Polynomial Algorithm for Minimal Complete Pareto Front of a Biobjective Minimum Spanning Trees Problem

Lasko M. Laskov[0000−0003−1833−8184] and Marin L. Marinov[0009−0003−9544−819X]

New Bulgarian University, Informatics Department
llaskov@nbu.bg, mlmainov@nbu.bg

Abstract. In this paper we propose an exact algorithm that finds the minimum complete Pareto front of a biobjectve minimum spanning trees problem. In the considered case the first objective function is a linear representing length, and the second objective function is nonlinear bottleneck representing risk. The method is based on an extension of the Prim's algorithm that finds a Pareto optimal spanning tree with minimum length and a spanning tree with minimum risk. The latter is used as a barrier tree by the algorithm that constructs a minimum complete Pareto front. We describe in details the proposed algorithms and illustrate them with numerical examples. We show that the computational complexity of the method is polynomial.

Keywords: Combinatorial optimization · MST · Pareto front.

1 Introduction

The standard single-objective minimum spanning tree (MST) problem is one of the earliest combinatorial optimization problems [7] having its origins in two papers by Borůvka published in 1926 [11]. It can be solved in polynomial computational complexity and the well-know algorithms include Borůvka's algorithm, Kruskal's algorithm [10] and Prim's algorithm [12] that was independently discovered by Dijkstra [5]. An important feature that unites all these algorithms is that they take advantage of the greedy property of the MST problem.

On the other hand, the introduction of an additional constraint or objective function causes MST problem to be classified as an NP-hard problem [14], [13]. In this case we are no longer looking to a single MST, but a whole set of non-dominated MSTs that are referred to as *Pareto optimal* [2]. Because of the complexity of the problem, often metaheuristic approaches are used [8], [13]. However, exact methods also are found in the literature [4], [1].

In this paper we propose an exact algorithm that finds the minimal complete Pareto optimal front of a particular biobjective MST problem, in which the first objective function is linear and represents length and the second objective function is non-linear bottleneck and represents risk. The polynomial computational

complexity of the proposed method allows its application on relatively big input networks.

2 Definitions and Problem Formulation

Let $N = (V, E, f, g)$ be a network with set of vertices $V = \{1, 2, \ldots, n\}$, set of undirected edges E with $|E| = m$, and two real-valued weight functions f and g, where $f(u, v)$ is the *length* and $g(u, v)$ is the *risk* of the edge (u, v). The algorithms assume that the network is given by its adjacency lists $N.Adj$:

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

For each subset of edges $A \subseteq E$, we define the *length* and the *risk* of A as the numbers:

$$x(A) = \sum_{i=1}^{|A|} f(u_i, v_i), \text{ and } y(A) = \max\{g(u, v) : (u, v) \in A\}. \tag{2}$$

We denote the set of all spanning trees in N with W, the set of all *minimum spanning trees* (MST) with respect to their length with W_f, and the set of all MST with respect to their risk – with W_g. For all trees $T \in W$:

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

Definition 1. *We call the spanning 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: (i) $x(T) < x(T^*)$ and $y(T) \leq y(T^*)$; (ii) $x(T) \leq x(T^*)$ and $y(T) < y(T^*)$.*

Definition 2. *We will say that the trees T_1 and T_2 are* ***equivalent****, denoted $T_1 \sim T_2$, when $x(T_1) = x(T_2)$ and $y(T_1) = y(T_2)$. We will say that the tree T_2 is* ***dominated*** *by the tree T_1, denoted $T_2 \prec T_1$, when $x(T_1) < x(T_2)$ and $y(T_1) \leq y(T_2)$ or $x(T_1) \leq x(T_2)$ and $y(T_1) < y(T_2)$.*

We denote with P the set of all POSTs in the network N, where:

$$P = \bigcup_{i=1}^{K} P_i. \tag{4}$$

The subsets P_i satisfy the following two conditions: (i) all trees in P_i, $i = 1, 2, \ldots, K$, are equivalent among each other; (ii) for each index $i = 1, 2, \ldots, K-1$ the following inequalities hold:

$$l_i < l_{i+1} \text{ and } r_i > r_{i+1}, \tag{5}$$

where l_i is the length and r_i is the risk of any tree in P_i. The representation of P in (4) is unique and each P_i is called *class of equivalent POSTs*.

Definition 3. *We will call* ***minimal complete Pareto front*** *(MinCPF) each set* $F = \{T_1, T_2, \ldots, T_K\}$ *for which* $T_i \in P_i, \forall i \in \{1, 2, \ldots, K\}$.

Problem 1. **MinCPF** Given the connected network $N = (V, E, f, g)$ by its adjacency lists find a MinCPF.

Problem 2. **Partial Pareto front** Let M be an arbitrary natural number. Find a subset P' of the set of POSTs that has the following properties: (i) $|P'| = \min\{M, K\}$; (ii) for each $i \in \{1, 2, \ldots, |P'|\}$ it is fulfilled $|P' \cap P_i| = 1$.

If M is big enough, then the set P' is a MinCPF. Problem 2 allows us to study large networks. In such cases, P' contains a part of the POSTs of a MinCPF and determine the area in which the remaining elements of the MinCP are located.

The presented algorithms will use the following edge relations. Let $e_1 = (u_1, v_1)$ and $e_2 = (u_2, v_2)$ are arbitrary edges from E.

Definition 4. *We say that two edges* e_1 *and* e_2 *are equivalent, denoted* $e_1 \sim e_2$, *when* $f(e_1) = f(e_2)$ *and* $g(e_1) = g(e_2)$.

Definition 5. *We say that the edge* e_2 *is dominated by the edge* e_1 *by length, denoted* $e_2 \prec_f e_1$, *when one of the conditions is fulfilled: (i)* $f(e_1) < f(e_2)$ *or (ii)* $f(e_1) = f(e_2)$ *and* $g(e_1) < g(e_2)$.

Definition 6. *We say that the edge* e_2 *is dominated by the edge* e_1 *by risk, denoted* $e_2 \prec_g e_1$, *when one of the conditions is fulfilled: (i)* $g(e_1) < g(e_2)$ *or (ii)* $g(e_1) = g(e_2)$ *and* $f(e_1) < f(e_2)$.

When it is fulfilled $e_2 \sim e_1$ or $e_2 \prec_f e_1$ we will write $e_2 \precsim_f e_1$, and when it is fulfilled $e_2 \sim e_1$ or $e_2 \prec_g e_1$ we will write $e_2 \precsim_g e_1$. For each set of edges B we define the subsets $f_{max}(B) = \{e' \in B : \forall e \in B,\ e \precsim_f e'\}$ and $g_{max}(B) = \{e' \in B : \forall e \in B,\ e \precsim_g e'\}$.

3 Spanning Tree Optimal with Respect to the Relation *h*

In this section we describe a procedure that finds a spanning tree that is minimal with respect to one of the objective functions f or g. We turn the objective function to a parameter of the algorithm and we denote it with h. The described algorithm is a modification of the Prim's algorithm [12] that is based on an extension of the greedy property of the problem.

The algorithm manages two subsets of the set of vertices V of the input network N: the set of traversed vertices U and the set of non-traversed vertices Q. It constructs a MST A that is minimal with respect to the objective function h. In the case in which $h = f$, the tree A is a POST. Initially, $U = \{r\}$, $Q = V \setminus U$ and $A = \varnothing$. On each iteration, a *light edge* (u, v), where $u \in U$ and $v \in Q$, is selected and is assigned to A, by also moving v from Q to U. The procedure stops after $n - 1$ iterations when $Q = \varnothing$.

The greedy step of the algorithm is in the selection of the optimal light edge (u, v) on each iteration, that in our case is based on the relations based on Definition 4, Definition 5 and Definition 6.

The algorithm manages the following data structures. A Boolean array a that stores the current sate of the sets U and Q, where $a[v] = true$ if $v \in U$ and $a[v] = false$ if $v \in Q$. The set Q is represented by a min-priority queue abstract data type, which in our case we implement using the advanced data structure Fibonacci heap [6]. The keys of the min-priority queue Q are represented in the array d that stores the distance measure of each vertex $v \in Q$ to the set U:

$$d[v] = \begin{cases} (f(u,v), g(u,v)), & \text{if } E' \neq \varnothing \text{ and } (u,v) \in h_{max}(E') \\ (\infty, \infty), & \text{otherwise,} \end{cases} \tag{6}$$

where E' is the set of edges connecting v with any vertex in U, and h_{max} is either f_{max} or g_{max}. The tree itself is represented by the list of parents t stored as an n-component array. If the tree contains an edge (u, v), then $t[v] = u$, meaning that u is the parent of v.

Algorithm 1 Find a MST by the objective function h

```
1: procedure MinTree(N.Adj, r, h)
2:     (a, Q, d, t) ← Initialize(N.Adj, r)
3:     while Q ≠ ∅ do
4:         v ← ExtractMin(Q, h), a[v] ← true
5:         for all (w, x, y) ∈ N.Adj[v] do
6:             if a[w] = false and d[w] ≺_h (v, w) then
7:                 t[w] ← v, d[w] ← (x, y), DecreaseKey(Q, w, (x, y))
8:             end if
9:         end for
10:    end while
11:    return t
12: end procedure
```

Algorithm 1 calls a procedure $Initialize(N.Adj, r)$ that initializes all elements of a with $false$, places all vertices in the min-priority queue Q, initializes all elements of d with (∞, ∞) except $d[r] = (0, 0)$ and initializes all elements of t with 0.

The procedure $ExtractMin$ is a modification of the standard min-priority queue function. It returns a vertex $v \in Q$ for which there exist a vertex $u \in U$ and it is fulfilled that $(u, v) \in h_{max}(B)$, where B is the set of edges with one endpoint in Q and the other in U. The function $DecreaseKey$ updates the key of a given vertex in the min-priority queue and starts the algorithm that keeps its internal structure.

The computational complexity of Algorithm 1 is $O(m + n \lg n)$ because it differs from a constant factor from the Prim's algorithm and because we use Fibonacci heap to implement the min-priority queue Q (see [9] and [3]).

Theorem 1. *Let A be the set of edges that correspond to the list of parents t calculated by the Algorithm 1 for the objective function f. Then A is a POST that has a minimum length among all spanning trees of the network N.*

Theorem 2. *Let A be the set of edges that correspond to the list of parents t calculated by the Algorithm 1 for the objective function g. Then A is a spanning tree that has risk that is equal to the minimum risk of the POSTs.*

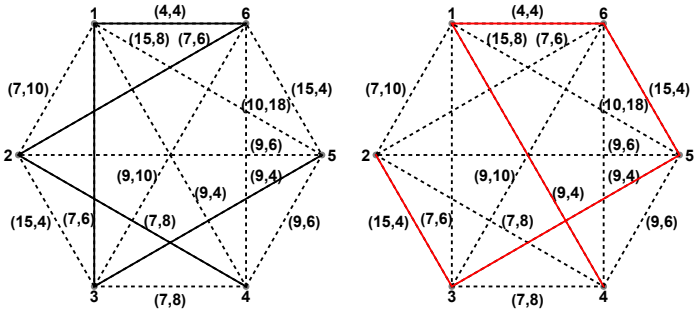

Fig. 1. Example network N_1 and (a) the POST fond by Algorithm 1 with function f, (b) the spanning tree fond by Algorithm 1 with function g which in latter steps will serve as a barrier

Example 1. We will track the calculations of Algorithm 1 for he input network N_1 given in Fig. 1.

Table 1. The lists of parents of the trees calculated by the ith iteration of Algorithm 1 with the input network N_1 and objective functions f and g

i	1	2	3	4	5	6
f	$[0,1,1,1,1,1]$	$[0,6,1,1,6,1]$	$[0,6,1,2,2,1]$	$[0,6,1,2,3,1]$	$[0,6,1,2,3,1]$	$[0,6,1,2,3,1]$
g	$[0,1,1,1,1,1]$	$[0,6,1,1,6,1]$	$[0,6,1,1,6,1]$	$[0,6,5,1,6,1]$	$[0,3,5,1,6,1]$	$[0,3,5,1,6,1]$

Table 1 contains the list of parents of the trees calculated by each of the six iterations of the algorithm for both objective functions f and g. After the last iteration $MinTree(N_1.Adj, 1, f)$ returns the POST $t_1 = [0,6,1,2,3,1]$ with minimum length given in Fig. 1a, and $MinTree(N_1.Adj, 1, g)$ returns the spanning tree with minimum risk $t_{bar} = [0,3,5,1,6,1]$ given in Fig.1b. As it is shown in the next section, the spanning tree with minimum risk plays the role of a barrier tree in the algorithm that finds the MinCPF.

4 Minimum Complete Pareto Front

In this section we describe our algorithm that solves the main Problem 1 considered in this paper. It uses Algorithm 1 for both objective functions f and g and also the helper procedure *Restrict*. *Restrict* takes as parameters the adjacency lists of the network $N.Adj$ and a risk value c, and returns as a result the network N' that has the same set of vertices as N but it contains only these edges that have risk strictly less than c. Also, we extend the definitions of the functions x and y in (2) on the domain of arrays that represent lists of parents of trees.

Algorithm 2

```
1: procedure MinCPF(N.Adj, r, M)
2:     let L be an empty list
3:     t_bar ← MinTree(N.Adj, r, g), c_bar ← y(t_bar), c_dif ← 1
4:     while M > 0 and c_dif > 0 do
5:         t ← MinTree(N.Adj, r, f), PushBack(L, t)
6:         c ← y(t), N.Adj ← Restrict(N.Adj, c)
7:         M ← M − 1, c_dif ← c − c_bar
8:     end while
9:     return L, c_dif, x(t_bar), M
10: end procedure
```

The procedure $MinCPF$ in Algorithm 2 takes as parameters the adjacency lists $N.Adj$ of the input network, the root vertex r and the natural number M. It returns the list L that contains the MinCPF or M number of its elements, the difference between the risk of the current POST and the risk c_{bar} of barrier tree t_{bar}, the length $x(t_{bar})$ of the barrier tree, and the current value that shows whether all K elements of the MinCPF are found.

Algorithm 2 works as follows. With the function call $MinTree(N.Adj, r, g)$ it calculates the barrier tree t_{bar} with respect to the risk objective function. The risk of t_{bar}, c_{bar} is used on each iteration of the loop to determine whether all elements of the MinCPF are found. The loop itself stops if all M elements are constructed, or the difference between the risk of the current POST and the risk of the barrier tree is 0. On each iteration a new element of the MinCPF is constructed using the function call $MinTree(N.Adj, r, f)$, and the network is restricted using the risk c of the newly found POST.

The computational complexity of Algorithm 2 is $O(\alpha(m+n\lg n))$, where $\alpha = \min\{M, K\}$, because the **while** loop will perform α number of iterations, and the computational complexity of $MinTree$ is $O(m + n\lg n)$. Since K cannot exceed the number of edges of the network m, we say that algorithm has polynomial complexity.

Theorem 3. *Let L be the list constructed by Algorithm 2. We define the set of trees P', such that: (i) $|P'| = |L|$, and (ii) $P'[i]$ is the tree with list of parents $L[i]$, for each $i = 1, 2, \ldots, |L|$. Then the set P' is solution of Problem 2.*

The proof of the theorem is based on the correctness of the procedures $MinTree$ and $Restrict$.

Corollary 1. *For the result of Algorithm 2 the following statements hold:*

1. *If $c_{dif} = 0$, then the set P' is a MinCPF.*
2. *If $c_{dif} > 0$ and $M = 0$, then $M < K$ and $P'[i] \in P_i$, $i = 1, 2, \ldots, M$. Besides that, $x(L[M]) < l_j \leq x(t_{bar})$ and $y(t_{bar}) \leq r_j < y(L[M])$, $j = M+1, \ldots, K$.*

Corollary 1 allows us to get an evaluation of those POSTs that are not included in the set P' by Algorithm 2.

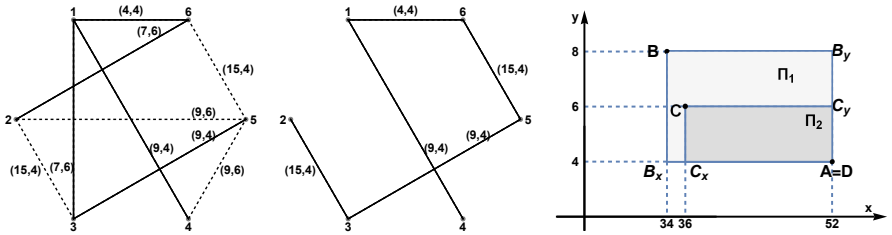

Fig. 2. Execution of Algorithm 2 for the input network N_1 with: (a) subnetwork and POST t_2 after second iteration; (b) subnetwork and POST t_3 after third iteration; (c) MinCPF represented by the points B, C, D and the rectangles Π_1 and Π_2

Example 2. Using Algorithm 2 for the input network N_1 in Fig. 1, we will find a MinCPF.

If $M \geq 3$, the algorithm stops after 3 iterations. The results after the first iteration are given in details in Example 1. The second iteration results in $t_2 = [0, 6, 1, 1, 3, 1]$, and the third iteration constructs $t_3 = [0, 3, 5, 1, 6, 1]$. Since $y(t_{bar}) = y(t_3)$, the algorithm terminates. The subnetworks and POSTs t_2, t_3 for the second and third iterations respectively are given in Fig. 2a and Fig. 2b.

Because $x(t_1) = 34$, $y(t_1) = 8$, $x(t_2) = 36$, $y(t_2) = 6$, $x(t_3) = 52$ and $y(t_3) = 4$ we represent graphically the points $B(34, 8)$, $C(36, 6)$ and $D(52, 4) = A(x(t_{bar}), y(t_{bar}))$. The MinCPF is shown by the points B, C and D in Fig. 2c.

In the case in which $M = 2$, the algorithm will execute only two iterations and the partial Pareto front will contain only the points $B(52, 4)$ and $C(36, 6)$. In this case the remaining POSTs will be illustrated by points contained in the rectangle Π_2.

5 Conclusion

In this paper we examine the biobjective MSTs problem in the case of which the first objective function is a linear and the second objective function is from the

bottleneck type (see (2)). We propose an exact algorithm that finds a MinCPF of the problem and we show that its computational complexity is $O(\alpha(m + n \lg n))$ where $\alpha = \min\{M, K\}$. Since the number of classes of equivalent POSTs K cannot exceed the number of edges m of the network, we say that the algorithm has a polynomial complexity. In the case of which $M < K$ and the algorithm constructs M number of POSTs, it also defines the rectangular area in which the remaining solutions are located.

References

1. Amorosi, L., Puerto, J.: Two-phase strategies for the bi-objective minimum spanning tree problem. International Transactions in Operational Research **29**(6), 3435–3463 (2022). https://doi.org/10.1111/itor.13120
2. Corley, H.W.: Efficient spanning trees. Journal of Optimization Theory and Applications **45**, 481–485 (1985). https://doi.org/10.1007/BF00938448
3. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to algorithms, chap. 21, pp. 585–603. MIT Press, Cambridge, Massachusetts, 4th edn. (2022)
4. de Sousa, E.G., Santos, A.C., Aloise, D.J.: An exact method for solving the bi-objective minimum diameter-cost spanning tree problem. RAIRO-Oper. Res. **49**(1), 143–160 (2014). https://doi.org/10.1051/ro/2014029
5. Dijkstra, E.W.: A note on two problems in connexion with graphs. Numerische Mathematik **1**(1), 269–271 (1959). https://doi.org/10.1007/bf01386390
6. Fredman, M.L., Tarjan, R.E.: Fibonacci heaps and their uses in improved network optimization algorithms. Journal of the ACM **34**(3), 596–615 (July 1987). https://doi.org/10.1145/28869.28874
7. Graham, R.L., Hell, P.: On the history of the minimum spanning tree problem. Annals of the History of Computing **7**(1), 43–57 (January-March 1985). https://doi.org/10.1109/MAHC.1985.10011
8. Knowles, J.D., Corne, D.W.: A comparison of encodings and algorithms for multiobjective minimum spanning tree problems. In: Proceedings of the 2001 Congress on Evolutionary Computation (IEEE Cat. No.01TH8546). vol. 1, pp. 544–551. IEEE, Piscataway, NJ (2001). https://doi.org/10.1109/CEC.2001.934439
9. Korte, B., Vygen, J.: Spanning Trees and Arborescences, pp. 133–157. Springer Berlin Heidelberg, Berlin, Heidelberg (2018). https://doi.org/10.1007/978-3-662-56039-6_6
10. Kruskal, J.B.: On the shortest spanning subtree of a graph and the traveling salesman problem. Proceedings of the American Mathematical Society **7**(1), 48–50 (1956)
11. Nešetřil, J., Milková, E., Nešetřilová, H.: O. Borůvka on minimum spanning tree problem Translation of both the 1926 papers, comments, history. Discrete Mathematics **233**(1), 3–36 (2001). https://doi.org/10.1016/S0012-365X(00)00224-7
12. Prim, R.C.: Shortest connection networks and some generalizations. The Bell System Technical Journal **36**(6), 1389–1401 (1957). https://doi.org/10.1002/j.1538-7305.1957.tb01515.x
13. Santos, A.C., Lima, D.R., Aloise, D.J.: Modeling and solving the bi-objective minimum diameter-cost spanning tree problem. Journal of Global Optimization **60**, 195–216 (2014). https://doi.org/10.1007/s10898-013-0124-4
14. Steiner, S., Radzik, T.: Computing all efficient solutions of the biobjective minimum spanning tree problem. Computers & Operations Research **35**(1), 198–211 (2008). https://doi.org/10.1016/j.cor.2006.02.023