# Complete Pareto Front of the Biobjective Minimum Length Minimum Risk Spanning Trees Problem

Lasko M. Laskov[1*] and Marin L. Marinov[1]

[1*] Department of Informatics, New Bulgarian University, 21 Montevideo Str., Sofia, 1618, Bulgaria.

Contributing authors: llaskov@nbu.bg; mlmarinov@nbu.bg;

**Abstract**

We analyze in details the biobjective *minimum spanning trees* (MSTs) problem in the case in which the first objective function is a linear one (minimum length) and the second objective function is a non-linear bottleneck (minimum risk). We propose an exact method that constructs the complete Pareto front of the problem with computational complexity $\boldsymbol{O(\beta(m + \alpha n \lg n))}$, where $\boldsymbol{n}$ and $\boldsymbol{m}$ are the sizes of the input network, $\boldsymbol{\alpha}$ is a relatively small parameter, and $\boldsymbol{\beta}$ depends on the number of MST in the original network and its restricted versions that are calculated on each iteration.

As a part of our solution, we propose an efficient exact algorithm that constructs all MSTs for the single-objective linear problem that is based on a modification of the Prim's algorithm. The proposed modification relies on a greedy property that is proved mathematically.

We provide detailed proofs of the correctness of the proposed algorithms, and we also illustrate them with numerical examples.

**Keywords:** combinatorial optimization, biobjective minimum spanning trees, Pareto front

## 1 Introduction

The single-objective minimum spanning tree (MST) problem is defined as the problem of determining a single acyclic subset of the edges of a connected, undirected, weighted graph that connects all its vertices and has minimum total weight [1]. It is a well-known classical problem in the fields of graph algorithms [2] and combinatorial optimization [3], and it has been a subject of extensive research since the early 20th

century. Indeed, the first known description of the MST problem was published by the Czech mathematician Otkar Borůvka in two papers in 1926, in which he formulates a greedy algorithm for construction of an electricity network for Moravia with the usage of minimum possible wire. Latter, his algorithm has been independently reinvented and reported in number of works [4], however the original discovery is attributed to the two papers by Borůvka [5].

With the early works of Borůvka, the MST problem becomes one of the first combinatorial optimization problems ever published, and still it is a subject of extensive research. Besides its direct application in the design of transportation, communication, telecommunication and computer networks [4], it is used as a part of the solution of more complex combinatorial optimization problems, such as the traveling salesman problem [3]. Also, it has applications in image and signal processing, pattern recognition [6] and takes part in clustering algorithms [7].

The MST problem can be solved in polynomial computational complexity and there are three well-known greedy algorithms: the already mentioned Borůvka's algorithm, Kruskal's algorithm [8] and Prim's algorithm [9]. While Kruskal's and Prim's algorithms are considered as standard solutions [1], Borůvka's algorithm more recently again gained attention as the basis of a randomized algorithm with expected linear computational complexity [10], and its application in parallel implementations [11].

The Prim's algorithm [9] has been independently discovered by Dijkstra [12] and some authors refer to it as Prim-Dijkstra [13]. Actually, the algorithm was originally published by the Czech mathematician Vojtěch Jarník in 1930 [4]. In fact, the MST Prim's algorithm and the single-source shortest path Dijkstra's algorithm are nearly identical [14], and both of them share the same greedy property that is manged by a min-priority queue. Thus, the computational complexity of Prim's algorithm can be significantly improved by the implementation of the min-priority queue using the advanced data structure Fibonacci heap [14].

While the standard single-objective MST problem has many applications, in most of the cases in practice they can be modeled more accurately by introduction of an additional constraint or objective function, forming a *biobjective* version of the problem. For example, the biobjective minimum diameter minimum cost spanning tree problem is reprted to model decision problems for high-speed trains infrastructure and design problem for network design and transportation [15]. Another prominent example is the usage of undetermined-delay-constrained MST problem in the design of wireless sensor networks [16]. The biobjective MST problems are reported to have applications in telecommunication networks, computer networks and distribution of dangerous products [17].

However, the introduction of an additional constraint or objective function to the MST problem results in its exponential complexity, and it is considered to be NP-hard [17–20]. Something more, in the presence of two or more objective functions it is no longer possible to find a single optimal solution, and the goal is to find a set of non-dominated MSTs that are also called *Pareto optimal* [21]. The set of Pareto optimal solutions is often referred in literature as *Pareto front* (see for example [20]).

In this paper we examine in details a special case of the biobjective MSTs problem in which the first objective function is linear (minimum length) and the second

objective function is non-linear bottleneck (minimum risk). We propose an exact algorithm that calculates the complete Pareto front of the problem. The computational complexity of the proposed method depends on the number of vertices in the min-priority queue with keys equal to the current minimum and the number of MSTs with the respect to the minimum length objective function.

As part of our solution we describe an exact algorithm that finds all MSTs for the single-objective linear version of the problem. It is a modification of the Prim's algorithm that relies on a greedy property that we prove mathematically. This greedy property allows us simultaneously with the construction of a given MST, to form also the list of initial conditions that help us to construct all the remaining MSTs in the network.

This paper is organized as follows. Sect. 2 gives a brief survey of the related works in the specialized literature. In Sect. 3 we provide problem formulation and basic notations and definitions that are used. In Sect. 4 we describe our method to find a list of all MSTs with respect to the length objective function, and in Sect. 5 we present the solution of the main problem considered in this paper. In the last Sect. 6 we provide our conclusions.

## 2 Related works

In this section we will give a brief survey of the classical and state-of-the-art related works found in the specialized literature. For all described methods we will assume that the input network is defined on a graph $G = (V, E)$, with $V$ denoting the set of vertices and $E$ denoting the set of edges and $n = |V|$, $m = |E|$.

A significant role for the improvement of the computational complexity of notable greedy algorithms on graphs plays the development of advanced data structures that are adopted in the implementation of the abstract data type *priority queue*. Such an important advanced data structure is the *Fibonacci heap*, and it is proposed by Fredman and Tarjan in [14]. For example, the Prim's algorithm when it is enhanced with the Fibonacci heap, in the general case runs with $O(m + n \lg n)$ time complexity, and even with $O(m)$ in the case of sparse graphs for which $m = \Omega(n \lg n)$ [1].

In the work [14] Fredman and Tarjan propose a MST algorithm that is based on Prim's algorithm and runs in $O(m \lg^* n)$ time complexity, where $\lg^* n$ denotes binary iterated logarithm of $n$. Their algorithm was improved by Gabow, Galil, Spencer, and Tarjan [22] to run in $O(m \lg \lg^* n)$, where authors also propose methods to find MSTs with an additional degree constraint in both undirected and directed graphs.

One of the earliest works that discuss the Pareto optimality in the context of MST is the technical note [21] where Corley proposes a generalization of the MST to efficient spanning trees for graph with vector weights. The proposed exact algorithm is related to Prim's algorithm and its goal is to find all Pareto optimal MSTs. The author remarks that graph may contain large number of Pareto optimal MSTs, however he does not investigate the problem for computational complexity of the presented approach.

In [17] the authors study the biobjective MSTs problem in the case of which the two objective functions are linear. They propose an exact method that is composed

by two phases. The first phase constructs those trees whose cost pairs are efficient points on the border of the convex hull of the finite set of solutions. The second phase is based on branch and bound methodology and obtains the rest of efficient spanning trees. The method uses a procedure that finds all spanning trees in a graph and a procedure that extends the Kruskal's algorithm that constructs the whole set single-objective minimum spanning trees. The computational complexity of the proposed method is analyzed based on the experimental results on graphs with $n \in [10, 20]$ and integer edge weights uniformly generated between in the interval $[100, 1000]$.

A number of the heuristic approaches that aim to find a solution of the degree-constrained MST problem are actually based on some modification of the Prim's algorithm [23]. In the latter work, Knowles and Corne show that such methods can be modified to solve the multiobjective MSTs problem. The authors propose a modification of the Prim's algorithm that finds solutions on the convex hull of the Pareto front which are called *supported efficient solutions*. Also, they describe a multiobjective evolutionary algorithm that stores a set of non-dominated solutions that are used to estimate the quality of the solutions that are newly generated.

The evolutionary algorithm described by Knowles and Corne is used for computational experiment comparison by [18] with their memetic algorithm for a biobjective MSTs problem that uses a modification of the Kruskal's algorithm. In the examined case both objective functions are linear. The authors test their method with graphs with $n \in [10, 500]$ and claim that their method finds a higher number of solutions, however they do not discuss the formation of the Pareto front of the problem.

In [19] a method that finds all Pareto optimal solutions of a biobjective MSTs problem with both objective functions being linear, is proposed. The authors first compute extreme efficient solutions and then the remaining non-extreme solutions. For the second phase the authors compare a $k$-best algorithm to a branch-and-bound method. The experiment results show that first approach overperforms the branch-and-bound approach. Also, the authors propose a heuristic enhancement of the speed of the $k$-best algorithm. The method was used for comparison with another two-phase method, presented in the more recent work [24] that finds the complete set of Pareto solutions of the same problem. In the first phase three different strategies are used, including Prim's greedy strategy embedded in a weighted sum approach. The second phase generates all the spanning trees by exploring recursively the structure of the problem and its basic feasible solutions.

A multi-objective metaheuristic approach is proposed in [20] that solves biobjective spanning trees with minimum total cost and minimum diameter problem. The described method is based on multi-objective evolutionary algorithm and on a nondominated sorting genetic algorithm. The same problem is solved with an exact method in [15] where the proposed approach finds the Pareto front of the problem. Both correctness and computational complexity of the algorithm are verified experimentally.

# 3 Problem formulation

Given a connected undirected graph $G = (V, E)$ with set of vertices $V$ and set of edges $E$ and $n = |V|$, $m = |E|$, without loss of generality we will assume that $V = \{1, 2, \ldots, n\}$. Let $f : E \to \mathbb{R}_+$ and $g : E \to \mathbb{R}_+$ are two objective functions defined on set of edges $E$. To each edge $(u, v) \in E$ the function $f$ maps the positive number $f(u, v)$ which we will call *length* of the edge $(u, v)$. To each edge $(u, v) \in E$ the function $g$ maps the positive number $g(u, v)$ which we will call *risk* of the edge $(u, v)$.

Then the network is denoted with $N = (V, E, f, g)$, and we will assume that it is given by its adjacency list:

$$N.Adj = [\langle (v_1, f(1, v_1), g(1, v_1)), (v_2, f(1, v_2), g(1, v_2)), \ldots \rangle, \ldots]. \tag{1}$$

For each subset of edges $A \subseteq E$, $A = \{(u_1, v_1), \ldots, (u_k, v_k)\}$, we define the following two numbers $x(A)$ and $y(A)$.

$$x(A) = \sum_{i=1}^{k} f(u_i, v_i) \tag{2}$$

$$y(A) = \max\{g(u_i, v_i) : i \in \{1, 2, \ldots, k\}\} \tag{3}$$

We will call the number $x(A)$ *length* and the number $y(A)$ *risk* of $A$.

With $W$ we denote the set of all spanning trees in the network $N$. Besides that,

$$W_f = \{t' \in W : x(t') = \min\{x(t) : \forall t \in W\}\}. \tag{4}$$

Every tree from $W_f$ is called *minimum spanning tree* (MST).

**Definition 1** We call the spanning tree $t^* \in W$ *Pareto optimal spanning tree* (POST) when there does not exist another spanning tree $t \in W$ for which any of the following two conditions is fulfilled:

- $x(t) < x(t^*)$ and $y(t) \leq y(t^*)$;
- $x(t) \leq x(t^*)$ and $y(t) < y(t^*)$.

**Definition 2** We will say that the trees $t^*$ and $t$ are *equivalent* when $x(t^*) = x(t)$ and $y(t^*) = y(t)$.

**Definition 3** We will say that the tree $t$ is *dominated* by the tree $t^*$ when $x(t^*) < x(t)$ and $y(t^*) \leq y(t)$ or $x(t^*) \leq x(t)$ and $y(t^*) < y(t)$.

With $P$ we will denote the set of all POSTs in the network $N$, which is also called *Pareto front*. In this paper we will use the representation:

$$P = \bigcup_{i=1}^{K} P_i, \tag{5}$$

where the subsets $P_i$ satisfy the following two conditions.

1. All trees in $P_i$, $i = 1, 2, \ldots, K$, are equivalent among each other.
2. For each index $i = 1, 2, \ldots, K-1$ the following inequalities hold:

$$l_i < l_{i+1} \text{ and } r_i > r_{i+1}, \tag{6}$$

where $l_i$ is the length and $r_i$ is the risk of any tree in $P_i$.

It is clear that the set of all POSTs $P$ has a unique representation (5). We will call each subset $P_i$ from (5) *class of equivalent POSTs*.

The goal of this research is to give a detailed as possible description of the full Pareto front of the biobjective MST problem defined by the network $N = (V, E, f, g)$, and it is formulated in Problem 1.

**Problem 1** (Complete Pareto front) Given the network $N = (V, E, f, g)$ defined by its adjacency list (1), construct the complete Pareto front $P$.

In order to isolate the cases where Problem 1 has an efficient solution, we will solve Problem 2.

**Problem 2** (Partial Pareto front) Given the network $N = (V, E, f, g)$ defined by its adjacency list (1), let $M$ be an arbitrary natural number. Construct the subset $P' \subseteq P$ with the following properties.

1. If $M \geq |P|$, then $P' = P$.
2. If $M < |P|$, then $|P'| = M$.

The subset $P'$ that is defined in Problem 2 allows us to obtain a description of the full Pareto front $P$ and at the same time to control the running time of the method.

# 4 List of all minimum length spanning trees

In this section we consider the standard single-objective MST problem with respect to the length objective function $f$, however we are looking for a description that can produce the set of all MST $W_f$ in the network $N$. Our method is based on the fact that each MST can be constructed using the greedy principle presented in Theorem 1. It allows us simultaneously with the construction of a given MST, to form also the list of initial conditions, that we will refer to as *subproblems*. The list of subproblems can

be used to construct all the remaining MSTs. We note that the described approach below generates a description of all MSTs in the network without actually generating other spanning trees, and is efficient from this point of view.

This task is given formally as Problem 3 below.

**Problem 3** (MSTs list) Given the network $N$ and a natural number $M$, compose a list $W'$ for which the following properties hold.

1. $W' \subseteq W_f$.
2. If $M \geq |W_f|$, then $W' = W_f$.
3. If $M < |W_f|$, then $|W'| = M$.

## 4.1 List of all subproblems procedure

We solve the problem for construction of MSTs list given in Problem 3 with the inductive Procedure 1 that is a modification of the Prim's algorithm [9]. The procedure constructs one MST, denoted with $A$, operates with the set of vertices that are assigned to $A$, denoted with $U$, and the set of vertices that are not yet assigned $Q = V \setminus U$. It starts from a vertex $r$ that is arbitrary selected to be the root of $A$. Initially $U = \{r\}$ and $A = \varnothing$. On each stage of the procedure, the triple $(U, Q, A)$ stores all possibilities to continue with the construction of the MST $A$, and we refer this triple as *subproblem*. In this way, on each iteration Procedure 1 executes two main actions:

- it determines the new edge $(u, v)$ that is assigned to the MST $A$, following the Prim's algorithm;
- it determines the current subproblem $(U, Q, A)$ and stores it in to a stack $S$.

*Procedure 1* (MST and list of all subproblems) The input of the procedure is the triple $(U, Q, A)$ and the stack $S$. Initially, $U = \{r\}$, $A = \varnothing$ and $S = \varnothing$.

1. Calculate the set

$$B = \{(u', v') \in E : u' \in U, v' \in Q, f(u', v') = \min\{f(u, v) : u \in U, v \in Q\}\}. \quad (7)$$

2. For each edge $(u', v') \in B$ define the sets $U' = U \cup \{v_1\}$, $Q' = Q \setminus \{v'\}$, $A' = A \cup \{(u', v')\}$. For each $(u', v')$ push the corresponding subproblem $(U', Q', A')$ into the stack $S$.
3. Let $(U, Q, A)$ be the top element from the stack $S$. Remove the top element from the stack $S$.
4. If $Q \neq \varnothing$, go back to step 1. Otherwise, go to step 5.
5. Return the MST $A$ and the stack of subproblems $S$.

The following Theorem 1 proves that the set $A$ constructed by Procedure 1 is an element of $W_f$. It also poofs that all the rest elements of $W_f$ can be constructed from the subproblems that are stored in the stack $S$.

**Theorem 1** *Let $A$ and $S$ are constructed using Procedure 1. Then the following two statements hold.*

1. *The set of edges $A$ is a MST in the network $N$.*
2. *Every MST of $N$ that is different from $A$ can be constructed using a subproblem stored in $S$.*

*Proof* The proof of the **first statement** of the theorem is analogous to the proof of the Prim's algorithm [1].

Since the empty set is a subset of any MST, we can assume that in the beginning of Procedure 1 the set $A$ is a subset of some MST. After the execution of the step 1, from equation (7) it follows that each edge in $B$ is a light edge that crosses the cut $(U, Q)$. From the definition of the set $U$, it contains all endpoints of edges of $A$ and therefore each edge in $B$ is not contained in $A$. Then, from Theorem 21.1 from [1] it follows that for the new set $A$ defined in step 3, there exists a MST $T$ that contains the new set $A$. It is clear that the extended in this way set $A$ does not contain cycles, since it is a subset of a tree. After the termination of Procedure 1, the set $A$ contains $n-1$ number of edges, and therefore coincides with the MST $T$ in which it is contained.

To prove the **second statement** it is enough to show that for an arbitrary selected MST $T$, Procedure 1 can be executed in such way, so that the MST $A$ that is constructed by it, coincides with $T$.

Let $T \in W_f$. Using mathematical induction we will prove that Procedure 1 can be executed in such way, so that for $1 \leq k \leq n-1$ the following properties hold:

1. The set of edges $A$ has $k$ number of elements.
2. The set $U$ coincides with the vertices that are endpoints of the edges that belong to $A$.
3. $Q = V \setminus U$ and $|Q| = n - k - 1$.
4. $A \subseteq T$.

*Base case.* We will prove that the four properties hold for $k = 1$. At the start of the first iteration, $U = \{r\}$, $Q = V \setminus U$ and $A = \varnothing$. Therefore $A \subseteq T$.

Step 1 calculates the set $B$ for which the following equality holds:

$$B = \{(r, v_1) \in E : v_1 \in Q, f(r, v_1) = \min\{f(r, v) : (r, v) \in E \text{ and } v \in Q\}\}. \tag{8}$$

Step 1 is correct because the network $N$ is connected. The constructed set $B$ contains at least one element.

For each element $(r, v') \in B$ step 2 builds the sets $U' = \{r, v'\}$, $Q' = V \setminus U'$, $A' = \{(r, v')\}$, and pushes the first subproblem $(U', Q', A')$ into the stack $S$. Therefore, by construction for each element $(U', Q', A')$ of the stack $S$ it is fulfilled that $|A'| = 1$, $U'$ is the set with the vertices that are endpoints of the edges that belong to $A'$, $Q' = V \setminus U'$ and $|Q| = n - 2$. This proves that properties 1, 2, and 3 hold for each element stored in $S$.

We will prove by contradiction that in the set $B$ exists an edge $(r, v')$ that belongs to $T$. We assume that for an edge $(r, v') \in B$ and $(r, v') \notin T$. We denote with $\alpha$ the unique path that connects the vertex $r$ with the vertex $v'$ and $\alpha \subseteq T$. We denote with $v''$ the vertex for which $(r, v'') \in \alpha$, and we define:

$$T' = \left(T \setminus \{(r, v'')\}\right) \cup \left\{(r, v')\right\}. \tag{9}$$

It is easy to establish that $T'$ is a MST. From (8) and our assumption it follows that $f(r, v') < f(r, v'')$. From (9) is apparent that the tree $T'$ is derived from the tree $T$, as the edge $(r, v'')$

is replaced by the edge $(r, v')$. Therefore,

$$x(T) - x(T') = f(r, v'') - f(r, v') > 0, \tag{10}$$

which is a contradiction with the fact that $T$ is a MST. From here it follows that in $B$ exists an edge $(r, v')$ that belongs to $T$. Then in step 3 we denote with $(U, Q, A)$ exactly the element of $S$ that is constructed from an edge $(r, v') \in B \cap T$. Then $A \subseteq T$ and the four properties hold for $k = 1$.

*Inductive step.* We assume that for some $k$ satisfying the inequality $1 \leq k < n - 1$, the four properties are satisfied. We will prove that the properties also hold for $k + 1$. In this assumption $|Q| = n - k - 1 > 0$, and therefore the procedure executes its $(k + 1)$ iteration.

Step 1 of the $(k+1)$ iteration of the procedure calculates the set $B$ for which the equality (7) holds. Step 1 is correct because the network $N$ is connected. The constructed set $B$ has at least one element.

Step 2 for each element $(u', v') \in B$ builds the sets $U' = U \cup \{v'\}$, $Q' = V \setminus U'$, $A' = A \cup \{(u', v')\}$ and pushes the corresponding subproblem $(U', Q', A')$ into the stack $S$. Therefore by construction for each newly constructed element $(U', Q', A')$ of the stack $S$ it is fulfilled that $|A'| = k + 1$, $U'$ is the set with the vertices that are endpoints of the edges that belong to $A'$, $Q' = V \setminus U'$ and $|Q| = n - k - 2$. This proves that the properties 1, 2, and 3 hold for $k + 1$ and for each newly constructed element that is stored in $S$. We denote with $S'$ the set of all newly constructed elements in $S$. Since $B \neq \varnothing$, $S' \neq \varnothing$.

We will prove by contradiction that in the set $B$ there exists an edge $(u', v')$ that belongs to $T$. Let $(u', v') \in B$ and $(u', v') \notin T$. We denote with $\alpha$ the unique path that connects the vertex $u'$ with the vertex $v'$ and $\alpha \subseteq T$. Since $u' \in U$ and $v' \in Q$ there exists an edge $(u'', v'')$ from the path $\alpha$ for which $u'' \in U$ and $v'' \in Q$. From (8) and our assumption it follows that $f(u', v') < f(u'', v'')$. We define:

$$T' = \left(T \setminus \{(u'', v'')\}\right) \cup \left\{(u', v')\right\}. \tag{11}$$

It is easy to establish that $T'$ is a MST, and it is derived from the tree $T$, as the edge $(u'', v'')$ is replaced by the edge $(u', v')$. Therefore:

$$x(T) - x(T') = f(u'', v'') - f(u', v') > 0, \tag{12}$$

which is a contradiction with the fact that $T$ is a MST. Therefore, in $B$ there is an edge $(u', v')$ that belongs to $T$. In step 3 we denote with $(U, Q, A)$ exactly this element from $S'$ that is constructed from an edge $(u', v') \in B \cap T$. Then $A \subseteq T$ and the four properties hold for $k + 1$.

From here we can conclude that the four properties hold for $k = n - 1$. For the corresponding set $A$ it holds that $|A| = n - 1$ and $A \subseteq T$. Therefore, $A = T$ and Procedure 1 stops because $|Q| = n - (n - 1) - 1 = 0$. □

**Corollary 1** *Let $(U, Q, A)$ is a subproblem resulting from Procedure 1 or it is a record of the network $N$ when only the root vertex $r$ of the MSTs that are under construction is selected. Then Procedure 1 can be executed in such way that it stops after $k = n - |A| - 1$ iterations and the resulting $A$ is a MST.*

To prove Corollary 1 it is enough to note that the subproblem $(U, Q, A)$ can result from the initial problem after $|A|$ iterations of Procedure 1. Therefore, there exists a tree $T \in W_f$ for which $A \subseteq T$.

## 4.2 List of all subproblems algorithm

Theorem 1 allows us to solve Problem 3 using Procedure 1, which we represent as an algorithm in this section. The algorithmic solution uses the following data structures.

The set of vertices that are not yet assigned to the MST $A$ that is constructed is stored in *min-priority queue* abstract data type $Q$. We implement $Q$ with the advanced data structure Fibonacci heap [14]. Its application plays an important role in the computational complexity analysis of our method. The Fibonacci heap functions that we use in our implementation are given along with their description and computational complexity in Table 1.

**Table 1** Fibonacci heap functions for min-priority queue $Q$

| Function | Description | Complexity |
|---|---|---|
| $Insert(Q, v)$ | inserts an element $v$ into $Q$ | $\Theta(1)$ |
| $Minimum(Q)$ | returns an element with minimum key | $\Theta(1)$ |
| $ExtractMin(Q)$ | extracts and returns the min element from $Q$ | $O(\lg n)$ |
| $DecreaseKey(Q, v, k)$ | decrease the element $v$ with the new key $k$ | $\Theta(1)$ |

The vertices stored in $Q$ are keyed by the distance to the set of the vertices that are already assigned to the $MST$ $A$ that is being constructed. We store the corresponding keys in an array $d$, such that:

$$d[v] = \begin{cases} \min\{f(u,v) : u \in U\}, & \text{if } v \in Q \text{ and } (u,v) \in E \\ \infty, & \text{otherwise.} \end{cases} \tag{13}$$

In the $n$-component array $p$ we store for each vertex $v$ in the min-priority queue $Q$ a list that contains all the vertices $u_i$ that are in $U$ for which $f(u_i, v)$ equals the estimated distance of $v$ to the set $U$ for $i = 1, 2, \ldots, k$, and $k$ is the length of the list:

$$p[v] = \begin{cases} \langle u_1, \ldots, u_k \rangle, u_i \in U, f(u_i, v) = d[v], & \text{if } v \in Q \text{ and } d[v] < \infty \\ \langle \varnothing \rangle, & \text{otherwise.} \end{cases} \tag{14}$$

The implementation of Procedure 1 does not maintain the sets $Q$ and $U$ in explicit form. For each vertex, whether it has been assigned to the MST $A$ that is constructed or not, it is stored in the Boolean array $a$ with $n$ components, such that:

$$a[v] = \begin{cases} false, & \text{if } v \in Q \\ true, & \text{if } v \in U. \end{cases} \tag{15}$$

Also, the MST $A$ that is constructed is stored with the list of parents represented by the $n$-components array $t$. When $A = \varnothing$, all the components of $t$ are zeroes. When an edge $(u, v)$ is assigned to $A$, in the element of $t$ with index $v$ we store the parent vertex $u$. When the algorithm traverses all vertices of the network $N$, it stops and then the only component of the array $t$ that has value 0 is $t[r]$, where $r$ is the root of the MST $A$.

Then, each subproblem $(U, Q, A)$ is stored with a six-tuple of the type:

$$X = (a, Q, d, p, t, v), \tag{16}$$

where $v$ is the latest vertex traversed by the algorithm.

The function $Initialize$ given in Algorithm 1 initializes the data structures for the input network $N$ given by its adjacency list $N.Adj$ and the root vertex $r$. It returns the record of the initial subproblem as a six-tuple of the type (16).

**Algorithm 1** Returns the record of the initial subproblem for the root $r$

```
1:  function Initialize(N.Adj, r)
2:      Q ← V \ {r}
3:      for i ← 1 to n do
4:          a[i] ← false
5:          d[i] ← ∞
6:          p[i] ← ⟨∅⟩
7:          t[i] ← 0
8:      end for
9:      a[r] ← true
10:     for i ← 1 to |N.Adj[r]| do
11:         (w, y, z) ← N.Adj[r][i]
12:         d[w] ← y
13:         p[w] ← ⟨r⟩
14:     end for
15:     return (a, Q, d, p, t, r)
16: end function
```

Algorithm 1 correctly constructs the initial subproblem with computational complexity that does not exceed $O(n + m)$.

*Example 1* Denote with $N_1$ the example network given in Fig. 1. It is composed by 9 vertices and 14 edges, and is defined by the adjacency lists given in Table 2.

**Table 2** Adjacency lists $N_1.Adj[v]$ of the network $N_1$ for each vertex $v$

| $v$ | Adjacency list | $v$ | Adjacency list |
|---|---|---|---|
| 1 | $\langle(2,4,2),(3,8,2)\rangle$ | 2 | $\langle(1,4,2),(3,11,6),(4,8,2)\rangle$ |
| 3 | $\langle(1,8,2),(2,11,6),(5,7,6),(6,1,2)\rangle$ | 4 | $\langle(2,8,2),(5,2,6),(7,4,4),(8,7,8)\rangle$ |
| 5 | $\langle(3,7,6),(4,2,6),(6,6,4)\rangle$ | 6 | $\langle(3,1,2),(5,6,4),(7,2,4)\rangle$ |
| 7 | $\langle(4,4,4),(6,2,4),(8,14,2),(9,10,4)\rangle$ | 8 | $\langle(4,7,8),(7,14,2),(9,9,8)\rangle$ |
| 9 | $\langle(7,10,4),(8,9,8)\rangle$ | | |

The execution of Algorithm 1 for network $N_1$ when the selected root vertex is $r = 1$ is given in Table 3. The result of the two loops of the algorithm and the modified values of the

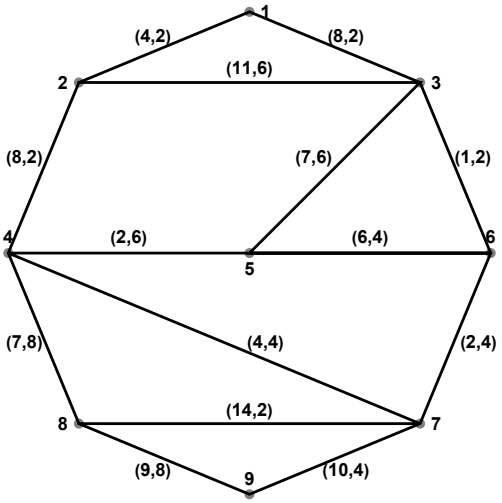


**Fig. 1** Example network $N_1$ composed by 9 vertices and 14 edges. The length and risk of each edge is given as an ordered pair next to it

components of the initial subproblem are given in the second column of the table. The result of the function call $Initialize(N_1, 1)$ is returned as a six-tuple of the type (16).

**Table 3** Calculations of Algorithm 1 for network $N_1$ and root vertex $r = 1$

| Lines | Subproblem components |
|---|---|
| From 3 to 9 | $a = [true, false, false, false, false, false, false, false, false]$<br>$Q = \{2, 3, 4, 5, 6, 7, 8, 9\}$<br>$d = [\infty, \infty, \infty, \infty, \infty, \infty, \infty, \infty, \infty]$<br>$p = [\langle\varnothing\rangle, \langle\varnothing\rangle, \langle\varnothing\rangle, \langle\varnothing\rangle, \langle\varnothing\rangle, \langle\varnothing\rangle, \langle\varnothing\rangle, \langle\varnothing\rangle, \langle\varnothing\rangle]$<br>$t = [0, 0, 0, 0, 0, 0, 0, 0, 0]$ |
| From 10 to 14 | $d = [\infty, 4, 8, \infty, \infty, \infty, \infty, \infty, \infty]$<br>$p = [\langle\varnothing\rangle, \langle 1\rangle, \langle 1\rangle, \langle\varnothing\rangle, \langle\varnothing\rangle, \langle\varnothing\rangle, \langle\varnothing\rangle, \langle\varnothing\rangle, \langle\varnothing\rangle]$ |

The function $ExtractAll$ in Algorithm 2 is an extension of the standard function $ExtractMin$ of the min-priority queue (see Table 1). It extracts all elements from min-priority queue $Q$ that have key that is equal to the current minimum key in the data structure. The function stores the extracted vertices and the corresponding state of the min-priority queue into a *deque* (double-ended queue) $D$ and returns it as a result. The first vertex $u$ that is extracted from $Q$ is used to determine where all consecutively extracted vertices $v$ are going to be pushed into the deque. If there is an edge $(u, v)$ in the network, the pair $(v, Q)$ is pushed into the front of $D$, otherwise it is pushed into the back of $D$. In this way, the order of the pairs in the deque ensures that at the later stages of the method, first the pairs that correspond to vertices that are adjacent to $u$ are going to be processed.

**Algorithm 2** Extracts all vertices with minimal key from min-priority queue $Q$

```
1:  function ExtractAll(Q, d)
2:      let D be an empty deque
3:      m ← d[Minimum(Q)]
4:      u ← ExtractMin(Q)
5:      while m = d[Minimum(Q)] do
6:          v ← ExtractMin(Q)
7:          if (u, v) ∈ E then
8:              PushFront(D, (v, Q))
9:          else
10:             PushBack(D, (v, Q))
11:         end if
12:     end while
13:     return D
14: end function
```

Since the computational complexity of the function $Minimum$ is $\Theta(1)$, the computational complexity of the function $ExtractMin$ is $O(\lg n)$ (see Table 1), the computational complexity of $ExtractAll$ is $O(\alpha \lg n)$ in the general case, where $\alpha$ is the number of vertices with keys that are equal to the current minimum. In the general case, $\alpha$ is much smaller than $n$.

For each element $(v, Q)$ of the deque that is returned by $ExtractAll$ we create a subproblem $X = (a, Q, d, p, t, v)$ that is a parameter of the function $TreeComplete$ that is given in Algorithm 3. The function constructs all the subproblems that can be used to complete the current tree $t$ to a MST and pushes them into the back of another predefined deque that is passed as a parameter.

The function $TreeComplete$ works with the versions of the data structures that are part of the subproblem $X$, but it modifies the deque $D$ that is defined in the caller of the function. Initially, the function sets in the array $a$ of the subproblem $X$ that the vertex $v$ is traversed and it is not part of the set $Q$.

The **for** loop in lines 5 – 15 modifies the array of keys $d$ and the list of parents $p$ that are result from the traversal of the vertex $v$. We denote with $w$ the current vertex from the adjacency list of $v$. If $w$ is not yet traversed, in other words $w \in Q$, then:

- if the current key of $w$ stored in $d[w]$ exceeds the length of the edge $(v, w)$, the key $d[w]$ is updated to store $f(v, w)$, and $p[w]$ stores the list composed only by the vertex $v$;
- otherwise, if the current key of $w$ stored in $d[w]$ equals the length of the edge $(v, w)$, then the array of keys $d$ remains unchanged and the vertex $v$ is inserted into the back of the list $p[w]$.

For each vertex $u$ in the array $p$, the **for** loop in lines 16 – 20 creates a new tree $t'$ in which $u$ is the parent of the $v$. It is used to construct a new subproblem which is then pushed into the back of the deque queue $D$.

Using helper functions $ExtractAll$ and $TreeComplete$ we define the function $Subproblems$ in Algorithm 4 that implements the steps of Procedure 1. The function

**Algorithm 3** All subproblems that can be used to complete the current tree $t$

```
1:  function TreeComplete(X = (a, Q, d, p, t, v), D)
2:      a[v] ← true
3:      d[v] ← ∞
4:      p[v] ← ⟨∅⟩
5:      for i ← 1 to |N.Adj[v]| do
6:          (w, l, ·) ← N.Adj[v][i]
7:          if a[w] = false then
8:              if d[w] > l then
9:                  d[w] ← l
10:                 p[w] ← ⟨v⟩
11:             else if d[w] = l then
12:                 PushBack(p[w], v)
13:             end if
14:         end if
15:     end for
16:     for u ∈ p[v] do                    ▷ exactly when u ∉ Q and f(u, v) = d[v]
17:         t' ← t
18:         t'[v] ← u
19:         PushBack(D, (a, Q, d, p, t', v))
20:     end for
21:     return D
22: end function
```

*Subproblems* takes as parameters a subproblem $X$ and a stack of subproblems $S$. It returns a MST $t$ that arises from $X$ and augments the stack $S$ with all corresponding subproblems that are formed simultaneously with its construction.

**Proposition 1** *The function Subproblems in Algorithm 4 implements Procedure 1.*

*Proof* The formation of the deque $D_1$ on line 3 forms the set $B$ from Procedure 1. The function call $ExtractAll(Q, d)$ finds all vertices $v$ from $Q$ for which there exists a vertex $u$ from $U$, such that the edge $(v, u) \in B$ (equation (7)).

The lines 5 – 13 implement step 2 of the procedure. The **while** loop in lines 5 – 9 constructs all subproblems $(U', Q', A')$. Its correctness follows from the correctness of Algorithm 3. The **while** loop on lines 10 – 13 pushes the elements of the deque $D_2$ into the stack starting from the back of the deque.

Lines 14 and 15 implement step 3 of the procedure.

The outer **while** loop stops when $Q$ becomes the empty set, exactly as Procedure 1. After the termination of the algorithm, the set $A = \{\{t[i], i\} : i \in V \text{ and } i \neq r\}$ is exactly the set $A$ that is constructed by the procedure, and the elements of the stack $S$ are the elements of the set $S$ that result by the procedure. □

Since the computational complexity of the function $ExtractAll(Q, d)$ is $O(\alpha \lg n)$, the complexity of the Algorithm 4 is $O(m+\alpha n \lg n)$, where $\alpha << n$ in the general case.

**Algorithm 4** Construct a MST and a list of all subproblems

1: **function** $Subproblems(X = (a, Q, d, p, t, v),\ S)$
2: **while** $Q \neq \varnothing$ **do**
3: $D_1 \leftarrow ExtractAll(Q, d)$
4: let $D_2$ be an empty deque
5: **while** $D_1 \neq \varnothing$ **do**
6: $(v', Q') \leftarrow Front(D_1)$
7: $PopFront(D_1)$
8: $D_2 \leftarrow TreeComplete((a, Q', d, p, t, v'), D_2)$
9: **end while**
10: **while** $D_2 \neq \varnothing$ **do**
11: $Push(S, Back(D_2))$
12: $PopBack(D_2)$
13: **end while**
14: $(a, Q, d, p, t, v) \leftarrow Top(S)$
15: $Pop(S)$
16: **end while**
17: **return** $t,\ S$
18: **end function**

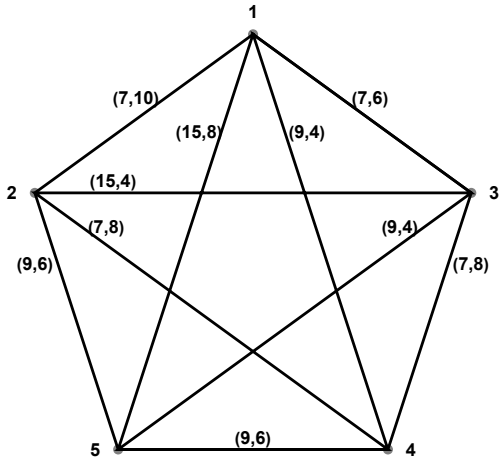


**Fig. 2** Example full network $N_2$ composed by 5 vertices and 14 edges. The length and risk of each edge is given as an ordered pair next to it

*Example 2* The example full network $N_2$ given in Fig. 2 is composed by 5 vertices and 9 edges, and is defined by the adjacency lists given in Table 4. We will trace the execution of Algorithm 4 for the network $N_2$ and the subproblem $X_0$. The subproblems that are computed during the execution of the algorithm are given in Table 5.

In the first iteration of the main loop of the algorithm the inner **while** loop (lines 5 – 9) executes two iterations. In each iteration the function $TreeComplete$ defines a subproblem that can complete the current tree $t$. The first iteration defines the subproblem $X_1$, while the second iteration defines the subproblem $X_2$. These are the subproblems that are denoted

**Table 4** Adjacency lists $N_2.Adj[v]$ of the network $N_2$ for each vertex $v$

| $v$ | Adjacency list | $v$ | Adjacency list |
|---|---|---|---|
| 1 | $\langle(2,7,10),(3,7,6),(4,9,4),(5,15,8)\rangle$ | 2 | $\langle(1,7,10),(3,15,4),(4,7,8),(5,9,6)\rangle$ |
| 3 | $\langle(1,7,6),(2,15,4),(4,7,8),(5,9,4)\rangle$ | 4 | $\langle(1,9,4),(2,7,8),(3,7,8),(5,9,6)\rangle$ |
| 5 | $\langle(1,15,8),(2,9,6),(3,9,4),(4,9,6)\rangle$ | | |

**Table 5** Subproblems and their components calculated by Algorithm 4

| | | | |
|---|---|---|---|
| $X_0$ | $a = [true, false, false, false, false]$<br>$Q = \{2,3,4,5\}$<br>$d = [\infty, 7, 7, 9, 15]$<br>$p = [\langle\varnothing\rangle, \langle1\rangle, \langle1\rangle, \langle1\rangle, \langle1\rangle]$<br>$t = [0,0,0,0,0,0,0,0,0,0]$<br>$v = 1$ | $X_1$ | $a = [true, false, true, false, false]$<br>$Q = \{2,4,5\}$<br>$d = [\infty, 7, \infty, 7, 9]$<br>$p = [\langle\varnothing\rangle, \langle1\rangle, \langle\varnothing\rangle, \langle3\rangle, \langle3\rangle]$<br>$t = [0,0,1,0,0]$<br>$v = 3$ |
| $X_2$ | $a = [true, true, false, false, false]$<br>$Q = \{3,4,5\}$<br>$d = [\infty, \infty, 7, 7, 9]$<br>$p = [\langle\varnothing\rangle, \langle\varnothing\rangle, \langle1\rangle, \langle2\rangle, \langle2\rangle]$<br>$t = [0,1,0,0,0]$<br>$v = 2$ | $X_{11}$ | $a = [true, false, true, true, false]$<br>$Q = \{2,5\}$<br>$d = [\infty, 7, \infty, \infty, 9]$<br>$p = [\langle\varnothing\rangle, \langle1,4\rangle, \langle\varnothing\rangle, \langle\varnothing\rangle, \langle3,4\rangle]$<br>$t = [0,0,1,3,0]$<br>$v = 4$ |
| $X_{12}$ | $a = [true, true, true, false, false]$<br>$Q = \{4,5\}$<br>$d = [\infty, \infty, \infty, 7, 9]$<br>$p = [\langle\varnothing\rangle, \langle\varnothing\rangle, \langle\varnothing\rangle, \langle3,2\rangle, \langle3,2\rangle]$<br>$t = [0,1,1,0,0]$<br>$v = 2$ | $X_{112}$ | $a = [true, true, true, true, false]$<br>$Q = \{5\}$<br>$d = [\infty, \infty, \infty, \infty, 9]$<br>$p = [\langle\varnothing\rangle, \langle\varnothing\rangle, \langle\varnothing\rangle, \langle\varnothing\rangle, \langle3,4,2\rangle]$<br>$t = [0,4,1,3,0]$<br>$v = 2$ |
| $X_{1112}$ | $a = [true, true, true, true, true]$<br>$Q = \{\varnothing\}$<br>$d = [\infty, \infty, \infty, \infty, \infty]$<br>$p = [\langle\varnothing\rangle, \langle\varnothing\rangle, \langle\varnothing\rangle, \langle\varnothing\rangle, \langle\varnothing\rangle]$<br>$t = [0,1,1,3,4]$<br>$v = 5$ | $X_{113}$ | $a = [true, true, true, true, true]$<br>$Q = \{\varnothing\}$<br>$d = [\infty, \infty, \infty, \infty, \infty]$<br>$p = [\langle\varnothing\rangle, \langle\varnothing\rangle, \langle\varnothing\rangle, \langle\varnothing\rangle, \langle\varnothing\rangle]$<br>$t = [0,1,1,3,2]$<br>$v = 5$ |

with $(U', Q', A')$ in Procedure 1. Since the network $N_2$ is relatively small, the results can be verified manually.

The subproblems $X_1$ and $X_2$ are stored into the stack $S$, and the top of the stack is $X_1$.

On lines 14 and 15 the current subproblem $X_1$ is extracted from the stack and $S = \langle X_2\rangle$. The array $a$ is a record of the sets $U = \{1,3\}$ and $Q = \{2,4,4\}$. The array $t$ is a record of the set of edges $A = \{(1,3)\}$. This implements step 3 of Procedure 1 and completes its first iteration.

Since $Q \neq \varnothing$, the outer **while** loop of Algorithm 4 starts its second iteration that implements the second iteration of Procedure 1.

The function call $ExtractAll(Q, d)$, for $Q$ and $d$ defined in $X_1$, returns the deque $D = \langle(4,\{2,5\}),(2,\{4,5\})\rangle$. The inner **while** loop executes two iterations, and the function $TreeComplete$ defines two subproblems that can complete the current tree $t$. The new subproblems are pushed into the stack $S$ and $S = \langle X_2, X_{12}, X_{11}\rangle$, where $X_{11}$ is the top of the stack.

On lines 14 and 15 the third step of the procedure is implemented, and the current subproblem $X_{11}$ is extracted from the stack $S$. Now the min-priority queue stores two vertices and $Q = \{2, 5\}$. Therefore, the outer **while** loop executes third iteration that corresponds to the third iteration of Procedure 1. The third iteration defines the new subproblem $X_{112}$, which is pushed into $S$ and $S = \langle X_2, X_{12}, X_{112}\rangle$.

Line 11 implements the third step of the procedure, and the subproblem $X_{112}$ is extracted from the stack $S$. The corresponding array $a$ defines the sets $U = \{1, 3, 4, 2\}$ and $Q = \{5\}$, while the array $t$ defines the set of edges $A = \{(1, 3), (3, 4), (1, 2)\}$. Since $Q \neq \varnothing$, the algorithm executes its fourth iteration, that corresponds to the fourth iteration of the procedure. Lines 14 and 15 implement the third step of the procedure. The resulting subproblem is composed by $a = [true, true, true, true]$, $Q = \{\varnothing\}$, $d = [\infty, \infty, \infty, \infty, \infty]$ , $p = [\langle\varnothing\rangle, \langle\varnothing\rangle, \langle\varnothing\rangle, \langle\varnothing\rangle, \langle\varnothing\rangle]$, $t = [0, 1, 3, 3]$ and $v = 1$. Since the min-priority queue $Q$ is empty, the algorithm stops and returns the tree $t = [0, 1, 3, 3]$ and the stack $S = \langle X_2, X_{12}, X_{112}, X_{1113}, X_{1112}\rangle$.

The array $t$ is a record of the set of edges $A = \{(1, 3), (3, 4), (1, 2), (3, 5)\}$ that results from Procedure 1, and according to Theorem 1 $A$ is a MST.

## 4.3 List of all MSTs algorithm

The functions *Initialize* given in Algorithm 1 and *Subproblems* given in Algorithm 4 allow us to traverse all MSTs in a network without actually visiting trees that are not MSTs. This allows us to solve Problem 3 with the function *MSTList* given in Algorithm 5.

The function *MSTList* takes as parameters the adjacency lists of the network $N.Adj$, the selected root node $r$ and a predefined natural number $M$ that determines the maximum number of MSTs to include in the list. It returns the list of MSTs $L$ and a Boolean flag *all* that is equal to *true* if all MSTs of the network are included in $L$, and is equal to $false$ when $M$ is less then the total number of MSTs in the network. During its iterations, the algorithm stores the subproblems that define the MSTs that are not yet included in $L$ in a stack $S$.

**Corollary 2** *Let $L$ and all are calculated with Algorithm 5. Then the set of MSTs $W'$ that is stored in $L$ is a solution of Problem 3. If all = true, then $W'$ is the set of all MSTs $W_f$.*

Corollary 2 follows directly from Proposition 1 and Theorem 1. Also, the complexity of Algorithm 5 is $O(\beta(m + \alpha n \lg n))$, where $\beta = \min\{M, |W_f|\}$.

**Corollary 3** *When all edges of the network have the same length, then Algorithm 5 constructs a list of the first $M$ MSTs.*

*Example 3* Given the network $N_2$ with adjacency lists in Table 4 we will calculate:

1. The set $W_1$ composed by 6 MSTs.
2. The set $W_f$ of all MSTs.
3. The set $W$ of all spanning trees.

**Algorithm 5** Construct a list of all MSTs in a network

```
1: function MSTList(N.Adj, r, M)
2:     let S be an empty stack
3:     let L be an empty list
4:     Push(S, Initialize(N.Adj, r))
5:     all ← true
6:     while S ≠ ∅ do
7:         X ← Top(S)
8:         Pop(S)
9:         (t, S) ← Subproblems(X, S)
10:        if t ∉ L then
11:            PushBack(L, t)
12:        end if
13:        if |L| = M then
14:            if S ≠ ∅ then
15:                all ← false
16:            end if
17:            break
18:        end if
19:    end while
20:    return L, all
21: end function
```

**Table 6** All MSTs of the network $N_2$ represented by the list of parents vectors $t_i$, $i = 1, 2, \ldots, 12$

| $i$ | $t_i$ | $i$ | $t_i$ | $i$ | $t_i$ | $i$ | $t_i$ |
|---|---|---|---|---|---|---|---|
| 1 | $[0, 1, 1, 2, 2]$ | 2 | $[0, 1, 1, 2, 3]$ | 3 | $[0, 1, 1, 2, 4]$ | 4 | $[0, 1, 1, 3, 2]$ |
| 5 | $[0, 1, 1, 3, 3]$ | 6 | $[0, 1, 1, 3, 4]$ | 7 | $[0, 1, 4, 2, 2]$ | 8 | $[0, 1, 4, 2, 3]$ |
| 9 | $[0, 1, 4, 2, 4]$ | 10 | $[0, 4, 1, 3, 2]$ | 11 | $[0, 4, 1, 3, 3]$ | 12 | $[0, 4, 1, 3, 4]$ |

1. We calculate $W_1$ with using the function call $MSTList(N_2.Adj, 1, 6)$. The resulting list of MSTs is $L = \langle t_4, t_5, t_6, t_{10}, t_{11}, t_{12} \rangle$. The vectors $t_i$ that represent lists of parents of the MSTs are given in Table 6. The list has 6 element, and since $all = false$, there exists a MST whose list of parents does not belong to $L$.
2. We calculate $W_f$ using the function call $MSTList(N_2.Adj, 1, 16)$. The resulting list of MSTs is $L = \langle t_1, t_2, \ldots t_{12} \rangle$, and $all = true$. From Corollary 2 it follows that $L$ contains the lists of parents of all MSTs of the network $N_2$.
3. To find the set $W$ of all spanning trees, we will use the result from Corollary 3. For this purpose, we define the helper network $N_3$ whose adjacency lists are identical to the adjacency lists of $N_2$ with the difference that all length weight are set equal to 1. Then $T$ is a spanning tree of $N_3$ exactly when it is a spanning tree of $N_2$.

   With the function call $MSTList(N_3.Adj, 1, 130)$ we calculate the list $L$ with 125 elements and $all = true$. Therefore, there are exactly 125 minimum spanning

trees in the network $N_2$. Note that $N_3$ is a full network, and therefore the number of all spanning trees is $n^{n-2} = 5^3 = 125$.

It is clear that as the number $m$ of edges increases, the number of spanning trees of the network grows very quickly. As we see in Example 3, for a network with $m = 10$ and the number of spanning trees is 125. For a full network with 9 vertices and 36 edges, the number of spanning trees is $9^7 = 4782969$. In this case, a large computational resource is required for their calculation.

In the next example we will examine the network $N_1$ (Fig. 1) that has 9 vertices and 14 edges.

*Example 4* Given the network $N_1$ with adjacency lists in Table 2 we will calculate:

1. The set $W_f$ of all MSTs.
2. The set $W$ of all spanning trees.

The solution is analogous to the solution given in Example 3.

1. The function call $MSTList(N_1.Adj, 1, 6)$ results in the list and the Boolean flag:

$$L = \langle [0, 1, 1, 7, 4, 3, 6, 4, 8], [0, 1, 6, 2, 4, 7, 4, 4, 8] \rangle, \ all = true.$$

   From Corollary 2 it follows that $L$ contains the lists of parents of all MSTs of the network $N_1$. Since the number of elements of $|L|$ is 2, $N_1$ has exactly 2 MSTs.
2. We define the helper network $N_4$ whose adjacency lists are identical to the adjacency lists of $N_1$ with all length weights are set equal to 1. With the function call $MSTList(N_4.Adj, 1, 670)$ we calculate the list $L$ that contains 662 elements and $all = true$. For example, $L[1] = [0, 1, 1, 2, 3, 3, 4, 4, 7]$, $L[300] = [0, 1, 2, 5, 6, 3, 6, 7, 8]$, $L[662] = [0, 4, 1, 8, 6, 3, 6, 9, 7]$, which are lists of parents of three MSTs of $N_1$.

   As is known, the number of spanning trees of a network can be calculated using the incidence matrix [2]. This calculation confirms our result.

# 5 Complete Pareto front algorithm

In this section we describe our solution of Problem 2 that describes the full Pareto front of the biobjective MSTs problem.

We denote with $W_1$ the subset of those MSTs that have minimum risk, in other words:

$$W_1 = \{t' \in W_f : y(t') \leq y(t), \forall t \in W_f\}. \tag{17}$$

Besides the functions $Initialize$ given in Algorithm 1 and $MSTList$ given in Algorithm 5, we will use the following functions.

The predicate function $IsConnected(N.Adj)$ performs breadth-first search of the input network and returns $true$ if the network is connected, or $false$ otherwise. Its implementation is a standard application of the breadth-first search algorithmic scheme (see for example [1]).

The function $Risk(t)$ takes as a parameter a vector $t$ that stores the list of parents of a given tree $T$ and returns the risk of the tree $y(T)$.

The function $Restrict(N.Adj, c)$ returns the adjacency lists of a subnetwork $N'$ of the input network $N$ that contains only these edges of $N$ that have risk that is strictly less than $c$.

The function $ImproveRisk$ given in Algorithm 6 takes as parameters a list $L$ of MSTs, represented by their lists of parents, that have risk $c$, the Boolean flag $all$, a MST $t$ and the maximum number of POSTs $M$. The parameter $c$ stores the currently minimum discovered risk, and the flag $all$ is $true$ when $|L| < M$, and $false$ otherwise. The function determines whether the MST $t$ can improve the current value of $c$. If so, the list $L$ stores only the MST $t$ and the current minimum risk $c$ is set to $Risk(t)$. Otherwise, it checks whether $Risk(t) = c$, and if the maximum number of POSTs is not exceeded, pushes $t$ into the back of the list $L$. If the number of elements in $L$ reaches $M$, then all solutions are found, and $all$ becomes $true$.

**Algorithm 6** Improve the current minimum risk with a MST $t$

```
1:  function ImproveRisk(L, c, all, t, M)
2:      if Risk(t) < c then
3:          L ← ⟨t⟩
4:          c ← Risk(t)
5:      else if Risk(t) = c and |L| < M then
6:          if t ∉ L then
7:              PushBack(L, t)
8:          end if
9:          if |L| = M then
10:             all ← true
11:         end if
12:     end if
13:     return L, c, all
14: end function
```

We define the following Problem 4 whose solution will help us to solve Problem 2.

**Problem 4** Let $N$ be a connected network and $M$ be a natural number. Besides that, let $P_1$ be the first class of equivalent POSTs, defined in equation (5). Find a set $W_0$ for which the following properties hold.

1. $W_0 \subseteq P_1$.
2. If $M \geq |P_1|$, then $W_0 = P_1$.
3. If $M < |P_1|$, then $|W_0| = M$.

We will solve Problem 4 with the function $MinRiskMST$ that is given in Algorithm 7. The functions takes as parameters the adjacency lists of the input network $N$, the root node $r$ and the natural number $M$. The result of the function is a list $L$

of the MSTs that have minimum risk, the minimum risk $c$ and a Boolean flag $all$. If $all = true$ then $|L| = M$, and if $all = false$ then $|L| < M$. The algorithm constructs the list $L$ by traversing all MSTs using the function $Subproblems$ given in Algorithm 4. The stack $S$ manages the subproblems that can be used to construct the corresponding MSTs. The variable $c$ stores the current minimum risk found by the algorithm, and the MSTs stored in $L$ have risk equal to $c$.

**Algorithm 7** Compose a list of MSTs with minimum risk

```
1:  function MinRiskMST(N.Adj, r, M)
2:      let S be an empty stack
3:      let L be an empty list
4:      all ← false
5:      c ← ∞
6:      Push(S, Initialize(N.Adj, r))
7:      while S ≠ ∅ do
8:          (a, Q, d, p, t, v) ← Top(S)
9:          Pop(S)
10:         (t, S) ← Subproblems((a, Q, d, p, t, v), S)
11:         (L, c, all) ← ImproveRisk(L, c, all, t, M)
12:         if S ≠ ∅ then
13:             (a, Q, d, p, t, v) ← Top(S)
14:         end if
15:         while S ≠ ∅ and Q = ∅ do
16:             Pop(S)
17:             (L, c, all) ← ImproveRisk(L, c, all, t, M)
18:             if S ≠ ∅ then
19:                 (a, Q, d, p, t, v) ← Top(S)
20:             end if
21:         end while
22:     end while
23:     return L, c, all
24: end function
```

The computational complexity of Algorithm 7 is $O(\beta(m + \alpha n \lg n))$, where $\beta = |W_f|$. This follow from the computational complexity of the function $Subproblems$ and because $MinRiskMST$ traverses only MSTs.

**Lemma 1** *Let $L$ and $all$ are calculated with Algorithm 7. Denote with $W_0$ the set of all MSTs such that*

$$A = \{(t[i], i) : i \in \{1, 3, \ldots, n\}, i \neq r\}, \tag{18}$$

*where $t \in L$. Then $W_0$ is a solution of Problem 4.*

*Proof* The proof follows from the correctness of function $Subproblems$. As Algorithm 7 traverses only MSTs, the function $ImproveRisk$ selects only those of them that have minimum risk.

What remains is to prove that each tree $A \in W_0$ is a POST. Let $T$ be an arbitrary spanning tree. We will assume that $T$ dominates $A$, and we will each contradiction. Since $A$ is a MST, therefore $x(A) \leq x(T)$. From the assumption it follows that $x(A) = x(T)$. Then $T$ is a MST and from the selection of $A$ it follows that $y(A) \leq y(T)$ which contradicts the assumption that $T$ dominates $A$.

Therefore, each tree from $W_0$ is a POST, in other words it holds that $W_0 \subseteq P$. Because by construction each tree from $W_0$ is a MST, it follows that $W_0 \subseteq P_1$. □

*Remark 1* From the correctness of the function $ImproveRisk$ given in Algorithm 6 it follows that $all = true$ exactly when $|L| = M$, and $all = false$ when $|L| < M$.

*Example 5* We will follow the calculations of Algorithm 7 for the example network $N_2$ with adjacency lists given in Table 4, when the selected root is $r = 1$ and $M = 16$.

**Table 7** The tree and stack of subproblems in the first iteration of Algorithm 7

| | | | |
|---|---|---|---|
| | $t = [0, 1, 1, 3, 3]$<br>$S = \langle X_5, X_4, X_3, X_2, X_1 \rangle$,<br>where $Top(S) = X_1$ | $X_1$ | $a = [true, true, true, true, true]$<br>$Q = \{\varnothing\}$<br>$d = [\infty, \infty, \infty, \infty, \infty]$<br>$p = [\langle\varnothing\rangle, \langle\varnothing\rangle, \langle\varnothing\rangle, \langle\varnothing\rangle, \langle\varnothing\rangle]$<br>$t = [0, 1, 1, 3, 4]$<br>$v = 5$ |
| $X_2$ | $a = [true, true, true, true, true]$<br>$Q = \{\varnothing\}$<br>$d = [\infty, \infty, 7, 7, 9]$<br>$p = [\langle\varnothing\rangle, \langle\varnothing\rangle, \langle\varnothing\rangle, \langle\varnothing\rangle, \langle\varnothing\rangle]$<br>$t = [0, 1, 1, 3, 2]$<br>$v = 5$ | $X_3$ | $a = [true, true, true, true, false]$<br>$Q = \{5\}$<br>$d = [[\infty, \infty, \infty, \infty, 9]$<br>$p = [\langle\varnothing\rangle, \langle\varnothing\rangle, \langle\varnothing\rangle, \langle\varnothing\rangle, \langle 3, 4, 2\rangle]$<br>$t = [0, 4, 1, 3, 0]$<br>$v = 2$ |
| $X_4$ | $a = [true, true, true, false, false]$<br>$Q = \{4, 5\}$<br>$d = [\infty, \infty, \infty, 7, 9]$<br>$p = [\langle\varnothing\rangle, \langle\varnothing\rangle, \langle\varnothing\rangle, \langle 3, 2\rangle, \langle 3, 2\rangle]$<br>$t = [0, 1, 1, 0, 0]$<br>$v = 2$ | $X_5$ | $a = [true, true, false, false, false]$<br>$Q = \{3, 4, 5\}$<br>$d = [\infty, \infty, 7, 7, 9]$<br>$p = [\langle\varnothing\rangle, \langle\varnothing\rangle, \langle 1\rangle, \langle 2\rangle, \langle 2\rangle]$<br>$t = [0, 1, 0, 0, 0]$<br>$v = 2$ |

During its first iteration the outer **while** loop determines the MST $t$ and the stack subproblems $S$ using the function $Subproblems$ on line 10 (see Table 7). On line 11 using the function $ImproveRisk$, it determines

$$L = \langle [0, 1, 1, 3, 3] \rangle, c = 10, all = false.$$

The inner **while** loop (lines 15 – 21) iterates while the current subproblem has an empty $Q$ which means that its corresponding $t$ stores a MST. Such subproblems are $X_1$ and $X_2$. For each of them $L$, $c$ and $all$ are recalculated and

$$L = \langle [0, 1, 1, 3, 2], [0, 1, 1, 3, 3], [0, 1, 1, 3, 4] \rangle, c = 10, all = false.$$

Also, from the stack $S$ the first two elements are removed, and now it contains three subproblems.

The outer **while** loop starts its second iteration. After the execution of the functions $Subproblems$ and $ImproveRisk$

$$L = \langle [0,4,1,3,3] \rangle, c = 8, all = false.$$

In this case the minimum risk is improved. The inner **while** loop discovers two more MSTs, and after its two iterations

$$L = \langle [0,4,1,3,2], [0,4,1,3,3], [0,4,1,3,4] \rangle, c = 8, all = false. \tag{19}$$

Next iterations of the algorithm find that all the rest MSTs have risk that is greater than the current minimum risk $c = 8$, and it returns the result given in (19).

Algorithm 7 can be used to construct list of all MSTs, and also list of all spanning trees of the input network $N$. The following two corollaries hold.

**Corollary 4** *Let the network $N'$ has identical adjacency lists to $N$ with all risk weights set to 1. Then Algorithm 7 executed for $N'$ constructs a list of all MSTs of $N$.*

For example, for the network $N_2$ we get the list $L = \langle t_1, t_2, \ldots, t_{12} \rangle$ of all MSTs with lists of parents given in Table 6.

**Corollary 5** *Let the network $N''$ has identical adjacency lists to $N$ with all length and all risk weights set to 1. Then Algorithm 7 executed for $N''$ constructs a list of all spanning trees of $N$.*

Using the functions defined above, Algorithm 8 solves the main Problem 1 for construction of the complete Pareto front. The function $CompletePF$ takes as parameters the adjacency list of the input network $N$, the root vertex $r$ and a natural number $M$. It returns as result the list $L_P$ that contains the discovered classes of equivalent POSTs, a Boolean flag $all$ that determines whether all $M$ number of POSTs are found, and a Boolean flag $isc$ that is $true$ if there are still POSTs that are not discovered and $false$ otherwise.

Further, we will clarify its calculations using the representation (5) of the Pareto front, and using inequalities (6).

The list $L_P$ that results from Algorithm 8 has $j$ elements, where $j \in \{1, 2, \ldots, K\}$. Every element of the list $L_P[k]$ is a list of parent lists of trees.

**Theorem 2** *Let Algorithm 8 has completed after $j$ iterations, where $j \in \{1, 2, \ldots, K\}$. Let us denote with $P'$ the set of trees*

$$A = \{\{t[i], i\} : i \in \{1, 2, \ldots, n\}, i \neq r\}, \tag{20}$$

*where $t \in L_P[k], \forall k \in \{1, 2, \ldots, j\}$. Then the set $P'$ is a solution of Problem 2.*

**Algorithm 8** Construct the complete Pareto front

```
1: function CompletePF(N.Adj, r, M)
2:     let L_P be an empty list
3:     all ← false
4:     isc ← true
5:     while all = false and isc = true do
6:         let L be an empty list
7:         (L, c, all) ← MinRiskMST(N.Adj, r, M)
8:         PushBack(L_P, L)
9:         M ← M − |L|
10:        N.Adj ← Restrict(N.Adj, c)
11:        isc ← IsConnected(N.Adj)
12:    end while
13:    return L_P, all, isc
14: end function
```

*Proof* We will prove the theorem using induction by the iterations of the **while** loop of the algorithm.

*Base case.* It is implemented by the first iteration of the **while** loop. According to Lemma 1 for the set of trees $W_1$ that is defined by the list $L$, it holds that $W_1 \subseteq P_1$. In this case the length of all trees in $W_1$ is $l_1$ and the risk of all trees in $W_1$ is $c = r_1$, where $l_1$ and $r_1$ are defined by inequalities (6).

From Remark 1 it follows that if $all = true$, then $|L| = M$, and then the algorithm will terminate. In this case the list $L_P$ will contain a single element $L$ and $P' = W_1 \subseteq P_1$. Problem 2 is solved, because $M$ number of POSTs are found. Besides that, if $all = false$, then $|L| < M$ and $W_1 = P_1$.

The result of the predicate function $IsConnected$ is stored in the Boolean variable $isc$. If $isc = false$, this means that the restricted network is not connected, therefore, it has no spanning trees and $P_1 = P$. In this case the algorithm terminates. If $all = true$, then $|L| = M$ and $W_1$ contains exactly $M$ number of POSTs. Therefore, Problem 2 is solved. If $all = false$, then $W_1 = P_1$. However, in this case $P_1 = P$ and therefore Problem 2 is again solved, because $P' = W_1$ contains all POSTs.

If $all = false$ and $isc = true$, then the loop will execute its next iteration. In this case $|W_1| < M$ because $all = false$. Then from $isc = true$ it follows that $P \setminus W_1 = \varnothing$ and $W_1 = P_1$.

With this we have proved the states of the sets of POSTs and the Algorithm 8 depending on the values of $all$ and $isc$, that are given in Table 8. In all of the four cases, each tree $A$ from $W_1$ has length $l_1$ and risk $r_1$.

*Inductive step.* We denote with $L_i$ the list returned by the function $MinRiskMST$ and with $N_i$ the network returned by the function *Restrict* on the $i$th iteration of the algorithm. We denote with $W_i$ the set of trees defined by $L_i$ and with $P'$ the set of trees $W_1 \cup \ldots \cup W_i$.

We suppose that $j$ number of iterations of the algorithm were executed, where $1 \leq j < K$. Also, let for each $i \in \{1, \ldots, j\}$ the statements given in Table 9 hold.

It is clear that for $i = 1$ the statement in Table 9 hold, because they were proved in the base step of the induction.

Suppose that after the execution of the $j$th iteration $all = false$ and $isc = true$. We will prove that after the execution of the of the $(j + 1)$ iteration the statements in Table 9 hold.

**Table 8** Sates of the sets of POSTs and Algorithm 8 depending on the values of the two Boolean variables *all* and *isc* for the first iteration

| *all* | *isc* | Sets of POSTs | Algorithm |
|---|---|---|---|
| *true* | *true* | $\|W_1\| = M$, $W_1 \subseteq P_1$, $P' = W_1$, $P \setminus P' \neq \varnothing$ | stops, Problem 2 is solved |
| *true* | *false* | $\|W_1\| = M$, $W_1 \subseteq P_1$, $P' = W_1$, $P = P'$ | stops, Problem 2 is solved |
| *false* | *true* | $\|W_1\| < M$, $W_1 = P_1$, $W_1 = P_1$, $P \setminus W_1 \neq \varnothing$ | iterates |
| *false* | *false* | $\|W_1\| < M$, $P' = W_1 = P_1 = P$ | stops, Problem 2 is solved |

**Table 9** Sates of the sets of POSTs and Algorithm 8 depending on the values of the two Boolean variables *all* and *isc* for the $i$th iteration

| *all* | *isc* | Sets of POSTs | Algorithm |
|---|---|---|---|
| *true* | *true* | $\|W_i\| = M - \|P'\|$, $W_i \subseteq P_i$, $P' = P_1 \cup \ldots \cup P_{i-1} \cup W_i$ contains $M$ POSTs | stops, Problem 2 is solved |
| *true* | *false* | $\|P'\| = M$, $W_i = P_i$, $P \setminus P' = \varnothing$, $P' = P_1 \cup \ldots \cup P_i$ contains $M$ POSTs | stops, Problem 2 is solved |
| *false* | *true* | $\|P'\| < M$, $W_i = P_i$, $P \setminus P' \neq \varnothing$ | iterates |
| *false* | *false* | $\|P'\| < M$, $P' = P_1 \cup \ldots \cup P_i = P$ | stops, Problem 2 is solved |

First we will prove that for the set contained in the list $L_{j+1}$, *all* and $c$, the following properties hold.

1. $W_{j+1} \subseteq P_{j+1}$, where $W_{j+1}$ is the set of POSTs defined by $L_{j+1}$.
2. $c = r_{j+1}$ and each POST from $W_{j+1}$ has length $l_{j+1}$.
3. If $all = true$, then $|W_{j+1}| = M$. Otherwise, if $all = false$, then $W_{j+1} = P_{j+1}$.

The restricted network $N_j$ is result of the function call *Restrict* with risk boundary $c = r_j$. The above three properties follow from the correctness of the function $MinRiskMST$. Indeed, from Lemma 1 it follows that each tree from $W_{j+1}$ is a POST for the network $N_j$ and it has minimum length among all POSTs of $N_j$.

Let $T_0 \in W_{j+1}$. Let us assume that there exists a spanning tree $T$ of the network $N$ that dominates $T_0$. Then $y(T) \leq y(T_0)$ and $T$ is a spanning tree of the network $N_j$. Therefore our assumption leads to a contradiction with the fact that $T_0$ is a POST of the network $N_j$. Something more, we get that the tree $T_0$ has minimum length among all POSTs with risk strictly less than $r_j$. Then, taking into account the definition (5) and the inequalities (6) we find that $x(T_0) = l_{j+1}$ and $y(T_0) = r_{j+1}$. This proves that $W_{j+1} \subseteq P_{j+1}$ and properties 1 and 2.

From Remark 1 it follows that if $all = true$, then $|S_{j+1}| = M_{j+1}$ and obviously $|W_{j+1}| = M_{j+i}$, where $M_{j+1}$ is the value of $M$ on the $(j+1)$ iteration. Otherwise, if $all = false$ then $|S_{j+1}| < M_{j+1}$ and obviously $|W_{j+1}| < M_{j+1}$. When $all = false$ the calculations of the function $MinRiskMST$ stop because outside $W_{j+1}$ there are no other POSTs of $N_j$ with risk $r_{j+1}$. Then $W_{j+1} = P_{j+1}$ and property 3 is also proven.

Let $L_P$ is composed by the $(j+1)$ iteration of the algorithm. Then from the third row of the Table 9 for the $j$th iteration it follows that

$$P' = P_1 \cup \ldots \cup P_j \cup W_{j+1}. \tag{21}$$

From properties 1, 2 and 3 and from equation (21) follow the statements in Table 9 for $(j+1)$ iteration.

From the principle of complete mathematical induction it follows that for $j \in \{1, 2, \ldots, K\}$, if $j$ iterations of the algorithm have been completed, then for each $i \in \{1, 2, \ldots, j\}$ the statements in Table 9 for $i$th iteration are fulfilled. Moreover, the $i$th iteration of the algorithm describes the elements of the class $P_i$. Therefore, after at most $K$ iterations, Algorithm 8 terminates and returns $L_P$, *all* and *isc*. Since the statements in Table 9 for $j$th iteration have been proven for each completed iteration, then the set $P'$ defined by $L_P$ is a solution of Problem 2. Moreover, if $M < |P|$ and the algorithm has completed exactly $j$ iterations, then

$$P' = P_1 \cup \ldots \cup P_{j-1} \cup W_j, \tag{22}$$

where $W_j \subset P_j$ and $P_j \setminus W_j \neq \varnothing$. □

From the proof of Theorem 2 directly follows the next corollary.

**Corollary 6** *Let Algorithm 8 has terminated after $k$ iterations. Then the following statements hold:*

1. $|P'| = M - M_k$.
2. $P' = P_1 \cup P_2 \cup \ldots \cup P_{k-1} \cup W_k$, *where* $W_k \subseteq P_k$.
3. *The list* $L_P$ *represents* $P'$ *and it stores* $\langle P_1, \ldots, P_{k-1}, W_k \rangle$, *where* $W_k \subseteq P_k$.
4. *When* $all = false$, *then in* $P'$ *are stored all POSTs. Otherwise,* $P \setminus P' \neq \varnothing$. *When* $all = true$ *and* $isc = false$, *then* $|P'| = M < |P|$.
5. *The computational complexity of Algorithm 8 is* $O(\beta(m + \alpha n \lg n))$, *where* $\beta = \min\{M, \overline{k}\}$, $\overline{k} = \max\{|W_f|, |W_f^1|, |W_f^2|, \ldots, |W_f^{k-1}|\}$ *and* $W_f^j$ *denotes the set of MST of the restricted network* $N_j$ *after the* $j$*th iteration.*

*Example 6* We will follow the calculations of Algorithm 8 for the input network $N_1$ with adjacency lists given in Table 2. For definiteness, we assume that $r = 1$ and $M = 10$.

The function call $CompletePF(N_1.Adj, 1, 10)$ will stop its calculations after 3 iterations. The values stored in the list $L$, Boolean variable *all*, the current minimum risk $c$, Boolean variable *isc* and current $M$ for each iteration are given in Table 10. The corresponding restricted networks are given in Fig. 3.

**Table 10** Calculations of Algorithm 8 for each its three iterations

| Iteration | $L$ | *all* | $c$ | $N'$ | *isc* | $M$ |
|---|---|---|---|---|---|---|
| 1 | $\langle[0,1,1,7,4,3,6,4,8], [0,1,6,2,4,7,4,4,8]\rangle$ | *flase* | 8 | Fig. 3a | *true* | 8 |
| 2 | $\langle[0,1,1,7,4,3,6,7,7], [0,1,6,2,4,7,4,7,7]\rangle$ | *flase* | 6 | Fig. 3b | *true* | 6 |
| 3 | $\langle[0,1,1,7,6,3,6,7,7], [0,1,6,2,6,7,4,7,7]\rangle$ | *flase* | 4 | Fig. 3c | *false* | 4 |

At the first iteration, the list $L$ contains the lists of parents of two trees. It can be easily verified in Fig. 1 that they define two spanning trees of the restricted network $N_1'$ with risk equal to $c = 8$. In this case $all = false$ and from the correctness of functions $MinRiskMST$ and $ImproveRisk$ it follows that the list $L$ contains all POSTs from the class $P_1$. For completeness, we can directly calculate that their length is $l_1 = 37$. The function *Restric* calculates the restricted network $N'$ given in Fig. 3a. Since $N_1'$ is connected, the value of *isc* is calculated *true*, and the algorithm proceeds to its next iteration.

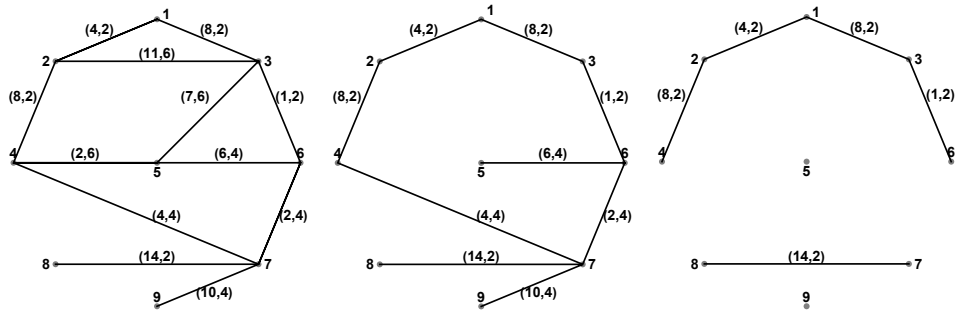


**Fig. 3** The restricted networks $N_1'$ (a) after the first iteration; (b) after the second iteration; (c) after the third iteration

Analogously, the second iteration calculates the list $L$ that contains the list of parents of two trees, that in this case are spanning trees for the network in Fig. 3a. They are the POSTs that form the class $P_2$ and have risk $c = 6$ and length $l = 45$. The restricted network $N_1'$ calculated in the second iteration is given in Fig. 3b, and since it is connected, $isc = true$ and the algorithm continues with its next iteration.

The third iteration of the algorithm calculates the POSTs of the class $P_3$ with risk $c = 4$ and length $l = 49$. In this case the restricted network $N_1'$ (Fig. 3c) is not connected, $isc$ is evaluated to $false$ and the algorithm terminates.

Theorem 2 and Corollary 6 prove that the network $N_1$ has exactly 6 POSTs, that are grouped in 3 classes of equivalent Pareto optimal solutions. Each of these classes contains exactly two trees.

To illustrate the obtained result graphically, we construct the list $W$ of all spanning trees of the network $N_1$. In the examined case their number is 662, which can be easily verified using the incidence matrix of the network. The list $W$ can be composed for example using Corollary 3. To each spanning tree $t$ we map the Cartesian coordinates $(x(t), y(t))$, and in the plane we get the set $\Gamma = \{A(x(t), y(t)) : t \in W\}$. For the network $N_1$, the set $\Gamma$ is composed by 58 points, because the equivalent spanning trees are mapped into the same point $(x(t), y(t))$.

In Fig. 4 the points that correspond to the equivalent classes of POSTs are $A(37, 8)$, $B(45, 6)$ and $C(49, 4)$, and they plotted in gray color. The other points that correspond to equivalent dominated spanning trees, are plotted in black. The following areas are colored in gray: $\delta_A = \{(x, y) : x \geq 37 \text{ and } y \geq 8\}$, $\delta_B = \{(x, y) : x \geq 45 \text{ and } y \geq 6\}$ and $\delta_C = \{(x, y) : x \geq 49 \text{ and } y \geq 4\}$. From Fig. 4 it can be seen that $\Gamma \subset (\delta_A \cup \delta_B \cup \delta_C)$, which means that an arbitrary spanning tree $t$ either belongs to $P$ or is dominated by at least one spanning tree from $P$.

We denote $\gamma_A = \{(x, y) : x \leq 37, y \leq 8 \text{ and } (x, y) \neq (37, 8)\}$, $\gamma_B = \{(x, y) : x \leq 45, y \leq 6 \text{ and } (x, y) \neq (45, 6)\}$ and $\gamma_C = \{(x, y) : x \leq 49, y \leq 4 \text{ and } (x, y) \neq (49, 4)\}$. From Fig. 4 it is seen that $\Gamma \cap (\gamma_A \cup \gamma_B \cup \gamma_C) = \varnothing$. This means that there does not exist a spanning tree $t$ that dominates a tree from the set $P$.

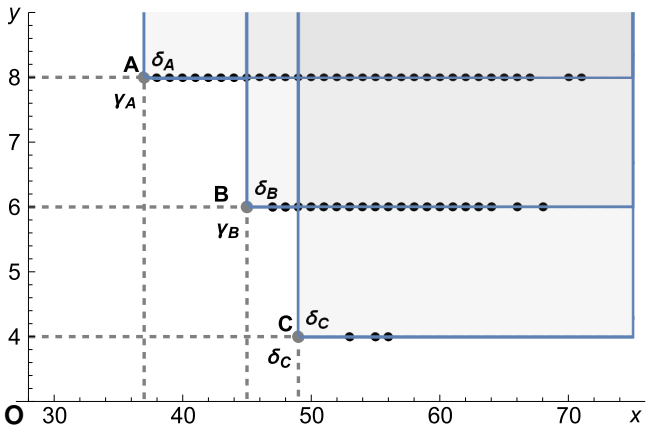


**Fig. 4** The complete Pareto front for the network $N_1$. The classes of equivalent POSTs are illustrated by the points $A$, $B$ and $C$. The remaining points depict equivalent dominated spanning trees

# 6 Conclusion

In this paper we propose an exact method that constructs the complete Pareto front of the biobjective MSTs problem in the case in which the first objective function is linear and represents minimum length, and the second objective function is nonlinear bottleneck and represents minimum risk. We take advantage of the advanced data structure Fibonacci heap that leads to computational complexity of the proposed algorithm $O(\beta(m + \alpha n \lg n))$, where $\alpha$ is relatively small compared to $n$, and $\beta = \min\{M, \overline{k}\}$, $\overline{k} = \max\{|W_f|, |W_f^1|, |W_f^2|, \ldots, |W_f^{k-1}|\}$, $W_f$ denotes the set of MSTs in the original network, and $W_f^j$ denotes the set of MST of the restricted network $N_j$ after the $j$th iteration. The running time of the method allows its application to relatively big networks, even in the extreme case in which the networks are full.

A major part of our solution is the proposed algorithm that constructs all MSTs for the single-objective minimum length problem, which is a problem being of interest of itself. We propose a modification of the Prim's algorithm with an extension of the greedy principle that allows simultaneously to construct a single MST and a list of subproblems that generate all the remaining MSTs of the network. The overall computational complexity of the algorithm that constructs the list of all MSTs represented by their lists of parents is $O(\beta(m + \alpha n \lg n))$, but in this case $\beta = \min\{M, |W_f|\}$.

We have implemented the described algorithms in Wolfram Language and Julia programming language. We have tested our implementations with random full networks networks with number of vertices up to $n = 100$ and number of edges up to $m = (n \times (n-1))/2 = 4950$.

# References

[1] Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to algorithms, 4th edn., pp. 585–603. MIT Press, Cambridge, Massachusetts (2022). Chap. 21

[2] Christofides, N.: Graph theory: An algorithmic approach (Computer science and applied mathematics), pp. 122–149. Academic Press, Inc., USA (1975). Chap. 07

[3] Bazlamaçcı, C.F., Hindi, K.S.: Minimum-weight spanning tree algorithms A survey and empirical study. Computers & Operations Research **28**(8), 767–785 (2001) https://doi.org/10.1016/S0305-0548(00)00007-1

[4] Graham, R.L., Hell, P.: On the history of the minimum spanning tree problem. Annals of the History of Computing **7**(1), 43–57 (1985) https://doi.org/10.1109/MAHC.1985.10011

[5] Nešetřil, J., Milková, E., Nešetřilová, H.: Otakar Borůvka on minimum spanning tree problem Translation of both the 1926 papers, comments, history. Discrete Mathematics **233**(1), 3–36 (2001) https://doi.org/10.1016/S0012-365X(00)00224-7

[6] Tapia, E., Rojas, R.: Recognition of on-line handwritten mathematical expressions using a minimum spanning tree construction and symbol dominance. In: Lladós, J., Kwon, Y.-B. (eds.) Graphics Recognition. Recent Advances and Perspectives, pp. 329–340. Springer, Berlin, Heidelberg (2004). https://doi.org/10.1007/978-3-540-25977-0_30

[7] Gagolewski, M., Cena, A., Bartoszuk, M., Brzozowski: Clustering with minimum spanning trees: how good can it be? Journal of Classification **41** (2024) https://doi.org/10.1007/s00357-024-09483-1

[8] Kruskal, J.B.: On the shortest spanning subtree of a graph and the traveling salesman problem. Proceedings of the American Mathematical Society **7**(1), 48–50 (1956)

[9] Prim, R.C.: Shortest connection networks and some generalizations. The Bell System Technical Journal **36**(6), 1389–1401 (1957) https://doi.org/10.1002/j.1538-7305.1957.tb01515.x

[10] Karger, D.R., Klein, P.N., Tarjan, R.E.: A randomized linear-time algorithm to find minimum spanning trees. Journal of the ACM **42**(2), 321–328 (1995) https://doi.org/10.1145/201019.201022

[11] Bader, D.A., Cong, G.: Fast shared-memory algorithms for computing the minimum spanning forest of sparse graphs. Journal of Parallel and Distributed Computing **66**(11), 1366–1378 (2006) https://doi.org/10.1016/j.jpdc.2006.06.001

[12] Dijkstra, E.W.: A note on two problems in connexion with graphs. Numerische Mathematik **1**(1), 269–271 (1959) https://doi.org/10.1007/bf01386390

[13] Cheriton, D., Tarjan, R.E.: Finding minimum spanning trees. SIAM Journal on Computing **5**(4), 724–742 (1976) https://doi.org/10.1137/0205051

[14] Fredman, M.L., Tarjan, R.E.: Fibonacci heaps and their uses in improved network optimization algorithms. Journal of the ACM **34**(3), 596–615 (1987) https://doi.org/10.1145/28869.28874

[15] de Sousa, E.G., Santos, A.C., Aloise, D.J.: An exact method for solving the bi-objective minimum diameter-cost spanning tree problem. RAIRO-Oper. Res. **49**(1), 143–160 (2014) https://doi.org/10.1051/ro/2014029

[16] Cheng, L., Niu, J., Luo, C., Shu, L., Kong, L., Zhao, Z., Gu, Y.: Towards minimum-delay and energy-efficient flooding in low-duty-cycle wireless sensor networks. Computer Networks **134**(1), 66–77 (2018) https://doi.org/10.1016/j.comnet.2018.01.012

[17] Ramos, R.M., Alonso, S., Sicilia, J., González, C.: The problem of the optimal biobjective spanning tree. European Journal of Operational Research **111**(3), 617–628 (1998) https://doi.org/10.1016/S0022-0000(05)80064-9

[18] Rocha, D.A.M., Goldbarg, E.F.G., Goldbarg, M.C.: A memetic algorithm for the biobjective minimum spanning tree problem. In: Gottlieb, J., Raidl, G.R. (eds.) Evolutionary Computation in Combinatorial Optimization, pp. 222–233. Springer, Berlin, Heidelberg (2006). https://doi.org/10.1007/11730095_19

[19] Steiner, S., Radzik, T.: Computing all efficient solutions of the biobjective minimum spanning tree problem. Computers & Operations Research **35**(1), 198–211 (2008) https://doi.org/10.1016/j.cor.2006.02.023

[20] Santos, A.C., Lima, D.R., Aloise, D.J.: Modeling and solving the bi-objective minimum diameter-cost spanning tree problem. Journal of Global Optimization **60**, 195–216 (2014) https://doi.org/10.1007/s10898-013-0124-4

[21] Corley, H.W.: Efficient spanning trees. Journal of Optimization Theory and Applications **45**, 481–485 (1985) https://doi.org/10.1007/BF00938448

[22] Gabow, H., Galil, Z., Spencer, T., Tarjan, R.: Efficient algorithms for finding minimum spanning trees in undirected and directed graphs. Combinatorica **6**(2), 109–122 (1986) https://doi.org/10.1007/BF02579168

[23] Knowles, J.D., Corne, D.W.: A comparison of encodings and algorithms for multiobjective minimum spanning tree problems. In: Proceedings of the 2001 Congress on Evolutionary Computation (IEEE Cat. No.01TH8546), vol. 1, pp. 544–551. IEEE, Piscataway, NJ (2001). https://doi.org/10.1109/CEC.2001.934439

[24] Amorosi, L., Puerto, J.: Two-phase strategies for the bi-objective minimum spanning tree problem. International Transactions in Operational Research **29**(6), 3435–3463 (2022) https://doi.org/10.1111/itor.13120