Preprocessing of the 20th Conference on Computer
Science and Intelligence Systems (FedCSIS) pp. 725–730

An algorithm for Direct Construction of all Pareto Optimal Biobjective Minimum Spanning Trees

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*—In this paper we describe an exact algorithm that constructs all Pareto optimal solutions of the biobjective minimum risk minimum length spanning trees problem. The method constructs directly the Pareto optimal spanning trees without constructing the entire classes of optimal spanning trees with respect to the length and with respect to the risk criterion.**

We formulate the theorems that prove the correctness and computational complexity of the proposed algorithms. Also, we illustrate the method using a comprehensive numerical example. The computational complexity of the algorithm that constructs the complete Pareto front of the problem is $O(s(m + n \lg n))$, where s is a constant that depends on the number of all Pareto optimal solutions and a predefined constant that can be used to limit the number of constructed solutions.

I. Introduction

THE minimum spanning tree (MST) problem is the problem to construct a single spanning tree in an undirected weighted graph that has minimum total weight. A spanning tree is an acyclic subset of the edges of the graph that connects all its vertices. In the standard single-objective case the weight of the spanning tree is defined as the sum of the weights of all edges that are contained in it (see for example [1]).

While in practice even the basic binary heap implementation gives a good performance of the Prim's algorithm [2], the advanced data structure Fibonacci heap [3] can result in even more significant speedup. In the general case, the Prim's algorithm implemented with Fibonacci heap runs in $O(m+n \lg n)$ computational complexity, where n is the number of vertices and m is the number of edges in the graph. In the case of sparse graphs for which $m = \Omega(n \lg n)$, the computational complexity is improved to $O(m)$.

On the other hand, by introducing of additional objective function to the MST problem, in specialized literature is reported that it results into a NP-hard problem [4], [5], [6], [7]. Also, in the case of multiple objective functions, we are no longer looking for a single optimal solution, but a whole set of non-dominated optimal spanning trees that are called Pareto optimal (see for example [8]).

Even though the NP-hardness of the biobjective MSTs problem suggests that exact methods are inefficient to apply, and heuristic or metaheuristic methods, like for example ant colony optimization [9], are more appropriate, our research shows that for particular objective functions the problem turns to be weak NP-hard. The latter means that an exact pseudo-polynomial or even polynomial algorithms can be formulated.

In this paper we describe an exact algorithm that constructs the complete Pareto front of the biobjective minimum risk minimum length spanning trees problem. The algorithm does not construct the entire classes of optimal spanning trees with respect to the length and with respect to the risk criterion and directly finds the Pareto optimal solutions of the problem with computational complexity $O(s(m + n \lg n))$, $s = \min\{M, |P|\}$, where M is a predefined constant that can be used to limit the number of constructed solutions, and P is the complete Pareto optimal set.

The paper is organized as follows. In the next section we provide definitions and problem formulation. Section III-A describes our algorithm that constructs the complete set of MSTs for the standard single-objective version of the problem. Section III-B describes our polynomial complexity algorithm that finds the minimum complete Pareto front. The algorithm that solves the main problem in this paper is given in section IV, and section V contains our conclusions.

II. Problem definition

We denote with $G = (V, E)$ a connected undirected graph with $n = |V|$ number of vertices and $m = |E|$ number of edges. The vertices of G are labeled with the first n natural numbers and $V = \{1, 2, \ldots, n\}$. The edges of the G are symmetric and we will denote each edge $(u, v) \equiv (v, u)$ as a pair of its endpoint vertices $u, v \in V$. We define two weight functions on the set of the graph edges: $f : E \to \mathbb{R}_+$ and $g : E \to \mathbb{R}_+$. The function f maps to each edge $(u, v) \in E$ the positive number $f(u, v)$ which we will call the *length* of (u, v). The function g maps to each edge $(u, v) \in E$ the positive number $g(u, v)$ which we will call the *risk* of (u, v).

The graph G together with the two weight function defines the network $N = (V, E, f, g)$ which is represented by the adjacency lists $N.Adj$ which is a vector of n elements, where each element $N.Adj[u]$ is a list of the adjacent vertices to the vertex u and corresponding edges weights:

$$\langle (v_{u_1}, f(u, v_{u_1}), g(u, v_{u_1})), (v_{u_2}, f(u, v_{u_2}), g(u, v_{u_2})), \ldots \rangle$$

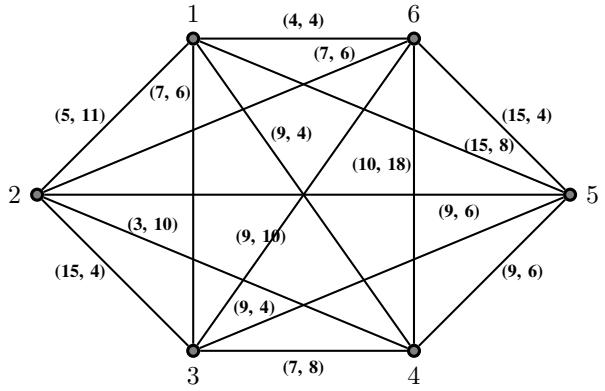

Fig. 1. Example complete network N_1

TABLE I
ADJACENCY LISTS OF EXAMPLE COMPLETE NETWORK N_1 GIVEN FOR EACH VERTEX $u \in V$

u	Adjacency list
1	$\langle(2,5,11),(3,7,6),(4,9,4),(5,15,8),(6,4,4)\rangle$
2	$\langle(1,5,11),(3,15,4),(4,3,10),(5,9,6),(6,7,6)\rangle$
3	$\langle(1,7,6),(2,15,4),(4,7,8),(5,9,4),(6,9,10)\rangle$
4	$\langle(1,9,4),(2,3,10),(3,7,8),(5,9,6),(6,10,18)\rangle$
5	$\langle(1,15,8),(2,9,6),(3,9,4),(4,9,6),(6,15,4)\rangle$
6	$\langle(1,4,4),(2,7,6),(3,9,10),(4,10,18),(5,15,4)\rangle$

This representation is illustrated with the adjacency lists given in Table I of the example complete network N_1 (see Fig. 1).

For each subset of edges $A \subseteq E$ we define the *length* of A:

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

and the *risk* of A with:

$$y(A) = \max\{g(u,v) : (u,v) \in A\}. \tag{2}$$

We denote with W the set of all spanning trees in the network N. The set of all spanning trees with minimum length is W_f, where:

$$W_f = \{T' \in W : x(T') \leq x(T), \forall T \in W\}, \tag{3}$$

and the set of all spanning trees with minimum risk is W_g, where:

$$W_g = \{T' \in W : y(T') \leq y(T), \forall T \in W\}. \tag{4}$$

Definition 1. *We call the tree $T' \in W$ **Pareto optimal spanning tree** (POST) 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* $y(T) < y(T')$*.*

We denote with P the set of all POSTs in the network N. P is the Pareto optimal set and is represented in the form:

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

where the subsets P_i satisfy the following two conditions.

1) All trees in P_i, $i = 1, 2, \ldots, K$, are equivalent.
2) For each $i = 1, 2, \ldots, K-1$ the following inequalities hold:
$$l_i < l_{i+1} \text{ and } r_i > r_{i+1}, \tag{6}$$
where l_i is the length, r_i is the risk of each tree in P_i.

The set P has an unique representation (5). We will call each subset P_i *class of equivalent POSTs*, and we will call the set P *complete Pareto front* (CPF).

Definition 2. *We will call each set $P_{min} = \{t_1, t_2, \ldots, t_K\}$, where $t_i \in P_i$, $i = 1, 2, \ldots, K$ a **minimum complete Pareto front** (MCPF).*

Let $e_1 = (u_1, v_1)$ and $e_2 = (u_2, v_2)$ are arbitrary edges in the network N. We will use the following edge relations.

- The edges e_1 and e_2 are *equivalent*, denoted $e_1 \sim e_2$, when $f(e_1) = f(e_2)$ and $g(e_1) = g(e_2)$.
- The edge e_2 is *dominated by* e_1 with respect the *length*, denoted $e_2 \prec_f e_1$ when: (i) $f(e_1) < f(e_2)$ or (ii) $f(e_1) = f(e_2)$ and $g(e_1) < g(e_2)$.
- The edge e_2 is *dominated by* e_1 with respect the *risk*, denoted $e_2 \prec_g e_1$ when: (i) $g(e_1) < g(e_2)$ or (ii) $g(e_1) = g(e_2)$ and $f(e_1) < f(e_2)$.

When $e_2 \sim e_1$ or $e_2 \prec_f e_1$ is fulfilled, we will denote it by $e_2 \precsim_f e_1$, and when $e_2 \sim e_1$ or $e_2 \prec_g e_1$ is fulfilled, we will denote it by $e_2 \precsim_g e_1$.

The aim of this paper is the solution of the problem given in Problem 1.

Problem 1. *(Main Problem) Let N be a connected network and let M be a natural number. Find a set of POSTs such that:*

1) *if $M \leq K$, then $|P'| = M$ and $P'[i] \in P_i$, for $i = 1, 2, \ldots, M-1$, and $P'[M] \in P_K$;*
2) *if $K < M \leq |P|$, then $|P'| = M$ and P' contains at least one MCPF;*
3) *if $|P| < M$, then $P' = P$.*

The solution of Problem 1 that we propose in this paper is given in Sec. IV. The proposed algorithm finds the list P' and determines which of the three cases of the problem is fulfilled. It also determines whether P' contains at least one MCPF. If P' contains a MCPF, the algorithm reports whether the CPF is found, and if it is not found, which classes of equivalent POSTs, given in equation (5), may contain elements that are not included in P'.

III. HELPER ALGORITHMS

A. Complete list of all minimum spanning trees

In this section we will solve the helper Problem 2.

Problem 2. *Let N be a connected network and let M be a natural number. Construct the list W'_f of MSTs in N that has the following properties:*

1) *if $M \geq |W_f|$, then $W'_f = W_f$;*
2) *if $M < |W_f|$, then $|W'_f| = M$.*

Algorithm 1 Function MSTLIST(N, r, M)

Input: network N, root vertex r, natural number M
Output: list of MSTs L, Boolean flag all

```
1: let S be an empty stack, L be an empty list
2: push into S the initial subproblem
3: while S ≠ ∅ and |L| ≠ M do
4:    X ← POP(S), (t, S) ← SUBPROBLEMS(S, N, X)
5:    if t ∉ L then
6:       PUSHBACK(L, t)
7:    end if
8: end while
9: if S ≠ ∅ then
10:   all ← false
11: else
12:   all ← true
13: end if
14: return L, all
```

The solution of Problem 2 is given by the function MSTLIST in Algorithm 1. The input of the algorithm are the network N, a root vertex of the constructed trees r and a natural number M. The output are the list of lists of parents of the constructed trees L that contains the set W'_f, and a Boolean value all. The set W'_f forms the solution of Problem 2. When $all = true$, it shows that $W'_f = W_f$, and when $all = false$, then $|W'_f| = M < |W_f|$.

The helper function SUBPROBLEMS is described in details in our work [10]. The computational complexity of MSTLIST is evaluated to $O(s(m + n \lg n))$, where $s = \min\{|W_f|, M\}$.

B. Minimum complete Pareto front

In this section we describe Algorithm 3 that solves the helper Problem 3.

Problem 3. *Let M be an arbitrary natural number. Find a subset P' of the set of POSTs P that has the following properties:*

1) $|P'| = \min\{M, K\}$, *where K is the number of classes of equivalent POSTs;*
2) $|P' \cap P_i| = 1$, *for $i = 1, 2, \ldots, |P'|$.*

Algorithm 3 uses the following two modifications of the Prim's algorithm.

1) Function MINTREERISK that finds a POST that has minimum risk. The pseudocode of the function is given in Algorithm 2.
2) Function MINTREELENGTH that finds a POST that has minimum length. In this case the algorithm is analogous to Algorithm 2, but the relation $\prec_f$ is applied.

Both functions MINTREERISK and MINTREELENGTH extend the greedy principle of the Prim's algorithm using the relations $\prec_g$ and $\prec_f$ respectively.

Theorem 1. *Let A is a tree that corresponds to the predecessor list t_{bar} calculated with* MINTREERISK. *Then A is a POST that has minimum risk among all POSTs.*

Algorithm 2 Function MINTREERISK(N, r)

Input: network N, root vertex r
Output: POST that has minimum risk t_{bar}

```
1: a ← [false, false, ..., false]
2: d ← [∞, ∞, ..., ∞], d[r] ← (0, 0), t ← [0, 0, ..., 0]
3: insert each u ∈ V into Q with key d[u]
4: while Q ≠ ∅ do
5:    u ← EXTRACTMIN(Q, ≺_g)
6:    a[u] ← true
7:    for all (v, f(u,v), g(u,v)) ∈ N.Adj[u] do
8:       if a[v] = false and d[v] ≺_g (u, v) then
9:          t[v] ← u, d[v] ← (f(u,v), g(u,v))
10:         DECREASEKEY(Q, v, (f(u,v), g(u,v)))
11:      end if
12:   end for
13: end while
14: E' ← ∅
15: for all e ∈ E do
16:   if g(e) ≤ y(t) then
17:      E' ← E' ∪ e
18:   end if
19: end for
20: E ← E', t_bar ← MSTList(N, r, 1)
21: return t_bar
```

Algorithm 3 Function MINCPF(N, r, M)

Input: network N, root vertex r, natural number M
Output: POSTs list L, barrier tree t_{bar}, integers c_{dif} and M'

```
1: let L be an empty list
2: M' ← M, t_bar ← MINTREERISK(N, r)
3: c_bar ← y(t_bar), c_dif ← 1
4: while M' > 0 and c_dif > 0 do
5:    t ← MINTREELENGTH(N, r)
6:    PUSHBACK(L, t)
7:    c ← y(t)
8:    E' ← ∅
9:    for all e ∈ E do
10:      if g(e) < c then
11:         E' ← E' ∪ e
12:      end if
13:   end for
14:   E ← E'
15:   M' ← M' − 1, c_dif ← c − c_bar
16: end while
17: return L, t_bar, c_dif, M'
```

If A is a tree with a predecessor list calculated with MINTREELENGTH, *then A is a POST that has a minimum length among all POSTs.*

Theorem 2. *The computational complexity of both functions* MINTREERISK *and* MINTREELENGTH *is $O(m + n \lg n)$.*

Algorithm 3 applies as a step also the restriction operation (lines 8-13) which separates a subnetwork N' of the input

network N. The resulting subnetwork N' has the same set of vertices, however it contains only these edges of the original network, which have a risk *strictly less* than the fixed risk value c.

Let the list L is a result of the calculations of Algorithm 3. We define the set of trees P' such that:

1) the number of elements of P' is equal to the length of the list, $|P'| = |L|$;
2) $P'[i]$ is the tree that has the predecessor list $L[i]$, for $i = 1, 2, \ldots, |L|$.

Then the following Theorem 3 holds.

Theorem 3. *The set P' is a solution of Problem 3.*

The proof of Theorem 3 directly follows from Theorem 1.

Corollary 1. *For the above defined set P', barrier tree t_{bar}, integers c_{dif} and M' that result from Algorithm 3, the following four statements hold.*

1) *The tree T_{bar} with predecessor list t_{bar} is a POST and $T_{bar} \in P_K$.*
2) *$P'[i] \in P_i$, for $i = 1, 2, \ldots, |L|$.*
3) *If $c_{dif} = 0$, then the set P' is a MCPF. In this case, if $T_{bar} \neq P'[K]$, then besides the MCPF P', Algorithm 3 have constricted a second tree in the class P_K.*
4) *If $c_{dif} > 0$ and $M' = 0$, then $M < K$. Then for each $j \in \{M+1, \ldots, K\}$ the inequalities $x(P'[M]) < l_j \leq x(T_{bar})$ and $y(T_{bar}) \leq r_j < y(P'[M])$ hold. In this case $M+1$ number of POSTs are constructed from the same MCPF. An additional verification is needed in order to determine whether the set $P' \cup \{T_{bar}\}$ is a MCPF.*

From Theorem 2 it follows that the running time of Algorithm 3 can be evaluated $O(\alpha(m + n \lg n))$, where $\alpha = \min\{M, K\}$. Because K cannot exceed the number of edges of the network m, we say that Algorithm 3 has polynomial complexity.

IV. Complete Pareto front

The helper algorithms, given in section III, are used in the definition of our algorithm for direct construction of all Pareto optimal biobjective minimum spanning trees, which is described in this section.

Let the CPF P of the network N is given by the equality (5). The solution of the main Problem 1 is given by the function DIRECTCPF, described in Algorithm 4. The algorithm stores the predecessor lists of the discovered POSTs in the list of lists C. Each list of lists in C corresponds to one class of equivalent POSTs. The algorithm itself implements the following **inductive procedure**:

1) **Base case.** Store in the list C the first MCPF. Complement C with the remaining POSTs from the class P_1.
2) **Inductive step.** Suppose that in C are stored all POSTs from the class P_j. The procedure complements C with the remaining POSTs from the class P_{j+1}.
3) The inductive procedure continues until all POSTs are constructed or until in C are stored M number of POSTs.

Algorithm 4 Function DIRECTCPF(N, r, M)

Input: network N, root vertex r, natural number M
Output: list of lists of POSTs C and seven indicators

```
let C be an empty list of lists of POSTs
M' ← M
(L, c_dif, t_bar, M') ← MINCPF(N, r, M')
k ← |L|, t_last ← L[k], j ← 0, all ← false
if c_dif = 0 then
    while j < k and M' > 0 do
        M' ← M' + 1, E' ← ∅
        for all e ∈ E do
            if g(e) ≤ y(L[j + 1]) then
                E' ← E' ∪ e
            end if
        end for
        E ← E'
        (L', all) ← MSTLIST(N, r, M')
        M' ← M' − |L'|
        PUSHBACK(C, L')
        j ← j + 1
    end while
    while j < k do
        PUSHBACK(C, ⟨L[j + 1]⟩)
        j ← j + 1
    end while
else
    for all t ∈ L do
        PUSHBACK(C, ⟨t⟩)
    end for
end if
return C, c_dif, t_last, t_bar, M', j, k, all
```

The function DIRECTCPF takes as parameters the input network N, a root vertex r and a natural number M that defines the maximum number of POSTs. As a result of its calculations, the function returns the list of lists of POSTs C and the following indicators:

- c_{dif} shows whether a MCPF is constructed by the function MINCPF;
- t_{last} is the last POST contained in the list of the MCPF L;
- t_{bar} is the barrier tree that belongs to the class P_K constructed by MINCPF;
- M' shows if the number of discovered POSTs is M and if not, how many POSTs are required;
- j shows which is the last class of equivalent POSTs that is stored in C
- k shows whether in C is stored at least one MCPF;
- all is a Boolean flag that shows whether the last call of MSTLIST found all MSTs in the restricted network.

Theorem 4, given below, follows from Theorem 1, Theorem

3 and the correctness of algorithms in section III.

Theorem 4. *Algorithm 4 solves correctly Problem 1.*

Besides that, the following six properties hold.

Property 1. *Each element of C is a list of the predecessors lists of the same class of equivalent POSTs. For example, $C[1]$ is is a list of the predecessors lists of the class P_1.*

Property 2. *If $M' = 0$, then in C are stored exactly M number of POSTs. If $M' > 0$, then exactly M' number of POSTs are still required.*

Property 3. *The indicator k can take exactly two values. It is equal to 0, when C does not store any MCPF. It is equal to the number of classes of equivalent POSTs K, when in C is stored at least one MCPF.*

Property 4. *The indicator $j \in \{0, 1, \ldots, K\}$. If $j = 0$ then C stores at most one element of each class of equivalent POSTs. If $j \in \{1, 2, \ldots, K\}$, then the calculations of the algorithm terminated at the stage in which in C were included the predecessors lists of the POSTs of the class P_j. When $j \in \{2, 3, \ldots, K\}$, then $C[i]$ stores the complete classes P_i, for $i = 1, 2, \ldots, j-1$.*

Property 5. *If $M' = 0$ and $k = 0$, then M number of POSTs has been discovered before a MCPF was completed. The trees t_{bar} and t_{last} define the rectangular area in which are located the POSTs of the MCPF that are not stored in C.*

Property 6. *When $j \neq 0$ and $all = false$, then P_j may contain elements that are not stored in C. If $j \neq 0$ and $all = true$, then all elements of P_j are stored in C.*

From Theorem 2 it follows that the computational complexity of Algorithm 4 is $O(s(m + n \lg n))$, where $s = \min\{M, |P|\}$.

The calculations of Algorithm 4 are illustrated in Example 1. Because of the small size of the network used in the example, the above six properties can be directly verified.

Example 1. *Consider the example network N_1 given in Fig. 1 with adjacency lists in Table I. We select the vertex $r = 1$ for the root vertex of the POSTs that will be constructed. We will track the calculations of Algorithm 4 in the following four cases: (a) $M = 2$; (b) $M = 5$; (c) $M = 18$ and (d) $M = 25$.*

Solution. Before we solve the example, in order to illustrate graphically the results of the calculations, first we find all the spanning trees in the network N_1 using function MSTLIST given in Algorithm 1. The number of all spanning trees of N_1 which has number of vertices $n = 6$ is $n^{n-2} = 1256$. For each spanning tree t we store the pairs $(x(t), y(t))$, and we store only those pairs that does not coincide. For N_1 there are 116 such pairs, which means that the set of all spanning trees in N_1 is divided in 116 classes of equivalent trees. In Fig. 2 and 3 we illustrate all classes of equivalent trees with a black dot with the corresponding Cartesian coordinates.

(a) We solve Problem 1 with Algorithm 4 for $M = 2$. The resulting list is:

$$C = \langle\langle[0, 1, 1, 2, 3, 1]\rangle, \langle[0, 6, 1, 2, 3, 1]\rangle\rangle.$$

From the correctness of Algorithm 4 it follows that the two trees stored in C are POSTs and both of them are from a different class of equivalent POSTs. Since $k = 0$, from Property 5 it follows that the entire MCPF is not yet constructed. If we denote with T_1 the POST with predecessor list $[0, 1, 1, 2, 3, 1]$ and with T_2 the POST with with predecessor list $[0, 6, 1, 2, 3, 1]$, then $T_1 \in P_1$ and $T_2 \in P_2$. In this case $l_{last} = 30$ and $c_{last} = 10$ are respectively the length and the risk of the last tree T_2.

In Fig. 2 we denote the trees T_1 with the point A and T_2 with the point B. Also, the algorithm calculates the barrier tree T_{bar} with predecessor list t_{bar}. We denote with G the point with coordinates $(x(T_{bar}), y(T_{bar})) = (52, 4)$. From the correctness of Algorithm 3 it follows that POSTs that does not belong to P_1 or P_2 are contained into the rectangular area Ω with diagonal BG and sides parallel to the coordinate axes. From Corollary 1 we know that the barrier tree T_{bar} (illustrated by the point G in Fig. 2) is a POST from the last class of equivalent POSTs.

(b) We solve Problem 1 with Algorithm 4 for $M = 5$. After

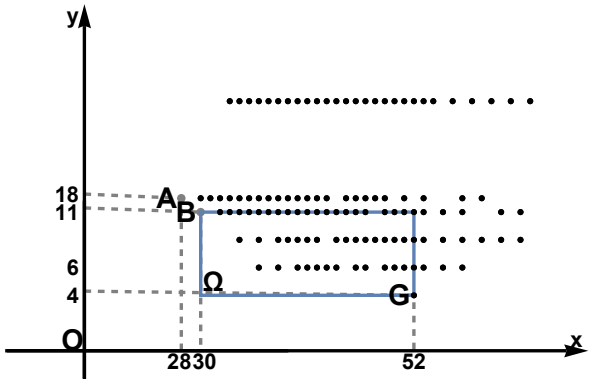

Fig. 2. Partial MCPF and the rectangle area Ω for the example network N_1. The POST that represent the class P_1 is denoted with A, and the POST that represent the class P_2 is denoted with B. The barrier tree is denoted with G

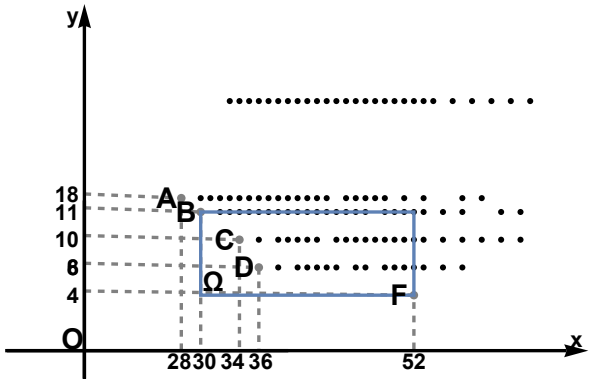

Fig. 3. MCPF for the example network N_1. The POSTs that represent the classes P_1 to P_5 are denoted with A, B, C, D and F respectively

the termination of the algorithm $M' = 0$ and therefore exactly $M = 5$ POSTs are found. In this case the resulting list is:

$$C = \langle\langle[0,1,1,2,3,1]\rangle, \langle[0,6,1,2,3,1]\rangle, \langle[0,1,1,2,3,1]\rangle, \langle[0,6,1,2,3,1]\rangle, \langle[0,1,1,2,3,1]\rangle\rangle.$$

Since now $k = 5 \neq 0$, from Property 3 it follows that C contains a MCPF and it is composed from exactly 5 POSTs. Besides that, $j = 0$ and from Property 4 it follows that for each class of equivalent POSTs a single element is included in the list C. Besides the trees T_1 and T_2 that were also calculated in the case (a), the list contains the trees T_3, T_4 and T_5 with predecessors lists respectively $[0,6,1,3,3,1]$, $[0,6,1,1,3,1]$ and $[0,3,5,1,6,1]$. The corresponding weights of these POSTs are $(34,8)$, $(36,6)$ and $(52,4)$. In Fig. 3 the POSTs T_3, T_4 and T_5 are denoted with the corresponding points in the Cartesian plane $C(34,8)$, $D(36,6)$ $F(52,4)$.

Also, in this case $all = false$ that gives us information that it is not clear whether in P_1 there is no more than one element. Besides that, it is directly verified that $T_5 = T_{bar}$ and in the class P_5 there is no second element that is discovered.

(c) We solve Problem 1 with Algorithm 4 for $M = 18$. After the termination of the algorithm, we get that $M' = 0$, and from Property 2 it follows that in the list C there are exactly 18 POSTs. It can be directly verified that in $C[1]$ there are 6 elements, in $C[2]$ there are 9 elements, and $C[3]$, $C[4]$ and $C[5]$ contain single element each.

Since $k = 5$, from Property 3 it follows that C contains at least one MCPF. Besides that, $j = 2$ and from Property 4 we get that the classes P_1 and P_2 are visited more than once by the algorithm and all elements of P_1 are stored in $C[1]$. Since $all = true$, from Property 6 it follows that all elements of P_2 are stored in $C[2]$. The POSTs stored in $C[3]$, $C[4]$ and $C[5]$ are the same as in the case (b).

(d) We solve Problem 1 with Algorithm 4 for $M = 25$. The algorithm returns $M' > 0$ and $k = 5 \neq 0$, and therefore the CPF is stored in C. The last class of equivalent POST that has been visited by the algorithm is defined by $j = 5$ and it is the class P_5. The classes P_1, P_2, P_3 and P_4 are stored respectively in $C[1]$, $C[2]$, $C[3]$ and $C[4]$. Since $all = true$, the entire class P_5 is stored in $C[5]$.

It can be directly verified that $|C[1]| = 6$, $|C[2]| = 9$, $|C[3]| = 3$, $|C[4]| = 5$, and $|C[5]| = 1$, and the CPF has totally 24 POSTs.

V. Conclusion

In the general case, the biobjective MSTs problem is widely considered an NP-hard problem, which means that an exact method will lead to exponential computational complexity. In our research we have shown that for particular objective functions, the problem is weakly NP-hard, and efficient exact algorithms can be found.

In this paper we have examined in details a version of biobjective MSTs problem in which the first objective function is linear (length), and the second objective function is non-linear bottleneck (risk). Our algorithm that finds the minimum complete Pareto front of the problem has polynomial complexity. The proposed algorithm in this paper that constructs the complete Pareto front has pseudo-polynomial complexity.

We have implemented and tested the described algorithms in this paper on complete random networks. The conducted experiments illustrate the effectiveness of Algorithm 4 that constructs the complete Pareto front. For example, for a complete random network with $n = 100$ number of vertices, the running time of the program written in Wolfram language is not more than 11 minutes on a standard computer configuration. In the particular case, the number of all spanning trees in the network are 100^{98}, the number of Pareto optimal spanning trees in the complete Pareto front are 4739, and it is distributed in 3986 number of classes of equivalent Pareto optimal spanning trees.

The results of the prototype program written in Wolfram language show that with a more optimized implementation (for example in C++ or Julia programming languages), we will be able to run the described algorithms with even bigger examples for much less running time. For example our Julia implementation of Algorithm 3 competes in 66.28 seconds on a complete random network with $n = 1000$ number of vertices.

References

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*, 4th ed. Cambridge, Massachusetts: MIT Press, 2022, ch. 21, pp. 585–603. ISBN 0-262-04630-X

[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] 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

[4] R. M. Ramos, S. Alonso, J. Sicilia, and C. González, "The problem of the optimal biobjective spanning tree," *European Journal of Operational Research*, vol. 111, no. 3, pp. 617–628, 1998. doi: 10.1016/S0022-0000(05)80064-9

[5] D. Rocha, E. Goldbarg, and M. Goldbarg, "A memetic algorithm for the biobjective minimum spanning tree problem," in *Evolutionary Computation in Combinatorial Optimization*, J. Gottlieb and G. R. Raidl, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006. doi: 10.1007/11730095_19. ISBN 978-3-540-33179-7 pp. 222–233.

[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] H. W. Corley, "Efficient spanning trees," *Journal of Optimization Theory and Applications*, vol. 45, pp. 481–485, 1985. doi: 10.1007/BF00938448

[9] S. Fidanova and M. Ganzha, "Ant colony optimization for workforce planning with hybridization," in *Proceedings of the 18th Conference on Computer Science and Intelligence Systems*, ser. Annals of Computer Science and Information Systems, M. Ganzha, L. Maciaszek, M. Paprzycki, and D. Ślęzak, Eds., vol. 35. IEEE, 2023. doi: 10.15439/2023F9586 p. 955–959. [Online]. Available: http://dx.doi.org/10.15439/2023F9586

[10] L. M. Laskov and M. L. Marinov, "Pareto optimal solutions of the biobjective minimum length minimum risk spanning trees problem," in *Proceedings of the of the 19th Conference on Computer Science and Intelligence Systems*, ser. ACSIS, M. Bolanowski, M. Ganzha, L. Maciaszek, M. Paprzycki, and D. Ślęzak, Eds., vol. 39. IEEE, 2024. doi: https://doi.org/10.15439/2024F2913. ISSN 300-5963 pp. 405–416.