Proceedings of the 19th Conference on Computer Science and Intelligence Systems (FedCSIS) pp. 405–416
DOI: 10.15439/2024F2913
ISSN 2300-5963 ACSIS, Vol. 39

Pareto Optimal Solutions of the Biobjective Minimum Length Minimum Risk Spanning Trees Problem

Lasko M. Laskov
0000-0003-1833-818
Informatics Department
New Bulgarian University
21 Montevideo Str., 1618 Sofia, Bulgaria
Email: llaskov@nbu.bg

Marin L. Marinov
0009-0003-9544-819X
Informatics Department
New Bulgarian University
21 Montevideo Str., 1618 Sofia, Bulgaria
Email: mlmarinov@nbu.bg

***Abstract*—We propose an exact method that finds the complete Pareto front of the biobjective minimum length minimum risk spanning trees problem. The proposed method consists of the solution of two problems. The first problem is to compute a list of all minimum spanning trees with respect of the length criterion. The second problem is to construct the complete Pareto front itself, based on the list of all minimum spanning trees, found using the solution of the first problem.**

We prove mathematically the correctness of all proposed algorithms, and we discuss their computational complexity. We also illustrate the presented solution with detailed numerical examples.

I. Introduction

THE minimum spanning tree problem is a classical problem in combinatorial optimization and graph theory. It is considered important because its numerous applications in telecommunication, electronics, clustering algorithms and pattern recognition. The classical well-known algorithms that solve this problem are proposed by Kriskal [1], Prim [2] and Dijkstra [3]. Since then, many algorithms have been published that solve the fundamental version of the problem [4] and its generalization [5].

Even though the solution of the single-criterion problem is essential for the field, many problems in practice require the construction of minimum spanning trees in networks with multiple objective functions, and in particular networks with two objective functions. Both exact and heuristic methods are developed to address the problem of biobjective minimum spanning trees.

In [6] authors propose a method that finds all Pareto optimal solutions of a biobjective minimum spanning trees problem that is based on computation of extreme efficient solutions and then the computation of the remaining non-extreme solutions. The second phase of the method is based on k-best algorithm that is sped up using heuristic enhancement.

In [7] is proposed a multi-objective metaheuristic approach to solve 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. Solution of the same version of the problem is proposed by [8], but this time the proposed method is exact, with both correctness and running time verified experimentally.

In this paper we propose an exact method that finds the complete Pareto optimal front of the biobjective minimum length minimum risk spanning trees problem. Our solution is based on the solution of two problems: (i) construct the list of all minimum spanning trees with respect of the length criterion; (ii) construct the complete list of all Pareto optimal solutions. Our algorithm that solves the first problem is an extension of the Prim's algorithm [2]. It finds a single minimum spanning tree according to the length criterion, with simultaneously completing a list of subproblems that define all the remaining minimum spanning trees in the network. The solution of the second problem is based on an algorithm that composes a list of all Pareto optimal trees that have minimum length.

Even though the classical minimum spanning tree problem is solved in polynomial computational complexity time (see for example [9]), the problem of calculation of the complete Pareto optimal front of the biobjective minimum spanning trees has exponential complexity and is classified as an NP-hard problem [6]. The exact solution that we propose in this paper depends on the number of classes of Pareto optimal solutions and on the number of minimum spanning trees with respect to the length criterion.

This paper is organized as follows. In Sec. II we provide basic notations and definitions used in the paper. In Sec. III we describe the algorithm that computes the list of all minimum spanning trees with respect to the length criterion. In Sec. IV we give the solution of the main problem of this paper and we describe the algorithm that constructs the complete Pareto front. Finally, in Sec. V we provide our conclusions.

II. Basic notations and definitions

Let $G = (V, E)$ is a connected undirected graph with $n = |V|$ number of vertices, and $m = |E|$ number of

edges. Without loss of generality we will assume that $V = \{1, 2, \ldots, n\}$ and $E \subseteq V^2$.

We define the following two objective functions: $f : E \to \mathbb{R}_+$ and $g : E \to \mathbb{R}_+$.

The function $f : E \to \mathbb{R}_+$ assigns to each edge $(u, v) \in E$ the positive real number $f(u, v)$, which we will call *length* of the edge $e = (u, v)$. The function $g : E \to \mathbb{R}_+$ assigns to each edge $(u, v) \in E$ the positive real number $g(u, v)$, which we will call *risk* of the edge $e = (u, v)$.

With N we denote the network $N = (V, E, f, g)$. We assume that the network N is given using the adjacency lists Adj (see Eq. (1)). We will treat the adjacency lists of the network as its attribute and we adopt the notation $N.Adj$.

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

The adjacency lists are represented as an array of lists, where each element of the array $N.Adj[u]$ corresponds to a vertex $u \in V$ of the network and it stores a list composed of ordered triples of the type $(v, f(u, v), g(u, v)$, where $v \in V$ is an adjacent vertex, $f(u, v)$ is the length and $g(u, v)$ is the risk of the edge (u, v).

For each subset of edges $A \subseteq E$, where

$$A = \{(u_1, v_1), \ldots, (u_k, v_k)\}, \tag{2}$$

we define the following two numbers $x(A)$ and $y(A)$.

$$x(A) = \sum_{i=1}^{k} f(u_i, v_i) \tag{3}$$

$$y(A) = \max\{g(u_i, v_i) : i \in \{1, 2, \ldots, k\}\} \tag{4}$$

We call the number $x(A)$ *length* of the subset of edges A, and we call the number $y(A)$ *risk* of the subset of edges A.

Also, if the set A given in (2) is acyclic and if it contains all the vertices of the network N, we will call it a *spanning tree* of N.

With W we denote the set of all spanning trees of the network N.

Definition 1. *We call the tree $t' \in W$* Pareto optimal *when there does not exist another 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 tree t' and t are equivalent and we will denote it with $t' \sim t$, 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' denoted $t \prec t'$, when $x(t') < x(t)$ and $y(t') \leq y(t)$ or $x(t') \leq x(t)$ and $y(t') < y(t)$.*

We denote with P the set of Pareto optimal trees in the network N. It is clear that

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

where P_i are the classes of Pareto equivalent optimal trees, and K is the number of such classes.

III. List of all minimum length spanning trees

In this section we present the method to construct all minimum spanning trees with respect of the length criterion, without actually constructing other spanning trees. The main part of the method is an inductive procedure that traverses a single minimum spanning tree A, starting from its root (Procedure 2). Each iteration of the procedure discovers all the possibilities by which the so far traversed part of A can be completed to another spanning tree of minimum length. These possibilities are called *subproblems* below. The procedure stops when the entire minimum spanning tree A is constructed. In other words, we solve the following Problem 1.

Problem 1. *Given the network $N = (V, E, f, g)$, find one minimum spanning tree A and compose a list of the subproblems that define the remaining minimum spanning trees in the network N.*

We give the solution of the above Problem 1 with the Algorithm 3 which is an extension of the classical Prim's algorithm [2] that finds a single minimum spanning tree in a connected, undirected, weighted graph. The proposed Algorithm 3 simultaneously constructs the minimum spanning tree and composes the list of the subproblems that define the remaining minimum spanning trees in the network N. This strategy is possible because for the minimum spanning tree problem a strong version of the greedy principle, known as *greedy-choice property* [9], is fulfilled, or in other words, the Theorem 2 hods.

In analogy to the Prim's algorithm, using an inductive procedure we discover the edges of a minimum spanning tree and we include them consecutively into a set A. Initially, the set A is the empty set. In Algorithm 3, instead of the set A itself, we explicitly maintain only its storage using the vector $tree$. When $A = \varnothing$, then $tree$ is the zero vector with n components. The inclusion of an edge (u, v) into A we store in $tree$ by $tree[v] = u$. In this case we say that for the currently visited vertex v the vertex u is its parent. We determine the edges that we will include into the set A by traversing the set of vertices V. With Q we denote the set of vertices which are not yet visited by the algorithm. In the initial state, $Q = V$. With U we denote the set of vertices that are already visited. Apparently, $U = V \setminus Q$.

We choose an arbitrary vertex r for the root vertex of the spanning trees. Without loss of generality, we can select $r = 1$. With the selected root r we define $U = \{r\}$ and $Q = V \setminus U$. Then, one minimum spanning tree can be discovered using the following Procedure 1.

Procedure 1. *Inductive procedure to construct single minimum spanning tree.*

I. Base step.

1) *Discover a vertex v_1 from the set Q for which*

$$f(r, v_1) = \min\{f(r, v) : v \in Q\}. \tag{6}$$

2) *Move the vertex v_1 from Q into the set U resulting in the new U and Q after the visiting of the vertex v_1.*

II. Inductive step.

1) *Discover the vertices v_1 and v_2, for which the conditions (7) and (8) are fulfilled.*

$$v_1 \in Q \text{ and } u_1 \in U \tag{7}$$

$$f(u_1, v_1) = \min\{f(u, v) : u \in U \text{ and } v \in Q\} \tag{8}$$

2) *Move the vertex v_1 from Q into the set U resulting in the new U and Q after the visiting of the vertex v_1.*
3) *Include the vertex (u_1, v_1) in A, which we record with $tree[v_1] = u_1$.*
4) *If $Q \neq \varnothing$, go to step II.1. Otherwise, stop.*

It is clear that the inductive Procedure 1 terminates after $n-2$ iterations of the inductive step. After the termination of the procedure, in A are contained $n-1$ edges, and each vertex of the network N is incident with at least one edge of A.

The following theorem holds.

Theorem 1. *The set A that is constructed by the Procedure 1 is a minimum spanning tree.*

Proof: In the proof we follow [9]. Since the empty set is a subset of every spanning tree, we can assume that in the beginning of Procedure 1 the set A is a subset of some minimum spanning tree T. In the execution of step II.1 of the procedure, we find an edge (u_1, v_1), such that it satisfies (7) and (8), which means that the edge (u_1, v_1) is a *light edge* that crosses the *cut* (U, Q). By the definition of the set U, it contains all endpoints of the edges in A, and therefore (u_1, v_1) is not contained in A. Then, from Theorem 23.1 from [9] it follows that there exists a minimum spanning tree T' that contains the set $A \cup \{(u_1, v_1)\}$. It is clear that the set A expanded in this way does not contain cycles because it is a subset of a tree. After the termination of the Procedure 1 the set A contains $n-1$ edges and therefore it coincides with the minimum spanning tree T' in which it is contained. ■

Besides Theorem 1, each minimum spanning tree can be constructed using an appropriate implementation of the Procedure 1. The following theorem holds.

Theorem 2. *Let T_0 be a minimum spanning tree. Then, the tree T_0 can be constructed using the Procedure 1.*

Proof: Without loss of generality, we assume that the root r is the vertex 1.

I. Base step. We calculate:

$$d = \min\{f(1, v) : v \in G.Adg[1]\}, \tag{9}$$

$$d_0 = \min\{f(1, v) : (1, v) \in T_0\}. \tag{10}$$

Apparently, $d \leq d_0$. We will prove that $d = d_0$. For the purpose of contradiction, we assume that $d < d_0$.

With v and v_0 we denote two vertices for which the equalities are fulfilled: $f(1, v) = d$, $f(1, v_0) = d_0$, $(1, v_0) \in T_0$ and $v \in N.Adj[1]$. Then, in the minimum spanning tree T_0 there is a unique path α that has end vertices v_0 and v. Therefore, $\beta = \{(1, v_0)\} \cup \alpha \cup \{(1, v)\}$ is a cycle, and $T_1 = (T_0 \setminus \{(1, v_0)\}) \cup \{(1, v)\}$ is a spanning tree. It is calculated directly that

$$x(T_0) - x(T_1) = f(1, v_0) - f(1, v) = d_0 - d > 0.$$

Then $x(T_0) > x(T_1)$ which contradicts the condition that T_0 is a minimum spanning tree. Therefore, $d = d_0$. This allows Procedure 1 to transfer the vertex v_0 from Q into U and to define $A = \{(1, v_0)\}$. The cut (U, V) respects the set A that is constructed by the above steps, and A is contained in the minimum spanning tree T_0.

II. Inductive step. We assume that the set $A \subseteq T_0$ is constructed using Procedure 1 and that the cut (U, Q) is defined. The set U contains the vertices that are endpoints of the edges that belong to A and $Q = V \setminus U$ is the set of vertices that are not yet visited. Besides that, we will assume that $Q \neq \varnothing$, because otherwise the theorem is proved.

We calculate

$$d_1 = \min\{f(u, v) : u \in U \text{ and } v \in Q\}, \tag{11}$$

$$d_0 = \min\{f(u, v) : u \in U,\ v \in Q \text{ and } (u, v) \in T_0\}. \tag{12}$$

For $i \in \{0, 1\}$ we select vertices u_i and v_i, such that $u_i \in U$, $v_i \in V$, $f(u_i, v_i) = d_i$ and $(u_0, v_0) \in T_0$. Apparently, $d_1 \leq d_0$. We will prove that $d_1 = d_0$. For the purpose of contradiction we assume that $d_1 < d_0$.

Since T_0 is a spanning tree, in T_0 exists a single path γ with end vertices u_1 and v_1. Then there exists at least one edge $(u, v) \in \gamma$, such that $u \in U$ and $v \in V$. From (12) it follows that $d_0 \leq f(u, v)$. From the assumption $d_1 < d_0$ it follows that:

$$f(u_1, v_1) = d_1 < d_0 \leq f(u, v). \tag{13}$$

It is cleat that $\delta = \gamma \cup \{(u_1, v_1)\}$ is a cycle and $T_1 = (T_0 \setminus \{(u, v)\}) \cup \{(u_1, v_1)\}$ is a spanning tree. Therefore, $d_1 = d_0$. This allows Procedure 1 to include into A exactly the edge (u_0, v_0). Following Procedure 1, we move the vertex v_0 from Q into U. Then it is obvious that the new set A is a subset of T_0, the set U contains the vertices of the edges that belong to A, and the number of the vertices that are not visited is decremented with one. After $n-2$ iterations of the inductive step of Procedure 1 we get the set A that coincides with T_0 and the theorem is proved. ■

Theorem 2 allows us to complement Procedure 1 in such a way that, simultaneously with the construction of the minimum spanning tree A, we can also form the list of subproblems with the help of which we can obtain all the remaining minimum spanning trees. We present this extension using Procedure 2. For this purpose, we use the fact that Procedure 1 builds the minimum spanning tree by processing in a standard way the current states of the sets A and Q. This allows us to preserve all possibilities for completing A to a minimum spanning tree that are different from the currently implemented completion. We will store the discovered possibilities in the list S. Such possibilities may appear both in the base step of the procedure and in the inductive step of the procedure.

In Procedure 2 we are not going to use explicitly the set A, but we are going to use its record using the vector $tree$. Besides that, instead of the explicit notation of the sets U and Q, we will store them using the n-components vector $bypassed$.

$$bypassed[v] = \begin{cases} false, & \text{if } v \in Q \\ true, & \text{if } v \in U \end{cases} \tag{14}$$

In this way, we will store each subproblem with the pair $(bypassed, tree)$.

When Procedure 2 starts, the vector $tree$ is the n-component zero vector. Since the vertex r is chosen for the root, all elements of the vector $bypassed$ are $false$, except $bypassed[r]$ which is set to $true$. Thus, the initial problem is stored with the so defined pair of vectors $(bypassed, tree)$, and also the list S is initialized with the empty set.

Procedure 2. *Inductive procedure to construct a minimum spanning tree and all subproblems that define the remaining minimum spanning trees.*

I. Base step.

1) *For each vertex v_1, for which*

$$f(r, v_1) = \min\{f(r, v) : v \neq r\} \tag{15}$$

define:

$$b_1 = bypassed \text{ and } b_1[v_1] = true, \tag{16}$$

$$t_1 = tree \text{ and } t_1[v_1] = r, \tag{17}$$

$$S = \{(b_1, t_1)\} \cup S. \tag{18}$$

2) *We choose one element from S and we denote it with $(bypassed, tree)$. This is the subproblem which the procedure will use in its calculations. We remove $(bypassed, tree)$ from S and proceed to the next step.*
3) *If at least one of the components of $bypassed$ is false, go to the inductive step. Otherwise, stop.*

II. Inductive step.

1) *For each pair of vertices u_1 and v_1 that satisfy the conditions:*

$$bypassed[u_1] = true \text{ and } bypassed[v_1] = false, \tag{19}$$

$$f(u_1, v_1) = \min\{f(u, v) : bypassed[u] = true \text{ and } bypassed[v] = false\}, \tag{20}$$

define

$$b_1 = bypassed \text{ and } b_1[v_1] = true, \tag{21}$$

$$t_1 = tree \text{ and } t_1[v_1] = u_1, \tag{22}$$

$$S = \{(b_1, t_1)\} \cup S. \tag{23}$$

2) *We denote the latest subproblem (b_1, t_1) that is included in S with $(bypassed, tree)$ and the calculations of the procedure will continue with it. We remove $(bypassed, tree)$ from S and proceed to the next step.*
3) *If at least one of the components of the vector $bypassed$ is $false$, go back for next iteration in step II.1. Otherwise, stop.*

We will follow the calculations of Procedure 2.

In step I.1 we discover all vertices v_1, for which the condition (15) is fulfilled. Each such v_1 satisfies the step I.1 of Procedure 1. With equality (16) we define the vector b_1 that stores the new sets U and Q, if the vertex v_1 is visited, which implements step $I.2$ of Procedure 1. Also, with equality (17) we record the fact that by traversing the vertex v_1 the tree A being built is augmented with the edge (r, v_1), which implements step I.3 of Procedure 1. In equality (18), we store in the list S all the subproblems that are obtainable by the base step of Procedure 1.

In step I.2 of Procedure 2 we separate from S the subproblem with which the computations continue.

In step II.1 of Procedure 2 we discover all the pairs of vertices u_1, v_1 that satisfy the conditions of step II.1 of Procedure 1. Also, in equality (21) we store the new sets U and Q, which are defined in step II.2 of Procedure 1. In t_1, defined by (22), we store the new set A, that would be the result of step II.3 of Procedure 1. We store all subproblems found in this way in the list S with equality (23).

In step II.2 of Procedure 2 we select from S a subproblem that does not violate the logic of Procedure 1.

The inductive step of Procedure 2 is repeated until the main tree $tree$ is constructed. We will emphasize that in such a way the main tree $tree$ is built by implementing Procedure 1. According to Theorem 1, the tree $tree$ is a minimum spanning tree.

The following theorem holds.

Theorem 3. *Let $tree$ and S are constructed by Procedure 2. Then the following two statements hold.*

1) *The vector $tree$ defines one minimum spanning tree A.*
2) *Each minimum spanning tree T that is different from A is represented by a subproblem stored in S using the inductive step of Procedure 2.*

Proof: The proof of the theorem follows from Theorem 1 and Theorem 2. Above we have discussed the second statement of the theorem. We only note that applying the inductive step to a subproblem from S computes the next minimum spanning tree and simultaneously completes the set of subproblems S. ■

Theorem 3 allows us to solve Problem 1 by following Procedure 2. Since Procedure 2 is an extension of the Prim's algorithm [2], as in the case of Dijkstra's algorithm [3], we can adopt the implementation of the abstract data type min-priority queue based on Fibonacci heap data structure [10] to significantly speed up the running time of the algorithm. In the case of the Prim's algorithm implemented with Fibonacci heap, the running time achieved is $O(m + n \lg n)$ [11]. In our case, Fibonacci heap is used to implement the min-priority queue Q that stores the set of vertices that is not yet traversed by the algorithm. The vertices that are stored in Q are keyed

by an attribute d, which we will represent as an n-component vector. In the initialization of the algorithm, the components of the vector d are set to ∞. At that stage, the set U of the traversed vertices is the empty set. Each change of the set U leads to a change in the min-priority queue Q together with the components of the vector d, for which it holds:

$$d[v] = \begin{cases} \min\{f(u,v) : u \in U\}, & \text{if } v \in Q \\ \infty, & \text{otherwise} \end{cases} \tag{24}$$

In this way, for a vertex v that is not yet traversed, $d[v]$ is the distance of v to the set of traversed vertices.

The implementation of Procedure 2 manages also the attribute vector p with n components, which stores for each vertex $v \in Q$ all the vertices that are in U for which $f(u,v)$ equals the estimated distance of v to the set U.

$$p[v] = \begin{cases} \{u : u \in U, f(u,v) = d[v]\}, & \text{if } v \in Q,\ d[v] < \infty \\ \{\varnothing\}, & \text{otherwise} \end{cases}$$

We will store each subproblem with the ordered six-tuple:

$$(Q, tree, p, d, bypassed, v), \tag{25}$$

where v is the vertex that was last visited.

In this paper we assume that the network N is given with adjacency lists $N.Adj$, and one of the vertices is selected as the root of the resulting trees. We define a helper function START$(N.Adj, r)$ that takes as input the adjacency lists $N.Adj$ and the selected vertex r for the root of the minimum spanning trees. The function returns the record of the initial problem and the selected root r with the ordered six-tuple of the type (25), where $Q = V \setminus \{r\}$ is min-priority queue keyed by the d attribute vector of the vertices that has components

$$d[w] = \begin{cases} f(r,w), & \text{if } w \in N.Adj[r] \\ \infty, & \text{otherwise} \end{cases} \tag{26}$$

Also, $tree$ is the zero vector, $v = r$, and

$$p[w] = \begin{cases} \{r\}, & \text{if } w \in N.Adj[r] \\ \{\varnothing\}, & \text{otherwise} \end{cases} \tag{27}$$

$$bypassed[w] = \begin{cases} true, & \text{if } w = r \\ false, & \text{if } w \neq r \end{cases} \tag{28}$$

Function START$(N.Adj, r)$ can be implemented, following Algorithm 1 that correctly stores the initial problem with running time complexity that cannot exceed $O(n+m)$.

Example 1. *We denote with N_1 the network given on Fig. 1. It is composed of 5 vertices and 10 edges. On the figure, the weights of each edge are given as ordered pair next to it, where the first element is the length and the second is the risk. The network is defined with the following adjacency lists:*

$$\begin{aligned} N_1.Adj = [&\langle(2,7,10),(3,7,6),(4,9,4),(5,15,8)\rangle, \\ &\langle(1,7,10),(3,15,4),(4,7,8),(5,9,6)\rangle, \\ &\langle(1,7,6),(2,15,4),(4,7,8),(5,9,4)\rangle, \\ &\langle(1,9,4),(2,7,8),(3,7,8),(5,9,6)\rangle, \\ &\langle(1,15,8),(2,9,6),(3,9,4),(4,9,6)\rangle]. \end{aligned} \tag{29}$$

Algorithm 1 Function START$(N.Adj, r)$

Input: adjacency lists $N.Adj$ and root vertex r
Output: the record of the initial problem

```
Q ← V \ {r}
for i ← 1 to |N.V| do
    d[i] ← ∞
    p[i] ← {∅}
    tree[i] ← 0
    if i ≠ r then
        bypassed[i] ← false
    else
        bypassed[i] ← true
    end if
end for
for i ← 1 to |N.Adj[r]| do
    (w, y, z) ← N.Adj[r][i]
    d[w] ← y
    p[w] ← {r}
end for
return (Q, tree, p, d, bypassed, r)
```

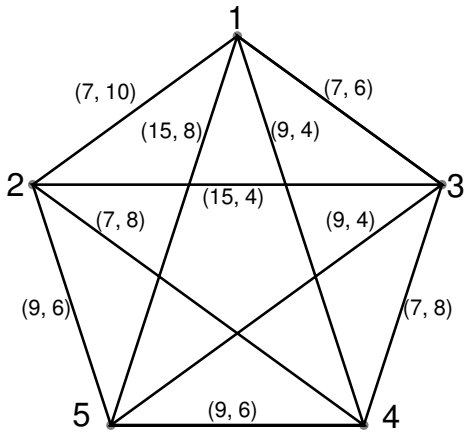

Fig. 1. Example network N_1 composed of five vertices with length and risk of each edge given next to it

We choose for root the vertex $r = 1$, and find the record of the initial problem.

Solution. Algorithm 1 determines:

$$\begin{aligned} Q = \{2,3,4,5\}, tree = (0,0,0,0,0), \\ p = (\{\varnothing\},\{\varnothing\},\{\varnothing\},\{\varnothing\},\{\varnothing\}), \\ d = (\infty,\infty,\infty,\infty,\infty), \\ bypassed = (true, false, talse, false, false). \end{aligned}$$

The adjacency list of the root vertex is $N.Adj[r] = \langle(2,7,10),\ (3,7,6),(4,9,4),(5,15,8)\rangle$.

The second **for** loop edits the vectors d and p. We get: $d = (\infty,7,7,9,15)$ and $p = (\{\varnothing\},\{1\},\{1\},\{1\},\{1\})$. As a result, the function START$(N_1.Adj, r)$ stores the initial problem for the selected root $r = 1$ as the ordered six-tuple:

$$X = (Q, tree, p, d, bypassed, v), \tag{30}$$

Algorithm 2 Function BRANCHING(V_1, S_1)

Input: element $V_1 \in D$ and queue S_1
Output: the modified queue S_1

1: $(v_1, Q_1) \leftarrow V_1$
2: $b_1 \leftarrow bypassed$, $b_1[v_1] \leftarrow true$
3: $d_1 \leftarrow d$, $d_1[v_1] \leftarrow \infty$
4: $p_1 \leftarrow p$, $p_1[v_1] \leftarrow \{\varnothing\}$
5: **for** $i \leftarrow 1$ **to** $|N.Adj[v_1]|$ **do**
6: $(w, l, \cdot) \leftarrow N.Adj[v_1][i]$
7: **if** $b_1[w] = false$ **then**
8: **if** $d_1[w] > l$ **then**
9: $d_1[w] \leftarrow l$, $p_1[w] \leftarrow \{v_1\}$
10: **else if** $d_1[w] = l$ **then**
11: $p_1[w] \leftarrow p_1[w] \cup \{v_1\}$
12: **end if**
13: **end if**
14: **end for**
15: $U \leftarrow p[v_1]$ $\{u \in U$ exactly when $u \notin Q$ and $f(u, v_1) = d[v_1]\}$
16: **for** $i \leftarrow 1$ **to** $|U|$ **do**
17: $u \leftarrow U[i]$
18: $t_1 \leftarrow tree$, $t_1[v_1] \leftarrow u$
19: $S_1 \leftarrow S_1 \cup \{(Q_1, t_1, p_1, d_1, b_1, v_1)\}$
20: **end for**
21: **return** S_1

where

$$Q = \{2, 3, 4, 5\}, tree = (0, 0, 0, 0, 0),$$
$$p = (\{\varnothing\}, \{1\}, \{1\}, \{1\}, \{1\}), d = (\infty, 7, 7, 9, 15),$$
$$bypassed = (true, false, talse, false, false), v = 1.$$

■

To implement Procedure 2 we will use the following three helper functions: EXTRACT(Q), BRANCHING(V_1, S_1) and OPEN-PRIM(X, S), with which we will analyze the current subproblem $(Q, tree, p, d, bypassed, v)$.

The function EXTRACT(Q) discovers all elements v_j in the priority queue Q for which $d[v_j] = \min\{d[v] : v \in Q\}$ and composes a *deque* (double-ended queue) D with elements (v_j, Q_j), where $Q_j = Q \setminus \{v_j\}$. In the beginning of D are stored the elements for which $v \in p[v_j]$.

To each element V_1 of the deque D we apply the helper function BRANCHING(V_1, S_1). It adds to a predefined queue S_1 all subproblems that can complete the current tree $tree$ if an element V_1 of D is selected. The function BRANCHING(V_1, S_1) can be implemented following Algorithm 2, in which we suppose that the queue S_1 is defined, and an element $V_1 \in D$ is chosen.

We will analyze the execution of Algorithm 2. In lines 2 and 3 the algorithm marks the vertex v_1 as traversed. Then it calculates the new values of d_1 and p_1. Initially, it records that v_1 is not an element of Q. Then it changes the values of the estimated distance d and list of parents p when v_1 becomes a traversed vertex. For the resulting d_1 and p_1 the following equalities hold.

Algorithm 3 Function OPEN-PRIM(X, S)

Input: subproblem X and the stack of subproblems S
Output: minimum spanning tree $tree$ and modified S

1: $(Q, tree, p, d, bypassed, v) \leftarrow X$
2: **while** $Q \neq \varnothing$ **do**
3: $D \leftarrow$ EXTRACT(Q)
4: let S_1 be an empty queue
5: **while** $D \neq \varnothing$ **do**
6: $V_1 \leftarrow$ FRONT(D)
7: POPFRONT(D)
8: $S_1 \leftarrow$ BRANCHING(V_1, S_1)
9: **end while**
10: $S \leftarrow S_1 \cup S$ {preserve the order of elements in S_1}
11: $(Q, tree, p, d, bypassed, v) \leftarrow S.top$
12: POP(S)
13: **end while**
14: **return** $tree$, S

1) If w belongs to the set $Q_1 \cap N.Adj(v_1)$, then

$$d_1[w] = \begin{cases} d[w], & \text{if } f(v_1, w) = d[w] \\ f(v, w), & \text{if } f(v_1, w) < d[w] \end{cases} \quad (31)$$

$$p_1[w] = \begin{cases} p[w] \cup \{v_1\}, & \text{if } f(v_1, w) = d[w] \\ \{v_1\}, & \text{if } f(v_1, w) < d[w] \\ p[w], & \text{if } f(v_1, w) > d[w] \end{cases} \quad (32)$$

2) If w belongs to the set $(Q_1 \setminus N.Adj[v_1]) \cup U$, then $d_1[w] = d[w]$ and $p_1[w] = p[w]$.
3) $d_1[v_1] = \infty$ and $p_1[v_1] = \{\varnothing\}$.

Therefore, Algorithm 2 correctly separates all resulting subproblems, if the element V_1 is selected from the deque D.

Using functions EXTRACT(Q) and BRANCHING(V_1, S_1) we define the function OPEN-PRIM(X, S) in Algorithm 3 that implements the inductive step of Procedure 2. We assume that are defined the subproblem $X = (Q, tree, p, d, bypassed, v)$ with $Q \neq \varnothing$ and the stack S of subproblems of the initial problem. The input of the function OPEN-PRIM(X, S) is the subproblem X and the stack S, and its output is a minimum spanning tree stored in the vector $tree$ and the modified stack S. During its execution the function can push new subproblems into S.

Algorithm 3 correctly implements the inductive step of Procedure 2, more precisely, lines from 3 to 10 implement step II.1, and lines 11, 12 implement step II.2.

Indeed, the function EXTRACT(Q) in line 3 of the algorithm finds all vertices v_1 for which there exists such vertex u_1 for which Eq. (19) and (20) hold. In the inner **while** loop the algorithm determines the queue S_1 of all subproblems that can complete the tree $tree$ if v_1 is chosen to be traversed. In this case, we use the correctness of the function BRANCHING(V_1, S_1). This, in particular, guarantees the fulfillment of Eq. (19), (20), (21) and (22). With line 10, the algorithm stores the elements of S_1 in the top of the stack S, preserving their

order, and thus the implementation of step II.2 of Procedure 2 is completed.

The running time of Algorithm 3 depends on the efficiency of the functions of the min-priority queue Q abstract data type. In the case in which Fibonacci heap data structure is used for the implementation of Q, the running time complexity of Algorithm 3 is $O(m + n \lg n)$ (see [11] and also [9]). This is true, because the proposed algorithm is an extension of the Prims's algorithm [2] and differs from it in a constant factor.

We note that the computations in step I.1 of Procedure 2 are a special case of the computations in step II.1. This enables the computations in step 1 to be implemented with the OPEN-PRIM(X, S) function as well.

We will illustrate the above with the following Example 2. In Example 1 we already proved that Problem 1 for the network N_1 when $r = 1$ is written with the ordered six-tuple X defined with Eq. (30). We denote by S an empty stack.

Example 2. *For the subproblem X and stack S defined above, we will prove that the first iteration of the outer while loop of Algorithm 3 implements the base step of Procedure 2.*

Solution. From the definition of X given in Eq. (30) we know that $Q = [2, 3, 4, 5]$ and $d = (\infty, 7, 7, 9, 15)$. Then, the function EXTRACT(Q) defines the deque $D = [\langle 3, (2, 4, 5)\rangle, \langle 2, (3, 4, 5)\rangle]$.

In this case, the inner **while** loop of Algorithm 3 executes two iterations. Each iteration, using the BRANCHING(V_1, S_1) function, defines a subproblem with which the current tree stored in the vector $tree$ can be completed. The first iteration defines the subproblem

$$X_1 = (Q_1, t_1, p_1, d_1, b_1, 3), \tag{33}$$

where

$$Q_1 = [2, 4, 5], t_1 = (0, 0, 1, 0, 0),$$
$$p_1 = (\{\varnothing\}, \{1\}, \{\varnothing\}, \{3\}, \{3\}),$$
$$d_1 = (\infty, 7, \infty, 7, 9), b_1 = (true, false, true, false, false).$$

The second iteration defines the subproblem

$$X_2 = (Q_2, t_2, p_2, d_2, b_2, 2), \tag{34}$$

where

$$Q_2 = [3, 4, 5], t_2 = (0, 1, 0, 0, 0),$$
$$p_2 = (\{\varnothing\}, \{\varnothing\}, \{1\}, \{2\}, \{2\}),$$
$$d_2 = (\infty, \infty, 7, 7, 9), b_2 = (true, true, false, false, false).$$

In this way, step I.1 of Procedure 2 is implemented, which is verified directly. Lines 10, 11 and 12 of Algorithm 3 implement step I.2 of Procedure 2. Thus, the first iteration of the outer **while** loop of the algorithm implements the base step of the procedure. Moreover, following the depth-first search principle, subproblem X_1 remains as the current problem, and only subproblem X_2 remains in the stack S. ■

Example 2 clearly demonstrates that Algorithm 3 implements Procedure 2. In this case, the outer **while** loop will perform $n - 1$ iterations. The first iteration will implement the initial step of the procedure, and the remaining $n-2$ iterations will implement the inductive steps. This allows the solution of Problem 1 to be obtained with Algorithm 4.

Algorithm 4 Function SUBPROBLEMS(N)

Input: the network N
Output: minimum spanning tree $tree$ and subproblems S
1: let S be an empty stack
2: $X \leftarrow$ START$(N.Adj, r)$
3: $(tree, S) \leftarrow$ OPEN-PRIM(X, S)
4: **return** $tree$, S

Example 3. *We will find a single minimum spanning tree and compile a list S of the subproblems that can be used to find the remaining minimum spanning trees of the network N_1 from Example 1 using Algorithm 4.*

Solution. In lines 1 and 2 the algorithm defines an empty stack S and stores the initial problem in the ordered six-tuple $X = (Q, tree, p, d, bypassed, 1)$. In Example 1 we have proved that X is defined with equality (30).

The computations in line 3 are implemented using Algorithm 3. We will follow these calculations for the given case.

In Example 2, we found that the first iteration of the outer **while** loop implements the base step of Procedure 2 and defines as the current subproblem X_1, which is given by equality (33). Additionally, the S stack is also defined, and it contains only the subproblem X_2 which is given by (34).

Since the current $Q = [2, 4, 5]$, the outer **while** loop executes its second iteration, and thus we execute the first iteration of the inductive step of Procedure 2. At that stage, the attribute vector d, that is used as min-priority queue keys, stores $d = (\infty, 7, \infty, 7, 9)$. The function EXTRACT$(Q)$ determines that $D = [\langle 4, (2, 5)\rangle, \langle 2, (4, 5)\rangle]$.

The inner **while** loop of Algorithm 3 executes two iterations. On each iteration, using the function BRANCHING(V_1, S_1), it defines a subproblem with which the tree stored in $tree$ can be completed. In line 10 these two subproblems are stored on the top of the stack S. We get $S = [X_{11}, X_{12}, X_2]$, where the subproblem X_2 is defined by (34), and

$$X_{11} = ((2, 5), (0, 0, 1, 3, 0), (\{\varnothing\}, \{1, 4\}, \{\varnothing\}, \{\varnothing\}, \{3, 4\}), (\infty, 7, \infty, \infty, 9), (true, false, true, true, false), 4), \tag{35}$$

$$X_{12} = ((4, 5), (0, 1, 1, 0, 0), (\{\varnothing\}, \{\varnothing\}, \{\varnothing\}, \{3, 2\}, \{3, 2\}), (\infty, \infty, \infty, 7, 9), (true, true, true, false, false), 2). \tag{36}$$

In line 11, we set the new current subproblem to be X_{11}, and then in line 12 we pop from S the top element. Now, the new queue Q has two elements and $Q = [2, 5]$. Therefore, the outer while loop implements a third iteration and simultaneously starts the execution of the next iteration of the inductive step of Procedure 2. After the execution of the third iteration, line 11 defines the new

current subproblem with $Q = [5]$, $tree = (0, 1, 1, 3, 0)$, $p = (\{\varnothing\}, \{\varnothing\}, \{\varnothing\}, \{\varnothing\}, \{3, 4, 2\})$, $d = (\infty, \infty, \infty, \infty, 9)$, $bypassed = (true, true, true, true, false)$ and $v = 2$.

Besides that, in line 12, the new stack S is defined, which differs from the stack S defined in the previous iteration in that the subproblem is pushed:

$$\begin{aligned} X_{112} = ((5), (0, 4, 1, 3, 0), (\{\varnothing\}, \{\varnothing\}, \\ \{\varnothing\}, \{\varnothing\}, \{3, 4, 2\})(\infty, \infty, \infty, \infty, 9), \\ (true, true, true, true, false), 2). \end{aligned} \tag{37}$$

In this way, $S = [X_{112}, X_{12}, X_2]$.

Since the current $Q \neq \varnothing$, the outer loop of the algorithm executes its fourth iteration and thus the next iteration of the inductive step of Procedure 2. This time, in line 11 the current $Q = \{\varnothing\}$, $tree = (0, 1, 1, 3, 3)$, every element of p is the empty set, every element of d is infinity, every element of $bypassed$ is $true$ and $v = 1$.

Besides that, in the stack S of the previous iteration are pushed two new subproblems:

$$\begin{aligned} X_{1112} = ((\varnothing), (0, 1, 1, 3, 4), (\{\varnothing\}, \{\varnothing\}, \\ \{\varnothing\}, \{\varnothing\}, \{\varnothing\}), (\infty, \infty, \infty, \infty, \infty), \\ (true, true, true, true, true), 5), \end{aligned} \tag{38}$$

$$\begin{aligned} X_{1113} = ((\varnothing), (0, 1, 1, 3, 2), (\{\varnothing\}, \{\varnothing\}, \\ \{\varnothing\}, \{\varnothing\}, \{\varnothing\}), (\infty, \infty, \infty, \infty, \infty), \\ (true, true, true, true, true), 5). \end{aligned} \tag{39}$$

At the end of the fourth iteration, the queue Q is the empty set. Therefore, no further iterations of the outer **while** loop of the algorithm are executed and, moreover, the iterations of the inductive step of Procedure 2 also stop. In this way, the calculations in line 3 of Algorithm 4 stop and it returns:

$$\begin{aligned} tree = (0, 1, 1, 3, 3) \text{ and} \\ S = [X_{1112}, X_{1113}, X_{112}, X_{12}, X_2], \end{aligned} \tag{40}$$

where the elements of the stack S are defined by Eq. (38), (39), (37), (36) and (34).

From Theorem 3 it follows that the tree $tree = (0, 1, 1, 3, 3)$ is a minimum spanning tree. It can be directly computed that the weight according to the length objective function of that tree is 30, and it can be stored as the set of edges $A = \{(1, 2), (1, 3), (3, 4), (3, 5)\}$.

Also, from Theorem 3 follows that that any other minimum spanning tree can be obtained from the stack of subproblems S. In the particular case, directly from Eq. (38) and (39) we establish that the first subproblem defines a tree $(0, 1, 1, 3, 4)$ and the second subproblem defines a tree $(0, 1, 1, 3, 2)$. We can directly verify that the two new spanning trees have length 30 again. ■

We will note that the correctness of Algorithm 4 follows from the correctness of Algorithm 3 and the fact that the first iteration of the outer **while** loop of Algorithm 3 implements the initial step of Procedure 2. Moreover, the running time complexity of Algorithm 4 coincides with the complexity of Algorithm 3.

Corollary 1. *Algorithm 4 can be edited in such a way that it returns a list of all minimum spanning trees.*

For example, using the algorithm from Corollary 1 we get that the network N_1 from Example 1 has exactly 12 minimum spanning trees. These are the trees given in the list S:

$$\begin{aligned} S = \{(0, 1, 1, 3, 3), (0, 1, 1, 3, 4), (0, 1, 1, 3, 2), \\ (0, 4, 1, 3, 3), (0, 4, 1, 3, 4), (0, 4, 1, 3, 2), \\ (0, 1, 1, 2, 3), (0, 1, 1, 2, 2), (0, 1, 1, 2, 4), \\ (0, 1, 4, 2, 2), (0, 1, 4, 2, 4), (0, 1, 4, 2, 3)\}. \end{aligned} \tag{41}$$

IV. Complete Pareto front algorithm

In this section we will give the solution of the main problem considered in this paper.

Problem 2 (Main problem)**.** *Let N be a connected network given with its adjacency lists $N.Adj$. Compose a list P of all classes of equivalent, Pareto optimal trees.*

We will solve Problem 2 using the following Procedure 3.

Procedure 3. *Compose a list of all classes of equivalent, Pareto optimal trees.*

1) *Calculate the minimum length l of a spanning tree, $l = \min\{x(t) : t \in W\}$.*
2) *Calculate the minimum risk r for the minimum spanning trees, $r = \min\{y(t) : t \in W$ and $x(t) = l\}$.*
3) *Define the set T of all spanning trees t for which $x(t) = l$ and $y(t) = r$.*
4) *Define the sets $P = P \cup T$ and $W_1 = \{t : t \in W$ and $y(t) < r\}$.*
5) *If $W_1 \neq \varnothing$, set $W = W_1$ and go to step 1. Otherwise, stop.*

After the termination of Procedure 3 in P are stored all equivalent, Pareto optimal trees.

The following lemma holds.

Lemma 1. *Procedure 3 correctly computes the list P of the classes of equivalent Pareto optimal trees of the network N.*

Proof: It is sufficient to prove that the invariant given in Procedure 3 correctly separates the consecutive class of equivalent Pareto optimal trees.

With steps 1, 2 and 3 we define:

$$\begin{aligned} l_1 = \max\{x(t) : t \in W\}, \\ X_1 = \{t : t \in W \text{ and } x(t) = l_1\}, \\ r_1 = \max\{y(t) : t \in X_1\}, \\ T_1 = \{t : t \in X_1 \text{ and } y(t) = r_1\}. \end{aligned} \tag{42}$$

We denote $W_1 = \{t : t \in W \text{ and } y(t) < r\}$ and $Z_1 = W \setminus (W_1 \cup T_1)$.

Let t_1 be an arbitrary tree from the set T_1. If $Z_1 \neq \varnothing$, then for each $t \in Z_1$ one of the cases (43) or (44) holds.

$$x(t) \geq x(t_1) \text{ and } y(t) > y(t_1) \tag{43}$$

$$x(t) > x(t_1) \text{ and } y(t) = y(t_1) \tag{44}$$

Therefore, the tree t is dominated by the tree t_1, written $t \prec t_1$.

If $W_1 \neq \varnothing$, then for each $t \in W_1$ the inequalities (45) hold.

$$x(t) > x(t_1) \text{ and } y(t) < y(t_1) \tag{45}$$

Therefore, in this case, t and t_1 are not comparable.

As a result, the first iteration of Procedure 3 partitions the set of all minimum spanning trees W into three disjoint subsets T_1, Z_1, and W_1.

If W_1 is the empty set, then Procedure 3 stops and the problem has single class of equivalent, Pareto optimal trees: the class T_1.

If W_1 is not the empty set, then again T_1 is a class of equivalent, Pareto optimal trees, because the elements of W_1 and T_1 are not comparable. Then Procedure 3 returns to step 1 for the next iteration. The next iteration partitions the set W_1 into three disjoint subsets T_2, Z_2, and W_2. In this case:

- T_2 is a class of equivalent, Pareto optimal trees;
- Z_2 contains the elements of W_1 that are dominated by the elements of T_2;
- W_2 contains the elements of W_1 that are not comparable to the elements of T_2.

Since W is a finite set, the procedure stops after finite number of iterations. At the end of the proof, we will emphasize that the procedure makes exactly as many iterations as there are different classes of equivalent, Pareto optimal trees. ■

In the implementation of Procedure 3, besides helper functions START$(N.Adj, r)$ and OPEN-PRIM(X, S) given in Sec. III, we will use the helper functions CONNECTED$(N.Adj)$, RESTRICT$(N.Adj, c)$, NEW(t, T, c) and MPOT$(N.Adj, r)$.

The function CONNECTED$(N.Adj)$ is a predicate function that performs depth-first search in the network, and returns $true$ if N is connected, and $false$ if N is not connected.

The function RESTRICT$(N.Adj, c)$ separates a subnetwork from the network N. The subnetwork contains only those edges of N that have risk strictly less than c. The function returns the adjacency lists of the separated subnetwork.

The function NEW(t, T, c) separates the trees with minimum risk. We assume that T is a list of trees that have risk c. We further assume that c is the minimum currently detected risk. The function NEW(t, T, c) checks whether t can improve the currently minimum c. If this is the case, then we define $T = \{t\}$ and $c = risk(t)$. Also, if $risk(t) = c$ and $t \notin T$, then the function NEW(t, T, c) also adds to the list T the tree t. The function NEW(t, T, c) can be implemented based on the Algorithm 5.

Problem 3. *Let N be a connected network represented by its adjacency lists $N.Adj$. Compose a list T of all Pareto optimal trees that have minimum length.*

The function MPOT$(N.Adj, r)$ given in Algorithm 6 solves Problem 3.

The correctness of the Algorithm 6 follows directly from Lemma 1 and the correctness of Algorithm 3 and Algorithm 5. It is easy to verify that the running time complexity of Algorithm 6 is evaluated to $O(L(m + n \lg n))$, where L is the number of minimum spanning trees and $O(m + n \lg n)$ is the complexity of the function OPEN-PRIM(X, S).

Algorithm 5 Function NEW(t, T, c)

Input: a tree t, list of trees T with risk c
Output: updated list of trees T with risk c

```
1: if risk(t) < c then
2:    T ← {t}
3:    c ← risk(t)
4: else if risk(t) = c and t ∉ T then
5:    T ← T ∪ {t}
6: end if
7: return T, c
```

Algorithm 6 Function MPOT$(N.Adj, r)$

Input: adjacency lists $N.Adj$ and root vertex r
Output: all minimum length Pareto optimal trees T, risk c

```
1: let T be an empty list, S be an empty stack
2: c ← ∞
3: PUSH(S, START(N.Adj, r))
4: while S ≠ ∅ do
5:    X ← S.top
6:    POP(S)
7:    (t, S1) ← OPEN-PRIM(X, S)
8:    (T, c) ← NEW(t, T, c)
9:    while S1 ≠ ∅ and S1[1, 1] = ∅ do
10:      (Q, t, p, d, b, v) ← S1.top
11:      POP(S1)
12:      (T, c) ← NEW(t, T, c)
13:   end while
14:   S ← S1 ∪ S {preserve the order of elements in S1}
15: end while
16: return T, c
```

Example 4. *Compose a list of all minimum spanning trees that are Pareto optimal for the network N_1 in Example 1.*

Solution. We will trace the calculations of the outer **while** loop of Algorithm 6.

First iteration of the loop, in lines 5 and 6 stores the initial problem with the six-tuple X defined with Eq. (30) and empties the stack S. In line 7 the call to the function OPEN-PRIM(X, S) calculates (t, S_1). In Example 3 we provided a detailed proof that $t = tree$ and $S_1 = S$, where $tree$ and S are defined by Eq. (40).

After that, in line 8 the function NEW(t, T, c) edits the current record and we get $c = 10$ and $T = \{(0, 1, 1, 3, 3)\}$.

In lines from 9 to 13, the inner **while** loop performs two iterations and completes the set T. The current record is given in Eq. (46).

$$c = 10, T = \{(0, 1, 1, 3, 3), (0, 1, 1, 3, 4), (0, 1, 1, 3, 2)\} \tag{46}$$

The elements of the stack S_1 are stored on the top of the stack S preserving the order of the elements in S_1 and $S = [X_{112}, X_{12}, X_2]$, where the elements of S are defined

respectively by Eq. (37), (36), and (34). Since $S \neq \varnothing$, the loop proceeds to its second iteration.

Second iteration of the loop applies the function OPEN-PRIM(X, S) to the subproblem X_{112} and the stack $S = [X_{12}, X_2]$. It calculates the minimum spanning tree $t = (0, 4, 1, 3, 3)$ and the stack

$$S_1 = [\langle Q_1, (0,4,1,3,4), p_1, d_1, b_1, 5\rangle, \langle Q_2, (0,4,1,3,2), p_2, d_2, b_2, 5\rangle],$$

where

$$Q_1 = Q_2 = [\varnothing], p_1 = p_2 = (\{\varnothing\}, \{\varnothing\}, \{\varnothing\}, \{\varnothing\}, \{\varnothing\}),$$
$$d_1 = d_2 = (\infty, \infty, \infty, \infty, \infty),$$
$$b_1 = b_2 = (true, true, true, true, true).$$

In line 8 the function NEW(t, T, c) improves the current record and we get $c = 8$ and $T = \{(0, 4, 1, 3, 3)\}$.

In lines from 9 to 13, the inner **while** loop again performs two iterations and completes the set T. The resulting current record is:

$$c = 8, T = \{(0,1,1,3,3), (0,4,1,3,4), (0,4,1,3,2)\}. \quad (47)$$

Since the stack $S = [X_{12}, X_2]$ is not empty, the loop proceeds to its next iteration.

The outer **while** loop executes six more iterations. The minimum spanning trees that are discovered by these iterations have a risk greater than $c = 8$. Therefore, the current record does not change. The function MPOT$(N.Adj, r)$ returns the set T and the risk c which are defined by Eq. (47).

From the correctness of the function OPEN-PRIM(X, S) it follows that the function MPOT$(N.Adj, r)$ has traversed all minimum spanning trees. From the correctness of the function NEW(t, T, c) it follows that in T are separated the minimum spanning trees that have minimum risk.

It is easy to observe that in such a way the function MPOT$(N.Adj, r)$ implements the first iteration of Procedure 3. Then, from Lemma 1, in particular, it follows that T is a class of equivalent, Pareto optimal spanning trees. ■

Corollary 2. *Let the network N_2 be obtained from the network N with the risk of each edge changed to 1. Then the function call* MPOT$(N_2.Adj, r)$ *composes the list T_1 of all minimum spanning trees of the network N.*

For example, for the network N_1 of Example 1, we get that $T_1 = S$, where S is given by Eq. (41).

Corollary 3. *Let the network N_3 be obtained from the network N with both risk and length of each edge changed to 1. Then the function call* MPOT$(N_3.Adj, r)$ *composes the list T_2 of all spanning trees of the network N.*

Using the helper functions defined above, we will solve the main Problem 2. The proposed solution is Algorithm 7.

The list P that results from Algorithm 7 is a solution of the main Problem 2. This follows directly from Lemma 1 and the fact that Algorithm 7 implements Procedure 3.

Indeed, let us denote with W the set of all spanning trees of the network N. The first iteration of the algorithm separates

Algorithm 7 Function CPOT$(N.Adj, r)$

Input: adjacency lists $N.Adj$ and root vertex r
Output: all classes of equivalent, Pareto optimal trees P

```
1: let P be an empty list
2: ind ← true
3: while ind = true do
4:     (T, c) ← MPOT(N.Adj, r)
5:     P ← P ∪ {T}
6:     N.Adj ← RESTRICT(N.Adj, c)
7:     ind ← CONNECTED(N.Adj)
8: end while
9: return P
```

the set T from those minimum spanning trees that have minimum risk. We denote with c the risk and with l the length of an arbitrary tree of T.

In line 5 of Algorithm 7 the set T is included into the list P. Then we denote $W_1 = \{t : t \in W \text{ and } y(t) < c\}$. This implements steps from 1 to 4 of Procedure 3.

In line 6 of Algorithm 7 the function RESTRICT$(N.Adj, c)$ separates the subnetwork N' that contains only those edges of the network N that have a risk strictly less than c. We note that a tree t belongs to W_1 if and only if it is a spanning tree of the subnetwork N'. Therefore, $W_1 \neq \varnothing$ if and only if the subnetwork N' is connected. This proves that Algorithm 7 will execute next iteration exactly when Procedure 3 executes its next iteration.

The **while** loop terminates when the subnetwork N' is not connected and all K number of classes of Pareto equivalent trees are discovered. Therefore, from the computational complexity of the function MPOT$(N.Adj, r)$ it follows that Algorithm 7 has running time $O(KL(m + n \lg n))$.

The following Example 5 clarifies the proof of the correctness of Algorithm 7.

Example 5. *We will examine the network N_5 that is composed by 9 vertices and 14 edges, and is defined by the adjacency lists given in (48).*

$$\begin{aligned} N_5.Adj = [&\langle (2,4,2), (3,8,2)\rangle, \\ &\langle (1,4,2), (3,11,6), (4,8,2)\rangle, \\ &\langle (1,8,2), (2,11,6), (5,7,6), (6,1,2)\rangle, \\ &\langle (2,8,2), (5,2,6), (7,4,4), (8,7,8)\rangle, \\ &\langle (3,7,6), (4,2,6), (6,6,4)\rangle, \\ &\langle (3,1,2), (5,6,4), (7,2,4)\rangle, \\ &\langle (4,4,4), (6,2,4), (8,14,2), (9,10,4)\rangle, \\ &\langle (4,7,8), (7,14,2), (9,9,8)\rangle, \\ &\langle (7,10,4), (8,9,8)\rangle] \end{aligned} \quad (48)$$

Using Algorithm 7 we will compose a list of all classes of equivalent Pareto optimal trees.

Solution. We set $r = 1$ and denote with W the set of all spanning trees of the network N_5.

First iteration of the **while** loop of the algorithm, using the function MPOT$(N.Adj, r)$ in line 4 calculates

$$T_1 = \{\{(0,1,6,2,4,7,4,4,8), (0,1,1,7,4,3,6,4,8)\}\} \text{ and } c_1 = 8. \tag{49}$$

We define the sets

$$W_1 = \{t : t \in W \text{ and } y(t) < c_1\},$$
$$Z_1 = \{t : t \in W, y(t) \geq c_1 \text{ and } t \notin T_1\}.$$

Then, obviously, the three sets T_1, Z_1, and W_1 have no common elements and

$$W = T_1 \cup Z_1 \cup W_1. \tag{50}$$

Let $t_0 \in T_1$ and $t \in Z_1 \cup W_1$. From the correctness of the function MPOT$(N.Adj, r)$ it follows that $x(t_0) \leq x(t)$. Moreover, if $t \in Z_1$, then the following two cases are possible:

- $x(t_0) < x(t)$ and $y(t_0) = c_1 = y(t)$;
- $x(t_0) \leq x(t)$ and $y(t_0) = c_1 < y(t)$.

Therefore, $t \prec t_0$.

If $t \in W_1$, then $y(t_0) = c_1 > y(t)$. From the correctness of the function MPOT$(N.Adj, r)$ it follows that $x(t_0) < x(t)$. Then, t_0 and t are not comparable. Therefore, every tree $t_0 \in T_1$ is Pareto optimal, and in line 5 of the algorithm T_1 is correctly included in the list P.

We denote with l_1 the length of any tree $t_0 \in T_1$. In the considered example $l_1 = x(t_0) = 37$.

In line 6 the subnetwork N' is defined that contains only those edges of N that have risk strictly less than $c_1 = 8$. We get

$$\begin{aligned} N'.Adj = [&\langle(2,4,2),(3,8,2)\rangle,\\ &\langle(1,4,2),(3,11,6),(4,8,2)\rangle,\\ &\langle(1,8,2),(2,11,6),(5,7,6),(6,1,2)\rangle,\\ &\langle(2,8,2),(5,2,6),(7,4,4)\rangle,\\ &\langle(3,7,6),(4,2,6),(6,6,4)\rangle,\\ &\langle(3,1,2),(5,6,4),(7,2,4)\rangle,\\ &\langle(4,4,4),(6,2,4),(8,14,2),(9,10,4)\rangle,\\ &\langle(7,14,2)\rangle,\langle(7,10,4)\rangle]. \end{aligned} \tag{51}$$

We note that W_1 is the set of all spanning trees of the subnetwork N'. This fact can be verified directly by proving that a tree $t \in W_1$ if and only if t is a spanning tree of N'.

Let $t \in W_1$. Then t is a spanning tree of N and, in particular, every vertex of N' is incident to an edge of t. Furthermore, every edge of t, by the definition of W_1 , has a risk strictly less than c_1 . Therefore, t is a spanning tree of N'. Analogously, it is verified that if t is a spanning tree of N', then $t \in W_1$.

In line 7 it is verified that the network N' is connected, and the loop proceeds to its second iteration. In particular, this means that $W_1 \neq \varnothing$.

Second iteration of the **while** loop, using function call MPOT$(N'.Adj, r)$ calculates

$$T_2 = \{\{(0,1,6,2,4,7,4,7,7), (0,1,1,7,4,3,6,7,7)\}\} \text{ and } c_2 = 6. \tag{52}$$

We denote with l_2 the length of any tree $t_0 \in T_2$. In the considered example, $l_2 = x(t_0) = 45$. Then we define the sets

$$W_2 = \{t : t \in W_1 \text{ and } y(t) < c_2\},$$
$$Z_2 = \{t : t \in W_1, y(t) \geq c_2 \text{ and } t \notin T_2\}.$$

Then apparently the three sets T_2, Z_2 and W_2 have no common elements and

$$W_1 = T_2 \cup Z_2 \cup W_2. \tag{53}$$

Then the equality (50) is written in the form

$$W = T_1 \cup Z_1 \cup T_2 \cup Z_2 \cup W_2. \tag{54}$$

Let $t_0 \in T_2$. We will prove that t_0 is a Pareto optimal tree.

Let t be an arbitrary tree from the set $W \backslash T_2$. Then obviously the following four cases are possible: 1) $t \in W_2$; 2) $t \in Z_2$; 3) $t \in Z_1$ and 4) $t \in T_1$. Repeating the reasoning from the first iteration of the loop, we immediately find the following.

1) If $t \in W_2$, then t and t_0 cannot be compared.
2) If $t \in Z_2$, then $t \prec t_0$.
3) If $t \in Z_1$ two subcases are possible:
   - $x(t) < l_2$ and then t and t_0 cannot be compared;
   - $x(t) \geq l_2$ and then $t \prec t_0$.
4) If $t \in T_1$, then t and t_0 cannot be compared because $t_0 \in W_1$.

Therefore, every tree $t_0 \in T_2$ is Pareto optimal, and in line 5 of the algorithm T_2 is correctly included in the list P.

In line 6 the subnetwork N'' is defined that contains only those edges of N that have risk strictly less than $c_2 = 6$. The resulting adjacency lists of the network N'' is:

$$\begin{aligned} N''.Adj = [&\langle(2,4,2),(3,8,2)\rangle,\langle(1,4,2),(4,8,2)\rangle,\\ &\langle(1,8,2),(6,1,2)\rangle,\langle(2,8,2),(7,4,4)\rangle,\\ &\langle(6,6,4)\rangle,\langle(3,1,2),(5,6,4),(7,2,4)\rangle,\\ &\langle(4,4,4),(6,2,4),(8,14,2),(9,10,4)\rangle,\\ &\langle(7,14,2)\rangle,\langle(7,10,4)\rangle]. \end{aligned} \tag{55}$$

As above, we find that W_2 is the set of all spanning trees of the subnetwork N''. In line 7 of the algorithm we find that N'' is connected and the loop proceeds to its third iteration.

Third iteration of the **while** loop calls MPOT$(N''.Adj, r)$ and calculates

$$T_3 = \{\{(0,1,6,2,6,7,4,7,7), (0,1,1,7,6,3,6,7,7)\}\} \text{ and } c_3 = 4. \tag{56}$$

We denote with l_3 the length of any tree $t_0 \in T_3$. In the considered example $l_3 = x(t_0) = 49$. Then we define the sets

$$W_3 = \{t : t \in W_2 \text{ and } y(t) < c_3\},$$
$$Z_3 = \{t : t \in W_1, y(t) \geq c_3 \text{ and } t \notin T_3\}.$$

Then, apparently, the three sets T_3, Z_3 and W_3 have no common elements and

$$W = T_1 \cup Z_1 \cup T_2 \cup Z_2 \cup T_3 \cup Z_3 \cup W_3. \tag{57}$$

Every tree t_0 of T_3 is Pareto optimal. The proof is completely analogous to the proof that T_2 contains only Pareto optimal trees. Therefore, T_3 is correctly included in the list P.

TABLE I
CLASSES OF PARETO OPTIMAL TREES OF THE NETWORK N_5

P	Class Pareto optimal spanning trees	(l, c)
T_1	$\{(0,1,6,2,4,7,4,4,8),\ (0,1,1,7,4,3,6,4,8)\}$	$(37,8)$
T_2	$\{(0,1,6,2,4,7,4,7,7),\ (0,1,1,7,4,3,6,7,7)\}$	$(45,6)$
T_3	$\{(0,1,6,2,6,7,4,7,7),\ (0,1,1,7,6,3,6,7,7)\}$	$(49,4)$

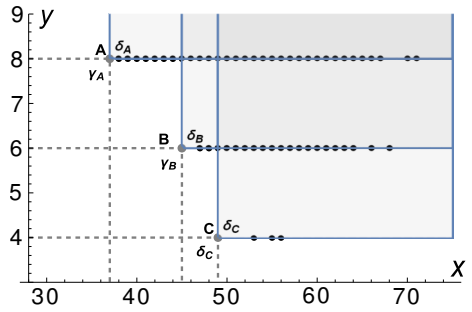

Fig. 2. The Pareto front of the biobjective spanning trees of Example 5

In line 6 is defined the subnetwork N''' that contains only those edges of N that have a risk strictly less than $c_3 = 4$. We get

$$N'''.Adj = [\langle(2,4,2),(3,8,2)\rangle, \langle(1,4,2),(4,8,2)\rangle, \langle(1,8,2),(6,1,2)\rangle, \langle(2,8,2)\rangle, \langle\varnothing\rangle, \langle(3,1,2)\rangle, \langle(8,14,2)\rangle, \langle(7,14,2)\rangle, \langle\varnothing\rangle]. \quad (58)$$

As above, we find that $t \in W_3$ if and only if t is a spanning tree of N'''. In line 7 we find that N''' is not connected and therefore $W_3 = \varnothing$. Also, the algorithm stops.

In this way, it is proved that the list P contains all classes of Pareto optimal trees.

The classes of Pareto optimal trees of the network N_5 are given in Table I. In the examined case, each class of equivalent Pareto optimal trees has two elements. ■

In order to illustrate graphically the obtained result in Example 5, we compose the list W of all spanning trees of the network N_5. In this case their number is 662 which can be easily achieved using the incidence matrix. The list W can be composed using Corollary 3. To each spanning tree t we correspond a point A_t with Cartesian coordinates $(x(t), y(t))$. In the plane is obtained the set

$$\Gamma = \{A(x(t), y(t)) : t \in W\}.$$

The set Γ has 58 points because the equivalent spanning trees are mapped to the same point on the plane.

In Figure 2, the points that illustrate the classes of equivalent Pareto optimal trees are in gray color, and the rest are in black color. More precisely, the points $A(37,8)$, $B(45,6)$, and $C(49,4)$ are in gray, and the remaining points of Γ are in black.

V. CONCLUSION

In this paper we propose an exact method that constructs the complete Pareto front of the minimum length minimum risk spanning trees problem. It is composed of the solution of two problems. For the solution of the first problem, the method calculates the list of all minimum spanning trees with respect to the length criterion. For the solution of the second (main) problem, it constructs the complete Pareto front itself, using the solution of the first problem to compose each of the classes of equivalent Pareto optimal trees.

The Algorithm 3 proposes an extension of the Prim's algorithm that allows us simultaneously to find a single minimum spanning tree and the complete list of all remaining minimum spanning trees, defined by their corresponding subproblems. This modification also uses Algorithm 2 that defines a branching that adds to a queue all subproblems that can complete the current tree. Because of the Fibonacci heap implementation of the min-priority queue abstract data type, the complexity of the algorithm that solves the first problem is $O(m + n \lg n)$.

In order to solve the main problem considered, we use the solution of the helper problem that gives us Algorithm 6 that composes a list of all Pareto optimal trees that have minimum length. The computational complexity of the final solution given in Algorithm 7 is $O(KL(m + n \lg n))$, where K is the number of classes of Pareto optimal trees and L is the number of minimum spanning trees with respect to the length criterion.

REFERENCES

[1] J. B. Kruskal, "On the shortest spanning subtree of a graph and the traveling salesman problem," *Proceedings of the American Mathematical society*, vol. 7, no. 1, pp. 48–50, 1956.

[2] R. C. Prim, "Shortest connection networks and some generalizations," *The Bell System Technical Journal*, vol. 36, no. 6, pp. 1389–1401, 1957. doi: 10.1002/j.1538-7305.1957.tb01515.x

[3] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959. doi: 10.1007 / bf01386390

[4] C. F. Bazlamaçcı and K. S. Hindi, "Minimum-weight spanning tree algorithms a survey and empirical study," *Computers & Operations Research*, vol. 28, no. 8, pp. 767–785, 2001. doi: 10.1016/S0305-0548(00)00007-1

[5] P. C. Pop, "The generalized minimum spanning tree problem: An overview of formulations, solution procedures and latest advances," *European Journal of Operational Research*, vol. 283, no. 1, pp. 1–15, 2020. doi: 10.1016/j.ejor.2019.05.017

[6] S. Steiner and T. Radzik, "Computing all efficient solutions of the biobjective minimum spanning tree problem," *Computers & Operations Research*, vol. 35, no. 1, pp. 198–211, 2008. doi: 10.1016/j.cor.2006.02.023

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

[8] de Sousa, Ernando Gomes, Santos, Andréa Cynthia, and Aloise, Dario José, "An exact method for solving the bi-objective minimum diameter-cost spanning tree problem," *RAIRO-Oper. Res.*, vol. 49, no. 1, pp. 143–160, 2014. doi: 10.1051/ro/2014029

[9] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*, 3rd ed. Cambridge, Massachusetts: The MIT Press, 2009. doi: 10.5555/1614191

[10] M. L. Fredman and R. E. Tarjan, "Fibonacci heaps and their uses in improved network optimization algorithms," *Journal of the ACM*, vol. 34, no. 3, pp. 596–615, July 1987. doi: 10.1145/28869.28874

[11] B. Korte and J. Vygen, *Spanning Trees and Arborescences*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2018, pp. 133–157. ISBN 978-3-662-56039-6