimal solutions were obtained for all Dr-inst instances. Thus, LH has been shown to be a powerful approach for DCMST problems.

Evolutionary Heuristic for a Generalization of BDMST

In this section, another way to handle difficult constraints is described for a generalization of the BDMST. The difficult constraints of the problem are defined on the paths of the tree rather than on the nodes as for DCMST. BDMST is defined in a connected and undirected graph $G = (V, E)$, where costs $c_{ij} \geq 0$ are associated with each edge $[i, j] \in E$. Consider $\mathcal{T}$ as a spanning tree of G. By MST definition, there is a unique path $\mathcal{P}_{ij}$ in $\mathcal{T}$ between any pair of nodes $i, j \in V$. Thus, given ρ_{ij} the number of edges in the path $\mathcal{P}_{ij}$, the diameter D of $\mathcal{T}$ is defined as $D = max\{\rho_{ij} : \forall i, j \in V, i \neq j\}$. The BDMST consists in finding an MST such that its diameter does not exceed a given $k \in \mathbb{N}^*$, i.e., $D \leq k$. This problem belongs to the NP-hard class whenever $3 < k < n - 1$ [22].

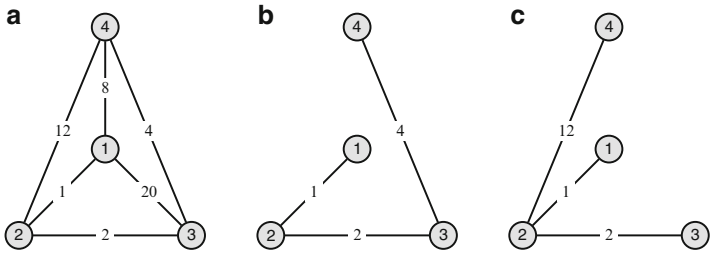

Fig. 11 Example of a tree with diameter constraints. (**a**) A graph G. (**b**) MST of G (**c**) A BDMST of G

An example is shown in Fig. 11 for $k = 2$, considering the graph given in Fig. 11a. The MST is presented in Fig. 11b. It violates the diameter since $D = 3$. The optimal BDMST solution is provided in Fig. 11c, where the longest path has two edges ($D = 2$).

Several works are found in the literature for BDMST such as mathematical formulations [25,27,53], exact methods [26,43,44], and heuristics [40,47,49]. Yet, optimality has not been proved for an important benchmark of instances proposed by [47], containing complete graphs whose size varies from 50 to 1000 vertices. Thus, an original strategy is to investigate the search space of this problem by using bi-objective approaches to compute the Pareto front. The idea is to drop the difficult constraints, i.e., diameter, and to consider them as a new criterion. Then a bi-objective optimization problem is solved. Thus the total tree cost and the diameter are minimized simultaneously. The problem remains difficult since the goal is to determine a Pareto front. However, the multiobjective strategy is interesting since every spanning tree is feasible and efficient algorithms are available to compute them. Moreover, in recent years, several advances have been done in solving multiobjective (bi-objective) problems using heuristics and metaheuristics. In addition, bi-objective heuristics do not depend on a good mathematical formulation. Another key point is that it is possible to bound the values for each criteria for problems relying on trees such as BDMST.

Bi-objective optimization for MSTs has already been investigated considering two cost objective functions. For example, a branch-and-bound was proposed in [56] and a two-phase enumeration was introduced in [48, 58]. Bi-objective metaheuristics are also used for two cost objective functions as in [6, 59]. A multiobjective evolutionary algorithm (MOEA) for the network design problem is presented in [59] to minimize the infrastructure cost and the maintenance cost, while a multiobjective greedy randomized adaptive search procedure (GRASP) is applied in [6] to find MST with two costs. Using this idea to handle difficult constraints seems to be more recent, in particular for the BDMST. Works [17, 50, 52] are dedicated to multiobjective strategies for the BDMST, referred as the bi-objective minimum diameter-cost spanning tree (bi-MDCST) problem. One of the most efficient multiobjective heuristics developed for bi-MDCST is the nondominated sorting genetic algorithm (NSGA-II), presented in the sequel.

Problem Definition

Without loss of generality, applying bi-objective heuristics for a problem $\mathcal{P}$ like the bi-MDCST implies the minimization of two objective functions $\{\min\ f_1(x), \min\ f_2(x)\ |x \in \mathcal{X}\}$. $\mathcal{X}$ is the feasible solution space of $\mathcal{P}$ and $f(x)$ is the vector of objective functions. For the bi-MDCST, $\{f_1(x), f_2(x)\}$ denote the cost and the diameter, respectively. Then a set of compromise solutions with respect to the two objective functions has to be obtained instead of a single solution as for the classical BDMST.

The concept of dominance is used to define the set of compromise solutions, usually called Pareto front. Considering two bi-MDCST solutions x and y, x *dominates* y if and only if the following conditions hold:

$$\begin{cases} f_k(x) < f_k(y) & \exists k \in \{1,2\} \quad \text{and} \\ f_k(x) \leq f_k(y) & \forall k \in \{1,2\} \end{cases} \tag{10}$$

A Pareto solution x^* is optimal whenever it is not dominated by any solution in $\mathcal{X}$. The Pareto-optimal front is composed of the Pareto-optimal solutions (nondominated). The bi-MDCST seeks a set of Pareto-optimal spanning trees $\mathcal{T}$ of G where f_1 and f_2 are simultaneously minimized. Figure 12 presents three solutions for bi-MDCST, considering the graph of Fig. 11a. The cost and the diameter for $\mathcal{T}_1$, $\mathcal{T}_2$, and $\mathcal{T}_3$ are, respectively, $\{29, 2\}$, $\{14, 3\}$, and $\{24, 2\}$. Solution $\mathcal{T}_1$ is dominated by $\mathcal{T}_3$ due to the cost. However, solutions $\mathcal{T}_2$ and $\mathcal{T}_3$ are not dominated by each other since $\mathcal{T}_2$ has a smaller cost than $\mathcal{T}_3$, while $\mathcal{T}_3$ has a smaller diameter than $\mathcal{T}_2$. One may note that solutions $\mathcal{T}_2$ and $\mathcal{T}_3$ are not Pareto-optimal because they are, respectively, dominated by the MST depicted in Fig. 11b of value {7,3} and the solution shown in Fig. 11c of values {15,2}.

For bi-MDCST, an obvious LB on the cost can be obtained by computing an MST on the graph. The diameter of this solution also provides an UB on the diameter. In addition, for complete graphs, the star is also a trivial LB on the diameter. The minimal cost of a star in the graph can provide an UB for the cost. Spanning trees of diameters $D \in \{2, 3\}$ can be computed in polynomial time as shown in [52]. Those

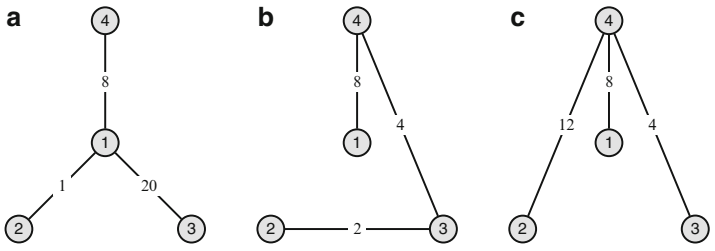

Fig. 12 Example of solutions for bi-MDCST. (**a**) Solution $\mathcal{T}_1$. (**b**) Solution $\mathcal{T}_2$ (**c**) Solution $\mathcal{T}_3$

LB on the diameter may not necessarily exist on general graphs. However, checking if a graph contains at least one such spanning tree can be done in polynomial time.

An NSGA-II for Bi-MDCST

The NSGA-II has been proposed by [19] and is one of the metaheuristics available to compute Pareto fronts for multiobjective optimization problems. This method has been applied to a number of multiobjective optimization problems for the past ten years. Moreover, it uses simple and efficient operators to manage the Pareto front convergence. Such operators are added to a classical genetic algorithm (GA), which is very popular in the scientific literature. The two special multiobjective operators are the *ranking* and the *crowding distance*. These two operators are computed for each solution and are responsible to manage the Pareto front convergence. Both the *ranking* and the *crowding distance* are associated with each solution. Given a target solution $\mathcal{T}$, the *ranking* contains the number of solutions that dominate $\mathcal{T}$. Thus, the lower the ranking, the better the solution. Fronts are then defined by solutions of same ranking. Given a front of M solutions, the *crowding distance* w is computed following Equation (11), where $sc(j)$ and $pr(j)$ are, respectively, the value which precedes j in w and the value which follows j in w. For any (nondegenerate) front with more than one solution, $z_m^{\max} \neq z_m^{\min}$. The bi-MDCST involves two objectives $m = 1, 2$; thus, $z_m^{\max}$ and $z_m^{\min}$ are the maximum and the minimum objective function values, which need to be properly normalized. The crowding distance is a kind of Manhattan distance between two solutions from the same front. Then, the higher the crowding distance is, the more distant the two solutions are. As a consequence, solutions with larger crowding distance are preferred since they belong to less covered areas.

$$w_j = \sum_{m=1}^{M} \left(\frac{z_m^{sc(j)} - z_m^{pr(j)}}{z_m^{\max} - z_m^{\min}} \right) \tag{11}$$

The algorithm performs the following steps: (1) generate the initial population (set of solutions), (2) compute and order the population using the ranking and crowding distance, (3) select the first half elements of the population, and (4) generate the remaining elements using genetic operators and go to step (2). These steps are repeated until stopping criteria are met and are detailed below. In addition, a local search can be applied to each new solution in the population, i.e., before moving to step 2. In the vocabulary of GA, the use of a local search within a GA is referred as a memetic algorithm (MA). The idea of generating all $2n$ solutions and ordering them before cutting n solutions avoids cutting good solutions.

Given a graph $G = (V, E)$, a solution for bi-MDSCT (also referred as chromosome in the GA vocabulary) can be encoded using a vector of size n containing the direct predecessor of a vertex in the tree. A population can be generated using any heuristics (greedy, randomized, etc.) or even randomly generated spanning trees

(step 1). The work [53] suggests an initial population of size $2n$ which contains (i) two specific solutions, an MST of G and a spanning tree of diameter $D = 2$ or $D = 3$, if it applies; (ii) half of the population obtained randomly; and (iii) the remaining solutions computed using a randomized version of Prim's algorithm (i.e., the vertex entering the solution is randomly selected from a list of good candidates).

As mentioned above, the *ranking* and the *crowding distance* are computed for each solution in the initial population (step 2). Then, these operators will be responsible for the selection of solutions, following a multiobjective evolutionary approach rather than directly considering their cost and diameter. The comparison between two solutions $\mathcal{T}_1$ and $\mathcal{T}_2$ with respect to their *ranking* r_1 and r_2 and their *crowding distance* w_1 and w_2 is as follows:

$$(r_1 < r_2) \text{ or } ((r_1 = r_2) \text{ and } (w_1 > w_2)) \tag{12}$$

$\mathcal{T}_1$ is said to be better than $\mathcal{T}_2$ whenever the condition (12) holds. In the following (step 2), the population is ordered. Now, the algorithm proceeds to the selection (step 3) by keeping the n best solutions for the next iteration. The n new solutions to enter the population can be generated by the crossover proposed by [11] (step 3). One advantage of this crossover is that feasibility of new solution is ensured, even for sparse graphs. The way to select the parents in the crossover can follow a classical elitist strategy, i.e., one parent from the elite set and the other one randomly selected from the remaining population. The crossover proposed by [11] works as follows: given two solutions (parents) $\mathcal{T}_1$ and $\mathcal{T}_2$, a support graph G' containing all edges from $\mathcal{T}_1$ and $\mathcal{T}_2$ is built. Then, an MST is computed on G' to obtain the new solution. Figure 13 illustrates this crossover, considering the graph G shown in Fig. 11a. The two parents $\mathcal{T}_1$ and $\mathcal{T}_2$ are, respectively, given in Fig. 13a and b. The resulting support graph G' is presented in Fig. 13c and the new individual is provided in Fig 13d, for which $Cost = 11$ and $D = 3$.

The local search iteratively performs a 2-opt move until no improvement can be done on the current solution. The 2-opt move uses the *edge drop* move previously presented. A specific feature of the 2-opt move for bi-MDCST suggested by [52] is that a move is accepted if and only if the diameter does not change and the cost

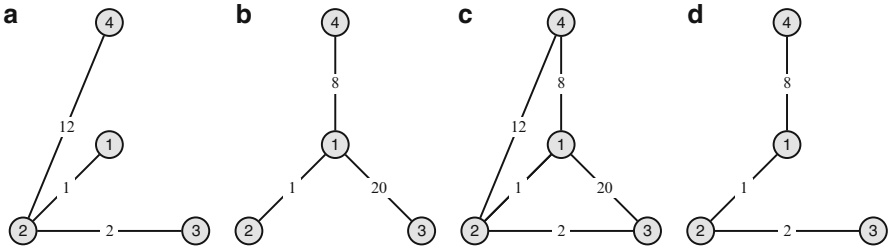

Fig. 13 Example of the crossover operator. (**a**) Solution $\mathcal{T}_1$. (**b**) Solution $\mathcal{T}_2$. (**c**) Graph G'. (**d**) MST of G'

is reduced. This means the local search will reach a local optima having the same diameter as the current solution.

The main insights obtained in [17, 52] by applying the NSGA-II for bi-MDSCT are summarized below. Three main sets of instances available in the literature for the BDMST were used in the experiments. They were proposed by [25, 40, 47], respectively. In addition, two test sets were proposed by [52] containing 9 and 11 instances, named Hamiltonian cycle and Hamiltonian path, respectively. Results produced by NSGA-II were compared with available optimal values for the BDMST from [26, 53]. The NSGA-II for bi-MDCST manages to find some optimal diameter values of the BDMST and it is very close in the other cases. The computational time does not exceed the computational time spent by dedicated and sophisticated methods for the BDMST. The Pareto-optimal fronts for the test sets used in [40, 52] are published on the site http://di.uern.br/dario/bi-mdcst-problem/.

Some interesting characteristics have been observed: (i) a significant difference in tree cost occurs in the Pareto-optimal front between solutions with $D = 2$ and $D = 3$; (ii) since the instances from [40] have edges with similar cost, there are several MSTs with different diameters; (iii) some diameters are not interesting because the best corresponding solutions are dominated; and (iv) for the majority of instances from the sets mentioned above, up to 15 target diameters exist. The multiobjective approach has shown to be effective. It works consistently well and can provide useful information about the search space and the Pareto front for bi-MDCST problems.

Conclusion

Trees and forests are very rich topics, both in terms of theoretical and practical issues. Integrating good ingredients in sophisticated heuristics is the first step to produce good, competitive heuristics and even find new better results for NP-hard problems relying on these structures. In this context, some basic components for trees and forests are summarized in this chapter and two ways of addressing difficult constraints (degree and diameter) are presented, with the hope it will inspire new and fruitful research on these topics.

In terms of constructive heuristics, two basic ways have been addressed in the literature: tree expansion and tree fusion. Concerning the improvement heuristics, two main classes appear: restoration and local search. Some of these strategies use tree properties such as the local searches based on k-opt moves. In fact, by definition of a tree, removing an edge from a tree results in two connected components and inserting an edge in a tree obviously creates a cycle. Thus even an 1-opt can be considered two ways, either by first removing an edge or by first adding an edge. Even if they basically are 1-opt moves, these may impact the way the search space is explored.

Some key points are also provided in order to represent a tree in terms of data structures. Encoding and decoding a tree can be done directly by means of predecessor vectors indexed on the vertex or edge sets. It can also be done indirectly

by using random keys. Additional data structures are also required to handle difficult constraints such as degrees and diameter. The choice of data structures will mainly impact the algorithm complexity. As a consequence, careful choices can save running time.

Two different strategies to address difficult constraints have been presented. The first, more classical, relies on a Lagrangian relaxation in which the constraints are relaxed and integrated into the objective function using the Lagrangian penalties. The second way consists of dropping the constraints, considering them as a new criterion. The resulting bi-objective problem is then solved by multiobjective approaches. The former presents the advantages of being simple to implement; it obtains solutions of good quality in a reasonable computational time, and lower and upper bounds are available at each iteration. A drawback is that it strongly depends on the mathematical formulation and the chosen decomposition. The latter is far less investigated in the literature, in particular for addressing difficult constraints. It has the following advantages: it provides additional information about the search space, and the computational time to obtain the Pareto front is very close to the time spent applying sophisticated dedicated methods. Moreover, limits in the search space can be computed and dominance rules can speed up the inspection of the solutions. A drawback is that the problem remains difficult since it may involve solving several NP-hard problems. However, facing NP-hard problems from a multiobjective perspective remains an interesting direction for further research.

Cross-References

- ▸ Evolutionary Algorithms
- ▸ Multiobjective Optimization
- ▸ Variable Neighborhood Descent
- ▸ Variable Neighborhood Search

References

1. Achuthan NR, Caccetta L, Caccetta PA, Geelen JF (1994) Computational methods for the diameter restricted minimum weight spanning tree problem. Australas J Comb 10:51–71
2. Ahuja RK, Magnanti TL, Orlin JB (1993) Network flows – theory, algorithms and applications. Prentice Hall, Upper Saddle River
3. Álvarez-Miranda E, Ljubić I, Toth P (2013) Exact approaches for solving robust prize-collecting Steiner tree problems. Eur J Ope Res 229(3):599–612
4. Andrade R, Freitas AT (2013) Disjunctive combinatorial branch in a subgradient tree algorithm for the DCMST problem with VNS-Lagrangian bounds. Electron Notes Discrete Math 41(0):5–12
5. Andrade R, Lucena A, Maculan N (2006) Using Lagrangian dual information to generate degree constrained spanning trees. Discrete Appl Math 154(5):703–717
6. Arroyo JEC, Vieira PS, Vianna DS (2008) A GRASP algorithm for the multi-criteria minimum spanning tree problem. Ann Oper Res 159:125–133

7. Barahona F, Anbil R (2000) The volume algorithm: producing primal solutions with the subgradient method. Math Program 87:385–399
8. Bean JC (1994) Genetic algorithms and random keys for sequencing and optimization. ORSA J Comput 6(2):154–160
9. Beasley JE (1993) Lagrangean relaxation. In: Reeves C (ed) Modern heuristic techniques for combinatorial problems. Wiley, New York, pp 243–303
10. Camerini PM, Galbiati G, Maffioli F (1980) Complexity of spanning tree problems: part I. Eur J Oper Res 5(5):346–352
11. Carrano EG, Fonseca CM, Takahashi RHC, Pimenta LCA, Neto OM (2007) A preliminary comparison of tree encoding schemes for evolutionary algorithms. In: IEEE international conference on systems, man and cybernetics, ISIC, Montreal, pp 1969–1974
12. Cayley A (1889) A theorem on trees. Q J Pure Appl Math 23:376–378
13. Cerrone C, Cerulli R, Raiconi A (2014) Relations, models and a memetic approach for three degree-dependent spanning tree problems. Eur J Oper Res 232(3):442–453
14. Cormen TH, Leiserson CE, Rivest R, Stein C (2009) Introduction to algorithms, 3rd edn. The MIT Press, Cambridge
15. da Cunha AS, Lucena A (2007) Lower and upper bounds for the degree-constrained minimum spanning tree problem. Networks 50(1):55–66
16. da Cunha AS, Lucena A, Maculan N, Resende MGC (2009) A relax-and-cut algorithm for the prize-collecting Steiner problem in graphs. Discrete Appl Math 157(6):1198–1217
17. de Sousa EG, Santos AC, Aloise DJ (2015) An exact method for solving the bi-objective minimum diameter-cost spanning tree problem. RAIRO Oper Rech 49:143–160
18. de Souza MC, Duhamel C, Ribeiro CC (2003) A GRASP heuristic for the capacitated minimum spanning tree problem using a memory-based local search strategy. In: Resende M, de Sousa J (eds) Metaheuristics: computer decision-making. Kluwer, Boston, pp 627–658
19. Deb K, Pratap A, Agarwal S, Meyarivan T (2002) A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans Evol Comput 6(2):182–197
20. Duhamel C, Gouveia L, Moura P, Souza M (2008) Models and heuristics for a minimum arborescence problem. Networks 51(1):34–47
21. Fisher ML (1981) The Lagrangian relaxation method for solving integer programming problems. Manag Sci 27:1–18
22. Garey MR, Johnson DS (1979) Computers and intractability: a guide to the theory of NP-Completeness. W.H. Freeman, New York
23. Gottlieb J, Julstrom BA, Raidl GR, Rothlauf F (2001) Prüfer numbers: a poor representation of spanning trees for evolutionary search. In: Spector L, Goodman ED et al (eds) Proceedings of the genetic and evolutionary computation conference (GECCO-2001). Morgan Kaufmann, San Francisco, pp 343–350
24. Gouveia L, Leitner M, Ljubić I (2012) On the hop constrained Steiner tree problem with multiple root nodes. In: Ridha Mahjoub A, Markakis V, Milis I, Paschos VangelisTh (eds) Combinatorial optimization. Volume 7422 of lecture notes in computer science. Springer, Berlin/Heidelberg, pp 201–212
25. Gouveia L, Magnanti TL (2003) Network flow models for designing diameter-constrained minimum-spanning and Steiner trees. Networks 41:159–173
26. Gouveia L, Simonetti L, Uchoa E (2011) Modeling hop-constrained and diameter-constrained minimum spanning tree problems as Steiner tree problems over layered graphs. Math Program 128:123–148
27. Gruber M, Raidl GR (2005) A new 0-1 ILP approach for the bounded diameter minimum spanning tree problem. In: Hansen P, Mladenovic N, Pérez JAM, Batista BM, MorenoVega JM (eds) The 2nd international network optimization conference, Lisbon, vol 1, pp 178–185
28. Guignard M (2003) Lagrangean relaxation. In: Cerdá MAL, Jurado IG (eds) Trabajos de Investigación Operativa, vol 11, chapter2. Sociedad de Estadística e Investigatión Operativa, Madrid, pp 151–228

29. Held M, Karp RM (1970) The travelling-salesman problem and minimum spanning trees. Oper Res 18:1138–1162
30. Held M, Karp RM (1971) The travelling-salesman problem and minimum spanning trees: part II. Math Program 1:6–25
31. Held MH, Wolfe P, Crowder HD (1974) Validation of subgradient optimization. Math Program 6:62–88
32. Henschel R, Leal-Taixé L, Rosenhahn B (2014) Efficient multiple people tracking using minimum cost arborescences. In: Jiang X, Hornegger J, Koch R (eds) Pattern recognition lecture notes in computer science. Springer, Cham, pp 265–276
33. Ho J-M, Lee DT, Chang C-H, Wong K (1991) Minimum diameter spanning trees and related problems. SIAM J Comput 20(5):987–997
34. Hwang FK, Richards DS, Winter P (1992) The Steiner tree problem. Annals of discrete mathematics, vol 53. Elsevier, Amsterdam
35. Kasperski A, Zieliński P (2011) On the approximability of robust spanning tree problems. Theor Comput Sci 412(4–5):365–374
36. Krishnamoorthy M, Ernst AT, Sharaiha YM (2001) Comparison of algorithms for the degree constrained minimum spanning tree. J Heuristics 7(6):587–611
37. Kruskal JB (1956) On the shortest spanning subtree of a graph and the traveling salesman problem. Am Math Soci 7:48–50
38. Leitner M, Ljubic I, Sinnl M (2015) A computational study of exact approaches for the bi-objective prize-collecting Steiner tree problem. INFORMS J Comput 27(1):118–134
39. Li Y, Yu J, Tao D (2014) Genetic algorithm for spanning tree construction in P2P distributed interactive applications. Neurocomputing 140(0):185–192
40. Lucena A, Ribeiro C, Santos AC (2010) A hybrid heuristic for the diameter constrained minimum spanning tree problem. J Glob Optim 46:363–381
41. Magnanti TL, Wolsey LA (1995) Optimal trees. In: Monma CL, Ball MO, Magnanti TL, Nemhauser GL (eds) Network models. Volume 7 of handbooks in operations research and management science. Elsevier, Amsterda, pp 503–615
42. Martinez LC, da Cunha AS (2012) A parallel Lagrangian relaxation algorithm for the min-degree constrained minimum spanning tree problem. In: Mahjoub AR, Markakis V, Milis I, Paschos V (eds) Combinatorial optimization. Volume 7422 of lecture notes in computer science. Springer, Berlin/Heidelberg, pp 237–248
43. Noronha TF, Ribeiro CC, Santos AC (2010) Solving diameter-constrained minimum spanning tree problems by constraint programming. Int Trans Oper Res 17(5):653–665
44. Noronha TF, Santos AC, Ribeiro CC (2008) Constraint programming for the diameter constrained minimum spanning tree problem. Electron Notes Discrete Math 30:93–98
45. Prüfer H (1918) Neuer beweis eines satzes über permutationen. Archiv für Mathematik und Physik 27:141–144
46. Raidl GR, Julstrom BA (2003) Edge sets: an effective evolutionary coding of spanning trees. IEEE Trans Evol Comput 7(3):225–239
47. Raidl GR, Julstrom BA (2003) Greedy heuristics and an evolutionary algorithm for the bounded-diameter minimum spanning tree problem. In: Proceedings of the 18th ACM symposium on applied computing, Melbourne, pp 747–752
48. Ramos RM, Alonso S, Sicilia J, Gonzalez C (1998) The problem of the optimal biobjective spanning tree. Eur J Oper Res 111(3):617–628
49. Requejo C, Santos E (2009) Greedy heuristics for the diameter-constrained minimum spanning tree problem. J Math Sci 161:930–943
50. Saha S, Kumar R (2011) Bounded-diameter MST instances with hybridization of multi-objective ea. Int J Comput Appl 18(4):17–25
51. Santos AC, Duhamel C, Belisário LS, Guedes LM (2012) Strategies for designing energy-efficient clusters-based WSN topologies. J Heuristics 18(4):657–675
52. Santos AC, Lima DR, Aloise DJ (2014) Modeling and solving the bi-objective minimum diameter-cost spanning tree problem. J Glob Optim 60(2):195–216

53. Santos AC, Lucena A, Ribeiro CC (2004) Solving diameter constrained minimum spanning tree problem in dense graphs. Lect Notes Comput Sci 3059:458–467
54. Shangin RE, Pardalos PM (2014) Heuristics for minimum spanning k-tree problem. Procedia Comput Sci 31(0):1074–1083
55. Silva RMA, Silva DM, Resende MGC, Mateus GR, Gonçalves JF, Festa P (2014) An edge-swap heuristic for generating spanning trees with minimum number of branch vertices. Optim Lett 8(4):1225–1243
56. Sourd F, Spanjaard O (2008) A multiobjective branch-and-bound framework: application to the biobjective spanning tree problem. INFORMS J Comput 20(3):472–484
57. Souza MC, Martins P (2008) Skewed VNS enclosing second order algorithm for the degree constrained minimum spanning tree problem. Eur J Oper Res 191(3):677–690
58. Steiner S, Radzik T (2008) Computing all efficient solutions of the biobjective minimum spanning tree problem. Comput Oper Res 35(1):198–211
59. Zhou G, Gen M (1999) Genetic algorithm approach on multi-criteria minimum spanning tree problem. Eur J Oper Res 114:141–152