Two-phase strategies for the bi-objective minimum spanning tree problem

Lavinia Amorosi[a,*], Justo Puerto[b]

[a] *Department of Statistical Sciences, Sapienza University of Rome, Italy*
[b] *Department of Statistical Sciences and Operational Research, University of Seville, Spain*

Abstract

In this paper we present a new two-phase algorithm able to generate the complete set of efficient solutions for the Bi-objective Minimum Spanning Tree (BMST) Problem. For the first phase, we adopt two alternative methods: the dual variant of Benson's algorithm and the weighted sum method. The second phase consists in a new enumerative recursive procedure that we propose based on the analysis of reduced costs.
We tested the algorithm on different problem instances, including complete and grid graphs.

Keywords: Multi-objective Optimization, Networks Algorithms, Spanning Trees

1. Bi-objective minimum spanning tree problem

Let $G = (N, E)$ be an undirected graph with node set N and edge set E. Let c_1 and c_2 be two different cost vectors on the edge set. The bi-objective minimum spanning tree problem is defined as:

5

*Corresponding author
Email addresses: lavinia.amorosi@uniroma1.it (Lavinia Amorosi), puerto@us.es (Justo Puerto)

$$\min Cx = \min(\sum_{e\in E} c_e^1 x_e, \sum_{e\in E} c_e^2 x_e) \tag{1}$$

subject to

$$\sum_{e\in E} x_e = n - 1 \tag{2}$$

$$\sum_{e\in E(S)} x_e \leq \mid S \mid -1 \ \forall S \subseteq N, \ S \neq \emptyset \tag{3}$$

$$x_e \in \{0,1\} \quad \forall e \in E \tag{4}$$

In the multi-objective context, and thus in the bi-objective case, the feasible set in the decision space (or decision set) $X = \{x \in R^n : Ax = b, x \geq 0\}$ is distinguished from the feasible set in the objective space (or outcome set) $Y = \{Cx : x \in X\}$, containing the points associated with the feasible solutions by means of the linear mapping defined by the problem criteria. Among the feasible solutions, we search for the ones which correspond to points in the outcome set for which it is not possible to improve one component without deteriorating another one.

Definition 1. *A feasible solution $x^* \in X$ is efficient or Pareto optimal if there does not exist another feasible solution $x \in X$ such that $Cx \leq Cx^*$ with strict inequality for at least one of the objectives. The corresponding vector $y^* = Cx^*$ is called non-dominated.*

Definition 2. *An efficient solution which defines an extreme point of $conv(Y)$ is called extreme efficient solution.*

Definition 3. *(Weakly efficiency). A feasible solution $\bar{x} \in X$ is weakly efficient if there does not exist another feasible solution $x \in X$ such that $Cx < C\bar{x}$. The corresponding vector $\bar{y} = C\bar{x}$ is called weakly non-dominated.*

Definition 4. *(Complete set of efficient solutions). Two feasible solutions x and $x^{'}$ are called equivalent if $Cx = Cx^{'}$. A complete set X_E is a set of efficient*

solutions such that all $x \in X \setminus X_E$ are either non-efficient or equivalent to at least one $x \in X_E$.

Definition 5. *(Pareto or non-dominated frontier). The set of non-dominated vectors, Y_N, is called Pareto or non-dominated frontier.*

There exist efficient solutions which are not optimal for any weighted sum of the objectives. These solutions are called non-supported efficient solutions, while the remaining are called supported efficient solutions. The set of supported efficient solutions is denoted by X_{SE}, while the set of non-supported efficient solutions is denoted by $X_{NE} = X_E \setminus X_{SE}$. Their images in the objective space are indicated respectively by Y_{SN} and Y_{NN}.

2. State of the art

Among the exact methods for solving the bi-objective minimum spanning tree problem proposed in previous works, it is possible to distinguish between generalizations of algorithms for the single objective case and generic approaches for multi-objective combinatorial optimization problems applied to this class. In [1] a generalization of Prim's algorithm is presented. The idea of this generalization is to build trees covering one more vertex at each iteration by selecting efficient edges in the subset connecting covered with non-covered vertices. However this procedure can also produce non-efficient spanning trees. In [2] it is presented an enhancement of Corley's algorithm, excluding in each iteration subtrees which are non-efficient. The resulting algorithm is also discussed in [3] and used in [4] to evaluate a genetic algorithm proposed by the authors able to approximate the Pareto frontier. However [5] showed that using that algorithm some efficient spanning trees may be not generated and consequently also some non-efficient spanning trees may be produced. In [6] a generalization of Kruskal's algorithm was proposed. Also in [7] generalizations of Prim's and Kruskal's algorithms are considered for determining the set of efficient spanning trees according with a preference (binary) relation defined on the set of the graph edges.

Turning to the generic approaches for multi-objective combinatorial optimization applied to the bicriteria minimum spanning tree problem, both two-phase methods and multi-objective Branch&Bound methods have been studied. In [8] a two-phase approach for the bi-objective minimum spanning tree problem, a Branch&Bound algorithm in the second phase, is presented. This algorithm evaluates search nodes according with their ideal point in order to explore the search area defined by the extreme efficient points generated in the first phase. The computational results reported in that paper shown that this approach may not be practical for big instances. In [9] the authors propose another two-phase approach based on a ranking algorithm, the k-best algorithm by [10]. This *k*-best algorithm is applied in the second phase to explore the triangles identified by the extreme non-dominated points given by the first phase. This last paper also provides a comparison with the procedure by [8] that shows the better performance of their approach in term of computational time and instances size that are solvable. Moreover, it is also proposed a heuristic enhancement of the k-best algorithm that, in general, speeds-up the running time. In the most recent work by [11], an improved Branch&Bound algorithm is proposed and applied on the bi-objective minimum spanning tree problem. The main idea is to perform the bounding at a given node in the search tree defining a separating hypersurface in the objective space between the set of the reachable solutions in the subtree and the set of improving solutions. The first ones are the solutions that derive from the partial solution of the current node of the Branch&Bound tree, while the improving ones are the solutions that are not dominated by the set upper bound. Experimental results show that this algorithm is able to solve instances with up to 400-500 nodes, improving also the algorithm by [9] in terms of instances size that are solvable.

3. Finding supported efficient solutions

In this section we describe two alternative approaches for implementing the first phase of the two-phase algorithm proposed for the BMST problem. The first

one is the weighted sum method (see for example [12]) which is a scalarization technique consisting in assigning a weight for each objective function and solving the single objective problem given by the weighted sum of the objectives. Then, solving iteratively these single objective problems, by varying each weight between 0 and 1, it is possible to generate all the supported efficient solutions of a multi-objective integer linear programming problem. The second method, adopted for the first phase, is the dual variant of Benson's algorithm proposed by [13], using the geometric duality theory for multi-objective linear programming developed by [14]. To apply it on the MST problem, it is necessary to consider a different formulation of the problem that ensures that its continuous optimal solutions are integer. At this point, we resort to a formulation by [15] characterized by the integrality property.

3.1. The weighted sum method on the flow formulation

The weighted sum method can be applied on the flow formulation of the MST problem (see [16]) reported below or in any other valid formulation. In our analysis, we have restricted ourselves to the following flow formulation.

Flow formulation of the MST problem

$$\min \sum_{e \in E} c_e x_e \tag{5}$$

subject to

$$\sum_{j:(1,j)\in E} f_{1j} - \sum_{j:(j,1)\in E} f_{j1} = n - 1 \tag{6}$$

$$\sum_{j:(j,i)\in E} f_{ji} - \sum_{j:(i,j)\in E} f_{ij} = 1, \ \forall i \in V, \ i \neq 1 \tag{7}$$

$$f_{ij} \leq (n-1)x_e \ \forall e = (i,j) \in E \tag{8}$$

$$f_{ji} \leq (n-1)x_e \ \forall e = (i,j) \in E \tag{9}$$

$$\sum_{e \in E} x_e = n - 1 \tag{10}$$

$$f_{ij}, f_{ji} \geq 0 \ \ \forall e = (i,j) \in E \tag{11}$$

$$x_e \in \{0,1\} \ \ \forall e \in E \tag{12}$$

This formulation is characterized by a polynomial number of constraints. Through this formulation the minimum spanning tree problem is represented as a single commodity flow problem in which the node 1 is considered as a source of $n-1$ units of flow (6) and the rest of the network nodes are sinks requiring each one one unit of flow (7). Each edge e of the network is represented by a binary variable x_e that is equal to 1 if the edge is taken in the solution. The flow f_{ij} and f_{ji} can be non-zero only if the corresponding edge is in the solution and is limited by the total flow from the source, equal to $n-1$ (8)-(9). Eventually, the total number of edges activated has to be equal to $n-1$ (10) to assure that the topological configuration of the solution is a spanning tree.

The weighted sum method can be applied on this formulation for generating the supported efficient solutions. The idea of such a procedure consists in identifying the partition in intervals of the parametric space $[0,1]$ of the weights $(\lambda, 1-\lambda)$, such that a supported efficient solution of the bi-objective problem does not change when λ varies within the same interval of the partition.

3.2. Benson's algorithm on the Kipp Martin formulation

An alternative approach for implementing the first phase, is represented by the use of the dual variant of Benson's algorithm, [13]. This is a solution method for multi-objective linear programming problems capable to generating all the extreme efficient solutions, that is, the vertices of the feasible region. The main idea of the primal version by [17] is to consider another polyhedron $\bar{Y}$, opportunely defined so that it has the same set of non-dominated points of Y, but where it is easier to identify the set of all efficient extreme points. In the integer

case, this algorithm can be applied only if the integrality property of the solutions is guaranteed, since it must be applied to a continuous linear program. In [15] a linear programming formulation for the MST problem, based on a totally dual integral system, has been proposed and it has been proved that this formulation satisfies the integrality property. This means that, solving it with the dual variant of Benson's algorithm, one can ensure that the solutions so obtained will be integer. In this formulation, reported below, new non-negative variables z_{kij}, associated with triplets of nodes, are introduced together with additional constraints that guarantee the covering of all nodes without the presence of cycles.

Kipp Martin formulation of the MST problem

$$\min \sum_{e \in E} c_e x_e \tag{13}$$

subject to

$$\sum_{e \in E} x_e = n - 1 \tag{14}$$

$$z_{kij} + z_{kji} = x_e \;\; k = 1, ..., n, \;\; e \in E \tag{15}$$

$$\sum_{s>i} z_{kis} - \sum_{h<i} z_{kih} \leq 1 \;\; k = 1, ..., n, \;\; i \neq k \tag{16}$$

$$\sum_{s>k} z_{kks} - \sum_{h<k} z_{kkh} \leq 0, \;\; k = 1, ...n \tag{17}$$

$$x_e \geq 0 \;\; \forall e \in E \tag{18}$$

$$z_{kij} \geq 0 \;\; \forall k, i, j \tag{19}$$

Constraints (15), (16) and (17) guarantee that the binary solution does not contain undirected or directed cycles and constraint (14) assures that the binary solution is a spanning tree.

4. A recursive algorithm for completing the set of efficient solutions

In the second phase of the algorithm, for generating the remaining non-dominated points, we perform an exploration of the outcome set limited to the triangles that can be obtained considering consecutive pairs of non-dominated points associated to extreme efficient solutions and the corresponding Local Nadir point, as shown in fig. 1.

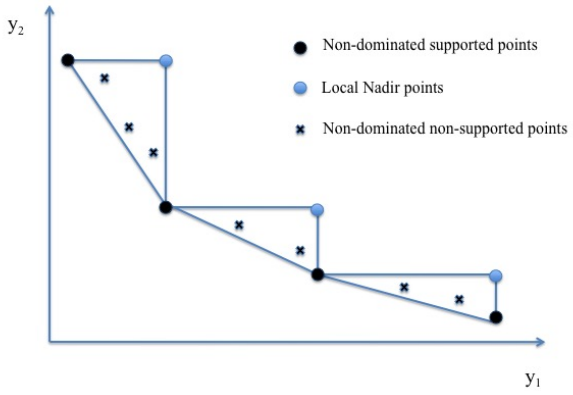

Figure 1: Example of triangles in the objective space

At each iteration of the algorithm, one of these triangles is analyzed, starting from the one associated with the first two non-dominated points, sorted in increasing order with respect to the first objective (y_1). Then, given the triangle under consideration at the generic iteration, the following single objective minimum spanning tree problem is defined:

$$\min \sum_{e \in E} (\lambda_1 c_e^1 x_e + \lambda_2 c_e^2 x_e) \tag{20}$$

subject to

$$\sum_{e \in E} x_e = n - 1 \tag{21}$$

$$\sum_{e \in E(S)} x_e \leq \mid S \mid -1 \ \forall S \subseteq N, \ S \neq \emptyset \tag{22}$$

$$x_e \in \{0, 1\} \quad \forall e \in E \tag{23}$$

where N is the set of nodes and E is the set of edges describing the network, c^1 and c^2 are the integer costs vectors and λ_1 and λ_2 are the weights associated with each objective function. These weights are defined as in [18]: given the two non-dominated points that define the current triangle, $y^i = (y_1(x^i), y_2(x^i))$ and $y^{i+1} = (y_1(x^{i+1}), y_2(x^{i+1}))$, the weights are computed as follows:

$$\lambda_1 = y_2(x^i) - y_2(x^{i+1}) \quad \text{and} \quad \lambda_2 = y_1(x^{i+1}) - y_1(x^i). \tag{24}$$

This definition of weights implies that this single objective MST problem has optimal solutions x^i and x^{i+1}. It is clear that the remaining efficient solutions of the original bi-objective problem correspond to spanning trees of the network under consideration such that the associated points in the objective space are non-dominated. Therefore, the strategy of the second phase consists in an enumeration of the spanning trees of the network restricted to some pre-specified regions. To this end, we design a new recursive algorithm, based on the analysis of the reduced costs associated with an initial optimal spanning tree, able to generate all the other spanning trees of a given network in a given order. This algorithm, differently from the k-best algorithm by [10], does not produce spanning trees in strict ranking order with respect to the scalarized objective function but, according to its operating principle: at each recursion step it generates a spanning tree that is certainly worse than the previous one in term of objective value. This algorithm can be applied in each triangle starting indifferently from any of the optimal vertices, x^{i+1} or x^i, of the weighted sum problem (20)-(23).

4.1. Procedure for generating all the spanning trees of a graph

The procedure proposed for generating all the spanning trees of a graph, works considering the reduced costs associated with the edge variables of the initial optimal solution. The reduced costs are sorted from the smallest one to the biggest one and analyzed in this order. More in details, given the smallest non-zero reduced cost rc_e^*, the fundamental cycle C_e^* created by edge e and the starting optimal spanning tree T^* is identified. The edges in the fundamental cycle are then sorted according with their costs, from the biggest to the smallest. For each of these edges a new spanning tree is generated by swapping (exchanging) the edge e with the edge under consideration. The order in which the edges in the fundamental cycle are exchanged with the arc e guarantees that the obtained spanning trees have non-decreasing objective function value.

At the first iteration of this algorithm, after the first exchange, the obtained spanning tree is exactly the second best with respect to the current weighted objective function. Indeed, this new spanning tree is generated inserting in the initial one, the edge with minimum non-zero reduced cost and removing from the fundamental cycle so created, the edge with maximum cost. Therefore, the corresponding deterioration of the objective function is minimal. When all the arcs in the fundamental cycle created by the arc e are exchanged with e, the second non-zero reduced cost is analyzed, starting again from the original optimal spanning tree.

Considering $e^{'}$, the edge with the second reduced cost, the same procedure described above is applied to generate new spanning trees (their number will be equal to the number of edges in the fundamental cycle that can be exchanged with the edge $e^{'}$). In this case, after each exchange, starting from the new spanning tree $T^{'}$ obtained, the first non-zero reduced cost has to be considered again, implementing all the possible exchanges of the edge e with the edges in the fundamental cycle created by e in the current spanning tree $T^{'}$. Indeed, in order not to miss feasible solutions and to obtain the minimum possible deterioration of the objective function this procedure has to be iterated considering all the non-zero reduced costs sorted from the smallest to the biggest, and all their

possible combinations. Therefore the resulting algorithm consists in a recursive procedure that is summarized in the following pseudocode.

Algorithm 1: Recursive procedure able to generate all spanning trees of a connected graph $G = (N, E)$.

INPUT: graph $G = (N, E)$, costs vector c, optimal solution T^*, vector of the non-zero reduced costs $\bar{rc}^* = \{rc_1^*, rc_2^* ... rc_K^*\}$ (sorted from the smallest to the biggest) where K is the number of non-zero reduced costs, and the list $FT = \{T^*\}$.

find_feasible_trees($G = (N, E)$, c, T^*, $\bar{rc}^*$, K, FT)

```
it = 1
while it ≤ K do
    Let e be the edge out of T* corresponding to the element rc*_it ∈ r̄c (c*_1 is the minimum non-zero reduced cost)
    Let C* be the cycle that the edge e creates with the edges belonging to the spanning tree T*
    Sort by cost (from the biggest to the smallest) the edges of the cycle C* different from e
    for e* ∈ C* with e* ≠ e do
        Add e to T* and remove e* from T*
        Add the tree T' so obtained to the list FT
        if it > 1 then
            Let rc' the vector containing the first it − 1 reduced costs
            find_feasible_trees(G = (N, E), c, T', rc', it − 1, FT)
        end if
    end for
    it = it + 1
end while
```

OUTPUT: Set FT containing all spanning trees of the graph G.

In order to prove the correctness of the recursive procedure **find_feasible_trees**, we recall Theorem 1 from [19].

Theorem 1. *Let G be a connected graph with n vertices and m edges. Starting from any spanning tree, one can obtain every other spanning tree of G by cyclic interchanges. Moreover, if T and T' are two spanning trees, then one can form the the tree T' starting from the tree T by at most $D(T, T')$ cyclic interchanges, where:*

$$D(T, T') \leq \min(\{n - 1, m - n + 1\})$$

Next, we prove that our recursive procedure generates all possible T^*-exchanges.

Theorem 2. *Let us assume that the minimum spanning tree problem (20)-(23) has an unique optimal solution T^*. Procedure* ***find_feasible_trees*** *examines all the feasible combinations of T^*-exchanges.*

Proof. The proof is by induction. Step 4 of the procedure **find_feasible_trees** examines first the fundamental cycle associated with the edge out of T^* with minimum non-zero reduced cost. The edges of this fundamental cycle are sorted by cost (from the biggest to the smallest) and they are exchanged one by one, in this order, with the edge under consideration. Once all the T^*-exchanges that can be generated in this manner have been produced, the procedure continues the generation of feasible combinations of T^*-exchanges in the following way. The algorithm finds the fundamental cycle associated with the second non-zero reduced cost and sorts its edges by their cost. Then it generates just one T^*-exchange between the edge under consideration and the first edge in its fundamental cycle. Now, differently from what it is done in the case of the first non-zero reduced cost, the recursive procedure is called on the new feasible tree so obtained. This means that it attempts to combine the first T^*-exchange associated with the second non-zero reduced cost with the first T^*-exchange associated with the first non-zero reduced cost. Note that continuing in generating T^*-exchanges associated with the edge related to the first non-zero reduce cost, all the T^*-exchanges from the first non-zero reduced cost are combined with the current T^*-exchange (in this case associated with the first edge in the fundamental cycle related to the second non-zero reduced cost). Then, when back on the feasible tree associated with the first T^*-exchange that can be generated by the fundamental cycle related to the second non-zero reduced cost, the procedure seeks a further T^*-exchange from the same fundamental cycle. If it succeeds, again the recursive procedure is called from the new generated feasible tree. By this mechanism, all the feasible combinations of T^*-exchanges generated by the first two fundamental cycles are examined. At a generic stage of the algorithm the non-zero reduced cost k is considered and with the same mechanism all feasible combinations of T^*-exchanges generated by the funda-

mental cycles associated with the first $k-1$-th non-zero reduced costs with the T^*-exchanges associated with the k-th non-zero reduced cost are generated. The algorithm continues generating feasible combinations of T^*-exchanges in order of increasing reduced cost until all non-zero reduced costs are examined, that is until all the feasible combinations of T^*-exchanges are generated.

Finally, the combination of theorems 1 and 2 leads us to correctness result.

Theorem 3. *Let T^* be the unique optimal solution of the minimum spanning tree problem (20)-(23) on the graph G. The procedure **find_feasible_trees**, starting from T^*, can generate all the other spanning trees of the graph G.*

Proof. For the hypothesis, the procedure **find_feasible_trees** starts from the unique optimal spanning tree T^*. Let G be the graph related to the minimum spanning tree problem (20)-(23).
According to Theorem 1, any spanning tree T can be obtained from T^* by at most $D(T^*, T)$ cyclic interchanges with $D(T, T^{'}) \leq \min(\{n-1, m-n+1\})$.
Note that the recursive procedure **find_feasible_trees**, as stated in Theorem 2, generates all feasible combinations of T^*-exchanges. Hence all the other spanning trees are produced.

Observation 1. *It is worth observing that using the recursive procedure above, it is possible to generate more than once the same spanning tree. We also observe that if the optimal solution of the minimum spanning tree problem is not unique we can proceed with any of them and the result still holds.*

4.2. The recursive procedure adapted to the second phase

The recursive algorithm described in Section 4.1, is used for generating the remaining efficient solutions in the second phase. In each triangle, starting from one of the two vertices corresponding to one of the extreme efficient solutions provided from the first phase, the procedure in the previous section is applied setting an upper bound on the value of the objective function for the weighted sum problem (20)-(23). For the first triangle this upper bound is defined (see

[18]), as follows:

$$\Delta = \lambda_1(y_1(x_{i+1}) - 1) + \lambda_2(y_2(x_i) - 1). \tag{25}$$

This upper bound is updated during the recursive procedure with the new spanning trees generated. More in details, when a new spanning tree is produced, the corresponding point in the objective space in computed. If this point is non-dominated and belongs to the current triangle, the corresponding solution is inserted in the list of current efficient solutions and it is used to update the upper bound by means of the formula below (see [20]):

$$\Delta = \max\{\lambda_1(y_1(x_{i,j+1}) - 1) + \lambda_2(y_2(x_{i,j}) - 1),\ \ j = 0, ..., r\} \tag{26}$$

The algorithm terminates when all the triangles has been explored by means of this recursive procedure. The pseudocode of the second phase is shown in Algorithm 2.

Theorem 4. *The set $\mathcal{E}^* = \cup_{i=1,\ldots,s-1}\mathcal{E}_i$ generated by the algorithm **Two_phase_btree** is a complete set of efficient solutions of the bi-objective minimum spanning tree problem.*

Proof.

Without loss of generality we assume $1 \leq i \leq s-1$. We shall prove that, $\mathcal{E}_i$ contains a complete set of efficient solutions in T_i.

Whenever a solution x^b is inserted into $\mathcal{E}_i$ the following conditions hold:

- its objective vector lies within T_i;
- its objective vector is non-dominated by the objective vector of any $x \in \mathcal{E}_i$;
- x^b is not equivalent to any $x \in \mathcal{E}_i$.

The recursive procedure enumerates all solutions x with $c_\lambda(x) \leq \Delta$. Among all enumerated solutions, a complete set of efficient solutions is inserted in $\mathcal{E}_i$. We shall prove this by contradiction. Assume that, after the recursive procedure stops, there exists an efficient solution $x^e \notin \mathcal{E}_i$ within the triangle T_i that

Algorithm 2: second phase.

INPUT: Network (G, c) with $c = (c^1, c^2)$, list of extreme efficient solutions $[x^1, ..., x^s]$.

Two_phase_btree

$i = 1$
$\mathcal{E} = \emptyset$
while $(i < s)$ **do**
  $\mathcal{E}_i = \{x^i, x^{i+1}\}$
  Compute $\lambda_1 = y_2(x^i) - y_2(x^{i+1})$, $\lambda_2 = y_1(x^{i+1}) - y_1(x^i)$ and $c_\lambda = \lambda_1 c^1 + \lambda_2 c^2$
  if $\mathcal{E} \neq \emptyset$ **then**
    for $x^b \in \mathcal{E}$ **do**
      if $y(x^b) \in T_i$, it is not dominated and not equivalent to any $x \in \mathcal{E}_i$ **then**
        Insert x^b in $\mathcal{E}_i$ and remove x^b from $\mathcal{E}$
      end if
    end for
    $\Delta = \max\{\lambda_1(y_1(x^{i,j+1}) - 1) + \lambda_2(y_2(x^{i,j}) - 1), j = 0, ..., r\}$, $r + 1 =\mid \mathcal{E}_i \mid$
  else
    $\Delta = \mu_\lambda = \lambda_1(y_1(x^{i+1}) - 1) + \lambda_2(y_2(x^i) - 1)$ /* initial value of Delta*/
  end if
  $\mathcal{E} =$ **find_feasible_trees**$[N, x^{i+1}, \bar{c}^{i+1}, \mathcal{E}, \Delta]$ /*with $\bar{c}^{i+1}$ vector of the non-zero reduced costs associated with x^{i+1} (sorted from the smallest to the biggest)*/
  for $x^b \in \mathcal{E}$ **do**
    if $y(x^b) \in T_i$, it is not dominated and it is not equivalent to any $x \in \mathcal{E}_i$ **then**
      Insert x^b in $\mathcal{E}_i$ and remove x^b from $\mathcal{E}$
    end if
  end for
  $i = i + 1$
end while
for $\mathcal{E}_i$ $i = 1, ...s - 1$ **do**
  Remove potential non-efficient solutions
end for

OUTPUT: Complete set $\mathcal{E}^* = \cup_{i=1,...,s-1}\mathcal{E}_i$ of efficient solutions.

is not equivalent to some $x \in \mathcal{E}_i$. The recursion stops as soon as $c_\lambda(x) > \Delta$. As the solution $x^e \notin \mathcal{E}_i$ was not obtained during the recursive procedure before it was stopped (otherwise x^e or an equivalent objective vector would be included in $\mathcal{E}_i$), it follows that $c_\lambda(x^e) > \Delta$. As it holds $\mid \mathcal{E}_i \mid \geq 2$, therefore $y_1(x^{ij}) < y_1(x^e) < y_1(x^{ij+1})$ and $y_2(x^{ij}) > y_2(x^e) > y_2(x^{ij+1})$ for some $j \in \{j' : j' = 0, ..., r$ and $y_1(x^{ij+1}) - y_1(x^{ij}) \geq 2$ and $y_2(x^{ij}) - y_2(x^{ij+1}) \geq 2\}$. In particular, $y_1(x^e) \leq y_1(x^{ij+1} - 1)$ and $y_2(x^e) \leq y_2(x^{ij} - 1)$. Consequently we obtain the following inequalities:

$$\lambda_1 y_1(x^e) + \lambda_2 y_2(x^e) \leq \Delta$$

equivalently

$$c_\lambda(x^e) \leq \Delta.$$

The latter inequality contradicts the existence of an efficient solution $x^e \notin \mathcal{E}_i$ within T_i that is not equivalent to some solution in $\mathcal{E}_i$ and that was not obtained during the recursive procedure. The above argument proves that $\mathcal{E}_i$contains a complete set of efficient solutions but it may contain, when the recursion stops, also some solutions potentially non-efficient. Indeed, it may occur that a solution $x \in \mathcal{E}_i$ generated at a given stage of the recursive procedure dominates a solution previously inserted in $\mathcal{E}_i$. Thus, the final step of the algorithm checks if $\mathcal{E}_i$ contains some non-efficient solutions, in order to remove them. This way when the algorithm ends, $\mathcal{E}_i$, is an actual complete set of efficient solutions whose image is within T_i.

5. Preliminary Experimental Results

The two-phase strategies (TP #1 and TP #2) whose first phases have been illustrated in Section 3.1 and Section 3.2, respectively, have been implemented, tested and compared on a testbed of instances generated from two different sets of graphs: grid and complete graphs. We implemented two simple generators of both typologies adopting the procedure described in [9], considering different

numbers of vertices. In our instances, we considered weak and strong correlation between edge costs. More in detail, we developed a procedure for generating costs with different correlation degrees based on three steps: in the first one two uniform random values ($rand_1$ and $rand_2$) between 0 and 1 are generated and a parameter δ (less than $45°$) is given as input. Then an angle $\alpha = [rand_1 * 2\delta - (45°) - \delta)]$ is defined. Finally, the two costs $c_1 = [100 * \cos(\alpha) * rand_2]$ and $c_2 = [100 * \sin(\alpha) * rand_2]$ are computed. The idea underlying this procedure is to convert one of the two initial random values in an angle α between $45° - \delta$ and $45° + \delta$ and to generate the two costs as functions of $\cos(\alpha)$ and $sin(\alpha)$, respectively, with values that range between 0 and 100. Through this procedure if $\alpha = 45°$ the costs are identical otherwise, depending on the value of the parameter δ, we have more or less correlation between the two costs (small values of δ correspond to stronger correlation).

For each set of parameters we generated 10 graphs. We set a time limit of 600s of CPU time for each run and we report the mean running time of the 10 runs distinguishing between the time devoted to the first and the second phase.

In the TP #1 the first phase has been developed using PolySCIP, [21], that is the SCIP module for multi-objective mixed integer programming. This module is based on a particular implementation of the Weighted sum method, namely the Lifted Weight Space Algorithm. The implementation of the first phase of TP #2 is based on Bensolve, [22], that is an open source solver for vector linear programs and, in particular, for multiple objective linear programs based on Benson's algorithm, [17]. More precisely, we used this solver resorting to the dual variant of Benson's algorithm, [13]. The recursive procedure on which the second phase is based has been written in C++. All tests have been run on a node of a cluster named Terastat for scientific computations of the Department of Statistical Sciences of University of Rome, Sapienza.

In table 1 the computational times related to grid graphs with strongly correlated costs are reported. Similarly in table 3 we show the results of grid graphs with weakly correlated edge costs. For each set of instances we distinguished the running times of the two phases, considering the two different strategies for the

first phase (First phase #1 and First phase #2). The last two columns of each table (TP #1 and TP #2) report the corresponding total running times. Table 5 and table 7 show the computational results corresponding to complete graphs always distinguishing between strongly and weakly correlated edge costs.

Table 1: Solution time (sec.) for grid graphs with strongly correlated costs

# nodes	First phase #1	First phase #2	Second phase	TP #1	TP #2
4	0.01	0.002	0	0.01	0.002
9	0.032	0.004	0	0.032	0.004
16	0.063	0.033	0.022	0.085	0.055
25	0.356	0.175	0.108	0.464	0.283
36	3.051	0.765	0.561	3.612	1.326

It can be noted the table 1 that the mean computational times for the first phase grow significantly with the size of the problem, however First phase #2 is more efficient than First phase #1. It can also be observed that for the second phase the increment of computational times in relation to problem size is much smaller than for the first phase. This can be explained also taking into account that the number of non-supported non-dominated points does not increase significantly with the problem size, as it can be verified in table 2 where the mean number of solutions (distinguishing between supported and non-supported) is reported.

Table 2: Mean number of solutions for grid graphs with strongly correlated costs

# nodes	$\mid Y_{SN} \mid$	$\mid Y_{NN} \mid$	$\mid Y \mid$
4	1	0	1
9	2	0	2
16	2	0	2
25	3	1	4
36	5	2	7

Table 3: Solution time (sec.) for grid graphs with weakly correlated costs

# nodes	First phase # 1	First phase #2	Second phase	TP #1	TP #2
4	0.006	0.001	0	0.006	0.001
9	0.06	0.009	0.005	0.065	0.014
16	0.252	0.102	0.118	0.37	0.22
25	2.414	0.520	0.545	2.96	1.06
36	9.927	2.748	4.022	13.95	6.77

A similar beahavior can be noted in table 3, related to grid graphs with weakly correlated costs, but for these instances the average computational times for the second phase increase more significantly with the problem size and more precisely with the mean number of non-supported non-dominated points, as shown in table 4.

Table 4: Mean number of solutions for grid graphs with weakly correlated costs

# nodes	$\mid Y_{SN} \mid$	$\mid Y_{NN} \mid$	$\mid Y \mid$
4	2	0	2
9	4	1	5
16	8	9	17
25	10	18	35
36	15	35	50

As far the number of supported versus non-supported non-dominated points, with the exception of graphs with 4 and 9 nodes, the number of non-supported points is higher than the supported ones (for instance for 36 nodes we have 15 supported and 35 non-supported non-dominated points).

For complete instances the generation of the Pareto frontier is in general more time consuming than for the grid graphs. This can be noted in table 5 and in table 7. For example the average time for grid graphs with 25 nodes and weakly

Table 5: Solution time (sec.) for complete graphs with strongly correlated costs

# nodes	First phase # 1	First phase #2	Second phase	TP #1	TP #2
10	0.087	0.022	0.045	0.132	0.067
14	0.295	0.081	0.411	0.706	0.492
18	1.309	0.230	4.646	5.773	4.876
22	1.479	0.682	5.021	6.5	5.703
26	3.256	1.641	2.618	5.874	4.259

Table 6: Mean number of solutions for complete graphs with strongly correlated costs

# nodes	$\mid Y_{SN} \mid$	$\mid Y_{NN} \mid$	$\mid Y \mid$
10	2	0	2
14	2	0	2
18	3	0	3
22	2	0	2
26	2	0	2

correlated costs is 1.06 seconds while in the case of complete graphs with 26 nodes the average computational time is 250.78 seconds. Clearly, complete graphs have an higher number of spanning trees and represent the most difficult instances for our problem.

We report also that for two complete instances with weakly correlated costs the algorithm is not able to terminate within the time limit (see the * in table 7).

For the complete graphs with weakly correlated costs the number of non-supported non-dominated points is bigger that the number of supported non-dominated points as it can be observed in table 8, differently from the class of complete graphs with strongly correlated costs.

Table 7: Solution time (sec.) for complete graphs with weakly correlated costs

# nodes	First phase # 1	First phase #2	Second phase	TP #1	TP #2
10	0.534	0.074	0.517	1.05	0.591
14	1.964	0.299	4.102	6.06	4.401
18	3.042	1.185	11.305	14.347	12.49
22	8.157	3.994	47.694	55.851	51.688
26	25.937	10.240	240.54 *	266.477*	250.78*

Table 8: Mean number of solutions for complete graphs with weakly correlated costs

# nodes	$\mid Y_{SN} \mid$	$\mid Y_{NN} \mid$	$\mid Y \mid$
10	7	8	15
14	8	10	18
18	11	13	24
22	13	19	32
26	15	24	39

6. Conclusions

In this paper we present a new two-phase algorithm able to generate the complete set of efficient solutions for the BMST problem. The algorithm has been implemented and tested on grid and complete graphs. The preliminary results are encouraging. Further research will be devoted to enlarge the set of instances to include graphs of higher dimensions and considering also instances with negatively correlated costs.

7. References

[1] H. W. Corley, Efficient spanning trees., Journal of Optimization Theory and Applications Vol. 45 (1985) 481–485.

[2] H. Hamacher, G. Ruhe, On spanning tree problems with multiple objectives., Annals of Operations Research Vol. 52 (1994) 209–230.

[3] M. Ehrgott, Multicriteria optimization, Springer, 2005.

[4] G. Zhou, M. Gen, Genetic algorithm approach on multi-criteria minimum spanning tree problem., European Journal of Operational Research Vol. 114 (1999) 141–152.

[5] J. D. Knowles, D. W. Corne, Enumeration of Pareto optimal multi-criteria spanning trees - a proof of the incorrectness of Zhou and Gen's proposed algorithm., European Journal of Operational Research Vol. 143 (2002) 543–547.

[6] P. Serafini, Some Considerations about Computational Complexity for Multi Objective Combinatorial Problems., Recent advances and historical development of vector optimization. Lecture Notes in Economics and Mathematical Systems. Berlin: Springer-Verlag. Vol. 294 (1987) 222–232.

[7] P. Perny, O. Spanjaard, A preference-based approach to spanning trees and shortest paths problems., European Journal of Operational Research Vol. 162 (2005) 584–601.

[8] R. M. Ramos, S. Alonso, J. Sicilia, C. González, The problem of the optimal biobjective spanning tree., European Journal of Operational Research Vol. 111 (1998) 617–628.

[9] S. Steiner, T. Radzik, Computing all efficient solutions of the biobjective minimum spanning tree problem., Computers & Operations Research Vol. 35 (2008) 198–211.

[10] H. Gabow, Two algorithms for generating weighted spanning trees in order., Journal on Computing Vol. 6 (1977) 139–150.

[11] F. Sourd, O. Spanjaard, A Multiobjective Branch-and-Bound Framework: Application to the Biobjective Spanning Tree Problem., INFORMS Journal on Computing .

[12] M. Ehrgott, A discussion of scalarization techniques for multiple objective integer programming., Annals of Operations Research Vol. 147 (2006) 343–360.

[13] M. Ehrgott, A. Löhne, S. Lizhen, A dual variant of Benson's "outer approximation algorithm" for multiple objective linear programming., Journal of Global Optimization Vol. 52 (2012) 757–778.

[14] F. Heyde, A. Löhne, Geometric duality in multiple objective linear programming., Journal of Optimization Vol. 12 (2008) 836–845.

[15] R. Kipp-Martin, Using separation algorithms to generate mixed integer model reformulations., Operations Research Letters Vol. 10 (1991) 119–128.

[16] T. Magnanti, L. A. Wolsey, Optimal trees., Handbooks in Operations Research and Managament Science Vol. 7 (1995) 503–616.

[17] H. P. Benson, An Outer Approximation Algorithm for Generating All Efficient Extreme Points in the Outcome Set of a Multiple Objective Linear Programming Problem., Journal of Global Optimization Vol. 13 (1998) 1–24.

[18] A. Raith, M. Ehrgott, A two-phase algorithm for the biobjective integer minimum cost flow problem, Computers & Operations Research Vol. 36 (2009) 1945–1954.

[19] J. Gross, J. Yellen, Graph Theory and Its Applications, Taylor & Francis Group, 2006.

[20] A. Raith, M. Ehrgott, A comparison of solution strategies for biobjective shortest path problems., Computers & Operations Research Vol. 36 (2009) 1299–1331.

[21] R. Borndörfer, S. Schenker, M. Skutella, T. Strunk, PolySCIP, Mathematical Software – ICMS 2016, 5th International Conference, Berlin, Germany, July 11-14, 2016, Proceedings 9725 (2016) 259–264.

[22] B. Weißing, A. Löhne, The vector linear program solver Bensolve – notes on theoretical background., European Journal of Operational Research Vol. 260 (2017) 807–813.