This is a pre print version of the following article:

Solution of minimum spanning forest problems with reliability constraints / Ahani, I.K., Salari, M., Hosseini, S.M., Iori, M.. - In: COMPUTERS & INDUSTRIAL ENGINEERING. - ISSN 0360-8352. - 142:(2020), pp. 1-28. [10.1016/j.cie.2020.106365]

Terms of use:

The terms and conditions for the reuse of this version of the manuscript are specified in the publishing policy. For all terms of use and more information see the publisher's website.

22/11/2026 15:25

(Article begins on next page)

arXiv:submit/2834441 [math.OC] 7 Sep 2019

Solution of minimum spanning forest problems with reliability constraints

Ida Kalateh Ahani[(1)], Majid Salari[(1)]*, Seyed Mahmoud Hosseini[(1)], Manuel Iori[(2)]

(1) Department of Industrial Engineering, Ferdowsi University of Mashhad, Mashhad, Iran

{i_ahani,msalari,sm_hosseini}@um.ac.ir

(2) Department of Sciences and Methods for Engineering, University of Modena and Reggio Emilia, Italy

manuel.iori@unimore.it

Abstract

We propose the reliability constrained k-rooted minimum spanning forest, a relevant optimization problem whose aim is to find a k-rooted minimum cost forest that connects given customers to a number of supply vertices, in such a way that a minimum required reliability on each path between a customer and a supply vertex is satisfied and the cost is a minimum. The reliability of an edge is the probability that no failure occurs on that edge, whereas the reliability of a path is the product of the reliabilities of the edges in such path. The problem has relevant applications in the design of networks, in fields such as telecommunications, electricity and transports. For its solution, we propose a mixed integer linear programming model, and an adaptive large neighborhood search metaheuristic which invokes several shaking and local search operators. Extensive computational tests prove that the metaheuristic can provide good quality solutions in very short computing times.

Keywords: Networks, Minimum Spanning Forest, Reliability, Adaptive Large Neighborhood Search

1 Introduction

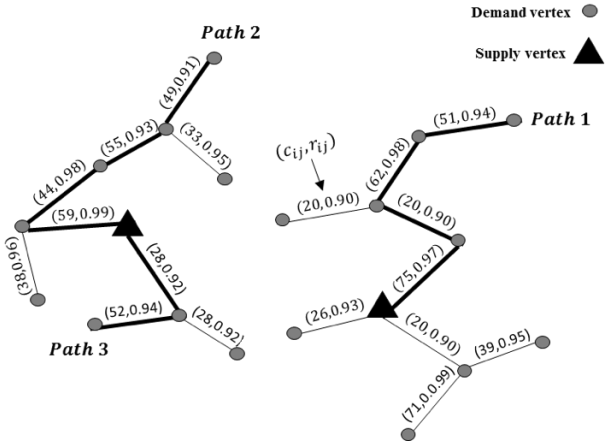
The *minimum spanning tree* (MST) is one of the most celebrated problems in the field of combinatorial optimization. It is defined on a connected, undirected and weighted graph $G = (V, E)$, where V is the set of vertices and E is the set of edges. An MST of G is a tree composed of a subset of edges of E spanning all the vertices of V and having minimum total weight. The MST has several applications, including design of computer and communication networks, picture processing, automatic speech recognition, clustering and classification problems (see, e.g., Wu and Chao 2004). Several polynomial time algorithms have been proposed to solve it, starting from Kruskal (1956) and Prim (1957).

*Department of Industrial Engineering, Ferdowsi University of Mashhad, P.O. Box 91775-1111, Mashhad, Iran. TEL: +98 051 38805113, e-mail: msalari@um.ac.ir

A forest consists of a set of mutually disjoint trees, and a *spanning forest* (SF) is a forest that spans all vertices of the graph. When the graph is connected, an SF is a forest where each vertex of the graph is included in one of the trees (see, e.g., Mekking and Volgenant 2010; Da Cunha *et al.* 2011). When instead the graph is not connected but consists of several connected components, an SF is a sub-graph containing a spanning tree of each component (see, e.g., Kang and Bader 2009; Nobari *et al.* 2012). A *k-rooted spanning forest* is an SF with k disjoint trees, where each tree has its own *root*. These problems have different applications in designing supply chain networks, image processing (Tarabalka *et al.* 2010; K. Bernard *et al.* 2012), political districting (Yamada 2009) and designing communication networks (Yamada *et al.* 1996), among others.

In this paper, we study the *reliability constrained k-rooted minimum spanning forest* (RCKRMSF). Essentially, in the RCKRMSF we are given two sets of vertices, one including demand vertices and the other supply vertices. Demand vertices can be seen as customers and should be connected by means of a path in the graph to one of the supply vertices. Each edge (i, j) of the graph is associated with a cost c_{ij} and a reliability r_{ij}. The reliability of an edge is the probability that no failure occurs on that edge, and the reliability of a path is the product of the reliabilities of all edges included in the path. The goal of the RCKRMSF is to construct a k-rooted minimum-cost SF that connects all customers to one or more supply vertices, and for which any path between a customer and its supply vertex satisfies a minimum required reliability. An RCKRMSF example is depicted in Figure 1, where 2 supply and 18 demand vertices are rooted using an SF with 2 trees. Costs and reliabilities are reported close to each arc. Three paths are highlighted: Path 1 has cost 75+20+62+51=208 and reliability 0.97×0.90×0.98×0.94≈0.80; similarly, path 2 has cost 207 and reliability 0.82 and path 3 cost 80 and reliability 0.86. It is easy to check that the solution guarantees minimum reliability 0.80 on all paths.

Figure 1: An illustrative example of the RCKRMSF problem.



The RCKRMSF is of interest because models many real-world applications in different fields. Re-

liability of paths is indeed a crucial component of systems arising in fields such as telecommunication, electricity and transport networks, just to cite some (see, e.g., Edrissi *et al.* 2015). For example, when broadcasting information or data from servers to terminal nodes of a network, it is often needed to guarantee that the data is received with at least a pre-specified reliability. Similarly, in radio communication networks, in case of a critical situation, the reliability of communication among central departments and stations should be greater than a minimum required level. Also, when a disaster occurs, reliability of delivering services to people in declared areas should be taken into account besides the minimization of delivery costs (Yücel *et al.* 2018). In all these examples, the design a network has to take into account both costs and minimum reliability, and could be modeled as an RCKRMSF. In general, problems where reliability matters in distributing services, data, information, or in general a given commodity from supply vertices to customers can be viewed as instances of the RCKRMSF.

Despite its large number of applications, to the best of our knowledge the RCKRMSF has not been previously addressed in the literature. In this paper, we first formally model the problem by using *mixed integer linear programming* (MILP). As the MILP model is effective only on small-scale instances, we also propose a metaheuristic, based on the concept of *adaptive large neighborhood search* (ALNS) and enriched with several local search routines and shaking operators. We prove the computational effectiveness of the proposed ALNS on a large set of instances. We also test the ALNS on other simplified versions of the RCKRMSF that have been tackled in the optimization literature, obtaining interesting insights.

The paper is organized as follows. A concise literature review of the wide area of SF applications is provided in Section 2. A formal problem description and a MILP model are presented in Section 3. The details of the proposed ALNS are given in Section 4. Extensive computational tests on the RCKRMSF and other problem variants are discussed in Section 5, and conclusions are drawn in Section 6.

2 Brief literature review

We are not aware of previous works focused on the RCKRMSF, but we found a large number of works that address different constrained variations of the classical MST that are related to the RCKRMSF.

The *capacitated MST* (CMST) is an MST generalization in which each vertex i of the graph, with the exception of a given supply vertex, has a non-negative demand d_i. The objective is to find an MST such that the sum of the demands in each sub-tree, rooted at the supply vertex, does not exceed a preset capacity value. The CMST is interesting because models many real-world situations and also because it can serve as a lower bound for capacitated vehicle routing (see, e.g., Toth and Vigo 2014) and pickup-and-delivery (see, e.g., Bruck and Iori 2017) problems. It has been tackled by many exact

algorithms, including polyhedral approaches (Gouveia and Lopes 2005) and branch-and-cut-and-price algorithms (Uchoa *et al.* 2008). In terms of heuristics, good computational results were obtained with a biased random-key genetic algorithm by Ruiz *et al.* (2015) and with a quick greedy heuristic by Kritikos and Ioannou (2017).

Narula and Ho (1980) proposed the *degree constrained minimum spanning tree problem*, an MST variation in which a threshold on the maximum number of edges incident to each vertex is imposed. The problem models a number of applications, including the branching of cables in offshore wind farms (Klein *et al.* 2015). It has been addressed with several exact and heuristic techniques, such as variable neighborhood search (de Souza and Martins 2008), branch-and-cut (Martinez and da Cunha 2014) and branch-and-cut-and-price (Bicalho *et al.* 2016). An analysis of problem variants as well as new MILP formulations have been recently presented by Dias *et al.* (2017).

Achuthan *et al.* (1992) studied the *bounded diameter MST* (BDMST). The objective of the BDMST is to find an MST such that the number of edges between any pair of vertices does not exceed a given input value. A relevant BDMST variation is the *hop constrained MST* (HCMST), where each path between a designed root vertex and any other vertex should not contain more than δ edges (hops). Hop constraints can be seen as a first attempt to guarantee a certain level of service with respect to some performance constraints such as reliability. Indeed, if a certain reliability value β is associated with each edge of the graph, then limiting the number of edges to δ has the effect of maintaining a minimum required level of reliability β^{δ} (see, e.g., Gouveia 1995; Leitner 2016).

More elaborated constraints are adopted in the *(rooted) delay constrained MST* (DCMST), also known as the *(rooted) distance constrained MST*. The DCMST is a generalization of the HCMST in which a delay l_{ij} is associated with each edge $\{i, j\} \in E$, and the sum of all delays in a path should not be greater than a pre-specified threshold L (see, e.g., Salama *et al.* 1997; Gouveia *et al.* 2008).

Another aspect that is relevant for the RCKRMSF is that the optimized solution is divided into clusters, at most one for each supply vertex. Clustering has been intensively studied in the area of network design. Here we only mention the related *generalized MST* (GMST), originally proposed by Myung *et al.* (1995). The GMST calls for finding an MST that either contains exactly one vertex for each cluster (as in, e.g., Myung *et al.* 1995; Pop *et al.* 2018), or alternatively contains at least one vertex for each cluster (as in Dror *et al.* 2000).

As previously mentioned, reliability is the probability of failure-free performance of a system for a specified duration and under stated conditions. There are many measurements for the reliability of a network, such as two-terminal, k-terminal and all-terminal reliability. Two-terminal reliability is defined as the reliability between two specified vertices, whereas the reliability of communication between every

pair of vertices in the network is called all-terminal reliability (see, e.g., Koide *et al.* 2001; Srivaree-ratana *et al.* 2002). A general case, covering the two previous cases, is the k-terminal reliability, in which minimum reliability in the designed network must be respected among all pairs of a specified subset of k vertices (see, e.g., Canale *et al.* 2013; Roy *et al.* 2016; Heidarzadeh *et al.* 2018).

In designing communication networks, both reliability and installation costs are of great importance, so many researchers have taken into account both these aspects when developing algorithms or models to design such networks. In general, designing networks with the presence of budget and reliability constraints can be achieved in two different ways: the objective can be the cost minimization subject to minimum reliability guarantee, or the reliability maximization subject to a maximum budget limit. In our work, we consider cost minimization subject to minimum reliability. For the literature on this type of problems, we mention the branch-and-bound algorithm by Jan *et al.* (1993), the method based on cross entropy by Altiparmak and Dengiz (2009), and the ant colony optimization approaches by Watcharasit-thiwat and Wardkein (2009) and Dengiz *et al.* (2010). For what concerns instead reliability maximization, we mention the simulated annealing by Atiqullah and Rao (1993), the hill-climbing, simulated annealing and genetic algorithms by Altiparmak *et al.* (2003), the so-called shrinking and searching algorithm by Shao *et al.* (2005), and the artificial neural network by Dash *et al.* (2012). Bi-objective models have been also studied to take care of the cost and reliability components in an integrated manner. Among these, we cite the genetic algorithms by Azaron *et al.* (2009) and Konak and Smith (2011). We also mention the recent label setting algorithm by Santos *et al.* (2018), which deals with multiple objectives and could be adapted to take care of reliability.

We refer readers interested in further pointers to the literature to the surveys by Ball *et al.* (1995) on network reliability, Bazlamaçci and Hindi (2001) and Nemani and Ahuja (2011) on MST algorithms, and de Almeida *et al.* (2015) on multi-objective models for reliability.

3 Problem formulation and mathematical model

In this section, we provide a formal RCKRMSF description and a MILP model. The model is based on a multicommodity network flow (see, e.g., Magnanti and Wolsey 1995), where each commodity is assigned to a given customer and its flow is used to model the path connecting the customer to one of the supply vertices. To this aim, we introduce a directed graph $G' = (V, A)$, obtained by replacing each edge $\{i, j\} \in G$ with two directed arcs (i, j) and (j, i). The vertex set is partitioned as $V = S \cup D$ and the arc set is defined as $A = \{(i, j) : i, j \in V\}$. Sets $S = \{1, 2, \ldots, m\}$ and $D = \{m + 1, m + 2, \ldots, n\}$ represent the supply and demand vertices, respectively. Two positive values, namely the cost c_{ij} and the reliability

r_{ij}, are associated with each arc $(i,j) \in A$. Recall that the reliability of a path is the product of the reliabilities of all arcs in the path, and let α be a minimum required level of reliability. The objective of the RCKRMSF is to construct a k-rooted minimum-cost SF of the vertices in V, such that each demand vertex (also denoted as customer) is connected by a path in the SF to exactly one of the supply vertices, and by ensuring that each such path has reliability at least equal to α. Without loss of generality, to ensure that a feasible solution exists we suppose that each demand vertex $i \in D$ can be linked to at least one source vertex $j \in S$ by a direct arc having reliability $r_{ij} \geq \alpha$.

The MILP model that we introduce makes use of two families of binary variables and a family of continuous non-negative variables, namely:

$$y_i^s = \begin{cases} 1 & \text{if customer } i \text{ is allocated to supply vertex } s, \\ 0 & \text{otherwise} \end{cases} \qquad \forall i \in D, s \in S,$$

$$x_{ij}^s = \begin{cases} 1 & \text{if arc } (i,j) \text{ belongs to a path originated from supply vertex } s, \\ 0 & \text{otherwise} \end{cases} \qquad \forall (i,j) \in A, s \in S,$$

$$f_{ij}^h = \text{flow of commodity } h \text{ along arc } (i,j) \qquad \forall (i,j) \in A, h \in D.$$

A commodity is assigned to each customer $h \in D$, and the flow variables f are used to build paths that start at one of the supply vertices and end at the customers. In particular, a unit of flow of each commodity h is imposed to leave a supply vertex and reach h. Our model is then as follows:

$$(\text{RCKRMSF}) : \min \sum_{s \in S} \sum_{(i,j) \in A} c_{ij} x_{ij}^s \tag{1}$$

$$\sum_{s \in S} \sum_{(i,j) \in A} x_{ij}^s = n - m \tag{2}$$

$$\sum_{s \in S} (x_{ij}^s + x_{ji}^s) \leq 1 \qquad \forall \{i,j\} \in E, \tag{3}$$

$$\sum_{s \in S} y_i^s = 1 \qquad \forall i \in D, \tag{4}$$

$$y_s^s = 1 \qquad \forall s \in S, \tag{5}$$

$$x_{ij}^s \leq y_i^s \qquad \forall (i,j) \in A : i \in D, s \in S, \tag{6}$$

$$x_{ij}^s \leq y_j^s \qquad \forall (i,j) \in A : j \in D, s \in S, \tag{7}$$

$$\sum_{j \in D} f_{sj}^h = y_h^s \qquad \forall s \in S, h \in H, \tag{8}$$

$$\sum_{i \in V} f_{ih}^h - \sum_{j \in D} f_{hj}^h = 1 \qquad \forall h \in H, \tag{9}$$

$$\sum_{i \in V} f_{ij}^h - \sum_{i \in D} f_{ji}^h = 0 \qquad \forall h \in H, j \in D, j \neq h, \tag{10}$$

$$f_{ij}^h \leq \sum_{s \in S} x_{ij}^s \qquad \forall h \in H, (i,j) \in A, \tag{11}$$

$$\prod_{(i,j) \in A} r_{ij}^{f_{ij}^h} \geq \alpha \qquad \forall t_h \in D, \tag{12}$$

$$x_{ij}^s \in \{0,1\} \qquad \forall (i,j) \in A, s \in S, \tag{13}$$

$$y_i^s \in \{0,1\} \qquad \forall i \in D, s \in S, \tag{14}$$

$$f_{ij}^h \geq 0 \qquad \forall (i,j) \in A, h \in H. \tag{15}$$

The objective function (1) minimizes the total cost of the SF. Constraint (2) forces the SF to contain exactly $n - m$ arcs. Constraints (3) guarantee that each edge $\{i,j\} \in E$ is assigned to at most one supply vertex and carries flow in at most one direction. Constraints (4) impose that each customer is assigned to exactly one supply vertex. Constraints (5) simply require that a supply vertex is allocated to itself. Constraints (6) and (7) ensure that if arc (i,j) is assigned to a path originating from $s \in S$, then both vertices i and j are allocated to s. Constraints (8)–(10) are used to impose flow conservation, in particular: the flow of commodity h leaving supply vertex s is set to be equal to y_h^s; the difference between the amount of commodity j entering and leaving a customer h is set to be 1 when $j = h$, and 0 when $j \neq h$. Constraints (11) link together x and y variables, by ensures that if a flow is sent along an arc, then such arc must be included in the solution. Constraints (12) impose the minimum reliability level α on each path in the solution. Then, constraints (13)–(15) define the domains of the variables.

Constraints (12) are non-linear, but can be linearized by using the logarithmic function (as in, e.g., Woolston and Albin 1988). Essentially, we can replace the left-hand side and the right-hand side of the constraint by their logarithms, obtaining:

$$\sum_{(i,j) \in A} f_{ij}^h \ln(r_{ij}) \geq \ln(\alpha) \qquad \forall h \in D. \tag{16}$$

The resulting MILP model, composed of (1)–(11) and (13)–(16), has been solved by means of a commercial solver, and the results that have been obtained are provided in Section 5 below.

4 Adaptive Large Neighborhood Search

The *large neighborhood search* (LNS) algorithm, originally proposed by Shaw (1998), attempts at improving a given solution by alternately destroying and repairing it (Mancinia and Steccab (2018)). The ALNS is an extension of the LNS that has been first proposed by Ropke and Pisinger (2006) and since then has obtained successful results for a large variety of optimization problems (Palomo-Martínez *et al.* 2017; Alinaghian and Shokouhi 2018). The ALNS allows the use of multiple destroy and repair algorithms

within the same search, and adopts an adaptive selection mechanism to decide which operators should be used. Essentially, at each iteration a given part of the solution is destroyed and repaired by applying the appropriate destroy and repair rules, and these rules are selected on the basis of given probabilities that depend on their performance during the search process.

In this section, we propose a heuristic algorithm for the RCKRMSF that is based on the ALNS paradigm and follows the pseudocode given in Algorithm 1. We are given a set of n_1 (resp. n_2) local search (resp. shaking) procedures. A weight w_{ls}^k (resp. w_{sh}^k) is associated with each local search (resp. shaking) procedure k, and is initially set to 1 for all procedures. With each local search (resp. shaking) procedure k, we also associate a score π_{ls}^k (resp. π_{sh}^k) representing the performance of the procedure during the search process.

Our ALNS contains a given number ϕ_1 of global iterations, called segments for short, each consisting of ϕ_2 inner iterations. At the beginning of each segment, all scores are set to zero. Then, at each iteration, a local search procedure i and a shaking procedure j are chosen by the roulette-wheel mechanism with probability $p_i = w_{ls}^i / \sum_{k=1}^{n_1} w_{ls}^k$ and $p_j = w_{sh}^j / \sum_{k=1}^{n_2} w_{sh}^k$, respectively. A new solution is obtained by applying first the shaking and then the local search on a current solution. Then, the scores are updated as follows: if the cost of the new solution is better than that of the incumbent, the scores of the involved local search and shaking procedures are augmented by ζ_1; if the cost of the new solution is better than that of the previous temporary solution but not better than that of the incumbent, the scores are augmented by ζ_2; if the new solution is non-improving but accepted by a *Simulated Annealing* (SA) criterion, the scores are increased by ζ_3. Let $Cost(X)$ be the cost of solution X. The SA criterion works as follows: given a current solution $CurrentSolution$, a neighbor solution $TempSolution$ is always accepted if $Cost(TempSolution) < Cost(CurrentSolution)$; otherwise, it is accepted with probability $e^{(Cost(TempSolution)-Cost(CurrentSolution))/\theta}$, in which θ is the current temperature. The initial temperature is set to θ_0 and is decreased gradually, by performing the update $\theta_{new} = \lambda\theta_{old}$ where $0 < \lambda < 1$ is the cooling rate. In addition, the final temperature is set to ϵ.

When a segment ends, the weights associated with the local search and shaking procedures are updated according to the scores obtained during the last ϕ_2 iterations using the following equation:

$$w_\mu^k = \begin{cases} w_\mu^k & \text{if } \gamma_\mu^k = 0, \\ (1-\eta)w_\mu^k + \eta\pi_\mu^k/\gamma_\mu^k & \text{otherwise.} \end{cases} \qquad \mu = sh, ls \tag{17}$$

In (17), $\eta \in [0,1]$ is an input parameter called reaction factor, whereas γ_{ls}^k (resp. γ_{sh}^k) gives the number of times a local search (resp. shaking) procedure k has been selected during the ϕ_2 inner iterations of the last segment.

Inputs: θ: initial temperature; ϵ: final temperature; λ: cooling rate; η: reaction factor; n_1: number of local search procedures; n_2: number of shaking procedures;
Output: *BestSolution*;
TempSolution = *CurrentSolution* = *BestSolution* = *Initialization*();
for each *local search* $l \in \{1, 2, \cdots, n_1\}$ set $w_{ls}^l = 1$;
for each *shaking procedure* $k \in \{1, 2, \cdots, n_2\}$ set $w_{sh}^k = 1$;
for ($Segment_{iter} = 1$ ***to*** ϕ_1) **do**
  for each *local search* $l \in \{1, 2, \cdots, n_1\}$ set $\pi_{ls}^l = 0$ **and** $\gamma_{ls}^l = 0$;
  for each *shaking procedure* $k \in \{1, 2, \cdots, n_2\}$ set $\pi_{sh}^k = 0$ **and** $\gamma_{sh}^k = 0$;
  for ($Local_{iter} = 1$ ***to*** ϕ_2) **do**
    Local search = choose a procedure i using roulette wheel with weights w_{ls}^l ($l \in \{1, 2, \cdots, n_1\}$);
    Shaking = choose a procedure j using roulette wheel with weights w_{sh}^k ($k \in \{1, 2, \cdots, n_2\}$);
    γ_{ls}^i += 1; γ_{sh}^j += 1;
    TempSolution = *Shaking*(*TempSolution*);
    while (*TempSolution* can be improved) **do**
      TempSolution = *Local search*(*TempSolution*);
    end
    if (*Cost*(*TempSolution*) < *Cost*(*BestSolution*)) **then**
      CurrentSolution = *BestSolution* = *TempSolution*;
      π_{ls}^i += ζ_1; π_{sh}^j += ζ_1;
    end
    else if (*Cost*(*TempSolution*) < *Cost*(*CurrentSolution*)) **then**
      CurrentSolution = *TempSolution*;
      π_{ls}^i += ζ_2; π_{sh}^j += ζ_2;
    end
    else if ($\theta > \epsilon$) **then**
      $\rho = e^{(Cost(BestSolution) - Cost(TempSolution))/\theta}$;
      τ = rand (0,1);
      if ($\tau > \rho$) **then**
        TempSolution = *BestSolution*;
      else
        CurrentSolution = *TempSolution*;
        π_{ls}^i += ζ_3; π_{sh}^j += ζ_3;
      end
      $\theta = \theta \times \lambda$;
    end
    else
      TempSolution = *BestSolution*;
    end
  end
  for each *local search procedure* $l \in \{1, 2, \cdots, n_1\}$ update w_{ls}^l using (17);
  for each *shaking procedure* $k \in \{1, 2, \cdots, n_2\}$ update w_{sh}^k using (17);
end

Algorithm 1: ALNS procedure for the RCKRMSF

In what follows, we provide the details of the inner procedures invoked by Algorithm 1.

4.1 Initialization

To construct a feasible initial solution, we adopted an initialization procedure that applies the Prim algorithm over a modified version of the input graph, namely: we set to 0 all costs associated with arcs connecting two supply vertices; we execute the Prim algorithm on the resulting graph, so obtaining an MST; we remove all arcs connecting two supply vertices from the solution, so obtaining an SF. Then, if the SF is infeasible because does not satisfy the minimum reliability constraint, the procedure proceeds to achieve feasibility as follows. Let $\mathcal{L}$ represent the set of all leaf vertices of the infeasible SF. The procedure randomly selects a vertex $l \in \mathcal{L}$ for which the reliability of the path (i.e., the path from the selected supply vertex to l) is not satisfied. Vertex l is removed from its position in the current SF and allocated to a new position that is feasible and has minimum reinsertion cost. Note that this position always exists because we assumed in Section 3 that there is at least a direct arc connecting a source vertex to a demand vertex having reliability greater than or equal to α. The procedure is reiterated until all vertices are assigned to feasible positions, thus producing a feasible solution in output.

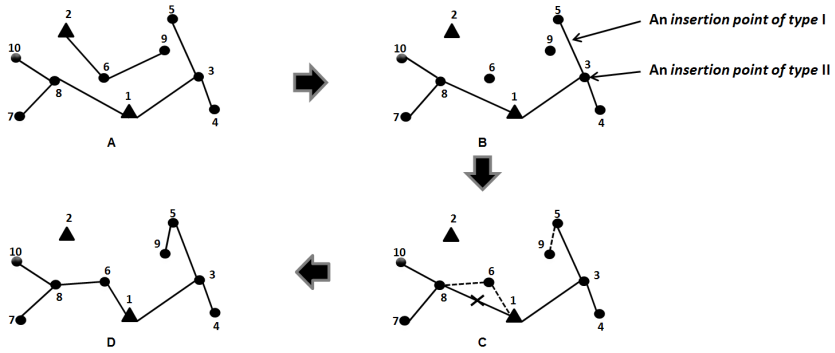
4.2 Local search procedures

Given an initial feasible solution $\mathcal{F}$, five different local search operators have been implemented to try to improve the solution cost. Essentially, in all such procedures modifications of different parts of the initial solution are examined to possibly find a better solution by reconnecting demand and supply vertices.

- *Local search* 1: a restricted solution $\mathcal{RSS}$ is obtained by removing an arc, chosen in random order, from the initial solution $\mathcal{F}$. As a result, a subset $\mathcal{L}^{'}$ of vertices are disconnected from the supply vertices. The procedure examines relocating each vertex in $\mathcal{L}^{'}$, one at a time, to feasible insertion points in $\mathcal{RSS}$. Essentially, insertion points can be an arc of $\mathcal{RSS}$ (*insertion type* I) or a vertex (*insertion type* II). A vertex $l \in \mathcal{L}^{'}$ is selected in random order and allocated to its lowest-cost feasible insertion point in $\mathcal{RSS}$. The procedure iterates until all vertices in $\mathcal{L}^{'}$ are relocated. In case the new solution has a better objective value than $\mathcal{F}$, it is accepted, otherwise the procedure proceeds by selecting a new random arc of $\mathcal{F}$. The process is reiterated until all arcs have been examined and no further improvement is possible. An example is represented in Figure 2: Vertices 6 and 9 are disconnected from the solution by removing arc $(2, 6)$ (Figures 2-A and 2-B) and allocated to new insertion points of types II and I, respectively (Figures 2-C and 2-D).

- *Local search* 2: the procedure chooses a random edge and removes it from $\mathcal{F}$. As a result, a sub-tree

Figure 2: An illustrative example of *local search* 1.



is disconnected in the restricted solution $\mathcal{RSS}$ that has been obtained. All insertion points of type II in $\mathcal{RSS}$ are examined for the re-insertion of the entire sub-tree. If a feasible and lower cost insertion point is found, then the sub-tree is moved there, otherwise it is put back in its original position. The procedure iterates until all arcs in $\mathcal{F}$ have been tested for removal.

- *Local search* 3: let $\mathcal{L}$ represent the set of all leaves of $\mathcal{F}$. The procedure selects a random leaf from $\mathcal{L}$ and removes it from its current position. It then attempts to re-insert it by considering all feasible insertion points of type I or II, and selecting the one having minimum cost and being feasible, if any. The move is accepted if it leads to an improvement in the objective function of the problem, and the procedure is re-iterated until all leaves have been examined and no further improvement is possible.

- *Local search* 4: the procedure randomly selects two vertices of $\mathcal{F}$ and swaps them. The move is accepted if the resulting solution is feasible and has a lower objective value. The procedure terminates when all possible swaps are examined and no further improvement is possible.

- *Local search* 5: This rule is a math-based procedure that aims at improving the objective function by finding a new rearrangement of the leaves' positions in the initial solution $\mathcal{F}$. The procedure starts by selecting all leaves of $\mathcal{F}$ and removing them from the solution. A restricted solution $\mathcal{RSS}$ is consequently obtained. Assuming $\mathcal{L}$ to represent the set of all extracted vertices, the procedure tries to find an improved cost solution by reallocating the vertices of $\mathcal{L}$ to $\mathcal{RSS}$ by means of an integer linear programming model.

  To this aim, all sequences consisting of one or two customers belonging to $\mathcal{L}$ are generated. Let SS' and SS'' denote the sequences of size one and two of the extracted vertices, respectively. We represent a sequence $s \in SS = SS' \cup SS''$ as $s = (b_s, t_s)$, in which b_s and t_s are the vertices belonging to s, and $b_s = t_s$ in case $s \in S'$. The set of all feasible insertion points corresponding to

the sequence s is represented by $\mathcal{J}_s$. Essentially, an insertion point is a demand or a supply vertex in $\mathcal{RSS}$ to which sequence s could be allocated through an *insertion type* II move. We denote re_j the reliability of the path ending at vertex j and starting from one of the available supply vertices. In addition, we denote re_s the reliability of the sequence $s \in S$. If $s \in SS'$ the corresponding reliability is set to one, otherwise it is set to the reliability of arc (b_s, t_s). Finally, we represent by SS_l the subset of sequences in SS that contain customer vertex $l \in \mathcal{L}$. We make use of the following decision variable:

$$y_{sj} = \begin{cases} 1 & \text{if sequence } s \in SS \text{ is connected to the insertion point } j \in \mathcal{J}_s, \\ 0 & \text{otherwise,} \end{cases}$$

for all $s \in SS, j \in \mathcal{J}_s$. The model for the optimal reallocation of the leaves is then:

$$\min \sum_{s \in S} \sum_{j \in \mathcal{J}_s} c_{sj} y_{sj} \tag{18}$$

$$\sum_{s \in SS_l} \sum_{j \in \mathcal{J}_s} y_{sj} = 1 \qquad \forall l \in \mathcal{L}, \tag{19}$$

$$re_j(1 - y_{sj}) + re_j \times re_s \times r_{jb_s} \times y_{sj} \geq \alpha \qquad \forall j \in \mathcal{RSS}, s \in SS, \tag{20}$$

$$y_{sj} \in \{0, 1\} \qquad \forall s \in SS, j \in \mathcal{J}_s. \tag{21}$$

The objective function (18) is to minimize the total reallocation cost. For each $l \in \mathcal{L}$, constraints (19) impose that each extracted vertex is allocated to exactly one position of the restricted solution. Constraints (20) impose the minimum required reliability. Essentially, the reliability of any path obtained by adding a sequence $s \in SS$ to a vertex $j \in \mathcal{RSS}$ cannot be less than the pre-specified value α. Constraints (21) define the variables' domain.

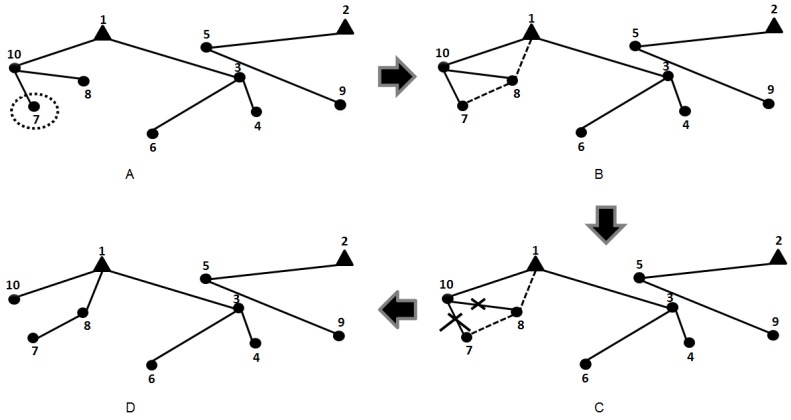
4.3 Shaking procedures

In this section, we describe six shaking procedures that are used within the developed ALNS. In all such procedures, regardless of the change in the cost of the solution, new ways of connecting the vertices are considered, so that a larger portion of the feasible space can be searched and new solutions can be given in input to the local search operators above. All procedures start from an initial feasible solution $\mathcal{F}$.

- *Shaking procedure* 1: The purpose of this procedure is to change a portion of $\mathcal{F}$ by substituting an available path with a new one. To this aim, a demand vertex is randomly selected and the *max-reliability* path is obtained. Essentially, the max-reliability path is the path that originates from one of the supply vertices, reaches the selected demand vertex and has maximum reliability among

all paths of this type. This path can be easily obtained by applying the Dijkstra algorithm. In case of having several paths with the same maximum reliability, one of them is randomly selected. The selected path is included in the solution, and the cycles that have been possibly generated by its addition are erased by removing arcs from $\mathcal{F}$. The new solution obtained by this shaking procedure is always feasible from the reliability point of view. Figure 3 gives an example in which vertex 7 is selected (Figure 3.A) and its corresponding max-reliability path is calculated (dotted path in Figure 3.B). As the newly generated solution has cycles, the procedure removes a subset of arcs belonging to the arc set of the solution before adding the max-reliability path and connected to the vertices visited by the new path (Figure 3.C), consequently restoring feasibility (Figure 3.D).

Figure 3: An illustrative example of *shaking procedure* 1.

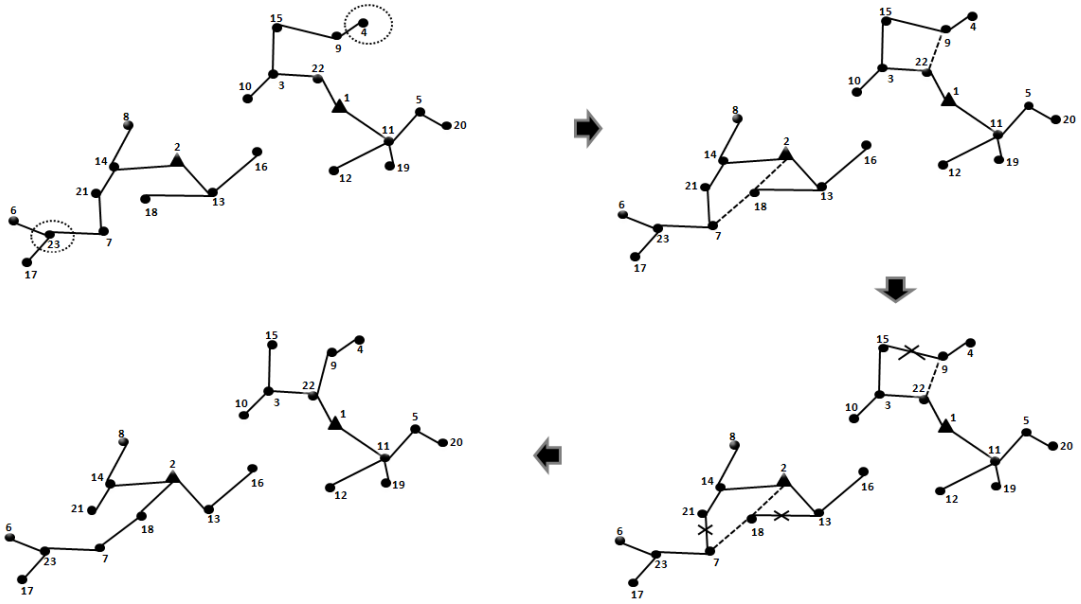


- *Shaking procedure* 2: This procedure is a generalization of *shaking procedure* 1, in which three demand vertices, instead of one, are randomly selected from $\mathcal{F}$. Taking into account all the available supply vertices, the max-reliability paths for the three vertices are obtained and implemented in the new solution, and possibly generated cycles are removed.

- *Shaking procedure* 3: We select two demand vertices, say i and j, having distance (c_{ij}) greater than a pre-specified input value $\mathcal{C}$. The maximum-reliability paths for the two vertices are obtained and implemented, and possible cycles are removed from the new solution that has been obtained. An illustrative example is shown in Figure 4, in which demand vertices 4 and 23 are selected to perform the shaking.

- *Shaking procedure* 4: In this procedure, two vertices of $\mathcal{F}$ are selected randomly, and their locations are swapped with each other. If the move is feasible from the reliability point of view, it is accepted, otherwise the procedure examines swapping other vertices to find the first feasible swap, if any.

Figure 4: An illustrative example of *shaking procedure* 3.



- *Shaking procedure* 5: A restricted solution $\mathcal{RSS}$ is obtained by randomly removing an arc from $\mathcal{F}$. As a result, a sub-tree $\mathcal{F}^{'}$ of $\mathcal{F}$ is disconnected from the forest. Then, a random vertex of $\mathcal{RSS}$ is selected to reallocate the extracted sub-tree $\mathcal{F}^{'}$. If the solution obtained in this way is not feasible because does not satisfy the minimum reliability constraint, another random vertex of $\mathcal{RSS}$ is selected for the reallocation of $\mathcal{F}^{'}$. In case no feasible position exists to reinsert $\mathcal{F}^{'}$, $\mathcal{F}^{'}$ is reallocated back to its original position, and the procedure iterates by selecting and removing a new random arc from $\mathcal{F}$.

- *Shaking procedure* 6: Similarly to what proposed for shaking procedure 5, in this procedure too an arc is randomly chosen from $\mathcal{F}$ and removed. The disconnected component $\mathcal{F}^{'}$ obtained by this operation is then reattached to $\mathcal{RSS}$. The only difference with respect to the previous procedure is that now the position in which $\mathcal{F}^{'}$ has to be connected to $\mathcal{RSS}$ is randomly selected. In case the resulting solution is infeasible, the procedure continues by selecting a new random vertex of $\mathcal{RSS}$ as insertion point.

5 Computational results

In this section, we provide extensive computational tests to evaluate the effectiveness of the proposed ALNS. All algorithms have been implemented in C++ and tested on a PC with a 2.8 GHz Intel Dual Core and 4 GB of RAM. We used ILOG Cplex 12.3 to run the RCKRMSF model of Section 3. We present

results on new RCKRMSF instances that we randomly generated and on DCMST benchmark instances from the literature. The ALNS has been executed five times on each attempted instance. Due to the large number of tests, in this section we only report aggregated results, so as to facilitate comprehension.

5.1 Instance generation

We generated two sets of instances, the first having Euclidean distances and the second non-Euclidean. The first set contains 180 instances having number of vertices $n \in \{20, 30, 40, 50\}$ and number of sources $m \in \{2, 3, 4\}$. Three values were selected for the minimum path reliability, namely $\alpha \in \{0.95, 0.90, 0.80\}$, and the reliabilities of the arcs were created as random numbers in the interval $[\alpha, 1]$. The coordinates of the vertices were randomly selected in the square $[0, 100] \times [0, 100]$, and, for each $i, j \in V$, c_{ij} was set equal to the Euclidean distance between the coordinates of i and j. To take randomness into account, for each combination of n, m and α, five random instances have been generated, differing one another in the distances and reliabilities on the arcs. The second set was created mainly to compare Euclidean with non-Euclidean distances. This set was indeed generated as the first one for what concerns n, m, α and arc reliabilities, but it contains distances that have been randomly generated for each arc in the interval $[1, 100]$. For each combination of n, m and α, two random instances have been generated.

All instances that we generated are provided as on-line supplemental material to this paper.

5.2 Parameter tuning

The ALNS of Section 4 contains a number of parameters that have to be finely tuned. To find their best configuration, we opted to use a simple *genetic algorithm* (GA). GAs (see, e.g., Sastry *et al.* 2014) are well suited for the optimization of complex problems, and parameter tuning is one of the fields in which they have been successfully applied, as recently shown in, e.g., Liu *et al.* (2014); Yu and Xu (2014). The sketch of the GA that we adopted for selecting the best ALNS parameters is described in Algotithm 2.

1. Generate an initial random population of individuals (each being a combination of parameters)
2. Calculate the fitness of each individual
3. Produce the next population by using the following operators
   - Selection: Individuals with high fitness values are selected to breed a new generation
   - Crossover: Groups of two parents are selected and combined to create offspring solutions
   - Mutation: Each solution is locally randomly modified to increase diversity
4. Repeat steps 2 and 3 until the stopping criterion is met
5. Return the last population

Algorithm 2: The general framework of the GA adopted for the ALNS parameter tuning

For this preliminary tuning, a test set of 20 instances having 20 or 40 vertices have been randomly

Table 1: ALNS parameters

Parameter	Description	Different tested values	Adopted value
θ	initial temperature	{500, 1000, 2000, 5000, 10000, 15000}	1000
ϵ	final temperature	{0.0001}	0.0001
λ	cooling rate	{0.1, 0.01, 0.001, 0.0001, 0.00001}	0.1
η	reaction factor	{0.1, 0.2, 0.3, 0.4, 0.5}	0.2
$\mathcal{C}$	distance between two vertices in shaking 3	{20, 30, 40, 50, 60}	50
ϕ_1	number of segments	{3, 5, 10, 20, 30, 40}	5
ϕ_2	number of iterations in each segment	{3, 5, 10, 20, 30, 40}	10
ζ_1	π increment value 1	{30, 50, 70, 90, 110, 130}	50
ζ_2	π increment value 2	{10, 20, 30, 40, 50, 60}	20
ζ_3	π increment value 3	{3, 5, 10, 15, 20, 25}	5

generated as in the first set of Section 5.1. The initial GA population was set to 20 random individuals. The chromosome of each individual consists of a list of the ALNS parameters to be tuned, and its fitness is the average value of the 20 solutions found by executing the ALNS with this list of parameters on the 20 instances in the test set. We adopted a simple truncation selection, in which the top $(1/s)^{th}$ of the individuals get s copies each in the mating pool, with $s = 2$ in our tests. One-point crossover was chosen as the operator for recombining the two parent chromosomes (a point is randomly selected over the chromosome length, and all data beyond this point is swapped between the two parents). As mutation procedure, we opted to change an arbitrary bit in a chromosome from its original state. We decided to stop the GA after the generation of 30 populations.

Table 1 shows the list of ALNS parameters, their brief descriptions, the sets of parameter values that have been tested with the GA, and, in the last column, the final values that have been adopted on the basis of the results. It is worth mentioning that the performance of the ALNS was not very sensitive to the set of parameters. We applied indeed the Wilcoxon signed rank test, a well-known nonparametric statistical test from Wilcoxon (1945), to perform a pairwise comparison of parameters. To do so, we selected the two parameter combinations having, respectively, the best and the worst average performance (by considering the average objective function over the 20 test instances). The Wilcoxon test showed that these combinations do not dominate each other at a significance level of $\alpha < 0.05$. This confirms the robustness of the developed ALNS.

5.3 Computational results on DCMST benchmarks

With the aim of assessing the effectiveness of the ALNS, we considered the DCMST discussed in Section 2. The DCMST is a special case of the RCKRMSF in which a single source is used to serve demand vertices and a maximum delay is allowed on each path. Our ALNS can be easily adapted to solve this problem by considering delays instead of reliabilities on the arcs.

A relevant work on the DCMST is that of Gouveia *et al.* (2008), who proposed a number of instances and solved most of them to proven optimality in one day of CPU time by means of algorithms based on

Lagrangian relaxation and column generation. The authors considered instances with 20 and 40 demand vertices, divided into three main groups, namely TR, TC and TE. While the TR group contains instances for which the edge costs are random values, the TC and TE groups both include instances with Euclidean costs, with the only difference that the source vertex is near the center of the grid of vertices for the TC group and near the border for the TE. Instances are denoted as X,Y [Z,W], where X is the group, Y is the number of demand vertices and [Z,W] gives the range in which the arc delays have been created, with Z=1 and W $\in \{2, 5, 10, 100, 1000\}$. For each such configuration, four instances have been generated by using different values on the maximum delay bound, resulting in a total of 120 instances. The ALNS was executed five times on each instance.

Similarly to Gouveia *et al.* (2008), we present separate results for the classes involving small and large delays. The former results are in Table 2 and the latter ones in Table 3. For each group of four instances per line, each table provides the average (over the four instances per line) of the best (over the five attempts) gap between the ALNS solution value and the best known solution value reported by Gouveia *et al.* (2008), and the average ALNS execution time (over the four instances and five attempts). The ALNS was executed with the default parameters proposed for the RCKRMSF. It is clear from the tables that the ALNS is very effective in tackling the DCMST, being able to achieve very low gaps in a matter of seconds. The average best gap is usually below 0.5% and just in two cases (both for TR,40) reaches 0.71%. Essentially, the average gap over all the available DCMST instances, with respect to the best know solutions, is around 0.21%, and the average CPU time is just 2.38 seconds.

Table 2: ALNS comparison with best solutions on DCMST benchmarks – small delays (4 instances per line)

Class 1	Avg. best gap	Avg. time	Class 2	Avg. best gap	Avg. time	Class 3	Avg. best gap	Avg. time
TR,20 [1,2]	0.00	1.61	TR,20 [1,5]	0.00	1.56	TR,20 [1,10]	0.00	1.64
TC,20 [1,2]	0.00	1.59	TC,20 [1,5]	0.00	1.68	TC,20 [1,10]	0.08	1.63
TE,20 [1,2]	0.00	1.27	TE,20 [1,5]	0.00	1.30	TE,20 [1,10]	0.03	1.31
TR,40 [1,2]	0.40	2.82	TR,40 [1,5]	0.52	3.16	TR,40 [1,10]	0.47	3.06
TC,40 [1,2]	0.11	2.82	TC,40 [1,5]	0.22	2.98	TC,40 [1,10]	0.22	3.08
TE,40 [1,2]	0.25	3.25	TE,40 [1,5]	0.64	3.24	TE,40 [1,10]	0.35	3.32

Table 3: ALNS comparison with best solutions on DCMST benchmarks – large delays (4 instances per line)

Class 4	Avg. best gap	Avg. time	Class 5	Avg. best gap	Avg. time
TR,20 [1,100]	0.35	1.69	TR,40 [1,1000]	0.20	1.76
TC,20 [1,100]	0.06	1.81	TC,40 [1,1000]	0.00	1.86
TE,20 [1,100]	0.00	1.29	TE,40 [1,1000]	0.00	1.35
TR,40 [1,100]	0.71	3.44	TR,40 [1,1000]	0.71	2.57
TC,40 [1,100]	0.17	3.05	TC,40 [1,1000]	0.25	2.19
TE,40 [1,100]	0.74	3.62	TE,40 [1,1000]	0.52	3.72

We use this set of instances also to provide a sensitivity analysis for ϕ_1 (number of global ALNS iterations) and ϕ_2 (number of inner iterations). Figure 5 shows the average best gap for all DCMST

instances of the three groups with respect to different values of ϕ_2, by keeping $\phi_1 = 5$. It can be noticed that the gap tends to zero by increasing the value of ϕ_2, showing that the ALNS tends not to be stuck in local optima. Figure 6 shows the same analysis for the case in which $\phi_2 = 10$ and ϕ_1 takes different values ranging from 5 to 40. Also in this case, by increasing the number of iterations the optimality gap for all TR, TC and TE groups decreases and gets below 0.1%.

Figure 5: The effect of parameter ϕ_2 on the average best gap for the three DCMST groups

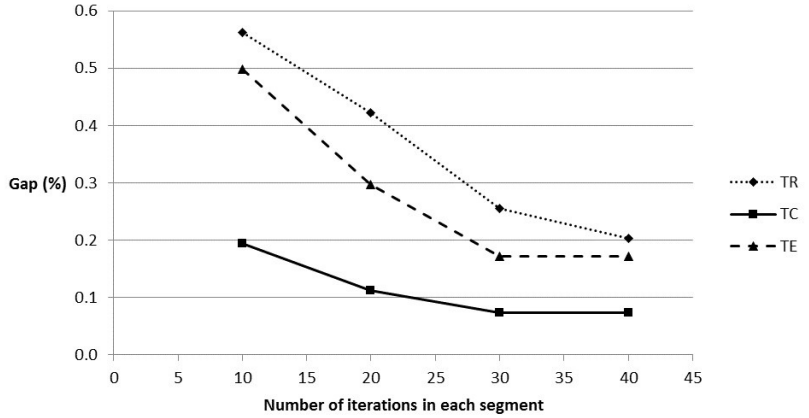
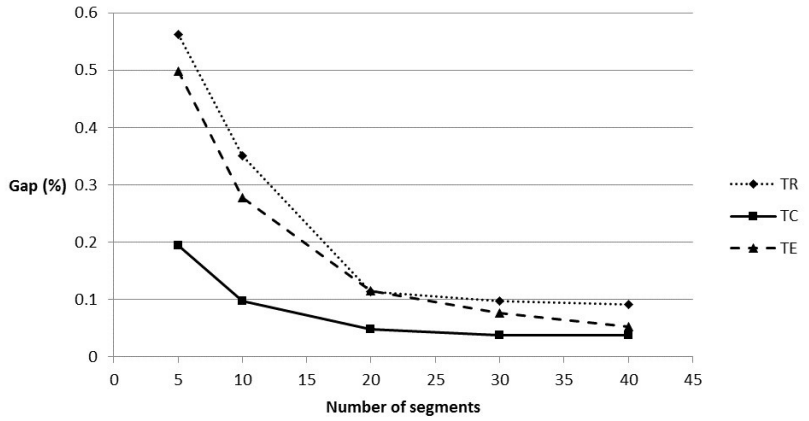


Figure 6: The effect of parameter ϕ_1 on the average best gap or the three DCMST groups



5.4 Computational results on Euclidean RCKRMSF instances

In this section, we focus on the solution of the new RCKRMSF instances that we generated. Tables 4, 5 and 6 give the results that we obtained on Euclidean instances with α=0.95, 0.90 and 0.80, respectively, by running the MILP model composed of (1)–(11) and (13)–(16) with CPLEX and the ALNS of Section 4. CPLEX was allowed a maximum run time of 18000 CPU seconds. Each line in the tables report average results on the selected group of five instances having the same values of n, m and α. For the

MILP model, we provide the lower bound value reported at the end of the run in column "LB", the solution value in column "UB", and the elapsed CPU seconds in column "time". For the ALNS, which was run five times on each instance, we provide, in order, the best, average and worst solution values, the average and best percentage gaps from the UB of the MILP model, and the average execution time. The gaps have been computed as $100\times(\text{UB}_{ALNS}-\text{UB}_{MILP})/\text{UB}_{MILP}$.

Table 4: Computational results on Euclidean instances with $\alpha = 0.95$ (5 instances per line)

n	m	MILP model			ALNS					
		LB	UB	Time	Best UB	Worst UB	Avg. UB	Avg. gap	Best gap	Avg. time
20	2	325.82	325.82	531.83	325.82	329.83	326.62	0.23	0.00	1.40
20	3	281.06	281.06	201.60	281.06	281.06	281.06	0.00	0.00	1.72
20	4	255.60	255.60	106.52	255.60	256.05	255.96	0.13	0.00	1.36
30	2	418.77	445.85	18000.00	446.04	447.46	446.58	0.16	0.04	1.76
30	3	394.98	420.63	18000.00	421.42	423.89	422.71	0.46	0.17	1.44
30	4	366.06	383.04	15701.08	383.04	387.12	384.71	0.45	0.00	1.69
40	2	483.48	556.20	18000.00	527.58	539.12	533.44	−3.86	−4.93	2.57
40	3	447.38	490.20	18000.00	488.15	494.41	490.49	0.08	−0.40	2.24
40	4	425.12	460.64	18000.00	459.61	463.28	460.64	0.01	−0.20	2.45
50	2	524.96	677.13	18000.00	605.31	617.26	610.20	−9.52	−10.23	5.88
50	3	495.23	622.93	18000.00	567.68	576.16	570.42	−8.24	−8.67	4.56
50	4	466.33	567.37	18000.00	530.30	538.87	533.77	−5.77	−6.37	4.68
Average		407.07	457.20	13378.42	440.97	446.21	443.05	−2.16	−2.55	2.65

Table 5: Computational results on Euclidean instances with $\alpha = 0.90$ (5 instances per line)

n	m	MILP model			ALNS					
		LB	UB	Time	Best UB	Worst UB	Avg. UB	Avg. gap	Best gap	Avg. time
20	2	332.77	332.77	431.80	332.77	334.93	333.21	0.12	0.00	1.14
20	3	293.89	293.89	166.57	293.89	294.65	294.19	0.10	0.00	1.30
20	4	266.55	266.55	44.40	266.55	267.64	266.77	0.08	0.00	1.44
30	2	402.61	444.93	18000.00	443.75	453.15	446.56	0.34	−0.27	1.62
30	3	382.78	406.83	16023.94	406.83	408.20	407.51	0.17	0.00	1.65
30	4	358.15	376.43	14631.73	376.43	377.71	376.85	0.11	0.00	1.67
40	2	483.34	551.44	18000.00	537.47	543.89	540.42	−1.97	−2.52	2.61
40	3	451.25	497.56	18000.00	496.57	501.25	498.68	0.22	−0.20	2.62
40	4	429.33	474.45	18000.00	472.62	477.42	474.29	−0.03	−0.37	2.59
50	2	524.00	702.91	18000.00	585.99	602.61	593.08	−14.70	−15.65	5.38
50	3	495.10	593.27	18000.00	550.57	560.99	554.76	−6.14	−6.82	4.92
50	4	466.72	560.15	18000.00	517.15	523.04	519.52	−6.90	−7.31	4.55
Average		407.21	458.43	13108.20	440.05	445.46	442.15	−2.38	−2.76	2.62

Out of the 180 instances addressed in the three tables, 49 instances were solved to proven optimality by the MILP model in the given time limit. These include all 45 instances with 20 demand vertices, and 4 with 30 vertices. For the unsolved instances, the difference between the UB and LB values produced by the model can be quite high. On average, the overall gap between these two values is slightly above 10%, for all values of α.

The ALNS finds all 49 proven optimal UBs in at least one of the five runs performed on those instances. On the other instances, it usually produces very good gaps. The best gap varies from 0.17% (for $n = 30$, $m = 3$ and $\alpha = 0.95$) to −15.65% (for $n = 50$, $m = 2$ and $\alpha = 0.90$). On average, the best

Table 6: Computational results on Euclidean instances with $\alpha = 0.80$ (5 instances per line)

n	m	MILP model			ALNS					
		LB	UB	Time	Best UB	Worst UB	Avg. UB	Avg. gap	Best gap	Avg. time
20	2	323.55	323.55	521.29	323.55	324.82	324.06	0.15	0.00	1.38
20	3	289.07	289.07	599.96	289.07	290.41	290.02	0.27	0.00	1.42
20	4	264.06	264.06	409.47	264.06	264.79	264.55	0.16	0.00	1.51
30	2	412.56	455.50	18000.00	455.33	461.63	457.85	0.51	−0.02	1.89
30	3	390.60	418.67	18000.00	419.20	419.98	419.51	0.20	0.13	1.84
30	4	363.04	379.28	16189.15	379.28	379.96	379.68	0.11	0.00	1.73
40	2	489.87	559.68	18000.00	546.71	559.95	552.10	−3.37	−4.30	2.60
40	3	453.61	503.92	18000.00	499.72	507.78	502.80	−0.15	−0.74	2.92
40	4	429.22	479.47	18000.00	474.93	479.34	476.88	−0.47	−0.90	2.61
50	2	514.04	647.95	18000.00	584.71	596.33	590.69	−8.56	−9.47	5.20
50	3	484.83	593.58	18000.00	548.41	562.12	552.54	−6.82	−7.49	4.70
50	4	455.51	563.24	18000.00	516.11	517.95	516.97	−7.78	−7.93	4.81
Average		405.83	456.50	13476.66	441.76	447.09	443.97	−2.14	−2.56	2.72

gap shows improvement of about 2.5% with respect to the UBs produced by the MILP model, but such improvements can be very relevant on instances with 50 vertices, with values ranging between −6% and −15%. The good performance of the ALNS can also be noticed by looking at the worst and average UBs, which are not far away from the best ones. The overall average gap is between −2.1% and −2.4%, and is quite consistent for the different values taken by α. These good-quality solutions are obtained in very short times, involving less than 3 seconds on average and nearly 6 seconds in the worst case (for $n = 50$, $m = 2$ and $\alpha = 0.95$). By looking at these tables, we can conclude that the MILP model is effective up to 20 demand vertices, whereas the ALNS is effective and fast on all instances. It can also be noted that the algorithms are not particularly sensitive to the value of α.

Tables 7, 8 and 9 provide average results on the non-Euclidean set of instances, focusing again on α=0.95, 0.90 and 0.80, respectively. The values reported have the same meanings as those in Tables 4–6, but now refer to just two instances per line instead of five. Out of 72 instances, the ALNS outperforms the UB value obtained by CPLEX in 36 cases at least once in the five runs for instance. For the remaining instances, it always finds the same objective value of the MILP model, still at least once in the five runs. Overall, all best gaps are always null or negative, whereas the average gaps might rarely take positive values, although these are very small. More variability with respect to the results on the Euclidean instances can be noticed on the overall average gaps, which vary from a minimum of −5.27% for $\alpha = 0.90$ to a maximum of −2.22% for $\alpha = 0.80$. This might be induced by the reduced number of instances in the second test bed. The MILP model is now a bit more effective on the instances having 30 demand vertices, but remains very slow for the larger instances. Also for this test bed, we can conclude that the ALNS is fast and effective.

Table 7: Computational results on non-Euclidean instances with $\alpha = 0.95$ (2 instances per line)

n	m	Mathematical model			ALNS					
		LB	UB	Time	Best UB	Worst UB	Avg. UB	Avg. gap	Best gap	Avg. time
20	2	162.50	162.50	93.31	162.50	164.00	163.20	0.35	0.00	0.82
20	3	144.00	144.00	134.52	144.00	144.00	144.00	0.00	0.00	0.89
20	4	132.50	132.50	64.05	132.50	133.00	132.60	0.06	0.00	0.86
30	2	199.91	208.00	14578.21	207.50	208.00	207.60	−0.16	−0.20	2.05
30	3	176.50	176.50	8939.29	176.50	176.50	176.50	0.00	0.00	2.10
30	4	163.00	163.00	4301.67	163.00	163.00	163.00	0.00	0.00	1.73
40	2	177.14	219.00	18000.00	212.00	213.50	212.30	−3.06	−3.20	2.36
40	3	167.55	207.00	18000.00	199.50	202.50	200.30	−3.24	−3.63	2.32
40	4	157.36	195.00	18000.00	186.50	190.50	187.90	−3.34	−4.10	2.06
50	2	160.72	239.00	18000.00	204.00	210.50	207.10	−13.05	−14.29	4.42
50	3	151.06	205.50	18000.00	189.50	196.00	192.30	−5.80	−7.22	3.76
50	4	144.37	204.50	18000.00	180.50	190.50	183.70	−8.88	−10.62	3.64
Average		161.38	188.04	11342.59	179.83	182.67	180.88	−3.09	−3.60	2.25

Table 8: Computational results on non-Euclidean instances with $\alpha = 0.90$ (2 instances per line)

n	m	Mathematical model			ALNS					
		LB	UB	Time	Best UB	Worst UB	Avg. UB	Avg. gap	Best gap	Avg. time
20	2	148.50	148.50	17.96	148.50	148.50	148.50	0.00	0.00	1.02
20	3	128.00	128.00	13.90	128.00	128.00	128.00	0.00	0.00	0.95
20	4	118.00	118.00	13.38	118.00	118.00	118.00	0.00	0.00	1.39
30	2	203.31	212.50	17624.33	211.50	217.50	213.90	0.79	−0.44	1.61
30	3	185.27	196.50	18000.00	196.50	198.00	196.80	0.14	0.00	2.61
30	4	170.89	183.50	14578.38	183.50	183.50	183.50	0.00	0.00	2.19
40	2	204.46	254.00	18000.00	245.50	249.00	248.30	−2.24	−3.31	2.68
40	3	188.65	252.00	18000.00	231.00	232.50	231.30	−7.95	−8.06	2.92
40	4	175.26	207.50	18000.00	205.00	205.00	205.00	−1.11	−1.11	2.17
50	2	157.98	299.50	18000.00	193.00	198.00	195.30	−34.82	−35.59	5.38
50	3	151.16	218.00	18000.00	184.50	190.50	187.30	−14.11	−15.38	3.94
50	4	142.21	184.00	18000.00	175.00	179.50	176.80	−3.93	−4.90	3.97
Average		164.47	200.17	13187.33	185.00	187.33	186.06	−5.27	−5.73	2.57

Table 9: Computational results on non-Euclidean instances with $\alpha = 0.80$ (2 instances per line)

n	m	Mathematical model			ALNS					
		LB	UB	Time	Best UB	Worst UB	Avg. UB	Avg. gap	Best gap	Avg. time
20	2	186.00	186.00	73.26	186.00	186.00	186.00	0.00	0.00	0.84
20	3	145.00	145.00	103.29	145.00	145.00	145.00	0.00	0.00	1.02
20	4	117.00	117.00	6.54	117.00	117.00	117.00	0.00	0.00	1.33
30	2	189.89	204.00	10162.24	204.00	209.50	205.80	0.98	0.00	2.07
30	3	169.50	169.50	6125.59	169.50	170.00	169.90	0.31	0.00	1.72
30	4	158.00	158.00	6984.14	158.00	158.50	158.30	0.24	0.00	1.88
40	2	195.31	238.50	18000.00	231.50	234.50	232.50	−2.66	−3.07	2.47
40	3	181.05	244.00	18000.00	224.00	225.50	224.60	−7.86	−8.11	2.20
40	4	170.96	211.50	18000.00	203.00	204.00	203.30	−3.89	−4.02	2.38
50	2	167.49	230.00	18000.00	212.50	217.50	213.90	−7.01	−7.61	4.97
50	3	158.77	206.50	18000.00	197.00	207.00	199.10	−3.32	−4.28	3.87
50	4	148.10	191.50	18000.00	183.50	189.50	184.80	−3.45	−4.14	3.96
Average		165.59	191.79	10954.59	185.92	188.67	186.68	−2.22	−2.60	2.39

6 Conclusions

We proposed the reliability constraint k-rooted minimum spanning forest. The goal of the problem is to find a k-rooted minimum cost forest that satisfies a minimum required reliability on each path from a source to a demand vertex. The problem has relevant applications in the design of several types of

networks. We solved it by means of a mathematical model and an adaptive large neighborhood search metaheuristic. The model is effective in solving instances with up to 20 demand vertices, whereas the metaheuristic is both fast and effective not only on all instances of the proposed problem, but also on the special case known in the literature as the delay constrained minimum spanning tree. Such conclusions are motivated by extensive computational tests on randomly created instances that have been now made publicly available to stimulate further research.

Instances with just 30 demand vertices remain unsolved to proven optimality despite relevant computing efforts, so, as future research, much could be done in terms of exact algorithms to attempt solving larger instances. Another interesting research direction is that of studying the budget-constrained version of the problem, that is, the problem variant in which the goal is to construct a maximum reliability spanning forest with the presence of a limitation on the available network construction cost.

References

Achuthan, N., Caccetta, L., Caccetta, P. and Geelen, J. (1992), Algorithms for the minimum weight spanning tree with bounded diameter problem. *Optimization: techniques and applications*, v. 1, n. 2, p. 297–304.

Alinaghian, M. and Shokouhi, N. (2018), Multi-depot multi-compartment vehicle routing problem, solved by a hybrid adaptive large neighborhood search. *Omega*, v. 76, p. 85–99.

Altiparmak, F. and Dengiz, B. (2009), A cross entropy approach to design of reliable networks. *European Journal of Operational Research*, v. 199, n. 2, p. 542–552.

Altiparmak, F., Dengiz, B. and Smith, A. E. (2003), Optimal design of reliable computer networks: A comparison of metaheuristics. *Journal of Heuristics*, v. 9, n. 6, p. 471–487.

Atiqullah, M. M. and Rao, S. S. (1993), Reliability optimization of communication networks using simulated annealing. *Microelectronics reliability*, v. 33, n. 9, p. 1303–1319.

Azaron, A., Perkgoz, C., Katagiri, H., Kato, K. and Sakawa, M. (2009), Multi-objective reliability optimization for dissimilar-unit cold-standby systems using a genetic algorithm. *Computers & Operations Research*, v. 36, n. 5, p. 1562–1571.

Ball, M. O., Colbourn, C. J. and Provan, J. S. Chapter 11 network reliability. *Network Models*, volume 7 of *Handbooks in Operations Research and Management Science*, p. 673–762. Elsevier, 1995.

Bazlamaçci, C. and Hindi, K. (2001), Minimum-weight spanning tree algorithms a survey and empirical study. *Computers & Operations Research*, v. 28, n. 8, p. 767–785.

Bicalho, L., da Cunha, A. and Lucena, A. (2016), Branch-and-cut-and-price algorithms for the degree constrained minimum spanning tree problem. *Computational Optimization and Applications*, v. 63, n. 3, p. 755–792.

Bruck, B. P. and Iori, M. (2017), Non-elementary formulations for single vehicle routing problems with pickups and deliveries. *Operations Research*, v. 65, n. 6, p. 1597–1614.

Canale, E., Cancela, H., Robledo, F., Rubino, G. and Sartor, P. (2013), On computing the 2-diameter-constrained k-reliability of networks. *International Transactions in Operational Research*, v. 20, n. 1, p. 49–58.

Da Cunha, A. S., Simonetti, L. and Lucena, A. Formulations and branch-and-cut algorithm for the k-rooted mini-max spanning forest problem. *Network Optimization*, p. 43–50. Springer, 2011.

Dash, R., Barpanda, N., Tripathy, P. and Tripathy, C. R. (2012), Network reliability optimization problem of interconnection network under node-edge failure model. *Applied Soft Computing*, v. 12, n. 8, p. 2322–2328.

de Almeida, A., Ferreira, R. and Cavalcante, C. (2015), A review of the use of multicriteria and multi-objective models in maintenance and reliability. *IMA Journal of Management Mathematics*, v. 26, n. 3, p. 249–271.

de Souza, M. C. and Martins, P. (2008), Skewed vns enclosing second order algorithm for the degree constrained minimum spanning tree problem. *European Journal of Operational Research*, v. 191, n. 3, p. 677–690.

Dengiz, B., Altiparmak, F. and Belgin, O. (2010), Design of reliable communication networks: A hybrid ant colony optimization algorithm. *IIE Transactions*, v. 42, n. 4, p. 273–287.

Dias, F. C., Campêlo, M., Souza, C. and Andrade, R. (2017), Min-degree constrained minimum spanning tree problem with fixed centrals and terminals: Complexity, properties and formulations. *Computers & Operations Research*, v. 84, p. 46–61.

Dror, M., Haouari, M. and Chaouachi, J. (2000), Generalized spanning trees. *European Journal of Operational Research*, v. 120, n. 3, p. 583–592.

Edrissi, A., Nourinejad, M. and Roorda, M. J. (2015), Transportation network reliability in emergency response. *Transportation Research Part E: Logistics and Transportation Review*, v. 80, p. 56–73.

Gouveia, L. (1995), Using the Miller-Tucker-Zemlin constraints to formulate a minimal spanning tree problem with hop constraints. *Computers & Operations Research*, v. 22, n. 9, p. 959–970.

Gouveia, L. and Lopes, M. J. (2005), The capacitated minimum spanning tree problem: On improved multistar constraints. *European Journal of Operational Research*, v. 160, n. 1, p. 47–62.

Gouveia, L., Paias, A. and Sharma, D. (2008), Modeling and solving the rooted distance-constrained minimum spanning tree problem. *Computers & Operations Research*, v. 35, n. 2, p. 600–613.

Heidarzadeh, A., Sprintson, A. and Singh, C. (2018), A fast and accurate failure frequency approximation for$k$-terminal reliability systems. *IEEE Transactions on Reliability*, v. 67, n. 3, p. 933–950.

Jan, R.-H., Hwang, F.-J. and Chen, S.-T. (1993), Topological optimization of a communication network subject to a reliability constraint. *IEEE Transactions on Reliability*, v. 42, n. 1, p. 63–70.

K. Bernard, K., Tarabalka, Y., Angulo, J. and Chanussot, J.and Benediktsson, J. (2012), Spectral-spatial classification of hyperspectral data based on a stochastic minimum spanning forest approach. *IEEE Transactions on Image Processing*, v. 21, n. 4, p. 2008–2021.

Kang, S. and Bader, D. A. An efficient transactional memory algorithm for computing minimum spanning forest of sparse graphs. *Proceedings of the 14th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '09, p. 15–24, New York, NY, USA. ACM, 2009.

Klein, A., Haugland, D., Bauer, J. and Mommer, M. An integer programming model for branching cable layouts in offshore wind farms. Le Thi, H., Pham Dinh, T. and Nguyen, N. (Eds.), *Modelling, Computation and Optimization in Information Systems and Management Sciences*, volume 359 of *Advances in Intelligent Systems and Computing book series*, p. 27–36. Springer, 2015.

Koide, T., Shinmori, S. and Ishii, H. (2001), Topological optimization with a network reliability constraint. *Discrete Applied Mathematics*, v. 115, n. 1-3, p. 135–149.

Konak, A. and Smith, A. E. (2011), Efficient optimization of reliable two-node connected networks: a biobjective approach. *INFORMS Journal on Computing*, v. 23, n. 3, p. 430–445.

Kritikos, M. and Ioannou, G. (2017), A greedy heuristic for the capacitated minimum spanning tree problem. *Journal of the Operational Research Society*, v. 68, n. 10, p. 1223–1235.

Kruskal, J. (1956), On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society*, v. 7, n. 1, p. 48–50.

Leitner, M. (2016), Layered graph models and exact algorithms for the generalized hop-constrained minimum spanning tree problem. *Computers & Operations Research*, v. 65, p. 1–18.

Liu, D., Niu, D., Wang, H. and Fan, L. (2014), Short-term wind speed forecasting using wavelet transform and support vector machines optimized by genetic algorithm. *Renewable Energy*, v. 62, p. 592–597.

Magnanti, T. L. and Wolsey, L. A. (1995), Optimal trees. *Handbooks in operations research and management science*, v. 7, p. 503–615.

Mancinia, S. and Steccab, G. (2018), A large neighborhood search based matheuristic for the tourist cruises itinerary planning. *Computers & Industrial Engineering*.

Martinez, L. C. and da Cunha, A. S. (2014), The min-degree constrained minimum spanning tree problem: Formulations and branch-and-cut algorithm. *Discrete Applied Mathematics*, v. 164, p. 210–224.

Mekking, M. and Volgenant, A. (2010), Solving the 2-rooted mini-max spanning forest problem by branch-and-bound. *European Journal of Operational Research*, v. 203, n. 1, p. 50–58.

Myung, Y.-S., Lee, C.-H. and Tcha, D.-W. (1995), On the generalized minimum spanning tree problem. *Networks*, v. 26, n. 4, p. 231–241.

Narula, S. C. and Ho, C. A. (1980), Degree-constrained minimum spanning tree. *Computers & Operations Research*, v. 7, n. 4, p. 239–249.

Nemani, A. K. and Ahuja, R. K. *Minimum Spanning Trees*. Wiley. ISBN 9780470400531, 2011.

Nobari, S., Cao, T.-T., Karras, P. and Bressan, S. Scalable parallel minimum spanning forest computation. *Proceedings of the 17th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '12, p. 205–214, New York, NY, USA. ACM, 2012.

Palomo-Martínez, P. J., Salazar-Aguilar, M. A. and Laporte, G. (2017), Planning a selective delivery schedule through adaptive large neighborhood search. *Computers & Industrial Engineering*, v. 112, p. 368–378.

Pop, P. C., Matei, O., Sabo, C. and Petrovan, A. (2018), A two-level solution approach for solving the generalized minimum spanning tree problem. *European Journal of Operational Research*, v. 265, n. 2, p. 478–487.

Prim, R. (1957), Shortest connection networks and some generalizations. *Bell system technical journal*, v. 36, n. 6, p. 1389–1401.

Ropke, S. and Pisinger, D. (2006), An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation science*, v. 40, n. 4, p. 455–472.

Roy, S., Daripa, K. and Datta, A. K. (2016), k-terminal reliability of d-trapezoid graphs. *IEEE Transactions on Reliability*, v. 65, n. 3, p. 1240–1247.

Ruiz, E., Albareda-Sambola, M., Fernández, E. and Resende, M. G. (2015), A biased random-key genetic algorithm for the capacitated minimum spanning tree problem. *Computers & Operations Research*, v. 57, p. 95–108.

Salama, H. F., Reeves, D. S. and Viniotis, Y. The delay-constrained minimum spanning tree problem. *Proceedings., Second IEEE Symposium on Computers and Communications, 1997.*, p. 699–703. IEEE, 1997.

Santos, J., Pugliese, L. D. P. and Guerriero, F. (2018), A new approach for the multiobjective minimum spanning tree. *Computers & Operations Research*, v. 98, p. 69–83.

Sastry, K., Goldberg, D. E. and Kendall, G. Genetic algorithms. Burke, E. K. and Kendall, G. (Eds.), *Search methodologies*, p. 97–125. Springer, 2014.

Shao, F.-M., Shen, X. and Ho, P.-H. (2005), Reliability optimization of distributed access networks with constrained total cost. *IEEE transactions on reliability*, v. 54, n. 3, p. 421–430.

Shaw, P. Using constraint programming and local search methods to solve vehicle routing problems. *International conference on principles and practice of constraint programming*, p. 417–431. Springer, 1998.

Srivaree-ratana, C., Konak, A. and Smith, A. E. (2002), Estimation of all-terminal network reliability using an artificial neural network. *Computers & Operations Research*, v. 29, n. 7, p. 849–868.

Tarabalka, Y., Chanussot, J. and Benediktsson, J. (2010), Segmentation and classification of hyperspectral images using minimum spanning forest grown from automatically selected markers. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, v. 40, n. 5, p. 1267–1279.

Toth, P. and Vigo, D. (Eds.). *Vehicle Routing: Problems, methods, and applications*. SIAM, Philadelphia, 2nd edio, 2014.

Uchoa, E., Fukasawa, R., Lysgaard, J., Pessoa, A., Poggi de Aragão, M. and Andrade, D. (2008), Robust branch-cut-and-price for the capacitated minimum spanning tree problem over a large extended formulation. *Mathematical Programming*, v. 112, n. 2, p. 443–472.

Watcharasitthiwat, K. and Wardkein, P. (2009), Reliability optimization of topology communication network design using an improved ant colony optimization. *Computers & Electrical Engineering*, v. 35, n. 5, p. 730–747.

Wilcoxon, F. (1945), Individual comparisons by ranking methods. *Biometrics bulletin*, v. 1, n. 6, p. 80–83.

Woolston, K. A. and Albin, S. L. (1988), The design of centralized networks with reliability and availability constraints. *Computers & Operations Research*, v. 15, n. 3, p. 207–217.

Wu, B. and Chao, K. *Spanning trees and optimization problems*. CRC Press, Taylor & Francis Group, Boca Raton, FL, 2004.

Yamada, T. (2009), A mini-max spanning forest approach to the political districting problem. *International Journal of Systems Science*, v. 40, n. 5, p. 471–477.

Yamada, T., Takahashi, H. and Kataoka, S. (1996), A heuristic algorithm for the mini-max spanning forest problem. *European Journal of Operational Research*, v. 91, n. 3, p. 565–572.

Yu, F. and Xu, X. (2014), A short-term load forecasting model of natural gas based on optimized genetic algorithm and improved bp neural network. *Applied Energy*, v. 134, p. 102–113.

Yücel, E., Salman, F. and Arsik, I. (2018), Improving post-disaster road network accessibility by strengthening links against failures. *European Journal of Operational Research*, v. 269, n. 2, p. 406–422.